DEVELOPMENT OF METHODOLOGIES FOR MEMORY MANAGEMENT AND DESIGN SPACE EXPLORATION OF SW/HW COMPUTER ARCHITECTURES FOR DESIGNING EMBEDDED SYSTEMS

Angeliki Stavros Kritikakou

Department of Electrical and Computer Engineering

School of Engineering, University of Patras

Dissertation for the degree of

Doctor of Philosophy (Ph.D)

PhD dissertation No: 305
Patras, May 2013

UNIVERSITY OF PATRAS
SCHOOL of ENGINEERING

Dissertation for the degree of
Doctor of Philosophy (PhD)
of

Angeliki Stavros Kritikakou
Electrical and Computer Engineer

submitted to the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

entitled

DEVELOPMENT OF METHODOLOGIES FOR MEMORY MANAGEMENT AND DESIGN SPACE EXPLORATION OF SW/HW COMPUTER ARCHITECTURES FOR DESIGNING EMBEDDED SYSTEMS

Patras, May 2013

[This page is intentionally left blank]

Certification

With this, it is certified that the PhD dissertation entitled:

DEVELOPMENT OF METHODOLOGIES FOR MEMORY MANAGEMENT AND DESIGN SPACE EXPLORATION OF SW/HW COMPUTER ARCHITECTURES FOR DESIGNING EMBEDDED SYSTEMS / Ανάπτυξη μεθοδολογιών διαχείρισης μνήμης και εξερεύνησης σχεδιασμών σε αρχιτεκτονικές υπολογιστών υλικού/λογισμικού για σχεδίαση ενσωματωμένων συστημάτων

of Angeliki Kritikakou (father's name Stavros), Electrical and Computer Engineer, has been publically presented and defended in the certified Teleconference Room of Library and Information Center of University of Patras in 14/05/2013 and has been examined and approved by the following seven member PhD Defense Committee:

- **Costas Goutis**, Emeritus Professor of the Department of Electrical and Computer Engineering of Polytechnic Faculty of University of Patras, Greece (supervisor).
- **Francky Catthoor**, Professor of the Department of Electrical Engineering of Engineering Faculty of Catholic University of Leuven and fellow in Interuniversity Microelectronics Centre (IMEC), Belgium (Advisory committee member).
- **Spyridon Nikolaidis**, Associate Professor of the Department of Physics of Aristotle University of Thessaloniki, Greece (Advisory committee member).
- **Odysseas Koufopavlou**, Professor of the Department of Electrical and Computer Engineering of University of Patras, Greece.
- **Dimitris Nikolos**, Professor of the Department of Computer Engineering and Informatics of University of Patras, Greece.
- **George Theodoridis**, Assistant Professor of the Department of Electrical and Computer Engineering of University of Patras, Greece.
- **Dimitrios Soudris**, Assistant Professor of the Department of Electrical and Computer Engineering of National Technical University of Athens, Greece.

Patras, May 2013

Supervisor

C. Goutis
Em. Professor

Vice Head of the Department

Gabriel B. Giannakopoulos
Professor

[This page is intentionally left blank]

Members of PhD Defense Committee

- **Costas Goutis**, Emeritus Professor of the Department of Electrical and Computer Engineering of Polytechnic Faculty of University of Patras, Greece (supervisor).
- **Francky Catthoor**, Professor of the Department of Electrical Engineering of Engineering Faculty of Catholic University of Leuven and fellow in Interuniversity Microelectronics Centre (IMEC), Belgium (Advisory committee member).
- **Spyridon Nikolaidis**, Associate Professor of the Department of Physics of Aristotle University of Thessaloniki, Greece (Advisory committee member).
- **Odysseas Koufopavlou**, Professor of the Department of Electrical and Computer Engineering of University of Patras, Greece.
- **Dimitris Nikolos**, Professor of the Department of Computer Engineering and Informatics of University of Patras, Greece.
- **George Theodoridis**, Assistant Professor of the Department of Electrical and Computer Engineering of University of Patras, Greece.
- **Dimitrios Soudris**, Assistant Professor of the Department of Electrical and Computer Engineering of National Technical University of Athens, Greece.

[This page is intentionally left blank]

To the ones that make me a better person...

[This page is intentionally left blank]

Acknowledgements

The present PhD dissertation conducted in the VLSI Laboratory of the Department of Electrical and Computer Engineering, School of Engineering, University of Patras in collaboration with Interuniversitair Micro-Elektronica Centrum (IMEC), Belgium. It has been partially funded by Public Welfare Foundation ``Propondis'' research funds.

Before going into the details of the PhD dissertation, I would like to strongly thank for their contribution in the completion of this work:

Initially, Dr. Costas Goutis, Em. Professor of Electrical and Computer Engineering Department of the University of Patras, whom I warmly thank for enriching my knowledge and research experience and for the moral support throughout the study.

I would also like to cordially thank Francky Catthoor, who has always been an excellent mentor and a role model, supportive and willing to help in any kind of problem during our collaboration. I would like to express my deep appreciation and especially thank him for broaden my horizons and helping me to improve myself, first of all as a person, and secondly as a researcher.

I also want to thank the member of the three-member advisory Committee Dr. Spyridon Nikolaidis, Associate Professor of Physics Department of Aristotle University of Thessaloniki, for his guidance during the PhD dissertation. I would like to thank Prof. Odysseas Koufopavlou, Prof. Dimitris Nikolos, Assistant Prof. Dimitrios Soudris and Assistant Prof. George Theodoridis. Their comments aided me substantially in the improvement of this dissertation.

I would like to give my cardial thanks to Sofia-Kalliopi Passa for her priceless help and support.

I would like to express my respect to my colleagues in our Lab for their support and the nice environment, Vasilis Kelefouras, George Athanasiou and Andreas Emeretlis.

This study would not have been completed without the full moral support of people that are standing close to me. As a minimum return, I sincere thank my friends that were and will be by my side, for our meaningful conversations, their advice and their different point of view, their motivation and support and for all our moments that played an very importnat role in my life. I would like to name some of them: Thodoris, Ntina, Angelos, Xristina, Tasoula, Giorgia, Iosif, Dimitris, Magda, Christos, Sofia, Maria, Vasilis, Dimitra, Lia, Ifigenia, Antonia, Anna, Nikos, Efi, Katerina, Sofia M, Kokkonas, Giannakos, Kostas...

Finally, I would like to express my gratitude to my family who always helps me by all means and always encouraged me.

Angeliki S. Kritikakou

Patras, May 2013

Abstract

This PhD dissertation proposes innovative methodologies to support the designing and the mapping process of embedded systems.

Due to the increasing requirements, embedded systems have become quite complex, as they consist of several partially dependent heterogeneous components. Systematic Design Space Exploration (DSE) methodologies are required to support the near-optimal design of embedded systems within the available short time-to-market. In this target domain, the existing DSE approaches either require too much exploration time to find near-optimal designs due to the high number of parameters and the correlations between the parameters of the target domain, or they end up with a less efficient trade-off result in order to find a design within acceptable time. In this dissertation, an alternative DSE methodology is presented, which is based on systematic creation of scalable and near-optimal DSE frameworks. The frameworks describe all the available options of the exploration space in a finite set of classes. A set of principles is presented which is used in the reusable DSE methodology to create a scalable and near-optimal framework and to efficiently use it to derive scalable and near-optimal design solutions within a Pareto trade-off space.

The DSE reusable methodology is applied to several stages of the embedded system design flow to derive scalable and near-optimal methodologies. The first part of the dissertation is dedicated to the development of mapping methodologies for storing large embedded system data arrays in the lower layers of the on-chip background data memory hierarchy, and the second part to the DSE methodologies for the processing part of SW/HW architectures in embedded systems including the foreground memory systems.

Existing mapping approaches for the background memory part are either enumerative, symbolic/polyhedral and worst case (heuristics) approximations. The enumerative approaches require too much exploration time, the worst case approximation lead to overestimation of the storage requirements, whereas the symbolic/polytope approaches are scalable and near-optimal for solid and regular iteration spaces. By applying the new reusable DSE methodology, we have developed an intra-signal in-place optimization methodology which is scalable and near-optimal for highly irregular access schemes. Scalable and near-optimal solutions for the different cases of the proposed methodology have been developed for the cases of non-overlapping and overlapping store and load access schemes. To support the proposed methodology, a new representation of the array access schemes, which is appropriate to express the irregular shapes in a scalable and near-optimal way, is presented. A general pattern formulation has been proposed which describes the access scheme in a compact and repetitive way. Pattern operations were developed to combine the patterns in a scalable and near-optimal way under all the potential pattern combination cases, which may exist in the application under study.

In the processing oriented part of the dissertation, a DSE methodology is developed for mapping instances of a predefined target application domain onto a partially fixed architecture platform template, which consists of one processor core and several custom hardware accelerators. The DSE methodology consists of uni-directional steps, which are implemented through parametric

templates and are applied without costly design iterations. The proposed DSE methodology explores the space by instantiating the steps and propagating design constraints which prune design options following the steps ordering. The result is a final Pareto trade-off curve with the most relevant near-optimal designs. As the scheduling and the assignment are the major tasks of both the foreground and the datapath, near-optimal and scalable techniques are required to support the parametric templates of the proposed DSE methodology. A framework which describes the scheduling and assignment of the scalars into the registers and the scheduling and assignment of the operations into the function units of the data path is developed. Based on the framework, a systematic methodology to arrive at parametric templates for scheduling and assignment techniques which satisfy the target domain constraints is developed. In this way, a scalable parametric template for scheduling and assignment tasks is created, which guarantees near-optimality for the domain under study. The developed template can be used in the Foreground Memory Management step and Data-path mapping step of the overall design flow. For the DSE of the domain under study, near-optimal results are hence achieved through a truly scalable technique.

Contents

List of Figures

List of Tables

Nomenclature

A	Access
A-FUs	Add Function Unit
A2A	Access to Access
A2H	Access to Hole
AI	Artificial Intelligent
ALAP	As Late As Possible
ASA	Adaptive Simulated Annealing
ASAP	As Soon As Possible
ASIC	Application-Specific Integrated Circuit
ASIP	Application Specific Instruction Set Processors
B&B	Branch & Bound
BF	Breadth First
BG	BackGround
CDFG	Control and Data Flow Graph
CS	Critical Subgraph
DF	Depth First
DL	Data Level
DL DTSE	Data Level DTSE
DLP	Data Level Parallelism
DMA	Direct Memory Access
DP	Data-Path
DPM	Dynamic Power Management
DPMa	Data Parallelization Management
DS	Design Space
DSE	Design Space Exploration
DSP	Digital Signal Processing
DTSE	Data Transfer and Storage Exploration
DVFS	Dynamic Voltage and Frequency Scaling
E	Evolution
ECH	Enumerative Conditions for Iteration Space with Holes
ECS	Enumerative Conditions for Solid iteration space
EDF	Earliest Deadline First

FCFS	First Come First Served
FDCT	Fast Discrete Cosine Transform
FFT	Fast Fourier Transformation
FG	ForeGround
FPCH	First PCH
FPS	Fixed Priority Scheduling
FR	Frame Rate
FSL	Fast Simplex Link
FSM	Finite State Machine
FU	Function Unit
GA	Genetic Algorithms
GCD	Greatest Common Divisor
H	Hole
H2A	Hole to Access
HDFG	Hierarchical Data Flow Graph
HID	Hole Iterator Domain
HNF	Heavy Node First
II	Iterative Improvement
IL	Instruction Level
ILP	Integer Linear Programming
IR	Iterator Range
ISH	Iteration Space with Holes
LB	Low Bound
LC	Linear Clustering
LCM	Least Common Multiple
LCTD	Clustering with Task Duplication
LoC	Lab-on-Chip
LP	Linear Programming
MAA	Memory Allocation & Assignment
MIBP	Mixed-Integer Bilinear Programming
MILP	Mixed Integer Linear Programming
MLP	Memory Level Parallelism
MSPCD	Multiprocessor Scheduling Problem with Communication Delays
NP-hard	Non-deterministic Polynomial-time hard
NPI	Native Port Interface
NRE	Non-Recurring Engineering
OCM	Operations Concurrency Management
OL	Ordered List
PAI	Processor Architecture Intergration
PCH	Parametric Conditions for Iteration Space with Holes

PCS	Parametric Conditions for Solid iteration space
PIR	Part Iterator Range
PL	Processor Level
PL DTSE	Processor Level Data Transfer and Storage Exploration
PLB	Processor Local Bus
PS	Pattern Size
PSGA	Problem-Space Genetic Algorithm
PT	Part Type
QEA	Quantum-inspired Evolutionary Algorithm
R	Repetition factor
RCS	Resource Constraint Scheduling
RD	Read
RISC	Reduce Instruction Set Computer
S-FUs	Shift Function Unit
SA	Simulated Annealing
SCBD	Storage Cycle Budget Distribution
SCIP	Standalone Custom IP
SE	Simulated Evolution
SHA	Secure Hash Algorithm
ShA	Shift-Add
SID	Segment Iterator Domain
SIMD	Single Instruction Multiple Data
SIS	Solid Iteration Space
SPCH	Second PCH
SW/HW	SoftWare/HardWare
TCM	Task Concurrency Management
TCS	Time Constraint Scheduling
TF	Thread Frame
TF DTSE	Thread Frame Data Transfer and Storage Exploration
TLP	Task Level Parallelism
TRCS	Time Resource Constraint Scheduling
UB	Upper Bound
UF	Unrolling Factor
VLIW	Very Long Instruction Word

Chapter 1

Introduction & Motivation

1.1 Goal & Context

Embedded systems are computer systems which execute applications dedicated to a specific goal, without indented to be a general purpose computer. Embedded systems contain a collection of programmable parts and components, which interact with the environment. Examples of embedded systems are mobile devices, bio-medical devices, security devices, multimedia devices etc.

The embedded applications are usually from multimedia, graphics, wireless, biomedical, automotive, signal processing application domains. Except their complexity, the embedded applications impose constraints and trade-offs on several metrics of the embedded systems that will execute them. Additional constraints and trade-offs are imposed by the environment, i.e. the market, the designing and manufacturing company and the users. The constraints describe the minimum acceptable value in the metric that should be satisfied in order the system to function correctly. Any design that has a value in a metric below the constraints is considered as inappropriate. The trade-offs are choices which can reside in a range of values of the metric, which are better points than the constraints.

Several applications of embedded systems have usually constraints on the performance, which is expressed through the execution time or the throughput of the system. Whether the performance and the time requirements of the application are a hard or soft constraint depends on the remaining characteristics of the application. For instance, when a strict deadline exist in the execution of the application, that imposes a constraint in the performance. Most embedded systems operate with batteries and thus energy consumption is very crucial in order to extend the system lifetime. Hence, the minimum acceptable lifetime defines the constraint in energy consumption. Then, a further reduction of execution time or the energy consumption can be decided during exploration of the multi-dimension trade-off space of the different design options.

To meet the imposed constraints, the embedded systems consists of heterogeneous multiprocessors, such as RISC, VLIW or SIMD, different operations modes, such as Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM), Application Specific Instruction Set Processors (ASIP), DSP and ASIC, reconfigurable processing units, complex data

memory hierarchies, advanced interfaces etc. Due to the system requirements, software designs cannot be performed independently from the under-laying hardware and both software and hardware must be taken into account during the design [122].

The multidimensional trade-off space is created by a set of objective axes. A set of high weighted objectives is performance, area and energy consumption. As embedded systems are usually real-time systems, the time execution, i.e. performance expressed in latency or throughput is a crucial objective. As embedded systems are portable, energy consumption is also crucial. Leakage energy consumption is indirectly affected by the area of the embedded systems. As the number of gates is increased, leakage energy is also increased. However, these major objectives cannot be concurrently satisfied, as one contradicts the other. We further describe this complicated trade-off. To further reduce the time execution, several design options exists, i.e. mapping of critical parts into custom HW accelerators and components, potential parallelization of the platform components and microprocessors, higher frequency in HW parts, more dense scheduling of the operations and the memory accesses. These design options lead to energy consumption and area increase. Any extra requirement in the hardware part increases the number of gates and thus the leakage energy. The more dense scheduling which optimize the elements accessing leads to increase in the required storage size, as the lifetimes of the variables are extended. The increase in the storage size requirements leads to larger memories, which also leads to an increase in the energy per access.

However, embedded systems are mass products in highly competitive markets and thus the final system should have a low cost unit, which expresses both the manufacturing and the design cost. The manufacturing cost is expressed by the unit cost, i.e. the cost of manufacturing one copy of the system. When the power consumption is increased, the unit cost is also increased due to the requirements of stronger power supply and of more expensive cooling system. When the area is increased due to size increase of the physical space, measured in bytes for software and gates for hardware is increased, the unit cost is also increased, since larger memories are required to be used. The design cost is expressed by the Non-Recurring Engineering (NRE), which is a pre-production effort of designing the system, and the engineering effort required for modification of the system. Hence, flexibility to target multiple application instances in a domain on the same shared platform instance is also an important objective, as it describes the ability to change system functionality without heavy NRE cost. To increase flexibility, additional area has to be used. Fortunately, when the system is targeting a large domain this also means that (much) higher chip production volumes can be realized. That heavily reduces the cost per chip so high flexibility is heavily weighted in the overall cost equation and thus a somewhat larger area is clearly acceptable.

The high competition of the markets imposes short time to market, i.e. the time required to design and manufacture the system in order to be ready to be sold. A short delay of the product release can have catastrophic financial consequences and thus it is a highly weighted objective in the overall trade-offs space. The time-to-market should remain within a market window, which affect the time available for the design of the system. The available design time allows a better exploration of the multi-objective design space. The more time is available for the system design, the better

exploration of the multi-objective space is performed, leading to closed to optimal designs and a better yield and a lower unit cost. However, the NRE cost is increased. When the design time is reduced or the NRE cost is lower, a more aggressive pruning is performed in the exploration space leading to results that are somewhat less optimal, but still within an acceptable tolerance margin.

Due to the increase in the complexity of the embedded system hardware and software, the strong time and power constraints of the applications, the low cost and the short time to market and the overall multidimensional objective space, ad-hoc design approaches based on the experience of the designer, which must be expert in both hardware and software fields, cannot lead to near-optimal results. The design process takes too much time and without upfront guarantee of meeting the constraints and designing a near-optimal system from the design space created by the several contradicting trade-offs. Hence, DSE methodologies, which also for large application codes efficiently explore the design options in the design space of the different trade-offs of the system and provide near-optimal designs in short time-to-market are required.

1.2 Existing Design Space Exploration methodologies & limitations

The DSE methodologies searches the options of the architecture, the components, the interfaces and the data mapping to achieve a near-optimal system which satisfies the constraints and minimizes the multidimensional trade-off space. During the DSE the input and output requirements, the storage requirements, the processing requirements and the system control are explored.

Academia research focuses more on DSE methodologies that provide near-optimal designs. The ideal DSE methodology is to address all mapping problems simultaneously in a single phase. However, the ideal DSE methodology cannot be achieved. The design of embedded systems is a very complex process, which consists of several mapping phases, thus no good way exist to optimally formulate the single phase solution. Hence, the DSE methodology has to be divided into a number of sub-tasks in order to be manageable [122].

Hence, the DSE methodologies divide the DSE process into steps. The division into step is performed in an ad-hoc way and thus the sub-steps are usually bidirectionally affect each other. In the case of bidirectionally connected steps, iterations between steps are required to search for near-optimal solutions. Due to the bi-directional correlation of the sub-steps, no guarantee exists that the DSE will be finalized in a reasonable execution time with a near-optimal design. When the number of software and hardware parameters is increased, the costly design iterations of the bi-directionally correlated sub-steps of the DSE lead to not scalable approaches. For instance, an iterative DSE method starts from the designer's base configuration, changes the value of one parameter each time and uses the results to predict the optimal design is proposed in [175]. The aforementioned DSE approach may lead to less efficient designs when a high number of parameters and interdependencies exists.

In addition, each mapping step consists of tasks which are solving NP-hard problems. The con-

ventional techniques applied in one step can achieve near-optimality only for small design problems in the available exploration time. When the complexity of the design problem is increased, which is usually the case, the conventional techniques are incapable of identifying the solution within reasonable search time. For instance, stochastic approaches require unacceptable exploration time to reach near-optimal solutions in a large exploration space. The stochastic approaches search the space based on different types of random moves. In order to reach a near-optimal result in the target domain, they need too many search attempts in near-by regions of the design exploration space. For instance, the quality of the Quantum-inspired Evolutionary Algorithm (QEA) for the multiprocessor mapping is highly based on the number applied generations. The increase in the number of generations increases the chances to reach a near-optimal solution [3]. A DSE methodology with stochastic algorithms is proposed in [147] and a simulated annealing DSE approach of object detection accelerators is proposed in [73]. Deterministic design approaches, like Integer Linear Programming (ILP) techniques, require too much exploration time when applied in medium and large design problems. The branch and bound design methods also require increased search time to guarantee optimal results. In order to reduce the exploration time, the branch and bound process has to apply a more aggressive pruning in the available design options, which reduces the final quality of the final design. Heuristic and greedy design methods search the design exploration space based on a set of predefined rules, which cannot guarantee optimal solutions to the problem in its most general form [39].

Compilers are mainly used by DSE methodologies in order to map the application in the different designs. The main subtasks of the compiler are: 1) code selection, i.e. mapping of machine instructions of the target processor, 2) the register allocation, i.e. mapping of scalars to registers in order to minimize the memory references during program execution, 3) register assignment, i.e. determine in which physical register a value is stored, 4) instruction scheduling, i.e. the reordering of the instructions sequence to exploit parallelism and 5) resource allocation, i.e. the assignment of functional units and buses to operations. The phases as defined above in conventional compiler flows, execute sub-tasks that are heavily interdependent, i.e. bi-directionally connected, as decisions of one phase may impose restrictions to other phases leading to sub-optimal overall solution. For instance, in an embedded system compiler the code selection phase assigns virtual registers from several classes and the class to be selected is only known during the register allocation phase. The register allocation phase cannot precede the code selection phase, since the required registers are only known after the code selection phase [121]. In the literature this is known as the phase coupling problem, and it is commonly believed that this dilemma is inevitable in practical compiler approaches due to its NP completeness in the traditional approaches [188]. Much research has been performed on the compiler phase ordering problem, i.e. to identify the best order between compiler phases. In [103] it is stated that no universal optimization phase order exists, as it depends on the function being compiled, the compiler and the architecture characteristics. However, in this dissertation we will show that this dilemma is only inevitable within the traditional step-wise approach. The conventional DSE approaches provide a sub-optimal combination of sub-optimal partial results per phase without the proper splitting into sub-steps with constraint propagation in

between. This leads to a less efficient overall trade-off solution.

In order to reduce the exploration time for bigger applications, dependencies between the bidirectional steps are partially overlooked. For instance, Ref. [176] sorts the parameters based on the impact determined by the maximum parameter value change. All combinations of the first two high impact parameters are considered. The independence of the parameters is used to prune the space, which usually is quite restricted, and to derive the Pareto curve in Platune [147]. Divide and conquer approaches cannot efficiently explore the structure of the design problem under study. Hence, they overlook the existing constraints during division. Then, they solve each design sub-problem independently and when they combine the partial results, they end up in at least partly sub-optimal solutions.

In industry, the design problems to be solved are large and complex, thus the DSE approaches have to fully give up on near-optimality in order to achieve scalability. Hence, design steps are applied partially independently and with low complexity heuristics per step to reduce exploration time. State-of-the-art tools do not take into account the inter-dependencies among processing, memory and communication constraints, leading to less optimal designs. When the near-optimal designs per step are combined, design quality is reduced due to conflict constraints in the different steps which lead to overlooking near-optimal designs.

Hence, a strong dilemma is present in existing DSE methodologies and tools: to either give up on design near-optimality or to give up on scalability.

1.3 Scope & Main Contributions

The scope of this dissertation is to provide a totally different direction to address the scalability/near-optimality dilemma by proposing a DSE approach which is capable of achieving both near-optimal designs and scalable exploration. The proposed DSE methodology divides the complex design problem in a special way into smaller and less complex design steps, achieving scalability. In contrast to existing DSE methodologies, the splits between the steps are selected in such a way that they can be connected through uni-directional constraint propagation. In this way, near-optimality is achieved as dependencies are not ignored, which happens when the steps are considered independent. Costly design iterations are avoided, whereas the conventional DSE methodologies split the design problem into bidirectionally connected steps, which require design iterations, highly increasing the exploration time. We apply the proposed reusable scalable and near-optimal DSE methodology in several stages of the higher layers of the design of embedded systems, especially for the intra-signal in-place optimization step of the background memory, which explores the performance-area-energy trade-off, and the processing mapping part, which explores the design time-performance-area trade-offs.

The main contributions of the dissertation are:

- **Reusable methodology for scalable and near-optimal DSE frameworks.**
  The design problem is divided into smaller and less complex design steps, which are connected through uni-directional constraint propagation. Near-optimality is achieved as constraints are not ignored and scalability is maintained as design iterations are not required.

- **Development of scalable and near-optimal intra-signal in-place optimization methodology.**
  The principles of the reusable methodology are applied in the context of intra-signal in-place background memory optimization to develop a scalable and near-optimal methodology for highly irregular access schemes. The methodology is described by a DSE framework with ordered sub-goals and the complete set of ordered cases per sub-goal.

- **Scalable and near-optimal representation based on patterns and pattern operations.**
  The context of irregular access schemes of the background memory has lead to the proposal of a representation, which uses patterns and pattern operation to near-optimally and in a scalable way describe the irregular access schemes. The pattern describe the access scheme per condition and the pattern operations are applied to consistently combine the patterns to describe the overall iteration space, when several conditions co-exist in the application code.

- **Scalable and near-optimal intra-signal in-place methodology for non-overlapping and overlapping store and load cases.**
  Scalable and near-optimal solution in closed form equations and functions, i.e. parametric templates, are proposed to solve the cases of each step of the developed intra-signal in-place methodology of irregular access schemes for the non-overlapping and the overlapping write and read access cases.

- **DSE methodology for the processing part, i.e. the instruction set processor mapping for an platform with a processor and several HW accelerators.**
  A scalable and near-optimal Design Space Exploration methodology is proposed for exploring the mapping of data-dominated and loop-dominated applications onto a partially fixed hardware platform with one microprocessor and several HW accelerators. Parametric templates are applied per methodology step. The proposed DSE methodology creates a partial pareto curve with the different near-optimal designs per mapping step. The partial pareto curve of the first step is propagated to the next DSE methodology steps and prunes sub-optimal options. The remaining options are merged with the propagated partial pareto curve to create a new partial pareto curve which is propagated to the next step. The process is repeated for all DSE methodology steps and the result is the final Pareto curve of the DSE.

- **Framework of near-optimal design-time scheduling techniques.**
  The partitioning and the ordering of the available options in scheduling and assignment techniques, applicable for foreground memory management and data path mapping, are defined

by a classification DSE framework. The classification framework efficiently describes the complete set of options of the exploration space and provides the constraint propagation between the different scheduling and assignment techniques.

- **Systematic methodology to develop parametric templates for near-optimal and scalable scheduling and assignment techniques.**
  A DSE methodology is proposed, which uses the classification framework as basis and projects it into a new framework taking into account the constraints of the target domain under study each time. The scheduling and assignment classes of the projected framework are described by parametric templates, which are combined following the class ordering and develop the final parametric template, which describes the scalable and near-optimal scheduling and assignment for the target domain.

The dissertation addresses the high layers of the DSE exploration and not at the final layers with the very detailed mapping into the hardware platform, i.e. scheduling and assignment mapping layers, without including code generation and net-list synthesis. As we propose a different type of DSE, the verification step of the design process is also a relevant research topic, which however is not addressed by the dissertation. Similar approaches to [85] can be re-projected to the methodologies described in this thesis.

1.4 Overview of Chapters

The dissertation consists of two parts. The first part consists of Chapters 3 to 5 and it is dedicated to the storing part of the embedded systems, i.e. background memory. The second part consists of Chapters 6 to 8 and is dedicated to processing part of SW/HW architectures of embedded systems. The dependencies between the chapters are depicted in Fig. 1.1. The overview of the chapters is:

- Chapter 2 describes a reusable methodology to develop and use scalable and near-optimal DSE frameworks. The DSE reusable methodology of Chapter 2 is applied in the different contexts of the dissertation to derive scalable and near-optimal methodologies.

- Chapter 3 defines the background memory mapping problem under study, i.e. the intra-signal in-place optimization, which remains scalable and near-optimal in irregular access schemes. The reusable DSE methodology is applied in the context of intra-signal in-place optimization to compose a framework with the general steps of the proposed scalable and near-optimal intra-signal in-place methodology. The goal is split into the intra-signal in-place sub-goal, i.e. the computation of final storage size, the translation sub-goal, i.e. the translation of the access scheme information into a scalable and near-optimal representation and the analysis sub-goal, i.e. the profiling of the access scheme information. The DSE reusable methodology is applied per sub-goal to develop an efficient partitioning of the potential cases. The parametric templates, which describe the solutions per case are described in Chapter 5.

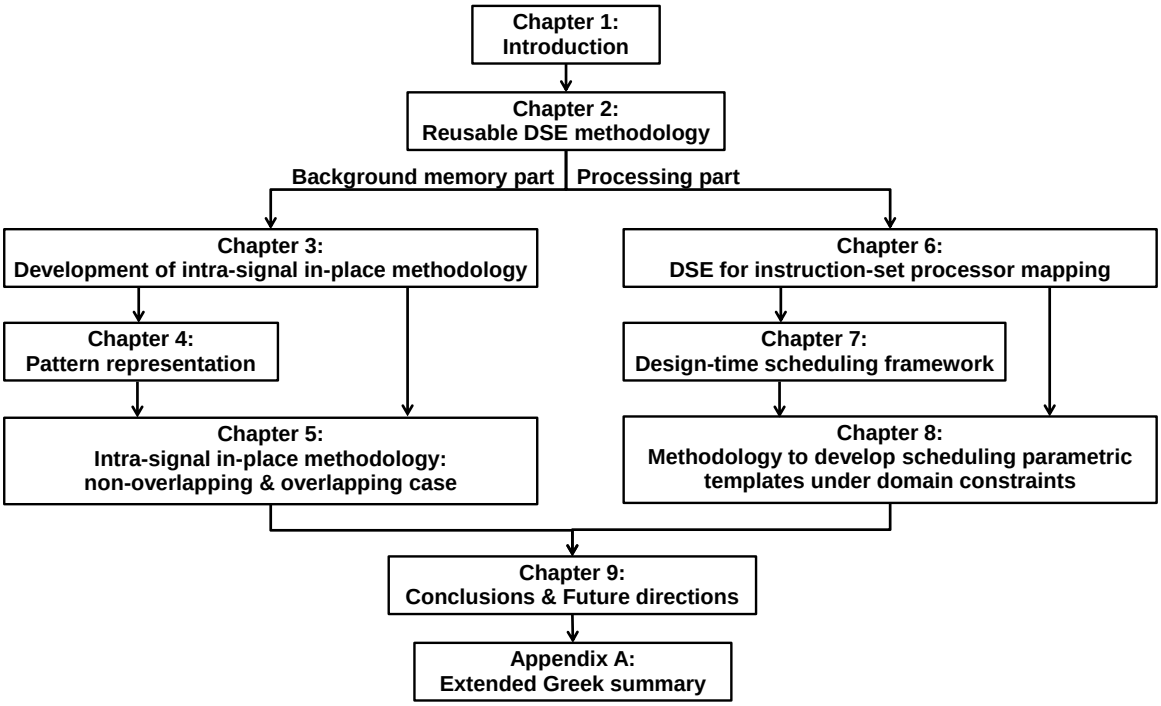


Figure 1.1: Dependency graph of chapters

- Chapter 4 describes the proposed representation of the array access schemes, which is appropriate to express the irregular shapes in a scalable and near-optimal way. A general pattern formulation has been proposed which describes the access scheme in a compact and repetitive way. Pattern operations were developed to combine the patterns in a scalable and near-optimal way under all the potential pattern combination cases, which may exist in the application under study. The complete set of potential combination cases is derived by the reusable DSE methodology of Chapter 2. The pattern representation is used to the solution to the translation cases.

- Chapter 5 describes the parametric templates of the developed methodology for the intra-signal in-place for irregular access schemes for the non-overlapping and overlapping store and load cases. The methodology consists of the analysis step, the translation step and the intra-signal in-place step. The analysis step describes the application domain and the different cases of the conditions that may exist in a unified application template and parses the required information from the application instance. The translation step describes the solution for the different cases using the patterns and the pattern operations described in Chapter 4.

- Chapter 6 describes a DSE methodology which creates the Pareto trade-off curve of the mapping of an application domain into a partially fixed architecture platform of instruction-set processors, which consists of one processor core and several custom hardware accelerators. The proposed methodology is derived by applying the principles of Chapter 2 in the context of mapping for partially fixed SW/HW architectures with a microprocessor and several accelerators. The methodology steps are the analysis and verification that the constraints are

satisfied through high level estimations, the SW/HW organization, the ForeGround (FG) memory management and the data path mapping. The parametric templates of each step are defined by the software and hardware parameters and constraints. The proposed methodology explores the space by profiling of the application and deriving the values in the parameters of the first step. The result is propagated to the next step, where options and values of the second template parameters are pruned due to the propagation from the first step. The valid options are merged with the propagated options of the first step and the process is repeated up to the Data-Path (DP) mapping step.

- Chapter 7 describes the result of applying the principles of the reusable DSE methodology of framework creation in the context of near-optimal design time scheduling techniques used in the Foreground Memory Management and Data-path mapping of the DSE methodology of Chapter 6. The scheduling and assignment of the scalars into the registers in the Foreground Memory management step and the scheduling and assignment of the operation into the function Units of the data path are the major mapping tasks. The result is a systematic classification with the complete set and the ordering of scheduling and assignment techniques for foreground memory management and data path mapping, which efficiently describes the available options of the exploration space.

- Chapter 8 applies the principles of the reusable DSE approach in the near-optimal design-time scheduling classification to create a systematic methodology to develop parametric templates for scheduling and assignment techniques which satisfy the target domain constraints. In this way, a scalable parametric template for scheduling and assignment tasks is created, which guarantees near-optimality for the domain under study. The developed template can be used in Foreground Memory Management step and Data-path mapping step when DSE is performed for the domain under study.

- Chapter 9 presents a summary with the conclusions of the dissertation and with a list of future work directions.

- Appendix A presents the publication list of the PhD dissertation results.

- Appendix B is dedicated to the extended summary of the PhD dissertation in Greek language.

[This page is intentionally left blank]

Chapter 2

Reusable methodology for scalable & near-optimal DSE

2.1 Introduction

Our goal is to provide a way to solve complex, dependent and large DSE problems in a near-optimal and scalable way. In this target domain, as shown in Chapter 1, the conventional solution techniques are less appropriate, because they are inherently based on bottom-up approaches without splits driven by constraint propagation. As a result, they either require too much exploration time to find near-optimal designs due to the high number of parameters and the correlations between the parameters of the target domain or they end up with a less efficient result in order to find a design within acceptable time.

In order to achieve near-optimal designs in reasonable exploration time, an approach which divides the problem into smaller sub-problems (to reduce exploration time) and which maintains the full functionality of the subproblems (to guarantee near-optimality) is required. When the design problem has been divided into small enough sub-problems, the more conventional design techniques can achieve near-optimal results in reasonable exploration time per sub-problem. The division of the design problem into sub-problems could be performed by enumeration of all the possible valid division options and combinations of those options. This enumeration process will lead to a huge explosion of the possible combinations and thus cannot be applied in large design problems. The alternative option is to apply a gray-box top-down approach, where instead of enumeration of all the possible division options, the different options are explored and refined in hierarchical abstraction layers and in a nested way. In each layer, the different options are grouped in cases with known interface (what aspect) but with unknown internal part (how aspect). The cases are not independent and in each layer the constraints are explicitly derived. The constraints show how the cases affect each other in a uni-directional way allowing an efficient combination of the diverse cases. Examples of the uni-directional propagation have been applied in the unified design meta-flow of [51] which have been applied in the abstract layers of design flow, as described in Section 2.5. The current PhD dissertation systematically presents the gray-box top-down approach

and applies it for DSE and mapping in the background memory and SW/HW co-design in the processor abstraction level of the unified design meta-flow. With this gray-box top-down process, the near-optimality is maintained, since constraints are not overlooked during combination. In addition, sub-optimal combinations are pruned by the uni-directional constraint propagation. The gray-box top-down process is iteratively applied starting from the problem to be solved in order to refine it in a complete set with all the cases and the constraint propagation between them. The refinement process is repeated until the cases are small enough to be near-optimally and within reasonable exploration time solved by more conventional techniques. Then, each subproblem is solved and the results are propagated following the uni-directional constraints.

In the rest of this chapter, we describe the principles of the reusable methodology to compose a scalable and near-optimal DSE framework in Section 2.2. Section 2.3 presents how the principles are applied to derive a complete DSE framework with the partitioning of all the available options into cases. Section 2.4 describes how the DSE framework is used to select the near-optimal and scalable solutions for the design problem of a target domain in a reasonable time. In the remaining chapters, we present the DSE frameworks derived by applying the reusable methodology. In this way we obtain scalable and near-optimal DSE frameworks in several contexts of large and complex design problems and we use the developed frameworks to determine effective solutions for the target domains under study.

2.2 Principles

The reusable methodology to develop scalable and near-optimal DSE frameworks is applied to a context in order to create the DSE framework with the partitioning of all the available options into cases. The framework derives by systematically applying a set of principles, which guarantee completeness during partitioning of the cases and provide uni-directional efficient ordering between the cases.

The first principle of the reusable DSE methodology is the gray-box top-down division principle and defines the space partitioning process.

Principle 1. *The gray-box top-down division principle rigorously applies top-down splits.*

Definition 1. *A top-down split divides the main characteristics of a parent case P into two children sub-cases $S1$ and $S2$, i.e. $S1 \subset P$ and $S2 \subset P$, by detecting an asymmetry in the characteristics. The sub-cases have the following requirements:*

1. *Complementary. The parent case P is spit into a sub-case describing the `X'' part and a sub-case describing the `not X'' part of the detected asymmetry, i.e. $\overline{X} = P - X$.*

2. *Positive. The `not X'' sub-case is reformulated into a positive description in order for both sub-cases to be positive, i.e. $S1 = X$ and $S2 = \overline{X}$.*

3. *Non-overlapping. The intersection of the sub-case is zero, i.e. $S1 \cap S2 = X \cap \overline{X} = \emptyset$.*

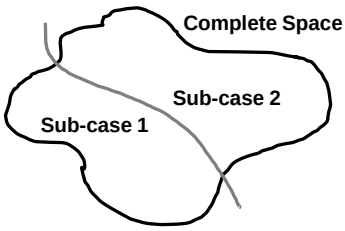


Figure 2.1: The partitioning of the complete space described by the black curve and the top-down split described by the gray curve, which divides the space into two sub-cases.

4. *Complete. The sub-cases together still compose all feasible options covering the total parent space P, i.e. $S1 \cup S2 = X \cup \overline{X} = P$.*

5. *Balanced. The size of the space describes by the sub-cases is similar, i.e. $S1 \approx S2$.*

Depending on the properties of the case, the detected asymmetry may describe the different instantiations of the parent case (top-down split of what type) or required steps to achieve the parent class (top down split of how type). This division also describes an asymmetry of what type: in the what type split one case is enough to instantiate the parent case, while in the how type both sub-cases are required to instantiate the parent case. A schematic description of the division of the space in a top-down split is depicted in Fig. 2.1.

The second principle of the reusable DSE methodology the horizontal propagation of constraints in a top-down split. It defines the ordering of the sub-cases based on the asymmetry of the split.

Principle 2. *The constraints of the sub-cases of a top-down split are described by horizontal propagation of constraints in a uni-directional way.*

In a top-down split, the source sub-case is solved first and the solution and the decisions made in the source are propagated following the horizontal uni-directional propagation of constraints to the destination case. The unidirectional-propagation of constraints is schematically depicted by an arrow between the sub-classes, as depicted in Fig. 2.2. The uni-directional ordering of the sub-cases is defined by a set of constraint propagation rules, which are instantiated based on the context of the problem under study. The constraint propagation rules are:

1. The source case provides parameters to the destination case, which are required in order to apply/perform the solution of the destination case.

2. The source case provides parameters to the destination case, which are required in order to define/select the solution of the destination case.

3. The characteristics and nature of the source case (heavily) prune unrealistic or sub-optimal case combinations.

4. The result and the decisions of the source case do not remove potential promising options in the destination case.

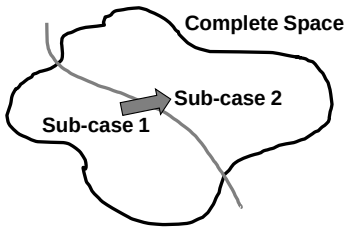


Figure 2.2: The partitioning of the complete space described by the black curve and the top-down split described by the gray curve and the arrow of the unidirectional constraint propagation.

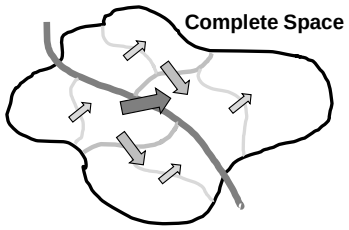


Figure 2.3: The partitioning of the complete space into the sub-cases and the sub-cases ordering by iteratively applying top-down splits and the horizontal propagation principle.

If no constraints exists between the sub-cases, they are independent and hence no arrow is imposed between the sub-classes. This is rarely the case in practice though.

The third principle of the gray-box top-down approach is the reusability principle, which is applied during the creation of the framework with all the available cases. When top-down splits are iteratively applied, we start from the more reusable splits. In this way, the more general sub-cases are at higher layers during the design problem partitioning, which can partially reused in similar goals and contexts. The splits in lower layers are refined more, and thus become more and more concrete.

Principle 3. *The reusability principle favors the application of the more reusable top-down splits first among the existing top-down splits.*

When the principles are applied into a context and a goal to create the framework with the partitioning of the available cases, top-down splits with horizontal propagation of constraints are iteratively applied. The result is an ordered partition of the cases, as depicted in Fig. 2.3. The first top-down split partitions the complete design space into two sub-cases, as depicted by the dark gray line of Fig. 2.3 and the horizontal propagation of constraints defines the direction of the dark gray arrow between the sub-cases. Per sub-case top down splits are applied, depicted by the gray lines in Fig. 2.3, which divide again each case into two complete and non-overlapping sub-cases. The final ordering is defined by the gray arrows. The process is repeated per sub-case to create further partitions in the space, e.g. light gray arrows etc.

The next set of principles is used after the creation of the framework with the partitioning into the available cases.

The fourth principle is the vertical propagation of constraints.

Principle 4. *The sub-cases of a top-down split should satisfy the properties of the parent case.*

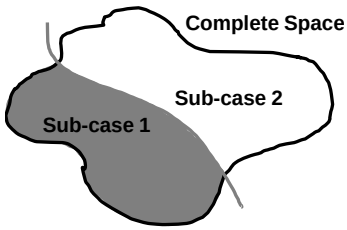


Figure 2.4: The pruning of a sub-case due to constraints vertically propagates to a top-down split.

The vertical propagation principle is responsible for the consistency of the sub-cases in a top-down split, when the split is used for a partial instantiation of the parent case. The partial instantiation of a parent case is more restricted than the initial parent case and thus inserts additional constraints. For instance, scheduling techniques are divided into stochastic and deterministic approaches. A partial instantiation of the parent case is represented by scheduling techniques intended specifically for dealing with small graphs. With the vertical propagation, the additional constraints are propagated to the sub-cases. If a sub-case is not compatible with the additional constraints, i.e. the options that describe are not valid under the propagated constraint, the sub-case is pruned. For instance, the additional constraint of small graphs is vertically propagated and prunes the stochastic techniques. In Fig. 2.4, the gray sub-case is pruned due to the constraint propagated from the partially instantiated parent case.

The fifth principle is the combination principle.

Principle 5. *When sub-cases are combined, the destination class should satisfy the properties and the solutions of the source class, which are propagated as design constraints by the arrows of the horizontal constraint propagation.*

The horizontal propagation of constraints has derived the direction constraints should be propagated between two sub-cases in order not to prune optimal solutions. Hence, when the solutions of the sub-cases are applied, the solutions of the sub-case in the source of the arrow are propagated to the destination of the arrow. Then, the solutions which are not compatible with the design constraints propagated by the solutions of the source class should be pruned to satisfy the horizontal uni-directional arrow. In this way, the sub-case is refined into a more specific sub-case.

In the next section we apply the principles to develop a methodology which creates DSE frameworks for a given goal and context and a methodology which uses the frameworks to derive near-optimal and scalable solutions for partially instantiated target domains, inside the goal and the context of the DSE framework.

2.3 Framework creation

In order to create the DSE framework with the partitioning and the ordering of all the available cases, the problem is defined and the principles of the Section 2.2 are systematically applied.

ALGORITHM 1: Pseudocode of the methodology to develop the framework with the partitioning of the complete space by applying the gray-box top-down principles.

Input: Context
Output: Framework
Define properties of context(v);
v $\leftarrow$ root ;
Q = list of parent cases;
Enqueue(Q,v);
while *($Q \neq \emptyset$)* **do**
    Select a parent P from Q;
    Identify relevant top-down splits within parent characteristics;
    Select based on reusability of candidate splits;
    Prefer more balanced splits, $|S1| \approx |S2|$;
    Apply split by creating $X = S1$ and $\overline{X}$ sub-cases;
    Positively reformulate $\overline{X}$ to S2 sub-sub-case;
    Apply horizontal uni-directional constraint propagation rules;
    if *(further refinement is required)* **then**
        Enqueue(Q,S1,S2);

2.3.1 Methodology steps

The steps of the reusable methodology to create the framework with the partitions of the complete design exploration space are summarized in the pseudocode of Alg. 1. The first step in the framework creation process is to carefully and unambiguously formulate the problem under study and the goal. The problem formulation and the goal are analyzed to identify the most important constraints, objective axes and properties. This step is very crucial since it describes the context of the framework. The root of the framework is described by the goal of the problem under study, which should meet the constraints. Then, the refinement process is applied. The root is divided into two smaller sub-cases by applying the DSE principles. The top-down split searches in the context of the problem under study and in the goal properties to detect an asymmetry in the space. The reusability principle favors the asymmetries that are more reusable and general in the initial layers of the top-down refinements. Then, the gray-box top-down principle is applied based on the detected and selected asymmetry. The result is two sub-cases, which describe smaller set of options that the root and satisfy the properties of the root. The size of the described space, i.e. the options, is similar between the sub-cases due to the balanced property. Each sub-case describes non-overlapping parts of the space and they, together, describe the complete space of the root. The horizontal uni-directional constraint propagation principle is applied in the top-down split to define the ordering of the sub-cases, except when they are independent. Based on the detected asymmetry, which is used in the top-down split, the corresponding constraint rules are applied. The result is an ordering of the two sub-cases connected with a uni-directional arrow, which describes the constraint propagation between the sub-cases. The negative sub-case is reformulated into a positive description, e.g. non-stochastic scheduling approaches are reformulated into the deterministic approaches. The top-down split with the uni-directional arrow is schematically de-

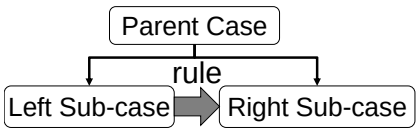


Figure 2.5: Schematic representation of the result of applying the principles of gray-box top-down approach in a parent case.

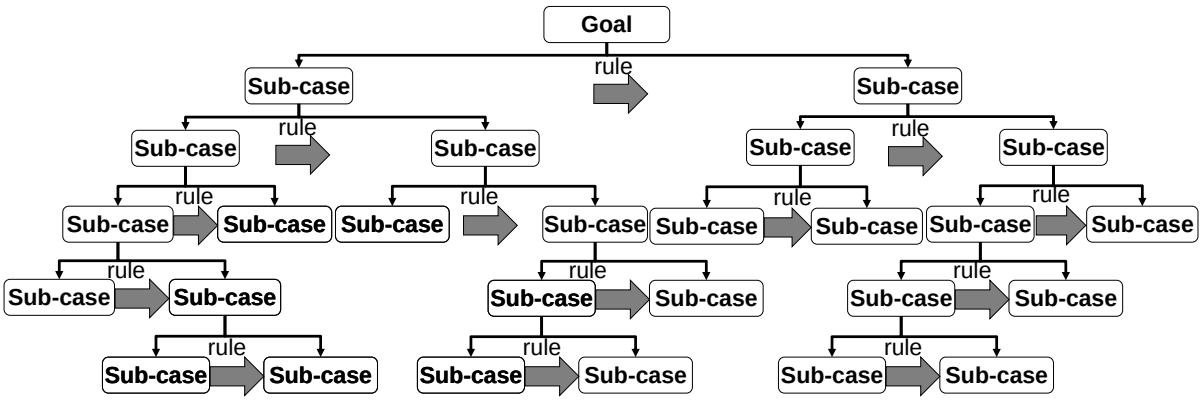


Figure 2.6: Framework with the sub-cases of the complete space of the problem under study after applying the principles of the reusable DSE methodology.

picted in Fig. 2.5, where the parent case is split into the left sub-case, which propagates constraints into the right sub-case and the rule above the uni-directional arrow describes the rule which is applied by the horizontal uni-directional constraint propagation principle to derive the corresponding arrow.

The process is iteratively applied for each of the sub-cases creating a tree structure T with the sub-cases and the unidirectional arrows, as schematically depicted in Fig. 2.6. The uni-directional arrows of higher layers are valid also in the sub-cases of the branches. The refinement process terminates, when the sub-cases are small enough to be solved by in a scalable and near-optimal way from bottom up approaches, e.g. specific conventional techniques or generalized techniques described by parametric templates.

2.3.2 Framework properties

2.3.2.1 Framework completeness

Each case i has a label $s(i)$ to indicate its position in T and describes the path from the root to case i. The elements of the label are L and R, where the L describes the Left sub-case and the R the Right sub-case in a top-down split in T. The length of the label $length(s(i))$ indicates the depth of the case in T (case layer).

Definition 2. *Two sub-cases i and j are called brother sub-cases when they derive from a split applied in the parent case p. They belong to the same layer of T, i.e. $length(s(i)) = length(s(j)) = length(s(p)) + 1$. The brother labels $s(i)$ and $s(j)$ differ in the last element. They derive from concatenating the parent label $s(p)$ with a L or a R element depending on the position of the child*

sub-case in T*, i.e.* $s(i) = s(p)|L$ *or* $s(i) = s(p)|R$.

Definition 3. *A couple of sub-cases* m *and* n *is:* $m - n$.

Based on the gray-box top-down division principle we remedy the fact that missing points exist, leading to the Theorem 1. The cases of the framework branches still cover all the possible options and describe non-overlapping areas of the search space, as characteristics and options loss cannot occur during the framework creation process. This completeness is crucial to guarantee a global DSE framework of all the different available feasible options, and to have near-optimality.

Theorem 1. *The derived framework provides a complete and consistent partitioning of the available cases of the goal under study.*

Proof. The process starts from the root, which describes the goal under study, and in each layer applies top-down splits which lead to complementary, non-overlapping, positive and complete sub-cases. This is enabled due to the positive reformulation of the sub-cases, which allows the splitting process to continue. Let's assume that the derived framework is not complete. Then an option exists which meets the goal properties and constraints but it is excluded from the framework. The characteristics of this option are not included in any of the proposed cases. That means that a split without the top-down split properties exists in the proposed framework. The latter means that the process to derive the framework does not strictly applies the gray-box top-down division principle, which contradicts the Principle 1. □

2.3.2.2 Compactness

The characteristics of the sub-cases can be shared following the constraint propagation arrows. This reuse allows the description of all options with a reduced number of unique sub-cases leading to a quite compact DSE framework, which is essential for an effective DSE.

Theorem 2. *The DSE framework consists of a limited set of the unique primitive sub-cases.*

Proof. Assuming that the DSE framework consists of more than the required sub-cases, then it is considered as redundant. Then, two sub-cases N and M describe the same area in the exploration space. If the redundant classes N and M are children of the same parent, this contradicts the properties of the top-down split, because they describe not complementary parts. If they are children of different parents, redundancy exists in their ancestors. Following the DSE framework tree in a backward way from the classes N and M to their ancestors, we can identify which splits are responsible for the redundant nodes. Due to the tree structure of the DSE framework, they will be children of the same parent which describe non-complementary set of characteristics, which contradicts the properties of the top-down splits. □

Theorem 3. *The DSE framework is a compact scheme.*

Proof. By Theorem 1 the DSE framework is a complete approach. By principle 2.4 the sub-cases are restricted by the characteristics of the parent. The Theorem 2 does not allow redundancy to exist. Hence, all options can be described by sharing of unique sub-cases indicating a compact scheme. □

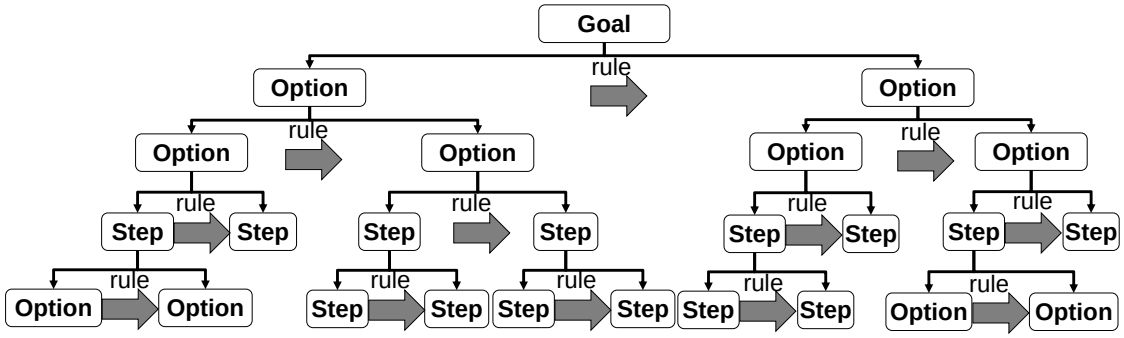


Figure 2.7: Combination of what type and how type splits in a framework. The sub-cases labeled options are derived by what type of top-down splits and the sub-cases labeled steps derive from how top-down splits.

2.3.3 Framework types

Depending on the properties of the goal and the context to be refined, the resulting framework may describe the different ways in which the goal can be achieved (what type framework) or may partitioning the goal into smaller tasks (how type framework) or a combination of a what and how type of framework. The hybrid framework of what and how type is achieved by applying what type of top-down splits in several layers and, then, per each sub-case of the what type top-down splits, we apply how type top-down splits and visa versa. A hybrid framework is depicted in Fig. 2.7. For instance, when the goal is to describe the complete set of existing design-time scheduling techniques, what type top-down splits are applied to define the complete set of different ways to perform scheduling. When the goal is to find the steps of a specific scheduling technique, e.g. Simulated Annealing (SA), how type splits are applied per scheduling case. We describe the creation of the resulting hybrid DSE framework for near-optimal design-time scheduling techniques in Chapter 7.

We apply the reusable methodology for scalable and near-optimal DSE framework creation in several chapters of the dissertation for different contexts. The most refined frameworks are worked out in the different DSE chapters. Chapter 3 describes how the different index expressions cases are defined (what type), how the intra-signal in-place methodology is divided into sub-goals (how type), and how each sub-goal is refined into the different cases (what type). The developed methodology combines the above frameworks, thus it is a hybrid framework. Chapter 4 contains the DSE framework for the intra-signal in-place step and it describes the different pattern combination cases. Chapter 5 continues with the further refinement of the sub-steps of the intra-signal in-place for non-overlapping and overlapping cases (what type), Chapter 6 define the sub-steps of mapping on platforms with one processor and several hardware accelerators (how type). Scheduling is a major problem to be solved during this mapping so Chapter 7 describes the classification framework for the creation of global design-time scheduling techniques (Hybrid type).

2.4 Framework usage

The framework describes the partitions and the ordering of the sub-cases which completely describe the space of the problem under study. When the goal is restricted, several of the sub-cases should be pruned due to incompatibility reasons.

2.4.1 Insight of the DSE options

The DSE framework has well-defined sub-cases describing non-overlapping areas which are reused and shared in a systematic way. Hence, the DSE framework offers useful insight into the sub-cases, as it efficiently supports their analysis. This is achieved by decomposing the properties of a restricted context and the goal into its primitive components, i.e. the sub-cases of the DSE framework. This enables the understanding of their similarities, differences, main characteristics and interrelationships of the different options in the DSE, which is essential for their efficient use and their further improvement and optimization. The DSE framework can precisely identify the similarities in every sub-case, as each sub-case is unique with non-overlapping characteristics. In addition, the DSE framework identifies the most relevant and unambiguous differences between the different sub-cases due to the asymmetries used to create the top-down splits. The similarities and differences do not form each other's direct complement. This process is essential for the in-depth understanding of the DSE options and thus for their efficient use and improvement with a minimized waste of time and effort comparing to the previous frameworks or to individual ad-hoc study of all the existing options.

2.4.2 Framework projection

The vertical propagation principle is responsible for the consistency of the splits in a DSE framework, when the DSE framework is used to select near-optimal solutions for a partially instantiated target domain. The target domain is a partial instantiation of the goal and the context of the DSE framework. For instance, assume a framework which describes the scheduling techniques, which are applied at design-time and that can search in the entire solution space for near-optimal solutions. That framework can be used for several partially instantiated target domains, such as scheduling of applications described by small graphs, of applications with large subgraphs and hidden hierarchy etc. The partially instantiated target domain is more restricted than the initial goal and context and thus inserts additional constraints to the parent case. Depending on the type of the goal and the context that the DSE framework describes, the result of the projection can be either an DSE methodology flow, the different required properties of a solution to the problem under study or the different cases that may occur during the problem solving.

The additional constraints of the restricted target domain are combined with the properties of the root leading to a restricted goal. The additional constraints of the restricted goal are vertically propagated to prune the initial DSE framework. The vertical constraint propagation propagates the constraints of the restricted goal down to the two sub-cases. First, for the sub-case, which is

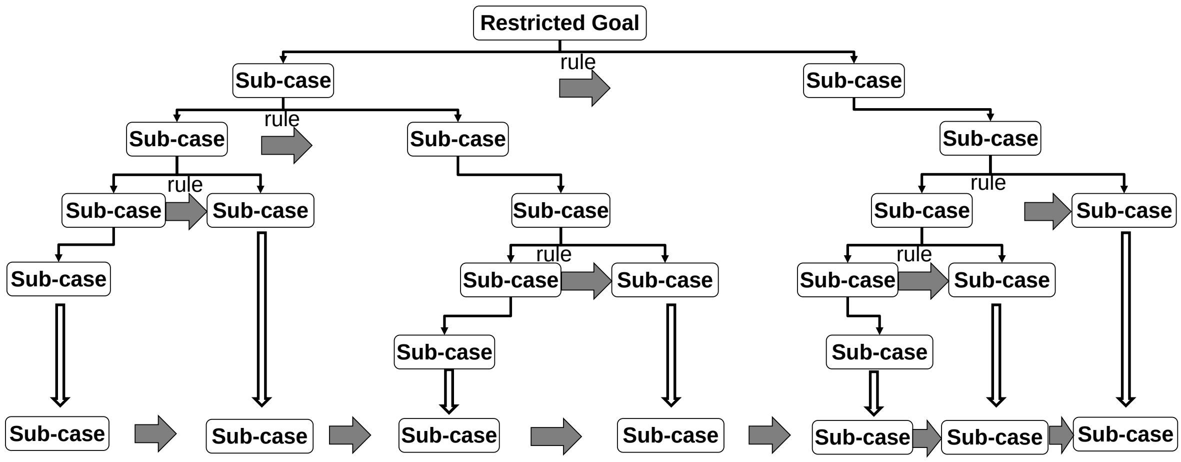


Figure 2.8: Pruned tree after applying the vertical constraint propagation principle. The characteristics of the sub-cases from the root to each leaf are merged and flattened into a sub-case.

the source of the uni-directional horizontal constraint propagation arrow, it is verified whether it meets the vertically propagated constraints. If it meets them, then the sub-case is not pruned. The vertical constraint propagation continues with the branch below the source sub-case. The sub-cases are compared with the vertically propagated constraints by verifying first the source sub-case of the unidirectional arrow etc. If the source sub-case is incompatible with the vertically propagated properties, it is pruned and the branch below the source sub-case is not further explored. Then, the destination sub-case is explored for compatibility with the vertically propagated constraints. After the process of the vertical constraint propagation, the framework has been pruned and the sub-cases that have been left are valid. In Fig. 2.6, the initial framework for the problem under study is depicted and Fig. 2.8 shows the pruned tree after the vertical constraint propagation. The result is a chain of the valid sub-cases following the uni-directional arrows, which derives after flattening the valid sub-cases. Flattening is performed by merging the characteristics of each leaf with the characteristics of the sub-cases that lead from the root to the leaf. After flattening, each sub-case can be filled with near-optimal and scalable conventional techniques or by parametric templates (see below) which describe the different options of each sub-case. The result is a chain of sub-cases connected through uni-directional arrows.

2.4.3 Trade-off exploration

During the framework creation process only the constraints of the problem under study have been taken into account. During the use of the framework, the trade-offs have to also be considered in order to guide the decisions towards the most promising solution of the design problem under study.

For each flattened sub-case of the framework, the different options can be defined by either a parametric template, which describes a generalized solution with parameters, algorithms and closed form equations, or by conventional existing approaches, which now are however near-optimal because the limited scope and size of each subproblem. In this way, they are also scalable for

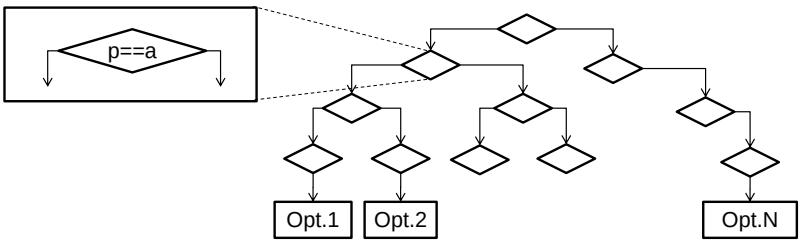


Figure 2.9: Schematic description of a parametric template. The boxes describe the options and the rhombus the if expressions to lead to the different options

the specific sub-case. In Fig. 2.9 we schematically depict a parametric template with the different options described in an efficient what-if structure. To have an overall Pareto curve to describe the overall trade-offs, the relevant trade-off axes have to be explored per sub-case and the combining principle describes how the trade-offs of the sub-cases are merged into the final Pareto curve. Initially, we estimate the value of the tradeoff axes for the different options in the first sub-case of the chain. In this way, a partial pareto trade-off curve with the near-optimal options and the corresponding decisions is created for the first sub-case. Based on the combining principle, the results of the first sub-case in the chain are propagated to the remaining sub-cases following the horizontal uni-directional constraint propagation. The options of the next sub-cases are explored taking into account the propagated results of the first sub-case. In this way, the options of the parametric template, which are incompatible with the decisions propagated from the first sub-case, are pruned and only the valid options in the parametric template of the second sub-case are considered. In Fig. 2.10, the gray options are pruned due to incompatibility with the propagated design constraints of the first parametric template. In this way the options of the second sub-case are explored under the constraints created by the valid options of the first sub-case. The pruning of the second sub-case does not remove promising solutions. The latter has been taken case by the horizontal propagation principle which decided the uni-directional arrows based on rules during the DSE framework creation. The result is a partial pareto curve, where the points describe the near-optimal valid points for both sub-cases. The process is repeated for all the sub-cases of the chain. In the last sub-case, the result is the final overall pareto curve, with points that describe near-optimal solutions for the problem under study.

The process is schematically depicted in Fig. 2.10, where we assume 3 sub-cases connected by uni-directional propagation of constraints. The options of each sub-case are described by a parametric template. The partial pareto curve of the first sub-case is created (P1, P2 etc. points in Fig. 2.10) by giving valid values to the parameters of the first parametric template. The results are propagated to the second sub-case, where the valid parameters of each point of the first sub-case prune incompatible options of the parameters of the second parametric template. The result is the second partial pareto curve, which describes the valid options of both sub-cases, i.e. points P1, P2 and P1, P3 etc. in Fig. 2.10. P1 is the point propagated from first sub-case and P2 and P3 are the only valid points of parametric template of second sub-case for the propagated values of P1, creating two pareto points. The final sub-case is explored based on the points of the propagated

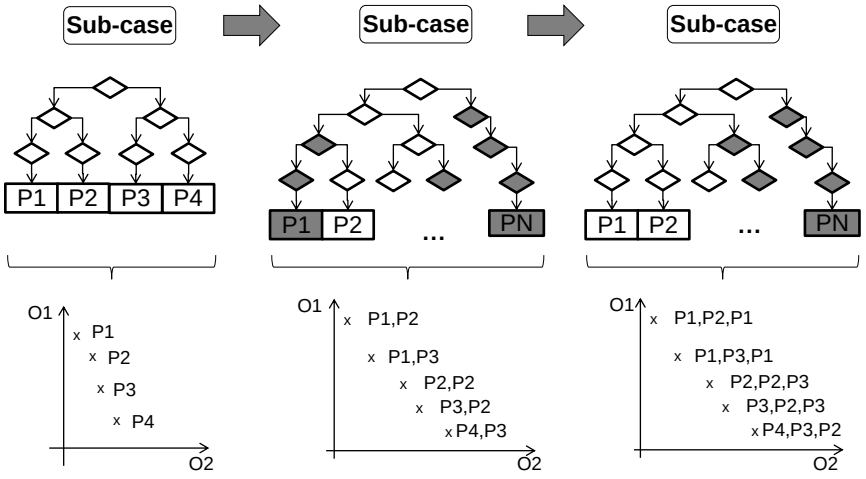


Figure 2.10: Three sub-cases with the exploration of the parametric templates. The parametric templates and the corresponding partial pareto curves are combined following the uni-directional constraint propagation.

pareto curve and the combined result is the overall pareto curve of the problem under study. The points of the final pareto curve are P1, P2, P1 and P1, P3, P1 in Fig. 2.10. The pareto curve can be used to select the most promising point based on the specification of the problem under study.

The use of the frameworks to perform scalable and near-optimal DSE is reapplied in several contexts in this dissertation. The most representative examples are in Chapter 5, where the intra-signal in-place storage size optimization methodology uses the unidirectional constraint propagation to remove suboptimal options of the parametric templates of the steps for the non-overlapping and the overlapping case, Chapter 6, where we present a design-space exploration methodology based on partial pareto curves and horizontal propagation of constraints for the context of mapping in a microprocessor and several hardware accelerators and in Chapter 8, where a methodology to develop scalable and near-optimal parametric templates for target domains in the context of near-optimal design-time scheduling techniques is presented.

2.5 Unified system design meta-flow

The principles of the DSE framework creation have been applied in the higher layers of the design flow of embedded systems in [21] [19]to develop an abstract but unified design flow, depicted in Fig. 2.11.

Due to the properties of the approach of Chapter 2, the unified design flow for global mapping DSE does not include overlapping design steps or redundant design iterations. Hence, it is separated into different abstraction layers which are connected through uni-directional propagation of constraints. The layers of the unified system design meta-flow derive from applying top-down splits in the context of system design. The goal is divided into the algorithm level and the Design Space Exploration. The algorithm level provides a complete and formal description of the system which should efficient deal with temporal correctness and the data types are refined. The result of

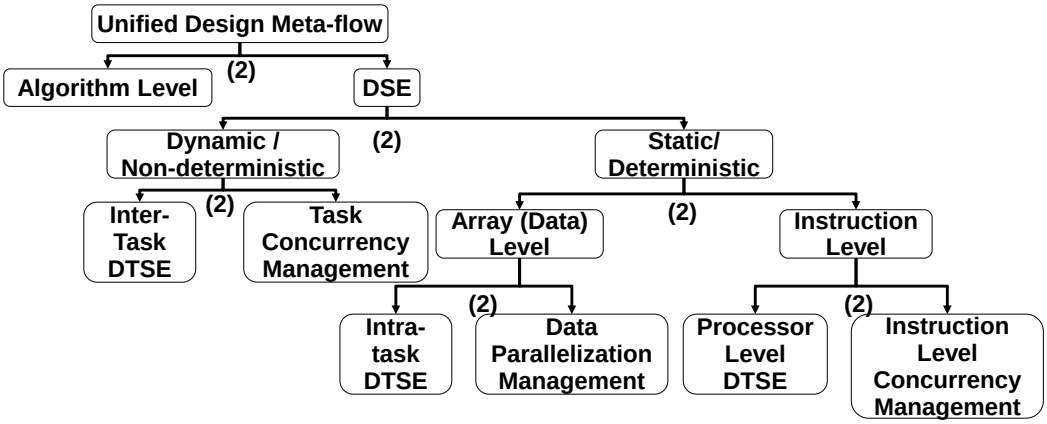


Figure 2.11: Unified system design meta-flow.

the algorithm level is an optimized system specification with a set of concurrent communication tasks with timing requirements. The DSE is divided into the part that is dedicated to the mapping of the dynamic and non-deterministic elements of the algorithm and the part which is dedicated to the mapping of the static/deterministic elements. The dynamic/non-deterministic part is dealt with by the stages at the task (or thread frame) level.

The Thread Frame Level (TF) mapping deals with the dynamism of the application which is expressed in a non-deterministic way. It mainly focuses on the not-stochastic events which dynamically trigger the creation of application thread frames. The Thread Frames are heterogeneous in nature. The use of the system scenario concept among thread frames, i.e. the inter-thread frame system scenario approach, allows to modify the thread frame behavior to have the overall application graph become more deterministic. However, the complete thread frame behavior cannot be fully transformed to a deterministic equivalent and thus a part of the thread frame remains non-deterministic. Since the thread frames are dynamic and heterogeneous, the hardware platform resources should also be dynamic and heterogeneous in order to support an efficient mapping of these thread frame characteristics to the hardware platform properties. The heterogeneous hardware platform resources should be described by a virtual layer to support the required dynamism, which results in a view based on abstract processors. In this way it is possible to dynamically assign parts of the existing heterogeneous resources to the concurrent thread frames. The task level is divided into the storage mapping stage, i.e. inter-task Data Transfer and Storage Exploration (TF DTSE) stage and the processing related mapping part, i.e. the Task Concurrency Management (TCM) stage. The TF DTSE stage decides for the assignment of the dynamically created and accessed data on the virtual run-time platform layer. The TCM stage performs the management of the concurrency between the thread frames; it explores the parallelization opportunities in the mapping of the dynamically created thread frames, i.e. the allocation of the dynamically created tasks to the virtual run-time layer heterogeneous processors; and it decides on the run-time management of the communication among the abstract processors, and up to the virtual memory hierarchy.

When the application has deterministic characteristics, the Array (or Data) Level (DL) and the

Processor Level (PL) or Instruction Level (IL) mapping are efficiently mapping the application. Deterministic characteristics can be manifest and static or data-dependent, which inserts dynamism. The deterministic dynamism is inserted due to the data dependencies in the conditions and the loops. It can be removed by using system scenarios inside the thread frames, i.e. the intra thread frame system scenario approach. The instantiations of the data dependent application with similar behavior are clustered in one intra-thread frame system scenario. Then, the worst-case instance inside each individual scenario is used as the representative case for further manifest analysis. In this way, for each scenario, the dynamism both among the thread frames and inside each thread frame has been converted to a manifest/static projection. Inside the resulting graph projection no dynamic task creation, event handling and synchronization or data-dependent conditions remain. In this way, these graphs (one per scenario) can be efficiently dealt by the Data Level and the Processor Level mapping.

The Data Level mapping is dedicated to homogeneous concurrency characteristics inside the thread frames of the heterogeneous Thread Frame Level mapping. A Thread Frame can be described by functions, which are clusters of non-uniform operations, and uniform operations among the clusters. The homogeneous software characteristics are defined by the operations which have uniform behavior, i.e. they are operations of the same type, they use the same type of operands (e.g. same word length) and the dependencies allow parallelism to exist. The uniform operations are mapped to an array of homogeneous super-processors with internal data memory hierarchy. The Data level is split into the Data Level DTSE (DL DTSE) and the Data Parallelization Management (DPM) stages. The Data Level DTSE stage decides the background memory management of the regularly accessed data of uniform operations inside a thread frame. The DPM is dedicated to the homogeneous mapping of the operations on the arrays of super-processors and the communication of the homogeneous super-processors.

The Processor Level mapping is dedicated to the remaining heterogeneous characteristics of the application, i.e. the functions with the operations which have non-uniform behavior. The non-uniform operations are mapped to heterogeneous resources, i.e. a super-processor with a data memory hierarchy, which include all the required resources to execute the functions of the processor level mapping. The instruction layer is split into the Processor Level DTSE (PL DTSE) and the Operations Concurrency Management (OCM) stages. The Processor Level DTSE is applied first to decide the background memory management of the accessed data of the irregular internal thread frame behavior, i.e. non-uniform operations. The Processor Level DTSE decision is propagated to the next steps of the OCM. The propagation is in this direction since the result of the Processor Level DTSE, i.e. the assignment and the access ordering of the arrays, is required for the address optimization, the communication and the intra-processor mapping. Moreover the background memory related decisions have a higher cost impact than the OCM decisions and they leave sufficient freedom for the OCM decisions to come up with near-optimal results (see [21] [19]). Finally, the circuit level stage maps the design on the target technology platform, i.e. custom IC or reconfigurable FPGA.

The unified system design meta-flow is not a specific detailed design flow for a specific ap-

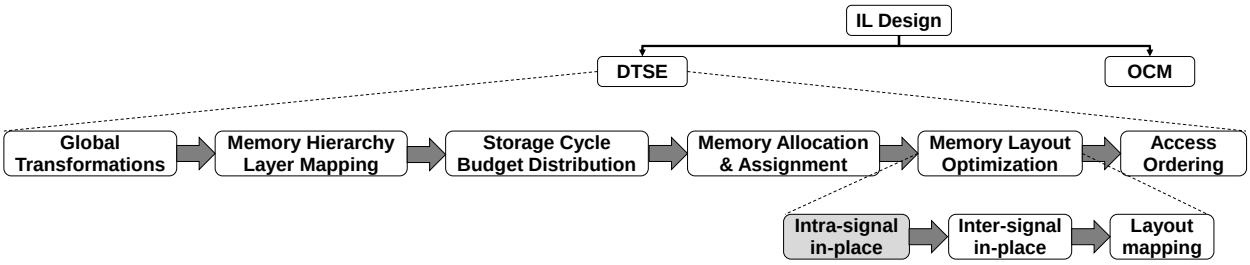


Figure 2.12: Design steps of instruction layer abstraction for DTSE.

plication domain and large research effort is needed to derive a customize flow for a specific domain [51]. For a specific domain, not all steps of the unified design meta-flow are crucial, as the characteristics and the constraints of the domain may be irrelevant to some steps. Hence, the irrelevant steps should be pruned when the customized design flow is created. For the remaining steps, appropriate techniques need to be selected or developed to near-optimally and in a scalable way implement the steps. The selection or development of the required techniques is domain dependent.

The scope of this dissertation is to further refine the steps of the abstract unified system design meta-flow in [21] [19], by applying the principles of the reusable methodology of Chapter 2 to create scalable and near-optimal DSE frameworks. In particular, we focus on the Processor Level of the unified system design meta-flow and apply the principles of the creation of scalable and near-optimal DSE framework to develop scalable and near-optimal DSE methodologies both for the background memory (PL DTSE) and the processing part (OCM). The steps of the PL DTSE have been presented in [20] and are summarized in Section 2.5.1. The steps of the OCM stage are described in [21] and are summarized in Section 2.5.2. The focus of the dissertation is to develop a complete methodology for near-optimal and scalable intra-signal in-place optimization design step of PL DTSE part, which is applicable to highly irregular access scheme, in contrast to polyhedral approaches of existing DTSE methodology. We also apply the principles to the OCM part to develop DSE methodologies for the target domain of a microprocessor with several HW accelerators and scheduling and assignment techniques for Foreground memory management and datapath mapping steps.

2.5.1 Processor Level DTSE

The Processor Level DTSE is dedicated to the management of data inside the application functions, which consist of non-uniform operations. The sub-steps of PL DTSE are described by the DTSE methodology in [20] and they are depicted in Fig 2.12.

The DTSE is split to the platform independent transformations and platform dependent mapping step. The platform independent transformations have as purpose to increase the data reuse and to improve the objectives, e.g. the access speed, the area, the power etc. They include data flow, loop and data reuse transformations. The platform dependent mapping step is further refined into the step that performs the memory hierarchy mapping and the memory layer mapping. The

memory hierarchy mapping is applied in a heterogeneous way, since the memories in the different layers have different characteristics, allowing an efficient use of the memory hierarchy. The memory hierarchy can be further divided into the platform-independent transformations to improve the initial access behavior and the actual mapping to the memory hierarchy platform. The latter is divided into the logical ordering and the assignment of the data to the memory hierarchy. The memory layer mapping is divided into the mapping of a few heterogeneous resources and into the mapping onto an increased number of homogeneous resources. This is the result of propagating the constraints from the real environment, where a high number of heterogeneous hardware platform resources cannot exist in reality and the mapping to a few homogeneous resources is trivial. The Processor Level mapping is dedicated to the non-uniform operations, which can have both irregular and regular accesses of the data. The irregular array accesses are mapped in the few heterogeneous resources mapping and the regular array accesses are mapped to the homogeneous resources mapping. The opposite option of mapping the regular accesses to heterogeneous resources is suboptimal and the mapping of irregular accesses to homogeneous resources is not possible.

In the case of the few heterogeneous resources a transformation step and a mapping step can be applied. The transformation step applies only the transformations relevant to the characteristics of the few heterogeneous resources. The mapping step describes the techniques to apply ordering in time and then assignment in space. In the PL DTSE, the ordering of the processor DTSE with few heterogeneous resources is described as Storage Cycle Budget Distribution (SCBD) and the corresponding assignment as Memory Allocation & Assignment (MAA). The ordering and assignment decisions are essential since they propagate constraints to the OCM steps.

In the case of several homogeneous resources, a similar distinction is applied. However, the transformations, the assignment and the ordering are dedicated to spatial characteristics. Hence, a set of spatial transformations is applied to increase the spatial locality. Then, a spatial assignment of the data to the homogeneous resources takes place, i.e. data layout. The data layout step applies especially the in-place mapping to reduce the required storage space, both for the intra-signal, i.e. accesses in the same array, and the inter-signal, i.e. accesses of different arrays. Then, the virtual memory data layout takes place where the data are assignment to a set of virtual memories. Finally, the physical memory data layout is composed by mapping the virtual memory data layout in the physical homogeneous memory elements of the platform. When a hardware-controlled cache is present, also a second data layout step is added related to conflict miss reduction. But that step will not be addressed here further. The result of the memory data layout is propagated to the special ordering to finalize the way the data are accessed.

2.5.2 Operations Concurrency Management

The sub-stages of the OCM stage described in [21] are depicted in Fig. 2.13. The first sub-stage of the OCM, namely the processor architecture integration or PAI, is dedicated to the internal structure of the heterogeneous super-processor, which is decided in terms of individual homogeneous

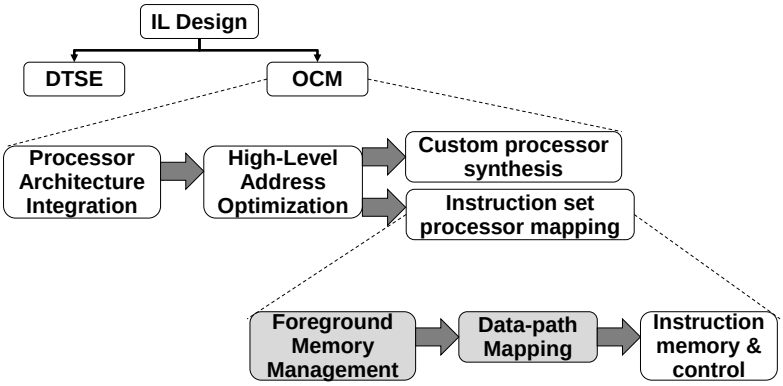


Figure 2.13: Design steps of instruction layer abstraction for OCM.

processor cores, e.g. instruction-set processor or custom processor cores, and the threads to be executed in the different processor styles are statically partitioned including decisions related to their communication and synchronization.

Then, the address optimization of the memory accesses of the array elements based on the result of the Processor Level DTSE is performed. The derived custom memory data layout includes complex addresses, which increase the overhead in the address generation unit and the memory accesses. The address optimization sub-stage is dedicated in the reduction of the complexity of the addresses and provide the final address sequence for the array data to the processor mapping, which is responsible for the actual hardware mapping for the address generation.

Then, the hardware mapping for all the application graph behavior is performed for the different processor styles. The processor mapping step should provide as a result a processor structure that is capable of executing both the address generation operations and the application arithmetic and logical operations. We will work this out in particular for the instruction-set processor style. Then, the processor mapping sub-stage takes as an input the decision of the processor style and decides over the Foreground Memory Organization, the Data Path execution mapping and the Instruction Memory/Control bits.

The foreground memory management step is dedicated to the local memory used to store the scalars or the individual array elements accessed from the data path. In the Foreground Memory step, if the assignment of the scalar data is over-constraint, then either spill to background memory is applied and the excess data are stored in the free space of the background data memory, or the foreground memory size is increased or the available cycles are increased up to the point where the constraints (e.g. timing deadlines) are met. In the foreground memory management step, platform independent and platform dependent transformations are applied, which can still modify the Control and Data Flow Graph (CDFG) graph of the application. Then, the ordering and the assignment of the scalars, which can be decided in this step, are performed on the heterogeneous memory resources of the foreground memory. If memory spilling is applied, several scalars have been decided to be stored in the background memory. Then, the background memory data layout and access ordering of the array data in the DTSE stage is updated to the final background memory access and data layout of both the arrays and the scalars. Since the array data

are the most dominant part of the background memory and communication overhead the result of the DTSE, the address optimization and the Processor Architecture Integration (PAI) communication (sub)stages are not modified. Hence, the result of the foreground memory management step regarding the background memory and the communication is augmented with the results of the previous steps. The data path mapping is dedicated to the allocation of the primitive operators, the scheduling and the assignment of the operations over the primitive operators, the potential pipelining and parallelization of the data-path. The foreground memory management step and the data-path mapping step will be discussed in more detail in Chapter 6.

2.6 Conclusions

In this chapter, we have presented the principles of the reusable methodology to develop scalable and near-optimal DSE frameworks for large and complex design problems. The problem under study is divided into a complete set of sub-problems connected with uni-directional propagation of constraints, i.e. a framework. The completeness of the framework achieves near-optimality, since sub-optimal approximations are avoided, and constraint propagation achieves scalability during exploration. We have described the process of applying the principles to create the corresponding framework with the partitioning and the ordering of the sub-cases. Then, we have described how the derived framework can be used to provide an efficient exploration of the solutions for (partial) instantiation of the framework goal. Finally, we have summarized the unified system design meta-flow created by uni-directional constraint propagation and define the steps which form the starting point and context of the work in the dissertation, i.e. the intra-signal in-place step and the instruction-set processor mapping sub-stage. In the latter we focus especially on the foreground memory management step.

Part I

Background memory management methodologies

Chapter 3

Development of intra-signal in-place methodology

3.1 Introduction

Storage size management techniques search the minimum number of resources required to store the elements, without imposing an inefficient addressing during element accessing. The storage size management techniques are applied in several domains, e.g. in the scratch pad memories of embedded systems [37], in the hardware controlled caches of the general purpose systems [18] and in factory storage management systems in industry, such as cargo systems [108]. The reduction of the number of the resources is essential, as it is directly coupled with the system cost, area and energy [18]. In the embedded systems, the power cost is heavily dominated by the storage of arrays, thus the memory units compose a large part of the overall cost [18]. For commonly used embedded applications, such as image, video and signal processing, where the dominated data are arrays, the organization of the array data storage becomes a very essential part of the overall design process. A less efficient organization leads to overestimation of the resources, which directly increases the requirements in memory size and chip area, which increases the system energy consumption. To achieve a globally near-optimal storage organization, both the individual size mapping for each array, i.e. intra-signal in-place optimization, and the concurrent size mapping for the arrays, i.e. inter-signal in-place optimization, are essential steps [42]. We present a methodology for the intra-signal in-place optimization, which remains scalable and near-optimal in highly irregular access schemes. A similar approach can be used to derive a scalable and near-optimal inter-signal in-place methodology, which is left as future work.

Existing techniques for storage size management and intra-signal in-place optimization are enumerative, symbolic/polyhedral or apply a worst case approximation of the storage size, as described in Section 3.3. The enumerative techniques lead to the optimal size, but they are not scalable. Hence, when the number of array accesses is increased, the exploration time reaches unacceptable values. The symbolic approaches, which are mainly polyhedral approaches [42] [31], are scalable to the number of accesses, but operate efficiently only in solid iteration spaces [194],

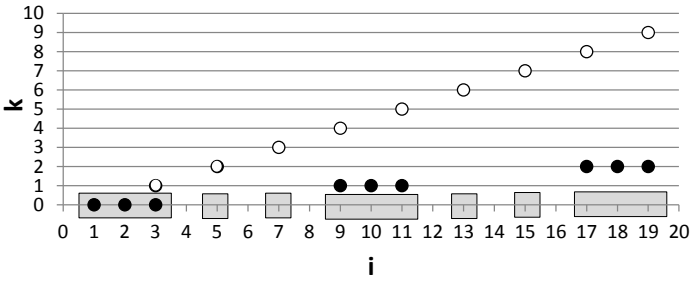
i.e. access schemes without holes. With additional preprocessing, the symbolic approaches are applicable up to piecewise regular spaces (e.g. [55]), i.e. iteration spaces with quite regularly placed holes. A polyhedral efficiently defines the storage size for one geometrical partition in the iterations space, i.e. accesses in the iteration space that are either regular and can be represented by lattice or have tiny holes that are approximated to solid regions. When several diverse geometrical partitions exist, the polyhedral approaches either require too much exploration time or they have to approximate too much leading to overestimation of the resources. Several geometrical partitions exist. when the irregularity of the holes is increased, due to condition statements which disturb the regularity of the access statements. Then, the symbolic approaches are not applicable any longer or they have to approximate the access scheme to the worst case situation [160]. The approximation approaches (e.g. [160]) consider the holes of the iteration space as solidly filled with accesses, leading to sub-optimal storage size result. Hence, a methodology for intra-signal in-place optimization for irregular access schemes, which remains near-optimal and scalable, is highly desired.

The intra-signal in-place optimization methodology should take into account all the access statements and the relative condition statements for one array in order to define the minimum storage requirements during the application execution. The applications in the target domain under study differ significantly in nature and thus they have different structures and condition and access statements in their code. Hence, the applications create different intra-signal in-place optimization cases, which require a different optimization process to achieve a near-optimal storage size. It is not possible to have a fully general intra-signal in-place step, which can provide a near-optimal result for all applications in a scalable way. Hence, to achieve both scalability and near-optimality of the intra-signal in-place optimization methodology, a general methodology which is split into steps with solutions dedicated to each possible intra-signal in-place case is required.

In this chapter, we apply the DSE framework of Chapter 2 to the goal of developing such a scalable and near-optimal intra-signal in-place methodology in the context of complex iteration spaces with irregular accesses created by array access statements in a loop structure with manifest conditions. In Section 3.2 we motivate our approach with an illustrative example. Section 3.3 presents existing approaches for intra-signal in-place and storage size optimization. Section 3.4 describes the target domain and the problem formulation, which describe the goal and the context. We apply the principles of the reusable DSE methodology of Chapter 2 to define all the possible index expressions of the applications of the target domain. We identify the most relevant index expression for the intra-signal in-place methodology based on how commonly it is used and how many index expression cases can be mapped through transformations in this case. We select this case as representative case to be used in the general intra-signal in-place optimization methodology. We show how we can map the remaining index expressions to the representative case and controllable approximations of the intra-signal in-place for the index expressions, where transformations are not applicable. Section 3.5 describes the sub-goals and the set with the different cases per sub-goal. We select two representative cases of position of write and read access statements to develop the intra-signal in-place step of the proposed methodology in chapter 5.

```
For (i=0; i<N; i++)
 For (k=0; k<M; k++)
  If ((i>8k)&&(i<8k+4)||(i≠2k))
   A[i]=...
  EndIf
 EndFor
EndFor
```

(a)



(b)

Figure 3.1: Motivational example: (a) Application code with three conditions and a write access statement and (b) Iteration space with the accesses to array A.

3.2 Motivational Example

When the access statements are regularly executed, the iteration space is solid. A condition disturbs the regularity of accessing an array, creating holes in the iteration space. When several conditions coexist, even in the case where the condition statements create regular holes, their combination leads to significantly complex and irregular iterations spaces. To illustrate the main problem, we use the example in Fig. 3.1. The application code consists of: two for loops over the iterator i and k with N and M are upper bounds, three manifest condition statements combined through an AND and OR operation and a store access statement. When N and M are increased, the enumerative approaches require too much CPU time, so they are not scalable. A schematic representation of the initial part of the iteration space with M=10 is depicted in Fig. 3.1(b). When only the conditions (i>8k)&&(i<8k+4) exist, the access shape consists of repetitive holes in a regular way (illustrated by the black dots in Fig. 3.1(b)). This access scheme can be efficiently represented by symbolic approaches [55] and the storage size is (near-)optimally computed. The symbolic approaches are inherently scalable and near-optimal for solid iteration spaces and for piece-wise regular iteration spaces, i.e. iteration spaces with quite regular holes. However, they cannot be directly applied to iteration spaces with increased number of irregular holes. Then, approximations would have to be applied, which insert suboptimalities, as holes of any size can be considered as accesses. By adding the i≠2k condition (white dots in Fig. 3.1(b)) in our illustration, the repetition of the first access shape is disturbed and irregularity is inserted, making the symbolic representation inappropriate. Hence, a scalable and near-optimal methodology for iteration spaces with irregular holes is highly required.

In realistic applications, the loop iterations are highly increased, several condition statements are combined in expressions and several array access statements co-exist in the application code, which create the irregularity in the iteration space. In this chapter, we present how to develop a methodology for near-optimal and scalable intra-signal in-place optimization, which handles this complex irregular iteration spaces, as explained in detail in the next sections.

3.3 Related Work

The existing storage size management techniques are enumerative, symbolic or worst case (heuristics). The enumerative approaches are optimal, but not scalable when the number of accesses is increased. The symbolic approaches are scalable and near-optimal for solid iteration spaces or piece-wise regular iteration spaces. Hence, they are not appropriate for iteration spaces with increased number of irregularly spread holes. Then, they have to approximate the iteration space, which leads to sub-optimal results, since holes are considered as accesses.

3.3.1 Enumerative approaches

Enumerative storage size management techniques describe each access individually. An enumerative storage size optimization approach is e.g. the symbolic evaluation for background memory size estimation based on enumeration of the indexed signals of each index expressions [135]. Ref. [183] proposes a custom memory data layout, i.e. placement of the elements into the memory resources, focusing on parallelization of memory accesses based on the access pattern. Each access in the array access pattern is analyzed and the accesses of independent array elements are placed in separate partitions. The enumeration of the memory accesses is described through reference lists-based schemes without applying any reference compression [143]. The enumeration of memory accesses is usually achieved through profiling and instrumentation tools. A profiling based strategy to generate a memory access trace and a heuristic approach to exploit the scratch pad memory hierarchy is proposed in [29]. Ref [8] enumerates through profiling the iterations, where the memory is accessed, to derive the information for selecting candidates for data remapping [146]. Ref. [164] searches all possible memory data layouts by iteratively prototyping candidate data layouts and evaluating them on a representative trace of memory accesses. SHMAP [49] tool annotates the application and collects memory reference traces for arrays. Gleipnir [76] collects memory access traces and associates each access with the corresponding internal structure. Pin [118] provides an instrumentation platform to create a trace of address and size of memory instructions. In [202] profiling is used for data accesses through the instrumentation framework presented in [139]. Although the enumerative approaches are optimal, when the number of accesses is increased, the exploration time is unacceptable.

3.3.2 Symbolic approaches (including polyhedral techniques)

Other storage size management techniques are symbolic and apply solvers to compute the storage size. For instance, in linear constraint-based schemes the array accesses are expressed as convex regions in a geometrical space [143]. The storage requirements derive from the integer points inside the convex regions of accesses. The symbolic approaches are scalable and near-optimal in solid iteration spaces, as they efficiently represent the bounds of one convex region. For instance, simplified constraint-based forms (e.g. [7]) are used to describe solid iteration spaces, e.g. diagonal or triangular shapes, which are not applicable in iteration spaces with holes. Other

symbolic approaches, with some extensions, can also efficiently handle piece-wise regular spaces. For instance, triple notation [177], i.e. lower bound, upper bound and stride per dimension, has been used to describe regular spaces. Vectors have also been explored for storage size management. Ref. [30] focuses on spatial locality optimization using utilization vectors to describe array references. Ref. [75] uses memory access vectors, the loop nest depth and the array dimension. In [82] an access matrix describes through the loop nest the array accesses to explore data locality. Distance vectors with data access matrices are used, which are, however, applicable for uniform references [160]. The survey in [148] describes several symbolic techniques for estimation of storage requirements. Polytope theory is commonly used for regular spaces, e.g. the iteration space is represented by placing polytopes of signals in a common place with ILP techniques [90]. Estimation on storage requirements with a partial fixed ordering through polytopes is proposed in [89]. IMEC Atomium [19] supports memory related steps through interactive or in a more automated way based on the polyhedral dependency graph [195]. The Data Transfer and Storage Exploration (DTSE) methodology uses the polyhedral dependency graph to explore the memory data layout optimization step. In [206] a data access graph based on polytopes is used to describe all the memory operations in time for a given array, which is used as input to the data reuse exploration and decision step of the memory hierarchy design. Ref [33] applies polytope theory to memory partition and scheduling problem. Philips' Phideo [115] is mainly oriented to stream-based video applications and represents the iteration space as linear function of the iteration index. In [38] lattice matrices are used to explore the memory allocation problem. In [110] a mathematical framework to study modular memory allocations through strictly admissible lattices is presented. Ref. [173] proposes a lattice intersection approach to count the integer points in Z-polytopes. SUIF [123] and PIPS [35] add additional constraint to the linear constraint-based representation to deal with a hole. In practical contexts, the application consists of several access and condition statements, which create irregularity in the array accesses and a high number of irregularly placed regions in the geometrical space. Then, the symbolic approaches are less appropriate, because to be applied, the access regions have to be widen in order to form a convex hull, i.e. holes are considered as accesses, reducing quality. If additional symbolic constraints are used to describe the high number of irregular holes, the exploration time is highly increased [143].

3.3.3 Approximation approaches

When the symbolic storage size management techniques are applied in complex iteration spaces with irregular holes, a worst case approximation is used, i.e. the invalid parts are considered as valid parts solidifying the iteration space. For instance, Ref. [160] approximates the number of distinct references of non-uniform access statements based on the values of the index expressions on the loop bounds. Other approximations are performed by compilers and techniques which usually assume a fixed memory data layout, e.g. row-wise or column-wise, and transform the code to improve the performance of the data accesses [214], [27]. When the application iteration space does not match with the assumed one, the tools can easily produce sub-optimal solutions.

The next section describe how to develop a proposed methodology to apply storage size management for intra-signal in-place optimization, which remains scalable and near-optimal for irregular access schemes.

3.4 Problem Formulation & Target Application Domain

3.4.1 Problem Formulation

The storage size management techniques are applied in memory units of the lower layers of on-chip background memory that can be controlled, e.g. scratch-pad memories in the embedded systems [37] and hardware controlled cache memories in the computer systems [18] and in industrial domain for the cargo storage systems [108]. The intra-signal in-place storage size optimization problem is to find the minimum storage requirements for a group of elements, e.g. an array, given a finite set of accesses. The minimum storage requirements derives from the maximum number of concurrently alive array elements. Scalability is of great importance, since the increase in the number of accesses may lead to prohibited exploration time. Near-optimality is crucial, as sub-optimal results lead to overestimation of the storage resources increasing area, cost and power consumption. Since the storage size result is used in the next mapping phases, i.e. inter-signal in-place optimization, memory access scheduling and memory address generation [18], the quality of the intra-signal in-place result affects the quality of the result of the next phases. An efficient address scheme is required for the overall good performance of the system and it is achieved when the addresses are regular. For instance, the address generation and the accessing of sequential elements is very efficient, e.g. when the burst mode of the memories can be efficiently used. Hence, the intra-signal in-place result should not impose increased irregularity to the address generation.

The cost function to be minimized is the size of the required memory to store the elements of an array. When the size is not minimized, the amount of required resources is overestimated, which increases the area and thus the energy consumption. The energy per memory access is also increased, since the memory partitions are increased due to the overestimation of the resources. The overestimation of the resources also affects indirectly the performance. When the data are fetch from the background memory, the process is performed in groups of data (e.g. defined by the width of the bus to the memory) to have a gain in performance by hiding cycles. When holes in the iteration space are considered as accesses, several of the data that are fetched are useless. In addition, this leads to increase of the data which may now not fit in the cache hierarchy increasing the misses and the performance loss and the energy due to increase in the transfers in the memory hierarchy. Hence, a tradeoff exists between the performance and the memory size.

Our goal is to develop a systematic and scalable methodology to near-optimally solve the intra-signal in-place storage size optimization problem for the target domain under study, described in Section 3.4.2, without imposing significant overhead to the address generation. The methodology can be applied with different control parameters to create a Pareto curve with the different memory requirements, i.e. from optimal to controlled near-optimal, and provide this information to the next

ALGORITHM 2: Parametric structure to describe the application instances of the target domain

a_{l}^{1}=..., a_{k}^{1}=... ...
for (*i*=LB_{i};$i < UB_{\mathrm{i}}$;*i*++) **do**
  ...
  for (*k*=LB_{k};$k < UB_{\mathrm{k}}$;*k*++) **do**
    S^1:...=$F1(A[f_{\mathrm{x}}^{iterator}]..[f_x^{A_{\mathrm{Dim}}}])$
    S^2:...=$F2(A[f_{\mathrm{x}}^{iterator}]..[f_x^{A_{\mathrm{Dim}}}])$
    ...
    if (C^1(*i*,a_{i}^{1},$type_{\mathrm{i}}^{1}$)..) **then**
      ...
      if (C^2(*k*,a_{k}^{1},$type_{\mathrm{k}}^{1}$)..) **then**
        ...
        S^3:...=$F3(A[f_{\mathrm{x}}^{iterator}]..[f_x^{A_{\mathrm{Dim}}}])$
...
for (*j*=LB_{j};$j < UB_{\mathrm{j}}$;*j*++) **do**
  ...
  for (*l*=LB_{l};$l < UB_{\mathrm{l}}$;*l*++) **do**
    ...
    if (C^1(*j*,a_{j}^{1},$type_{\mathrm{j}}^{1}$)..) **then**
      ...
      if (C^2(*l*,a_{l}^{1},$type_{\mathrm{l}}^{1}$)..) **then**
        ...
        S^1: ...=$F4(A[f_{\mathrm{x}}^{iterator}]..[f_x^{A_{\mathrm{Dim}}}])$

memory optimization steps to compose the final pareto curve.

3.4.2 Target Application Domain

The target domain consists of applications of one thread frame, which has deterministic behavior, i.e. consists of several condition statements and nested loops, but without including any event triggered task generation or non-deterministic elements. The applications are highly loop-dominated and data dominated, i.e. a high percentage of the executed code handles indexed array signals in the context of loops. For instance, high-speed data intensive applications in the fields of speech, image and video processing, which require significant amount of storage resources [77]. Manifest condition statements on the iterators and one to many loop nests with single and multiple dimensions in the array create complex iteration spaces with irregular holes. The data dependent conditions and bounds can be translated to static based on the concept of the System Scenarios [161]. The ranges of the data-dependent conditions and the performance of the corresponding application is explored and grouped in scenarios. Per scenario, the representative application code is used and mapped to the application template. The target domain uses single assignment code, i.e. every array element can be written (stored) only once, but read (load) several times. We focus on the non-overlapping and the overlapping write and read access schemes, i.e. for the former case all write statements are executed before all read statements and for the second case writes and reads can be concurrently executed in a loop iteration.

The applications of the target domain are described by the parametric structure of Alg. 2. This representation is useful, as it is used as input to the analysis step and characterizes the application instances of the target domain. The control flow consists of loops and conditions. The parameters to describe the loop structure are: the loop dimension $Loop_{\mathrm{Dim}}$, the low loop bound LB_{Iter}, the upper loop bound UB_{Iter} and the iterator step ST_{Iter} for each loop iterator. The condi-

tions are expressed through manifest data-dependent control statements: C^X(iterator, expression, comparison-operator) describes a condition of: ``iterator comparison-operator expression'', e.g. $C^1(i, 5, <)$ describes i<5 condition. The iterator dependent kernels are executed when the corresponding manifest control statements are valid, i.e. the iterator has specific values. The array access statements include the function $Fx(A[f_{\mathrm{x}}^{iterator}]..[f_x^{A_{\mathrm{Dim}}}]))$. Fx is a function of accessing arrays, fx is the index expression per dimension, which describes a set of array accesses to the background memory, A_{Dim} is the number of array dimensions and A is the array being accessed. Array A is used as an example to describe the arrays of the application domain and many arrays similar to A may exist in the application, for which the methodology is repeated. By parsing the input application code, the values of the parameters of the parametric kernel can be identified. Memory optimization steps before intra-signal in-place have decided over which elements are considered as arrays and which as scalars. The target domain does not include scalar values, which are mainly stored in the foreground memory and not in the background memory. The scalar data layout is explored in later design steps, i.e. Foreground Memory management.

In contrast to existing approaches, the developed methodology maintains the scalability and near-optimality properties, when it is applied to the target domain of iteration spaces with several holes irregularly spread, as shown in Chapter 5.

3.4.3 Analysis of index expression (fx function)

The intra-signal in-place methodology depends on the type of the index expression of the array access statements, i.e. fx. Different in nature index expressions propagate different characteristics to the intra-signal in-place optimization methodology, which require different in nature solutions to derive near-optimal result. We define the complete set of the index expressions cases, we select the representative index expression case to be used in proposed intra-signal in-place methodology and provide transformations and controllable approximations for the remaining index expression cases.

We apply the principles of the reusable DSE methodology of Chapter 2 to partition the available options, which can potentially occur in the applications of the target domain under study, into a set of cases, which describe the different index expressions and their characteristics. The result and the splits in the index expression cases are depicted in Fig. 3.2. The number under each split describes the rule that has been applied to derive the ordering of the cases, as described in Chapter 2.

3.4.3.1 Index expression framework

The first split in the index expression types is between the index expressions of **general piece-wise affine** and **general non-piece-wise affine** index expressions. The general non-piece-wise affine index expressions are characterized by highly irregular structure of the accesses in the iteration space and the access regions that they describe cannot be divided into a set of non-degenerate geometrical regions, where an affine expression describes each access region. The geometrical region includes tiny irregularly spread holes, which do not allow the split in piece-wise regions. The

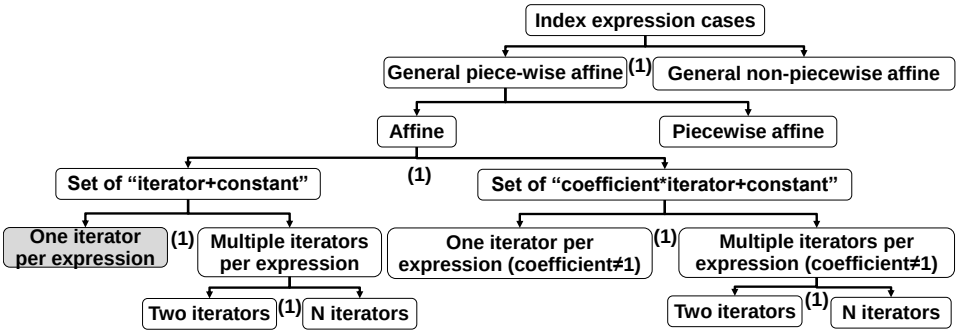


Figure 3.2: The set with the index expressions cases after applying the principles of the reusable DSE methodology.

index expressions of this case are irregularly modified with the iterators. The general piece-wise affine expressions potentially describe a highly irregular structure of accesses, but the asymmetry with the non-piecewise case is that the accessed regions are non-degenerate intervals described by affine index expression. In this sense, they internally consist of pseudo regular structures. For instance, the index expression 4*i describes an access to the array every four iterations and the index expression 2*i+1 describes the odd elements. When both co-exist, the describe an irregular access scheme. Hence, the piecewise affine propagates the solution to the general non-piecewise affine case, since the latter it requires it in order to provide a solution. The intra-signal in-place solution for the general non-piece-wise affine index expression case is to approximate the irregular structure of the accesses through a piece-wise affine convex hull. In this way the solution of the general non-piece-wise affine index expression case is provided by the solutions of the piece-wise affine index expression case.

The second split inside the general piece-wise affine case is the **affine** and the **piece-wise** affine index expressions. The affine case describes index expressions are determined by a basis of n vectors, which are not necessarily orthonormal. Hence, the axes of such an expression are not necessarily mutually perpendicular nor have the same unit [203]. In a way, affine expressions are a generalization of linear expressions. The piece-wise affine case describes the cases when the accesses can be split into geometrical regions, which are are regularly placed due to affine expressions in each region. Hence, transformations exists which can modify the geometrical regions and map them to a regular geometrical space with different set of vectors. For instance, a piecewise affine expressions, such as a modulo operation over the iterator, e.g. (i%a)+b, is translated into a affine expression and the solution of the affine case is used by the solution of the constant modulo operator. The constant modulo inserts an extra loop with size equal to the constant and the index expression becomes k. For instance, for the index expression of i%4, a one k loop from 0 to 4 is inserted and the index expression becomes k. Hence, the affine expressions propagate the solutions to the piece-wise affine expressions, since the latter are mapped through transformations to the affine expressions.

We further refine the affine index expression case into the cases of the set of **``iterator+constant''** index expressions (e.g. i+b) and the set of **``coefficient*iterator+constant''** in-

dex expressions (e.g. a*i+b). The asymmetry between these cases is that no coefficient is used to multiply the iterator, thus the index expression describes accesses to sequential elements. The set of ``**iterator+constant**'' index expressions propagates the solution into the set of ``**coefficient*iterator+constant**'' index expressions, since the solution of the latter uses the solution of the former. The set of ``**coefficient*iterator+constant**'' index expressions case is translated into the set of ``**iterator+constant**'' index expressions case and a set of parametric manifest conditions, which describe the holes in the accesses in the iteration space due to the coefficient. For instance, index expression 2i+1 is expressed as index expression i+1 and a parametric condition i==2k. Hence, the access statement with index expression i+1 is executed only when the i has even value. This transformation is always possible, since it is one-to-one mapping, i.e. the coefficient and the constant of the index expression create one manifest condition. In this way, the sub-cases of the ``coefficient*iterator+constant'' are translated to the corresponding index expressions sub-cases of the ``iterator+constant'' and use their solutions. The proposed methodology can always deal through the representative index expression case with the ``coefficient*iterator+constant''. The opposite transformation, usually used by symbolic approaches, as described in Section 3.3, is not always possible, since a simple combination of manifest conditions may not lead to a closed form index expression. For instance, (i>8k)&&(i<8k+4) requires enumeration of the different accesses and the corresponding iterator step increase, i.e. 8i+1,8i+2 and 8i+3. When expressions with different coefficients co-exist, the transformation may not possible. In this cases, the existing symbolic approaches cannot be applied, because their transformations can only support few different coefficients without highly increasing the number of constraint equations, e.g. [55] [160] [6].

The set of ``iterator+constant'' type of index expressions is further refined into the cases of **one iterator** per expression (e.g. i+b) or **multiple iterators** may exist in one expression (e.g. i+j+...+b). The multiple iterators case is divided into the **two** iterators (A[i+j+b]) and **multiple - N** iterators (A[i+j+...+b]) case. The results of the N iterators case derive by generalizing the 2 iterators case. The asymmetry between the one iterator and the multiple iterators cases is that the second case couples different iterators. The one iterator case propagates the solution to the multiple iterators case. When ``multiple iterators'' co-exist in an index expression, they are replaced by a new global iterator that describes the accesses in the coupled dimensions. Internally, the coupled iterators can use symbolic approachers, i.e. polytope approaches are applied, which achieve optimal results, when the accesses are described by one or few geometrical regions, and provide approximations, when the accesses are parametrically described by geometrical regions with holes between. Similar splits can be applied to the index expression of the ``coefficient*iterator+constant'' case. It may may have **one iterator** per expression, e.g. A[a*i+b], or **multiple** iterators, e.g. A[a1*i+a2*j+..+b]. In the case of multiple iterators exist in the index expressions, we divide into the case of **two** (A[a1*i+a2*j+b]) and **multiple - N** iterators (A[a1*i+a2*j+...+b]).

3.4.3.2 Representative case

We select the representative index expressions based on the occurrence in real-life applications, the number of remaining index expression cases that can be mapped to the representative case and the convenience in presenting the developed intra-signal in-place methodology principles. The representative case is the pure iterator of ``iterator+constant'' type, depicted by the gray box in Fig. 3.2. The representative index expression case is the most occurring case in real-life applications and in several benchmark suites, e.g. [159] [107] [63], the ``iterator+constant'' type of index expressions is highly used. The selected index expression allows a better explanation of the intra-signal in-place methodology principles. The remaining index expressions can be transformed to the ``iterator+constant'' case, since the pure iterator case combined with the manifest condition statements of the application may describe solid iteration spaces or iteration spaces with regular and irregular holes. In summary, the proposed methodology with the representative index expression case can near-optimally handle all piece-wise affine index expressions by applying the appropriate preprocessing transformations and provide controllable approximations for the remaining cases.

3.5 Development of intra-signal in-place methodology

The goal is to develop an intra-signal in-place methodology, which defines the minimum storage size required to store the elements of the application of the target domain, in a near-optimal and scalable way. By applying the principles of Chapter 2, the goal is divided into a set of sub-goals starting from the final sub-goal and propagating to the first sub-goal through ``how'' type of top-down splits of Chapter 2. As depicted in Fig. 3.3, the first split, based on this constraint analysis, is between the final sub-goal, i.e. the intra-signal in-place sub-goal which computes the final storage size, and the required preprocessing to have the information to reach the intra-signal in-place sub-goal. The preprocessing is divided into the second sub-goal, i.e. the translation of the access scheme information into a representation, which supports near-optimality and scalability of the intra-signal in-place step, and the third sub-goal, i.e. the analysis step, which maps the application instance to a unified parametric template and extracts the information for the access scheme.

Per sub-goal we define the set of all the available diverse cases to create the methodology step by applying the principles of Chapter 2. Initially, we define the cases of the intra-signal in-place sub-goal having as context the target domain and as goal a scalable and near-optimal intra-signal in-place optimization. The relevant part of the unified template is the loop nest structure, the condition types and the type of the condition expressions for the representative index expression case and also the position of the write and the read access statements, as described in Section 3.5.1. The cases are still abstract as they describe a large group of similar application instantiations. The result is propagated to the next sub-goal in order to take into account what it will be needed in the next step during methodology execution. The step of the translation sub-goal is defined by applying principles of the reusable DSE methodology, described in Chapter 2, having as context the target domain and as goal a scalable and near-optimal intra-signal in-place optimization and taking into

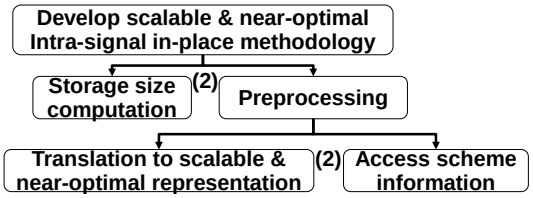


Figure 3.3: The sub-goals of the intra-signal in-place methodology after applying the principles of the reusable DSE methodology.

account the cases propagated by the intra-signal in-place sub-goal. The results are propagated to the analysis sub-goal.The different cases of analysis are defined based on the intra-signal and translation cases propagated from the previous sub-goals. The analysis cases are a set of primitive conditions and primitive operations in a unified parametric template, which support all the cases propagated from the translation and the intra-signal in-place sub-goals. The methodology steps are described by parameterized templates, i.e. structures with closed form equations, algorithms and parameters, which provide scalable and near-optimal solutions per step case.

The corresponding steps per sub-goal are the intra-signal in-place step, the translation step and the analysis step. When the methodology is applied to an application instance, the steps are executed in the opposite direction, but with only uni-directional dependencies between them, as described in Chapter 5. The process of the analysis step is the parsing and the mapping of the application instance to the unified parametric template and the extraction of the access information and which primitive condition cases are valid for this application instance. This information is propagated to the translation step to select the translation cases that are valid for the specific application. Then, the results are propagated to the intra-signal in-place step to compute the final storage size by selecting the valid intra-signal in-place cases based on the propagated information. The division into steps is required to achieve a scalable and near-optimal methodology avoiding the bi-directional dependencies in the final steps to be executed. Scalability is achieved during the execution of the methodology, as the propagated values from the previous steps select the active cases in the next step step by efficiently comparing the propagated values with the different possible cases. This uni-directional propagation allows to keep the overall complexity low due to the avoidance of global iteration loops, as no bi-directional dependency circle occurs. Near-optimality is achieved, as the different cases of the steps are grouped based on similar characteristics and the available cases describe all the possible options. Hence, no instance exists that does not fit into a dedicated case, i.e. it provides a near-optimal solution. In this way, solutions which are inappropriate for a specific instance cannot be used.

In Section 3.6 we present the parametric template of the analysis step. The analysis step is the parsing of the essential information. It is used for the intra-signal in-place mapping and other memory optimization steps, thus it can be reused. The scalable and near-optimal parametric templates with closed form equations and algorithms to compute the storage requirements of the translation and the intra-signal in-place step are described in Chapter 5. However, to support the solutions of the cases of translation step, we define a scalable and near-optimal representation for

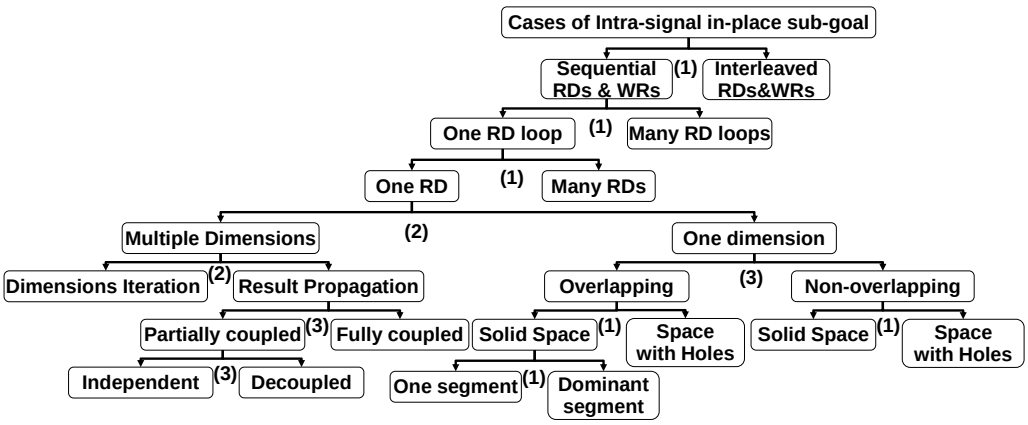


Figure 3.4: The set with the cases of the intra-signal in-place sub-goal after applying the principles of the reusable DSE methodology.

the information derived by the analysis step in Chapter 4.

3.5.1 Intra-signal in-place cases

Based on the number of loops, the number and the ordering of the write and the read statements, we define the complete set of the cases that may exist in the applications of the target domain under study. We select the representative cases to compose the parametric intra-signal in-place templates and provide transformations and approximation for the remaining cases.

We apply the principles of the reusable DSE methodology in the context of the structure of the application and the goal of computing the final storage size, The derived different cases, are depicted in Fig. 3.4, where the WR stands for WRite and RD for ReaD access statement. The numbers under the top-down splits describe the rule applied to derive the ordering of the sub-cases, as explained in Chapter 2.

The first split is based on the sequence of the different types of accesses and divides the available options into the **sequential** case and the **interleaved** case. The sequential case has all the write access statements executed before the read access statements. For instance, the sequential case is valid when only one write access statement exists in the application code, i.e. single-assignment code, when the write statements are in different loop nests and they are independent or when hey are in the same loop nest following a sequence of writes and then a sequence of reads statements. In the interleaved case, the write and the read statements are interleaved in the application code, thus write statements may exist after the read statements, The interleaved case is active when the statements are combined in the same loop nest. The sequential cases propagates the solution to the interleaved case to by used as solutions for the interleaved case after applying some transformations. The interleaved case can be split into couples of write and read statements that are sequential. Then, per couple, the solutions of the sequential case are applied.

3.5.1.1 Sequential case

Focusing on the sequential case, the next split derives between **one loop** with read statements or **multiple loops** with read statements. The one read loop propagates the solution to the multiple read loop nest case, which it uses to create the solution for the multiple read loop case. The latter is transformed by applying a virtual loop merging. The result is one loop that includes all the read statements. Then, the solution of one read loop is applied.

In the case of one loop with read statements, a further division derives if only **one read statement** or several **read statements** exist in the loop of the application code. The one read statement case propagates its solution to the several read statements case. The latter uses the propagated solution, as the several read statements can be merged into one statement with inserting additional manifest conditions. The inserted manifest conditions allow the statement to be correctly executed. The next division is based on the number of dimensions in the RD statement, i.e. the case of **multiple dimensions** and the case of **one dimension**. The multiple dimensions case propagates the required information to the one dimension case in order to select the corresponding sub-case and apply the solution. The case of multiple dimensions is divided into the **iteration over the dimensions** and the **propagation of the result** between the dimensions. The iteration over the dimensions provides the required information to apply a consistent propagation of results between dimensions. The iteration over the dimensions selects the next dimension to be explored based on the order of the loops. The dimension selection is performed through the loop dimensions, from the outer to the inner dimension. The outer loop dimension is explored first, because it heavily affects the storage occupation of the inner loop dimensions. This information is propagated to the one dimension case. The propagation of the result is further refined into the **decoupled dimensions** and the fully **coupled dimensions**. The result of the partially coupled dimensions is propagated to the fully coupled dimensions, since the solution of the fully coupled dimensions cannot be used by the partially coupled dimensions. The partially coupled dimensions are split into the fully **independent** dimensions and the **decoupled dimensions**. The partial storage size of the fully independent dimensions is calculated and the propagation is performed in a straightforward way, as the complete size of both dimensions is used. In the decoupled case, the partial storage size can be defined separately, but the combined storage size is computed by using parts of the partial storage sizes. The solution of the independent case is propagated to the decoupled case. The decoupled solution adds or removes parts of the size computed by the independent cases, depending on the type of decoupling.

The next split in the one dimension case is derived by the potential overlapping between the iterations instances while executing the write and read statements. Depending on the difference in the index expressions of the write and the read statements and the iteration range of the statements, the write and the read statements may be **overlapping** or **non-overlapping** in the iteration space. When the statements are overlapping, in at least in one iteration both the write and the read access statements are executed. In the non-overlapping case all the write instances of the write statement are executed in previous iterations than the read statements. The overlapping case is first in the

ordering in order to avoid non existing combinations, i.e. if the non-overlapping case is valid, then all the remaining inner dimensions are non-overlapping. The non-overlapping case is further split into the **solid iteration space** and the **iteration space with holes**. The solution of the solid iteration space is propagated to the iteration space with holes, as it can be used to provide the solution per segment between the holes. The overlapping case has a similar split into the **solid iteration space** and the **iteration space with holes**. The solid iteration space of the overlapping case is further refined into the **one segment** and the **dominant segment** case. In the one segment case, no holes exist. In the case of the dominant segment, several segments exist, but the storage size is derived only by the dominant segment, since it covers the storage size of the remaining segments. The solution of the one segment is propagated to the dominant segment as it is used for the dominant segment solution. This split is not valid in the non-overlapping case, where all segments have to be taken into account to compute the storage size, since no overlapping exists.

3.5.1.2 Combinations

The combinations of above different cases are also valid cases. When both sequential and interleaved cases co-exist, the interleaved cases are transformed into a coupled combination of sequential cases. Then, the storage size is computed per sequential couple and the global size is given by the dominant couple, i.e. the storage size of the couple that requires more storage size. The combination of the multiple dimension solution and the one dimension solution describes the solution to compute the storage size for all the dimensions. The multiple dimension selects the outer dimension and propagates to the one-dimensional case to apply the overlapping or the non-overlapping solution and have the first partial storage size. The process is iteratively repeated over all the dimensions by verifying if the dimension is overlapping or non-overlapping, applying the corresponding solution and merge the result with the propagated partial storage size from the previous dimensions. When the outer dimension is non-overlapping, this is propagated to the inner dimensions, which are considered as non-overlapping. If the outer dimension is non-overlapping, then the inner dimensions cannot be overlapping. In the case that reads have not been merged into one read statement and both overlapping and non-overlapping cases occur, then the dominant case gives the storage size, if reads are referring to overlapping to different iteration values.

The parametric templates for the computation of the storage size of the different cases and the combination of the cases are described in Chapter 5 for the two representative cases of non-overlapping and overlapping write and read access statements in Section 5.3 and Section 5.4, respectively.

3.5.2 Translation cases

The translation sub-goal is to provide a scalable and near-optimal representation of the access scheme, which avoids enumeration of the accesses and efficiently describes the potential irregularly spread holes of the iteration space. Hence, we determine the different cases that the translation step will have to represent having as constraints the cases of the intra-signal in-place sub-goal. Hence,

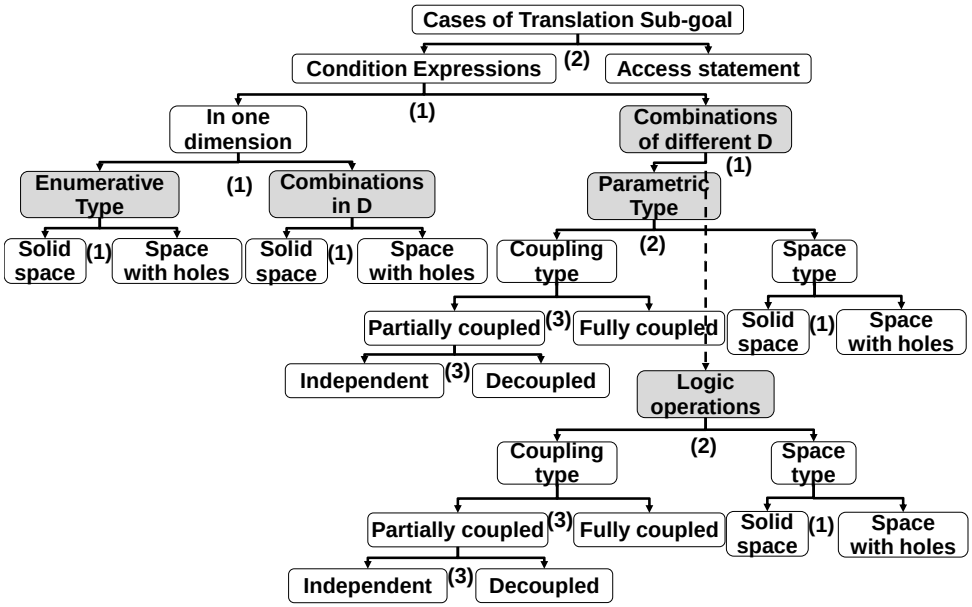


Figure 3.5: The set with the translation cases after applying the principles of the reusable DSE methodology.

the translation cases should support all intra-signal in-place cases, meeting both intra-signal in-place and translation sub-goals.

The first splits derive from the access information which is relevant to the access scheme, as it will have to be translated to find the in-place. The first split is the **condition expressions** and the **access statement**. The condition expressions propagate their solution to the access statement which is combined with the solid space described by the solution of the access statement to have the final representation solution. The condition expressions are further divided into the condition expressions in **one dimension** and **combination of conditions in different dimensions**. The result of the one dimension condition expressions is used as part of the solution the combinations of different dimensions case. The conditions in one dimension case are divided into the **enumerative type** and **combinations in one dimension**. The enumerative conditions use one dimension and a constant value. The enumerative type is propagated to the combinations in one dimension, since the translation of the combination case used the translation of the enumerative conditions in one dimension. The combinations of different dimensions is divided into the **parametric type** of conditions and the **logic operations** which combine conditions of different dimensions. The parametric conditions use two iterators of different dimensions and describe a coupling between dimensions and the space in a parametric way. The parametric type is propagated to the logic operations, since the parametric result is used to compose the result of the condition expression in the translation step.

The parametric conditions, the enumerative conditions and the combinations translation cases in one or multiple dimensions satisfy the intra-signal in-place cases propagated from the intra-signal in-place sub-goal. Since the goal is to support the cases of the intra-signal in-place step, we further refine the type of intra-signal cases that the translation cases can cover. The propagated constraints from the intra-signal in-place cases describe asymmetries which create top-down splits, which are

refined in a similar way as for the low layer splits of the intra-signal in-place. The enumerative type is refined based on the type of spaces that can describe, i.e. **solid space** and **space with holes**. The solid space is propagated to the space with holes as the latter consists of several segments of solid space. The space that the combinations in one dimension can describe is **solid space** and **space with holes**. The parametric conditions are further split into the **coupling type** and the **space type** that they describe. The coupling type is propagated to the space type, as it the type of space described is affected by the type of coupling between dimensions. The coupling type can be either **partially coupled** or **fully coupled**. The partially coupled propagate constraints into the fully coupled, otherwise the fully coupled case will dominate over the partially coupling case. The partially coupling case is divided into the **independent** or **decoupled** case. In a similar way, the independent case propagates the solution into the decoupled case, otherwise the decoupled case would be dominant over the independent case. Similar splits are applied to define the coupling and space type that describe the logic operations in combinations of different dimensions case.

The solutions for the representation for parametric, enumerative and their combinations are described in Chapter 4 and in Chapter 5, where we illustrate how they are applied in the translation step of the intra-signal in-place methodology.

3.5.3 Analysis cases

The Analysis sub-goal maps the applications of the target domain into a unified parametric structure, which provides the information for the access scheme to the next intra-signal in-place steps, when the methodology is executed. The translation cases and the intra-signal in-place cases propagated from the previous sub-goals should be satisfied by the cases of the analysis sub-goal.

The first splits derive from the application information that has to be propagated to the translation and intra-signal analysis step to define the storage size. The first split consists of the **loop structure** and the **kernel**. The information of the loop structure is propagated to the kernel, as the value of the iterator affects the kernel statements. The loop structure is further refined into the **dimensions information** and the **dimensions order**. The information of the dimensions is required to define their order, e.g. we have to know which dimensions exist in order to define an ordering between them. This information is relevant, as it is required from the intra-signal in-place step in order to apply the parametric templates. Then, the kernel is split into the **condition expression** statements and **access** statements. The condition expression statements are propagated to the access statements, as their result affect the execution of the access statements. The condition expressions statements are further refined into condition expressions in **one dimension** and **different dimensions**. The result of the one dimension condition expressions is used as part of the solution the combinations of different dimensions case.

The conditions in one dimension case are divided into the **enumerative type** and **combinations** of enumerative conditions. The enumerative type is propagated to the combinations, since the latter combines the enumerative conditions in one dimension. Since the goal is to provide a unified representation, which represents the application of the target domain under study, the enumerative

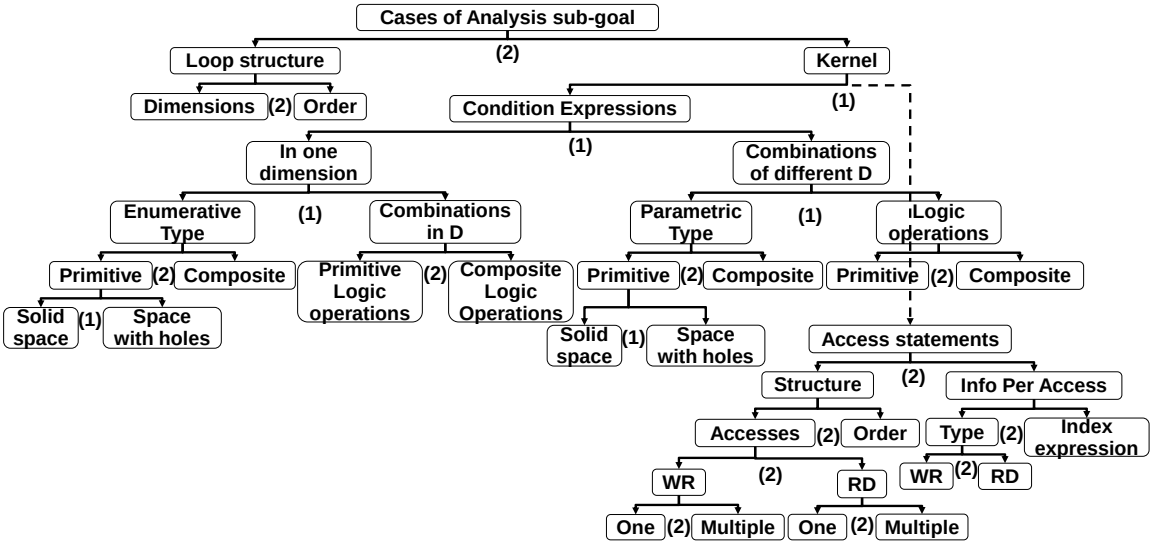


Figure 3.6: The set with the analysis cases after applying the principles of the reusable DSE methodology.

conditions are divided into the **primitive enumerative** and the **composite enumerative** conditions. The primitive conditions are propagated to the composite solutions, since the latter are transformed into primitive conditions. The primitive enumerative conditions are used in the analysis and they can describe solid iteration spaces and iteration spaces with holes. The combinations are also divided into the **primitive logic operations** and the **composite logic operations**. The composite logic operations are transformed into the primitive logic operations.

The case of combinations of condition expressions in different dimensions is divided into the **parametric type** of conditions, which are capable of coupling two iterators of different dimensions and the **logic operations**, which also couple conditions of parametric or enumerative type between different dimensions. The parametric type is propagated to the logic operations, since the latter is applied on top of conditions. The parametric type of condition is further refined into the **primitive parametric** and the **composite parametric** conditions. The primitive parametric is explored in the analysis step, whereas the composite type solution is created by using the solution of the primitive type by applying transformations to the primitive type. In a similar way, the logic operations are divided into the **primitive logic operations** and the **composite logic operations**.

The access statements are divided into the **accesses structure** and the **information per access**. The access structure provides information to define the one access case. The access structure is divided into the **access statements** and the **access statements order**. The former propagates to the later, as the information of the accesses is required to provide the order between them. The accesses are further refined to the **WR** access type and **RD** access type. Both are divided into **one** or **many** access statements. The one access statement propagates the solution to the many access statements, as the latter can be transformed to the one access case.

In a similar way with the translation refinement, the lower layer splits are similar to the lower layer splits of the translation cases and the intra-signal in-place cases. This similarity is created due to the propagated constraints from the previous sub-goals. The major goal of the analysis step

is to be able to provide the required information for execution of the translation and intra-signal in-place step. For instance, the dimensions and the order are required from the intra-signal in-place step in order to decide in which way the patterns are explored. The information of the enumerative type, the parametric type and the combinations of conditions are required for the translation step to decide the active translation case and translate the conditions into patterns and perform the corresponding operations. The primitive conditions and operations of the unified application template should support solid iteration spaces and iteration spaces with holes, as required by the propagated constraint from the intra-signal in-place step. The information of the structure analysis of the access statements is required as it is partially used from the translation case and used in the intra-signal in-place step in order to select which intra-signal in-place case is valid and to compute the storage size.

3.6 Step 1: Analysis

We present an unified parametric template to describe the cases of the analysis sub-goal. This step is not worked out in that much detail because it requires less research in terms of its refinement. We believe indeed that it does not contain any complex exploration or decision-making so it is relatively easy to provide near-optimal results, and it is also quite straightforward to make it scalable to large codes. Hence, the development of this step mostly involves applying the proper (existing) software engineering principles. Still, it is useful to identify the major sub-steps/cases which have to be considered, as shown below.

The unified parametric template provides the required information to translation and intra-signal in-place steps, which are executed after the analysis step, when the methodology is applied to an application instance. Since the analysis step is used for all cases and for next storage size optimization steps, i.e. inter-signal in-place optimization, it is more reusable and, thus, it is presented in this chapter. The Analysis step maps the applications of the target domain into the unified parametric structure of Alg. 3. The template consists of a for loop nested structure with manifest conditions. To map the application loops to the unified structure, the loops are virtually merged.

Since the index expressions are of ``iterator+constant" type, the nested loops are virtually merged by replacing the iterators with the unified iterators, e.g. I, and inserting manifest if statements to guarantee correct iteration bounds per statement. The bounds of the virtually merged loop are given by $LB_{\mathrm{I}} = min(LB_{\mathrm{i}}, ..., LB_{\mathrm{j}})$ and $UB_{\mathrm{I}} = max(UB_{\mathrm{i}}, ..., UB_{\mathrm{j}})$ and the manifest conditions can affect the Low Bound (I>LB), the Upper Bound (I<UB) or both bounds (I>LB && I<UB). Each access statement $FX(A(fx))$ is labeled as S^{m}_{ii}, where m gives the statement numbering and ii is the initial iterator, where the statement belonged before the virtual loop merge. The conditions are labeled as C^{n}_{ii}, where n gives the condition numbering and ii is the initial iterator, where the statement belonged before the virtual loop merge.

The unified template applies primitive operations, which are defined as OR (||) and AND (&&) operations and are used to combine primitive conditions into condition expressions. The remain-

ALGORITHM 3: Unified parametric structure of Analysis step

a_i^1=..., a_k^1=... ...
for *(I=*LB_l*; I<* UB_l*; I++)* **do**
...
  for *(K=*LB_K*; K<* UB_K*; K++)* **do**
    if *(I>* LB_i*) && (I<* UB_i*)* **then**
    ...
      if *(K>* LB_k*) && (K<* UB_k*)* **then**
        $S^1_{i,..,k}$:...=F_1(A[I+b_i^1]..[K+b_k^1])
        $S^2_{i,..,k}$:...=F_2(A[I+b_i^2]..[K+b_k^2])
        ...
        if *(*C^1_i*(I,*a_i^1*,*$type_i^1$*)..)* **then**
        ...
          if *(*C^1_k*(K,*a_k^1*,*$type_k^1$*)..)* **then**
          ...
            $S^3_{i,..,k}$:...=F_3(A[I+b_i^3]..[K+b_k^3])
    if *(I >* LB_j*) && (I<* UB_j*)* **then**
      if *(K >* LB_l*) && (K<* UB_l*)* **then**
        if *(*C^1_j*(I,*a_j^1*,*$type_j^1$*))* **then**
        ...
          if *(*C^1_l*(K,*a_l^1*,*$type_l^1$*)..)* **then**
          ...
            $S^4_{j,..,l}$:...=F_4(A[I+b_j^1]..[K+b_l^1])

ing logical operations that may exist in the applications can be expressed as combinations of the primitive operations. Two types of conditions can exist in applications: 1) the conditions which describe a Solid Iteration Space (SIS) and 2) the conditions which describe an Iteration Space with Holes (ISH). The SIS conditions change the bounds of the iteration space. The ISH conditions describe accesses or holes in the internal iteration space. The primitive conditions are either enumerative conditions, i.e. conditions which use constants, or parametric conditions, i.e. conditions which use linear expressions. The conditions in the application can be expressed through the following 4 classes of primitive conditions. So this covers all realistic cases in an application code belonging to our target domain.

The SIS conditions which use constants, e.g. i<3, are mapped to primitive SIS conditions, called Enumerative Conditions for Solid iteration space (ECS). The ECS use the $<$ comparison operator or the $>$ comparison operator or a combination of the $<$ and the $>$ with the AND logic operator to represent concurrent conditions for the low and upper bound of the iterator in a compact form. If the constant SIS condition in the application uses another comparison operator, it is mapped to the primitive condition operator. For instance, the condition i$\geq$LB is mapped to the condition i>LB' with LB'=LB-1, e.g. i$\geq$3 is mapped to i>2.

The ISH conditions which use constants, e.g. i==4, are mapped to primitive ISH conditions, called Enumerative Conditions for Iteration Space with Holes (ECH) , which are defined as conditions that describe accesses, thus they are expressed by the $==$ comparison operator or by a combination of an $>$ comparison operator with a $<$ comparison operator through an OR operator. Whenever the application constant ISH conditions are not primitive, they are mapped to ECH primitive conditions. For instance, the condition i$\neq$d is mapped to the condition (i<d)||(i>d).

The parametric expression for SIS is called Parametric Conditions for Solid iteration space (PCS) , e.g. i<2*k+1. They use the $<$ or $>$ comparison operator for the iteration space bounds.

The primitive parametric conditions for ISH are called Parametric Conditions for Iteration Space with Holes (PCH), which use the $==$ or $\neq$ comparison operator or a combination of $<$ and $>$ comparison operators using an AND logic operator, e.g. i>c*l+d_1 && i<c*k+d_2 (with d_1<c

and d_2<c). The parametric ISH conditions can be primitive PCH or a combination of PCHs.

When the analysis step is applied to an application of the target domain, the unified parametric template is instantiated and the required information for the next steps is extracted, i.e. the number and the ordering of the dimensions and the ordering of the condition and access statements and the parameters of each condition and access statement. This information is propagated to the translation and intra-signal in-place steps, which selects the active cases from the corresponding parameterized templates and applies the corresponding solution processes.

3.7 Conclusions

In this chapter, we have illustrated the results of applying the principles of the reusable DSE methodology to develop a scalable and near-optimal intra-signal in-place methodology for the applications of the target domain. We have applied the principles of Chapter 2 to define a set with all the possible index expressions and to select a representative index expression case for the intra-signal in-place methodology. The developed methodology consists of the sub-goal of computing the storage size (intra-signal in-place sub-goal), the translation of the access scheme information to a scalable and near-optimal representation for intra-signal in-place optimization and the analysis sub-goal which provides the information of the access scheme. The possible cases of the intra-signal sub-goal have been defined by applying top-down splits and based on the uni-directional arrows, they have been propagated, as constraints to the next sub-goals. The process has been repeated for the translation and analysis sub-goal, taking into account the propagated constraints each time. Per sub-goal a step is required with scalable and near-optimal parametric templates per case. During the methodology execution, the steps are applied following the opposite order in order to reach the final goal, as described in Chapter 5.

Chapter 4

Pattern representation

4.1 Introduction

The storage size management techniques take as input the access scheme, i.e. the global valid iteration space, which is defined by the application structure, e.g. the loops, the conditions statements and the memory access statements. For instance, commonly used applications in embedded systems, such as image, video and signal processing, have, as dominated data, arrays with very regular memory accesses inside application loops with conditions. The conditions disturb the regularity of the memory access statements making parts of the iteration space invalid, i.e. ``holes'' in the iteration space. When a high number of holes (due to the conditions) and several array access statements exist, the valid iteration space becomes highly complicated. Previous approaches describe the valid iteration space in an enumerative way, in a symbolic way or by worst case approximation, as explained in Section 3.3 of Chapter 3. The enumerative representations (e.g. [135]) are optimal, but not scalable, as the storage size exploration time is increased to unacceptable values. Further, the symbolic representations are scalable and near-optimal, but are applicable up to iteration spaces with holes of regular structures, i.e. piecewise regular (e.g. [55]). In irregular iteration spaces, they approximate the iteration regions. The approximation representations (e.g. [160]) use a worst case approximation by considering invalid parts as valid leading to overestimation of the storage size. Hence, a near-optimal and scalable representation is highly desired to be used in the translation step of the developed intra-signal in-place optimization methodology of Chapter 3 to enable efficient and scalable storage solutions in intra-signal-in-place step of the developed methodology.

In this chapter, we propose a representation of the access scheme, which is scalable and near-optimal for complex iteration spaces with irregular holes created by the application array access statements in a loop structure with several manifest conditions. Section 4.2 motivates this work through the motivational example used in Chapter 3. To achieve a near-optimal and scalable storage size management we require a representation, which avoids the enumeration of the different valid iteration space parts and can describe a high number of parts. Parts are pruned by a finite and, usually, small set of condition statements in the application. One enumerative condition describes

one pruned/accessed part, whereas one parametric condition describes several pruned/accessed parts in a regular way. When several conditions exist, the iteration space is the combination of the individual iteration spaces of each condition and access statement. The result is several invalid parts irregularly spread in the iteration space, increasing the complexity of shape. We introduce the concept of the patterns to achieve a scalable and near-optimal representation of the iteration space and the storage management, which is described in detail in Section 4.3. A pattern represents the valid iteration space of a condition or an access statement through a compact and repetitive description in the iteration space, thus avoiding enumeration. By including the holes, the iteration space can be described in a regular and repetitive way due to the loop structure. The percentage of near-optimalities introduced in the pattern representation is controlled by the size of the holes, which create address irregularity and are not expressed in the pattern, i.e. holes of size one. In this way, the iteration space shape due to a parametric condition can be scalable and efficiently represented. The pattern is defined as a sequence of two parameters: 1) the number of consecutive iteration values where the statement has the same behavior and 2) the statement behavior, i.e. Access (A) or Hole (H). In this way, the invalid iteration space parts are described avoiding suboptimal approximations. E.g. {1H 1A} repeated 5 times represents the behavior of a condition statement, which accesses only in the odd iterators from 0 to 10.

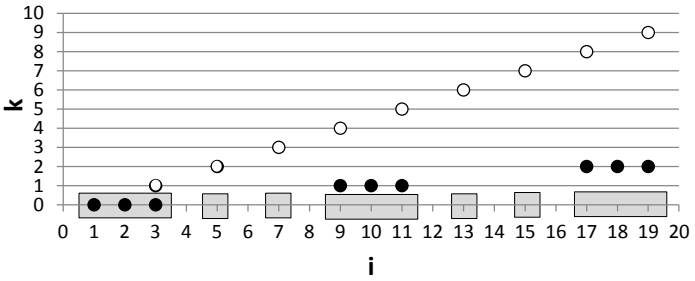
The application under study has different conditions, loop structure and access statements, which lead to different patterns and pattern combinations. The iteration space part of each condition is represented by a pattern. To describe the valid iteration space per access statement, the relevant condition patterns and the access pattern have to be consistently combined in a scalable and near-optimal way. To achieve the consistent combination of patterns, we define a set of pattern operations, which are described in Section 4.4. Based on the application structure, the patterns can be combined under different cases. Hence, we identify a complete set of the different pattern combination cases that may occur in the unified application structure applying the reusable DSE methodology of Chapter 2. Then, we provide scalable and near-optimal operations per combination case to systematically combine the corresponding patterns. Section 4.5 describes the pattern operations per case verifying the scalability and near-optimality per operation. In realistic contexts, the process of combining patterns is applied in a finite small set of patterns, which is defined by the number of conditions and access statements. We demonstrate how the patterns are composed and how operations are applied to compose a final read pattern, from where the final storage size is computed, through a demonstration case study. The final storage size is computed by combining the final patterns of the read access statements through OR pattern operations and adding the size of the parts with access behavior.

4.2 Motivation

As described in motivational example of Chapter 3, which is repeated in Fig. 4.1, the enumerative approaches require too much exploration time, when the number of accesses is increased, and

```
For (i=0; i<N; i++)
 For (k=0; k<M; k++)
  If ((i>8k)&&(i<8k+4)||(i≠2k))
   A[i]=...
  EndIf
 EndFor
EndFor
```

(a)



(b)

Figure 4.1: Motivational example: (a) Application code with three conditions and a write access statement and (b) Iteration space with the accesses to array A.

the symbolic approaches are inappropriate for iteration spaces with increased number of irregular holes. Then, approximations would have to be applied, which insert suboptimalities. Instead, the proposed methodology represents the irregularities in a near-optimal and scalable way through patterns. The first and the second condition are expressed through pattern {1H 3A 4H} repeated N/8. The third condition is expressed by pattern {1H 1A} repeated N/2 times. By applying operations on the patterns, a global pattern is computed, which efficiently represent the valid iteration space, i.e. {1H 3A 1H 1A 1H 1A} repeated N/8 times. With the pattern representation the minimum storage size can be computed. As the example belong to non-overlapping store and load statements case, the minimum storage size is given by the summation of the array elements that are accessed in the iteration space. The storage size is given by the A in the final pattern multiplied by the repetition factor, (N/8)*5. The gray boxes over I axis in Fig. 4.1(b) show the array elements which are accessed. In realistic applications, the loop iterations are highly increased, several complicated condition expressions and array access statements coexist, which create a complex iteration space with irregular holes. In this chapter, we present a near-optimal and scalable representation to handle this complex irregular iteration spaces.

4.3 General Pattern Formulation

This section defines the pattern formulation of the proposed representation and describes how primitive conditions and access statement are translated into patterns.

The pattern represents in a scalable and near-optimal way the iteration space described by an access/condition statement and the loop structure by using a compact and repetitive description of the statement behavior in the iteration space. A statement based on the value of the iterator either accesses or not the array. When the statement has the same behavior (A or H) in consecutive iterator values, the iterator values are grouped in iteration space parts. An iteration space part is described by the length and the behavior. The length is the number of subsequent values of the iterator where the statement has the same behavior (Part Iterator Range - PIR). The behavior is A for access or H for hole in the PIR (Part Type - PT). When the iterator values are traversed following the loop structure, a new iteration space part is created when the behavior of the access

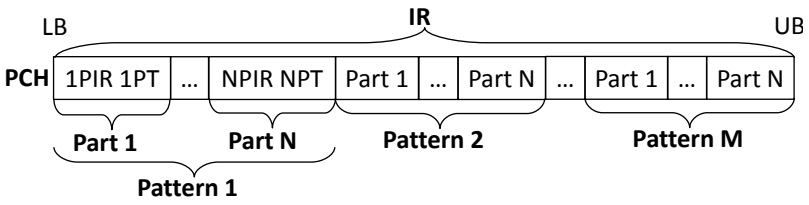


Figure 4.2: The pattern consists of N parts, each part has a PIR and a PT, the pattern is valid from LB up to UB and it is repeated M times.

statement is modified. The loop structure describes the repetition of the statement execution and thus, the repetition of the pattern in the iteration space. Hence, one pattern with few parameters is sufficient to describe the behavior of the statement avoiding enumeration. Fig. 4.2 shows a pattern which consists of N different parts and each part has a PIR and a PT. The pattern is repeated M times starting from the lower loop bound (LB) and terminating in the upper loop bound (UB). Based on the type of conditions the LB and UB are computed by Eq. 4.1 and Eq. 4.2. The Iterator Range (IR), i.e. the range of the iterator values where the pattern is valid, is computed by Eq. 4.3. The Pattern Size (PS) is the summation of the lengths of all parts in the pattern (Eq. 4.4) and the Repetition factor (R) describes the times the pattern is repeated in the IR (Eq. 4.5). In the next paragraphs, we describe the behavior of an access statement and different primitive conditions using the pattern formulation. The PIRs of the pattern with PT equal to A are access segments and the summation of their length is the Segment Iterator Domain (SID) defined by Eq. 4.6. The summation of the length of the parts with type equal to hole is defined by Hole Iterator Domain (HID) (Eq. 4.7).

$$\begin{aligned} &\textbf{SIS\&ECH: (I}>a\textbf{)}: LB = max(LB_{\mathrm{I}}, a) \\ &\textbf{PCH: (I}>\text{c*K+b}\textbf{)}: LB = max(LB_{\mathrm{I}}, c * (LB_{\mathrm{K}} + 1) + b - 1) \end{aligned} \tag{4.1}$$

$$\begin{aligned} &\textbf{SIS \& ECH: (I}<b\textbf{)}: UB = min(UB_{\mathrm{I}}, b) \\ &\textbf{PCH: (I}<\text{c*K+b}\textbf{)}: UB = min(UB_{\mathrm{I}}, c * (UB_{\mathrm{K}} - 1) + c) \end{aligned} \tag{4.2}$$

$$IR = UB - LB - 1 \tag{4.3}$$

$$PS = \sum_{i=1}^{Parts} iPIR \tag{4.4}$$

$$R = \frac{IR}{PS} \tag{4.5}$$

$$SID = \sum_{i=0}^{PIRs} PIR(RD)_{(\mathrm{PT==A})} \tag{4.6}$$

$$HID = \sum_{i=0}^{PIRs} PIR(RD)_{(\mathrm{PT==H})} \tag{4.7}$$

The simplest case is when no conditions exist in the code and the pattern is one part, as the code in Fig. 4.3(a). In this example, the LB is 2 (due to consistency reasons with the primitive conditions, where the LB is defined by $>$ comparison operator), the UB=10 and the IR=7. In the first iteration, the iterator has the value of 3 and the statement accesses the array. The A behavior

```
For (I=3; I<10; I++)
  ...=A[I-3]
EndFor
```

(a)

```
For (I=3; I<10; I++)
  If (I>6)
    ...=A[I-3]
  EndIf
EndFor
```

(b)

```
For (I=3; I<10; I++)
  If (I<4) || (I>6)
    ...=A[I-3]
  EndIf
EndFor
```

(c)

```
For (I=3; I<10; I++)
  For (K=0; K<4; K++)
    If (I==2*K+1)
      ...=A[I-3]
    EndIf
  EndFor
EndFor
```

(d)

Figure 4.3: Application code examples with one access statement for the array: (a) Without conditions, (b) With one ECS, (c) With one ECH and (d) With one PCH

is valid up to i=9, where the loop is terminated. The pattern is {7A}, i.e. the length is 7 and the type of behavior is A and the repetition factor is 1.

An ECS condition modifies the access pattern, as depicted in the example of Fig. 4.3(b). The i>a type is used, where the LB is derived from the maximum value between the condition expression and the LB of the loop structure, i.e. LB = max(LB,a) = max(2,6) = 6. If an ECS condition of i<b exists, the UB of the pattern is the minimum value between the condition and the UB of the loop structure, i.e. UB=min(UB,b). The IR=UB-LB-1=3, the PS=IR, the R=1, the PT=A, the PIR=IR and the pattern is ECS=PIR PT={3A}.

An ECH condition disturbs the regularity of the A behavior. In Fig. 4.3(c) an ECH of type (i>a)||(i<b) is used as example. The array A is not accessed in iterator values 4, 5 and 6. The LB=2, UB=4, IR=1 and the pattern is {1A}. For the second condition LB=6, UB=10, IR=3 and the pattern is {3A}. By combining both patterns the result is {1A 3H 3A} with LB=2, UB=10, IR=7 and R=1.

The regularity of A accesses is also disturbed by PCH condition, e.g. i==c*K+b type in Fig. 4.3(d). The LB of the pattern is the maximum value between the value of the condition expression in the LB of K iterator and the LB of the loop structure. The LB=max(LB_{I},(c*(LB_{K}+1)+b-1)=max(2,0)=2. The UB of the pattern is the minimum value between the condition expression in the UB of K iterator extended by the PS of PCH, i.e. c, and the LB of the loop structure, UB=min(UB_{I},(c*(UB_{K}-1)+c)=min(9,9)=9. The IR=6, PS=2, R=3, the first part is 1PIR=1 and 1PT=A, the second part is 2PIR=PS-1PIR and 2PT=H. The pattern is {1A 1H}. If the UB of iterator k is increased to 5, the pattern is {1A 1H} with R=5, since the UB=10.

In realistic applications, several control statements with more complicated conditions and array access statements are combined in different ways creating complex iteration spaces with irregular holes which require an efficient way to represent them. We discuss how our methodology handles this issue in the next paragraphs.

4.4 Pattern Combination Cases

In order to handle all the possible combination cases of the patterns that may exist, we define a complete set with the different cases, which derives by applying the reusable DSE methodology to the goal of identifying the possible pattern combination cases. Fig. 4.4 depicted the different possible pattern combination cases (white boxes) and the corresponding pattern operations per case (gray boxes). In the following paragraphs, we describe the different combination cases (text in bold) and link them to the corresponding pattern operations (text in italic). The developed pattern operations are described in Section 4.5.

The first split in the combination cases is the overlapping or non-overlapping patterns. The **overlapping patterns** have parts that are referring to the same iterator values and potentially describe different behavior for the statement. For instance, the patterns in Fig. 4.3(a) ({7A} from i>2 till i<10) and in Fig. 4.3(d) ({1A 1H} for i>3 till i<9) are overlapping. The **non-overlapping patterns** describe the statement behavior for different iterator values. The non-overlapping patterns may be **sequential**, i.e. one pattern describes up to the iterator value x and the other pattern describes from $x + 1$ and on. The *sequential non-overlapping* operation is described in Section 4.5.1.1. The **non-sequential** patterns have iterator values between them which are not described by either pattern, i.e. one pattern describes up to the value x and the other pattern from $x + y$. The *non-sequential non-overlapping* operation is described in par. 4.5.1.2. The overlapping patterns are further divided into the cases where they have same or different PS. In case of **same PS**, the patterns may be **fully aligned** or **not-aligned**. When the patterns are fully aligned, the LB and UB of the patterns are equal and the primitive operations can be safely applied. The primitive operation can be an **OR operation**, which is explained in Section 4.5.2.1, or an **AND operation**, which is described in Section 4.5.3. The resulting pattern may require post-processing due to the existence of **sequential parts of the same behavior** or **repetition of a smaller pattern** inside the resulting pattern. The sequential parts of the same behavior in the internal of the resulting pattern are identified and merged into one part during the primitive operation. The merging of the first part with the last part due to same behavior is explored through the *skewing operation*, which is described in detail in Section 4.5.4. For instance, the PCH={3A 4H 3A} is modified to PCH'={4H 6A} and two ECS conditions. The repetition of a smaller pattern is explored through the *repetition search* operation, described in Section 4.5.4.1. If PCH={3A 4H 3A 4H} with a R=10, the PCH is modified to PCH'={3A 4H} with a R=20. When the patterns have the same PS and are **not-aligned**, **LB misalignment** or/and **UB misalignment** may exist. The alignment operations are applied by first applying the *LB alignment* operation described in Section 4.5.4.2.1, and then the *UB alignment* operation, described in Section 4.5.4.2.2.

When the patterns have **different PS**, operations for modifying the pattern size are applied. The Least Common Multiple (LCM) of the PS of the patterns is computed, e.g. when PS1=3 and PS2=4, LCM is 12. The LCM is marked as acceptable when the modified PS after unrolling based on the LCM is quite low and thus not close to enumeration of the iteration space. If the **LCM is acceptable**, a *PS modification* operation, which is described in Section 4.5.4.3, is applied, which

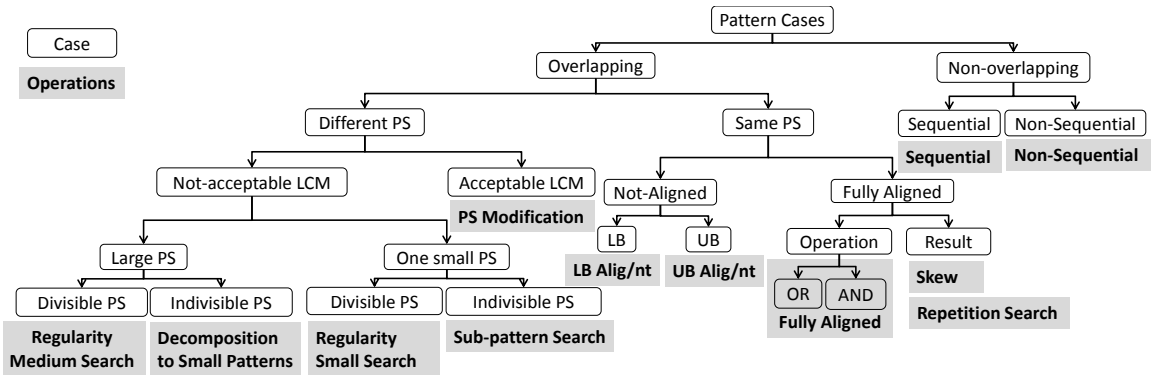


Figure 4.4: Set of possible pattern combination cases with the corresponding operations.

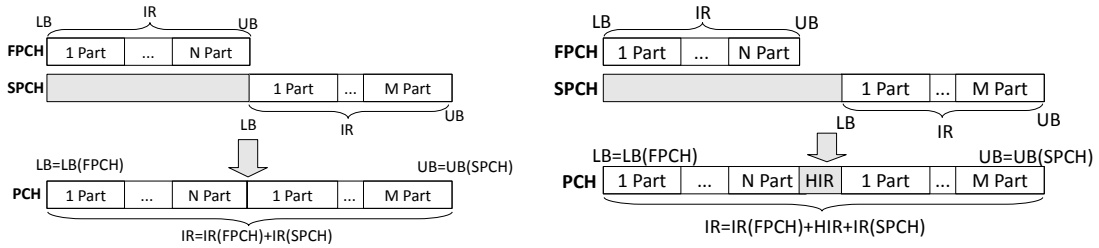
partially unrolls the patterns to have a PS equal to the LCM. If the **LCM is not acceptable**, the OR and AND operations should be performed in a way that avoids enumeration of the iteration space. We propose search operations based on the relative PS of the patterns. If one pattern has a **relative small PS** comparing with the other pattern, two cases exist: the pattern with the small PS that is a factor of the PS of the other pattern (**Divisible PS**) or the PS of the patterns are **Indivisible**. In the first case, *regularity small search* operations of Section 4.5.4.6 are applied to search for sub-pattern repetition. In the latter case, *sub-pattern search* operations of Section 4.5.4.4 apply an iterative partial OR between the small pattern and a part of the other pattern and searches for repetition in the result. When both patterns have **large PS**, the PS of one pattern (medium pattern) may be a factor of the PS of the other pattern (**Divisible PS**) or the PS of the patterns are **Indivisible**. In the first case, *regularity medium search* operations of Section 4.5.4.7 are applied to search for pattern repetition. In the latter case, the *decomposition operation to small patterns* of Section 4.5.4.4 is introduced to split the medium pattern to smaller patterns. Then, the operations for the case of one small PS are applied.

4.5 Pattern Operations

This section describes the pattern operations using two patterns, First PCH (FPCH) and Second PCH (SPCH), and illustrates that scalability and near-optimality are maintained per operation. The following sections describe the pattern operation required per pattern combination case presented in Section 4.4.

4.5.1 Non-overlapping Operations

The non-overlapping operations join two patterns, which refer to different iterator values, into one pattern. The length and the behavior of the pattern parts is not modified and thus near-optimality is maintained. The operations are applied in a scalable way as the patterns are not required to be unrolled. In non-overlapping patterns only the OR primitive operation is valid, which is a pattern concatenation. The AND operation over different iterator values results to the null pattern.



(a) Sequential OR Operation

(b) Non-sequential OR Operation

Figure 4.5: Result of non-overlapping operations.

4.5.1.1 Sequential Operation

In the general case, the UB of the first pattern is equal to the LB of the second pattern and the OR operation results to a concatenated pattern. The operation is schematically depicted in Fig. 4.5(a). The FPCH and the SPCH are not modified and are combined through concatenation into a common pattern PCH. In case the patterns to be combined are the same, the result is a pattern with the same part and modified LB, UB and a repetition factor increased to $R = \frac{IR(PCH)}{PS}$.

4.5.1.2 Non-Sequential Operation

The non-sequential OR operation is similar to the general case of the sequential operation but with a new part of length HIR=LB(SPCH)-UB(FPCH) of PT=H is inserted between the FPCH and the SPCH. The OR operation result is depicted in Fig. 4.5(b), where during concatenation of FPCH and SPCH, a third pattern, which describes holes, is inserted.

4.5.2 Fully aligned Operations

They are applied to patterns with the same PS, aligned LB and UB. Due to the address generation overhead, the irregular holes of size one are considered as virtual accesses inserting near-optimality in the result. The near-optimality is fully controllable, since the methodology is aware of where it is introduced and it can explicitly compute it. The operations are applied pattern part by pattern part without unrolling. Since the patterns consist of small number of parts (due to pattern nature and post-processing operations) scalability is maintained.

4.5.2.1 OR Operation (||)

The LB, the UB and the PS of the result pattern (PCH) are the same with that of the initial patterns FPCH and SPCH. The OR operation defines the PCH pattern based on the type and the position of the FPCH and SPCH parts. The operation is iteratively applied part by part until the result pattern reaches the PS. In each iteration, an OR operation is applied between a FPCH and a SPCH part. The size of the parts is potentially modified depending on the size of the part with the dominant type, i.e. A, as explained in detail in the next paragraph.

ALGORITHM 4: Fully aligned OR Operation.

```
UB(PCH)=UB(FPCH)
LB(PCH)=LB(FPCH)
PS(PCH)=PS(FPCH)
MPS=0
while (MPS≠PS(PCH)) do
    if (PIR(FPCH)==PIR(SPCH)) then
        PIR(PCH)=PIR(FPCH)
        if (PT(FPCH)==A||(PT(SPCH)==A) then
            PT(PCH)=A
        else
            PT(PCH)=H
    else
        if (PIR(FPCH)>PIR(SPCH)) then
            max=FPCH
            min=SPCH
        else
            max=SPCH
            min=FPCH
        if (PT(max)==A then
            PT(PCH)=A
            PIR(PCH)=PIR(max)
            next PIR(min)=next PIR(min)-(PIR(max)-PIR(min))
        else
            PIR(PCH)=PIR(min)
            next PIR(max)=next PIR(max)-(PIR(max)+PIR(min))
            if (PT(min)==A&&PT(max)==H) then
                PT(PCH)=A
            else
                PT(PCH)=H
    PCH=MPC|PIR PT
    MPS=MPS+PIR
    Next PIR(FPCH), PIR(SPCH)
```

The pseudocode of the operation is described in Alg. 4. In each iteration, the PIR and the PT of one result part are defined based on the PIR and the PT of the FPCH and SPCH parts that are merged in the current iteration. In case the PIR of FPCH and SPCH parts are equal, the PIR of the PCH part is equal to PIR(FPCH). If the PT of both FPCH and SPCH parts is H, the PT of the PCH part is also H. Otherwise, it is A. When the FPCH and SPCH PIRs are different, the PIR of the result depends on the PT. If both parts are of H type, the length is defined by the minimum PIR and the PT is H. The part with the maximum PIR is modified by split into two parts: one part with PIR equal to the minimum PIR of FPCH and SPCH used in the current OR iteration and another part with PIR equal to max(PIR(FPCH),PIR(SPCH))-min(PIR(FPCH),PIR(SPCH)) used in the next iteration. This modification is depicted in Fig. 4.6(b), where the SPCH has larger PIR and is split into a part equal to the PIR of FPCH and a gray part that is left for the next iteration. If at least one part is of A type, the PT of the result is A. The PIR of the result depends on which part has PT=A. If the maximum PIR has A type, the PIR of the result part is equal to the max(PIR). Then, the pattern with the minimum PIR is modified: the PIR of the next part is reduced by max(PIR(FPCH),PIR(SPCH))-min(PIR(FPCH),PIR(SPCH)), as depicted in Fig. 4.6(c). The next part of the FPCH is reduced by the gray part. If only the part with the minimum PIR is of A type, the PIR of the result part is the min(PIR(FPCH),PIR(SPCH)). The PIR of

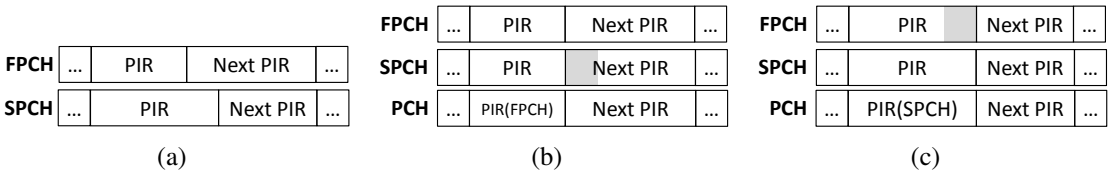


Figure 4.6: Fully aligned OR operation: (a)Two parts of the patterns FPCH and SPCH, (b) When PT of both parts is H or the part with the smaller PIR has PT=A, the PIR of the PCH part is equal to the small PIR and the PIR of the next part of the pattern with the larger PIR is increased by the PIR difference, i.e. large(PIR)-small(PIR) and (c) When PT of both parts is A or the part with the larger PIR has PT=A, the PIR of the PCH part is equal to the larger PIR and the PIR of the next part of the pattern with the small PIR is reduced by the PIR difference.

the next part of the pattern with the maximum PIR is increased by max(PIR(FPCH),PIR(SPCH))-min(PIR(FPCH),PIR(SPCH)). The PT is checked for equality with the PT of the previous PCH part in order to be stored in one part and reduce the total number of pattern parts. The process is repeated for the next FPCH and SPCH parts.

4.5.3 AND Operation (&&)

The LB, the UB and the PS of the result pattern (PCH) remain the same with that of the initial FPCH and SPCH. The way the AND operation is applied is similar to the OR operation, but the modification of the parts is adapted to the AND operation.

In more details, when the AND operation is applied into the FPCH and SPCH parts and these parts have equal PIR, the PIR of the resulting pattern, i.e. PCH, is equal to the parts. If the PT of both parts is A, the PT of the result is A. If at least one of the FPCH or SPCH parts has PT of H type, the PT of the result is H. If the PIR of the FPCH part is different from the PIR of the SPCH part, the PIR of the result depends on the PT. If both parts are of A type, the PIR of the result is defined by the minimum PIR and the PT is A. The part with the maximum PIR is modified by splitting into: 1) one part with PIR equal to the minimum PIR and one part with PIR equal to the difference of the FPCH and SPCH parts, i.e. max(PIR(FPCH),PIR(SPCH))-min(PIR(FPCH),PIR(SPCH)). When at least one part of the FPCH and SPCH parts has PT of H type, the PT of the result is H. The PIR of the result depends on which of the two initial parts has PT equal to H. If the maximum PIR is H, the PIR of the result is max(PIR). Then, the PIR of the next part of the pattern with the minimum PIR part is reduced by the difference of the two parts. If only the minimum PIR is of H type, the PIR of the result is equal to the minimum. The PIR of the next part of the pattern with the maximum PIR is increased by the difference of the two parts. If the PT of the PCH part is equal with the PT of the previous PCH part, the parts are stored as one part. The process is repeated for the next couple of FPCH and SPCH parts.

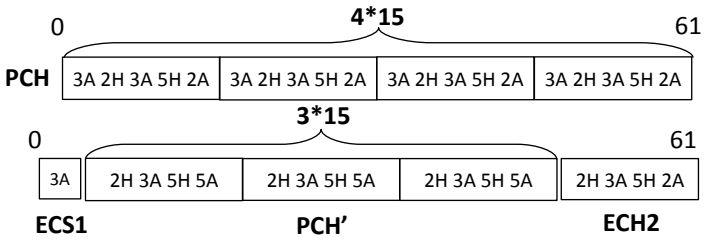


Figure 4.7: Example of applying Skew operation. The initial PCH pattern is {3A 2H 3A 5H 2A} and after applying the skew operation the ECS1, ECH2 and PCH' are created.

4.5.4 Skew Operation

The skew operation merges the first and the last part, when they have the same type. In this way, the number of pattern parts is reduced, the length of the parts is increased and the corresponding iteration space is described in a more compact way. The PT of the parts are not modified and thus near-optimality is maintained. The operation is applied on the pattern without requiring to enumerate the behavior in the overall iteration space maintaining scalability. The skew operation splits one repetition of the pattern to allow the skewing of the first part into the last part. The skew operation modifies the PIR of the first and last part, the LB, UB, R and IR. The initial PCH is split into an ECS (e.g. ECS1 in Fig. 4.7), which describes the first part of the first repetition of the PCH, a new PCH (e.g. PCH' in Fig. 4.7), with the first and the last part merged, and an ECH (e.g. ECH2 in Fig. 4.7), which describes the remaining parts of the first repetition of the initial PCH. The different patterns are given by Eq. 4.8, 4.9 and 4.10.

$$\textbf{ECS1=\{1PIR 1PT\} :}\ LB = LB(PCH), UB = LB + 1PIR + 1 \tag{4.8}$$

$$\textbf{PCH'=\{2PIR 2PT ... (NPIR+1PIR) NPT\} :}$$

$$LB = UB(ECS1) - 1, UB = UB(PCH) - \sum_{i=1}^{PCHParts} iPIR, R = R(PCH) - 1 \tag{4.9}$$

$$\textbf{ECS2=\{2PIR 2PT ... NPIR NPT\} :}\ LB = UB(PCH') - 1, UB = UB(PCH). \tag{4.10}$$

For instance, in Fig. 4.7 the PCH is {3A 2H 3A 5H 2A} with LB=0, PS=15, R=4 and UB=61. After skew operation: the ECS1=3A with LB=0, PS=3, UB=4, the PCH' is {2H 3A 5H 5A} has LB=3, PS=15, R=3 and UB=49, and the ECS2=2H 3A 5H 2A with LB=48 and UB=61.

4.5.4.1 Repetition Search Operation

The repetition search operation searches inside the PCH result to identify potential repetition of smaller patterns to reduce the number of pattern parts. The operation is applied in groups of parts and it is repeated for a small number of times, i.e. half the number of parts in the pattern, avoiding enumeration of the statement behavior in the iteration space. The potential new internal pattern is initialized with the first two parts and the remaining parts are compared in groups of two to verify repetition. The search continues by increasing the new internal pattern by one part and comparing the remaining pattern in groups of three etc. The operation terminates when the internal pattern

ALGORITHM 5: LB Alignment Operation

```
Size2=0, R=0, IR=(bound+1)-LB(PCH)-1, flag=0
if (IR/PS(PCH)≠0) then
    PCH1=PCH
    LB(PCH1)=LB(PCH)
    R(PCH1)=int(IR/PS)
    UB(PCH1)=R(PCH1)*PS
if ((bound%PS) ≠0) then
    Size1=UB(PCH1)
    ECS1, PIR(PCH), Diff=Find Pattern(Size1, PCH,
    bound+1)
    LB(ECS1)=max(UB(PCH1)-1,LB(PCH))
    UB(ECS1)=bound+1
    Size2=PIR(PCH)-Diff
LB(PCH2)=bound
if (Size2≠0) then
    R(PCH2)=R(PCH)-R(PCH1)-1
    PCH2=Size2 PT
    ECH2=Size2 PT
    Next PIR(PCH)
    while (Size2+PIR(PCH)≤PS) do
        PCH2=PCH2|PIR PT
        Size2=Size2+PIR(PCH)
        if
        (flag==0&&((bound%PS)+Size2+PIR(PCH))≤PS)
        then
            ECS2=ECS2|PIR PT
        else
            flag=1
            Diff=PS-Size2
            ECS2=ECS2|Diff PT
        Next PIR(PCH)
    Diff=PS-Size2
    PCH2=PCH2|Diff PT
if (Size2==0) then
    UB(PCH2)=UB(PCH)
    R(PCH2)=R(PCH)-R(PCH1)
Find Pattern(Size, PCH, bound){
while (Size+PIR(PCH)<bound) do
    Size=Size+PIR(PCH)
    PCH'=PCH'| PIR(PCH) PT(PCH)
    Next PIR(PCH)
Diff=(bound)-Size
return PCH'|Diff PT(PCH), PIR(PCH), Diff
}
```

consists of half the parts of the initial pattern. The operation does not modify the parts of the pattern maintaining near-optimality.

4.5.4.2 Alignment Operations

The alignment operations are applied in patterns with not-aligned bounds and split the pattern into: one pattern with aligned bound and one non-overlapping sequential pattern. The alignment operation depends on where the new bound cuts the initial pattern. The operation is applied in the pattern parts avoiding enumeration. The PT of the parts is not modified, i.e. the number of the holes is identical to the number before applying the operations, maintaining near-optimality.

4.5.4.2.1 LB alignment The LB alignment operation, depending on where the new bound cuts the initial pattern, may split into maximum four sub-patterns. The PCH represents the initial pattern to be aligned and the bound represents the new LB in the pseudocode of Alg. 5. The most general case is when the bound cuts the PCH somewhere in the middle of the pattern and in the middle of the repetitions. Then, two patterns with R>1, i.e. PCH1 and PCH2 in Fig 4.8(a), and two patterns with R=1, i.e. ECH1 and ECH2 in Fig 4.8(a), are created. The operation explores the initial pattern part by part to define the new patterns. The repetition factor of the initial PCH is R1+R2+1. The

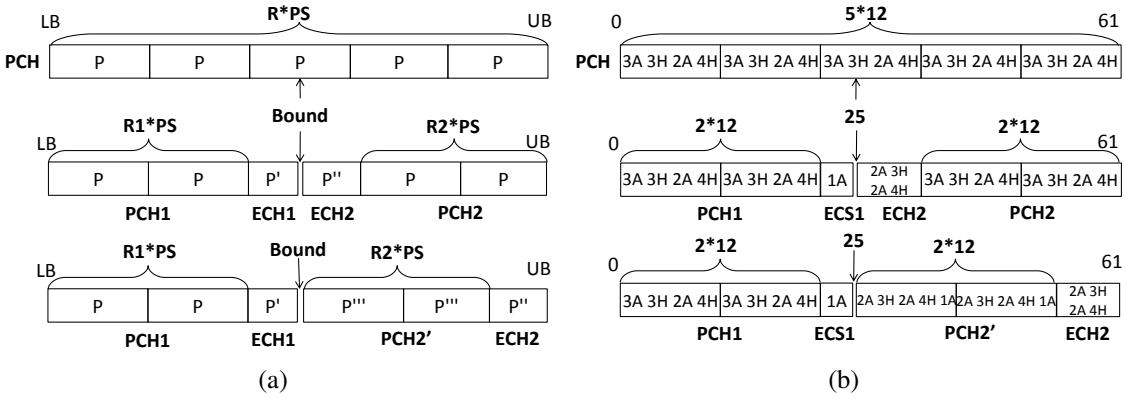


Figure 4.8: LB Alignment operation: (a) General Case: The PCH is the initial pattern, the Bound is the new LB and defines the position to split the PCH. The result is a new PCH1, ECS1, ECS2 and PCH2. The right section is skewed to align the PCH2 to the new LB and (b) Example: The initial pattern is {3A 3H 2A 4H} and the Bound is 25.

first section of the PCH, i.e. the section on the left side of the bound, has IR=(bound+1)-LB(PCH)-1. If the left section is larger than two times the PS, the repetition occurs and a PCH1 is created with LB equal to the LB of the initial pattern, R equal to (int)$\frac{IR}{PS}$ and UB=LB+R*PS+1. The repetition, where the new bound cuts the initial pattern, creates the ECH1 and ECH2 patterns. The LB of ECH1 is the maximum value of the UB(PCH1)-1 and the LB of the PCH. The UB is equal to bound+1. To define the parts of ECH1 and ECH2, the initial pattern (P) is traversed part by part to identify in which part the bound belongs to ($Part_{Bound}$). The parts of the ECH1 are the parts of P before the bound, including the corresponding part, i.e. up to the bound, of $Part_{Bound}$). The remaining $Part_{Bound}$ and the parts after the bound compose the ECH2 pattern. This pattern split does not yet lead to LB alignment with the PCH2, since the ECH2 has to be moved after the PCH2. This is achieved by skewing the PCH2. The PS is maintained, whereas the pattern and the bounds are modified. The skewed PCH2 pattern, i.e. PCH2' in Fig. 4.8(a), is {ECH2 ECS1}, the LB is equal to the bound and the UB is given by R2*PS+bound+1. The ECH2 has LB equal to R2*PS+bound and UB equal to the UB of the initial PCH. A degenerate case is when the bound is a factor of the PS of the initial pattern. Then, the PCH is split into two PCH1 and PCH2 patterns with parts that are the same with the parts of the initial PCH. The PCH1 has LB equal to the LB of the PCH, UB equal to bound+1, repetition factor equal to $\frac{IR(PCH1)}{PS}$. The PCH2 has LB equal to bound, UB equal to the UB of the initial PCH and $R = \frac{IR(PCH2)}{PS}$.

An example (Fig. 4.8(b)) is the pattern {3A 3H 2A 4H} with LB=0, UB=61, R=5, PS=12 and the bound at 25. The LB alignment operation splits the pattern to {1A} | {2A 3H 2A 4H}. The PCH1 is {3A 3H 2A 4H} with LB=1, UB=2*12+1=25, R=2, IR=24, PS=12. The ECS1={1A} with LB=24, UB=26, R=1, PS=1. The ECS2 is the remaining pattern of PCH, i.e. {2A 3H 2A 4H}. To align at the LB, the pattern of the PCH2 is skewed resulting to PCH2'={2A 3H 2A 4H 1A}, as depicted in Fig. 4.8(b). The PCH2' has LB=25, UB=50, R=2, PS=12 and ECS2 has LB=49, UB=61, PS=11 and R=1.

4.5.4.2.2 UB alignment The UB alignment operation is applied after LB alignment operation, thus the LB of the patterns is already aligned. Since the PS of both patterns is the same, the position of the new aligned UB, i.e. the smaller UB of the patterns, is always a factor of the PS. Hence, the pattern is not modified. The initial PCH pattern is split into a left pattern and the right pattern due to the new bound. The LB of the left pattern is the LB of the PCH, the UB is the new UB, the R is defined by $\frac{IR(PCH1)}{PS}$. The remaining pattern has LB equal to the bound-1, UB equal to the UB(PCH), the R is defined by $\frac{IR(PCH2)}{PS}$. Both new patterns can be a PCH1 (R>1) or an ECS1 (R=1). The PT of the parts is not modified and near-optimality is not affected.

4.5.4.3 PS modification Operation

When the LCM of the PS is acceptable, i.e. has a low value avoiding enumeration, the patterns are modified to have PS equal to the LCM. This is achieved by unrolling the patterns by the required Unrolling Factor (UF), which is quite low because of the acceptable LCM. The parts of the patterns are not modified and thus near-optimality is not affected. The UF for a pattern PCH derives from Eq. 4.11. The LB and the UB remain the same, the R is updated using Eq. 4.12. For instance, the FPCH is {2A 2H} with PS=4, R=30 and SPCH is {2A 1H} with PS=3 and R=20. The LCM is 12. The modified patterns are: FPCH={2A 2H 2A 2H 2A 2H} with PS=12, R=10 and SPCH={2A 1H 2A 1H 2A 1H 2A 1H} with PS=12, R=10.

$$UF(PCH) = \frac{LCM}{PS(PCH)} \tag{4.11}$$

$$R(PCH) = \frac{R(PCH)}{UF(PCH)} \tag{4.12}$$

4.5.4.4 Sub-pattern Search Operation

The operation is applied between a pattern with small PS and a pattern with large PS when the LCM is not acceptable. If a primitive operation is fully applied to the small and large pattern, it requires a high UF which may lead to results close to enumeration. To maintain the scalability of our methodology, the sub-pattern search operation is proposed to apply the operation in sections. The operation searches for the sub-patterns, which are repetitive in the complete result of the operation but without computing through enumeration the operation result, which has unacceptable size. The operation is applied in iterations and each iteration explores a sub-pattern. The sub-patterns are described by the parts where the result is not upfront defined. For instance, for the OR operation in patterns {2A 3H} and {3A 4H 2A 4H}, the result may not be A only in the positions where both parts have PT=H. Hence, it is sufficient to search for sub-patterns with parts of H type. A similar reasoning hold for the AND operation, where the differentiation derives from the parts of A type. We present the operation for the OR case.

The operation is applied in iterations. In each iteration the small pattern and a section of the large pattern with size equal to the small PS are explored. In this way enumeration is avoided, since the exploration of the sub-patterns is performed in an finite set of positions and the operation is applied for a small PS. In Alg. 6 the process is described in detail. Variable x defines the

ALGORITHM 6: Sub-pattern search operation

```
sub-pattern_exist=0
for (x=0; x<LCM; x=x+PS(min)) do
    section=Find Section(max, x, PS(min))
    sub-pattern=Merging(section, min)
    if (Find zeros(sub-pattern)!=0 then
        if (sub-pattern_exist==0 then
            subpatterns[k]=sub-pattern
            pos_cnt=0
            positions[sub-pattern][pos_cnt]=x
            pos_cnt++
            R=1
            sub-pattern_exist=1
        else
            for (j=0; j<i; j++) do
                if (sub-pattern==subpattern[j] then
                    positions[j][pos_cnt]=x
                    R=R+1
                    k++
for (j=0; j<k; j++) do
    irregular=Find Regularity(sub-pattern[j])
    if (irregular==0) then
        PCH=sub-pattern[j]
        LB=position[j][0]
        UB=R*PS(min)
    else
        Split-pattern=sub-pattern[j]
        while (PS(Split-pattern))>1) do
            Split-pattern=Split
            sub-patterns(Split-pattern)

Find Section(max, x, PS(min)){
crt_size=0
section=Find Pattern(crt_size, max, x)
Next PIR
crt_size=Diff
return Find Pattern(crt_size, max, PS(min))
}
Find Regularity(sub-pattern){
period=position[sub-pattern][1]-position[sub-pattern][0]
for (m=1; m<positions(sub-pattern); m++) do
    period(m)=position[sub-pattern][m+1]-
    position[sub-pattern][m]
    if (period(m)!=period) then
        return 1
return 0
}
Split sub-patterns(sub-pattern){
PS(sub-pattern)=PS(sub-pattern)-1
size=0
return Find Pattern(size, sub-pattern, PS(sub-pattern))
}
```

starting point of the positions that are searched, i.e. the start of the large pattern section. To define the section, we start from the x position and add large pattern parts until the section size is equal to the small PS. Then, the OR operation is applied between the small pattern and the section. If the sub-pattern contains H, the x position and the sub-pattern are stored. If the sub-pattern already exists, only the x position is stored and the R is increased. After the sub-patterns identification, the repetition period of the sub-patterns is computed. This exploration affects the near-optimality because if the sub-pattern is not repetitive (or has a low R), it can be assumed as an access pattern. The gain of maintaining the non-repetitive patterns achieved by the storage size management techniques cannot be compensated with the significant overhead that they introduce in the address generation phase, as explained in Section 3.4. In this way we control the near-optimalities introduced in the resulting pattern. If the different starting positions of the sub-pattern are regular, the sub-pattern is maintained as a pattern with LB=min(x) and UB=R*PS. Otherwise the sub-pattern is further split to smaller sub-patterns and regularity over the period is explored for each sub-pattern. The process is scalable as it is repeated few times, i.e. from PS(sub-pattern) which is small up to a PS equal to two.

4.5.4.5 Decomposition to Small Patterns Operation

This operation is similar to the sub-pattern search operation, but it is applied when both patterns have large PS and it searches for small patterns instead of sub-patterns. The operation is applied in iterations. Each iteration explores a small pattern by applying the operation between the medium pattern and a section of the large pattern of medium size. The positions that are searched are factors of the medium PS. If the small pattern contains H, the start position and the small pattern are stored. If the small pattern has been identified in a previous iteration, the new start position is stored and the R is increased. Scalability is maintained, since the operation is applied in a finite set of positions. The repetition period of the small patterns is computed. If the small pattern is not repetitive (or has a low R), it can be assumed as a virtual accesses due to address generation overhead. The virtual accesses are small and few in number, due to low repetition factor. In this way, the proposed methodology controls the near-optimalities in the pattern representation.

4.5.4.6 Regularity Small Search Operation

This operation is applied when the LCM is not acceptable for a small and a large pattern, where the PS of the small pattern is a factor of the PS of the large pattern. The operation explores the potential repetition of smaller patterns which may exist in the result without computing the final result. It is applied in sections of the pattern in a similar way to the post-processing repetition search, which is however applied in pattern parts. Initially, the potential repetitive pattern is computed by the OR operation of the first small pattern and a section of the large pattern with size equal to the PS of the small pattern. The next candidate to be searched is computed by an OR operation of the second small pattern and the corresponding section of the large pattern. If the result is equal to the first result, the result pattern is repetitive and the next iteration and searches the remaining sections. Otherwise, the potential repetitive pattern is increased by one small pattern (until the half the size of the large pattern) and the process is repeated. The scalability is maintained since the operation is applied in a few pattern sizes (PScnt) each time and in a limited set of specific positions in the large pattern (POScnt). When scaling up the pattern size, PScnt and POScnt will not rise proportionally in practical cases.

4.5.4.7 Regularity Medium Search Operation

The operation is similar to the Regularity Small Search Operation, but it is applied when both patterns are large and it uses the medium pattern in the position of the small pattern.

4.5.5 Pattern Combination Process

This section describes how the patterns and the pattern operations of the proposed representation are applied to combine a set of different patterns of read statements and manifest conditions. The proposed representations describes each primitive condition and access statement by a pattern. It applies pattern operations between the relevant condition patterns to compose the final pattern per

ALGORITHM 7: Finding the active combination case

```
if (PS(FPCH) ≠ PS(SPCH)) then
    if (PS(FPCH) ≥ PS(SPCH)) then
        max=FPCH
        min=SPCH
    else
        max=SPCH
        min=FPCH
    Find GCD(max,min)
    Find LCM(max,min, GCD)
    if (LCM < Threshold-LCM) then
        PS Modification(FPCH,SPCH,LCM)
    else
        if (PS(max) && PS(min) > Threshold-PS)
        then
            if (GCD > 1) then
                Regularity Medium Search(max,min)
            else
                Decomposition to small
                Patterns(max,min)
            if (PS(min) < Threshold-PS) then
                if (GCD > 1) then
                    Regularity Small
                    Search(max,min)
                else
                    Sub-pattern Search
else
    if (LB(FPCH)+IR(FPCH) > LB(SPCH)) then
        if (LB(FPCH)+IR(FPCH) ≠ LB(SPCH) +
        IR(SPCH)) then
            if (LB(FPCH) < LB(SPCH)) then
                LB alignment(FPCH,LB(SPCH))
            if (UB(FPCH) ≠ UB(SPCH)) then
                if R(FPCH) > R(SPCH) then
                    max=FPCH
                    min=SPCH
                else
                    max=SPCH
                    min=FPCH
                UB alignment(max,UB(min))
        if
        (LB(FPCH)+IR(FPCH)==LB(SPCH)+IR(SPCH))
        then
            PCH=Fully aligned(FPCH,SPCH)
            Repetition Search(PCH)
if (LB(FPCH)+IR(FPCH)==LB(SPCH)) then
    Sequential(FPCH,SPCH)
if (LB(FPCH)+IR(FPCH) < LB(SPCH)) then
    Non-sequential(FPCH,SPCH)
```

access statement. The final patterns per access statement are combined by applying OR pattern operations. Since the read statements are applied after the write statements in the problem under study, the final storage derives from the summation of the length of the A in the final pattern.

To compute the pattern per access statement, the pattern operations are applied on a sorted pattern list in increasing LBs and in increasing IR, as second criteria. The first and the second patterns are selected and their parameters are compared to decide in which pattern combination case they belong to. Alg. 7 describes the conditions which decide the active pattern combination case. Initially, the PS of the patterns are compared. When the patterns have different PS, the pattern with the larger PS is defined (max pattern). The Greatest Common Divisor (GCD) and the Least Common Multiple (LCM) of the pattern sizes are computed. If the LCM has a value that is lower than the acceptable LCM threshold, the acceptable LCM is active and the PS modification operation is applied to the patterns. If the LCM is not acceptable, the PS of both patterns is compared with the PS threshold, which distinguishes between the Large PS and the Small PS cases. The value of the GCD decides between the Divisible PS case (GCD>1) or the Indivisible PS case (GCD=1). When the patterns have the same PS, potential overlapping of the patterns is explored. When the sum of the LB and the IR of the FPCH is larger than the LB of the SPCH, the overlapping case is active. To distinguish the different overlapping cases, the sum of the LB and IR of the two patterns are compared. If they are different, the patterns are not aligned. The LBs and the UBs are respectively compared to define where the misalignment occurs and the corresponding operations are applied. In the UB alignment case, the repetition factor is compared to define the larger pattern. When the sums of the LB and IR of the two patterns are equal, the fully aligned case is active.

```
For (I=0; I<N; I++)
  For (K=0; K<M; K++)
    If ( (I≥8K) && (I≤8K+2) || (I<1) || (I==4K+1) )
      ...=A[I]
    EndIf
  EndFor
EndFor
```

Figure 4.9: Demonstration case study code.

Based on the type of the primitive operation, the OR or AND case is selected. When the sum of the LB and IR of the FPCH is equal to the LB of the SPCH, the active case is the non-overlapping sequential patterns. When the sum of the LB and IR of the FPCH is less than the LB of the SPCH, the active case is the non-overlapping non-sequential patterns. The corresponding operations are performed as described in previous sections. The result pattern is stored and the initial patterns are removed from the sorted list. The new patterns created during the pattern operation are also sorted in the list. The process is repeated, until the list has one pattern.

To define the final storage requirements, the OR pattern operation is applied to the final patterns of the read statements. The result is the global read pattern of the application, which describes in a near-optimal and compact way the global valid access scheme. The result pattern describes the storage requirements (Eq. 4.13): each part with a PT equal to A has elements that are accessed and thus are required to be stored, whereas the parts with PT=H describe zero resource requirements. The repetition factor describes the times the pattern is applied in the iteration space.

$$Size_{\text{A}} = \sum_{i=0}^{PS} iPIR(PCH)_{(\text{PT==A})} * R + \sum_{i=0}^{ECS} PIR(i)_{(\text{PT==A})} \tag{4.13}$$

4.6 Demonstration case study

This section demonstrates how the proposed pattern representation and the pattern operations are applied to find the required storage size. The application code consists of two for nested loops and three manifest conditions over the iterator I, which control the accesses to the array A, as depicted in Fig. 4.9. For the PCH conditions I≥8K && (I≤8K+2), the LB of loop I is defined by LB'=max(-1, 8*(LB_K+1)-1)=max(-1,-1)=-1, the UB is defined by UB'=min(N, 8*(UB_K-1)+8=min(N, 8(M-1)+8). The PCH condition i==4k+1 has LB'=max(-1,4*(LB_k+1)+1-1)=max(-1,0)=0. The UB is defined by UB'=min(N, 4*(UB_k-1)+4=min(N, 4(M-1)+4).

In Fig. 4.10(a) the patterns of the three conditions are schematically depicted. The OR primitive operation is applied between the patterns. We describe the OR operation between PCH1 and PCH2. The result is merged with ECS1. The PCH1 and PCH2 have different PS, i.e. PS(PCH1)=8 and PS(PCH2)=4. The LCM is 8 which is acceptable. The pattern PCH2 is modified to PCH2' by applying the PS modification operation, as depicted in Fig. 4.10(b). The PCH1 and new PCH2 patterns have both the LB and the UB not aligned. First the LB alignment operation is applied. The result is depicted in Fig. 4.10(c), where the PCH1 has been divided into an ECS2 and an ECS3

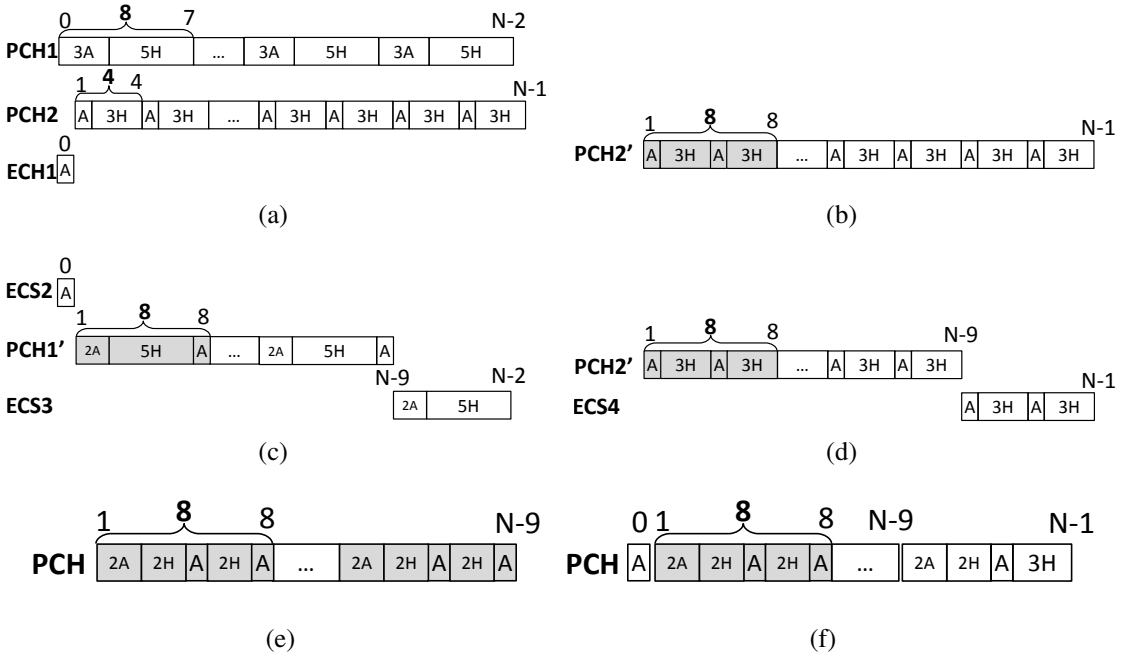


Figure 4.10: Process to compute the storage requirements through pattern operations: (a) Initial patterns derived from primitive conditions, (b) New PCH2 after applying PS modification operation, (c) New PCH1, ECS2 and ECS3 after applying LB alignment operation, (d) New PCH2 and ECS4 after applying UB alignment operation, (e) Result pattern after applying fully aligned OR operation and (f) Global pattern describing the storage requirements

and a new PCH1 (PCH1'). The next step is to align the UB of the PCH2' in order to fully align the two patterns. The result is depicted in Fig. 4.10(d) where UB of PCH2' has been reduced to the UB(PCH1') and a new ECS4 has been inserted. The PCH1 and PCH2 are fully aligned and the OR operation is applied in Fig. 4.10(e). The ECS conditions are also merged and attached to the global pattern, which describes the final pattern (Fig. 4.10(f)) Since the read statements are applied after the write statements, the final pattern describes the storage size. The latter is computed by the summation of the PIRs of A type, i.e. 1+R*4+3.

4.7 Conclusions

This chapter present a systematic representation to achieve a near-optimal and scalable intra-signal in-place step, which is applicable for iteration spaces with irregularly placed holes. The proposed methodology: 1) introduces the formulation of the patterns to describe the condition and the access statements, 2) defines the different pattern combination cases and present scalable and near-optimal operation to perform the combination per case and 3) applies the patterns and the operations to compute the final read pattern, used to compute the storage requirements in the next step of the developed intra-signal in-place methodology.

Chapter 5

Intra-signal in-place methodology for non-overlapping & overlapping scenario

5.1 Introduction

In this chapter we describe how the pattern representation of Chapter 4 is used in the translation step and the parametric templates for intra-signal in-place step of the developed methodology of Chapter 3 for the cases of non-overlapping store and load access scheme, i.e. all the store statements are executed before the load statements, and overlapping store and load access scheme, i.e. the store statements are executed concurrently with the load statements. The first step is the Analysis, which maps the application instance to a unified parametric template and extracts the information for the access scheme, as described in Chapter 3. The translation step translates the access scheme information into a representation, which supports near-optimality and scalability of the intra-signal in-place step, using patterns. The composed patterns, are propagated to the intra-signal in-place step, where the final storage size is computed. The analysis and the translation step is similar for both the non-overlapping and the overlapping store and load case. We fully describe the methodology for the non-overlapping case for all the steps, whereas in the overlapping case we fully describe the intra-signal in-place step and provide the differences with the non-overlapping case for the translation step. The analysis step remains the same. The proposed methodology is systematically applied: the analysis of the application instance gives specific values to the parameters of the first step. The constraint propagation of the values of the source step selects the valid cases in the destination step. The corresponding solutions are applied and the parameters of the destination step are instantiated. We support our contribution by demonstrating how the proposed methodology is applied and evaluate our methodology for several representative test vehicles of PolyBench, MediaBench and Mibench benchmark suites.

5.2 Step 2: Translation

The translation step provides a scalable and near-optimal representation of the access scheme per statement, which avoids enumeration of the different iteration space parts and efficiently describes the potential irregular holes. The conditions of enumerative type (e.g. i<5) describe one pruned/accessed part, whereas the conditions of parametric type (e.g. i!=2k+1) describe a potentially high number of pruned/accessed parts. When several conditions co-exist, the access scheme is the result of the consistent combination of the iteration spaces of the conditions and the access statements. The result consists of irregularly spread access regions. To deal with the complexity of the iteration space shape, the translation step uses the concept of patterns and pattern operations to represent the access scheme. The output of the translation step is combined patterns, depending on the condition types, per dimension. Hence, the translation process is applied per access statement: it translates the information of the access statement and of the primitive conditions relevant to the access statement into primitive patterns per dimension and depending on which translation option is valid, the primitive patterns may be combined through pattern operations into the final patterns per dimension.

The translation cases, described in Chapter 3, depend on the loop structure dimensions, the number of index expressions in the access statement, i.e. one index expression (e.g. A[i-1]) or several index expressions (e.g. A[i-1][j+1]), the type of the index expressions, the type of the primitive condition cases and the potential coupling of the loop dimensions due to conditions. The cases, which are valid per application instance, are selected by the information propagated by the analysis step. In realistic contexts, the translation process is applied in a quite small set of primitive patterns, which is defined by the number of conditions and access statements in the application kernel, maintaining the scalability of the translation step. As the storage size result should support a pseudo-regular address generation, the holes in the iteration space, which are too small to provide a gain in the storage size, i.e. parts of size one, while they introduce a high address generation cost due to disabling repetition can be considered as ``virtual'' accesses. This process will insert near-optimalities, which are, however, fully controllable. The next paragraphs describe the translation cases.

5.2.1 One loop dimension

When the loop consist of one loop dimension, the primitive conditions that may exist in the unified application template are: 1) SIS of ECS type and 2) ISH of ECH type, since a second iterator does not exist to form a parametric condition. The valid condition expressions are combinations of ECS and ECH type through AND and OR operations. We schematically describe the translation process through Fig. 5.1. Each primitive condition and the access statement is translated into a primitive pattern following the process described in Chapter 4. Fig. 5.1(b) schematically depicts the patterns of three primitive conditions (C1, C2, C3) and one read statement (RD). Pattern operations are applied to the primitive condition patterns to compose the final condition pattern, following the

process in Chapter 4. The pattern operations pre-process the patterns, when needed it (e.g. alignment operations are required), and perform the OR or AND operation of the application condition expression. For instance, when the primitive condition patterns are not aligned, LB alignment or UB alignment operations are applied to align the bounds, as depicted in Fig. 5.1(c). The overlapping patterns with non-align bounds (C1, C2) are split into patterns, which have aligned bounds, e.g. C1 is split into two patterns: C1 pattern, which is non-overlapping sequential with the C2, and C1' pattern, which is overlapping with C2. Then, C2 is split into C2 pattern, which is fully aligned pattern with C1', and a non-overlapping sequential C2' pattern. Then, the OR condition expression operation is applied over C1' and C2, the sequential non-overlapping operations between C1, the resulting pattern of the OR operation and C2' and non-sequential non-overlapping operation between the result of previous operations and C3. The resulting pattern is depicted in Fig. 5.1(d). After the final condition pattern has been computed, it is combined with the access pattern through an AND operation. The AND operation is used because both condition and access statement should be concurrently valid in order to access the array elements. The result is depicted in Fig. 5.1(e). We take as reference that the index expression of the write pattern has a constant equal to zero. In case the index expression of the read access pattern has also a constant equal to zero, the pattern describes both the access scheme, i.e. the iterators where the statement is executed, and the array elements that are accessed. When the constant is not zero, the pattern requires a shift in order to describe the accessed elements. If the constant is negative, the pattern is shifted left (Fig. 5.1(f)), otherwise is shifted right.

For instance, in the SIS example of Fig. 5.2(a), the condition pattern of first read statement is {5A}, LB=4, UB=10, IR=5, PS=5 and R=1, the condition pattern of second read statement is {4A}, LB=4, UB=9, IR=4, PS=4 and R=1. The condition pattern of write statement is {5A}, LB=-1, UB=5, IR=5, PS=5 and R=1, the access pattern of first and second read and the write statement is {11A}, LB=-1, UB=10, IR=11, PS=11 and R=1. By applying the AND operation between the condition pattern and the access pattern, the results are: for the first read {5A}, LB=4, UB=10, IR=5, PS=5 and R=1, for the second read {4A}, LB=4, UB=9, IR=4, PS=4 and R=1 and for the write {5A}, LB=-1, UB=5, IR=5, PS=5 and R=1, and R=1. Since the index expression of both read is ``iterator+constant'', the pattern of the first read is shifted left by 5 and the second by 3, i.e. R1: {5A} LB=-1, UB=5, IR=5, PS=5 and R=1 and R2: {4A}, LB=0, UB=5, IR=4, PS=4 and R=1. In the ISH example of Fig. 5.2(b), the pattern of first primitive ECH condition in the first read statement is {1A}, LB=4, UB=6, IR=1, PS=1 and R=1 and the pattern of the second primitive ECS condition is {2A}, LB=7, UB=10, IR=2, PS=2 and R=1. By applying the OR operation in the condition expression the result is {1A 2H 2A} with LB=4, UB=10, IR=5, PS=5 and R=1. The pattern of the read statement is {11A} with LB=-1, UB=11, IR=11, PS=11 and R=1. After applying the AND operation between the combined condition pattern and the read pattern the result is {1A 2H 2A} with LB=4, UB=10, IR=5, PS=5 and R=1. Since the index expression is i-5, the pattern is shifted left by 5, i.e. LB=-1 and UB=5. The final pattern of the second read statement after applying the AND and the shift operation is {1A} with LB=1, UB=3, IR=1, PS=1 and R=1. In a similar way, the final pattern of the write statement is {1A 1H 3A},

```
For (i=0; i<UB(RD); i++)
  If C1(i>LB(C1)&&i<UB(C1))||
     C2(i>LB(C2)&&i<UB(C2))||
     C3(i>LB(C3)&&i<UB(C3))
       A[i]=...
  EndIf
EndFor
```

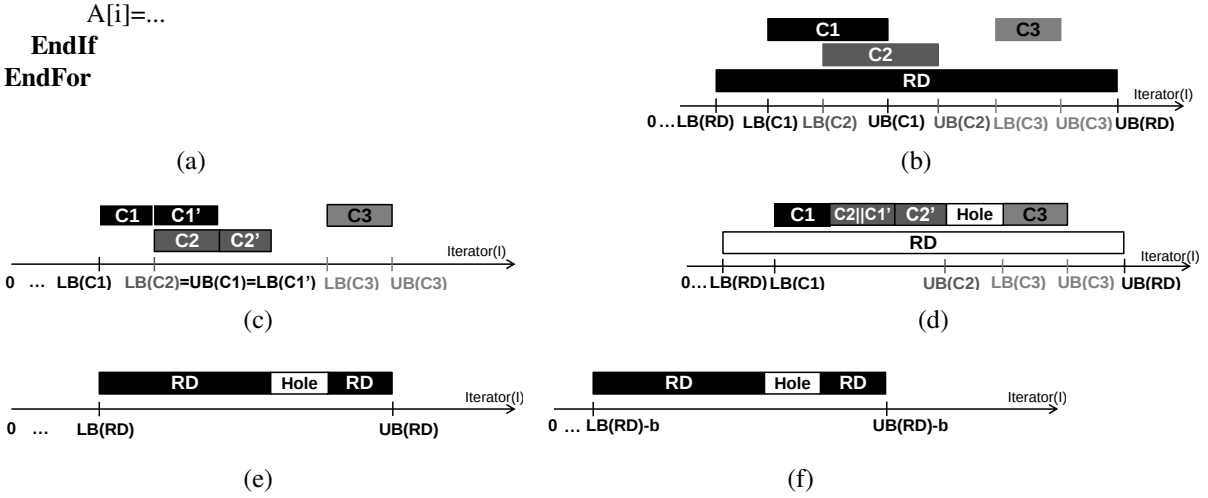


Figure 5.1: Schematic description of translation step: (a) Initial patterns derived from primitive conditions (C1, C2, C3) and read access pattern (RD), (b) New patterns after applying LB and UB alignment pattern operations, (c) Combined condition pattern results after applying OR, sequential non-overlapping and non-sequential non-overlapping pattern operation, (d) Final pattern for read statement after applying AND pattern operation between the read pattern and the combined condition pattern and (e) Shifted pattern of the accessed elements due to index expression i+b.

LB=-1, UB=5, IR=5, PS=5 and R=1.

5.2.1.1 Several loop dimensions

When the loop consist of more than one loop dimension, each loop dimension describes part of the final storage size. The partial storage size result of the outer dimension affect the size described by the inner dimensions. The exploration of the partial storage size of each dimension cannot be performed independently due to potential coupling of the dimensions. The coupling is created due the parametric primitive conditions, where more than one loop iterators are required, and in condition expressions, which combine primitive conditions of different iterators. However, in some cases the loop dimensions can be decoupled. Then, each dimension is translated independently, as described in Section 5.2.1.1.1 and Section 5.2.1.1.2. If decoupling is not possible, the coupled loop dimensions are explored together, as explained in Section 5.2.1.1.3.

5.2.1.1.1 Independent iterators The loop dimensions are independent when: 1) no condition expressions exist, 2) condition expressions of ECS and ECH are in one loop dimension, 3) condition expressions of ECS and ECH are coupled through AND operation, 4) PCH of $==$ or $<$ && $>$ type for only two dimensions, 5) PCH of $==$ or $<$ && $>$ type combined through AND primitive operation and PCH of $==$ or $<$ && $>$ type combined through AND primitive operation with $\neq$ type for only two dimensions and 6) PCH combined through AND primitive operations with

```
For (I=0; I<11; I++)
  If (I<5)
    A[I]=...
  EndIf
  If (4<I<10)
    ...=A[I-5]
  EndIf
  If (4<I<9)
    ...=A[I-4]
  EndIf
EndFor
```

(a)

```
For (I=0; I<11; I++)
  If (I==0) || (1<I<5)
    A[I]=...
  EndIf
  If (I==5) || (7<I<10)
    ...=A[I-5]
  EndIf
  If (I==6)
    ...=A[I-4]
  EndIf
EndFor
```

(b)

Figure 5.2: Examples with one loop dimension: (a) SIS and (b) ISH with ECH

ECS and ECH of different dimensions. Then, each loop dimension is translated independently into patterns, following the process of one loop dimension case of Section 5.2.1 for ECS and ECH conditions and the decoupled case of Section 5.2.1.1.2 for PCH conditions.

5.2.1.1.2 Decoupled iterators The decoupling of the loop dimensions is possible in the cases described in the next paragraphs.

a. Iterator excluded from index expressions: When the iterator of the dimension which is explored (e.g. I) is coupled to another loop iterator (e.g. K), which is not in any index expression of the array, the dimension can be eliminated for the intra-signal in-place optimization. The loop iterator extends the accesses of the elements by one dimension (when K is inner dimension) or repeats the access scheme (when K is outer dimension), but it does not affect the storage size. The same elements are accessed, but in different times. The decoupling process is a consistent replacement of the coupling conditions. The replacement policy derives from the different type of coupling conditions: i) Parametric condition type: The accesses in the k iterator are described by the parametric pattern in I iterator. For instance, a condition I==2K is described by the pattern {1A 1H}, I>6K && I<6K+3 is described by the pattern {1H 2A 3H} and I≠2K is described by the pattern {1H 1A}. The I<c*K+d is mapped to i<((c*UB_k-1)+d)+1 and I>c*K+d is mapped to i<((c*LB_k+1)+d)-1 and ii) Condition expressions through primitive operations: 1) AND: The conditions over the K iterator are eliminated without replacement, because for one valid K iterator, only the valid I iterators (due to AND) will allow the access statement to execute and 2) OR: The conditions over the K iterator are replaced by the condition which describes all the iterator values on I iterator, because for one valid K iterator, all the I iterators (due to OR) will allow the access statement to execute.

b. Iterators included in different index expressions: When the iterator of the dimension which is explored (e.g. I) is coupled to another loop iterator (e.g. J), decoupling can be achieved in the cases of: i) Parametric conditions of PCH of ≠ type and ii) Condition expressions: 1) PCH combined through AND operation with ECH or ECS applied in the same dimensions, 2) ECS/ECH

through OR operation, 3) PCH with ECH or ECS with OR operation and 4) PCH through OR.

The translation step creates the primitive patterns following the translation process in one dimension and combines the ECH and ECS conditions to a combined pattern. The PCH conditions are translated into primitive patterns per coupling iterators and are explored in the next step due to their coupling effect.

5.2.1.1.3 Coupled iterators The coupling is due to conditions of parametric type of PCS for SIS. When the loop dimensions are fully coupled, the translation step does not modify the information and propagate it to the intra-signal in-place step.

5.3 Step 3: Intra-signal in-place optimization for non-overlapping case

The intra-signal in-place optimization step computes in a scalable and near-optimal way the storage size for the target domain described in Section 3.4. The different intra-signal in-place cases depend on the combinations of the unified template parameters and the translation step cases. Based on the patterns and application structure information, which is propagated from the previous steps, the appropriate intra-signal in-place cases are selected and the corresponding solutions are applied. The next paragraphs describe the non overlapping intra-signal in-place cases and the corresponding solutions.

5.3.1 One loop dimension

When one dimension exists in the loop structure, the array dimension of the access statements is also one and the index expression includes the loop iterator. The storage size depends on the position of the read and write access statements. To compute the final storage requirements, the patterns of the read access statements are combined through an OR operation. The OR operation is used, since the final access scheme, thus the storage size, should take into account every read of the array in the iteration space. An element is accessed when at least one of the read statements is executed and the corresponding condition expressions are true. The result is the global read pattern, which describes in a near-optimal and compact way the access scheme. We perform the analysis of the storage size requirements based on the read patterns, since they provide the life of the variables for the problem under study. The global read patterns describe the storage requirements: each part with a PT equal to A has elements that are accessed and, thus, are required to be stored, whereas the parts with PT equal to H describe zero resource requirements. The repetition factor of the final pattern describes the times the pattern is applied in the iteration space.

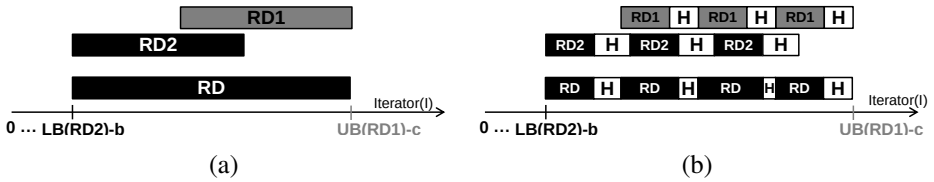


Figure 5.3: Schematic description of two read patterns RD1 and RD2 and the result of the OR pattern operation (RD) for I dimension. The intra-signal in-place storage size is the summation of the black parts of the final RD pattern: (a) for SIS and (b) for ISH with ECH conditions.

5.3.1.1 SIS

This case is valid, when all patterns describe solid iteration spaces. Fig. 5.3(a) schematically illustrates the patterns of two read statements, i.e. RD1 with index expression ``i+c" and RD2 with index expression ``i+b" and their OR combination in the iteration space, RD. The final read pattern (RD) describes a SIS and the storage size is given by Eq. 5.1. For instance, in Fig. 5.2(a) the OR pattern operation between the read patterns results to {5A}, LB=-1, UB=5, IR=5, PS=5 and R=1, which is identical to the write pattern and the final storage size is 5.

$$Size = IR(RD) \tag{5.1}$$

5.3.1.2 ISH

This case is valid when at least one pattern describes holes in the iteration space. The schematic illustration of the ISH for RD1 with index expression ``i+c" and for RD2 with index expression ``i+b" is depicted in Fig. 5.3(b). The storage size is given by the regions that are accessed, i.e. the summation of the PIRs parts of the final RD with PT equal to A (defined as SID in Eq.. 4.6) multiplied by the corresponding repetition factor, as described in Eq. 5.2. For instance, in Fig. 5.2(b) the OR operation between the two read patterns propagated by the translation step results to {1A 1H 3A} with LB=-1, UB=5, IR=5, PS=5 and R=1 and the final storage is 5, i.e. 1*1+3*1.

$$Size = \sum_{i=0} NumPatternsSID_i * R_i \tag{5.2}$$

5.3.2 Several loop dimensions

When several loop dimensions exist, the final storage size derives from consistent combination of the partial storage size result of each dimension through propagation. The intra-signal in-place propagation case is selected by the type of the patterns of the outer dimension. The corresponding process is applied to compute the combined storage size. When more than two dimensions exist, the combined storage size result is used as size of outer dimension in the next combined size computation. The solutions per case are described in Section 5.3.2.1 and the process for the final storage size computation in Section 5.3.2.4.

5.3.2.1 Propagation cases: Independent iterators

When the patterns propagated from the translation step belong to independent loop dimensions, the intra-signal in place step computes the combined storage size by propagating the partial storage size of the outer dimension to the inner using Eq. 5.3. We assume that I-J-K is the order of loop dimensions throughout the paper. The partial storage size per dimension is described by the propagated patterns. For enumerative conditions, the partial storage size is computed by the intra-signal in-place step of one loop dimension process, as described in Section 5.3.1. The partial storage of the parametric condition depended on the propagated case. In the next paragraphs, we illustrate the instantiation of Eq. 5.3 depending on the type of the propagated patterns.

$$Size_{I,J} = Size_I * Size_J \tag{5.3}$$

a. SIS: This case is valid when both patterns describe solid iteration spaces through ECS conditions potentially over both dimensions, i.e. ECS of different iterators coupled by an AND operation. Fig. 5.4(a) schematically illustrates the RD patterns of two independent iterators I and J and the combined storage size, i.e. the gray area. The storage size is given by the multiplication of the number of accesses in the I dimension by the accesses in J dimension (Eq. 5.4). For instance, in Fig. 5.4(c) the final RD pattern of I is {6A}, LB=-1, UB=6, IR=6, PS=6 and R=1 and the final RD pattern of J is {3A}, LB=1, UB=5, IR=3, PS=3 and R=1 and the final storage size is 18, i.e. 6*3.

$$Size_{I,J} = IR(RD_I) * IR(RD_J) \tag{5.4}$$

b. ISH: This case is valid, when at least one pattern describes ISH:

1. ECH conditions only in one dimension. The combined size is given by Eq. 5.5, when the ECH are in I dimension. The schematic illustration of the final RD patterns of I and J and the combined storage size (sum of gray areas) are depicted in Fig. 5.4(b). For instance, in Fig. 5.4(d) the RD pattern for I iterator is {1A 1H 2A} with LB=1, UB=6, IR=4, PS=4 and R=1, the RD pattern for J iterator is {5A} with LB=-1, UB=6, IR=5, PS=5 and R=1 and the final storage is 15, i.e. (1+2)*5.

$$Size_{I,J} = SID_I * IR(L_J) \tag{5.5}$$

2. ECH conditions coupled with AND operator. The storage size is given by Eq. 5.6, when both I and J have ECH conditions.

$$Size_{I,J} = SID_I * SID_J \tag{5.6}$$

3. PCH conditions of $==$ or $<$ && $>$ type for only two dimension, i.e. Eq. 5.7, since I iterator

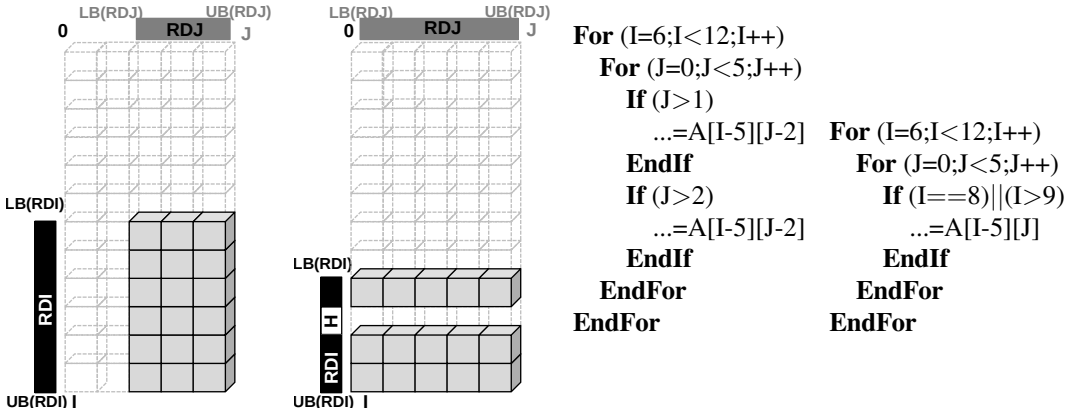


Figure 5.4: Schematic description of the RD patterns for I and J independent dimensions. The intra-signal in-place storage size is the sum of the gray areas for: (a) for SIS and (b) for ISH and ECH conditions in I loop dimension and code examples for (c) SIS and (d) ISH with ECH in I loop iterator

is dominant and, thus, the $Size_J$=1.

$$Size_{I,J} = SID_I * R_I \tag{5.7}$$

4. PCH conditions of $==$ or $<$ && $>$ coupled through an AND primitive operation and PCH conditions of $==$ or $<$ && $>$ coupled with $\neq$ type through an AND primitive operation for only two dimensions. The combined size is given by Eq. 5.8, where CP is the combined pattern of the primitive condition pattern with AND operation.

$$Size_{I,J} = SID_{CP} * R_{CP} \tag{5.8}$$

5. PCH conditions coupled through an AND primitive operation with ECH/ECS of different dimensions. The size is given by Eq. 5.9, where $Size_{PCH}$ is derived by Eq. 5.7, for $==$ and $<$ && $>$ type, or by Eq. 5.14, for $\neq$ type. We assume that PCH couples I and J dimensions and ECS/ECH is in K dimension.

$$Size_{I,J,K} = Size_{PCH} * SID_K \tag{5.9}$$

5.3.2.2 Propagation cases: Decoupled iterators

When the dimensions can be decoupled, the partial storage size of each dimension is independently computed by applying the 1 loop dimension case or the independent case. The propagation of the outer dimension to the inner dimension is given by Eq. 5.10, where $Size_{L_J}$ is the size of the complete loop of J iterator. When more than two dimensions exist, the result of the partial size computation is used as size of outer dimension (i.e. $Size_{L_I}$) in the next combination of partial storage sizes and the number of holes is the total holes in both dimensions of previous propagation, as explained in Section 5.3.2.4. The instantiation of Eq. 5.10 depends on the type of the iteration

space described by the patterns, on the type of the conditions (enumerative of parametric) and on the type of coupling, as described in the next paragraphs.

$$Size_{I,J} = Size_I * Size_{L_J} + Holes_I * Size_J \tag{5.10}$$

5.3.2.2.1 SIS: This case describes the combined storage size, when both patterns describe solid iteration spaces and are combined with OR primitive operator. Fig. 5.5(a) schematically illustrates the final read patterns of two decoupled iterators I and J and the final storage size (total gray area). In case the IR of the I dimension is equal to the IR of the RD pattern of the I dimension, the I loop is dominant, i.e. no holes exist in the outer dimension, thus no additional space is required to be computed due to the J iterator. In this case, the I iterator includes all the information, since the dark gray area of J iterator of Fig. 5.5(a) will be completely covered by the gray area of the I dimension. The storage size is given by the multiplication of the number of accesses in the I dimension by the size of the J dimension loop ($IR(L_J)$) in the dominant case (Eq. 5.11). Otherwise, the storage size has to be increased by the accesses of the RD pattern of J that have not been yet computed. The latter is given by the number of holes in I dimension, i.e. the difference of the size of loop and the size of the RD in I dimension, multiplied by the size of the RD pattern in the J dimension. The corresponding equation is Eq. 5.12. The additional storage size due to the J iterator is the dark gray area in Fig. 5.5(a). For instance, in Fig. 5.5(e) the final RD pattern of I is {4A}, LB=-1, UB=6, IR=4, PS=4 and R=1 and the final RD pattern of j is {2A}, LB=1, UB=4, IR=3, PS=2 and R=1 and the final storage size is 28, i.e. 4*6+2*2.

$$Size_{I,J} = IR(RD_I) * IR(L_J) \tag{5.11}$$

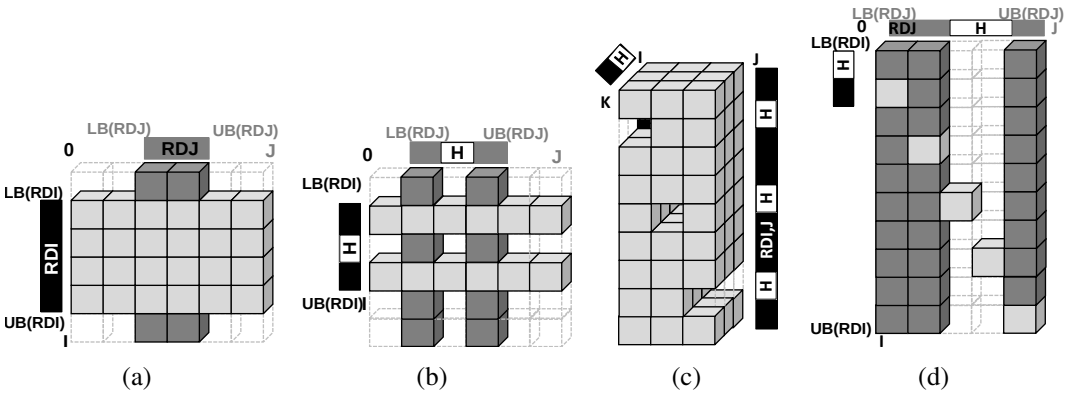
$$Size_{I,J} = IR(RD_I) * IR(L_J) + (IR(L_I) - IR(RD_I)) * IR(RD_J) \tag{5.12}$$

5.3.2.2.2 ISH

i. ECH coupled with OR primitive operation: This case is valid when ECH conditions exist in the iterators, which are coupled through OR primitive operations. The schematic illustration of the RD patterns in the iteration space and the corresponding storage size are depicted in Fig. 5.5(b). The combined storage size is given by Eq. 5.13, since both dimensions describe iteration spaces with holes. For instance, in Fig. 5.5(f) the I dimension is not dominant. The RD pattern for I iterator is {1A 1H 1A} with LB=0, UB=4, IR=3, PS=3 and R=1, the RD pattern for J iterator is {1A 1H 1A} with LB=0, UB=4, IR=3, PS=3 and R=1 and the final storage is 20, i.e. 2*6+4*2.

$$Size_{I,J} = SID(I) * IR(L_J) + HID_I * SID(J) \tag{5.13}$$

ii. PCH $\neq$ type: The size is given by Eq. 5.14, where the number of accesses in the I dimension are multiplied by the length of the J dimension and adds the accesses in the J dimension, when holes exist in the I dimension. The latter is the size of the loop in J dimension minus 1 due



(a) (b) (c) (d)

```
For (I=0;I<6;I++)
 For (J=0;J<6;J++)
  If (0<I<5)||(1<J<4)
   ...=A[I][J]
  EndIf
 EndFor
EndFor
```

(e)

```
For (I=0;I<6;I++)
 For (J=0;J<6;J++)
  If (I==1)||(I==3)||
     (J==1)||(J==3)
   ...=A[I][J]
  EndIf
 EndFor
EndFor
```

(f)

```
For (I=0;I<9;I++)
 For (J=0;J<3;J++)
  For (K=0;K<3;K++)
   If (I≠3J+1)||(I==2K)
    ...=A[I][J][K]
   EndIf
  EndFor
 EndFor
EndFor
```

(g)

```
For (I=0;I<10;I++)
 For (J=0;J<5;J++)
  If (I==2J+1)||
     (J<2)||(J>3)
   ...=A[I][J]
  EndIf
 EndFor
EndFor
```

(h)

Figure 5.5: Schematic description of the final RD patterns for 2 decoupled dimensions. The combined storage size is the sum of the gray areas: (a) SIS, (b) ISH with ECH coupled with OR primitive operation, (c) ISH with PCH coupling conditions and (d) ISH with PCH and ECH combined through OR primitive operation. Code examples are presented in : (e) SIS, (f) ISH with ECH, (g) ISH with PCH and (h) ISH with PCH and ECH.

to hole of I dimension.

$$Size_{I,J} = SID(I) * R_I * IR(L_J) + HID(I) * R_I * (IR(L_J) - 1) \tag{5.14}$$

iii. PCH coupling of different dimensions through OR primitive operation: The schematic description of the patterns and the combined storage size is given in Fig. 5.5(c). The storage described by the PCH1 is the gray area. The size of PCH2 is the black area. The combined storage is the summation of the gray and black areas. When at least one PCH is of $\neq$ type, it is considered as dominant. The combined storage size is the size of the $PCH_{\neq}$ multiplied by the size of the dimension not used in $PCH_{\neq}$ plus the number of holes of pattern $\neq$ that are becoming accesses in the second PCH ($H2A$), multiplied by 1, if PCH is $==$ or $<$ && $>$ type (Eq. 5.15), or by the dimension not used in PCH pattern, minus potential accesses only in J dimension, if PCH is $\neq$ type (Eq. 5.16). We assume that the I and J dimension are coupled by PCH of $\neq$ type and the second PCH couples I with K dimension. If both patterns are of $==$ or $<$ && $>$ type, the size is derived by Eq. 5.17, i.e. the size of PCH1 is multiplied by the size of the dimension not used in PCH1 plus the number of holes of PCH1 pattern that are becoming accesses in the PCH2

pattern ($H2A$) multiplied by the dimension not used in PCH2, minus potential accesses only in J dimension, and the number of accesses of PCH1 pattern that are also accesses in the PCH2 pattern ($A2A$) multiplied by the dimension not used in PCH2, minus potential accesses only in J dimension, minus 1. For instance in Fig 5.5(g), the $\neq$ pattern for I-J iterator is {1A 1H 1A} with LB=-1, UB=12, IR=12, PS=3 and R=4. The pattern of I-K iterator is {1A 1H} with LB=-1, UB=6, IR=6, PS=2 and R=3. The size of $Size_{I,J}$=8*4+4*3 from Eq. 5.14. The total size is 24*3+1=25. The first term of Eq. 5.15 is 2*3*3+1*3*2=24 and the second term is 1 since one H has become A in I-K pattern.

$$Size_{I,J,K} = Size_{PCH_{\neq}} * IR(L_K) + H2A(PCH_{\neq}, PCH) \tag{5.15}$$

$$Size_{I,J,K} = Size_{PCH_{\neq}} * IR(L_K) + H2A(PCH_{\neq}, PCH) * (IR(L_J) - SID_J) \tag{5.16}$$

$$\begin{aligned} Size_{I,J,K} &= Size_{PCH1}) * IR(L_K) + H2A(PCH1, PCH2) * (IR(L_J) - SID_J) \\ &+ A2A(PCH1, PCH2)(IR(L_J) - SID_J - 1) \end{aligned} \tag{5.17}$$

iv. PCH (at least one $\neq$ type) coupling of different dimensions through AND primitive operation: This case is similar to the previous, but due to AND operation, we subtract the elements not accessed due to the second pattern. If one pattern is of $==$ or $<$ && $>$ type, it is considered as dominant, i.e. PCH1. The size is the size of the PCH1 multiplied by the size of the dimension not used in PCH1 minus the number of accesses of PCH1 that are becoming holes in the PCH2 ($A2H$) (Eq. 5.18). If both PCH are of $\neq$ type, last term is multiplied by the dimension not used in PCH pattern (Eq. 5.19). We assume that the I and J dimension are coupled by PCH1 and PCH2 couples I with K dimensions.

$$Size_{I,J,K} = Size_{PCH1} * IR(L_K) - A2H(PCH1, PCH2) \tag{5.18}$$

$$Size_{I,J,K} = Size_{PCH1} * IR(L_K) - A2H(PCH1, PCH2) * IR(L_J) \tag{5.19}$$

v. PCH coupling of same dimensions through OR primitive operation: This case is similar to the previous cases, but due to same dimension the sizes are not multiplied by a third dimension and the additional elements are always of size 1. In case where at least one PCH is of $\neq$ type, the combined size is given by Eq. 5.20. When all PCH are of $==$ or $<$ && $>$ type, the size is given by the size of PCH1 plus the number of holes that are becoming accesses in PCH2 plus the number of accesses of PCH1 that are also accesses in PCH2 minus 1, due to the first repetition (Eq. 5.21).

$$Size_{I,J} = Size_{PCH_{\neq}} * IR(L_K) + H2A(PCH_{\neq}, PCH) \tag{5.20}$$

$$Size_{I,J} = Size_{PCH1} + H2A(PCH1, PCH2) + A2A(PCH1, PCH2) - 1 \tag{5.21}$$

vi. PCH coupling with ECH\ECS of same dimensions: When the PCH and ECH conditions are applied to the same dimensions and are combined with OR primitive operator, the iterator of the ECH condition type (J dimension) is considered as dominant and the size of ECH is considered as basis. The combined size is given by the size of ECH in J dimension adding the storage due to the additional accesses in PCH, i.e. the holes in J dimension multiplied by $SID_{I,J}$ in PCH (Eq 5.22).

The $SID_{I,J}$ is for: 1) == type: 1, 2) $\neq$ type: $IR(L_I) - 1$ and 3) $<$ && $>$ type: given by Eq. 4.6. In case of AND operation, the iterator of the PCH is dominant and the size is reduced by the holes created due to J dimension. The combined size is given by Eq. 5.23. The size of PCH is given by Eq. 5.7, for == and $<$ && $>$ type, and by Eq. 5.14, for $\neq$ type. The schematic illustration of the RD patterns and the combined storage size are depicted in Fig. 5.5(d). For instance, in Fig. 5.5(h) the RD pattern for PCH between I and J iterator is {1H 1A} with LB=-1, UB=10, IR=10, PS=2 and R=5, the RD pattern for ECH in J iterator is {2A 2H 1A} with LB=-1, UB=5, IR=5, PS=5 and R=1 and the final storage is 32, i.e. 3*10+2*1.

$$Size_{I,J} = SID_J * IR(L_I) + HID_J * SID_{I,J} \tag{5.22}$$

$$Size_{I,J} = Size_{PCH} - HID_J * SID_{I,J} \tag{5.23}$$

vii. PCH coupling with ECH/ECS of different dimensions through OR primitive operation: This case is similar to the coupling of PCH and ECH/ECS with same dimensions. Instead of the 1 dimension size and the SID, the size of both PCH dimensions and the size of PCH are respectively used, as shown in Eq. 5.24.

$$Size_{I,J,K} = SID_K * IR(L_I) * IR(L_J) + HID_K * Size_{PCH} \tag{5.24}$$

5.3.2.3 Propagation cases: PCS Coupled iterators

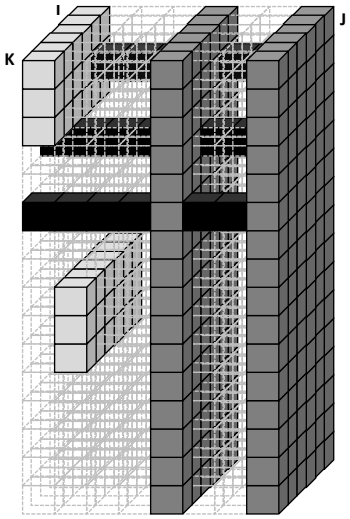
The coupled loop dimensions are due to parametric conditions of solid iteration space. The intra-signal in-place solution for this case is to apply intra-signal in-place methods based on polytope theory per PCS. In realistic applications, the number of coupled loop dimensions, which cannot be decoupled after translation step, is small. Hence, polytope theory provides near-optimal partial storage size results for this case. Then, the PCS partial storage sizes are combined by propagating the result of the outer dimensions to the inner dimensions. To merge the partial storage size of PCS with the remaining cases, the active case from Section 5.3.2.2 is selected, as PCS are PCH without holes. The partial storage size computed by polytope theory is used as $Size_{PCH}$ in the corresponding equations or as outer dimension size, as explained in Section 5.3.2.4.

5.3.2.4 Computation of the final storage size

When several dimensions exist in the application, many of the intra-signal in-place combinations may be concurrently valid. To compute the final storage size, we propagate the results from the outer dimension to the inner dimensions. Initially, the outer dimension, e.g. dimension 0, is combined with the first inner dimension, e.g. dimension 1. Based on the type of the patterns and the potential coupling, the intra-signal in-place cases are selected from Section 5.3.2.1. The corresponding closed form equations are applied to compute the combined storage size. Then, the next dimension is explored, which either is independent from both previous dimensions or it is decoupled/coupled to the previous dimensions. The result of the previous combination is used as the size of the outer dimension, i.e. dimension I in the closed form equations of Section 5.3.2.1.

```
For (I=0; I<1024; I++)
 For (J=0; J<128; J++)
  For (K=0; K<512; K++)
   If ((I≥8J)&&(I≤8J+2))||(J==7)||(J==4)||(I==2K+1)
    A[I][J][K]=...
   EndIf
  EndIf
 EndFor
EndFor
```

(a)



(b)

Figure 5.6: Demonstration case: (a) Code and (b) initial part of the iteration space, where each different color indicate accesses due to one condition.

When information is required from the outer dimension, it is given by the patterns of the combined dimensions.

5.3.3 Demonstration case study description

This section demonstrates how the proposed methodology is applied to compute the final required storage size. The application code is depicted in Fig. 5.6(a). The analysis step provides the information of three for nested loops in the order I-J-K, five manifest conditions: two PCH conditions, which couple iterator I and J combined with an && primitive operation, one PCH condition, which couples the I and K iterator and two ECH conditions on iterator H, combined by the || operation. The propagated information from the analysis step selects: the independent case for PCH of $<$ && $>$ type in I and J, the independent case for ECH in one dimension and the decoupled case of ISH for $\neq$ type. The translation step creates the primitive patterns of the PCH conditions per couple of dimensions: for condition I≥8J && (I≤8J+2), the LB of loop I is defined by LB'=max(-1, 8*(LB_J+1)-1)=max(-1,-1)=-1, the UB is defined by UB'=min(1024, 8*(UB_J-1)+8=min(1024, 1024)=1024, the PS is 8, IR is 1024, the R is 128 and the PCH1 pattern is {3A 5H}. Fig. 5.6(b) depicts the first part of the accessed elements of the three dimensional array. The PCH1 access the light gray elements. The ECH condition of $J == 4$ creates a ECH2 of {1A} pattern with LB'=max(-1,7)=3, UB'=min(1024,5)=5, the PS is 1, IR is 1, the R is 1. The ECH condition of $J == 7$ creates a ECH1 of {1A} pattern with LB'==max(-1,6)=6, UB'=min(1024,8)=8, the PS is 1, IR is 1, the R is 1. In Fig. 5.6(b), the ECH1 and ECH2 access the dark gray elements. The PCH condition I==2K+1 has LB'=max(-1,2*(LB_K+1)-1)=max(-1,-1)=-1. The UB is defined by UB'=min(1024, 2*(UB_J-1)+2=min(1024, 1024)=1024, the PS is 2, the IR is 1024, the R is 512 and the PCH3 pattern is {1H 1A}. The PCH2 access the black elements. During translation step, the ECH conditions are combined in a pattern by applying non-sequential non-overlapping

operation and the result is ECH {1A 2H 1A} pattern with LB'=3, UB'=8, the PS is 5, IR is 5, the R is 1. In the intra-signal in-place step, the process of computing the final storage size starts from the outer dimension. The ISH for PCH coupling with ECS/ECH conditions of same dimensions through || operation is selected. The combined size is given by Eq. 5.22. The combined storage size is $Size_{I,J}$=2*1024+126*3=2,426. The combined patterns of PCH and ECH after || operation are {3A 1H 1A 2H 1A} with LB'=-1, UB'=8, the PS is 8, IR is 8, the R is 1 and {3A 5H} with LB'=7 UB'=16, the PS is 8, IR is 1016, the R is 127. Then, we proceed to the next dimension, where the PCH for coupling of different dimensions through || primitive operations is selected and the final storage size is given by Eq. 5.17. The first term of Eq. 5.17 is 2,426*512=1,242,112, the second terms due to the two combined patterns in I and J dimension are given by (2*512/4)*(128-2)=32,256 and (3*512/4)*(128)=49,152, and the third terms by (1*512/4)*(128-2-1)=16,000 and (1*512/4)*(128-1)=16,256 .The total size is $Size_{I,J,K}$=1,355,776.

5.3.4 Results

In this section, the proposed methodology is compared with an enumerative approach (lower bound) and an approximation approach (upper bound). The enumerative approach uses explicitly each access in the iteration space to compute the storage size by consistently adding the number of accesses, taking into account all the read statements. Hence, the enumerative approach has optimal results. An approximation approach to estimate an upper bound in the storage size is used when the techniques cannot be applied due to irregularity/non-uniformity of the accesses in the iteration space. The upper bound is computed using the approximation of non-uniform access of [160]. The upper bound is given by $Size = max(UB_x) - min(LB_x) + 1$, where $LB_x = f_x(i = LB_i, j = LB_j, ...), UB_x = f_x(i = UB_i, j = UB_j, ...)$ and f_x is index expression of each access. Whenever the function cannot be calculated due to conditions, the approximation solidifies the iteration space.

We have implemented a prototyping tool for the proposed methodology, which fully computes the partial storage size for a dimension and manually propagate the partial sizes between dimensions. We also implemented the enumerative approach and performed experiments for a set of benchmarks from the Polybench [159], MiBench [63] and the MediaBench [107] to evaluate both the exploration time and the quality of the results. For each benchmark, we explore different sizes in the number of accesses in the overall iteration space by increasing the loop bounds by a factor. The results are depicted in Tables 5.1 and 5.2 for MediaBench, Table 5.3 for PolyBench, Mibench benchmarks and for the demonstration case study of Chapter 4, which explores a large set of different pattern operations. From the experimental results, the proposed methodology achieves optimal results and the exploration time remains stable, as the loop bounds are increased, because only few parameters are changed, i.e. R, UB and IR, which do not affect the exploration time of the proposed methodology. The exploration time of enumerative approach is highly coupled with the number of accesses in the iteration space. The exploration times for the proposed methodology are quite close to each other for all the benchmarks, which is a very promising indication for the

Table 5.1: Comparison results for MediaBench. The '-' mark is used in the cases where Memory Error produced during simulation.

Algorithm: Array (init.bounds)	Bound Increase Factor	Proposed Methodology		Enumerative Appr.		Approx.
		Storage Size	Expl. Time (ms)	Storage Size	Expl. Time (ms)	Storage Size
Jpeg-Decode&Quant.: DCTblock (256)	1	7,680	0.148	7,680	34.667	10,239
	3	23,040	0.147	23,040	103.736	30,719
	5	38,400	0.149	38,400	162.533	51,199
	7	53,760	0.145	53,760	228.764	71,679
	9	69,120	0.147	69,120	281.893	92,159
	11	84,480	0.145	84,480	350.931	112,639
	13	99,840	0.145	99,840	413.799	133,119
Jpeg-Decode&Quant.: DCTblock (512)	1	15,360	0.146	15,360	65.987	20,476
	2	30,720	0.150	30,720	128.877	40,959
	3	46,080	0.147	46,080	212.997	61,439
	4	61,440	0.146	61,440	264.078	81,919
	5	76,800	0.149	76,800	318.667	102,399
	6	92,160	0.149	92,160	389.390	122,879
Epic: image (240)	1	4,257	1.313	4,257	101.116	11,520
	2	17,265	1.302	17,265	1,067.983	138,144
	3	34,609	1.301	34,609	3,888.289	522,016
	4	69,297	1.337	69,297	16,329.751	2,027,040
	5	138,673	1.367	138,673	64,824.346	7,986,208
	6	277,425	1.318	277,425	249,110.047	31,701,024
Mpeg: curr_frame (64x32)	1	34,833	1.801	34,833	852.955	65,537
	2	69,649	1.840	69,649	1,703.624	131,073
	3	139,281	1.806	139,281	3,317.909	262,145
	4	278,545	1.813	278,545	7,167.751	524,289
	5	557,073	1.818	557,073	30,241.290	2,097,185
Mpeg: curr_frame (128x32)	1	69,649	1.840	69,649	1,703.624	131,073
	2	139,281	1.806	139,281	3,317.909	262,145
	3	278,545	1.813	278,545	7,167.751	524,289
	4	557,073	1.818	557,073	30,241.290	2,097,185
Motion estimation-full pel: p1/p2 (32)	1	3,576	0.147	3,576	51.922	12,759
	2	11,512	0.146	11,512	232.104	65,991
	3	39,672	0.147	39,672	1,399.778	389,031
	4	145,144	0.147	145,144	9,093.884	2,589,543
	5	552,696	0.146	552,696	66,325.246	18,713,319
	6	2,154,232	0.147	-	-	141,892,071
Motion estimation-full pel: p1/p2 (80)	1	39,672	0.147	39,672	1,399.778	389,031
	2	145,144	0.147	145,144	9,093.884	2,589,543
	3	552,696	0.146	552,696	66,325.246	18,713,319
	4	2,154,232	0.147	-	-	141,892,071
Motion estimation-half pel: p1/p2 (272)	1	40,800	0.149	40,800	2,181.419	657,152
	2	79,200	0.151	79,200	8,034	2,492,160
	3	156,000	0.149	156,000	34,129.439	9,701,120
	4	309,600	0.149	309,600	133,794.569	38,274,816
	5	616,800	0.148	-	-	152,045,312

limited complexity of our approach. The cases where the time is increased is when more operations are required, e.g. increased number of patterns which are misaligned (benchmarks demonstration case study, epic, mpeg). The approximation approach describes an upper bound in the storage size which leads to high quality loss depending on the number of holes that are considered as accesses.

In Fig. 5.7 we compare the exploration times of the proposed and the enumerative approach, when the number of accesses is increased. Fig. 5.7 and Fig. 5.7(c) present the exploration times, when the overall accesses are increased due to increase of the loop bounds by a factor for benchmark pgp-outdec and blowfish-decode/encode, respectively. Fig. 5.7(b) and Fig. 5.7(d) present the exploration times, when the overall accesses are increased due to an increase in the number of patterns of one iteration of the application by a factor for benchmark pgp-outdec and blowfish-decode/encode, respectively. From the experimental results, it is verified that the exploration time

Table 5.2: Comparison results for MediaBench. The '-' mark is used in the cases where Memory Error produced during simulation.

Algorithm: Array (init.bounds)	Bound Increase Factor	Proposed Methodology		Enumerative Appr.		Approx.
		Storage Size	Expl. Time (ms)	Storage Size	Expl. Time (ms)	Storage Size
Motion estimation-half pel: p1/p2 (48)	1	7,200	0.151	7,200	69.523	19,231
	2	12,000	0.153	12,000	193.411	55,071
	3	21,600	0.155	21,600	615.794	182,047
	4	40,800	0.149	40,800	2,181.419	657,152
	5	79,200	0.151	79,200	8,034	2,492,160
	6	156,000	0.149	156,000	34,129.439	9,701,120
	7	309,600	0.149	309,600	133,794.569	38,274,816
	8	616,800	0.148	-	-	152,045,312
Motion estimation-half pel: p1a (272)	1	27,200	0.157	27,200	2,239.512	657,183
	2	52,800	0.154	52,800	8,325.536	2,492,191
	3	104,000	0.159	104,000	36,410.136	9,701,151
	4	206,400	0.156	206,400	129,134.210	38,274,847
	5	411,200	0.165	-	-	152,045,343
Motion estimation-half pel: p1a (48)	1	4,800	0.155	4,800	93.294	19,231
	2	8,000	0.158	8,000	192.457	55,071
	3	14,400	0.156	14,400	640.237	182,047
	4	27,200	0.157	27,200	2,239.512	657,183
	5	52,800	0.154	52,800	8,325.536	2,492,191
	6	104,000	0.159	104,000	36,410.136	9,701,151
	7	206,400	0.156	206,400	129,134.210	38,274,847
	8	411,200	0.165	-	-	152,045,343
Pgp-outdec: p (48)	1	3,072	0.143	3,072	11.911	3,074
	3	12,288	0.148	12,288	43.058	12,290
	6	196,608	0.149	196,608	683.534	196,610
	8	786,432	0.155	786,432	2,732.710	786,434
	11	6,291,456	0.146	6,291,456	21,838.798	6,291,458
	13	15,165,824	0.153	15,165,824	88,559.484	15,165,826
	16	134,217,728	0.150	-	-	134,217,730
Pgp-outdec: p (262,144)	1	4,194,304	0.146	4,194,304	21,838.798	6,291,458
	2	8,388,608	0.147	8,388,608	40,522.5	12,582,914
	3	16,777,216	0.153	16,777,216	88,559.484	25,165,826
	4	33,554,432	0.147	33,554,432	171,213.572	50,331,650
	5	67,108,864	0.150	-	-	100,663,298
Mesa-light: frontcolor & backcolor (10)	1	5	0.147	5	0.131	10
	10	50	0.145	50	0.410	10
	10^2	500	0.146	500	3.290	100
	10^3	5 K	0.149	5 K	34.657	1 K
	10^4	50 K	0.147	50 K	375.118	10 K
	10^5	500 K	0.152	500 K	3,492.222	100 K
Mesa-light: frontcolor & backcolor (10^2)	1	50	0.147	50	0.410	10^2
	10^4	$5*10^3$	0.149	$5*10^3$	34.657	10^4
	10^6	$5*10^5$	0.152	$5*10^5$	3,492.222	10^6
	10^8	$5*10^7$	0.151	$5*10^7$	338,464.414	10^9

of the enumerative approach is highly increased with the loops bounds, whereas the exploration time of the proposed approach remains stable (around 0.146 ms), as the patterns are not modified. When the number of patterns is increased, the exploration time of the enumerative approach is significantly increased, whereas the exploration time of the proposed technique is lower by a magnitude of 2. In realistic applications, the number of patterns in our approach remains quite low, as it is described by the number of condition and access statements in one iteration of the application.

Table 5.3: Comparison results for the PolyBench(*1), MiBench(*2) and the translations demonstration case study. The '-' mark is used in the cases where Memory Error produced during simulation.

Algorithm: Array (init.bounds)	Bound Increase Factor	Proposed Methodology		Enumerative Appr.		Approx.
		Storage Size	Expl. Time (ms)	Storage Size	Expl. Time (ms)	Storage Size
Correlation: data (32) *1	1	0.25 M	0.343	0.25 M	2,886.159	0.25 M
	2	1 M	0.340	1 M	11,732.231	1 M
	3	4 M	0.337	4 M	48,614.943	4 M
	4	16 M	0.347	16 M	188,547.256	16 M
	5	64 M	0.333	64 M	780,152.034	64 M
	6	256 M	0.345	-	-	256 M
Jacobi-1D: A (500) *1	1	1,004	0.674	1,004	5.428	1,004
	2	2,004	0.660	2,004	9.998	2,004
	3	4,004	0.676	4,004	21.646	4,004
	4	8,004	0.664	8,004	40.701	8,004
	5	16,004	0.656	16,004	78.056	16,004
	6	32,004	0.658	32,004	159.161	32,004
Blowfish Decode/Encode: p (2,048) *2	1	256	0.146	256	6.899	2,040
	4	1,024	0.147	1,024	29.753	8,184
	7	1,792	0.149	1,792	45.805	14,328
	10	2,560	0.148	2,560	66.517	20,472
	13	3,328	0.148	3,328	86.228	26,616
	16	4,096	0.147	4,096	105.264	32,760
	19	4,864	0.149	4,864	120.938	38,904
Blowfish Decode/Encode: p (4,096) *2	1	512	0.146	512	15.065	4,088
	2	1,024	0.147	1,024	29.753	8,184
	3	1,536	0.148	1,536	39.888	12,288
	4	2,048	0.148	2,048	51.536	16,376
	5	2,560	0.148	2,560	66.517	20,472
	6	3,072	0.148	3,072	79.722	24,568
	7	3,584	0.149	3,584	90.756	28,664
	8	4,096	0.147	4,096	105.264	32,760
	9	4,608	0.147	4,608	114.122	36,856
	10	5,120	0.146	5,120	132.905	40,952
Translation Demonstration case: A (64)	1	32	0.799	32	0.487	507
	3	128	0.804	128	1.694	2,043
	5	512	0.798	512	6.308	8,187
	7	2,048	0.810	2,048	25.776	32,763
	9	8,192	0.793	8,192	97.852	131,067
	11	32,768	0.796	32,768	395.624	524,283
	13	131,072	0.793	131,072	1,629.237	2,097,147

5.4 Step 3: Intra-signal in-place optimization for overlapping case

This section describes the cases of the intra-signal in-place step, which have been carefully selected to group those instances that resemble each other sufficiently to share the same in-place size equations, while they still do not lead to an explosion. In the next sections, we provide the fully analytical parametric templates per case, which provide a scalable solution to near-optimally compute the minimum required storage size, and illustrate the templates instantiations for different condition statements.

The minimum required storage size for the array depends on the relative position of the write and read access statements, which defines the lifetime of the array elements. The minimum storage size is the maximum of the number of elements that have been written, but not yet read, for the instances of write and read access statements in all values of the loop iterator, i.e. the iteration range (IR). As the methodology aims at computing the near-optimal case, in one iteration, we

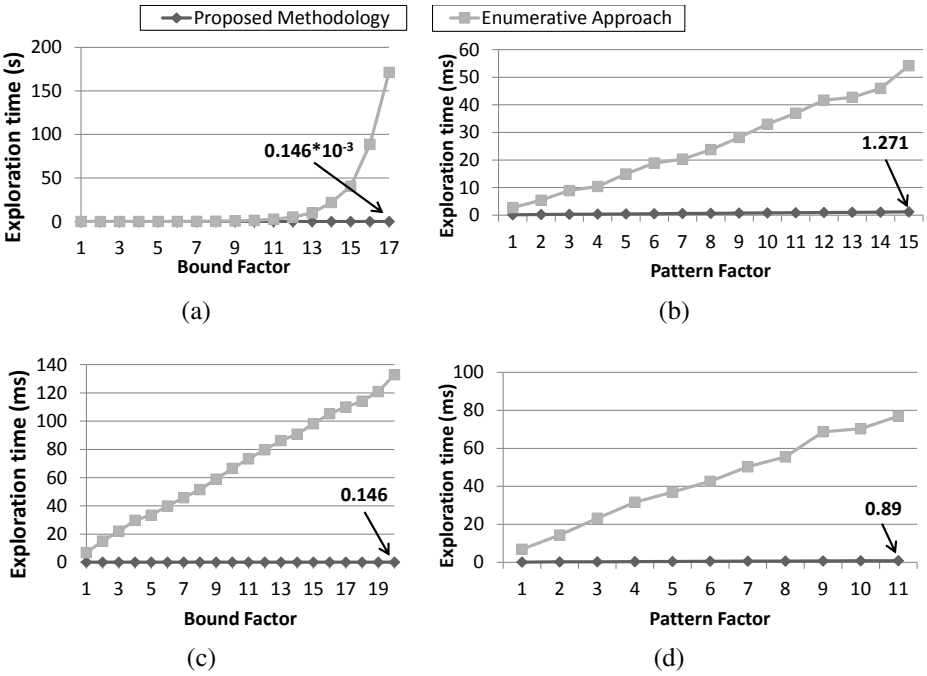


Figure 5.7: Exploration time comparison, when the number of accesses is increased due to an increase by: i) a factor over the loop bounds for pgp-outdec benchmark (a) and blowfish-decode/encode (c) and ii) a factor over the number of patterns in the application kernel for pgp-outdec benchmark (b) and blowfish-decode/encode (d).

first read the array element to free a memory position and then write the array element, whenever this is possible, i.e. the array element that is read is not the array element written in this iteration. To avoid the enumeration of the write and read instances in the iteration space, we use the relative position of the write and the read access statements, which defines the exploration window of the written, but not yet read array elements. The relative position is given from the $Diff_{\text{Iterator}}$.

5.4.1 Intra-signal in-place cases

We define the intra-signal in-place cases based on the loop structure, the size of the exploration window and the condition statements. The different cases in our carefully organized classification, which covers all the possible cases in the target domain, are depicted in Fig 5.8. The first split is between the *one loop dimension* and *several loop dimensions*. The one loop dimension case is divided into the *dominant segment case*, i.e. an accessed part of the iteration space is large enough (larger than the index difference) and thus defines the overall required size, and the *non-dominant segment*, i.e. all accessed parts of the iteration space are small and the size cannot be found by exploring only one part. In the several loop dimensions the computation is performed from the outer to the inner dimensions, by computing the partial outer size due to the outer index difference and adding the elements required to be stored due to the inner index difference. Hence, the split is between the cases of *dominant segment in outer dimension* and *non-dominant segment in outer dimension*. In the first case, the outer dimension has a large segment, which is solid, whereas in the second case, the sizes of the segments are too small to cover the index dif-

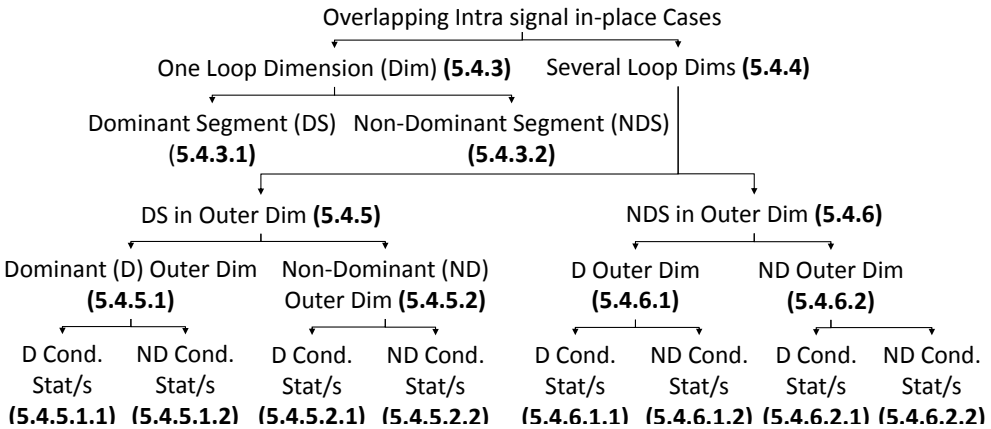


Figure 5.8: The overlapping intra-signal in-place cases.

ference of the outer dimension. The dominant segment case is further split into *dominant outer dimension* and the *non-dominant outer dimension*. The aforementioned split is essential, because in the dominant outer dimension case, the iteration after the exploration window is A, as the index difference is smaller than the dominant segment size. Because of the A in the I dimension, the J pattern is accessed. Hence, computing the additional written elements in the I iteration gives the maximum additional storage size that may exist. The condition statements and the J pattern defines the way to compute the additional written elements. However, some condition statements are dominant in the sense that they affect the additional elements, as they prohibit the A in the inner dimension, if no A exist in the outer dimension, as explained in Section 5.4.2. For instance, a condition I>5&&I<10&&J>2&&J<8, requires an A in the I dimension, i.e. an iterator above 5 and below 10, in order to access the 5 elements described in J dimension. Hence, a further split exists between the *Dominant Condition Statements* and the *Non-Dominant Condition Statements*. In the non-dominant outer dimension case, the iteration after the exploration window is H, because the size of the dominant segment is equal to the index difference. Hence, the conditions and the J pattern have to be explored to define the additional written and not-yet read elements due to the inner index difference. The split between the *Dominant Condition Statements* and the *Non-Dominant Condition Statements* also exists, as it simplifies the exploration process. The Dominant Condition Statements do not allow additional elements in the inner dimension and, thus no further exploration is required. The Non-Dominant Condition Statements require exploration to define the additional elements. Similar splits exist in the non-dominant segment in the outer dimension case.

5.4.2 Condition Statements

The set of all the condition statements, that may exist in the application domain, are described by the leaves and the combinations of the leaves in Fig. 5.9. The splits are refined based on the loop structure and the condition statements. The first split is between *one* and *several loop dimensions*. The one loop dimension may have *one condition* or several conditions in *condition expressions*. Based on the primitive condition the one condition is enumerative (due to one dimension) and it

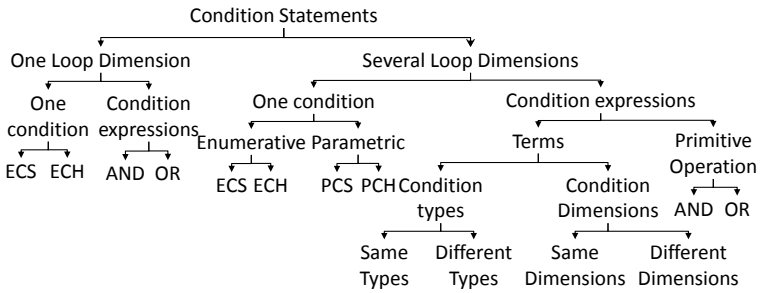


Figure 5.9: The cases for the condition statements of the target domain are derived by the leaves and their combinations.

can be *ECS* and/or *ECH*. The condition expression combines the conditions with *AND* and/or *OR* primitive operations. The several loop dimensions are refined to *one condition* and *condition expressions*. Due to the existence of several dimensions, both *enumerative* and *parametric* conditions may exist in the application. The parametric ones are *PCS* and/or *PCH*. The condition expressions are split into the *terms* and the *primitive operation*. The terms are refined into the *condition types* and the *condition dimensions*. The condition types can be of *same type* or of *different type*. The same split applies to condition dimensions.

We cluster the aforementioned condition cases into the six representative condition cases depicted in Table 5.4, which are used to illustrate the proposed methodology through the paper. We focus on ECS, ECH and PCH conditions, which are considered as representative. The ECH and PCH describe iteration spaces with holes, which is crucial for the target domain. The ECS and the PCS both describe solid spaces and thus are similar in nature. We present the ECS conditions in this paper. The PCS can be computed in a similar way to ECS, since the worst case size in the PCS is the ECS described by the loop bounds. The process and an illustrative example is given in Section 5.4.8. In Table 5.4 the condition cases are: i) It describes the primitive enumerative conditions ECS, ECH and the combination of ECS and/or ECH with AND primitive operation for same dimensions. ii) This case describes the primitive parametric conditions PCH of $==$, which is representative for the PCH of $<\&\&>$ type, as only the SID is increased. The case ii is representative for the cases: 1) PCH of $==$ type combined with OR for same dimensions and same PS, 2) PCH of $==$ type combined with AND with PCH, ECH, ECS for same dimensions, 3) PCH of at least one $==$ type combined with AND for different dimensions. In these cases, the result of the operation is a pruned/modified PCH of $==$ type, which determines the size. iii) The condition case iii describes the enumerative conditions ECS, ECH with OR primitive operation with the PCH of $==$ type for same dimensions. iv) It describes the primitive parametric conditions PCH of $==$ combined with OR for same dimensions and different PS. The condition statements i, ii, iii and iv belong to the dominant condition statements case of Fig. 5.8, because they do not allow accesses in the inner dimension, if no A exist in the outer dimension. For instance, the condition statement $i \geq 5 \&\& j > 8$, does not allow accesses in J pattern to occur, when the I dimension has holes, i.e. the iterator is lower than 5. v) The condition case v describes the combination of ECS and/or ECH with OR primitive operation. vi) The condition case vi describes the primitive parametric condi-

Table 5.4: Representative condition cases.

Cond. cases	Representative description
i	ECS,ECH (AND)
ii	PCH of $==$ type
iii	ECS,ECH (OR) PCH of $==$ type same dimensions
iv	PCH of $==$ type (OR) for same dimensions, different PS
v	ECH/ECS (OR)
vi	PCH of $\neq$ type

tions PCH of $\neq$ type and it is representative for the cases: 1) PCH of $\neq$ type combined with AND for same dimensions, 2) At least one PCH of $\neq$ type and PCH, ECH, ECS combined with OR for same dimensions, 3) PCH of $\neq$ type and ECH/ECS combined with AND for same dimensions. The result of the operation leads to a PCH of $\neq$ type, which defines the storage size. The condition statements v and vi belong to the non-dominant condition statements case of Fig. 5.8, because accesses may exist in the inner dimension, even if a H exist in the outer dimension. For instance, the condition statement i>5||j>8, does not prohibit accesses in J pattern, when the I dimension has holes. The solution for the combination of conditions in different dimensions is derived by combining the solutions of the representative cases, as illustrated Section 5.4.7.

5.4.3 One loop dimension

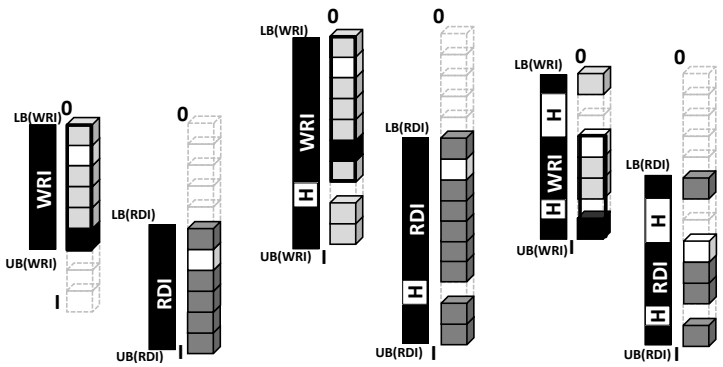
The one loop dimension cases are split into the Dominant Segment and the Non-Dominant Segment cases. The dominant segment case is valid when the exploration window, i.e. the distance between the WR and RD index expressions, is smaller than or equal to the size of the SIS or the maximum segment with accesses in ISH: $Diff_{\mathrm{I}} \leq ID(I))||Diff_{\mathrm{I}} \leq max(\sum_{i=LB_{\mathrm{I}}}^{UB_{\mathrm{I}}} SID(i))$, where ID is the iteration domain, i.e. the UB-LB-1 for SIS. The analysis of only the dominant segment is enough to define the minimum storage size, as a single rectangular in the iteration space is sufficient to describe the lifetime of the elements. The non-dominant segment is defined when the exploration window is larger than the maximum segment size. In this case, the storage size is derived from exploration of the segments inside the exploration window.

5.4.3.1 Dominant Segment

The storage size is defined by the size of the exploration window, since the dominant SID has only A, i.e. Eq. 5.25.

$$Size_{\mathrm{I}} = Diff_{\mathrm{I}} \tag{5.25}$$

The dominant segment case for SIS is schematically depicted in Fig. 5.10(a). The gray area in WRI describes both the iterations where the elements are written and the written elements, since the WR index expression is I. The dark gray area in RDI describes the iterations where the elements are read. The black line shows the dominant segment. One instance of the storage size in a specific iteration is depicted in the WRI of Fig. 5.10(a). The size is given by the cells between the element which is written in this iteration (black cell) and the element which is read in the next



(a) (b) (c)

```
For (I=0;I<11;I++)
 If (I<6)
  B[I]=...
 If (4<I<11)
  ...=B[I-5]
```

(d)

```
For (I=0;I<10;I++)
 If (I<7)||(I>7)
  B[I]=...
For (I=5;I<15;I++)
 If (I<12)||(I>12)
  ...=B[I-5]
```

(e)

```
For (I=0;I<11;I++)
 If (I==0)||(2<I<6)||(I==7)
  B[I]=...
 If (I==5)||(7<I<11)||(I==12)
  ...=B[I-5]
```

(f)

Figure 5.10: Iteration spaces of WR (gray cells in WRI) and RD (dark gray cells in RDI) for I dimension. The 0 to I indicates the direction of increasing the iterator. For one iteration instance, the WRI black cell is the written element and the WRI white cell is the read element in next RD iteration (white cell in RDI). The black line shows the dominant segment/pattern section. The examples are: (a) Dominant segment & SIS, (b) Dominant Segment & ISH and (c) Non-Dominant segment & ISH. The application codes are (d), (e) and (f), respectively.

iteration (white cell). The white cell in RDI shows the next read iteration. In Fig. 5.10(a) the storage instance is given for the 6th iteration. The element B[5] is written, the element B[1] is read in the next iteration and maximum 5 elements are required to be stored. The corresponding code is depicted in Fig. 5.10(d). The RD pattern which describes the read iterations is {6A}, LB=4, UB=11, IR=6, PS=6 and R=1. The RD pattern, which describes the elements, is derived by shifting left 5 positions, due to the -5 of the RD index expression. The result is {6A}, LB=-1, UB=6, IR=6, PS=6 and R=1, which is identical to the WR pattern, as the application writes only the elements that are read. The final storage size is 5. The dominant segment case for ISH is schematically depicted in Fig. 5.10(b). The instance is given in the 6th iteration, where the B[5] is written (black cell in WRI) and the B[1] (white cell in WRI) is read in the next iteration (white cell in RDI). The corresponding code is in Fig. 5.10(e). The RD pattern which describes the elements is {7A 1H 2A}, LB=-1, UB=10, IR=10, PS=10 and R=1. The index difference is 5 and thus, the storage size is 5.

5.4.3.2 Non-dominant Segment

This case is valid when the maximum segment has smaller size than the exploration window. As no dominant segment exists, the storage size is given by the pattern section with the maximum number of elements that are accessed in the exploration window. The exploration for the section

ALGORITHM 8: Selection of pattern section.

```
max=0; Part=0; Start=0
for (k=LB_I+1; k< UB_I-Diff_I; k++) do
    Size=0
    if (k==Start+ID(part)) then
        Start+=ID(part)
        Offset=0
    else Offset++
    Length=0
    while (Length+(ID(Part)-Offset) <= Diff_I) do
        if (PT(Part)==A) then Size+=SID(Part)
        Length+=ID(Part)-Offset
    Rem=Diff_I-Length+(ID(Part)-Offset)
    if (PT(Part)==A) then Size+=Rem
    if (Size>max) then max=Size
```

is scalable, as it is performed only in the pattern and in sections of size $Diff_{\mathrm{I}}$. Per section, the SIDs are added. The first pattern section with the maximum number of A is used to define the storage size. The storage size is computed by the maximum sum of SIDs in the pattern section, i.e. $MaxAInIndex$. The size is described by Eq. 5.26 and the process is described in Alg. 8. The schematic illustration of the non-dominant segment case is depicted in Fig. 5.10(c). The application code is in Fig. 5.10(f). The read pattern is {1A 2H 3A 1H 1A} with LB=-1, UB=8, IR=8, PS=8 and R=1. The index difference is 5, the 1st SID is 1, the 2nd SID is 3 and the 3rd SID is 1. The size is 4.

$$\begin{aligned} &\text{For}(k = LB_{\mathrm{I}} + 1; k < UB_{\mathrm{I}} - Diff_{\mathrm{I}}; k++)\{ \\ &\quad Size_{\mathrm{I,curr}} = \sum_{l=k}^{Diff_{\mathrm{I}}} SID_{\mathrm{I}}(l) \\ &\quad \text{if } (Size_{\mathrm{I}} < Size_{\mathrm{I,curr}}) : Size_{\mathrm{I}} = Size_{\mathrm{I,curr}}\} \end{aligned} \tag{5.26}$$

5.4.4 Extension to Several loop dimensions

When several loop dimensions exist, the intra-signal in-place is given by following the loop ordering and computing the partial storage size of the outer dimension and propagate the result to the inner dimensions to potentially add extra storage size. We assume that I-J-K is the order of loop dimensions throughout the description of the different cases.

In the two dimensions, the storage size is given by the partial storage size of outer dimension $PSize_{\mathrm{I,J}}$ created by the I index difference and taking into account the conditions in I and J dimension and the additional size $ASize_{\mathrm{J}}(J)$ due to J index difference, taking into account the conditions in J. The $PSize_{\mathrm{I,J}}$ is the elements that are written in the outer dimension, $Writes_{\mathrm{I}}$, multiplied by the size of the inner dimension, $Size_{\mathrm{J}}$. The $Writes_{\mathrm{I}}$ for the two dimensions case are computed using the equations of one loop dimension in Section 5.4.3. The size of the inner dimension is depends on the conditions and the operations that combine the conditions. The $ASize_{\mathrm{J}}(J)$ is the maximum between the elements that have been additionally written and the elements that have not yet been read due to the J index difference. The additional written elements are defined by the writes in the next WR iteration of I dimension of the exploration window. The additional not yet read elements are defined by the elements stored, but not yet read, before the

RD exploration window, i.e. the last WR iteration of the I dimension of the exploration window. The general equation is Eq. 5.27. In one loop dimension case, Eq. 5.27 is simplified into $Writes_{\mathrm{I}}$, which describes the accesses in the outer dimension, since no inner dimension exist, i.e. $Size_{\mathrm{J}}$=1 and $ASize_{\mathrm{J}}(J)$=0. For more than two dimensions, the Eq. 5.27 is refined and iteratively applied, as described in Eq. 5.28 for three dimensions. Initially, the storage size is computed for the two outer dimensions I and J, as described in the next sections. The result is used in the position of the $Size_{\mathrm{I,J}}$ for the computation of the size of the next dimension. The additional size is given by the additional elements due to the K index difference and depends on the type of operations which combine the conditions. Due to page limitation, we focus on the two dimensions cases for all the intra-signal in-place cases and the condition statement cases of Table 5.4. We illustrate the extension to more dimensions and provide representative examples in Section 5.4.7.

$$\begin{aligned} Size_{\mathrm{I,J}} &= PSize_{I,J} + ASize_{J}(J) \\ &= Writes_{\mathrm{I}} * Size_{\mathrm{J}} + max(Writes_{\mathrm{J}}, Reads_{\mathrm{J}}) \end{aligned} \tag{5.27}$$

$$Size_{\mathrm{I,J,K}} = Size_{\mathrm{I,J}} * Size_{k} + \sum_{k=J}^{K} ASize_{\mathrm{k}}(k) \tag{5.28}$$

5.4.5 Dominant Segment in Outer Dimension

This case is valid when the size of the SIS or the maximum segment of ISH of the outer dimension is larger than or equal to the index difference of the outer iteration. Due to the dominant segment, the exploration window has only A and the writes in I are given by index difference, as described in Eq. 5.25. The size, the additional writes and reads of J dimension depend on the dominant segment size, the index difference, the type of operations, as explained in the next paragraphs.

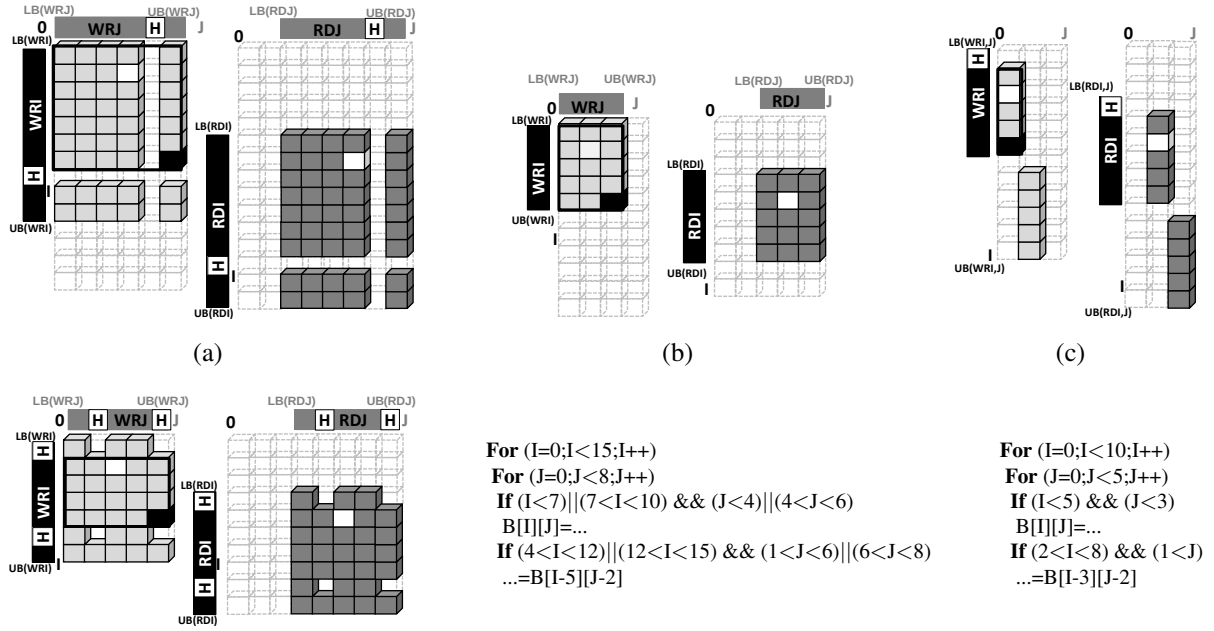
5.4.5.1 Dominant Outer Dimension

This case is valid when the dominant segment has larger size than the index difference. Hence, the next WR iteration after the exploration window is always A and the J pattern is always accessed. The additional writes will always cover or be equal to the not yet read elements. Hence, Eq. 5.27 is simplified to Eq. 5.29. In the next paragraphs, we illustrate the instantiation of Eq. 5.29 on the different condition cases of Table 5.4.

$$Size_{\mathrm{I,J}} = Writes_{\mathrm{I}} * Size_{\mathrm{J}} + Writes_{\mathrm{J}} \tag{5.29}$$

5.4.5.1.1 Dominant Condition Statements

Cond. Case i: The size of the J dimension is given by the summation of A in the J pattern due to the AND operation. The additional writes are the SIDs of the J pattern inside the exploration window of the J dimension. When the index difference is larger than the $UB(J)_{\mathrm{WR}}$, the accumu-




```
For (I=0;I<15;I++)
 For (J=0;J<8;J++)
  If (I<7)||(7<I<10) && (J<4)||(4<J<6)
   B[I][J]=...
  If (4<I<12)||(12<I<15) && (1<J<6)||(6<J<8)
   ...=B[I-5][J-2]
```

```
For (I=0;I<10;I++)
 For (J=0;J<5;J++)
  If (I<5) && (J<3)
   B[I][J]=...
  If (2<I<8) && (1<J)
   ...=B[I-3][J-2]
```

```
For (I=0;I<15;I++)
 For (J=0;J<3;J++)
  If (I>6J)&&(I>6(J+1))&&(J<2)
   B[I][J]=...
  If (J>0)&&(I>6J-3)&&(I>6(J+1)-3)
   ...=B[I-3][J-1]
```

```
For (I=0;I<8;I++)
 For (J=0;J<5;J++)
  If (0<I<5)||(I==6)||(J==0)||(1<J<4)
   B[I][J]=...
For (I=2;I<10;I++)
For (J=2;J<8;J++)
 If (3<I<8)||(I==9)&&(J==3)||(4<J<7)
  ...=B[I-3][J-3]
```

Figure 5.11: Iteration spaces for I and J dimension for the Dominant Segment in Outer Dimension & Dominant Outer Dimension for: (a) ECH/ECS combined with AND for ISH, (b) ECH/ECS combined with AND for SIS, (c) PCH of $<$ && $>$ type and (d) ECH/ECS combined with OR. The corresponding application codes are in (e), (f), (h) and (g), respectively.

lation reaches the upper bound of the pattern. The result of the instantiation of Eq. 5.29 is given in Eq. 5.30. Fig. 5.11(a) schematically illustrates the WR and the RD patterns of two iterators I and J, where J describes ISH. The dominant segment is marked by the black line. The code is depicted in Fig. 5.11(e). The RD pattern of I is {7A 1H 2A}, LB=-1, UB=10, IR=10, PS=10 and R=1 and the RD pattern of J is {4A 1H 1A}, LB=-1, UB=6, IR=6, PS=6 and R=1. The index difference is 5 in I dimension and 2 in J dimension. The final storage size is 27, i.e. 5*(4+1)+2. When the J dimension is SIS, the Eq. 5.30 can be simplified to the Eq. 5.31 by replacing the sums with the ID(J). Fig. 5.11(b) gives an example of SIS. One instance of the size is the cells between the black cell (element that is written) and the white cell (element that is read) in WRI. The dominant segment is marked by the black line. The code is depicted in Fig. 5.11(f). The RD pattern of I is {5A}, LB=-1, UB=5, IR=5, PS=5 and R=1 and the RD pattern of J is {3A}, LB=-1, UB=3, IR=3, PS=3 and R=1. The index difference is 3 in I dimension and 2 in J dimension. The storage size is 11, i.e. 3*3+min(2,3).

$$Size_{\mathrm{I,J}} = Diff_{\mathrm{I}} * \sum_{k=LB_{\mathrm{J}}}^{UB_{\mathrm{J}}} SID_{\mathrm{J}}(k) + \sum_{k=LB_{\mathrm{J}}}^{min(Diff_{\mathrm{J}},UB_{\mathrm{J}})} SID_{\mathrm{J}}(k) \tag{5.30}$$

$$Size_{\mathrm{I,J}} = Diff_{\mathrm{I}} * ID(J) + min(Diff_{\mathrm{J}}, ID(J)) \tag{5.31}$$

Cond. Case ii: The size of the J dimension is 1. The additional elements are computed based on $case_{\mathrm{PCH}_{==,\mathrm{A},\mathrm{I}-\mathrm{J}}}$ in Table 5.5. The $\frac{Diff_{\mathrm{I}}}{PS(PCH)}$ shows the repetition of the PCH in the outer index difference. When the inner index difference is larger than $\frac{Diff_{\mathrm{I}}}{PS(PCH)}$, an additional element requires to be stored. The storage size is given by Eq. 5.32. In the case of different dimensions with AND, an additional element is required to be stored when at least one of the inner index differences is larger than $\frac{Diff_{\mathrm{I}}}{PS(PCH)}$ and the PCH is the combined pattern. Fig. 5.11(c) illustrates the WR and the RD patterns of PCH $<$ && $>$ type and the corresponding code is in Fig. 5.11(g). The RD pattern of I is {1H 5A}, LB=-1, UB=12, IR=12, PS=6 and R=2. The dominant segment is marked by the black line. The index difference is 3 for I dimension and 1 for J dimension. The final storage size is 4, i.e. 3*1+1.

$$Size_{\mathrm{I,J}} = Diff_{\mathrm{I}} + case_{\mathrm{PCH}_{==,\mathrm{A},\mathrm{I}-\mathrm{J}}} \tag{5.32}$$

Cond. Case iii: The size derives from combination of the condition case i and case ii. Case i computes the size due to the ECH condition. The result of Eq. 5.30 is increased by the elements accessed due to the PCH, i.e. the A of the PCH in the H of the ECH. The A of PCH in the A of ECH have been already computed by equation of case i. As the PCH has a dominant segment in the outer dimension, we find the holes that exist in the ECH, i.e. $MaxHInLen_J$, for length equal to $L = \frac{Diff_{\mathrm{I}}}{PS(PCH)} + 1$. The number of holes are multiplied by the SID of the PCH pattern up to $Diff_{\mathrm{I}}$. We also add the $case_{\mathrm{PCH}_{==,\mathrm{A},\mathrm{I}-\mathrm{J}}}$. The result is Eq. 5.33.

$$\begin{aligned} Size_{\mathrm{I,J}} = Diff_{\mathrm{I}} * \sum_{k=LB_{\mathrm{J}}}^{UB_{\mathrm{J}}} SID_{\mathrm{J}}(k) + \sum_{k=LB_{\mathrm{J}}}^{min(Diff_{\mathrm{J}},UB_{\mathrm{J}})} SID_{\mathrm{J}}(k) \\ + MaxHInLen_J * \sum_{k=LB_{\mathrm{I}}}^{Diff_{\mathrm{I}}} SID_{\mathrm{PCH}}(k) + case_{\mathrm{PCH}_{==,\mathrm{A},\mathrm{I}-\mathrm{J}}} \end{aligned} \tag{5.33}$$

Cond. Case iv: The size derives from combination of the condition case ii for two PCH. The size of the larger PCH pattern is given by Eq. 5.32. The result is increased by the written elements of the smaller PCH due to the J index difference, which are the elements of the larger pattern that are also accessed in the second pattern, i.e. Access to Access (A2A). For the A2A elements in the dominant segment, we check the RD iteration in the Combined Pattern (CP). If the RD iteration is H, an extra element is required to be stored due to the A in the second PCH. If the RD iteration is A, an extra element is required to be stored only when the J index difference is larger than one. The last term exists when the repetition factor of the CP is larger than one. The result is Eq. 5.34.

Table 5.5: Computation of the $cases$ in intra-signal in-place equations.

Conditions	Case Value
$case_{\text{PCH}_{==,\text{A},\text{I}-\text{J}}}$	
$\frac{Diff_{\text{I}}}{PS(PCH)} \geq Diff_{\text{J}}$	0
$\frac{Diff_{\text{I}}}{PS(PCH)} < Diff_{\text{J}}$	1
$case_{\text{PCH}_{==,\text{OR}}}$	
$(RD(CP) == H)$ || $(RD(CP) == A$ && $Diff_{\text{J}} > 1)$	1
$case_{\text{WR}}$	
$Diff_{\text{J}} \leq LB(J)_{\text{WR}} + 1$	0
$Diff_{\text{J}} > LB(J)_{\text{WR}} + 1$ && $Diff_{\text{J}} < UB(J)_{\text{WR}}$	MaxALen($Diff_{\text{J}}$-$LB(J)_{\text{WR}}$-1)
$Diff_{\text{J}} \geq UB(J)_{\text{WR}}$	$\sum_{k=LB_{\text{I}}}^{UB_{\text{I}}} SID_{\text{J}}(k)$
$case_{\text{RD,A}}$	
$UB(J)_{\text{RD}} \leq IR(J)$	0
$LB(J)_{\text{RD}} + 1 < IR(J)$ && $UB(J)_{\text{RD}} > IR(J)$	MaxAPos($IR(J)$-$LB(J)_{\text{RD}}$-1)
$LB(J)_{\text{RD}} + 1 \geq IR(J)$	$\sum_{k=LB_{\text{I}}}^{UB_{\text{I}}} SID_{\text{J}}(k)$
$case_{\text{RD,H}}$	
$UB(J)_{\text{RD}} \leq UB(J)_{\text{WR}}$	0
$LB(J)_{\text{RD}} + 1 < UB(J)_{\text{WR}}$ && $UB(J)_{\text{RD}} > UB(J)_{\text{WR}}$	MaxAPos($UB(J)_{\text{WR}}$-$LB(J)_{\text{RD}}$-1)
$LB(J)_{\text{RD}} + 1 \geq UB(J)_{\text{WR}}$	$\sum_{k=LB_{\text{I}}}^{UB_{\text{I}}} SID_{\text{J}}(k)$
$case_{\text{PCH}_{\neq,\text{H}}}$	
$Diff_{\text{J}} \leq IR(J) - 1$	0
$Diff_{\text{J}} > IR(J) - 1$	-1

$$Size_{\text{I,J}} = Diff_{\text{I}} + case_{\text{PCH}_{==,\text{A},\text{I}-\text{J}}} + \sum_{k=LB_{\text{CP}}}^{A2A} case_{\text{PCH}_{==,\text{OR}}} \tag{5.34}$$

5.4.5.1.2 Non-Dominant Condition Statements

Cond. Case v: The size of the J dimension is the J values, i.e. IR(J), due to the OR primitive operation type. In contrast to case i, whether J dimension describes SIS or ISH does not affect the J size. The additional stored elements are defined by J index difference and IR(J), because the next WR iteration after the exploration window is A and the complete J dimension is accessed due to the OR operation. The size is given by Eq. 5.35. An example is depicted in Fig. 5.11(d) and the code in Fig. 5.11(h). The RD pattern of I for the elements is {1H 4A 1H 1A}, LB=-1, UB=7, IR=7, PS=7 and R=1 and the RD pattern of J is {1A 1H 2A 1H}, LB=-1, UB=5, IR=5, PS=5 and R=1. The index difference is 3 in I dimension and 3 in J dimension. The dominant segment is described by the black line. The final storage size is 18, i.e. 3*5+min(3,5).

$$Size_{\text{I,J}} = Diff_{\text{I}} * IR(J) + min(Diff_{\text{J}}, IR(J)) \tag{5.35}$$

Cond. Case vi: The size of the J dimension is IR(J) due to the $\neq$ type of condition. The size is given by Eq. 5.36. The equation is same to Eq. 5.35, since in both cases the additional writes are given by the J index difference, because of the A in the next WR iteration (Dominant Outer Dimension case). However, we consider the $\neq$ type as a different case, because when the outer dimension is not dominant, the additional elements are computed by the conditions type, as depicted in Section 5.4.5.2.2. In the non-dominant segment case in Section 5.4.6.1.2, the type of

Table 5.6: Dominant Segment in Outer Dimension, Non-Dominant Outer Dimension case & Dominant Condition Statements.

Case	Equation for storage size
i	$Diff_{\text{I}} * \sum_{k=LB_{\text{J}}}^{UB_{\text{J}}} SID_{\text{J}}(k)$
ii	$Diff_{\text{I}}$
iii	$Diff_{\text{I}} * \sum_{k=LB_{\text{J}}}^{UB_{\text{J}}} SID_{\text{J}}(k) + MaxHInLen_J * \sum_{k=LB_{\text{I}}}^{Diff_{\text{I}}} SID_{\text{PCH}}(k)$
iv	$Diff_{\text{I}} + \sum_{k=LB_{\text{CP}}}^{A2A} case_{\text{PCH}==,\text{OR}}$

the condition also affects the size of J, i.e. IR(J) for an A in the I dimension and IR(J)-1 for a H in the I dimension. In the case of PCH of $\neq$ type and ECH/ECS combined with AND for same dimensions, the size of J is the sum of SIDs in the J pattern and the additional elements are the sum of SIDs in J pattern up to J index difference, as the AND operation prunes accesses in I but not in J dimension.

$$Size_{\text{I,J}} = Diff_{\text{I}} * IR(J) + min(Diff_{\text{J}}, IR(J)) \tag{5.36}$$

5.4.5.2 Non-Dominant Outer Dimension

In this case, the dominant segment has size equal to the exploration window of the outer dimension. The next WR iteration after the exploration window is always H. Further exploration is required to define the additional written and not yet read elements due to the J index difference. The size is given by Eq. 5.27. The solutions for the partial outer storage size $PSize_{I,J}$ are the same with Section 5.4.5.1. However, the additional size $ASize_J(J)$ depends on the position of the dominant segment, the type of conditions and primitive operations, as explained in the next sections.

5.4.5.2.1 Dominant Condition Statements As the next WR iteration in I dimension is always H, no access is allowed in the J dimension, due to the dominant conditions. Hence, the terms that describe the additional WR elements in Section 5.4.5.1 are pruned and the results are depicted in Table 5.6. For instance, the term $\sum_{k=LB_{\text{J}}}^{min(Diff_{\text{J}},UB_{\text{J}})} SID_{\text{J}}(k)$ of Eq. 5.30 is pruned in case i of Table 5.6, since no additional elements are accessed in the J dimension. The other cases are derived in a similar way.

5.4.5.2.2 Non-dominant Condition Statements The next WR iteration in I has H and the conditions are non-dominant, additional elements exist due to the J index difference.

Cond. Case v: The position of the dominant segment defines if not yet read elements exist before the RD dominant segment, i.e. in the last I iteration of the WR dominant segment.

a. Dominant segment is in the 1st iteration. The first RD is just executed after the I index difference, so not yet read elements do not exist. Hence, Eq. 5.27 is simplified to Eq. 5.29, which is instantiated to Eq. 5.37, where $case_{\text{WR}}$ describes the additional written elements due to the J dimension, i.e. $Writes_{\text{J}}$. The computation of $case_{\text{WR}}$ is described in Table 5.5. If the J index difference is lower than the first WR element, i.e. $LB(J)_{\text{WR}}+1$ (row 1 in $case_{\text{WR}}$ of Table 5.5), no

additional elements have been writen. If the index difference is larger than the last stored element, i.e. $UB(J)_{\text{WR}}$ (row 3 in $case_{\text{WR}}$ of Table 5.5), all the SIDs in the J pattern have been written. Otherwise the J index difference resides between the lower and the upper bound (row 2 in $case_{\text{WR}}$ of Table 5.5). Then, the additional written elements are given by the summation of SIDs in the J pattern for length equal to the $Diff_{\text{J}} - LB(J)_{\text{WR}} - 1$. An example is depicted in Fig. 5.12(a). Due to space limitations, we do not present the codes in the rest of the paper, as they can be derived from the figures. The RD pattern of I for the elements is {4A 2H}, LB=-1, UB=6, IR=6, PS=6 and R=1, the RD pattern of J is {1A 1H 2A}, LB=-1, UB=4, IR=4, PS=4 and R=1. The index difference is 4 in I dimension and 2 in J dimension. The size is 21, i.e. 4*5+1, as 2>0&&2<4 and 1 A exists in J pattern for length equal to 2.

$$Size_{\text{I,J}} = Diff_{\text{I}} * IR(J) + case_{\text{WR}} \tag{5.37}$$

b. Dominant segment is not in the 1st iteration. In this case, RDs exist before the RD dominant segment, so not yet read elements exist. Due to the dominant segment in outer dimension case, the last WR iteration in I, before the RD dominant segment, is always A. Eq. 5.27 computes the size, which is instantiated into Eq. 5.38. Eq. 5.37 derives from Eq. 5.38 by simplifying the $max(case_{\text{WR}}, case_{\text{RD,A}})$ into $case_{\text{WR}}$, as no yet reads exist. The $case_{\text{WR}}$ gives the additional written elements in J dimension and it is computed by Table 5.5. The $case_{\text{RD,A}}$ describes the additional not yet read elements in J dimension and it is computed by Table 5.5. If the last read element is lower than or equal to the IR(J) (row 1 in $case_{\text{RD,A}}$ of Table 5.5), all elements of J pattern have been read. If the first read element is lower than the IR(J) and the last read element is larger than the IR(J) (row 2 in $case_{\text{RD,A}}$ of Table 5.5), the not yet read elements are given by the summation of SIDs in J pattern starting at the position $IR(J) - LB(J)_{RD} - 1$. Otherwise the first read element is larger than the IR(J) (row 3 in $case_{\text{RD,A}}$ of Table 5.5) and the not yet read elements are given by the summation of the SIDs in the complete J pattern. An example is depicted in Fig. 5.12(b). The RD pattern of I for the elements is {2H 4A 2H}, LB=-1, UB=8, IR=8, PS=8 and R=1, the RD pattern of J is {1A 1H 2A}, LB=-1, UB=4, IR=4, PS=4 and R=1. The index difference is 4 in I dimension and 1 in J dimension. The size is 21, i.e. 4*5+1, as $case_{\text{WR}}$=1 (1>0&&1<4 and 1 A in length 1) and $case_{\text{RD,A}}$=0 (4<5).

$$Size_{\text{I,J}} = Diff_{\text{I}} * IR(J) + max(case_{\text{WR}}, case_{\text{RD,A}}) \tag{5.38}$$

Cond. Case vi: The size is given by Eq. 5.39. The $case_{\text{PCH}_{\neq,\text{H}}}$ is used to describe the case in which the J index difference is smaller or equal to the parts created by the hole and, thus, it defines the additional elements (row 0 in $case_{\text{PCH}_{\neq,\text{H}}}$). Otherwise the hole should be subtracted from the overall size (row 1 in $case_{\text{PCH}_{\neq,\text{H}}}$). When an additional J pattern exists, e.g. due to combination with an ECH, we verify if the hole of PCH in the next WR iteration still exists or it is filled with an A due to the ECH. The position of the PCH hole in the J dimensions for the next WR iteration is given by $\frac{Diff_{\text{I}}}{PS(PCH)}$. When the ECH is combined with AND, the IR(J) is replaced by the sum of the

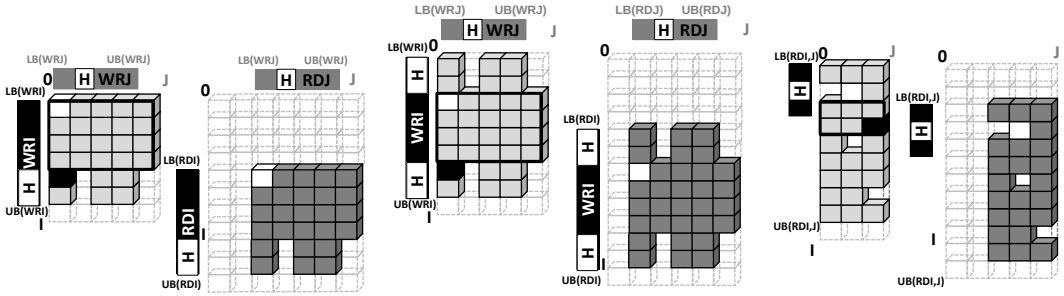


Figure 5.12: Iteration spaces for I and J dimensions for the Dominant Segment in Outer Dimension & Non-Dominant Outer Dimension for (a) ECS/ECH combined with OR at 1st iteration, (b) ECS/ECH combined with OR & not in the 1st iteration and (c) PCH $\neq$.

SIDs in the J pattern and the $Diff_{\mathrm{J}}$ by the sum of SIDs in the J pattern for length equal to $Diff_{\mathrm{J}}$.

$$Size_{\mathrm{I,J}} = Diff_{\mathrm{I}} * IR(J) + min(Diff_{\mathrm{I}}, IR(J)) + case_{\mathrm{PCH}_{\neq,\mathrm{H}}} \tag{5.39}$$

5.4.6 Non-Dominant Segment in Outer Dimension

In this case the size of the SIS or the maximum segment of ISH is smaller than the index difference of the outer iteration. The $Writes_I$ is given by the summation of SIDs in the selected pattern section, i.e. $MaxAInIndex$, as explained in Section 5.4.3.2, and the sum of HIDs in the selected pattern, i.e. $MaxHInIndex$, depending of the type of conditions. The J size and the additional elements due to the J index difference depend on the iteration space shape, the type of conditions and primitive operations, as explained in the next sections.

5.4.6.1 Dominant Outer Dimension

In this case, the next WR iteration after the selected pattern section is always A. This case is similar to dominant segment in outer dimension case in Section 5.4.5.1, but the $Writes_I$ are defined by accesses and the holes in the exploration window of the outer dimension and the condition cases.

5.4.6.1.1 Dominant Condition Statement Due to the dominant conditions, when a H is in the I dimension, the J dimension is not accessed. For instance, in Fig. 5.13(a), when I is 6, which is H, the J dimension is not accessed. Hence, the $Writes_I$ are given by the $MaxAInIndex$. The equations are similar to Section 5.4.5.1.1, but with $MaxAInIndex$ instead of $Diff_{\mathrm{I}}$, as depicted in the top part of Table 5.7. An example for case i is depicted in Fig. 5.13(a). The RD pattern of I is {1A 3H 2A 1H 3A}, LB=-1, UB=10, IR=10, PS=10 and R=1 and the RD pattern of J is {2A 1H 1A}, LB=-1, UB=4, IR=4, PS=4 and R=1. The selected pattern section is marked with the black line. The index difference is 5 and 2 for the I and the J dimension, respectively. The size is 14, i.e. (2+2)*(2+1)+2.

Table 5.7: Non-Dominant Segment in Outer Dimension case & Dominant Condition Statements.

Case	Equation for Storage Size
Dominant Outer Dimension (5.4.6.1)	
i	$MaxAInIndex_{\text{I}} * \sum_{k=LB_{\text{J}}}^{UB_{\text{J}}} SID_{\text{J}}(k) + \sum_{k=LB_{\text{J}}}^{Diff_{\text{J}}} SID_{\text{J}}(k)$
ii	$MaxAInIndex_{\text{I}} + case_{\text{PCH}_{==,\text{A,I}-\text{J}}}$
iii	$MaxAInIndex_{\text{I}} * \sum_{k=LB_{\text{J}}}^{UB_{\text{J}}} SID_{\text{J}}(k) + \sum_{k=LB_{\text{J}}}^{Diff_{\text{J}}} SID_{\text{J}}(k) + MaxHInLen_{J} * \sum_{k=LB_{\text{I}}}^{Diff_{\text{I}}} SID_{\text{PCH}}(k) + case_{\text{PCH}_{==,\text{A,I}-\text{J}}}$
iv	$MaxAInIndex_{\text{I}} + case_{\text{PCH}_{==,\text{A,I}-\text{J}}} + \sum_{k=LB_{\text{I}}}^{A2A} case_{\text{PCH}_{==,\text{OR}}}$
Non-Dominant Outer Dimension (5.4.6.2)	
i	$MaxAInIndex_{\text{I}} * \sum_{k=LB_{\text{I}}}^{UB_{\text{J}}} SID_{\text{J}}(k)$
ii	$MaxAInIndex_{\text{I}}$
iii	$MaxAInIndex_{\text{I}} * \sum_{k=LB_{\text{I}}}^{UB_{\text{J}}} SID_{\text{J}}(k) + MaxHInLen_{J} * \sum_{k=LB_{\text{I}}}^{Diff_{\text{I}}} SID_{\text{PCH}}(k)$
iv	$MaxAInIndex_{\text{I}} + \sum_{k=LB_{\text{I}}}^{A2A} case_{\text{PCH}_{==,\text{OR}}}$

5.4.6.1.2 Non-Dominant Condition Statements Due to the non-dominant condition statements, when a H exists in the I dimension, an A may exist in the J dimension. The partial outer size is given by the summation of SIDs in the selected pattern section $MaxAInIndex$ multiplied by the IR(J) and the number of H in the selected pattern section $MaxHInIndex$ multiplied by the summation of SID in the J pattern. The additional elements due to the J index difference are computed by the additional written elements, as explained in the cases of Section 5.4.5.1.2. In the remaining paragraphs we illustrate the equation instantiation for the different condition cases.

Cond. Case v: The size is given by Eq. 5.40, where the additional elements are $min(Diff_{\text{J}}, IR(J))$. An example is depicted in Fig. 5.13(b). The selected section with the maximum A is marked by the black line. The RD pattern for the elements in I dimension is {1H 1A 1H 2A 1H 1A 2H}, LB=-1, UB=9, IR=9, PS=9 and R=1, the RD pattern of J is {1A 2H 2A}, LB=-1, UB=5, IR=5, PS=5 and R=1. The index difference is 5 in I dimension and 1 in J dimension. The selected pattern section is {1A 1H 2A 1H}. The final storage size is 22, i.e. (1+2)*5+(1+1)*(1+2)+min(1,5).

$$Size_{\text{I,J}} = MaxAInIndex_{\text{I}} * IR(J) + MaxHinIndex_{\text{I}} * \sum_{k=LB_{\text{J}}}^{UB_{\text{J}}} SID_{\text{J}}(k) + min(Diff_{\text{J}}, IR(J)) \tag{5.40}$$

Cond. Case vi: The outer dimension is given by the multiplication of the sum of SIDs in the selected pattern by the IR(J) plus the sum of HIDs in the selected pattern multiplied by IR(J)-1, i.e. the sum of SID in the J pattern due to the $\neq$ type. The additional elements are $min(Diff_{\text{I}}, IR(J))$ due to the dominant outer dimension. The result is Eq. 5.41.

$$Size_{\text{I,J}} = MaxAInIndex_{\text{I}} * IR(J) + MaxHInIndex_{\text{I}} * (IR(J) - 1) + min(Diff_{\text{I}}, IR(J)) \tag{5.41}$$

5.4.6.2 Non-Dominant Outer Dimension

In this case, the next WR iteration of I dimension is H and the storage size depends on the type of conditions. The case is similar to Section 5.4.5.2, but the $Writes_I$ are defined by accesses and the holes in the exploration window of the outer dimension and the condition cases.

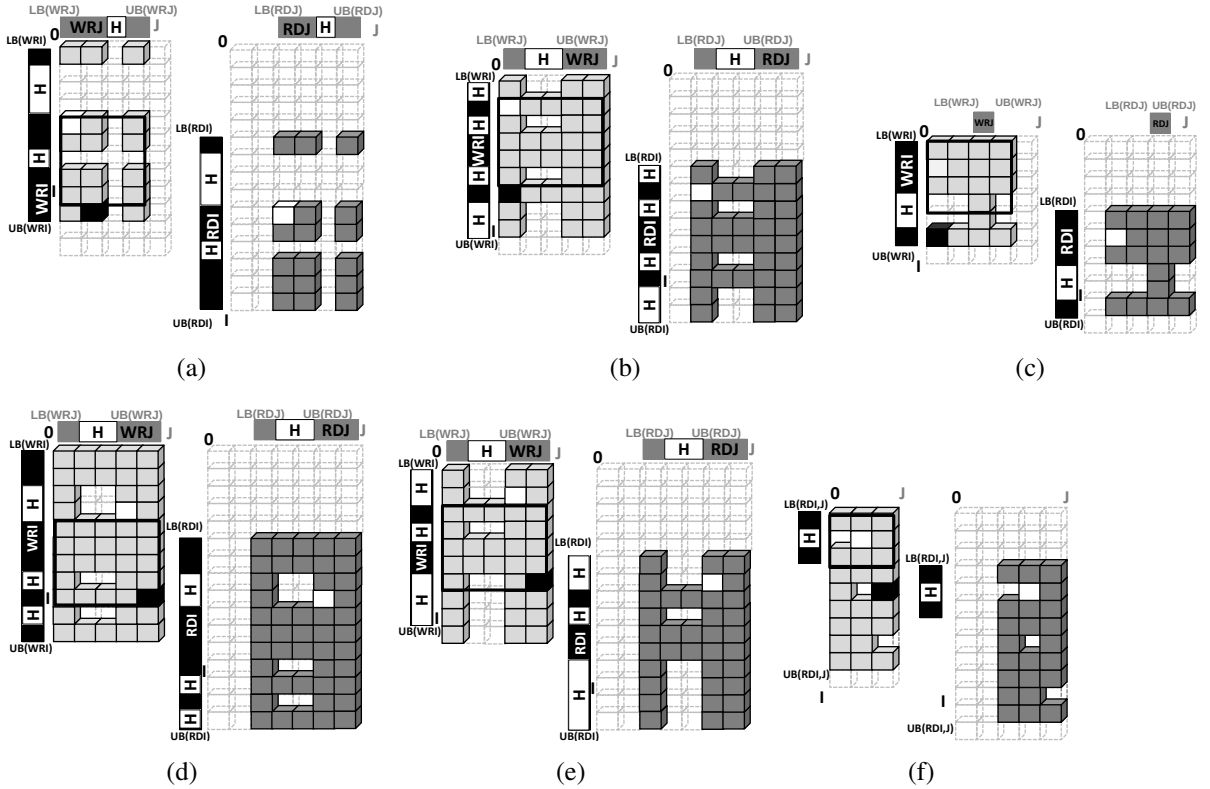


Figure 5.13: Iteration spaces for I and J dimensions for the Non-Dominant Segment in Outer Dimension for: (a) ECS/ECH combined with AND for ISH, (b) ECS/ECH combined with OR when the next WR iteration is A, (c) ECS/ECH combined with OR when the next WR iteration is H and the selected pattern section is in 1st iteration, (d) ECS/ECH combined with OR when the next WR iteration is H, the selected section is not in the 1st iteration and the last WR iteration is H, (e) ECS/ECH combined with OR when the next WR iteration is A, the selected section is not in the 1st iteration and the last WR iteration is H and (f) PCH $\neq$.

5.4.6.2.1 Dominant Condition Statements Similar to Section 5.4.5.2.1, the terms which describe the additional elements are pruned due to the dominant condition statements, as illustrated in the equations at the bottom of Table 5.7.

5.4.6.2.2 Non-Dominant Condition Statements In this case, the additional elements due to the J index difference depend on type of conditions and operations, the RD pattern and the index difference of J dimension.

Cond. Case v: a. Pattern section is in the 1st iteration. The selected pattern section is in the first iteration, so not yet read elements do not exist. The size is given by Eq. 5.29, where the $Writes_I$ are given by the sum of A and sum of H in the outer dimension multiplied by the corresponding J size. The additional writes $Writes_J$ due to J dimension are computed by $case_{\text{WR}}$ using Table 5.5. The result is Eq. 5.42. An example is depicted in Fig. 5.13(c). The selected section is marked with the black line. The RD pattern of I for the elements is {3A 2H 1A}, LB=-1, UB=6, IR=6, PS=6 and R=1, the RD pattern of J is {1A}, LB=1, UB=3, IR=1, PS=1 and R=1. The index difference is 4 in I dimension and 1 in J dimension. The final storage size is 16, i.e. 3*5+1*1+0.

$$Size_{\mathrm{I,J}} = MaxAInIndex_{\mathrm{I}} * IR(J) + MaxHinIndex_{\mathrm{I}} * \sum_{k=LB_{\mathrm{J}}}^{UB_{\mathrm{J}}} SID_{\mathrm{J}}(k) + case_{\mathrm{WR}} \tag{5.42}$$

b. Pattern section is not in the 1st iteration. In this case, RDs exist before the execution of the selected pattern section. The size is given by Eq. 5.27 which is instantiated to Eq. 5.43. The $Writes_I$ are given by the sum of A and sum of H in the outer dimension multiplied by the corresponding J size. The $case_{\mathrm{WR}}$ describes the $Writes_J$, which are computed by Table 5.5. The $case_{\mathrm{RD}}$ describes the $Reads_J$. Due to the non-dominant segment in the outer dimension, the last WR iteration in I dimension can be either A or H, affecting the computation of the $case_{\mathrm{RD}}$. If the last Iteration in the selected pattern section is A, the $case_{\mathrm{RD}}$=$case_{\mathrm{RD,A}}$. If it is H, $case_{\mathrm{RD}}$=$case_{\mathrm{RD,H}}$. The $case_{\mathrm{RD,A}}$ and $case_{\mathrm{RD,H}}$ are computed in Table 5.5. An example with A as last WR iteration is depicted in Fig. 5.13(d). The selected pattern section is depicted by the black line. The RD pattern for the elements in I dimension is {2A 2H 3A 1H 1A 1H 1A}, LB=-1, UB=11, IR=11, PS=11 and R=1, the RD pattern of J is {1A 2H 2A}, LB=-1, UB=5, IR=5, PS=5 and R=1. The index difference is 5 in I dimension and 2 in J dimension. The final storage size is 25, i.e. (3+1)*5+1*(1+2)+2. An example with H as last WR iteration is depicted in Fig. 5.13(e). The selected section is marked with black line, i.e. the first section which starts with A and has the maximum number of A. The RD pattern for the elements in I dimension is {2H 1A 1H 2A 4H}, LB=-1, UB=10, IR=10, PS=10 and R=1, the RD pattern of J is {1A 2H 2A}, LB=-1, UB=5, IR=5, PS=5 and R=1. The index difference is 5 in I dimension and 2 in J dimension. The final storage size is 23, i.e. (1+2)*5+(1+1)*(1+2)+2.

$$Size_{\mathrm{I,J}} = MaxAInIndex_{\mathrm{I}} * IR(J) + MaxHinIndex_{\mathrm{I}} * \sum_{k=LB_{\mathrm{J}}}^{UB_{\mathrm{J}}} SID_{\mathrm{J}}(k) + max(case_{\mathrm{WR}}, case_{\mathrm{RD}}) \tag{5.43}$$

Cond. Case vi: The size is given by Eq. 5.44, where the partial outer size is defined by the max of SIDs in the selected pattern multiplied by the IR(J) plus the holes in the pattern multiplied by the IR(J)-1, as one element is not accessed in the J dimension due to the H in the outer dimension. The additional elements are defined by $case_{\mathrm{PCH}_{\neq,\mathrm{H}}}$ in Table 5.5.

$$Size_{\mathrm{I,J}} = MaxAInIndex_{\mathrm{I}} * IR(J) + MaxHInIndex_{\mathrm{I}} * (IR(J) - 1) + min(Diff_{\mathrm{I}}, IR(J)) + case_{\mathrm{PCH}_{\neq,\mathrm{H}}} \tag{5.44}$$

5.4.7 Combinations in different dimensions

When multiple dimensions are combined, the computation of the storage size starts from the outer dimension and propagates the result to the inner dimensions. The propagation is performed by using the computed size of the outer as $Size_{I,J}$ in Eq. 5.28 when instantiated for the inner dimensions, as described in Section 5.4.4. The initial $Size_{\mathrm{I,J}}$ in Eq. 5.28 is computed by the equations

from Section 5.4.5 to Section 5.4.6, which explore the combinations of the conditions in the same dimensions. The propagation of the $Size_{\mathrm{I,J}}$ between different dimensions depends on the operations which combine the conditions, which affect both the size of the inner dimension and the additional size. We instantiate Eq. 5.28 for the primitive operations || and &&. When both || and && exist between the different dimensions, the equations are combined and the || operation dominates in the computation of the storage size. We illustrate the processes through a selected set of representative examples. The selection has been performed based on the principle that the examples have multiple condition statements, which couple conditions in more than two dimensions affecting the shape of the required storage size. In case the conditions are applied only in two dimensions, the terms of equations of Sections 5.4.5.1 to Section 5.4.6 are multiplied by the size of the loops without conditions.

5.4.7.1 AND primitive operation

If the operation is &&, the $Size_K$ is given by the valid elements in K dimension, i.e. $\sum_{k=LB_{\mathrm{K}}}^{UB_{\mathrm{K}}} SID_k$. The additional elements are given by the inner dimension, i.e. $ASize_{\mathrm{K}}(K)$, and the computation is performed as described in the intra-signal in-place cases of Section 5.4.5 to Section 5.4.6. The result is Eq. 5.45.

$$Size_{\mathrm{I,J,K}} = Size_{\mathrm{I,J}} * \sum_{k=LB_{\mathrm{K}}}^{UB_{\mathrm{K}}} SID_K(k) + ASize_{\mathrm{K}}(K) \tag{5.45}$$

We illustrate an instantiation of Eq. 5.45 in the example of *PCH condition combined with ECS/ECH with AND primitive operation in dominant segment in outer dimension case & non-dominant inner dimension case:* When a PCH in I-J and a ECH in K iterator are combined through AND primitive operation, the case iii (PCH of == type) is combined with with case i (ECS/ECH with AND). Due to AND operation, the partial outer storage size is multiplied with the summation of SID in the K dimension. The additional elements in K dimension are also added. The result is Eq. 5.46.

$$Size_{\mathrm{I,J,K}} = Diff_{\mathrm{I}} * \sum_{k=LB_{\mathrm{K}}}^{UB_{\mathrm{K}}} SID_{\mathrm{K}}(k) + \sum_{k=LB_{\mathrm{K}}}^{Diff_{\mathrm{K}}} SID_{\mathrm{K}}(k) \tag{5.46}$$

5.4.7.2 OR primitive operation

If the combined operation is ||, the $Size_K$ is given by IR(K), since all the elements of the inner dimension are accessed. The additional size due to inner dimension is: 1) the $ASize_I(K)$, i.e. the additional elements due to the conditions in I for K index difference, 2) the $PSize_{K,J}$ multiplied by the IR(J) minus the accessed elements already counted in the conditions of the I dimension, 3) the $ASize_K(J)$, i.e. the additional elements due to the J index difference for the conditions in K, minus $ASize_I(J)$, i.e. the already counted elements due to conditions in I, multiplied by the $Size_K$ and 4) the $ASize_K(K)$, i.e. elements due to K index difference due to conditions in

K, minus $ASize_I(K)$, i.e. the already counted elements in K due to conditions in I. The result is Eq. 5.47. We illustrate in a set of representative instantiations of Eq. 5.47 in the following examples.

$$\begin{aligned} Size_{\mathrm{I,J,K}} = {} & Size_{\mathrm{I,J}} * IR(K) + ASize_I(K) + PSize_K * IR(J) \\ & - \sum_{k=LB_{\mathrm{I}}}^{Diff_{\mathrm{I}}} \sum_{l=LB_{\mathrm{J}}}^{UB_{\mathrm{J}}} SID_J(l) + (ASize_{\mathrm{K}}(J) - ASize_{\mathrm{I}}(J)) \\ & * IR(K) + ASize_{\mathrm{K}}(K) - ASize_{\mathrm{I}}(K) \end{aligned} \tag{5.47}$$

a) PCH of $==$ type (dominant segment in outer dimension & non-dominant outer dimension case) combined with OR with ECS/ECH (dominant segment in outer dimension & dominant outer dimension case): The PCH in I-J defines the partial outer size (Eq. 5.32) which is multiplied by the size of the inner dimension IR(K) due to the $||$ operation. The result is extended by the additional elements due to ECH (Eq. 5.30), i.e. the size due to the dominant segment multiplied by IR(J) minus the A in the PCH for I index difference and the additional elements due to K index difference multiplied by IR(J) minus the A in the PCH for J index difference. The result is Eq. 5.48.

$$\begin{aligned} Size_{\mathrm{I,J,K}} = {} & Diff_{\mathrm{I}} * IR(K) + \sum_{k=LB_{\mathrm{K}}}^{Diff_{\mathrm{K}}} SID_{\mathrm{K}}(k) + (Diff_{\mathrm{I}} * \sum_{k=LB_{\mathrm{K}}}^{UB_{\mathrm{K}}} SID_{\mathrm{K}}(k) \\ & * IR(J)) - \sum_{k=LB_{\mathrm{I}}}^{Diff_{\mathrm{I}}} PCH + (\sum_{k=LB_{\mathrm{J}}}^{Diff_{\mathrm{J}}} SID_{\mathrm{K}}(k) - \sum_{k=LB_{\mathrm{J}}}^{Diff_{\mathrm{J}}} PCH) \\ & * IR(J) + \sum_{k=LB_{\mathrm{K}}}^{Diff_{\mathrm{K}}} SID_{\mathrm{K}}(k) - \sum_{k=LB_{\mathrm{I}}}^{Diff_{\mathrm{I}}} SID_{\mathrm{K}}(k) \end{aligned} \tag{5.48}$$

b) PCH of $==$ type (non-dominant segment in outer dimension & dominant outer dimension case) combined with OR with PCH of $==$ type (dominant segment in outer dimension & non-dominant outer dimension case) in different dimensions: The PCH in I-J, defines the partial outer size (Eq. 5.32) which is multiplied by the size of the inner dimension IR(K) due to the $||$ operation. The result is extended by the additional elements required to be stored due to the PCH in I-J and the K index difference and due to the PCH in I-K, i.e. the dominant segment size of PCH in I-K multiplied by the IR(J) minus one access in the PCH I-J dimension. Other additional elements due to K index difference do not exist due to the dominant segment in outer dimension case and non-dominant outer dimension case of second PCH. The result is Eq. 5.49.

$$\begin{aligned} Size_{\mathrm{I,J,K}} = {} & (MaxAinIndex_{\mathrm{I}} + case_{\mathrm{PCH_{==,A,I-J}}}) * IR(K) \\ & + \sum_{k=LB_{\mathrm{K}}}^{Diff_{\mathrm{K}}} SID_{\mathrm{I}}(k) + (Diff_{\mathrm{I}} + case_{\mathrm{PCH_{==,A,I-K}}}) * IR(J) \\ & - MaxAinIndex_{\mathrm{I}} \end{aligned} \tag{5.49}$$

c) PCH of $\neq$ type (dominant segment in outer dimension) combined with OR with ECS/ECH (dominant segment in outer dimension): The size of PCH in I-J is multiplied by IR(K) due to OR operation. The result is extended by the K index difference due to PCH. The additional elements

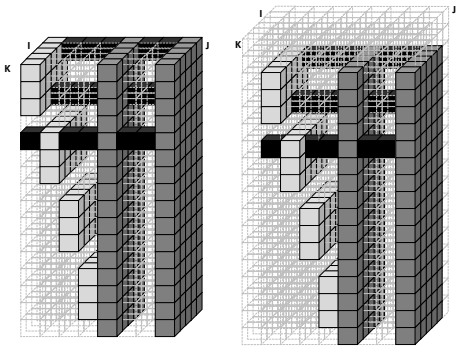
are the size of ECH multiplied by IR(J) minus the accesses in the I-J dimensions, i.e. . The result is Eq. 5.50.

$$
\begin{aligned}
Size_{\mathrm{I,J,K}} = {} & (Diff_{\mathrm{I}} * IR(J) + min(Diff_{\mathrm{J}}, IR(J))) * IR(K) \\
& + \sum_{k=LB_{\mathrm{K}}}^{Diff_{\mathrm{K}}} SID_{\mathrm{I}}(k) + (Diff_{\mathrm{I}} * \sum_{k=LB_{\mathrm{K}}}^{UB_{\mathrm{K}}} SID_{\mathrm{K}}(k) * IR(J)) \\
& - (\sum_{k=LB_{\mathrm{I}}}^{Diff_{\mathrm{I}}} SID(PCH) * IR(J) + \sum_{k=LB_{\mathrm{I}}}^{Diff_{\mathrm{I}}} HID(PCH) \\
& * (IR(J) - 1)) + (\sum_{k=LB_{\mathrm{J}}}^{Diff_{\mathrm{J}}} SID_{\mathrm{K}}(k) - min(Diff_{\mathrm{J}}, IR(J))) \\
& * IR(J) + \sum_{k=LB_{\mathrm{K}}}^{Diff_{\mathrm{K}}} SID_{\mathrm{K}}(k) - \sum_{k=LB_{\mathrm{K}}}^{Diff_{\mathrm{K}}} SID_{\mathrm{I}}(k)
\end{aligned}
\tag{5.50}
$$

5.4.7.3 Demonstration case study

In this section we apply the overlapping intra-signal in-place methodology for the example in Fig. 5.14(a). The analysis step provides the information: three for nested loops, loop order I-J-K, a PCH condition which couples iterator I and J (PCH1) of $<$ && $>$ type, two ECH conditions on iterator J (ECH1 and ECH2) and a PCH condition which couples iterator I and K (PCH2). All conditions are combined with || operation. The I index difference is 2, the J index difference is 1 and the K index difference is 1. The translation step creates the primitive patterns of the PCH conditions per couple of dimensions: for condition I$\geq$4J && (I$\leq$4J+2), the LB of loop I is defined by LB=max(-1, 4*(LB_J+1)-1)=max(-1,-1)=-1, the UB is defined by UB=min(1024, 4*(UB_J-1)+8=min(1024, 1024)=1024, the PS is 4, IR is 1024, the R is 256 and the PCH1 pattern is {3A 1H}. The light gray elements of Fig. 5.14(b) depict the first part of the accessed elements of PCH1 of the three dimensional array. The ECH1 condition of $J == 4$ creates a ECH2 of {1A} pattern with LB=max(-1,3)=3, UB=min(1024,5)=5, the PS is 1, IR is 1, the R is 1. The ECH2 condition of $J == 7$ creates a ECH1 of {1A} pattern with LB==max(-1,6)=6, UB=min(1024,8)=8, the PS is 1, IR is 1, the R is 1. The ECH1 and ECH2 access the dark gray elements in Fig. 5.14(b). The PCH2 condition I==2K has LB=max(-1,2*(LB_K+1)-1)=max(-1,-1)=-1, UB=min(1024, 2*(UB_J-1)+2=min(1024, 1024)=1024, the PS is 2, the IR is 1024, the R is 512 and the PCH2 pattern is {1A 1H}. The PCH2 access the black elements of Fig. 5.14(b). In the intra-signal in-place step, the process of computing the final storage size starts from the outer dimension. The combined pattern of ECH1 and ECH2 is {1A 2H 1A} pattern with LB'=3, UB'=8, the PS is 5, IR is 5, the R is 1. The dominant segment case in outer dimension is selected since ECH is in inner dimensions and due to || operation the complete I dimension is accessed. The case iii of Table 5.4 is selected and the partial storage size is given by Eq. 5.33, i.e. $Size_{I,J} = 2 * (1 + 1) + 1 + 1 * 2 + 1 = 8$. Then, we proceed to the next dimension K. The propagation multiplies the partial storage size $Size_{I,J}$ by size of K due to || operation based on Eq. 5.47. For the K index difference, the dominant segment in outer dimension and the non-dominant outer dimension case is valid. The partial storage size of K is given by case ii in Table 5.6. The additional elements in all dimensions due to K index difference are pruned.

```
For (I=0; I<1024; I++)
 For (J=0; J<256; J++)
  For (K=0; K<512; K++)
   If ((I≥4J)&&(I≤4J+3))||
    (J==7)||(J==4)||(I==2K)
     B[I][J][K]=...
For (I=2; I<1026; I++)
 For (J=1; J<257; J++)
  For (K=1; K<513; K++)
   If ((I≥4J)&&(I≤4J+3))||
    (J==8)||(J==5)||(I==2K)
     B[I-2][J-1][K-1]=...
```



(a) (b) (c)

Figure 5.14: Demonstration case: (a) Code, (b) initial part of the iteration space for the WR iterations and the accessed elements. Each color corresponds to a condition and (c) the execution of the RD statements in the iteration space.

Hence, the additional elements in K dimension are given by the size in K dimension multiplied by the IR(J) minus the sum of A in the combined pattern of I and J dimensions. The final result is $Size_{I,J,K} = 8 * 512 + 1 * (256 - 3) = 4,349$.

5.4.8 PCS storage size

The PCS conditions describe solid iteration spaces in a parametric way, which creates triangular type of shapes in the iteration space. Since the computation of the storage size is based on the maximum number of concurrently alive elements during the execution of the application, the PCS storage size is defined by the worst case in the parametric expression, i.e. the difference of the inner loop bounds. The size derives from Eq. 5.27. The $Writes_{\mathrm{I}}$ are multiplied by the $Size_{\mathrm{J}}$ minus the holes due to the triangular type of shape included in the I index difference, i.e. the PS of PCS multiplied by the index difference minus one. The additional elements are defined by the index difference in J dimension and the iteration before the exploration window, if the PCS is increasing over I dimension and the iteration after the exploration window, if the PCS is decreasing over I dimension. In Fig. 5.15, the index difference is 5 in the I dimension and 2 in the J dimension and the PCS is increasing over I. The worst case is described in the end of the iterations, where ID(J) elements are required to be stored. Hence, the exploration window is from UB(I)-$Diff_{\mathrm{I}}$ up to UB(I)-1, e.g. 1 to 5 for Fig. 5.15. The last A in the exploration window is multiplied by ID(J) and the remaining 4 A in the exploration window of A are multiplied by the size of J is decreased by 1 (due to the size of PCS) each time. This is expressed by multiplying the index difference in I dimension by the size of J and subtracting the holes in the J dimension, as depicted in Eq. 5.51. The additional elements are computed based on the index difference in J and the size of J in the iteration after/before the exploration window of I, i.e. $Size_{\mathrm{PCS}}$. The $Size_{\mathrm{PCS}}$ is given by multiplying the times PCS has been executed in the I index difference with the PS(PCS) and subtract the result from ID(J), $ID(J) - PS(PCS) * Diff_{\mathrm{I}}$. In the example of Fig. 5.15 the I index difference is 5, the PS(PCS) is 1 and, thus the $Size_{\mathrm{PCS}}$=6-5*1=1. Hence, the additional elements are given by

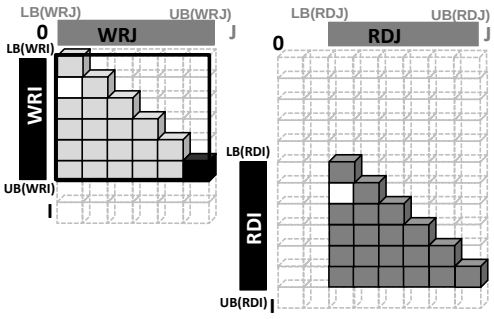


Figure 5.15: Schematic illustration of the WRI and RDI of an increasing PCS in the two dimension case.

min(2,1)=1.

$$Size_{\mathrm{I,J}} = Diff_{\mathrm{I}} * ID_{\mathrm{J}} - (Diff_{\mathrm{I}} - 1) * PS(PCS) + min(Diff_{\mathrm{J}}, Size_{\mathrm{PCS}}) \tag{5.51}$$

5.4.9 Experimental Results

This section presents the results of comparing the proposed methodology with an enumerative approach (lower bound) and an approximation approach (upper bound). The enumerative approach computes the storage size by consistently adding the number of stored elements between the write and the last read of an array element. The process is applied for all elements and the maximum result defines the required storage size. The enumerative approach produces optimal results because all the cases are exhaustively explored. The approximation approach to estimate an upper bound in the ISH cases derives from approximating the H of the patterns with A and computing the storage size based on the equations of Section 5.4.5.1, i.e. based on the index differences and the loop sizes. This approach is quite effective and scalable, and novel on its own already. It also mimics the result of an efficient approximation applied by the symbolic/polyhedral approaches, when applied in irregular iteration spaces. The polyhedral approaches compute the size in irregular ISH cases by creating the convex hull in the iteration space between the WR and the RD of the elements. For instance, in Fig. 5.13(d) the polyhedral approaches approximate the 2x2 and 1x2 holes by accesses to have a convex hull 11x5. The optimistic heuristic of polytopes approximation computes the storage size as in SIS, i.e. solidifies the exploration window. Hence, this approximation approach computes an optimistic upper bound. Other heuristics, which solidify the iteration space, are proposed, e.g. the heuristic in [160] applied when the symbolic techniques cannot be applied due to irregularity/non-uniformity of the accesses. However, it leads to quite pessimistic upper bounds for the case of overlapping store and load accesses. Hence, we have not incorporated it in our experiments.

We present the exploration time and the storage size in number of stored elements for benchmarks from the PolyBench [159] and the MediaBench [107]. The presented benchmarks are

Table 5.8: Dominant Segment in Outer Dimension.

Alg.: Func: Array (init.bounds)	Solution	F	Proposed Meth.		Enumerative Appr.	
			Size (elem.)	Time (ms)	Size (elem.)	Time (ms)
Atax*: A (32)(32)	Eq. 5.31 Sec. 5.4.5.1.1	1	32	0.101	32	14.866
		4	128	0.103	128	816.279
		8	512	0.104	512	52,059.418
		12	2,048	0.101	-	-
Reg_detect*: path (20)(20)	Eq. 5.35 Sec. 5.4.5.1.2	1	21	0.104	21	4.107
		2	61	0.102	61	89.263
		3	201	0.104	201	3,070.95
		4	601	0.102	601	86,142.682
		5	2001	0.100	-	-
Gsm**: Update_residual_signal P3: drp (100)(3)(40)	Table 5.6: case i Sec. 5.4.5.2.1	1	80	0.100	80	153.295
		2	160	0.102	160	608.821
		3	240	0.104	240	1,341.206
		4	480	0.103	480	5,376.844
		5	920	0.104	920	19,166.318
Jacobi-2D*: A (128)(32)	Eq. 5.37 Sec. 5.4.5.2.2	1	34	0.177	34	31.102
		2	66	0.181	66	113.017
		3	1,002	0.190	1,002	23,439.816
		4	8,002	0.175	8,002	1,577,880.183
Doitgen*: sum (10)(10)(10)	Eq. 5.28 Sec. 5.4.4	1	111	0.108	111	41.545
		2	273	0.104	273	452.183
		3	813	0.103	813	6,585.703
		4	1,057	0.103	1,057	12,954.013

(*) Polybench, (**) MediaBench, (-) Memory Overflow

Table 5.9: Non-Dominant Segment in Outer Dimension.

Alg.: Func: Array (init.bounds)	Solution	F	Proposed Meth.		Enumerative Approach		Approximation
			Size (elem.)	Time (ms)	Size (elem.)	Time (ms)	Size (elem.)
Jpeg**: idct_2x2: wsptr (32)(256)	Table 5.7: 5.4.6.1: case i Sec. 5.4.6.1.1	1	84	0.901	84	131.853	132
		2	164	0.896	164	2,558.200	260
		3	644	0.902	644	44,243.614	1,028
		4	1,924	0.899	1,924	5,842,417.563	3,076
Jpeg**: decompress_smooth_data: coef_bits (4)(228)	Eq. 5.41 Sect. 5.4.6.1.2	1	5	0.158	5	18.782	6
		2	10	0.175	10	65.887	12
		3	15	0.183	15	138.009	18
		4	20	0.219	20	234.046	24
		5	25	0.219	25	365.003	30
Jpeg**: idct_2x2: wsptr (un) (32)(256)	Table 5.7: 5.4.6.2: case i Sec. 5.4.6.2.1	1	80	0.798	80	113.259	132
		2	160	0.793	160	2,542.592	260
		3	640	0.795	640	44,142.714	1,028
		4	1,920	0.792	1,920	5,843,373.740	3,076
Gauss-Seidel: A (32)(32)	Eq. 5.44 Sec. 5.4.6.2.2	1	31	0.161	31		32
		2	499	0.162	499	47,432.427	500
		3	999	0.164	999	397,211.781	1,000
		4	1,999	0.163	1,999	3,066,944.504	2,000

(**) MediaBench

selected to cover all the multidimensional intra-signal in-place cases of Fig. 5.8. Each benchmark belongs to one condition case in the intra-signal in-place cases used as representative example in order to have a reasonable result tables size. We have selected to present the multidimensional case as it is more representative of realistic application codes. The time and the size for the one loop dimension cases have also been verified, e.g. for y array in Atax benchmark the time of the proposed methodology is stable around 0.093 ms and the enumerative is from 1.486 - 86,270.273 ms depending on the number of accesses. We also provide one case study of a three dimensional case, i.e. the Doitgen benchmark in Table 5.8. Table 5.8 describes the results of the proposed methodology and the enumerative approach for the dominant segment in outer dimension cases. When the segment which defines the size is solid, the approximation is identical to the proposed methodology, which achieves optimal size with low exploration time. Table 5.9 describes the results for the proposed methodology, the enumerative approach and the approximation for the

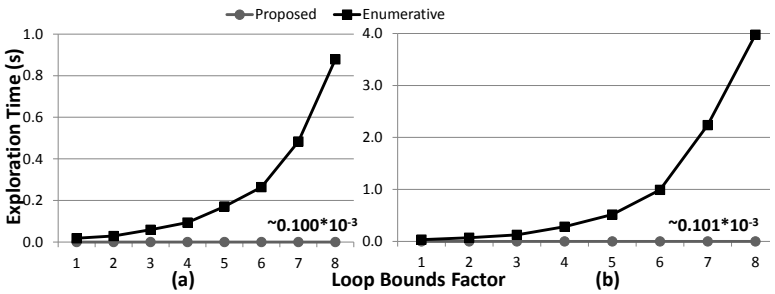


Figure 5.16: Exploration time comparison, when the number of accesses is increased due to an increase by a factor over the loop bounds for (a) Jpeg**: xbuf1 benchmark and (b) Pegwit**: roundKeys_e benchmark.

non-dominant segment in the outer dimension cases. Due to the holes, the approximation leads to size overestimation, as H are considered as A. In contrast, the proposed methodology still has optimal results.

In both tables, the column ``Solution" describes the applied equation of the intra signal in-place case and the corresponding section per benchmark. For each benchmark, different sizes in the number of accesses in the overall iteration space are explored by increasing the loop bounds by a factor F. From the experimental results, the exploration time of the enumerative approach increases super-polynomially with the increase of the accesses in the iteration space. In contrast, the proposed methodology achieves optimal storage size with stable exploration time both for the SIS and the ISH spaces. The increase in the loop bounds modifies the pattern parameters R, UB and IR, which, however, does not affect the methodology exploration time. The time is dominated by the setup times for the equations and the computation of their main primitive components, which are not directly depending on the main size parameters of the loop nests. The exploration times are quite close for all the benchmarks indicating a limited complexity of the proposed methodology. The cases, where the time is slightly increased, is in the non-dominant segment case, where exploration of the outer pattern is applied to select the pattern section, e.g. benchmarks in Table 5.9. We further explore the behavior of our approach in the non-dominant segment cases, when also the index difference is increased by a factor. Then, the proposed methodology adds the SIDs of larger pattern sections during the selection of the pattern section. The exploration time of the proposed methodology is slightly increased due to the increase in the index difference and only for the non-dominant segment in outer dimension cases. In contrast, the exploration time of the enumerative approach is increased both by the increase in the index difference and the increased in the loop bounds for all the cases. For instance, in array coef_bits of the Jpeg** benchmark, the proposed methodology is increased by a factor of 10.9% due to the increase in the index difference, whereas the enumerative approach is increased by a factor of 317.77%, which is the multiplication of factor due to the index difference (39.19%) and the factor due to the loop bound increase (81.09%). We illustrate the behavior of the exploration time of the proposed methodology and the enumerative approach, when the number of accesses is increased for several factors for two additional benchmarks, i.e. array xbuf1 in benchmark Jpeg:make_funny_pointers in Fig. 5.16-(a) and array

roundKeys_e in Pegwit: squareexpandkey benchmark in Fig. 5.16-(b).

When holes exist in the iteration space, the proposed methodology also maintains optimal results and low exploration times, as depicted in Table 5.9. The approximation approach describes an upper bound in the storage size, which overestimates the storage size as the holes are considered as accesses. The size overestimation is increased with the increase in the loop bounds, when more than one loop dimension exist. When the approximation approach considers a H as an A in the I dimension, the complete J dimension is considered as accessed, which increases the size loss. For instance, in the Jpeg benchmark for the wsptr cases in Table 5.9, although the cases are different, the approximation approach leads to the same results, as it does not distinguishes between the dominant outer dimension case and the non-dominant outer dimension case (wsptr with uninteresting components). The size loss in the non-dominant outer dimension case is larger, since the H of the next WR iteration is considered as an A. The time of the enumerative approach in both cases is similar, as the benchmarks difference is the behavior of the outer dimension.

5.5 Conclusions

The proposed methodology computes the storage requirements and remains near-optimal and scalable in iteration spaces with highly irregularly placed holes. The proposed methodology: 1) is split into steps connected by uni-directional constraint propagation, 2) describes the possible cases per step and provides solutions with closed form equations and 3) uses the propagated constraints to select the valid cases per step and apply the corresponding solutions for the non-overlapping and the overlapping case. From the result, the proposed methodology achieves optimal storage size with low exploration time.

Part II

Processing related mapping methodologies

Chapter 6

Design Exploration Methodology for Microprocessor & HW accelerators

6.1 Introduction

Embedded systems usually have hard real-time constraints, which require custom HW designs. Although, they improve the performance, they have a high design cost and very limited flexibility, even when they are made partly configurable. The SW designs provide the required flexibility for a wide range of applications at the cost of reduced performance. Hence, a hybrid SW/HW approach is a promising solution, as it balances the SW flexibility with the HW performance [92]. Existing design tools offer a partially automatic customization of soft microprocessors. These tools usually require a high exploration time as they may explore a wide range of design instances, which are, however, restricted and focused on a relatively limited area of the Design Space (DS) (Section 6.2). When the application characteristics match less with the explored area, this approach leads to suboptimal designs. A very broad DSE corresponds to a very difficult and time consuming task due to the high number of SW and HW design parameters. Hence, the designers usually create SW/HW designs by following ad-hoc or trial and error ways based on their experience [131], which usually require costly design iterations. Hence, a systematic methodology is desired to support a scalable DSE for near-optimal SW/HW co-design [176].

The main contribution of this paper is a scalable DSE methodology to near-optimally map an application domain to a SW/HW FPGA design with a microprocessor core and HW accelerators. The proposed methodology splits the co-design process into sequential mapping steps connected with uni-directional propagation of design constraints avoiding non-scalable steps and needless design iterations. The steps are the Inter-Organization between the microprocessor and the HW accelerators, the Foreground (FG) Memory Management (i.e. the memory attached to the data path) and the Data Path (DP) Mapping. They are described by parametric templates, i.e. a scalable structure with the relevant parameters, equations and functions connected with propagation of constraints, which support efficient and scalable exploration. By giving valid values to the parameters and applying the functions, a Pareto curve with performance-area tradeoff points is

produced per step. The decisions in a mapping step are propagated as design constraints to the next mapping steps to prune incompatible options. An option is incompatible when, in order to be feasible, requires to overlook the design constraints propagated from the previous steps. For instance, scheduling decisions for the critical part are propagated to decide the scheduling of the not-critical part. The options which require to change the scheduling of the critical part are incompatible and thus pruned from the DS. In this way, only the valid and promising DS part is explored based on an efficient pruning through a what-if analysis of the parameters and a scalable DSE is achieved from the early mapping steps of the co-design processing. Scalability is maintained as illustrated in Section 6.5 where the application size increase still allows linear exploration time. The design objectives is to reduce the area and, thus, the required gates for the design, and indirectly the energy consumption, meeting the real-time constraints. Our second main contribution is to demonstrate how the proposed methodology is effectively applied in a set of benchmarks. The most detailed results are provided for a real-time bio-imaging application of a microfluidic-based FPGA. The design gain is 47.11% on performance and 72.29% on area compared with pure SW and HW implementations respectively. Further experimental results for 10 PolyBench benchmarks [159] are presented to substantiate the effectiveness and sufficiently broad applicability of our approach.

The chapter organization is: Section 6.2 presents existing co-design approaches, Section 6.3 describes the proposed methodology, Section 6.4 demonstrates the approach by a real-time bio-imaging application, Section 6.5 shows experimental results for Section 6.4 and for the PolyBench benchmarks. Section 6.6 concludes this study.

6.2 Related Work

Both academic and commercial customizable microprocessors, e.g. Microblaze or NiosII, allow the designer to insert HW components for SW/HW implementations. Several approaches exist in the literature which modify the HW platforms to provide more efficient ways to insert HW accelerators. Some approaches provide a more generic HW accelerator which is capable of executing several applications domain but less efficient for a specific application domain, as it includes redundant resources. For instance, an generic custom unit extension is presented in [196]. Ref [127] presents a generic accelerator which includes several processor clusters and shared memory. In [140] a parameterizable embedded FPGA architecture is used as a HW accelerator and the flow to create the layout, vhdl code and the configuration is described.

However, the tools and the development methodologies are less supportive for a scalable exploration of the efficient mapping options of the applications to (re-)configurable embedded systems [80]. Several design tools exist to partially customize soft microprocessors. Synopsys Synphony C compiler [186] creates accelerators from sequential code. CriticalBlue cascade [36] is an automated co-processors synthesis solution. Cosmos, Handel-C and ImpulseC (a survey available in [32]), provide RTL extensions to C for FPGA design, but providing less efficient results than custom HW designs. The tools identify automated design flows and implement the custom

instructions, but they also require specification of new HW resources and rewriting part of the application [92]. In [48], the compiler generates custom instructions by finding the instruction data paths that can be reused across similar pieces of code and adding them to the customizable processor. However, the exploration and verification time induces a significant overhead [92], while the design may still remain quite suboptimal [48]. ROCCC [62] generates VHDL code for the data path and the control flow of operations for the HW execution in the FPGA. These approaches usually require a high exploration time to create the accelerators focusing on a relatively limited part of the design space and applying costly design iterations. They typically avoid exploration of the options in the organization of the cores and the FG memory by narrowing their search space and thus promising solutions in the unexplored design space cannot be identified. When the application characteristics match less with the search space, the tools can produce sub-optimal solutions. Designers' effort is required to improve the co-design. The alternative broad DSE is difficult and time consuming due to the high number of SW and HW parameters [147].

The designers in industry typically propose designs for the specific applications by following ad-hoc or trial and error ways increasing the costly design iterations. They propose several designs using FPGA manufactures tools, such as Xilinx EDK and Altera SOPC Builder. For instance, a design of an object tracking co-processor was proposed in [174] and a design for an object detection application in [54]. Since substantial time and effort are required to evaluate a design, they usually evaluate few final choices, usually the ones evaluated quickly based on previous experience [56]. Hence, potentially promising options are easily overlooked. In [15] a reconfigurable HW is used as a general-purpose accelerator. Application blocks are mapped using a module library partially exploring the design options as custom units details are overlooked.

Several of the existing DSE methodologies mainly focus on a step of the overall mapping process without taking into account the design constraints of the other mapping steps. For instance, a methodology based on evolutionary Multi-Objective Optimization creates the DP of a HW accelerator for a JPEG algorithm [52]. The compiler in [113] extracts loop parameters to decide memory code transformations, such as unrolling and SW pipelining. Most of the DSE methodologies are recursive approaches, which usually require too much exploration time and are less scalable. Stochastic approaches require too much exploration time to reach near-optimal solutions in a large exploration space. For instance, a Quantum-Inspired Evolutionary algorithm (QEA) is proposed for the multiprocessor mapping problem in [3]. The QEA is an improved heuristic which, however, the optimality of the solution is highly based on the number applied generations, since the increase in the number of generations increases the chances to reach a near-optimal solution. A DSE methodology with stochastic algorithms is proposed in [147]. The independence of the parameters is used to prune the space, which usually is quite restricted, and to derive the Pareto curve in Platune [147]. A simulated annealing approach for DSE of object detection accelerators is proposed in [73]. Another iterative method starts from the designer's base configuration, changes the value of one parameter each time and uses the results to predict the optimal design [175]. It may lead to less efficient designs when a high number of parameters and interdependencies exists. Ref. [176] sorts the parameters based on the impact determined by the maximum parameter value

change. All combinations of the first two high impact parameters are considered.

The proposed methodology describes a scalable DSE which finds the near-optimal co-designs of an application domain to a FPGA with one microprocessor and several HW accelerators. This highly supports the design process since the near-optimal designs can be identified even for large benchmarks. The final design can be selected based on the requirements from the early stages of the design process. The proposed methodology is scalable as it consists of uni-directionally ordered mapping steps, which are required to be applied once and they propagate the design constraints to the next step to prune suboptimal options, as explained in the next Section. This is substantiated by experimental results (section 6.5.3) where show a near-optimal result as opposed to what can be achieved with local iterative improvement techniques.

6.3 Systematic Template-Based Mapping Methodology

The proposed methodology is a scalable DSE to provide the near-optimal designs including the design of the required HW accelerators, which meet timing and resource constraints under performance - area objectives (Pareto Curve). The input of the methodology is the application and the hardware platform and their characteristics are incorporated as domain constraints, which restrict the behavior and the design of the HW accelerators. For instance, the application deadline gives the maximum time that can be allowed to execute the application in the HW accelerators. Hence, any design that has a performance above the deadline constraint is incompatible and pruned. Although constraints exist, the platform provide a significant flexibility in the co-design. The methodology steps are described by a parametric template. A parametric template is created by finding the relevant SW and HW parameters, the functions and how the parameters and the functions affect each other in order to define the direction of propagating the design constraints. The methodology explores per mapping step the valid options inside the available flexibility by giving specific values to the not-constraint parameters of the step (template instantiation). The partial mappings and the decisions are uni-directionally propagated as design constraints to the next step to systematically prune sub-optimal and over-constraint design options from the large exploration space based on constraint analysis. The remaining potentially optimal options are mainly trade-offs which are explored based on a scalable what-if analysis of template parameters. In this way the costly design iterations are avoided. At the final step, the near-optimal designs are depicted in a Pareto curve, where the Pareto points are placed quite close to each other, and from where the designer selects a high quality SH/HW co-design based on the specifications each time.

The main design objective is the area reduction. It reduces the number of gates and indirectly the leakage energy consumption. The dynamic energy consumption is not proportional to the active area. Hence, when constraints are met, the methodology tries to further improve the activity in order to reduce the dynamic energy consumption as a second objective. The real-time constraints imposed from the application context should always remain guaranteed though.

The methodology flow chart is depicted in Fig. 6.1 and the mapping steps are explained in

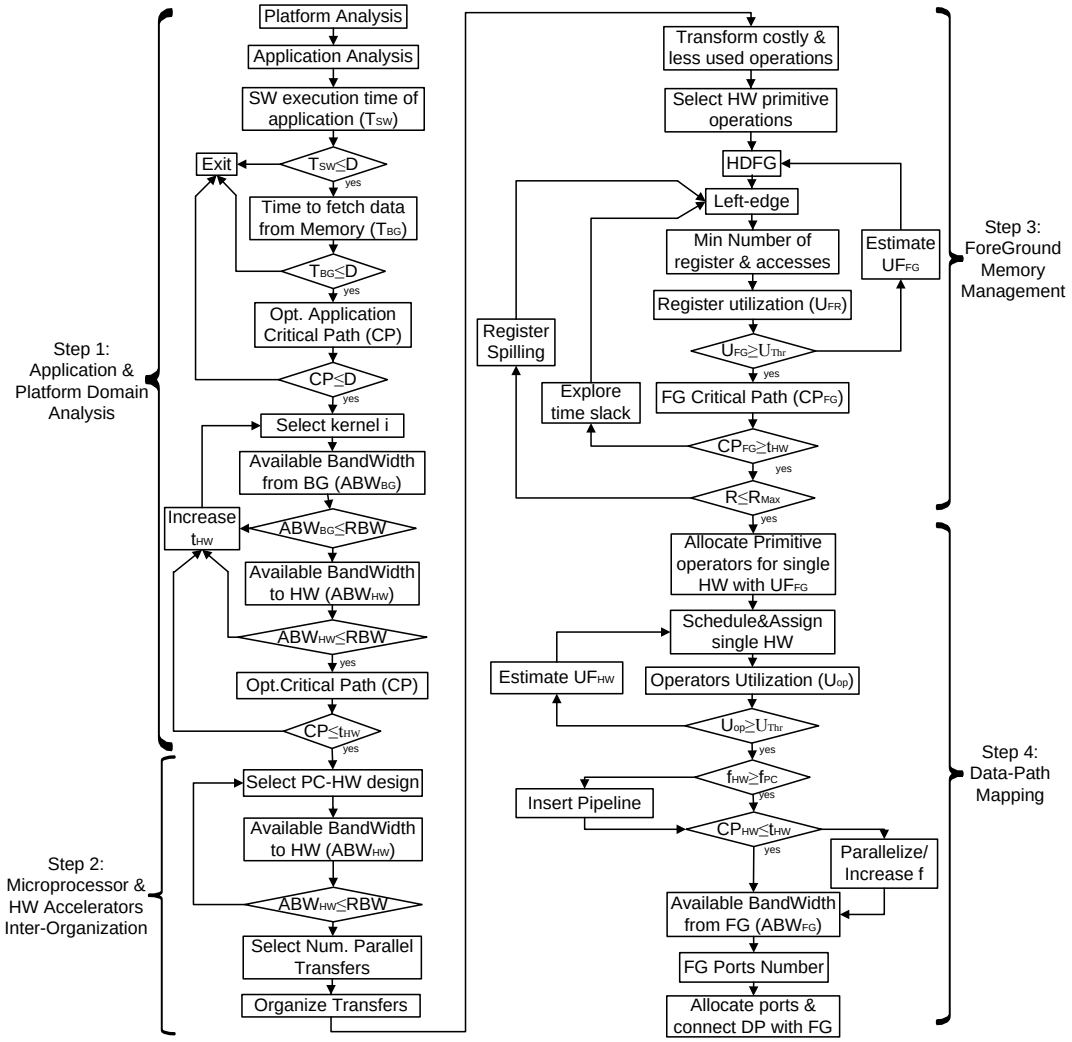


Figure 6.1: The flow of the proposed methodology.

the remaining section following the design constraints propagation order. The Application & Platform Domain Analysis step identifies the SW and HW parameters, e.g. the real-time constraints, the critical kernels and their characteristics, which are propagated as constraints to the Inter-Organization step to decide the microprocessor and the HW accelerators connection. The result is propagated to the FG Memory Management step and then to the DP Mapping, where the final SH/HW design is composed. In the scope of this paper, the communication of the Background (BG) Memory is organized by the SW executed on the microprocessor, using the HW of the memory and the cache controllers of the target platform. Hence, the array accesses in SW are compiled into load/store operations and the cache controller handles the data [70]. Application platform independent transformations have been upfront applied.

The application domain under study is described by embedded systems applications with one thread frame. The thread frame has deterministic behavior, i.e. consists of several condition statements and nested loops, but without including any event triggered task generation or non-deterministic elements. The application is highly data dominated with increased computation requirements. The application real-time constraints are expressed as latency constraints, i.e. the

ALGORITHM 9: Application & Platform Domain Analysis

Step 1.1: Platform Analysis\
Determine HW parameters();
Step 1.2: Application Analysis\
Identify basic kernels;
for *(i=0;i<NumOfKernels;i++)* **do**
    Profile t_i;
    Identify SW parameters();
Kernels=Sort based on t_i;
Identify control flow;
Step 1.3: Decide SW & HW execution\
Step 1.3.1: Application Constraints\
$f_{PC} = max(Av_{f_{PC}})$;
t_{SW}=Assign(Application, PC);
t_{tot}=t_{SW};
if *(t_{tot} <D)* **then**
    Exit;
High level estimate($t_{TR,Opt,CDFG}$);
if *($t_{TR,Opt,CDFG} > t_{HW}$)* **then**
    Exit(Change HW parameters);
High level estimate($CP_{HW,Opt,CDFG}$);
if *($CP_{HW,Opt,CDFG} > t_{HW}$)* **then**
    Exit(Change HW parameters);
High level estimate($t_{TR,Opt,CDFG}$);
if *($t_{TR,Opt,CDFG} > t_{HW}$)* **then**
    Exit(Change HW parameters);
Step 1.3.2: Kernel Constraints\
while *($t_{tot} > D$)* **do**
    k=select i from Kernels;
    t_{SW}=t_{tot}-t_i;
    t_{HW}=(1-slack)*(D-t_{SW});
    Assign(k,HW);
    Compute $BandW$, $ABandW_{BG,Opt}$;
    if *($BandW > ABandW_{BG}$)* **then**
        Repeat=1; break;
    Compute $ABandW_{PC-HW,Opt}$;
    if *($BandW > ABandW_{PC-HW,Opt}$)* **then**
        Repeat=1; break;
    High level estimate($CP_{HW,Opt}$);
    if *($CP_{HW,Opt} > t_{HW}$)* **then**
        Repeat=1; break;
    if *(Repeat==1 && SWsolution==1)* **then**
        t_{tot}=t_{SW}+t_{HW};
    else
        Exit(Change HW parameters);

allowed time between two parts of the application, and as throughput constraints, i.e. the number of application executions that should be completed in a given time interval, or as their combination. The throughput is translated into a latency constraint and the kernel is unrolled based on the Iteration Interval [106]. The derived latency constraint and the transformed kernel are inputs to the methodology. When a combination of latency and throughput constraints exists, the throughput constraint is translated to a latency constraint and the most restricting one is selected. For instance, wireless applications usually have a throughput constraint per incoming sample and a latency constraint over the payload processing of the wireless baseband. Multiple real-time constraints for different parts of the application are handled by distributing their effect over the different parts/kernels of the application code. When they focus on the same part, their requirements are combined into a common delay constraint set. The platform domain is a heterogeneous FPGA with a microprocessor core and parallel HW accelerators. When the platform or application domain is partly modified with similar characteristics, the main principles will remain valid and they can be reprojected to produce the mapping methodology. For larger modifications to domains with clearly different characteristics, a more extensive exploration of the new principles and projections has to be initiated.

6.3.1 Step 1: Application & Platform Domain Analysis

The pseudocode of Application & Platform Domain Analysis step is depicted in Alg. 9 and explained in the next paragraphs.

Table 6.1: Main Application and platform domain Parameters used by the proposed method.

SW Parameters		HW Parameters	
Parameter	Description	Parameter	Description
D	Execution Deadline	f_{HW}	Hardware Frequency
TP	Throughput	$Av_{\text{f}_{\text{HW}}}$	Available HW Frequencies
t_{Tot}	Total Execution Time	f_{PC}	MicroPC Frequency
t_{SW}	Exec. Time in MicroPC	$Av_{\text{f}_{\text{PC}}}$	Av. MicroPC Frequencies
t_{HW}	Exec. Time in HW acc.	f_{Mem}	Memory Frequency
For each kernel i		f_{MemBus}	Memory Bus Frequency
Regular	Data-Dependent execution	W_{MemBus}	Memory Bus Width
Data	Number of transfered Data	$f_{\text{Bus}_{\text{PC}-\text{HW}}}$	MicroPC & HW acc. Bus Freq.
DType	Type of Data	$W_{\text{Bus}_{\text{PC}-\text{HW}}}$	MicroPC & HW acc. bus Width
RF	Iterations between Data reuse	$Bus_{\text{PC}-\text{HW}}$	MicroPC & HW acc. buses
RData	Number of reused Data	$Av_{\text{Bus}_{\text{PC}-\text{HW}}}$	Av. MicroPC & HW acc. buses
RType	Type of RData	$Size_{\text{FG}}$	Size of FG memory
Res	Number of results	$Av_{\text{Size}_{\text{FG}}}$	Av. Sizes of FG memory
ResType	Type of results	W_{PLB}	PLB bus Width
Var_{Data}	Variables for Ops.	f_{PLB}	PLB bus Frequency
$VType_{\text{Data}}$	Type of variables for Ops.	MemBus	Memory Buses
Var_{Cntr}	Variables for control flow		
$VType_{\text{Cntr}}$	Type of control flow variables		
OPs_{Arithm}	Number of Arithmetic Ops.		
$OPsType_{\text{Arithm}}$	Type of Arithmetic Ops.		
$Cost_{\text{OPsType}_{\text{Arithm}}}$	Arithmetic Op Cost		
$Occ_{\text{OPsType}_{\text{Arithm}}}$	Arithmetic Op Occupation		
OPs_{Cntr}	Number of Control Flow Ops.		
$OPsType_{\text{Cntr}}$	Type of Control Flow Ops.		
$Cost_{\text{OPsType}_{\text{Cntr}}}$	Control Flow Cost		
t	execution time		
iter	Loop iterations		

6.3.1.1 Step 1.1: Platform Analysis Step

The platform is analyzed to determine the HW parameters, which describe the physical constraints of the main memory, the memory buses, the microprocessors, the buses between the cores, the local memory, the HW accelerator. The main HW parameters are depicted in Table 6.1.

6.3.1.2 Step 1.2: Application Analysis Step

This steps determines the SW parameters by analyzing the application. The main SW parameters are described in the left part of Table 6.1. The application real-time constraints determine the throughput TP and the deadline D, which defines the maximum bound on the application execution time t_{Tot}, $t_{\text{Tot}} \leq D$. For instance, in video applications the TP derives from the Frame Rate (FR), i.e. frames per second, and $D = 1/TP$.

Based on the application structure, the loops, the basic kernels and the kernels execution time t_{i} are profiled. The kernel type is defined through the parameter $Regular$, i.e. when $Regular$=1 it is executed in every loop iteration, otherwise the execution depends on the specific values of the data in the control flow operations. The kernel arithmetic operations are identified and characterized by the cost (in terms of number of gates) and by the occupation factor. The number and the type of the data required from the memory, the results, the variables of the arithmetic operations, their dependencies etc. are identified. The control flow operations (e.g. parameters $OPs_{\text{Cntr}}(p)$) and the corresponding variables (e.g. $Var_{\text{Cntr}}(k)$) are identified (Table 6.1). The main SW parameters are depicted in left part of Table 6.1. The kernels are sorted by t_{i} to identify the most critical ones.

6.3.1.3 Step 1.3: Decide SW & HW execution

This step decides the kernels to be mapped to SW and to HW. The microprocessor frequency f_{PC} is set to the maximum available. The time required to execute the application in SW, i.e. in the microprocessor, is computed. If the t_{tot} is smaller than the available time D, the SW design meets the timing constraints. A further exploration to decrease f_{PC} while timing constrains are met, can be applied. If t_{tot} is larger than D, SW/HW designs are required. Then, a lower frequency for the microprocessor is less efficient, because in order to meet D, parallelization of HW accelerators would be required increasing the FPGA active area and energy consumption. This option will be efficient if the TP and the D of the application requirements are quite low. Then, the microprocessor probably also meets the real-time constraints. To motivate a microprocessor lower frequency, the platform should provide a fine-grained voltage selection or have a frequency reduction by at least a factor of 2 for significant gain.

6.3.1.3.1 Step 1.3.1.: Application Constraints In this step is verified if the overall design is possible based on constraint reasoning. In case operator strength reduction [34] has not been applied during platform independent transformations, we replace the constant operations with simpler operations, e.g. a constant multiplication is replaced by a sequence of Shift-Add (ShA) operations [142]. The bandwidth required to transfer the data from the memory is given by Eq. 6.2, where the number and the type of data per kernel are accumulated ($Data_{\text{CDFG}}$ given by Eq. 6.1) and divided by the available time D. The available bandwidth is given by Eq. 6.3. If D is less than the time required to transfer the data, the problem is over-constraint. Hence, changes in the platform characteristics, i.e. HW parameters changes, are required. Otherwise, SW/HW designs are explored.

$$Data_{\text{CDFG}} = \sum_{i=0}^{kernels} \sum_{j=0}^{DTypes} Data(j) * DType(j) * Iter(i) \tag{6.1}$$

$$BandW = \frac{Data_{\text{CDFG}}}{D} \tag{6.2}$$

$$ABandW_{\text{BG,Opt}} = MemBus * W_{\text{MemBus}} * min(f_{\text{Mem}}, f_{\text{MemBus}}) \tag{6.3}$$

The critical path of the application $CP_{\text{CDFG,Opt}}$ is estimated in a high level way considering an optimal mapping to the HW accelerator and taking into account potential HW area constraints, e.g. based on the method of [47]. If the estimated critical path is higher than the available time, D, the problem is over-constraint and HW parameters changes should be applied.

6.3.1.3.2 Step 1.3.2.: Kernel Constraints The SW/HW DSE is mainly based on the basic kernels, which take nearly all of the required workload on the platform resources. The remaining non-critical part of the code, which is usually dedicated to initializations, is assumed to be absorbed in the slack that is introduced during the mapping steps. For instance, the preamble/postamble code, which may introduce non-regularity, is mapped to the microprocessor when the speed up factor

required to meet real-time constraints is quite low. When the speed up factor is significantly high or the communication overhead is highly increasing [87], it is mapped to the HW accelerators. In this way we can restrict the design-time spent on the entire application and the effort of industrial designers. The available time is then fully focused on obtaining a near-optimal result for the kernels. That part of the design effort should remain scalable because real-life applications will still involve a substantial amount of code (several kernels) to be dealt with. The proposed methodology ensures these characteristics.

The control flow operations can be executed either in the microblaze, i.e. parameter $OPs_{\text{Cntr}} \in SW$ or implemented in the HW accelerator, $OPs_{\text{Cntr}} \in HW$. In the first case, the complexity is moved to the microprocessor, whereas in the second case, dedicated control and diverse FUs are inserted to the HW accelerator. By propagating the design objectives to these two options, the control statements are selected to be executed in the microprocessor. In this way, the HW accelerator design complexity and the synchronization between the microprocessor and the HW accelerators is reduced, which allows a very efficient and high-performance HW accelerator CP design dedicated only to the execution of the arithmetic operations.

The design objectives, i.e. area reduction, are propagated to select the kernels for mapping in the HW accelerators. Hence, the smallest number of kernels should be mapped in the smallest number of HW accelerators. The kernel with the highest t_{i} is initially selected. When two kernels have similar t_{i}, the kernel with $Regular = 1$ is favored, due to the high HW accelerator use, the execution regularity and the simpler synchronization scheme between microprocessor and accelerators. The irregular part of the application is mapped to the microprocessor.

When kernel i is selected, the HW part has in the worst case $t_{\text{HW}} = (1 - slack) * (D - t_{\text{SW}})$ using a $slack$ percentage for the non-critical code. The kernel constraint verification is performed based on the HW parameters and the optimal design. Hence, pipelining between the data fetching from the memory, the data transferring to the HW accelerator and the computation of the HW accelerators is assumed. The number of data required from BG memory are given by Eq. 6.4, where $Data(j)$ is the number of data with data type j for one iteration of the kernel i. The total number of data is given by Eq. 6.5, where $Iter(i)$ are the loop iterations of kernel i in the available time t_{HW}. The required (available) transfer bandwidth is given by Eq. 6.6 (Eq. 6.7). Similar equations exists for the microprocessor to the HW accelerator transfer assuming optimal design, i.e. the bus with the maximum bandwidth (Eq. 6.8). If the required bandwidth from the memory (to the HW accelerator) is larger than the available one, the problem is over-constraint. In this case two potential options exist: 1) HW solution: change the HW parameters, or 2)SW solution: increase the available time t_{HW}. The latter is expressed by mapping in HW the best candidate to reduce

ALGORITHM 10: Microprocessor & HW Accelerators Inter-Organization

```
Dependent=1;CoProcessor=1;Control=0;
Compute ABandW_PC−HW;
if (BandW > ABandW_PC−HW) then
    Dependent=0;NPI=0;
    Compute ABandW_PC−HW_PLB;
    if (t_TR,PC−HW_PLB > t_HW) then
        NPI=1;
        Explore(SCIP-NPI);
High level estimate of t_TR,PC−HW;
if (t_TR,PC−HW << CP_HW,OPt) then
    t_HW = t_HW − t_TR,PC−HW
Compute Bus_PC−HW;
```

t_{SW}, and, thus increase t_{HW} and reduce the required bandwidth.

$$Data = \sum_{j=0}^{DTypes} Data(j) * DType(j) \tag{6.4}$$

$$Data_{Tot} = Data * Iter(i) \tag{6.5}$$

$$BandW = \frac{Data_{Tot}}{t_{HW}} \tag{6.6}$$

$$ABandW_{BG,Opt} = MemBus * W_{MemBus} * min(f_{Mem}, f_{MemBus}) \tag{6.7}$$

$$ABandW_{PC-HW,Opt} = max(Av_{Bus,PC-HW,max}) * W_{Bus,PC-HW,max} * f_{Bus,PC-HW,max} \tag{6.8}$$

The next step is to estimate in a high-level way the critical path $CP_{HW,Opt}$ of the HW accelerator DP. This is achieved considering an optimal HW accelerator design that the platform constraints allow based on the minimum possible path in terms of delay for the given target technology [47]. In this way, we can verify that the optimal HW accelerator can support the execution of the kernel. If the optimal estimated critical path is larger than the available time, either the HW parameters have to be modified or the t_{HW} has to be increased.

6.3.2 Step 2: Microprocessor & HW Accelerators Inter-Organization

This step decides the organization of the microprocessor and the accelerators and the transferring of the data between the cores. The pseudocode is depicted in Alg. 10.

6.3.2.1 Step 2.1: Microprocessor & HW acceleration connection

This step decides the organization of the microprocessor and the accelerators. A HW accelerator can be integrated into the platform in several ways as depicted in the parametric template of Fig. 6.2. Table 6.2 shows the corresponding design options.

A HW accelerator can be fully independent or partially dependent on the microprocessor (parameter $Dependent$). When the parameter $Dependent = 1$, the HW accelerator partially depends on the microprocessor. It reuses the microprocessor resources, e.g. memory interface, which reduces the custom design complexity, the area and the energy consumption. When

Table 6.2: Truth table of Microprocessor & HW Accelerators Inter-Organization

Parameters				HW Accelerator Connection
Dependent	Co-processor	Control	NPI	
0	Not-care	Not-care	1	Standalone Custom IP-NPI
0	Not-care	Not-care	0	Standalone Custom IP-PLB
1	0	Not-care	Not-care	Internal FU
1	1	0	Not-care	Base Co-processor
1	1	1	Not-care	Control Co-processor

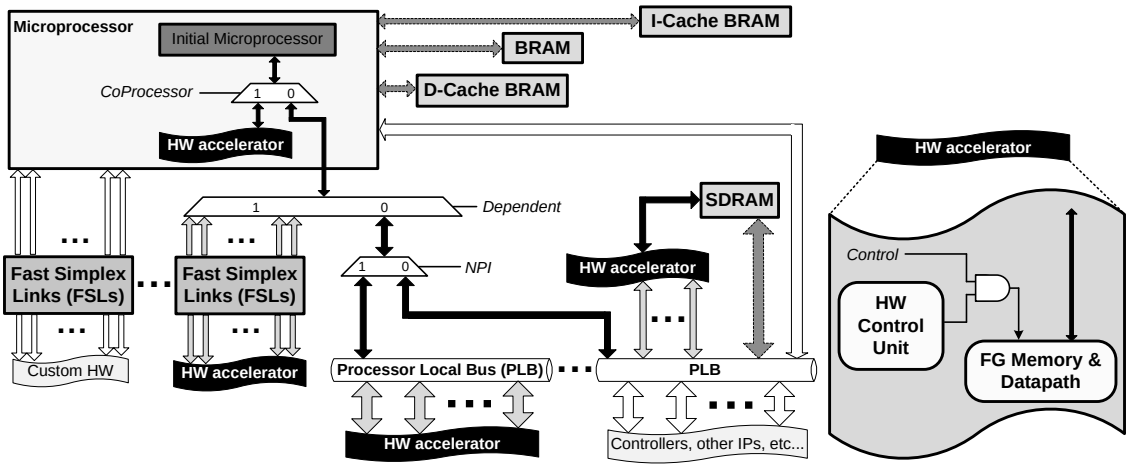


Figure 6.2: Inter-Organization of the microprocessor & the HW accelerator parametric template.

$Dependent = 1$, the HW accelerator can be implemented as an extension of the internal Function Units (FUs) of the microprocessor or as external co-processor (parameter $CoProcessor$). When $CoProcessor = 0$, the HW accelerator is implemented as an internal FU. This implementation affects the critical path of the microprocessor potentially reducing the $f_{\text{HW}}(f_{\text{PC}})$, which is not acceptable since real-time constraints may be not met. The implementation of the HW accelerator as a co-processor ($CoProcessor = 1$) removes these limitations. The microprocessor and the co-processor can execute different parts of the application at the same time. Hence, the microprocessor can be used for the memory address generation and the fetching of data from the memory, while the co-processor executes the kernel operations. The interconnection of the microprocessor and the co-processor is quite fast, a small latency (one or two cycles) is required to write (read) the data to (from) the co-processor, which makes this option Pareto (near-)optimal for our domain, when constraints allow it.

When parameter $Dependent = 0$, the HW accelerator is connected independently from other HW resources to the communication channel (Standalone Custom IP - SCIP). In this design, apart from the FG memory and the DP, the HW accelerator has a BG memory interface, similar to ``Fire and Forget'' model. The BG interface can be implemented by: 1) a custom design, e.g. an Native Port Interface (NPI) [207] (expressed through parameter NPI=1) designed to control the memory in the most efficient way, or 2) using a common bus protocol, e.g. Processor Local Bus (PLB) (NPI=0). The first case is optimal, but with higher area and design effort, which is acceptable when other approaches are over-constraint (see below). The second case is less efficient

due to the bus protocol bottleneck. In the case when the bus option cannot provide the required bandwidth, a NoC based communication topology can be explored as valid option to replace the bus option.

In the co-processor implementation two further options exist for the control, i.e. loop organization, control operations and structure. It can be common or different between the microprocessor and the co-processor. When parameter $Control = 0$, the microprocessor takes care of the control of the kernel operations that are executed on the DP of the co-processor. The responsibility for the synchronization of the data between the two cores resides on the microprocessor. This allows a more efficient, smaller area and lower energy consumption design of the co-processor (Base co-processor). This option fully meets the design objectives, so it is a Pareto (near-)optimal for simple synchronization schemes. The microprocessor is responsible for invoking the co-processor, which executes the operations and writes back the result. If $Control = 1$, different control structures can exist in the processor and in the co-processor. The co-processor requires a HW Control Unit, e.g. Finite State Machine (FSM), to support the correct functionality of the DP and the communication from/to the microprocessor. By propagating the design objectives, this option is potentially sub-optimal due the increased co-processor area. However, when the design synchronization is more complex, this option should be valid.

6.3.2.2 Step 2.2: Microprocessor & HW acceleration communication

This step is dedicated to organizing the transferring of the data between the cores. The available bandwidth to transfer the data between the cores for the less costly design is verified if it is sufficient. The available bandwidth of the Base co-processor design is given by Eq. 6.9 considering optimal data transfer, i.e. the maximum number of buses between the cores. When the bandwidth is insufficient, the alternative designs are explored. The next less costly option is the Standalone Custom IP with PLB and last the high design effort option of the Standalone Custom IP with NPI.

$$ABandW_{\text{TR,PC}-\text{HW}} = max(Av_{\text{Bus,PC}-\text{HW}}) * W_{\text{Bus,PC}-\text{HW}} * f_{\text{PC}} \tag{6.9}$$

If the bandwidth of the optimal parallel transfer is acceptable and the time required to transfer the data $t_{\text{TR,PC}-\text{HW}}$ is computed. The available time t_{HW} is updated based on the time required to transfer the data from the background memory and it is verified if enough time is available to transfer the data. The number of minimum required parallel transfers, i.e. the $Bus_{\text{PC}-\text{HW,min}}$, derives from Eq. 6.10. If $t_{\text{TR,PC}-\text{HW}}$ is negligible compared to the estimated critical path of the DP, sequential data transfer can be selected to reduce the design effort. In this case, the lifetime of the variables in the FG memory is increased, which restricts the next mapping step. By giving different values to the number of transfers (which still satisfy the minimum requirements) different pareto points are produced. The available time for computation is updated, t_{HW}=t_{HW}-$t_{\text{TR,PC}-\text{HW}}$.

$$Bus_{\text{PC}-\text{HW}} = (int)\frac{BandW}{ABandW_{\text{TR,PC}-\text{HW}}} \tag{6.10}$$

The application is transformed accordingly with the corresponding operations to write and read the data to the bus connecting the microprocessor and the HW accelerator. The scheduling of the transfers and their address generation (e.g. DMA or Load/Store instructions) is taken care of by the Microprocessor compiler.

6.3.3 Step 3: Foreground Memory Management

The dimensioning and the management of the FG Memory is decided in this step. The corresponding pseudocode is depicted in Alg. 11. The size of the FG Memory of the microprocessor is given by the platform parameters, whereas the FG memory of the HW accelerator is determined by the required data for the kernel execution. The FG dimensioning and the operations executed in the HW accelerator depend on the constraints propagated from the previous steps: When $Dependent = 1$ and $Coprocessor = 0$, the FG of the HW accelerator is the FG of the microprocessor. When the $Coprocessor = 1$, the scalars to be stored in the FG Memory are based on the $Control$ parameter. If $Control = 0$ only the scalars for arithmetic operations are stored in the FG, i.e. the Data from BG memory, the arithmetic variables, the intermediate results and the final results. If $Control = 1$, the scalars for the control flow are also stored in the FG, since the control flow operations are executed in the HW accelerator. The FG should be sufficiently large to support all the control and flow operations, increasing energy consumption and reducing the opportunities for DP parallelism. When the parameter $Dependent = 0$, the FG Memory of the HW accelerator should store the scalars required for control and arithmetic operations.

The bandwidth and the energy requirements for BG memory are reduced by replacing several BG accesses with FG ones. In the Pareto optimal case, only the new $Data$ are transfered from the BG memory and the intermediate results are kept in the FG memory avoiding register spilling. Otherwise the most used scalars are maintained in the FG memory and the remaining ones are stored back to the BG memory, increasing the latency and the number of future accesses to FG memory. The spilling option is acceptable, if HW parameters restrict the optimal option.

Initially, a cost operations analysis takes place, as it affects the scalars required to be stored in the FG memory. Each operation type of the kernel is evaluated based on the number of gates and the occupation factor. If an operation type is characterized as costly (during application analysis) and has a low occupation factor, exploration is applied to replace it by smaller and simpler operations, which require less gates to be executed. In this way the area is reduced and the use of the simple resources is highly increased. The kernel loop is transformed accordingly.

The next step is to define the HW accelerator primitive operations, i.e. the operations used in the data path. This step is similar to the instruction set selection process in microcoded processors [91], but modified for the HW accelerator operations. The primitive operation selection can be achieved through exploration based on techniques that use cost functions related to the utilization of the operations, the area cost, the critical path etc. After the primitive operation set selection, the application code is modified accordingly and a new estimation of the critical path is used to verify that the real-time constraints are met.

ALGORITHM 11: Foreground Memory Management

$OPs = \sum_{n=0}^{OPsType_{\text{Arithm}}} OPs_{\text{Arithm}}(n)$
if *(Coprocessor==0)* **then**
  $Size_{\text{FG}} = Size_{\text{RegFile}}$; Exit;
if *(Dependent==0)||(Dependent==1 && Coprocessor==0 && Control=1)* **then**
  $OPs = OPs + \sum_{n=0}^{OPsType_{\text{Cntr}}} OPs_{\text{Cntr}}(n)$
Evaluate(OPs);
Select primitive operations;
UF_{FG}=1;
Evaluate Reg(UF_{FG});
Compute U_{FG};
Num_{Reg}=FG;
Exploration(UF_{FG}, ∞);
if *($Num_{\text{Reg}} < Num_{\text{Reg,Max}}$)* **then**
  Num_{Reg}=FG;
  Exploration($UF_{\text{FG}}, Num_{\text{Reg,Max}}$);
$Accesses_{\text{FG}}$=Count(RD,WR,MLT);
FG Exploration(U_{FG},Max){
while *($U_{\text{FG}} < U_{\text{Thr}}$)* **do**
  Update UF_{FG}; Evaluate Reg(UF_{FG},Max);
  Compute U_{R};
while *($CP_{\text{FG}} < t_{\text{HW}}$)* **do**
  Reduce Num_{Reg};
  Evaluate Reg(UF_{FG},Num_{Reg});
}
Evaluate(OPsType){
for *(j=0;j<OPsType;j++)* **do**
  if *($Cost_{\text{j}}$ >Threshold)* **then**
    Explore for transformation of operation(j);
    Update(SW Parameters);
    Transform(Code);
}
Evaluate Reg(UF,PF,Max){
Compose HDFG(UF,PF);
Compute life time(HDFG);
SLT=Sort(life time);
MLT=Left edge algorithm(SLT,Max);
Num_{Reg}=Count(MLT);
}

A Hierarchical Data Flow Graph (HDFG) with the primitive operations is introduced, which includes the data dependencies, (satisfying the control dependencies), the operations of both branches in a control flow operation and the loop structure information. The HDFG provides a relevant ordering of the operations based on a realistic optimistic scheduling, i.e. As Soon As Possible (ASAP) of the critical path and As Late As Possible (ALAP) of the remaining operations. This realistic optimistic scheduling leads to smaller scalar life times. The next step computes the life time of the scalars of the HDFG, e.g. using technique in [157]. The result is sorted per slot in increasing scalar lifetime to enable an efficient register allocation algorithm, e.g. left edge [170]. It merges the scalars with non-overlapping lifetime. It starts from the second slot and moves the scalars to the left to be merged with scalars whose lifetime has terminated. The result Num_{Reg} provides the minimum number of registers for the realistic optimistic scheduling.

The HDFG of the one execution of the kernel i is explored and the minimum number of registers is computed (Num_{Reg}) for this maximally parallel execution order. The utilization of the scalars U_{FG} is computed by Eq. 6.11. If the scalars are distributed in a less balanced way and several holes exists, unrolling by a factor of UF is explored to increase the utilization of the FG resources. The UF_{FG} is estimated by Eq. 6.12 and the FG dimensioning process is repeated.

$$U_{\text{FG}} = \frac{\sum_{i=0}^{Num_{\text{Reg}}} Holes(i)}{Num_{\text{Reg}} * Slots} \quad (6.11) \qquad UF_{\text{FG}} = \frac{1}{U_{\text{FG}}} \quad (6.12)$$

The FG critical path is computed based on the control and data dependencies and on an estimation of the f_{HW}, e.g. the lower bound of f_{PC}. If the critical path is lower than t_{HW}, the time slack can be used to increase the schedule length and to reduce the number of registers. In case

ALGORITHM 12: Data Path Mapping

Step 4.1: DP Mapping\
Allocate(Primitive Ops,Single HW);
Schedule, Assign(UF_{FG},Single HW);
UF_{HW}=1;
Compute U_{Op};
while *($U_{\text{Op}} < U_{\text{Thr}}$)* **do**
  Update UF_{HW};
  Schedule, Assign(UF_{HW},Single HW);
  Compute U_{R};
if *($f_{\text{HW}} > f_{\text{PC}}$)* **then**
  Estimate PL stages;
  Pipeline(PL);
if *($CP_{\text{HW}} < t_{\text{HW}}$)* **then**
  Parallelize(PF,Single HW);
if *($CP_{\text{HW}} < t_{\text{HW}}$)* **then**
  Pipeline(PL);
Step 4.2: FG Memory Connection\
Estimate $ABandW_{\text{FG}}$;
$Num_{\text{FG,Ports}}$=$\frac{BandW}{ABandW_{\text{FG}}}$;
Allocate ports;
Connect(DP,FG);

a HW parameter constraints the number of available registers ($Num_{\text{Reg,max}}$) and it is lower than the Num_{Reg} of the realistic optimistic scheduling with explored potential time slack, the left edge algorithm is repeated by taking the $Num_{\text{Reg,max}}$ constraint into account. The scalars with small lifetimes are allocated in the available registers, whereas the scalars with long lifetimes are the first candidates for spilling to BG memory. Although the register spilling increases the latency and the number of FG accesses, it also reduces the maximal size of the simultaneously alive scalars and, thus, also the FG memory size, as desired. Based on the requirements different design points for the FG memory management are produced. The register width is $max(DType, VType, Rtype)$. The number of accesses, i.e. reads and writes, to the FG memory are computed by Eq. 6.13. The available FG memory bandwidth is also required to determine the ports of FG memory $Num_{\text{FG,Port}}$. However, we can postpone the computation of the available bandwidth as it is not essential for DP mapping. The number of ports and the final FG memory scheduling are decided after the data path mapping, where the required information is available.

$$Accesses_{\text{FG}} = \sum_{i=0}^{Num_{\text{i}}} \sum_{j=0}^{Scalars(i)} (WR_{\text{j}} + RD_{\text{j}}) \tag{6.13}$$

6.3.4 Step 4: Data Path Mapping & Final Design

The data path design and the connection to the FG memory are decided in this step. The pseudocode is depicted in Alg. 12.

6.3.4.1 Step 4.1: DP mapping

This step determines the data path design based on propagated design constraints of previous steps. When the parameter $Dependent = 0$, the DP of the HW accelerator executes both control and arithmetic operations. When $Dependent = 1$ and $Coprocessor = 0$, the DP executes only the arithmetic operations and the control is at the microprocessor. When the $Coprocessor = 1$, the size of the DP and the executed operations are based on the $Control$ parameter. If $Control =$

1, the control is executed on the co-processor. If $Control = 0$, the HW accelerator control is executed on the processor DP and the arithmetic operations on the HW accelerator DP. The size of the DP of both the processor and the HW accelerator should support the maximum length of the $Data$ and the $Variables$.

The FG memory management step propagates the kernel i potentially unrolled by the estimated UF_{FG}. The primitive operators are allocated, the kernel is scheduled and assigned on the single HW accelerator (a survey of techniques is available in [101]). The utilization of the primitive operators U_{HW} is given by Eq. 6.14 and it is used to decide on further unrolling to better utilize the HW operators. The scheduling and assignment step is reapplied with UF_{HW}.

$$U_{\text{HW}} = \frac{\sum_{i=0}^{Num_{\text{HWOps}}} Holes(i)}{Num_{\text{HWOps}} * Slots} \quad (6.14) \qquad UF_{\text{HW}} = \frac{1}{U_{\text{HW}}} \quad (6.15)$$

The critical path of the single HW accelerator CP_{HW} is computed by accumulating the latency of the operators in the critical path of the design. and the f_{HW} is determined ($f_{\text{HW}} = \frac{1}{CP_{\text{HW}}}$). If $f_{\text{HW}} \geq f_{\text{PC}}$, the f_{HW} is set equal to f_{PC} in order not to increase unnecessarily the area. If a further increase is required, it will be verified in the next step. If the $f_{\text{HW}} < f_{\text{PC}}$, pipelining is inserted to increase the f_{HW} up to f_{PC}. The number of pipeline stages PL (Eq. 6.16) is determined by a balanced split of the critical path. An unbalanced split is a less efficient trade-off option since the unbalanced ``fast" pipeline stages consume more energy than required. If the $CP_{\text{HW}} \leq t_{\text{HW}}$, where t_{HW} describes the available time left, the single HW accelerator design meets the real-time constraints. Otherwise parallelization across multiple HW accelerators (each with their own FG memory access ports) is considered to meet the real-time constraints. The Parallelization Factor PF is given by Eq. 6.17. The critical path is reduced (Eq. 6.18), since the different iterations are executed in PF HW accelerators.

$$PL = \frac{f_{\text{PC}}}{f_{\text{HW}}} \quad (6.16)$$

$$PF = \frac{t_{\text{HW}}}{CP_{\text{HW}}} \quad (6.17)$$

$$CP_{\text{HW}} = \frac{CP_{\text{HW}}}{PF} \quad (6.18)$$

If HW constraints exist over the PF and we still cannot meet the application timing requirements, the last option is to use a different frequency in the HW accelerator and the Microprocessor. This introduces additional overhead due to the required synchronization of the cores, which increases the area and the operations. However, it is a valid option when the other design options are over-constraint. A further exploration is to produce a design by selecting the next kernel in the sorted list of candidates to be assigned to a HW accelerator. The time required on the Microprocessor (which is in the overall critical path) is reduced and hence it allows to increase the available time for the HW accelerator execution. The objective function results of these two alternatives should then be compared to identify the most Pareto-optimal one. Hence, different pareto points are finally developed and the over-constraint ones are removed (as illustrated in Section 6.5).

6.3.4.2 Step 4.2: FG Memory Connection

This step determined the FG Memory Connection. The FG memory bandwidth is determined and the available bandwidth per register with one port is computed by Eq. 6.20. The required bandwidth derives from Eq. 6.19 and thus the required number of ports is given by Eq. 6.21. The allocation of ports and the final scheduling of the FG memory is performed, e.g. by using the technique in [16]. The final connections between FG registers and the accelerator logic are inserted [11].

$$RBandW_{\text{FG}} = \frac{Accesses_{\text{FG}}}{t_{\text{HW}}} \tag{6.19}$$

$$ABandW_{\text{FG}} = f_{\text{HW}} * Num_{\text{Reg}} \tag{6.20}$$

$$Num_{\text{FG,Port}} = \frac{RBandW_{\text{FG}}}{ABandW_{\text{FG}}} \tag{6.21}$$

6.4 Demonstrator Design: Real-Life Microfluid Application

This section illustrates the proposed methodology by deriving a near-optimal SW/HW mapping of a bioimaging application on a FPGA board. For different application characteristics, the proposed methodology develops different near-optimal designs which create the Pareto curve (Section 6.5).

6.4.1 Step 1: Application & Domain Analysis

6.4.1.1 Step 1.1: Platform Analysis

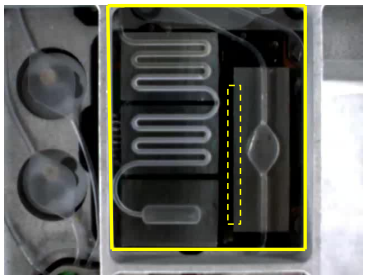
Our target HW platform is the Virtex-5 FPGA ML-507 Evaluation Platform with one Microblaze Soft processor set. A SDRAM DDR2 main memory, a data and an instruction cache of 16 KB and a local memory of 32 KB are used. The data and the instructions are fetched by the HW cache controller. The HW parameters are identified from the board characteristics, e.g. $W_{\text{Bus}_{\text{PC-HW}}}$=32 bit, $Bus_{\text{PC-HW}}$=16, $Av_{\text{Bus}_{\text{PC-HW}}}$=1 to 16 etc.

6.4.1.2 Step 1.2: Application Analysis

The demonstrator is a bio-image analysis used in a blood analysis application executed on a Lab-on-Chip (LoC) micro-fluid device. The image taken by the FPGA camera is depicted in Fig. 6.3(b). During the set-up, the frame (the continuous box of Fig. 6.3(b)) and the coordinates of the micro-fluid pipes are detected. Based on the application specifications, the frame can be rotated only $\pm 3^{\circ}$. During the LoC device normal function, an angle detection algorithm and a detection of the fluid's fronts coordinates algorithm is executed in each frame. The pseudocode of the application is depicted in Fig. 6.3(a). The fluid velocity is computed based on the coordinates of the fronts determining the provision of required liquid quantity. The angle detection algorithm requires only the vertical line in the window where it is applied (dotted box of Fig. 6.3(b)). It applies the Canny algorithm for finding the intensity gradient of the image using a horizontal contrast 3x3 Sobel

```
uint8 f[N][M];
int Coordinates, Mask[3][3], Gx[M]; Angle[M];
if (Set-up):
  Detect Frame position(f);
  Identify pipes coordinate(f);
/*AngleDetection*/
for (row=1; row<N-1; row++):
  Initialization();
  for (col=1; col<M-1; col++):
    for (rowOff=-1; rowOff≤1; rowOff++):
      for (colOff=-1; colOff≤1; colOff++):
        Gx[col] +=
        f[row+rowOff][col+colOff]*Mask[rowOff+1][colOff+1];
  Gx[col] = abs(Gx[col]);
  if (Gx[col]! =0):{Angle[col]=90;}
  else:{Angle[col]=0;}
  if (Angle[col-1]==90):
    if (Gx[col-2]<Gx[col] || Gx[col-1]<Gx[col-2] )):
      if (Gx[col-1]>Threshold): {HoughAccum.(row,col);}
LineSuppression;
FluidDetection(f);
```

(a)



(b)

Figure 6.3: Real-time bioimaging application:(a) Pseudocode and (b) image taken by the micro-fluid device camera: the box is the outline frame and the dotted box is the angle detection window.

(middle column multiplicands are 0) and a Hough transform version, since the vertical line computations are required with small angles, e.g. $[-3°, +3°]$. If the Sobel kernel's result is an edge point, the Hough transform maps it to the Hough space and stores the results to the accumulator matrix. The final line is detected by suppressing the neighborhood lines. The fluid coordinates are derived by subtracting the micro-fluid pipes of two successive frames and by computing the centroid of the result. The SW parameters are defined, e.g. for the intensity gradient kernel $Data$=6, since 6 pixels are required for the horizontal contrast 3x3 Sobel mask, Res=2 for the angle and the gradient, $ResType$=32 bits etc.

We demonstrate the proposed methodology for a 200x16 window and FR=100 frames/sec. The throughput is given by the video frame ratio, i.e. TP=100. The D=10 msec to execute the angle detection algorithm and the algorithm for the detection of the fluid fronts, $t_{\text{Tot}} = t_{\text{Angle}} + t_{\text{Fluid}}$.

6.4.1.3 Step 1.3: Decide SW & HW execution

The Microblaze frequency is set to the maximum allowed, i.e. f_{PC}=$max(Av_{\text{f}_{\text{PC}}})$= 125 MHz, t_{Fluid}=4.87 msec and the t_{Angle}=2.87 msec. The SW design meets the deadline constraints and thus no HW accelerator is required. To demonstrate the next steps of the proposed methodology we use a frequency of f_{PC}=83,33 MHz. In this case, the platform is compatible with Avnet Spartan-6 LX150T Development Kit, where $max(Av_{\text{f}_{\text{PC}}})$=83,33 MHz. The detection of the fluid fronts' coordinates depends on the number of fluids fronts and the frame resolution. For small frame resolution (640x480) and 2 fluids fronts, it executes in 1.23 msec in SW. For the minimum quality application specifications (small frame and 3 edges) the average execution time is estimated at 3.9 msec and for the maximum quality (1024x1024 and 7 edges) is 7.3 msec. The angle detec-

tion requires 4.31 msec (Subsection 6.4) when executed on the Microblaze Soft microprocessor. The execution time of the application for a 200x16 window is t_{Tot} = 4.31+(3.9 to 7.3) msec=8.21 to 11.61 msec. Hence, real-time behavior is not always achieved ($t_{\text{Tot,MinQ}} < D < t_{\text{Tot,MaxQ}}$), and thus SW/HW designs are required.

6.4.1.3.1 Step 1.3.1: Application Constraints We apply strength reduction to remove the costly multiplications by constant, e.g the constant values of multiplications are analyzed, the multiplications with the value of 1 and 0 are removed and the remaining ones are replaced by Shift and Add operations. Based on application profiling the most time consuming regular task is the angle detection algorithm and thus its critical kernels should be explored for mapping on HW accelerators. Since the application time exceeds the D for a small value, it is possible by mapping part of it to the HW accelerator to meet the real-time constraints. The bandwidth required to transfer the data for the kernels to the microprocessor $BandW_{\text{CDFG}}$=227,556 bits/msec and the bandwidth provided by the BG memory of the platform in the optimistic case $ABandW_{\text{BG}}$=10,666,240 bits/msec.

6.4.1.3.2 Step 1.3.2: Kernel Constraints To safely meet the real-time constraints, the available time for executing the angle detection is $t_{\text{Angle}} = D - t_{\text{Fluid}}$=10.0-(3.9 to 7.3) msec=6.1 to 2.7 msec, and the worst-case is considered, i.e. t_{Angle}=2.7 msec. Based on the profiling of the application analysis, the main loop takes 68% to execute the kernel for the intensity gradient of the image and the kernel for creating the Hough accumulator array. The execution time of the two kernels is similar (31% and 33% respectively) with the intensity gradient kernel to be regular. The application of the horizontal 3x3 Sobel mask is the 90% of the intensity gradient kernel, so it is the first candidate for implementation in HW ($Regular_{\text{Sobel}}$=1). When a slack of 10% is used for the non-critical part of the code, the available time is t_{HW}=0.9*(0.9*0.31*2.7)= 0.677 msec. The data required to be transfered are $Data$=614,400, when 32 bits are used to store the data of the image and the required bandwidth to transfer the data is $BandW$=906,235 bits/msec. The available bandwidth of the BG memory $ABandW_{\text{BG}}$ is 10,666,240 bits/sec., when 128 bits are transfered and 2,666,560 bits/msec when 32 bits are transfered. The available bandwidth to transfer the data from the microblaze to the HW accelerator is $ABandW_{\text{PC-HW}}$=42,664,960 bits/msec when all Fast Simplex Links (FSLs) are used in parallel. An estimation on the optimal critical path is $CP_{\text{HW,Opt}}$=0.028 msec without area constraints using a $Latency_{ADD/SUB/COMP}$=2 nsec from the FPGA platform. Hence, mapping the most critical kernel with $Regular = 1$, i.e. the 3x3 Sobel mask, on HW accelerator is enough to meet the real-time constraints.

6.4.2 Step 2: Microprocessor & HW Accelerators Organization

Since the design objectives are minimizing the area and thus reducing the energy consumption while the real-time constraints are met, the HW accelerator is efficiently connected through FSL to the Microblaze Soft Processor as a co-processor ($CoProcessor = 1$ and $Dependent = 1$).

The Microblaze is responsible for the synchronization and the loop organization between the components ($Control = 0$). Hence, the application loops are modified accordingly to support the common control of both cores. The Microblaze provides the appropriate information for one execution of the co-processor in the corresponding FSL, i.e. the required pixels accessed from the memory. Then the co-processor reads the data, executes the operations and writes the results, i.e. the gradient and the angle, to the FSL. The FSL width should be $W_{\text{Bus}_{\text{PC}\rightarrow\text{HW}}} = 32bits$ to support the maximum width of transfered data. The required bandwidth to transfer the data within t_{HW} is 906,235 bits/msec and the available bandwidth in the optimistic case, i.e. 16 FSL are used, is 42,664,960 bits/msec, which is sufficient. The time required to transfer the data is t_{TR}=0.014 msec when all FSL are available. The bandwidth of one FSL is 2,666,560 bits/msec. The minimum required parallel transfers is 1 and the transfer time is 0.23 msec.

6.4.3 Step 3: Foreground Memory Management

The cost operation analysis does not modify the kernel, since the operations after strength reduction are simple and highly used. In this case study, the inner kernel is mapped to a primitive operation. The HDFG is the degenerated case of 6 registers for the inputs, each one with 1 write and 1 read operation to the FG memory, and 2 registers for the results, each one with 1 write and 1 read. The registers width is 32 bits to support the maximum data type. The total $Accesses_{\text{FG}}$=56,000 and one register port provides 83,330 accesses/msec. The FG critical path is estimated by 11,200 accesses and the lower allowed frequency (f_{PC}=83,33 MHz) at 0.134 msec.

6.4.4 Step 4: Data Path Mapping & Final Design

6.4.4.1 Step 4.1: Data Path Mapping

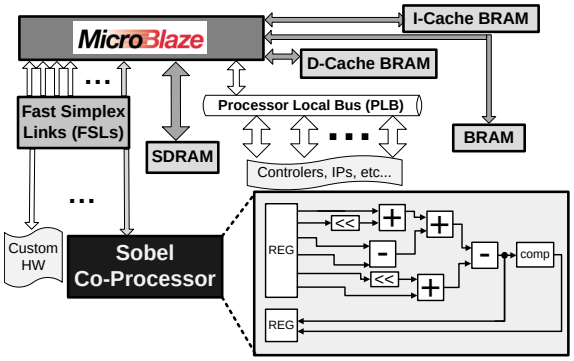


Figure 6.4: The HW platform architecture and the final design for the Microblaze and the HW accelerator of the demonstrator application.

The control of the loops, the FG memory scheduling and the initialization code are executed on the Microblaze DP. The primitive operator is described by the arithmetic operations executed on the co-processor, i.e. $OPs = Num_{\text{SHIFT}} + Num_{\text{ADD/SUB/COMP}}$=2+6, which are implemented by 2 Shift FUs (S-FUs) and 6 Add/Sub/Comp FUs (A-FUs). To reduce the area, a S-FU can

be combined with an A-FU in one Shift-Add (SA) FU and thus 4 SA-FUs and 8 A-FUs. The DP bus width should support the result and the operands of the primitive operation, i.e. 32 bits. Further unrolling is not required since full utilization of the primitive operations is achieved. The CP_{HW}=0.0177 msec and f_{HW}=157,679 MHZ. The frequency is set to f_{PC} to avoid extra synchronization and meet the real-time constraints, CP_{HW}=0.0336 msec. When the data transfer and the computation are executed sequentially the total time is 0.628 msec.

6.4.4.2 Step 4.2: FG Memory Connection

The required FG bandwidth $RBandW_{\text{FG}}$ is 82,600 accesses/msec, the available FG bandwidth of one port is 83,330 and thus 1 port is sufficient to meet the bandwidth.

6.5 Experimental Results

In this section, we show a broad range of different designs derived from the proposed methodology based on the application characteristics to create a Pareto curve for mapping of the application to the FPGA. The performance is measured by execution and the microcode provided by SDK for ML-507 cx5vfx70t-ff1136 platform, the HW accelerator area from XST and the Total area from the XPS EDK Xilinx tool.

6.5.1 Real-Life Microfluid Application

The angle detection algorithm is applied for a 640x480 frame resolution, a 200x16 window and 100 frames/sec. The results for the different designs are depicted in Table 6.3. The SW Sobel MUL in the execution of the reference angle detection routine with multiplications on the Microblaze Soft processor, the SW/HW MUL design puts the critical kernel in a co-processor with multipliers and it is representative for the existing state-of-the-art HW/SW FPGA mapping techniques (Section 6.2). The estimated results based on the microcode provided from the Xilinx compiler lead to at least 225,900 cycles. The extra area is quite large, i.e. 32 slices and 12 DSP48e Slices. The DSP48e Slices are more complex and the total area will be significantly larger. A lower bound is computed based on $DSPSlices = 4 * CLBTot = 4 * 2 * Slices$, i.e. $\approx$ 128 slices. The SW/HW-1FSL design derived from our methodology achieves gain of 47.11% and the extra HW area is 231 slices. The proposed SW/HW design hides the overhead of the address generation and the memory accesses as they are executed by the processor, while the co-processor computes the set of the Sobel masks. The DP of the co-processor includes no idle cycles during the mask execution, since the data are already available and efficient mapping of the operations to the co-processor is achieved. To compare the proposed design, we insert the HW design with the memory management performed by the Microblaze with a lower bound of 69,484 cycles and of 375 slices. The most performance-efficient design is the application-specific HW with custom memory management and DP dedicated to the specific application, which requires a very large

Table 6.3: Performance & area for demonstration case study.

Design	Performance 200x16 Window		Performance 300x75 Window		Area	
					HW Acc.	Total
	Cycles	Time(ms)	Cycles	Time(ms)	Slices	Slices
SW MUL Ops	358,996	4.31	1,458,006	17.50	0	4210
SW/HW MUL	231,444*	2.78*	765,901*	9.19*	≈128*	4284*
SW/HW-1FSL	189,864	2.28	439,591	5.27	114	4255
SW/HW-6FSL	176,004	2.11	330,821	3.97	114	4268
SW/HW-2PF-1FSL	174,618	2.09	319,944	3.84	231	4374
SW/HW-2PF-6FSL	167,688	1.98	265,559	3.19	231	4385
SW/HW Sobel&Hough-6FSL	98,705*	1.19*	196,576*	2.36*	314*	4505*
HW & Microblaze M.M.	>70,000*	>0.84*	>182,638*	>2.19*	>375*	>4581*
HW & Custom M.M.	>46,000*	> 0.56*	>142,218*	>1.77*	>550*	>4721*

*estimated

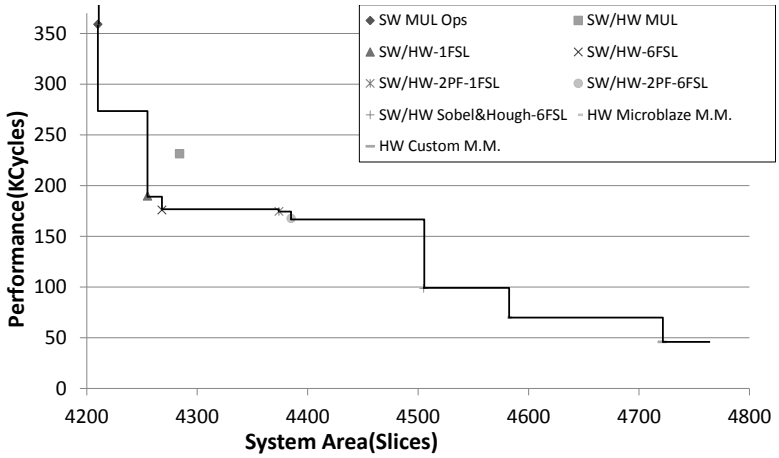


Figure 6.5: Pareto curve for 200x16 window.

design effort and time. Those are not reusable across different applications so the NRE cost will be shared for a relatively small market volume. For deep sub-micron process technologies with hugely increasing NRE costs, that is a clear disadvantage. The proposed methodology design achieves gain of 79.29% in area compared to the conventional HW design.

We modify the application requirements and briefly describe the different SW/HW designs derived from the proposed methodology to compose the Pareto Curve depicted in Fig. 6.5. When the frame rate is increased to 105 frames/sec, D=9.52 msec, t_{Angle}=2.22 msec and t_{HW}=0.558 msec (Alg. 9). The time estimation of transferring the data is 0.23 msec from the BG memory (Eq.6.1-Eq.6.3), 0.23 msec to the co-processor (Eq.6.4-Eq.6.8) and 1.168 msec is the critical path of the FG and the DP (Alg. 10). Then, parallel FSL (Eq. 6.10) are required to transfer the data to decrease the transfer time to 0.038 msec (SW/HW-6FSL). When the frame rate is increased to 115 frames/sec, t_{Angle}=1.39 msec and t_{HW}=0.351 msec. The time estimation is 0.23 msec for transferring from the BG memory (Eq.6.1-Eq.6.3) and to the co-processor (Eq.6.4-Eq.6.8), which exceeds the available time. Hence, parallel FSL are required (Eq. 6.10). Even then the required time is 0.439, which exceeds the available time. Parallelization is explored with a factor of PF=3 (Eq. 6.17). The total time is estimated at 0.32 msec (Alg. 12). Since the PF is highly increased, the option of mapping the second kernel to a HW accelerator is explored. Then, the t_{HW}=0.765 msec, the $BandW$=1,271,706 bits/msec (Eq. 6.6), the estimated critical path of the Hough kernel is $CP_{\text{HW,Hough}}$=0.039 msec and the total critical path based on the dependencies is

Table 6.4: Performance & area for PolyBench Benchmark Suite.

Bench.	Design	Performance (Cycles)	HW Acc.					
			Primitive Operations	CP (ns)	f (MHz)	Slices	DSP Slices	Total Slices
2mm	SW	6,000,208	-	-	-	-	-	-
	SW/HW	2,155,152	1-2PL	10.17	98.32	38	6	134
			2	10.11	98.87	24	3	
	SW/HW-2PF	1,298,043	1-3PL	10.44	95.8	71	12	263
			2	11.49	87.06	48	6	
3mm	SW	9,067,317	-	-	-	-	-	-
	SW/HW	3,227,592	1	10.13	98.67	22	3	46
	SW/HW-2PF	1,832,904	1	11.48	87.08	50	6	98
Bicg	SW	128,678	-	-	-	-	-	-
	SW/HW	48,256	1	10.11	98.91	42	6	90
Gemm	SW	3,648,404	-	-	-	-	-	-
	SW/HW	2,246,593	1-2PL	10.22	97.81	41	6	89
	SW/HW-3PL	2,190,881	1-2PF	10.65	93.9	73	12	169
Gesummv	SW	143,301	-	-	-	-	-	-
	SW/HW	70,753	1	10.11	98.87	24	3	48
	SW/HW-2PF	65,633	1	11.49	87.06	48	6	96
Atax	SW	139,420	-	-	-	-	-	-
	SW/HW	80,790	1	10.13	98.67	22	3	46
Cholesky	SW	374,334	-	-	-	-	-	-
	SW/HW	70,753	1	10.31	96.99	25	3	49
Gemver	SW	251,550	-	-	-	-	-	-
	SW/HW	96,650	1	11.48	87.08	50	6	203
			2-2PL	10.17	98.32	39	6	
			3	2.67	374.96	18	0	
Jacobi_2d	SW	1,333,510	-	-	-	-	-	-
	SW/HW	303,464	1	6.59	151.86	80	0	80
Correlation	SW	4,413,504	-	-	-	-	-	-
	SW/HW	1,084,299	1	2.65	376.65	16	0	1194
			2	11.91	83.97	32	6	
			3-3PL	44.82	22.31	1028	3	
			4	10.13	98.67	22	3	

SW: Software, SW/HW: Co-design, PF=Parallel Factor, PL=Pipeline stages

$CP_{\text{HW,Tot}}$=0.082 msec. When 1 FSL is used (Eq. 6.10), the total time exceeds the available. With 6 parallel transfers for the Sobel kernel and a sequential transfer for the Hough, the total estimated time is 0.675 msec. The window of applying the angle detection is increased (300x75) and 8 bits are used to store the data. Then, more cycles are required for the execution and less bandwidth for the transfer. For 60 frames/sec, D=16.06 msec, t_{Angle}=9.36 msec and t_{HW}=2.351 msec. The required bandwidth is $BandW$=115,303 bits/msec (Eq. 6.6), the estimated time for BG transfers is 0.40 msec (Eq.6.1-Eq.6.3), for parallel FSLs is 0.067 msec (Eq.6.4-Eq.6.8) and the critical path is 2.33 msec. Parallelization is explored (with PF=2 (Eq. 6.17)) to meet real-time constraints.

Industrial design practices with experienced designers will potentially also reach these results, but with substantial design effort and without the guarantee of systematically finding the relevant Pareto points.

6.5.2 PolyBench Benchmark Suite

The PolyBench benchmarks [159] is a polyhedral benchmark suite with static control parts and has as purpose to uniformize the execution and monitoring of kernels. It includes linear-algebra, data-mining, medley and stencils kernels. The PolyBench benchmarks are usually used as parts of more complex applications with real-time constraints, e.g. matrix matrix multiplication is used in signal processing applications, jacobi is used to determine solutions of linear equations etc. A

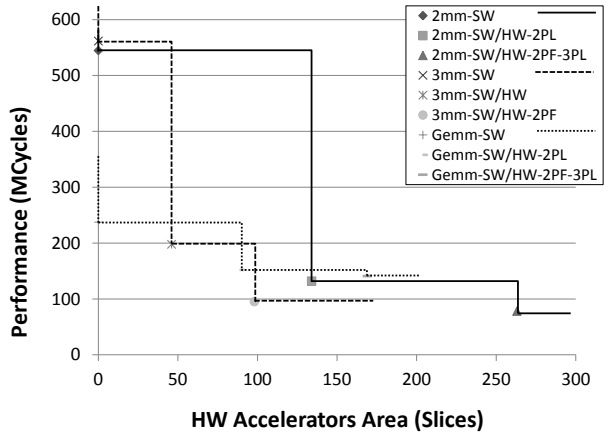


Figure 6.6: Pareto curves for 2mm, 3mm and Gemm for data size 128.

Table 6.5: Comparison results for the proposed and the iterative improvement approach.

Algorithm	Iterative Improvement			Proposed Methodology		
	Perf. (ms)	Area (LUT)	Time (ms)	Perf. (ms)	Area (LUT)	Time (ms)
Seidel-2d	0.02020892	72	118,686.089	0.02020892	72	1,125.692
	0.02024000	64		0.02022202	64	
	0.02524778	56		0.02521426	56	
	0.02526088	48		0.02521426	48	
	0.03525846	40		0.03062306	40	
	-	-		0.04059266	32	
Gemm	0.1609017	40	50,451.987	0.1609017	40	256.695
	0.1609075	32		0.1609075	32	
	0.2418297	24		0.2418297	24	
Syr2k	0.3217926	40	55,155.037	0.3217926	40	265.931
	0.3217984	32		0.3217984	32	
	0.48365575	24		0.48365575	24	

selected set of SW reference designs and SW/HW co-designs for 10 different PolyBenchmarks derived from our methodology is depicted in Table 6.4. The Pareto Curves for a set of PolyBenchmarks are summarized in Fig. 6.6. They show the effectiveness and broad applicability of our approach. Notably we also produce a wide range of Pareto working points which are crucial in practical design contexts where trade-offs typically exists between the different objectives.

6.5.3 Relative Comparison

Existing methodologies and frameworks mainly use different architectural assumptions and valid options in the designs. To provide a useful comparison in terms of performance, area and the exploration time, we have implemented an Iterative Improvement (II) approach, similar to the existing in the current literature (as much as possible mimicking it). The comparison is performed for one of the most complicated steps of our methodology, i.e. the ForeGround memory management step. The remaining steps have in the worst case similar behavior and thus the relative comparison remain representative for the other steps also. The II approach applies a register scheduling and assignment step and an improvement step. The first step is based on the mobility of the FG memory operations. After the first step and the scheduling and assignment of the selected operations, the improvement step is applied. The improvement step selects different nodes

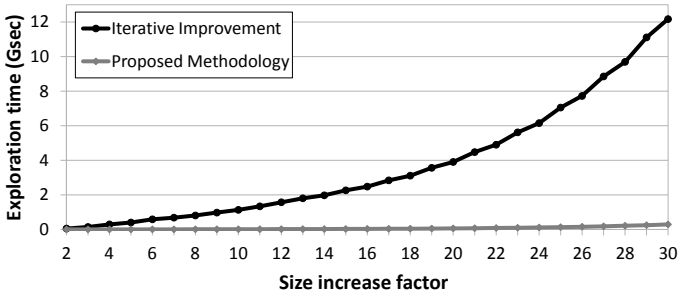


Figure 6.7: Exploration time comparison when the application size is increased by a factor.

in previous iteration steps to search for potential improvements. The results of our approach and the II approach for a set of test cases are depicted in Table 6.5. For the seidel-2d benchmark the II has less optimal results because it selects to schedule later a node whose successors affect the critical path, as it has the same mobility with the other ready to be scheduled nodes. The proposed methodology uses different types of nodes. It schedules and assigns based on the node type and unidirectionally propagates constraints, i.e. scheduling decisions, to the nodes of the next-to-be-scheduled type. In this way it can identify points, which are outside the local scope of the II. For smaller benchmarks, the II potentially has similar quality with the proposed approach but the points require a much higher exploration time to be produced. When the number of nodes in the application graph is increased the exploration time of the II is strong polynomially increased and the proposed methodology remains linear (Fig 6.7).

6.6 Conclusions

A systematic stepwise template-based methodology is described to compose a Pareto Curve for near-optimal mapping of an application to a SW/HW design with a processor and a (set of) HW accelerator(s) taking into account the SW/HW organization, the FG Memory Management and the DP Mapping. The suboptimal options are pruned early in the design process based on scalable what-if analysis and the constraints propagation of each step leading to a scalable and efficient approach.

Chapter 7

Design-time scheduling techniques framework

7.1 Introduction

The scheduling problem is an optimization problem with fundamental principles applicable in several fields [141], e.g. computer science, economics, job scheduling, project management, production e.t.c. Both research and development community have already invested decades of research and experiments to the techniques solving several instances of the scheduling problem in computer science. Important advances have appeared to the scheduling problem as it applies to design time mapping on multiprocessing platforms emphasizing on ordering in time and assignment in place. However, it cannot be considered as fully solved yet. Publications are becoming available and significant space for improvement is present, as e.g. explained in [182].

The prerequirement to achieve this improvement is the understanding of the scheduling approaches. As it is explicitly stated in [182], the ``Better understanding of the behavior of search algorithms in scheduling search spaces should ultimately lead to development of more effective scheduling procedures". This understanding is supported by effectively classifying the scheduling techniques. A complete classification provides a thorough overview of all the available and diverse categories in the wide variety of existing and future scheduling techniques. It efficiently partitions the techniques into classes with unique and verifiable characteristics. The main features, the similarities and the differences of the distinct categories are clearly illustrated. It supports the meaningful evaluation and comparison of techniques with similar characteristics. In this way, an in-depth understanding of each category and of their interrelationships is acquired. The latter assists in contributing in novel ways to the broad domain of (near-) optimal scheduling techniques. The existing techniques can be improved, the less explored areas can be identified and new techniques can be proposed.

The literature presented until recent years has relatively few classification proposals ([125], [126], [17], [151], [152], [171], [141] and [60]) on design time mapping on single and multiprocessing platforms emphasizing on ordering in time and assignment in place. They are

less systematic and thus relatively incomplete, partially redundant lacking efficiency. Other recent studies ([114], [79] and [182]) focus on addressing the advances instead of proposing a new complete classification. The existing approaches are capable of partially covering the entire space of scheduling techniques in a less guaranteed way and less efficiently supporting their understanding and improvement (Section 7.3).

In this chapter, we propose a new classification scheme for design time mapping techniques on single and multiprocessing platforms focusing on ordering in time and assignment in place. The classification is derived by using the reusable DSE principles of the framework creation described in Chapter 2 in the context of global design-time scheduling techniques. The context does not include operation system or frequent run time and preemptive approaches though. Our main contribution is to present a new systematic way of classifying the scheduling techniques. We present and describe the output of our approach and we show that the proposed classification systematically provides a complete overview of the existing distinct categories of techniques. We also show why our approach will remain valid for future techniques as long as they belong in our target domain. Our second contribution is to illustrate how the proposed scheme efficiently classifies representative examples which cover the exploration space. The proposed classification encounters a technique as a potential combination of classes and it decomposes it to primitive components which belong to different classes. In the Chapter 7, the derived framework, i.e. the proposed classification scheme, is used as a basis in order to develop parametric templates for design-time scheduling techniques for a target domain, which is an instantiation of the context of the classification.

In Section 7.2 we describe the context, the problem formulation and the goal of the classification framework. Section 7.3 presents existing classification schemes of scheduling techniques and motivates the need for a new systematic classification approach. Section 7.4 presents the derived classification framework after applying the reusable DSE principles in the context of global design-time scheduling techniques, describes the classes and the uni-directional arrows. Section 7.5 illustrates and verifies the classification scheme through several representative examples.

7.2 Target domain and problem formulation

This Section describes the bounds of which scheduling techniques are under study by the proposed classification. This is achieved by defining the properties of the scheduling problem solved by the under study techniques and the assumptions made for the scheduling techniques. This is a crucial step as it determines the initial class, i.e. the root of the proposed classification tree.

An application is modeled by a graph $T = \{T_i : i = 1, ..., N\}$, or a set of graphs T, with N partitions describing individual operations or group of operations, memory references or communication transactions of the application. The edges of the graph(s) describe the control and data dependencies of the application. A set $R = \{r_i : i = 1, ..., M\}$ of M same resources describes each resource type. A hardware resource may refer to homogeneous or heterogeneous, coarse or fine grained processing elements, e.g. processors or function units respectively, memories or

intercommunication networks. In this way, a quite generic computing platform is considered.

The constraints of the problem are specified by a set $D = \{d_i : i = 1, ..., K\}$, where d_i describes the timing and resource constraints. Constraints are potentially imposed by the environment, e.g. a deadline before which the task must finish its execution, by the task, e.g. ordering in data fetching and operation execution, and by the different computing platforms, e.g. number of interconnections, storing and execution components.

The objective function is given by a set $C = \{w_i * c_i : i = 1, ..., L\}$, which describes the crucial c_i elements of the system weighted properly by a w_i constant. Crucial objectives can be the number of control steps required to execute the task parts, the total number of required hardware modules, the energy consumed by the execution of the task parts to a given platform, the area of the required architecture etc. and also combinations of different objectives.

The mapping problem is the assignment of a finite set of task parts T onto the sets of different resources R_j (assignment in space) and the determination of the start time of the execution of a task part $CS = \{cs_i : i = 1, ..., N\}$ on the assigned resource (ordering in time) in a way that all dependencies and constraints D are satisfied and the objective function C is minimized. The assignment of a task part to a resource and its execution start time is described by $b_i = \{T_i, r_i, cs_i\}$.

A solution is defined by a set of N assignments $X = \{b_i : i = 1, ..., N\}$. The exploration space S is all the possible combinations of N assignments. A solution is feasible when it satisfies all dependencies and constraints D. The feasible explorations space F is all the combinations of feasible solutions. The mapping problem is to find a solution X^* which belongs to F and minimizes the objective function C, i.e. $C^* = min\{C\}$.

This study focuses on the classification of the existing and future global techniques solving design time mapping problem focusing on ordering in time and assignment in space. A technique is global when it is capable of searching the optimal solution X^* or a near-optimal solution in the complete exploration space. The non-global techniques are omitted, i.e. the techniques which do not intend to reach the optimal solution or guarantee that potentially optimal areas are excluded from their search space. The context of the problem under study is described by:

1. The systematic classification refers to design time mapping techniques focusing on ordering in time and assignment in space.

2. The classified scheduling techniques search for an (near-)optimal solution, thus the techniques are global, i.e. they potentially search in the entire exploration space. Otherwise they could not be (near-) optimal scheduling approaches.

3. The constraints met by the techniques are imposed by the specific context of the scheduling problem to be solved. They refer to timing constraints occurring in the present or near future and to resource constraints of the platform they are executed on.

4. The objectives (costs) for the solution techniques are not restricted to one dimension, but they potentially express different axes which compete one another formulating a multi-dimensional objective space.

The target domain (context) of the proposed classification focuses on scheduling techniques applied to problems where the major characteristics of the application are known and execution time of the scheduling technique is not a constraint [208], i.e. design time (offline) and infrequent set-up time. The latter case describes techniques applied in pre-run time or techniques which can be applied infrequently during the execution to reschedule the application in a near-optimal way. The classification refers to design time and infrequent set-up time techniques which solve Instruction Level Parallelism (ILP), Task Level Parallelism (TLP) and Data Level Parallelism (DLP) mapping problem on generic single and multiprocessing platforms. It also applies in Memory Level Parallelism (MLP) on distributed memory platforms and in communication transactions ordering in time and assignment on communication networks.

Techniques dedicated to scheduling problems where the major characteristics of the processes are unknown and with very hard time restrictions, where a new different task arrives very frequently and rescheduling of the application is required at fast run-time or on-line, are excluded from our study. Run time decisions to find (near-) optimal solutions within a limited available time do not meet our assumptions. This target domain of scheduling techniques has additional properties, e.g. it strictly requires reduced execution time to achieve timing constraints of the application. However, parts of these very dynamic techniques are included in the design time and infrequent set-up time classification, due to the nature of the scheduling techniques. But since the current study does not focus on fast run-time and online domain, guarantee on completeness for this domain is not claimed.

7.3 Related work in global scheduling classifications

The current literature consists of less systematic classifications of global scheduling techniques which may belong to one of the two categories: a classification with independent classes, which do not allow any class interaction, or a classification with partial interaction between different parts of the classification, i.e. classes and policies. Survey type studies also exist, but they only present the new advances in this field.

The previous classifications of global scheduling techniques are less systematically composed and thus they have several limitations: a) they are partially complete approaches since they exclude several existing well-known scheduling classes, b) they are less consistent, as they include unrealistic scheduling techniques in their scheme, c) they are partially redundant, as the characteristics of a class overlap with characteristics of other classes confusing the classification process, d) difficulties exist during the classification of several techniques leading to less useful schemes which support only a partial understanding of the scheduling techniques methodology. These limitations are explained and illustrated through examples for existing classifications in the next paragraphs. Other recent studies are mostly of survey type addressing the new advances of scheduling techniques without indenting to update the previous classifications or to propose new complete ones. Hence, an updated complete and efficient classification of the wide variety of the global scheduling

techniques is highly desired to systematically support the understanding and improvement of the techniques, the identification of less explored areas and the proposal of new promising approaches.

One category of the existing classifications consists of independent classes which do not interact with each other. One example is the classification of [126] and [125] depicted in Fig. 7.1, which is adapted by a majority of publications [43], [167], [119], [187], [74] and [198].

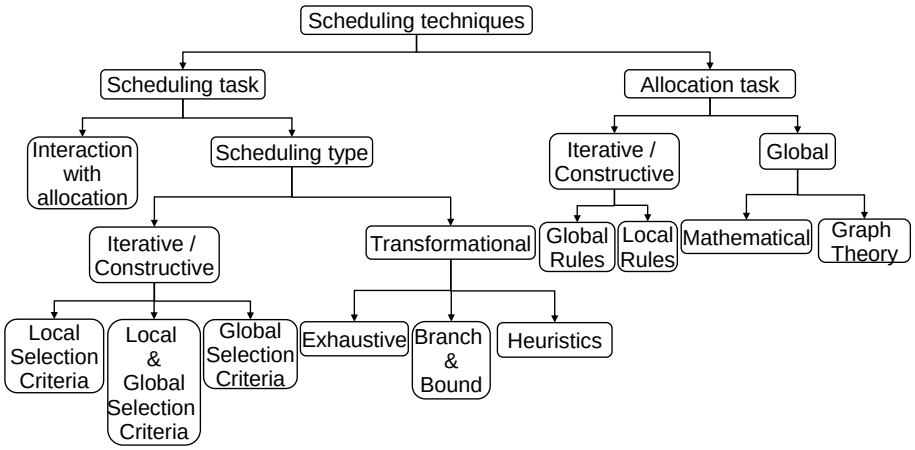


Figure 7.1: A classification scheme of the scheduling techniques consisting of independent classes presented by McFarland et al.

The ``scheduling'' techniques are divided into the transformational and the iterative/constructive. A transformational technique begins with a default maximally serial or maximally parallel schedule and applies transformations, which move serial operations in parallel and visa versa, to obtain other schedules. Based on how they choose what transformations to apply they are divided into the exhaustive, the branch-and-bound and the heuristics. The exhaustive try all possible combinations of serial and parallel transformations [9]. The branch-and-bound cut off the exploration space by recognizing suboptimal paths. The heuristics guide the process to choose the transformations which promise to move the solution closer to optimal [154]. The iterative/constructive techniques add operations one at a time until all operations have been scheduled. Based on how they choose the next operation to be scheduled and on how they determine where to schedule each operation they are divided into a local selection criteria, a global selection criteria and a local affected by more global operation selection criteria class. For instance, the ``As Soon As Possible'' (ASAP) technique belongs to the local class [71]. The freedom-based and force-directed [151] techniques reside in the global class. The list scheduling techniques [124] belong to the local class with global criteria. The ``allocation'' techniques are divided into iterative/constructive and global techniques. The iterative/constructive techniques assign elements one at a time selected based on global [71] or on local rules [93]. The global techniques assign a number of elements each time. They can be mathematical techniques, i.e. exhaustive search [64], or graph theoretic techniques, i.e. clique partitioning [190].

This approach separately classifies the ordering in time task, ``scheduling'', and the assignment in space, ``allocation'' as defined in Section 7.2. Although it also includes classes describing non-global techniques, we consider the global classes of this approach in our study.

This classification is partially complete, since it excludes several classes that describe global techniques. Several stochastic classes are not implicitly included, e.g. Tabu search ([168], [50], [53]), Seed techniques [144], Simulated Annealing (SA) ([145], [137], [43]), Genetic Algorithms (GA) ([112], [44], [184], [67]) and Simulated Evolution (SE) [119]. The classification of these techniques using this classification is ambiguous, since the class they belong to is missing. For instance, the SE technique can be viewed as ``a constructive synthesis with the ability to escape from local minimum or it may be viewed as transformational synthesis in which arbitrarily complex design transformations are generated on the fly" [119]. Hence, it may belong to the transformational heuristic class, where normalized cost and freedom based priority guides the transformations to be applied. But it may also be assigned to the iterative/constractive global class, where it starts from a partial solution based on normalized cost and where freedom based priority scheduling is applied. The GA based techniques may be classified in the transformational heuristic class, where schedule decoding and fitness guide the applied transformations. But they may also be assigned to the iterative/constractive global class, where schedule decoding selects the operation to be scheduled. Other important scheduling categories are also excluded, e.g. partitioning, clustering and adaptive techniques. For instance, the technique in [150] is classified to the local with global selection criteria class. However, the partitioning characteristics of this technique are not described. The technique [109] belongs to the local with global criteria class but the clustering characteristic is not included. The learning process of the adaptive technique of [69] is excluded from the classification tree. Therefore, the way to extend this classification to include the techniques whose class is ambiguous, is currently too vague.

Another limitation derives from the less systematic composition of the classification. For each different combination of main characteristics of a technique a new class is created. In this way, characteristics of one class may overlap with those of other classes leading to a partially redundant and non-compact scheme. For instance, the local with global selection criteria class has overlapping characteristics with local selection criteria and global selection criteria class.

Difficulties and non-trivial decisions appear during classification of techniques. Hence, a useful understanding of the techniques cannot be obtained and a less meaningful comparison of existing techniques is achieved. This classification encounters the techniques as one component and does not allow their systematic decomposition. Hence, they classify hybrid techniques in one class. For instance, the formally optimal techniques, e.g. [74], belong to the exhaustive class. The hybrid technique in [179] is also classified to the same class, although it has additional recursive characteristics that distinguish it from a classical exhaustive technique. An approach that decomposes the techniques into more primitive components to express their main characteristics is missing, e.g. the rigid and the optimal class for [179]. The pruning techniques [61], [128], [134] can be partially classified to the Branch and Bound (B&B) class, as this class expresses the way to reduce the solution space. However, their fundamental methodology is different from a typical B&B technique. Section 7.5 illustrates through examples how the limitations of the existing classifications are solved by the proposed approach.

Other classifications belonging to the same category have also been proposed in the literature.

But they usually restrict their scope and focus only on a part of the global scheduling techniques. One instance is the scheduling classification of [60]. The first proposed split is based on the instantiation of the scheduling problem to be solved heavily restricting the generality of the techniques. And a significant part of this classification is dedicated to non-global techniques. The approach of [152] and [151] is dedicated only to one part of the overall scheme of the scheduling techniques, i.e. the non-global techniques. The approach of [171] addresses on the stochastic part while the taxonomy [141] focuses on the different Artificial Intelligent (AI) approaches. The review in [65] is dedicated to evolutionary techniques. The taxonomy in [104] focuses on the task graph, the cost, the communication and the resource constraints of the techniques.

The other category of existing classifications consists of a part of independent classes and a part of policies which are allowed to interact with the classes. One instance is the classification depicted in Fig. 7.2 as it applies to distributed resource management scheduling problem [17].

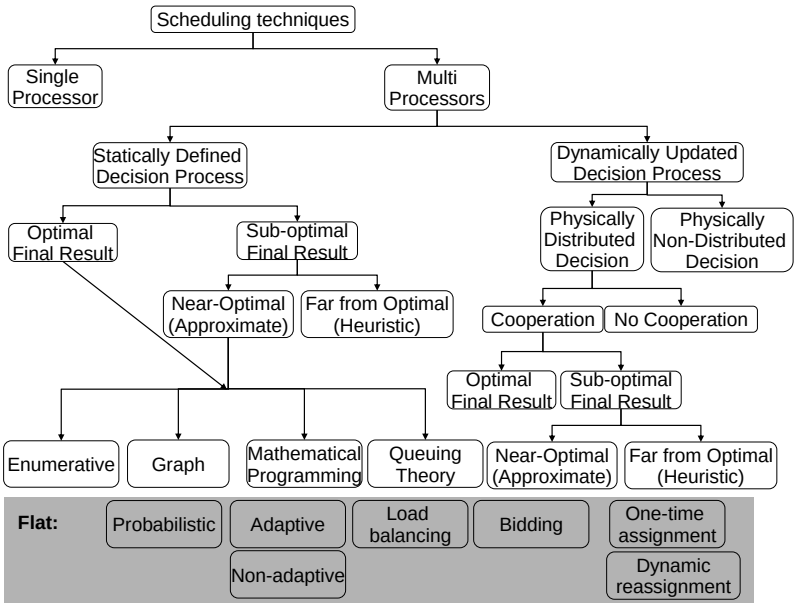


Figure 7.2: The taxonomy of the scheduling techniques for distributed resource management scheduling problem consists of interacting classes and policies presented by Casavant et al.

The techniques are distinguished between techniques applied for a single processor, i.e. ``local'' techniques, and for multiprocessors, i.e. ``global'' techniques (the term ``global'' used in the study of [17] is different from our definition in Section 7.2). The ``global'' branch is further refined based on the time a decision is taken to static and dynamic scheduling. Static scheduling is divided into optimal and sub-optimal final result classes. The latter is separated to near-optimal final result (approximate) and far away from optimal final result (heuristic) classes, including also non-global scheduling techniques, which are not considered in this study. The remaining classes, i.e. approximate sub-optimal and optimal classes, are maintained since they refer to the global scheduling techniques which potentially reach the (near-)optimal solution. They are separated to: solution space enumeration and search, graph theoretic, mathematical programming and queuing theoretic classes. The authors further present a flat classification of policies which scheduling techniques may possess. These policies may fit in any of the above classes. The flat classification consists of

adaptive or non-adaptive, load balancing, bidding, probabilistic, one-time assignment or dynamic reassignment policies.

The limitation over completeness property exist also for the classification of [17]. It is still only partially complete without any hard guarantee on the completeness, as it excludes important categories of the existing techniques, e.g. pruning, partitioning and clustering approaches. For instance, the approach does not explicitly indicate which class describes the pruning techniques, e.g. [61], [128], [134]. The technique in [150] is classified to the graph class. However, the partitioning characteristics of the technique are not described. The load balancing policy used in the classification of [17] is one instance of the priority class. However, other approaches for the priority class also exist, e.g. cost, which are not depicted in any class of the classification tree. Although this hybrid classification allows class and policies combinations to exist, a combination among the classes of the hierarchical part is excluded. For instance, a combination of an approximate technique and an optimal approach, where the former finds a near-optimal solution and this output is used as an initial solution by the latter technique to achieve optimality is excluded.

Another limitation is that the classification is composed in a less systematic way and unrealistic techniques are potentially included in the classification. Although it is a hybrid classification and it is the first attempt which allows combinations of characteristics to occur, it is less consistent. The combinations among classes and policies are performed without any restrictions or rules following an ad-hoc way. In this way, it potentially includes scheduling techniques which cannot be realistically implemented, e.g. a combination of an enumerative technique with a probabilistic policy.

Difficulties may appear during the classification of several techniques affecting the understanding of their methodology. For instance, the technique in [150] with partitioning characteristics is classified to the graph class. A technique based on the priorities [213] belong to the same class, although it does not apply any partition to the scheduling problem. The technique in [74] and in [179] belong to the mathematical programming class. Although the latter applies the mathematical technique for several iterations, this recursive characteristic cannot be distinguished from a classic mathematical programming technique. Another example is the Tabu search, Seed technique, SA based, GA based and SE based techniques which reside in the probabilistic policy of the flat classification without presenting further refinement among these techniques. Hence, an less meaningful distinction among techniques is offered since different in nature techniques belong to the same class. An approach to refine the probabilistic policy by providing classes to describe the distinct non-overlapping areas of the stochastic techniques, to clearly illustrate their interface and provide the main components of each technique is required.

Other more recent studies [114], [182] and [79] present the new advances of the scheduling techniques. However they do not adapt the previous classifications to include the new described approaches or provide a new complete classification scheme.

The existing related work in classification of global scheduling techniques from early till recent years has not been sufficiently matured yet. The previous classifications are partially complete and not future-proved, i.e. they do not prove their capability of including new techniques, which is a

very crucial missing feature. They use a wide set of classes describing overlapping characteristics. In this way, several scheduling techniques are ambiguously classified, a meaningful comparison is not achieved and useful insight in the techniques is not obtained. Hence, a complete, consistent and compact classification for design time global techniques to support our goals (Section 7.1) is strongly desired. In this study, we present a systematic methodology to compose this classification, we describe the derived scheme and we illustrate how the proposed classification classifies existing techniques while remedying the limitations of the previous approaches.

7.4 The proposed systematic classification

The proposed systematic classification offers a complete, future-proved and updated picture of the global scheduling techniques. It classifies in a systematic way any existing technique which meets our assumptions (Section 7.2). As long as future techniques still meet these assumptions they will be included into the systematic classification due to its completeness property and the way it is composed. Other classifications presented in the literature have not been systematically created and typically miss parts of the existing techniques (Section 7.3).

In our proposal every class is described by unique set of characteristics through the whole classification scheme. Each branch differs in at least one unambiguous characteristic. This still means that the characteristics can be partly shared among classes. Indeed, the characteristics of different classes are reused in the hybrid combinations following the horizontal arrows of the unidirectional propagation of constraints. In this way the redundancy is avoided and the classification uses a reduced amount of required classes in a compact form to describe the features of all the techniques. With the proposed classification scheme every global scheduling technique following our assumptions (Section 7.2) can be classified either by belonging to one specific class or to a hybrid combination of primitive classes, as illustrated in the next sections. The classification completeness is experimentally supported by a quite large but still selected set of publications which is used in order to representatively cover the entire exploration space as effectively and globally as feasible, as presented in Section 7.5. These publications are widely spread in the exploration space and their characteristics have been chosen to be as diverse as possible. In this way we maximize the probability of covering all the different existing techniques. The publications are studied and analyzed using the systematic classification in order to verify its completeness and consistency. We use a subset of them also as illustration examples throughout this chapter.

The classification decomposes a technique to primitive components with specific characteristics and into interrelationships that are easier to be analyzed and understood. Each component belongs to the class matching its characteristics. This leads to a straightforward classification and in an unequivocal scheme even for techniques that are difficult to be classified in the conventional approaches. In this way the main features and the structure of the techniques are clearly illustrated. Their similarities and differences are made more concrete. Hence, a better understanding of the wide variety of the techniques is achieved which is essential for their efficient use. The proposed

classification can be used to supports the selection of the most promising scheduling techniques for a given application domain by instantiating the principles and the usage framework process of chapter 2. This can be achieved by analyzing the requirements of the domain and by matching them with the classes, which efficiently address them. The set of the matched classes describe the hybrid combination of scheduling techniques which efficiently schedule the given application domain. The selection of different members of this hybrid combination creates different scheduling techniques for applications within this domain. Suppose that an application is (near-)optimally scheduled by a technique which belongs into a hybrid combination. Another application with same requirements should have a solution that belongs to the same hybrid combination. The classification framework can also be used to support the development of new approaches. Different classes can be combined and different options inside classes can be used to create new promising techniques. If the required characteristics of a technique are known upfront, the classes that exclude these characteristics will not be part of the final technique. For instance, if optimality is strictly required, the near-optimal class is excluded. In this way, they can be discarded early in the decision process from the potentially available solutions. Based on the proposed systematic classification, the components of scheduling techniques belonging to non-global classes can be identified. They are replaced or modified to fit in a global class converting the initial non-global scheduling technique changes to a global one. The development of parametric templates for the target domain of large and complex CDFGs is presented in Chapter 8.

Following the framework creation methodology of Chapter 2 we derive the systematic classification depicted in Fig. 7.3. Initially, the properties of the techniques to be classified are clearly determined. This step is crucial since it describes the target domain of the classification. These properties hold for the complete classification tree and thus during refinement they are vertically propagated to the subclasses. The next step is the iterative refinement process. Each iteration applies the general step of the framework creation methodology, i.e. create a top-down split and the uni-directional constraint propagation arrow. In the initial iterations, i.e. high levels of the classification tree, more general splits are preferred in order to compose a stable classification. In addition, we prefer to apply the split, which leads to more balanced branches, as it approximately equally divide the fundamental characteristics of the parent class into the children subclasses and the granularity of primitive classes is similar through the total classification scheme. In this way, a close to equal search is applied in the space of the available techniques. For instance, the split between stochastic and deterministic techniques creates two balanced branches describing the approaches to compose an algorithmic process. After the application of the selected top-down split, the description of the negative subclass is reformulated to a positive term to prepare the subclass for the next iteration. For instance, the not stochastic techniques are renamed to a term that encompasses all the techniques that are not stochastic, i.e. the deterministic techniques. If more refinement is required, the subclasses are inserted to list of parent classes. The process is repeated until the list of parent classes is empty. The iterative refinement process terminates when the distinction among techniques is meaningful and the classification can be efficiently used. In case further distinction among techniques residing in one or more classes is desired, the framework creation process should

be applied. Our classification is scalable since after refinement it will be expanded in depth keeping the upper levels identical and maintaining its important properties. During the iterative refinement process the characteristics of each class are vertically propagated by the consistent top-down splits to the leaves following parent-children relationships. This allows the classification to obtain very meaningful results, as the derived classes, i.e. the leaves, are very restricted and concrete.

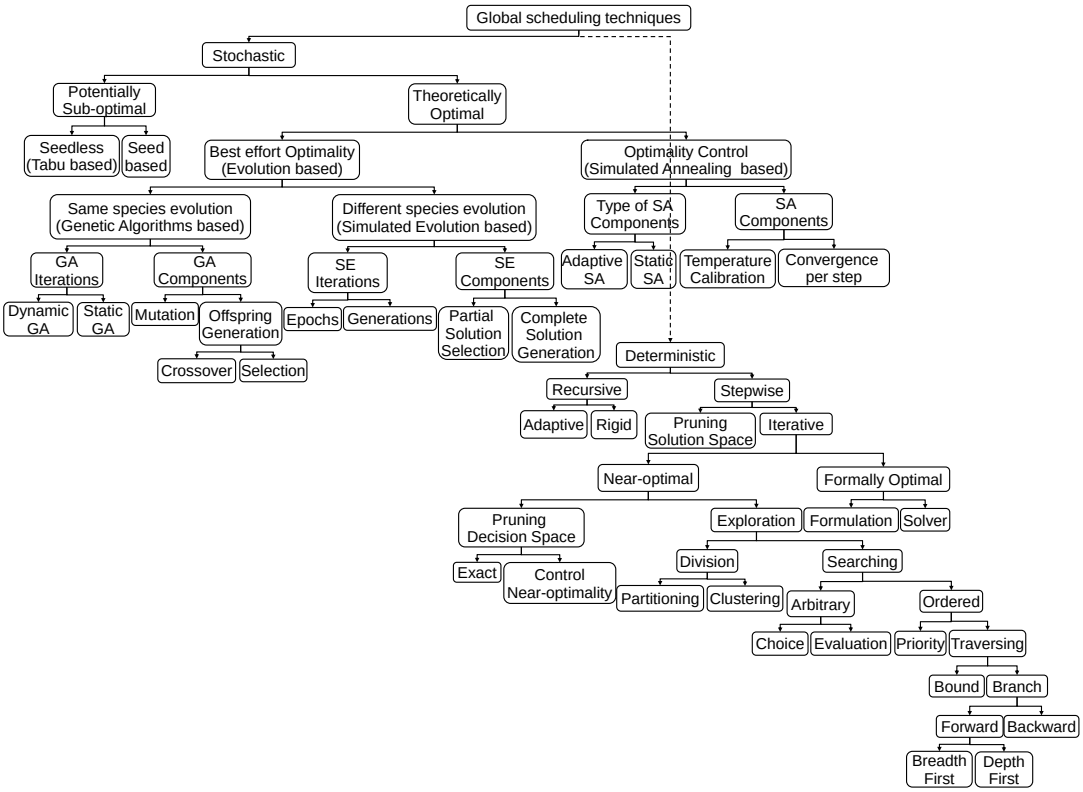


Figure 7.3: The proposed systematic classification of techniques globally solving design time mapping emphasizing in ordering in time and assignment in place. The gray classes are the leaves which describe a set of characteristics belonging to a component of a technique or a technique.

In the derived classification of Fig 7.3, the deterministic branch is split into the recursive and the stepwise (not recursive) branch. The recursive branch is refined into the adaptive and rigid (not adaptive) classes. The stepwise branch is divided into the pruning of solution space and the iterative classes. The iterative class is split into the near-optimal and the formally optimal classes. The near-optimal class is divided into the pruning of the decision space class, which is split into the exact and near-optimal classes, and the exploration class, which is refined into the division and the searching classes. The division class can be achieved through partitioning and clustering and the search class through arbitrary search and ordered search. The arbitrary search class is split into the way to perform the arbitrary choice and the evaluation of that choice. The ordered class is refined into the class which describes the priority of the nodes and the class which describes the way to traverse the nodes. The traversal class is split into the bound and the branching classes. The branching class is split into the forward branch class, which is refined into the Breadth first and Depth first classes, and the backward branch class. The formally optimal search class is split into the formulation and solver classes.

The stochastic branch is split into the potentially sub-optimal and the theoretically optimal classes. The former is divided into the Tabu search based and the Seed based classes. The latter is refined to the class which provides best effort on the optimality of the solution (evolution based) and the class which provides a control mechanism over the optimality (Simulated Annealing based). The evolution based class is refined to the same species evolution and different species evolution classes. The same species evolution class is divided into the class which describes the way to perform the iterations, which are refined into dynamic and static classes, and the class which describes the components. The latter is split into the mutation and the offspring generation classes. The offspring generation class is split into the crossover and selection classes. The different species evolution class is refined to the class which describes the way to perform the iterations, which is split into the Epochs and generations class, and class which describes the components. The latter are refined into the partial solution selection and complete solution generation classes. The Simulated Annealing based class is refined into the class which describes the type of the components, which is split into the adaptive and the static classes, and the class which describes the actual components. The latter is refined into the class which describes the temperature calibration and the class which describes the conversion per step.

In the next subsections, we describe the classes of the classification tree starting from top layers towards the final leaves. Then, we present the uni-directional propagation of constraints between the classes.

7.4.1 Deterministic techniques

One major branch of the global scheduling techniques is the techniques which are deterministic in nature, i.e. they occur in a completely predictable fashion. Note though that their process will still contain potentially complex structures, e.g. heavily data-dependent loops and branches. But for a given set of input stimuli, they will always provide exactly the same outcome. So the deterministic process do not contain any random number generators or the input stimuli does not trigger any unpredictable task of the deterministic scheduling process. The deterministic techniques search the exploration space to find the (near-)optimal solution generating a subsequent state by following a predefined sequence of steps. The latter explicitly determines how the transition from the current state to the next one is performed and when it converges to the (near-)optimal solution. The quality of the obtained solution is imposed by the objective function and the properties of the sequence. The deterministic techniques provide (near-) optimal solutions usually in specific and not generic cases. Their performance depends on the instance of the problem to be solved [167] and on the effectiveness of the the deterministic technique's nature to this type of problem [189]. Hence, explicit exploitable characteristics and correlations are present and the classes are potentially combined to produce a global scheduling technique with the required attributes each time.

7.4.1.1 Recursive techniques

A part of the deterministic techniques has the repeatability property. These techniques belong to the recursive class and control the iterations of their entire process until it converges, i.e. the termination criteria define that the (near-)optimal solution is reached. The recursive techniques are split to the techniques which have adaptability and into the techniques which maintain the same process through the iterations.

The **adaptive** class defines the recursive techniques which are capable of self-training and self-learning. They start from an initial state and they indicate when and how the applied process is modified and when it converges to the (near-)optimal solution. For example, some adaptive deterministic techniques use neural nets [69] which are composed of the net topology, the node characteristics and a set of training rules indicating the initial weights and their adaptation [116]. The adaptive techniques try to converge to the (near-) optimal solution by following the characteristics of the solution space shape. These techniques are quite effective in solution spaces with implicit correlations which can be exploited to vary the effort spent during the search progress.

In the **rigid** class belong the rest of the recursive techniques, which are stable. They start from an initial state, i.e. an arbitrary solution [179], and they iterate the same fixed process, i.e. update solution based on the worst incremental cost direction [179]. Their aim is to converge to the (near-)optimal solution after a number of iterations without modifying the applied process and they can be used when no hidden correlations exist in the solution space.

7.4.1.2 Stepwise techniques

The remaining deterministic techniques belong to the stepwise class. The process of the stepwise techniques is applied only once. However, it may consist of several intermediate steps. The stepwise techniques are divided into the pruning of solution space approaches and into the iterative techniques.

The **pruning of solution space** class describes the part of the stepwise techniques, which reduces in an exact way the solution space before search. These techniques describe the implementation of the space pruning which is achieved by removing areas that include unfeasible and sub-optimal solutions. Unfeasible solutions can be pruned by satisfying the problem constraints, e.g. by analyzing the imposed constraints and establishing extra edges which exclude unfeasible solutions [128]. Sub-optimal solutions can be pruned based on the objective function, e.g. by inserting penalty terms to avoid sub-optimal solutions. A solution space pruning technique is useful in complex problems which provide information which can be used to prune solutions from the space.

The **iterative** class includes the remaining stepwise techniques. The iterative class determines how the detection of the (near-)optimal solution into the exploration space is performed. Based on the quality of the obtained solution the techniques are classified to near-optimal techniques and to formally optimal ones. The near-optimal techniques control the near-optimality of their solutions and in special cases they can lead to a guaranteed optimal result. The formally optimal techniques

use an exact problem formulation and global solver.

7.4.1.2.1 Near-optimal The set of the near-optimal classes describe the different potential steps applied during the search, thus they are usually met in hybrid combinations in global scheduling techniques. The near-optimal class is divided into the way the pruning in decision space is applied and into the way the exploration of the search space is performed.

The way the **pruning of the local decision space** is implemented in each component determines the quality of the solutions of the near-optimal techniques. In case **exact** pruning approaches are used in all the components of the technique, an optimal approach is composed. Otherwise, at least one component uses an approach which is in a **controlled way near-optimal** to reduce the available options. Using exact or controlled near-optimality depends on the problem to be solved. For instance, exact decision pruning may be useful in cases where optimality is required but within a reasonable time, whereas controlled near-optimality is effective when time is crucial and optimality is not so essential. However, it should still be controlled, i.e. it would not become lower than the acceptable usually user-defined threshold.

The implementation of the **exploration** of a near-optimal technique is performed in several ways. A part of the near-optimal techniques apply **division** of the scheduling problem to smaller regions. This division is implemented by partitioning or clustering. The **partitioning** techniques divide the overall problem in smaller sectors based on specific rules, e.g. the operations are partitioned to critical path and non-critical path ones [150]. The **clustering** techniques combine small problem parts to create common sectors based on rules, e.g. operations are grouped if they have no further parallelism to be exploited and to reduce the number of clusters to the number of processing units [109]. Division classes can be applied in problems with quite incremented set of potential solutions.

The remaining near-optimal techniques solve the entire scheduling problem and belong to the **searching** class. This class contains the techniques which find their way through the exploration space with a search based strategy. The search is implemented by ordered and arbitrary techniques.

The **arbitrary** techniques perform a pure arbitrary choice of the node to be scheduled without using any random number generator though. They include the class describing how the **choice** of the node is implemented, e.g. critical operations are scheduled in a ``First Come First Served'' (FCFS) strategy [150], and the class which **evaluates** the choice.

The **ordered** techniques solve the scheduling problem in an ordered predefined way. They are split into the sorting of the nodes and the way the traversing of this sorting is implemented. They use the **priority** class to sort the nodes based on a function, which potentially supports the efficient search of the solution space. For example, the non-critical path operations are scheduled based on the freedom priority [150]. The **traversing** class defines how the nodes are traversed. This is divided into the **bounds** used in each node and the how the **branch** of the nodes is performed. A good estimation of the bounds leads to more efficient and faster techniques. The branch has a **forward** direction and a **backward** direction. The forward direction of branching can be performed in either **Breadth-First (BF)** or **Depth-First (DF)** strategy.

7.4.1.2.2 Formally optimal The formally optimal techniques converge to the optimal solution. They are divided into the **formulation** class which describes in an appropriate way the problem to be solved and the **global solver** class which describes the actual solver of the problem. For example, constraint analysis through polyhedral theory determine the scheduling polytope to be solved. It is formulated as an Integer Linear Programming (ILP) model [25]. The solver class provides the actual solver of the formulated problem, such as a Mixed Integer Linear Programming solver (MILP) [23]. The formally optimal approaches are quite effective in scheduling problems are sufficiently decreased in size, have a largely linear solution space and require guarantee on optimality.

7.4.2 Stochastic techniques

The remaining techniques are the stochastic ones which systematically guess the (near-)optimal solution in the exploration space. The stochastic techniques start from an initial state and use probabilities and randomness to determine the next state which is potentially closer to the (near-)optimal solution. The next state generation iterates until some termination criteria are met. These usually indicate that the obtained solution has reached the (near-)optimal one or that the search for a better solution is too costly, whereas the gained quality is too low; accordingly, the process terminates. The way of generating subsequent states and which objective function is minimized determine the quality of the obtained solution. The stochastic techniques typically behave quite well in solution spaces with several local minima. Due to their probabilistic acceptance they are capable of applying ``hill climbing" moves. However, in the initial steps of execution they move very fast to a good solution, but then they are not capable of quickly identifying the global (near-) optimal. Based on the optimality of the obtained solution, the stochastic techniques are divided into the techniques which are theoretically proven to be optimal and the techniques which miss the optimality proof.

7.4.2.1 Theoretically Optimal

A part of the stochastic techniques show or even prove that their process converges to the (near-)optimal solution under specific conditions, which are only theoretically achievable though. These form the theoretically optimal stochastic class, which is further divided into the techniques which provide control on the solution optimality. In the current literature, this class can be highly represented by Simulated Annealing (SA) based techniques. In the next sections, the SA based term is used to describe this class. The remaining techniques provide less control on optimality and are described as best effort techniques. In the existing literature they can be represented by the Evolution (E) based techniques. The E based term to refer to this class in this chapter.

7.4.2.1.1 Control over optimality According to existing theoretical results, the techniques with effective termination criteria or SA based techniques, asymptotically approach the global optimal of the exploration space [163]. The SA based class is split into the type of SA components

and the SA components themselves.

The **type of the SA based components** may be adaptive or static determining the final applied process. The stochastic process of the SA based techniques may be implemented using **adaptive** parameters, which are modified during the execution to more efficiently search the exploration space. The SA based techniques have properties which allow such an efficient adaptive implementation. They are useful for irregular solution spaces with hidden correlations. In an irregular area or in the edges between areas of the exploration space, the stochastic process continues more carefully by decreasing the step of the temperature reduction and increasing the number of iterations in the conversion step. In smooth areas the process performs larger steps and thus reduces the required execution time. The other class is to implement the SA based stochastic process using **static** parameters and equations, i.e. fixed number of iterations and parameter reduction factor. These approaches are useful in rather homogeneous solution spaces, otherwise they require too much time to reach equilibrium.

The **SA based components** consist of the calibration of the ``temperature-like" component and the conversion step for each temperature. The SA based process uses through the **calibration** component a parameter to specify the allowed range of the random moves, e.g. the temperature [43]. In the initial process steps, the parameters allow moves with large changes to occur, e.g. a high temperature allowing the interchange of distant placed operations. The parameter can be modified, e.g. temperature reduction to allow smaller changes to occur. They SA based techniques start from an initial temperature and an initial solution, e.g. the one created by a random configuration [13]. In the **conversion step** component, random moves are applied to the current solution, i.e. a random displacement changing the scheduling and the outcome of the objective function, e.g. an interchange of two operations [43]. If the result has a positive impact, i.e. it reduces the objective function, the move is maintained. In case it has a negative impact, the decision of its acceptance is treated probabilistically, e.g. if a random number is smaller than the value $e^{-\frac{CostChange}{T}}$, it is accepted. The probabilistic acceptance allows ``hill climbing" moves to occur during search. The process is iterated until conversion, i.e. it reaches a stable or determined state, e.g. no more changes occur or a predefined number of iterations is reached.

The SA based techniques differ in the type of components, the temperature calibration component, the way they implement the random moves and the termination criteria. The implementations of techniques which lead to sub-optimal approaches and thus do not satisfy our assumptions are not included in our proposal. For instance, a SA based technique implemented with a ``predictable" acceptance criterion, e.g. using always the same ``seed", is converted to a deterministic but not global approach.

7.4.2.1.2 Best effort optimality The best effort in optimality techniques or the E based techniques mimic the process of the natural evolution. They start from an initial state, i.e. solution or population of potential solutions, such as the ones created by applying ``As Soon As Possible" (ASAP) or ``As Late As Possible" (ALAP) [168]. They generate a next state which is similar but not the same to the current one. This is achieved by probabilistically maintaining the superior

elements of the current state, e.g. the most fit ones [168], and discarding the inferior ones. The new state includes the survived elements and new ones, e.g. deriving from crossover and mutation genetic operators. The generation step iterates to obtain new solutions until a termination criterion is met. The state with the minimum value in the objective function is returned as solution with the evolution process termination. The E based techniques differ in the way they create the similar solutions to search the exploration space. The E based techniques are divided into the evolution of same species techniques, which are represented by the Genetic Algorithm (GA) based techniques, and the evolution of different species, which are represented by the Simulated Evolution (SE) based techniques.

Both the SA and the E based techniques apply iteratively a probabilistic step to reach the optimal solution until some termination criteria are met. Their main difference is the way of implementing the probabilistic modification of the current solution. The SA based techniques apply and accept random moves whereas the E based ones create a similar state where the superior elements of the current state are maintained and the remaining ones are new.

The **GA based techniques** theoretically converge to the optimal solution [59] and it is stated that ``a tradeoff between their convergence and their long-time performance exist'' [78]. The GA based techniques retain a population of potential solutions and search concurrently many dimensions of the exploration space. They start with a population of potential encoded solutions in chromosomes, e.g. created by ASAP or ALAP [168]. They iteratively apply GA components to create a new population until a termination criterion is met. The GA based techniques are split into the type of GA applied iterations and the GA components which are applied.

The **type of GA based iterations** are divided into the **dynamic** approaches, which apply disruptive changes in the populations, and the **static**, which apply several fixed genetic operations to a number of populations. Several ways exist to dynamically control the process, e.g. by controlling the crossover and mutation parameters during the GA based process [133].

The **GA based components** are the mutation and the offspring generation. A **mutation** component can be applied to the population for disruptive changes of the chromosomes. The **offspring generation** component consist of crossover and selection of the offsprings is applied. In the offspring generation component recombination genetic **crossover operators**, e.g. crossing and copying, mate parent chromosomes to generate offsprings. In this way new chromosomes are produced which potentially initiate the next population depending on the **selection** component. The decoding of each solution allows the calculation of its cost. The latter determines its fitness which is used to distinguish superior from inferior solutions. The solutions with best fitness values have more possibilities to be chosen as parent chromosomes since they are selected as members of the new population. After the termination criteria are met, the best solution of the populations is returned.

The GA based techniques may differ in the way they implement the mutation, the recombination genetic operators, the decoding of the chromosomes, the fitness evaluation and the termination criteria. Our classification is dedicated to global techniques and thus the GA based implementations which lead to sub-optimal approaches are excluded as they do not satisfy our assumptions (Section 7.2).

The **SE based techniques** asymptotically obtain the global optimum independently of the initial solutions [119]. The SE based techniques generate a subsequent state by applying iteratively a select and a generate component. The SE based techniques are split into the type of SE based iterations and the SE based components.

The **type of SE based iterations** may be epochs or generations. In the **epochs** case, it is a dynamic approach which has as scope to increase the quality of the obtained solution. Each epoch may have a different stop criterion, a different duration and its generations may be developed in a quite different way. In the **generations** case, the SE based techniques are static approaches that apply several generations under one stop criterion.

The **SE based components** are split into the partial solution selection component and the complete solution generation component. The **partial solution selection** component probabilistically discards the inferior elements based on their contribution to the objective function, i.e. the elements increasing more the objective function, composing a partial solution. The **complete solution generation** component completes the solution by reconstructing the missing elements based on priority rules, e.g. freedom scheduling of nodes [119]. The state with the minimum value of the objective function through the iterations is the best solution. The approaches differ in the determination of the superior parts, the reconstruction of the missing elements and the termination criteria. The ambiguous classification of the SE technique in [119], as presented in Section 7.3, is resolved since the proposed scheme classifies it to the SE based class. The SE based techniques potentially may incorporate a form of dynamism in their search processes, which significantly differs from an adaptive SA based technique, as they are quite complex implementations which cannot provide guarantee on optimality.

Both the GA based techniques and the SE based techniques apply iteratively a probabilistic step to the current state to create a similar state. Their main difference is the way of creating this similar next state. The GA applies genetic operators for perturbation on the surviving encoded solutions. The SE based techniques modify the current solution by keeping the least costly elements and by generating new ones.

7.4.2.2 Potentially sub-optimal

The remaining stochastic techniques provide weak or no proof of converging to the (near-) optimal solution. However, they still belong to our target class because in practice they behave quite well in specific domains.

A part of these stochastic techniques are independent of the initial solution and belong to the **seedless** class. The latter can be represented by Tabu search based approaches. They scan the neighborhood by creating several candidates applying a number of not prohibited moves selected by probabilities. From the derived candidates, they choose the best one. In this way they accept moves that increase the cost function to avoid local minimum solutions. If the move is accepted, the reverse move can be prohibited to avoid cycling for several iterations depending on the aspiration criterion. The latter is recorded as a restriction in their short term memory, namely the Tabu list.

The process is iterated until it reaches a specified number of moves with no improvement.

The remaining techniques depend on the initial solution and compose the **Seed based** class. They require an initial solution which affects the quality of the outcome, like Monte Carlo sampling [144]. They use random samples derived from a distribution which are weighted and the best permutation is selected, hoping to reach the optimal solution.

With our systematic classification a technique may belong to one primitive class or to a hybrid combination of such classes. Any class can be selected or not to classify the final global scheduling technique and several options of the selected classes can be combined.

7.4.3 Horizontal uni-directional constraint propagation

During our study we analyzed a large set of techniques selected to cover the complete space. Based on their main characteristics, nature and on the classification splits, we derive several restrictions in the combinations. These are expressed through uni-directional arrows imposed on the systematic classification.

In every top-down split, it is determined if a uni-directional arrow between subclasses exists or not. If it exists, the source subclass of the arrow propagates its decision as a constraint to the destination subclass. Uni-directional arrows are included in the T splits to describe the design constraint propagation. The ordering is guided by four rules of horizontal uni-directional constraint propagation instantiated in the context of scheduling top-down splits: (1) The source class provides parameters to the destination class, which are required for the execution of the scheduling approaches they describe, e.g. a schedule derived from a tabu based technique is given as an initial solution (``seed'') to the seed based techniques (left top part of tree in Fig. 7.4-(a)), (2) the parameters of the source class characterize the destination class, e.g. when the pruning of the exploration space applied during scheduling decisions is exact (Exact class under Pruning Decision Space class) and it is propagated to the Exploration class, it requires from the Exploration class parameters to guarantee that the parts of the search space, which are pruned during the exploration, do not include the optimal schedule, (3) the source class nature prunes unrealistic or sub-optimal class combinations, e.g. the unrealistic combination of a formally optimal class with a near-optimal class, since the optimal schedule would have been already found by the optimal class before applying the near-optimal one and (4) the source class does not remove potential promising options in the destination class leading to near-optimal hybrid techniques, e.g. using a deterministic solution as initial solution to a stochastic class removes several options in the stochastic parameters, e.g. range of moves, termination criteria etc. The number in each top-down split of Fig. 7.4-(a) describes which rule is applied to derive the class ordering. In the next paragraphs, we describe the potential combination and the arrow between the subclasses of each split.

The first design constraint propagation has as source the stochastic class and propagates the decision in the deterministic class (top part of Fig 7.4-(a)). This arrow prohibits the combination of a deterministic class followed by a stochastic class, which is a less efficient combination as propagating the deterministic result would heavily constrain the available options in the stochastic

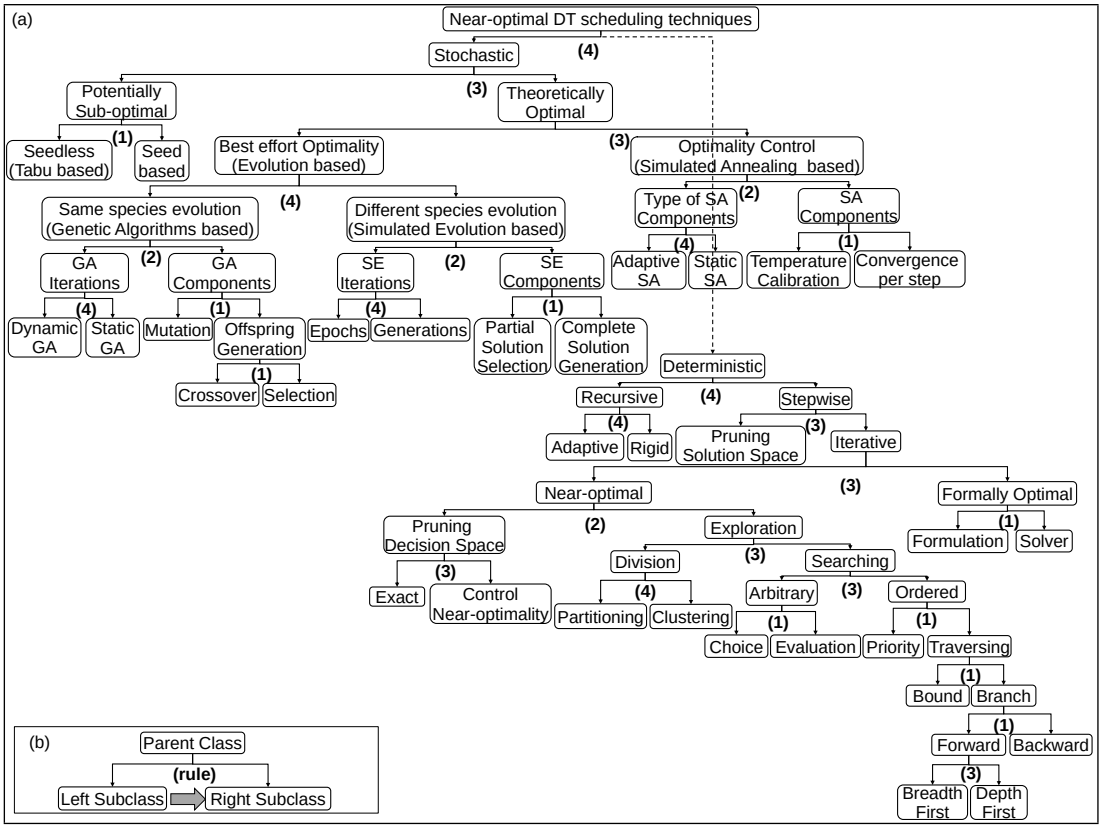


Figure 7.4: The proposed systematic classification with the number of rule of the horizontal constraint propagation principles for each top-down split.

class because it restricts its freedom, it localizes its process and leads to potential loss on optimality and convergence control. For instance, the imposed initial solution by the deterministic class may significantly increase the exploration time of the stochastic part and destroying convergence. In the stochastic split, the potential sub-optimal component is applied first and its output is used by a theoretically optimal component. Otherwise the sub-optimal part would overwrite the convergence benefit of the theoretically optimal component creating a suboptimal final technique. The tabu search based class determines a solution which is used as a ``seed'' by the seed based component. The opposite direction composes less meaningful combination of these classes. The best effort optimality or Evolution (E) based class determines the population evolution and the output is used by the optimality control or Simulated Annealing (SA) based class. The latter provides the effective termination criteria to obtain optimality. The opposite combination would not benefit from the control on optimality of the SA based class. The decision in the type of SA based components is propagated to the actual components to decide their implementation. First the adaptive type is decided and then the static one. Otherwise the adaptive part would be restricted, as the static decision may not allow several modifications to be applied. In the SA components class, the temperature calibration is decided and then propagated to the conversion per temperature step to decide the equilibrium. The way the temperature is reduced is required to efficiently define how the convergence in each temperature step will be performed supporting the normal process of the conversion per temperature step. For the E class, the same species evolution or Genetic Al-

gorithms (GA) based class gives a partial solution to the different species evolution or Simulated Evolution (SE) based class. The opposite direction would overwrite the result of SE based process because of the mutation characteristic of the GA based process. The GA iterations are decided and propagated to the implementation of the GA components. The GA iterations combine dynamic and then static type, otherwise several dynamic options may be prohibited. The GA components combine the mutation with the offsprings generation. The opposite would destroy the impact of the mutation effect to the offspring generation. The offspring generation applies crossover operation to generate the offsprings and then the offsprings to be maintained are selected. A similar reasoning holds for the components of the SE based class.

In the deterministic class, the recursive class is determined and the stepwise class, i.e. the step of the technique's kernel, is selected. Determining first the steps would heavily constrain the available options in the recursive class. For example, using an ILP technique with a poor ability of modifications constrains the options in the adaptive class. In the recursive split, the adaptive class determines the modification of the kernel and then the iterations of one kernel instance are defined, i.e. the rigid class. The opposite way would constrain the adaptive options, e.g. selecting a specific number of kernel iterations restricts the modification options which cooperate well with this number. In the stepwise class, the solution space pruning is decided first and is propagated to the solution search process. The reverse combination neglects the reduced space characteristics which may indicate the iterative technique to be used in the iterative class. First a near-optimal technique is applied and its output, which is a good initial solution, is propagated to the optimal class. In the reverse direction the solution would have been found before applying the near-optimal part. The selection of the decision space pruning class is propagated to the exploration class to indicate how the scheduling exploration should be performed and thus affecting the implementation of the exploration components. The exact way is combined with the controlled near-optimal way. When the pruning is exact, it indicates to the exploration class that the next classes should be implemented in a way that the optimal solution is always included in the remaining search space. In the exploration class a technique may first cut the problem to smaller regions (division) and then searches for a solution within each region (searching class). Otherwise the searching part would reduce the options of the division part. For example, deciding first to use the the arbitrary search demands from the division part to create appropriate regions where this search can be applied. In the division split, the partitioning class first partitions the problem to smaller sectors and then parts within each sector are clustered. Otherwise, the clustering decision diminish the possible ways of partitioning the problem, because the partitioning is applied on already clustered groups which prohibit potentially promising partitions to be applied.

In the searching class, the arbitrary part first determines the jumps in the search space and then the ordered part searches in the place indicated by the arbitrary. The combination following the opposite direction would (heavily) reduce the potential arbitrary jumps due to the imposed constraints to the search. For the arbitrary part, the way to select a node is decided and propagated to the choice evaluation, since the evaluation cannot be performed if the way of choosing is unknown. In the order class, the priority of the CDFG nodes is determined. This information is propagated

to the traversing class, as it is required to decide an efficient way of traversing the CDFG based on the priority function. The reverse way does not take into account the characteristics of the priority function while deciding the traversing strategy. In the traversing class, the bound class propagates to the branching class, as it is required to decide an efficient branching. For example, determining first a depth-first traversing strategy and then the priority, e.g. force priority, leads to a less optimal result. The force priority is not taken into account during the traversal strategy selection. The bounds are decided and then propagated to the branching class. The opposite direction does not support efficient decisions on how the branching is performed. In the branching class, the forward direction is propagated to the backward direction in order to decide how to implement the latter step. In the forward branching, the more promising combination is to apply a Breadth-First (BF) and then a Depth-First (DF) approach. In the formally optimal class, the formulation class of an optimal technique defines the problem. This is propagated to the solver class to describe the way to solve it. The inverse combination does not seem to produce any promising solution, so it is discarded.

In case of propagating the decisions following the opposite direction would remove potential promising solutions. For instance, the output of the best effort optimality class is propagated to the optimality control class. Following the opposite in this combination, the composed techniques would not benefit from the effective termination criteria on optimality of the optimality control class leading to a suboptimal technique. The constraint propagation is performed following one-way directions in the top-down splits. They impose the way to propagate the decisions of the source class as constraints to the next classes. In case bi-directional relationships are used in the top-down splits, the subclasses would strongly affect each other. This lead to infinite iterations and increased CPU time and to obtain a result, the process should be preliminary terminated, leading to suboptimal approaches.

7.5 Illustration of the systematic classification

In this section we classify a series of illustration examples using the proposed systematic scheme. We use as much as possible representative and well established techniques covering a wide range of the published literature. The purpose is to verify our approach by showing how existing approaches are valid members, i.e. instances, of the proposed primitive classes, to illustrate the fundamental philosophy of all the distinct classes and to indicate the main principles in the way the classes can be combined. The global scheduling techniques are often hybrid combinations, which combine instantiations of different classes. Hence, we present the illustration examples for several different groups which cover all the classes. In this way, we can better illustrate how classes are combined in one scheduling technique. The groups have derived their name from the higher common class of the classification tree. For the deterministic branch, the illustration examples are described in the groups: adaptive techniques, rigid techniques, solution space pruning techniques, near-optimal hybrid techniques and formally optimal techniques. The groups for the stochastic part are: SA

based techniques, GA based techniques, SE based techniques, Tabu search based techniques and Seed based techniques.

7.5.1 Adaptive global scheduling techniques

A high-level synthesis scheduling problem is solved by a hybrid technique of the adaptive and the priority class [69]. The adaptive characteristics derive from the learning process based on the Kohonen's rule for self organization. The operations T_i to be scheduled are described by a CDFG. The set of constraints is $D = (N, T, PD, GT, LT)$, where N is Number of resources, T is Type of resources, PD is Propagation Delay, GT is global time constraint and LT is local time constraints. The network consists of the input node pair, i.e. random variables with uniform probability distribution, and the output nodes, i.e. the operations T_i. Every output node is connected to the input node by a weight pair (r_i, cs_i) corresponding to the node's position in the schedule space. From an initial solution, the net is trained on every iteration by presenting input vectors at the input nodes and adapting the weights of the output nodes. The operation selected to be scheduled is the one with the minimum distance, i.e. the data dependency distance between the input vector and the weight pair of the operations. The selected operation is moved towards the position specified by the input vector pulling its neighbors. The objective function C is proportional to the data dependency distance and the neighborhood radius. The process is repeated until it converges to the near-optimal solution, i.e. until the current neighborhood radius $\sigma(t)$ is equal to zero.

7.5.2 Rigid global techniques

A hybrid technique of the rigid and the optimal class solves a data path synthesis scheduling problem in [179]. The technique starts from an arbitrary initial solution which may not satisfy all constraints $D = (C_{mod}, GT, LT)$, where C_{mod} is the cost of module, GT is global time constraint and LT is local time constraints. In each iteration an improved solution is found based on the gradient of the objective function, i.e. the direction of most rapidly increasing. Given the maximum number of control steps and satisfying the constraints set they minimize the objective function $C = \Sigma(w_i * VTerm_i) + C_{mod}$, where w_i is the weight for each violation term per constraint $VTerm_i$. The weights are small at initial steps, whereas at the end increase in order the solution to satisfy all constraints. The solution is updated by an optimal technique, where the set of constraints are formulated as integer variables and an ILP solver is used. If the solution change is small, the weights w of the constraint violation parameters are increased. Then, the same process is repeated. When the solution satisfies all constraints the problem has converged.

A technique combining the rigid and the optimal branch to solve high-level synthesis is presented in [25]. The Time-Constrained Scheduling (TCS) problem is solved by generating missing resource constraints d_i, i.e. tight lower bounds on the functional units number of each type. It is converted to an easier-to-solve Time-Resource-Constrained Scheduling (TRCS) problem. A careful ILP formulation through polyhedral theory is used which characterizes the set of feasible

schedules in terms of assignment, precedence, and resource constraints [26]. The objective function is $C = f(Num_{cs})$, where Num_{cs} is the total number of control steps. If TRCS produces a feasible schedule, it is guaranteed to be optimal, otherwise the resource constraints are increased and the process is repeated. A similar approach holds for the Resource-Constrained-Scheduling (RCS) problem with lower bounds on timing constraints.

A similar rigid and optimal hybrid technique [57] formulates the size of the search space through polyhedral theory [138]. The operations T_i are nodes of a DFG, constraints are $D = (area, latency)$ and objective function is $C = f(throughput)$. It starts with the initial ILP problem and it continues solving successive ILP problems. When it reaches the point with no further improvement, it terminates as it has obtained the optimal solution under the specific constraints.

7.5.3 Pruning techniques

A pruning technique reduces the exploration space by analyzing the problem constraints $D = (L, RC, TC)$, where L is latency, RC is resource conflicts and TC is timing conflicts. It can identify sequencing constraints d_i between operations additional to the precedence ones. Extra edges are added based on rules to reduce the solution space [128]. For instance, two conflicting operations cannot be scheduled at the same ``potential" b_i. If their distance would cause them to be scheduled at the same b_i, the execution start time cs_i has to be increased by at least one clock cycle.

Another pruning technique [134] reduces the search space by identifying possible conflicts and by posing constraints d_i to resolve them. The initial set of constraints is $D = (Temp, CR)$, where $Temp$ is temporal and CR is capacity resources. The technique estimates the activity demand and the resource contention. The objective function C determines the activities which are most likely to contribute to the bottleneck contention. They are sorted by inserting appropriate temporal constraints according to the activity demand. Another pruning technique starts from a maximal parallel flow graph and establishes supplementary edges [61]. Unallowed sets are used, i.e. sets with operations T_i which cannot be executed concurrently, i.e. they have different r_i, cs_i. Rules are assigned which determine all the alternatives to solve conflicts in an unallowed set. The conflict solution can be applied by introducing edges in the graph to serialize the conflict operations. However, the resolution of conflicts should be applied in an exact way without excluding the optimal solution.

These pruning techniques can be improved by combination with a technique which belongs to the iterative class to further optimize their output, e.g. a pruning [61] with an iterative optimal technique [74].

A global known periodic task scheduling problem on heterogeneous multiprocessor platforms is solved in [37]. This technique analyzes the tasks and their periods and prunes the search space by propagating constraints, ordering variables, ordering values and adding constraints. Although the technique searches for a feasible solution, we use it as an illustration example to show how our approach applies also for real-time techniques within our target domain.

7.5.4 Near-optimal hybrid techniques

A technique combines the near-optimal pruning of the decision space, the partitioning, the ordered and the arbitrary class in [150]. It partitions the operations T_i to the critical and non-critical paths. The critical ones are scheduled by an arbitrary technique without indicating a strict priority in the way the scheduling is performed, i.e. a FCFS strategy. The non-critical ones are scheduled by an ordered technique using the least freedom priority, i.e. the difference between the time the input values are needed and the time the operation's result is required. The objective function is $C = f(cost)$ or $C = f(speed)$ and the set of constraints is $D = (temporal)$ or $D = (cost)$. If a constraint is not met, the schedule length is increased and operations are rescheduled. The technique does not guarantee that can visit all the possible combinations and thus it is non-global. But we use it as an illustration example of a hybrid combination of near-optimal techniques of our classification since the authors intuitively mention that adding the backtracking property, i.e. backward branching, modifies it to a global one.

A scheduling technique deals with the scheduling problem of instructions T_i through three steps: clustering, compiler merging and placement [109]. The instructions with no parallelism are clustered together. The clusters are merged to be reduced to the number of processing units. Placement maps the merged clusters to the processing units by minimizing the objective function $C = f(B_{com}, C_{tot})$, where B_{com} is communication bandwidth and C_{tot} is total cost. It does not provide guarantee of the near-optimal solution. However, we use it to illustrate the potential of our classification scheme to extend such heuristic, i.e. non-global techniques with additional elements to reach near-optimal behavior. In particular, we can combine the near-optimal pruning of the decision space, the clustering and the ordered class of our scheme by properly modifying the technique's non-global part to include all the potential near-optimal solutions by including backward branch and near-optimality control in decision space pruning.

The memory scheduling problem is solved by a clustering technique where the memories are merged based on high cost gain [115]. Another technique maps arrays to multiple memories and combines the partitioning, the clustering and the ordered class [149]. The arrays are partitioned to ``logical arrays" based on their access. Then they are clustered to ``logical memories" based on their possibility of interleaving and mapped to physical memories minimizing the objective function $C = f(TC)$, where TC is transition count. These approaches do not guarantee the optimal solution as they miss the backward branching class, but are used as examples of applying the proposed scheme to classify memory scheduling techniques.

Many existing techniques belong to the ordered class using different priorities and traversing strategies.

The technique in [81] uses an exact ordered technique for the tasks T_i assignment in heterogeneous computing systems. The authors use the objective function $C = f(C_{execution})$, where $C_{execution}$ is the cost of execution and the cheapest solution through a node is selected. Each node has an estimated lower bound for the cost consisting of the path from the root to the node and the lower bound estimation of the path from node to the total solution. The branching is performed

in a combination of Breadth-First and Depth-First strategy and backtracks based on the lower cost priority.

A hardware-software system implementation scheduling problem is solved by an ordered technique which assigns tasks T_i to SoftWare (SW) and HardWare (HW) processors in [24]. The tasks are sorted based on their suitability and are assigned to balance the load between processors using depth-first strategy. It binds exhaustively the tasks which are not inclined towards either HW or SW implementation and keeps fixed the ones which are strongly inclined. If the constraints $D = (time, area)$ are not satisfied, it backtracks and changes the previous solution. The task assignment to processor cores is solved by a similar approach [117] by merging nodes with the maximum edge weight priority. When a solution is not found a backtracking mechanism is applied.

Another near-optimal ordered technique has the least function cost as priority function [213]. It expands the W most promising nodes in breadth-first order, where W is the search width. With W is equal to one is a depth-first strategy and with W equal to unlimited is a breadth-first strategy. All the intermediate values represent a hybrid technique of these classes. Systematic backtracking is integrated to make the technique global. The technique does not terminate when it finds a solution, but continues until all layers are ``backtracking" complete.

An ordered technique for instruction scheduling in VLIW processors uses the amount of slack as priority [1]. Another technique schedules applications in VLIW architecture with height priority and backtracking capability [197].

A near-optimal real-time multiprocessor scheduling technique meeting all deadline constraints is presented in [209]. It uses the lateness of any schedule from a node as a bound estimation. The node with the least bound is selected for forward branching each time. The problem of assigning and scheduling periodic task modules T_i to processing nodes in distributed real-time systems is solved by a near-optimal technique in [72]. The objective function C is the probability of meeting tasks deadlines. The bound is a tight upper bound in the objective function and the forward branching generates children of an intermediate nodes following a dominant relationship. The version of the A* algorithm of [156] is an ordered technique applied in multiprocessor task scheduling problem in [155]. It is the Branch Join Path isomorphism, where the priority is the cost function and the bound is the static level of a task. The forward branch of the nodes is performed based on the minimum cost function.

7.5.5 Formally optimal techniques

One optimal technique describes mathematically scheduling objectives and constraints which are easily translated into ILP formulations [74]. A feasible scheduling formulation is performed by carefully arranging the data dependencies and by using ASAP, ALAP and list scheduling techniques which allow computing an upper limit on control steps number. A lower and an upper limit on the function units number R of each type prevents many unnecessary searches. The objective function $C = f(Cost_{fu}, C_{in}, Cost_{reg})$. Then, they solve the feasible scheduling problem.

Another optimal technique derives the IP formulation model from polyhedral theory [58]. The

objective function is $C = f(time, C_{fu}, C_{reg})$, where C_{fu} is the cost of functional units and C_{reg} is the cost of registers. Equations ensure the unique use of constraints per cycle. The constraints D, e.g. each operator, functional unit, and register, are transformed into a graph to derive integral facets, i.e. cliques. The solver is based on the cubic polynomial Karmarkar algorithm for Linear Programming (LP) [83]. Algebraic equations satisfy the constraints D in [64]. Relations model the execution of an operation on an operator, the storage of a value in a storage element and the correct hardware resource sharing. The model is based on a system of constraints which is solved as a MILP problem [120]. Another optimal technique provides a flexible ILP problem formulation to find periodic schedules under timing and resource constraints D and power consumption objective function C [23]. In [40] they propose a Mixed-Integer Bilinear Programming (MIBP) formulation to find optimal solutions to the Multiprocessor Scheduling Problem with Communication Delays (MSPCD). They minimize the objective function is $C = t_{makespan}$ to run a set of tasks with fixed lengths Li on homogeneous processors.

7.5.6 Simulated Annealing based techniques

The main principles of the SA based technique have been introduced in [88]. A scheduling problem is the formulation of the data path synthesis as a two-dimensional placement of microinstructions T_i in space and in time, i.e. $b_i = (T_i, r_i, cs_i)$ [43]. It is solved by a SA based technique which searches the near optimal placement of microinstructions. It is a static approach which fractionally reduces the temperature and the number of states to converge per temperature value is a multiple of the number of code operations. Its random moves are: operations interchange, operations displacement and variable interchange in symmetric operations [43]. A move is always accepted if it reduces the objective function, e.g. $C = w_1 * T_{exec} + w_2 * N_{alu} + w_3 * N_{reg} + w_4 * N_{bus}$. Otherwise it is accepted if a random number is smaller that the $e^{-\frac{CostChange}{T}}$. When no change occurs for three successive temperature points, the process terminates.

A SA technique searches for an optimal design considering the set of constraints D in speed, area e.t.c. in [166] and [165]. It belongs to the SA based class with static parameters. The temperature is $T_k = T_o * a^k$, with a a constant equal to 0.92 and k equal to the number of moves. Its random moves are global, i.e. a move up or down from a control step cs_i of a randomly chosen branch to another, or local, i.e. ``rescheduling" of one operation T_i or merging of two consecutive operations. The moves causing reduction of objective function C are always accepted; otherwise, probabilities are used. Another technique performs ordering in time, assignment in space and module selection in high level synthesis [167]. Based on the proposed classification, the authors use a SA based technique with static parameters for ordering in time and assignment in space combined with pseudo-deterministic control for the module selection. They apply global or local moves [166] [165] and the initial temperature is computed such that any new configuration is accepted.

The multiprocessor scheduling problem is solved by a SA with static parameters based technique [137]. The temperature is reduced based on $T = T * K$, where K a constant value. They

apply random interchanges on tasks T_i. A move is accepted if the new system weight is smaller or if a random number is smaller than $e^{\frac{W_{old}-W_{new}}{T}}$. When the temperature is less than 1, the process terminates. Another SA technique solves multiprocessor scheduling with an minimum execution-time objective function C, which includes scheduling an application T on an appropriate subset of the resources R [211]. The problem of scheduling periodic and aperiodic tasks T_i onto pipelined parallel architectures is solved by a SA technique in [145].

7.5.7 Genetic Algorithm based techniques

The high-level synthesis resource constrained scheduling problem is solved by a GA based technique [67]. They use a data flow graph, the set of constraints D are the number and the type of the hardware resources and the objective function C is the completion time of the schedule. Recombination elements are selected through roulette wheel. Generation of new elements is implemented by uniform crossover and inversion [130]. The selection is based on the schedule encoding, which is performed by permutation of operations used as priorities for a list scheduler. The process terminates when it meets the lower time bound or the number of iterations reaches 100. The authors also modify the genetic recombination operators introducing rules to avoid identical schedules.

The GA based technique solves a time constrained scheduling problem in [68]. They use a data flow graph, the constraint D denotes the cycle step before which each operation must have finished its execution and the objective function C is the set R of the number of the function units required. Different genetic recombination operators and solution decoding methods are used. Relative displacements for each operation specify the order, the uniform crossover, the inversion and the mutation genetic operations. Permutations are used as priorities for a list scheduler or for a topological order scheduler to decode the schedule [68] and drive the selection. An absolute displacement to each operation, simple crossover and mutation genetic operators are used in [201]. They also use a special crossover operator to avoid illegal solutions. In [41] a chromosome is mapped to a solution through an order-based encoding of a chromosome, a uniform order-based crossover operator and a decoder. The execution time of the derived schedule is used as an objective function C to calculate the fitness of the solution. The selection of the encoded solutions is implemented by roulette wheel. The process terminates after a specified number of runs or when the solution meets the lower bound estimation.

The data path synthesis scheduling problem is solved by a GA based technique [46] known as Problem-Space Genetic Algorithm (PSGA) [184]. They use a different encoding structure called ``problem space''. The PSGA chromosome encodes a solution as a Heuristic/Problem pair (H,P). It has a ``functional-units part'' which describes the constraints D, i.e. the number and type of functional units, and a ``work-remaining part''. The latter is used as priority rule to generate the global schedule and to calculate the objective and fitness function. The exploration space is searched by perturbing the problem input data parameters. The recombination of chromosomes is performed through a simple one-point crossover genetic operator while the mutation genetic operator randomly selects a gene and perturbs its value.

A GA based technique solves the global scheduling of precedence constrained task graphs with non-negligible intertask communication onto fully connected multiprocessor systems [44]. It is based on problem-space genetic algorithm using a simple one-point crossover, mutation and a list based scheduling method for decoding the chromosomes.

A real-time task scheduling for multiprocessors using a problem-specific GA scheduling technique is presented in [112]. The tasks T_i have a set of constraints $D = f(t, tl, m)$, where t is arrival time, tl is the computation length on any processor and m the number of processors. The objective function to be minimized is $C = average(time_{response})$. The chromosome is coded to describe the task and the corresponding processor, the fitness function is the normalized average response time, specialized crossover and mutation are used.

7.5.8 Simulated Evolution based techniques

A high level synthesis scheduling problem is solved by a SE based technique [119]. Their SE technique solves separately the assignment of the control data flow graph nodes T_i to control steps cs_i and the assignment r_i of hardware cells to the nodes and the edges. The selection of partial solution is based the objective function $C = f(Cost_{overall})$ and the inferior elements have high contribution to it. An element is removed from the solution, if its normalized cost $norm = \frac{C(element) - min(C)}{max(C) - min(C)}$ is less than a random cost between -δ and 1. The complete solution is generated by creating the missing elements with a freedom based priority scheduling, where the highest priority element is chosen. If a random number between 0 and 1 is less than the mutation probability, a mutation operator is performed, else the assignment with the lowest incremental cost is applied. When a satisfactory solution is found or when the number of SE iterations reaches a maximum, the process is terminated. The authors also present an extended SE based technique which generates the next state applying the SE technique and probabilistically accepts it.

7.5.9 Tabu search based techniques

A Tabu search based technique schedules operations T_i in a high-level synthesis problem [168]. The objective function is $C = W_{CS} * N_{CS} + W_{reg} * N_{reg} + W_{bus} * N_{bus} + W_{fu} * N_{fu} + W_{ic} * N_{ic}$, where W is the weight, N the number, cs the control steps, reg the registers, bus the buses, fu the functional units and ic the interconnections. Three types of moves are performed: moves to change the control step, moves to change the functional unit assignment and moves to change the functional unit input assignment of variables. The move type is selected probabilistically based on its importance on the objective function. A number of the selected move is generated. Although the derived solutions are always feasible, not all of them pass the aspiration criterion. If the best applied move is not in the Tabu list or if the cost after performing the move is less than the aspiration level criterion, it is accepted. Otherwise another set of the same candidate moves is generated. The process terminates after a given number of iterations. Another tabu search based technique is the ant colony optimization. For a number of colonies each ant gradually reaches a solution by

probabilistically deciding the applied moves [50] [53]. The latter are maintained in a list and the best solution of each colony reinforces its moves in the next colony.

7.5.10 Seed based techniques

A Seed based technique is a crude Monte Carlo scheduling technique [144]. The objective function can be expressed as $C = f(Time_{tot})$ or $C = f(Time_{idle})$. The constraints D include the number of resources. It samples permutations at random. The best permutation is selected from a large number of samples without checking for repetition or using already found better solutions. Another similar technique is a Chain Monte Carlo which selects permutations at random within a certain distance according to some current permutation measure until a better one is found. Then the latter becomes the current permutation and the sampling is continued until no improvement occurs for some time [144].

7.6 Conclusions

In this chapter, we present a systematic classification which provides a complete and consistent overview of the different categories of techniques solving design time mapping problem on single and multiprocessing platforms emphasizing on ordering in time and assignment in place. The classification has derived by applying the reusable DSE framework principles in the classification context. Our classification is capable of classifying in a systematic way any existing and future technique satisfying our assumptions creating a reliable and generally applicable scheme. In contrast with previous approaches, our proposal classifies a technique to one primitive class or to a hybrid combination of such classes. The decomposition into primitive components enables their systematic and efficient classification. The classes have unique set of characteristics describing non-overlapping search areas. Hence, our classification is able to straightforward classify even complicated techniques allowing their in-depth understanding, i.e. their main characteristics, their structure and the interrelationships of their classes. This is essential for the efficient use of the techniques. The thorough understanding is of crucial importance in the identification of less explored areas and the improvement of the existing and in the development of new global scheduling techniques, as described in Chapter 7.

Chapter 8

Methodology to develop design-time scheduling techniques under constraints

8.1 Introduction

The scheduling techniques heavily affect the system design and performance, as they are responsible for meeting the system specifications, e.g. real-time behavior, minimal energy consumption, reliability etc. The scheduling technique assigns operations, groups of operations, memory references or communication transactions to control steps and hardware resources (homogeneous or heterogeneous processing elements, function units, memories or intercommunication networks) to these operations. The dependencies and constraints should be satisfied, whereas the values of the functions, which describe the crucial objectives of the system, should be minimized. Constraints are imposed by the environment, e.g. a deadline before which the task must finish its execution, by the task, e.g. ordering in data fetching and operation execution, and by the target architecture, e.g. number of interconnections, storing and execution components. Crucial objectives are the required control steps to finish execution, the required hardware resources etc. and also combinations of different objectives.

Although decades of research and experiments have been invested in the design-time scheduling on hardware platforms with single and multi instruction set processors, significant space for improvement exists [182]. Software tools offer automatic scheduling of applications to the target architectures. However, they provide limited coverage, as they apply only specific scheduling techniques for all application domains, e.g. Cheddar [181]. Usually the tools explore a wide range of scheduling instances, which are, however, restricted and focused on a relatively limited exploration area. When the application characteristics do not match very well with the explored area (i.e. the scheduling technique to reach (near-)optimal schedule for this application domain has different characteristics than the available tool techniques), this approach leads to less efficient designs. Tools cannot provide advice on which scheduling techniques are most promising to minimize the crucial objectives for a given application domain. A framework to support the decision of a promising scheduling strategy for a problem domain is missing [5]. Applying a very broad exploration is

a very difficult and time consuming task due to the high number of different in nature scheduling approaches and scheduling parameters. Hence, the designers, usually, develop scheduling techniques which can potentially achieve (near-) optimality for very similar applications by following ad-hoc or less systematic ways, e.g. trial and error, based on their previous expertise. This process requires costly design iterations. Usually, the practical application domains include different in nature application instances with complex and large CDFG and real-time constraints to be met. Then, the aforementioned techniques sacrifice the schedule quality or significantly increase their execution time, e.g. Integer Linear Programming (ILP) scheduling techniques guarantee optimality, but are applicable for small CDFG. A systematic methodology to develop, from the very early steps of the design process, scheduling techniques, which meet domain constraints, e.g. schedule (near-)optimality and technique scalability, is highly desired, as design effort and time are reduced.

In this chapter, we use the framework for design-time scheduling techniques created in Chapter 7 by applying the principles of the reusable DSE to create a systematic methodology which develops parametric template for design-time techniques to schedule single-thread applications of embedded systems in single and multiprocessing platforms, under near-optimality and scalability constraints for large and complex CDFGs. The principles of the usage of the framework described in Chapter 2 are applied in the context of developing design-time scheduling techniques which are near-optimal and scalable for a target domain, which is covered by the root of the classification tree of Chapter 7. The proposed methodology is a projection of the framework usage process described in Chapter 2. The proposed methodology consists of four steps. The first step initializes the structures used in the next methodology steps. The second step analyses the application and platform domain properties, such as the size and the structure of CDFGs, the required optimality of the solution, the available search time etc. These properties are characterized as the domain constraints, which should be satisfied by the developed scheduling techniques. In the third step, the domain constraints are vertically propagated based on the vertical constraint propagation principle to a complete set of non-overlapping classes, which are structured as a tree. Each class describes a different scheduling approach through a prestored template with parameters and functions, as described in Chapter 7. During vertical constraint propagation, the domain constraints prune the incompatible classes. For instance, an optimality domain constraint prunes the near-optimal scheduling class. The result is a reduced tree with the compatible classes. In the forth step, the tree is flattened, the prestored templates of the compatible classes are selected and combined into a parameterized template by following the horizontal design constraint propagations. The combined parameterized template is the output result of the proposed methodology and describes the proposed scheduling techniques for the targeted application domain. We demonstrate the proposed methodology in an application domain with complex and large CDFG and real-time constraints. We also present variations of the objective functions and the CDFG characteristics to show how they affect the space exploration. A lower bound on performance gain of the combined parameterized template of 13-18% is estimated for several real-time applications of the Mibench benchmark suite, e.g. 18,2% for Forward DCT in jpeg encoding. The output result can be used to guide the automatic scheduling provided by existing software frameworks: the most promising

technique for a specific application of the domain can be efficiently selected from the template by instantiating the parameters. This process will generate the actual instantiated scheduling technique, which will be implemented in the software framework. The instantiation process is not covered by the dissertation and it is considered as future work.

In the remaining sections, we present the projection of the usage of the framework in the context of developing design-time scheduling techniques, which are scalable and near-optimal for a partially instantiated target domain. The latter, however, should be covered by the goal described in the root of the used framework. Initially, we provide an example to illustrate that the existing scheduling techniques cannot achieve scalability and near-optimality for large and complex CDFGs. In this way we motivate the necessity for a methodology that systematically creates scheduling techniques, which meet the scalability and near-optimality constraints for the domain under study, from the early stages of the design process. In Section 8.3 describes existing design-time scheduling approaches and scheduling software tools. The proposed methodology is presented in Section 8.4 presents and a demonstration case study and several variations of it are presented in Section 8.5.

8.2 Motivation

8.2.1 Target domain

The main objective function is to minimize the execution time meeting the hard real-time constraints, while reducing secondary objectives, e.g. energy consumption, which is proportional to cycle count for important practical target architectures, like VLIWs, that are instruction and data memory/register-file dominated, or RISCs with simple arithmetic units [169]. The platform domain includes a wide range of hardware architectures, where the system control is derived by the instruction set of the processing elements, such as Application Specific Instruction Processors (ASIP). They support loop buffer structures to maintain near-optimal performance and reduce energy consumption [22]. Hence, code transformations to increase schedule freedom in cost of code size are considered less appropriate, as they prohibit the efficient use of loop buffers. Pure hardware controlled architectures, such as platforms with Finite State Machine (FSM) control and pure Application Specific Integrated Circuits (ASIC) are excluded.

The application domain consists of applications for embedded systems, which have hard constraints on time execution, and thus (near-) optimal schedule is crucial to achieve real-time behavior. The applications are described by large and complex CDFG (1st domain constraint) and the operations of the application are separated to basic blocks. The blocks may be parametric on the data input, e.g. the loops iterations in an image processing application depend on the frame size at start-up time. As design-time techniques cannot exploit this information, optimizing transformations across parametric blocks, to create more freedom in the scheduling process, cannot be applied. Several blocks can be optimally scheduled following design-time conventional techniques, e.g. stochastic or deterministic approaches. But some blocks, due to the position of several operations, are hard-to-be optimally scheduled by conventional techniques, containing critical sub-

graphs.

Definition 4. *A critical CDFG subgraph consists of operations with dependencies which allow only few opportunities for parallelism and in order to derive optimal schedule, the ready operations with flexibility should be postponed.*

When the critical subgraphs are optimally scheduled, the complete application can be optimally scheduled, since the remaining operations schedule is relatively easy. Otherwise, potential optimality is not feasible. Hence, the critical subgraph optimality is the second domain constraint. We assume that the critical subgraphs have weak dependencies and thus they can be separated in relative small and uncoupled CDFG subgraphs (3rd domain constraint). On practical industrial contexts, where the overall application CDFG is very large, an upper bound exists for the available execution time of the design process. Hence, the fourth domain constraint is to keep the search time low and thus to have a technique with complexity scalable to large application graphs and as close as possible to linear (and certainly not exponential) in the overall problem size. Section 8.5 presents the results of the methodology applied in the target domains with near-optimal critical subgraphs constraint and with strongly connected subgraphs constraint.

8.2.2 Performance of scheduling techniques

This section describes the performance of existing conventional scheduling techniques and the techniques developed by the proposed methodology for the target domain of Section 8.2.1. We explore the worst case, i.e. the critical subgraphs position is such that the impact on the overall schedule cannot be hidden by scheduling other operations. The critical subgraph is in the very last operations of a basic block or all next operations fully depend on the subgraph output.

One critical subgraph is depicted in Fig. 8.1-(a), which shows the end of a block of a large application CDFG. It consists of five operations, which require the same type of hardware resources. Two hardware resources exist to perform this operation type. The number next to each node describes the execution time of the operation. A limited amount of parallelism available through operation (e). The critical path is a-b-c and a-d-c. Applying conventional scheduling techniques, which are presented in Subsection 8.3.2, a near-optimal schedule is derived for this critical subgraph. The deterministic techniques compose a schedule based on the dependencies of the CDFG and start scheduling from the entry nodes of the large CDFG [105], pulling the nodes upwards. When they reach the last operations, they end up with a near-optimal schedule due to the lack of a look ahead scheme and the lack of compromises between early and late decisions [200]. In the examples of Fig. 8.1-(a), they select the operation a, as it belongs to the critical path. The operation e, which is ready to be scheduled, is also selected for scheduling, since no other node of the critical path is ready. The schedule result is depicted in Fig. 8.1-(b). The stochastic techniques do not intent to place the operations at any specific position. The postponement of the operation e, required to achieve optimal schedule (Fig. 8.1-(c)), is based on random moves. To achieve an optimal solution, a more global view on the overall scheduling situation involving more than the

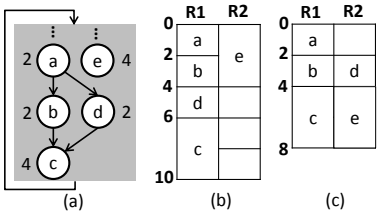


Figure 8.1: (a) Critical subgraph at the end of one loop of a large CDFG. A near-optimal schedule (b) has two cycles difference with the optimal schedule (c) in one loop iteration.

neighboring CDFG nodes is required. In existing conventional techniques, the global view can only be systematically obtained by a time-consuming global exploration techniques, such as approaches that apply a very detailed search at the fine-grain operation level (e.g. ILP solvers) or very gradually evolving stochastic techniques. Both types are too slow to be scalable to large real-time CDFGs with critical subgraphs with more operations and with platforms with multiple hardware resources. Efficient partitioning which leads to optimal schedule, is not possible, as conventional techniques are incapable of exploiting the application structure on the fly to identify the critical subgraphs.

The developed template by the proposed methodology describes techniques which optimally schedule the critical subgraphs of the CDFG. The detailed description is provided in Section 8.5. An Adaptive Simulated Annealing (ASA) based stochastic part is used to quite fast and optimally schedule the easy to be scheduled operations (outside the critical subgraphs). The time that ASA requires to converge in a temperature step is analyzed to find the critical subgraphs. When the convergence time is increased, a critical subgraph exist within the range of the ASA random moves. Then, the ASA part is terminated. The critical subgraphs are propagated to a $B\&B$ part to finalize the search. Since the $B\&B$ is performed only in the critical subgraphs with a good initial solution (ASA schedule), it can apply a fine-grained search within the limited CPU time and identify that the postponement of the operation with parallelism, i.e. e, achieves the optimal schedule, as depicted in Fig. 8.1-(c). In this way significant exploration space is pruned and the complexity is reduced, even for large application sizes. We will come back in more detail in Section 8.5.

8.3 Related Work

8.3.1 Scheduling software tools

Several existing software tools automatically schedule an application to the target hardware platform. They include a library with a range of conventional scheduling techniques, but, they provide a relatively limited coverage. In practical application domains, e.g. domains with complex and large CDFG and real-time constraints which need to be met, the tools use conventional techniques, which either sacrifice the schedule quality or significantly increase execution time leading to less efficient results, as explained in Section 8.3.2. For instance, Cheddar [181] includes most of classical real-time scheduling techniques and provides tools to the designer to compose design

specific schedulers. Times [102] uses automata to describe scheduling strategies, e.g. Fixed Priority Scheduling (FPS) or Earliest Deadline First (EDF). A tool which uses a greedy algorithm to allocate tasks to processors is presented in [14]. GAST [79] decomposes several of the existing scheduling policies into independent parameters in its library. CASCH [2] includes several of the existing techniques, e.g. list scheduling and clustering [79], allowing the designer to select different techniques to evaluate their performance in specific applications and architectures. Parallax [111] offers an environment with seven classical scheduling techniques on various architectures [104]. Hypertool [205] is used in message passing systems and applies scheduling based on the critical path methods. PYRROS [210] provides automatic scheduling for static graphs based on the Dominant Sequence algorithm to cluster the task graph applying a balanced mapping. PARSA [180] applies automatic scheduling on multiprocessors systems based on Heavy Node First (HNF), Linear Clustering (LC) and Linear Clustering with Task Duplication (LCTD) partitioning and scheduling techniques. The existing tools offer a wide range of scheduling instances, which are, however, (near-)optimal approaches for dedicated applications, i.e. the applications which match very well with the scheduling techniques that the tool explores. The existing tools miss a way to guide which scheduling technique is most promising to be used in an application domain. For instance, SUIF [185] consists of scheduling modules, which allow the designer to create a scheduling approach. Parafrase-2 [158] is a tool for experimentations with transformations and scheduling approaches for parallel supercomputers. Hence, the development of the scheduling technique is left fully to the designer. In order to improve the design process of scheduling techniques, effort from the designers is required. The alternative is a very broad design space exploration which involves a very difficult and time consuming task due to the high number of different in nature scheduling classes and scheduling parameters. Hence, the designers develop conventional scheduling techniques to potentially achieve (near-)optimality for very similar application instances by following ad-hoc or trial and error ways based on their previous expertise.

8.3.2 Conventional Scheduling Techniques

The scheduling techniques considered as conventional may belong to one scheduling class, i.e. stochastic or deterministic, or they are hybrid techniques, which are combinations of different scheduling classes. In practical application domains with complex and large CDFGs, as described in Section 8.2, which require (near-) optimal solutions, the conventional techniques sacrifice quality or require prohibited execution time.

The *deterministic techniques* cannot achieve (near-) optimal solution in these practical application domains, as they are applied in the complete large CDFG leading to significant increase to their execution time. The time required for deterministic techniques to converge is relative to the problem size [12], which is crucial for large CDFGs. In addition, they cannot efficiently select the direction of perturbation to visit all local minima in the search space [66] and a local minima may trap in a sub-optimal search [84]. The deterministic techniques compose a schedule based on the dependencies of the CDFG. They start from the initial operations of the basic blocks of the

CDFG, as they traverse the graph from the entry nodes to the exit nodes [105], pulling the nodes as much as possible upwards. When they reach the operations of the critical subgraphs at the end of the basic blocks, they end up in near-optimal schedule for the last operations. In order the deterministic techniques to achieve an optimal schedule for the critical subgraphs, a very detailed search at the fine-grain operation level is required to identify the optimal schedule from the large number of near-optimal schedules. Otherwise the result will not be optimal, due to the lack of a look ahead scheme and the lack of compromises between early and late decisions [200]. As the number of application operations increases, the deterministic techniques would require too much CPU time to perform this very detailed exploration.

For instance, a *formally optimal/enumerative* technique is an unacceptable option due to the prohibited CPU time that requires [136]. Such techniques are: e.g. ILP formulation to find periodic schedules under timing and resource constraints presented in [23], the Mixed-Integer Bilinear Programming (MIBP) formulation to find optimal solutions to the Multiprocessor Scheduling Problem with Communication Delays (MSPCD) in [40] etc. An *adaptive deterministic* technique tries to converge by using implicit correlations in the shape of the search space. Due to the critical subgraphs, they cannot efficiently exploit the space, and thus the complexity grows (far) beyond linear in the problem size. Similarly, applying a *Branch & Bound (B&B)* method directly to the initial space leads to a prohibitive complexity and CPU time, when a domain constraint for optimality exists. Such techniques are: e.g. the technique to assign tasks to heterogeneous computing systems of [81], the task assignment to processor cores of [117], where nodes with the maximum edge weight priority are merged and a backtracking mechanism is available, Ref. [213] which uses the least function cost as priority to expand the W most promising nodes in breadth-first order, where W is the search width, a version of the A* algorithm of [156] is applied in multiprocessor task scheduling problem in [155] etc. A *heuristic* approach cannot guarantee the schedule optimality in a broad domain. It searches with predefined rules with which is almost impossible to achieve optimal results. No single good heuristic exist for prioritizing the CDFG nodes across a range of applications using list scheduling [200], as none heuristic can be applied to the problem in its most general form [39]. The best choice depends on the CDFG structure [199].

The *stochastic techniques* cannot find the (near-) optimal solution with linear complexity in terms of the problem size for these practical application domains due to their probabilistic nature, which does not intent to place the operations at any specific position. The postponement of the scheduling of the ready operations with flexibility of the critical subgraphs, which is required to achieve optimal schedule, as explained in Section 8.2, is based on randomness. The identification of the optimal schedule of the large and complex CDFGs will require too large time. The more theoretically optimal stochastic techniques may lead to (near-)optimal solution but they require too much exploration time. Hence, using only a pure stochastic scheduling technique leads to suboptimal schedules, as they either require too much time to converge or when they are early terminated, a good, but typically not sufficiently optimal schedule is found. For instance, such techniques are: ant colony optimization [53], where each ant gradually reaches a solution by deciding probabilistically the applied moves. A tabu search scans the neighborhood by creating several candidates

selected by probabilities [168], [50]. A Simulated Annealing (SA) technique for periodic and aperiodic tasks onto pipelined parallel architectures is presented in [145]. The multiprocessor scheduling problem is solved by a SA in [211] and by a SA with static parameters in [137]. A Genetic Algorithm (GA) for multiprocessors is presented in [112], where the chromosome describes the task and the processor and the fitness function is the normalized average response time. A problem-space GA technique for precedence constrained task graphs with non-negligible intertask communication is presented in [44]. Simulated Evolution (SE) [119] selects a partial schedule by the objective function and generates the complete schedule using a freedom based priority.

The alternative is to use techniques which are combinations of the deterministic and the stochastic classes. Existing combinations are techniques which implement some features of the first class, e.g. stochastic (deterministic), using the second class, e.g. deterministic (stochastic). In this way the combination is a heuristic of the primary class, which behaves in a similar way to the primary class without really exploiting the benefits of the second class. For instance, the chromosomes of Genetic Algorithms (GA) in [10] and [45] provide the priority of the CDFG nodes in order to be scheduled. In [204] a Simulated Annealing (SA) technique defines the CDFG nodes to be scheduled. Other existing combinations use completely the stochastic and the deterministic class. They are usually heuristics which may achieve the optimal schedule but for one specific application. They provide a less efficient schedule for the different applications in the target application domain since their complexity is increased to find the (near-)optimal solution. For instance, [178] presents a combination of a GA heuristic with a $B\&B$ method for large scale resource allocation which achieves a near-optimal schedule. However, the GA heuristic cannot identify the critical subgraphs and thus the $B\&B$ method has to be applied in the overall search space. In this way, the complexity is increased and the optimal solution cannot be identified within the available CPU time. A Simulated Annealing (SA) technique [153] is interleaved with a local algorithm to search in nearby regions for a better solution. This combination has still increased complexity and cannot guarantee optimality for the critical subgraphs.

Conventional techniques have deficiencies to fully meet the domain constraints and achieve a (near-)optimal schedule for domains with complex and large CDFGs. The proposed methodology supports the development of scalable and (near-) optimal design-time scheduling techniques for applications with complex and large CDFG and real-time constraints by efficiently searching the overall exploration space from the early stages of the design process without applying costly iterations, as explained in the next Sections.

8.4 Proposed Methodology

The target domain of scheduling techniques under study is described by:

1. Techniques which are applied at Design-time,

2. Techniques which search for the schedule in the complete exploration space,

Table 8.1: Summary of methodology notation.

Definitions	
Top-down Split	P class split into complementary, positive,non-overlapping, complete subclasses $S1$ and $S2$
Brother classes	Subclasses $S1$ and $S2$
Couple of Classes	Couple of classes m and n is: $m - n$
Class Combination	An order list of classes
Axiom	
Valid Brother classes	Uni-directional arrow of $S1$ and $S2$ is met
Lemmas	
Valid Couple of classes	Layer where m and n ancestors are valid brothers
Valid class combination	The order list describes only valid couples of classes
Assumptions	
Scheduling technique	Combination of classes
Combination of classes	Satisfying uni-directional arrows
Theorem	
Valid Scheduling technique	Valid Ordered list of classes

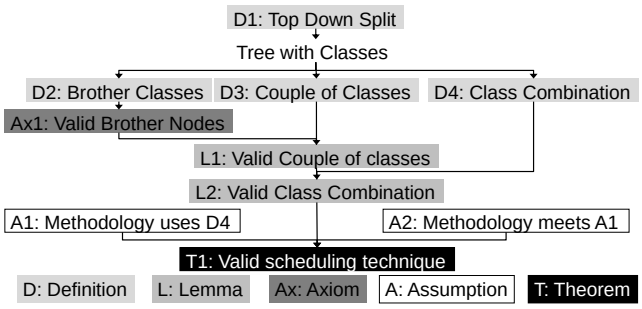


Figure 8.2: Dependencies of the methodology notation.

3. Time and Resource constraints, e.g. real-time constraints which require an optimal solution

4. The schedule objectives are not restricted to one dimension, but they may express competitive axes, formulating a multi-dimensional objective space.

A summary of the notation of Chapter 2 and Chapter 7 is depicted in Table 8.1. Fig. 8.2 describes how the notation of Chapter 2 is combined to derive the theorem of the proposed methodology. The top-down split definition is used to compose the tree with the design-time scheduling classes. Based on the tree, the terms of the brother classes, the couple of classes and the combination of classes are defined. The axiom of the valid brother nodes, in combination with the couple of classes, leads to the valid couple of classes lemma. The latter, combined with the class combination definition, leads to the valid class combination lemma. The assumptions that the proposed methodology uses class combinations, which satisfy the axiom of valid brother nodes, combined with the valid class combination lemma, leads to the main theorem of a valid scheduling technique.

The framework with the partitioning of all the available scheduling approaches of the techniques under study is described by the classification tree T depicted in Fig. 8.3-(a), which derives by applying top-down splits as described in Chapter 7. Further potential in-depth extensions of the tree structure can be applied by applying new top-down splits following the methodology of Chapter 2.

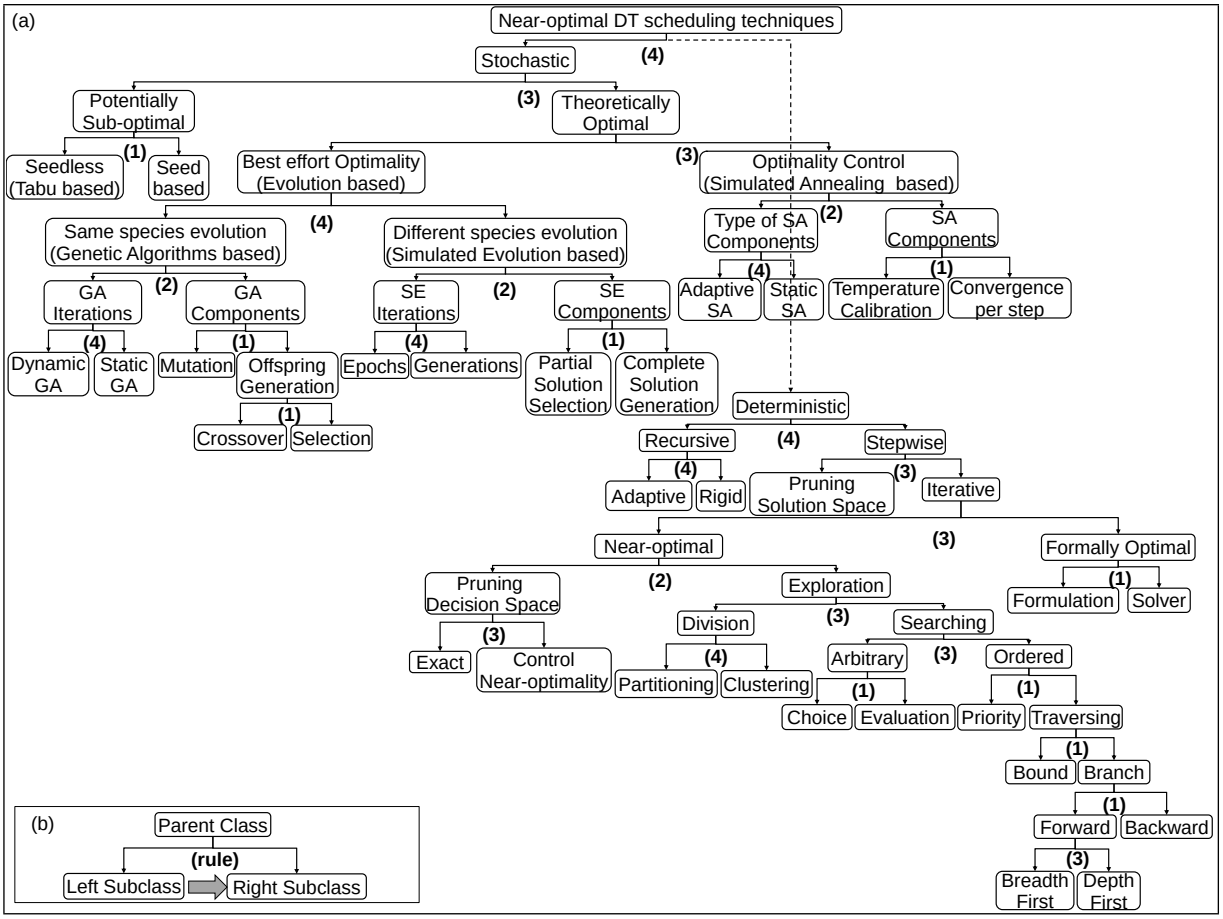


Figure 8.3: (a) The tree describing the complete set of design time scheduling techniques and (b) Uni-directional design constraints propagation of brother nodes.

For instance, the root, which is the scheduling techniques under study, is split to stochastic and not stochastic class. The latter is reformulated to a positive description, i.e. the deterministic class. In each level, top-down splits are applied, which lead to complementary, non-overlapping, positive and complete subclasses. This process is enabled by the positive reformulation of the subclasses which allows the splitting process to continue. Hence, characteristics and class loss cannot occur during the class creation process. The final classes cover all the possible options and describe non-overlapping areas of the search space, as explained in detail in Chapter 7. Since the classes are allowed to be combined following the uni-directional arrows, the complete set of techniques is described.

A scheduling technique may belong to only one class or to a combination of classes in order to share the unique characteristics of each class. The class combinations are performed by propagating the characteristics of one class, as design constraints, to the next classes. The ordering of the propagation expressed through an arrow in the top-down split (Fig. 8.3-(b)). The arrow source is the left subclass and propagates the design constraints to the destination, i.e. the right subclass.

Based on the combination principle, axiom 1 is introduced to describe the valid combination of brother sub-cases.

Axiom 1. *A combination of brother classes is valid, when the uni-directional arrow is satisfied.*

Lemma 1. *A couple of classes $m - n$ is valid if: (1) the classes m and n or (2) the class m and one ancestor of the class n or (3) the class n and one ancestor of the class m or (4) the ancestors of both classes m and n, is a valid combination of brother classes (Axiom 1)*

Proof. The classes are structured in a tree T. Based on the tree structure, the brother classes can be: (1) the classes m and n of a couple of classes $m-n$, (2) one class m (class n) with one ancestor of the class n (class m) and (3) the ancestors of class m and class n. In the first case, the classes m and n of the couple $m - n$ are brother classes. Combining two brother classes, following the opposite direction of the uni-directional arrow of the corresponding split in T, composes an invalid combination of brother nodes (Axiom 1), which is an invalid couple. When the ancestors of both classes m and n or a class m (class n) with the ancestor of the class n (class m) are brother classes, they belong to a lower depth of T. In this level, the brother classes should satisfy the uni-directional arrow in the corresponding split to be a valid brother combination and, thus, the classes to be a valid couple. □

Definition 5. *A class combination is described by an ordered list of classes $\{i, ..., n\}$. The order describes the direction of combining the classes, which indicates a path in T.*

Lemma 2. *A class combination is valid iff every couple of classes, which follows the order of the list, is valid.*

Proof. -If the ordered list is valid, then every couple of classes, which follows the order of the list is valid: Only one class is an ordered list of one element, which is always valid since no combinations can be created. When the ordered list consists of more classes $\{i, ..., n\}$, the list describes several couples of classes which follow the list ordering, i.e. how the design constraints have been propagated, when the classes have been combined to form couples and thus the final path in T. A valid ordered list describes an ordering of combining classes which satisfies all the arrows in the path. Hence, all the class couples, that have been combined based on the order provided by the list, are valid.

-If every couple of classes in the list is valid, then the ordered list is valid: When all the couples of classes are valid, they follow the uni-directional arrow in the corresponding level for each couple. The combination of all the valid couples describe the final path in T. The final path in T is the ordered list of classes, which is valid since the path in T is valid and is provided by valid couples. □

We assume that the following assumption hold in the proposed methodology:

Hypothesis 1. *The proposed methodology develops a scheduling technique by combining T classes.*

Hypothesis 2. *The class combination is performed by following the uni-directional arrows in T splits.*

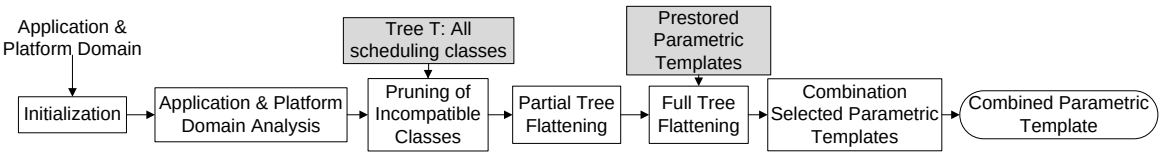


Figure 8.4: Flow chart of the proposed methodology steps.

The above assumptions are verified in the next sections, which analyze the proposed methodology and provide a demonstration case study. The main theorem of the developed methodology is:

Theorem 4. *A valid scheduling technique is a valid ordered list of classes.*

Proof. Based on Hyp. 1, a scheduling technique is composed by combining one or more classes. In the former case, one class is an ordered list of one element, which is always a valid ordered list, thus the technique is also valid. In the second case, several classes are combined forming a class combination expressed by an ordered list of tree classes $\{i, ..., n\}$. The list is valid, when the class combination is valid (Lemma 2). Suppose that the proposed methodology composes a valid scheduling technique, which is described by an invalid ordered list of classes. Then, at least one couple of classes, which follows the order of the list, is invalid. Hence, the corresponding uni-directional arrow for the invalid couple is not satisfied. The latter contradicts the Hyp. 2. □

The proposed methodology develops design-time scheduling techniques for an application domain by guiding in a systematic way the valid combination of classes, i.e. by creating valid ordered lists. The flow chart and the steps of the proposed methodology is depicted in Fig. 8.4 and Alg. 13, respectively. In the next paragraphs, we explain in details the steps.

The inputs of the proposed methodology are the application and platform domain. The tree T with all scheduling classes and the individual parametric templates of each class are prestored. A prestored code template describes the parameters and the functions of the scheduling class. For instance, the prestored template of an SA component consists of parameters and functions which describe: the type of the components used in the SA, the way the SA temperature is calibrated during iterations, e.g. $Temp = Temp * K$ [137], the range of the random moves that the technique applies in each step, the number and type of the random moves that are applied in each step, the criteria that are used to accept a new schedule, e.g. $e^{\frac{W_{old}-W_{new}}{Temp}}$ [137], the criteria that are used to decide if equilibrium has been reach, i.e. the technique has converged for a specific temperature and the final convergence criteria, which terminate the technique, e.g. when $Temp < 1$. The prestored template of a B&B consists of the parameters which describe how the pruning of the search space is performed in each step, the priority function to sort the CDFG nodes, the initial bound estimation, e.g. a tight upper bound in the objective function [72], the function of updating the bounds, the forward and backward CDFG branch, e.g. the node with the least bound is selected for forward branching [209]. All T classes have similar predefined parametric templates.

ALGORITHM 13: Proposed methodology.

Input: Application and Platform Domain
Output: Combined Parameterized Template of Developed Techniques
Step 1\
Class list Q ← The set of all T classes
v ← root of T
Search list S ← ∅
Enqueue(S,v);
Pruned Classes list P ← ∅
Step 2\
Constraints = Analysis(App.&Platf.Domain);
Step 3a\
while *(S ! = ∅)* **do**
  Dequeue(u,S);
  for *(i=0;i<NumOfConstraints;i++)* **do**
    if *(Label(u) ≠ constraint[i])* **then**
      Dequeue(u,Q);
    else
      Enqueue(S,LeftChild(u));
      Enqueue(S,RightChild(u));
Step 3b\
for *(i=0;i<length(Q);i++)* **do**
  Dequeue(i,Q);
  while *(NumOfChildren(i)==1)* **do**
    Dequeue(Child(i),Q);
    u=Merge(i,Child(i));
    Enqueue(Q,u));
Step 4a\
Search list S ← ∅ ;
v ← root of Q ;
Enqueue(S,v));
while *(S! = ∅)* **do**
  Dequeue(u,S);
  if *(NumOfChildren(u)==0)* **then**
    n=Merge(u, acc);
    Enqueue(F,n);
    if *(Arrow(u) exists)* **then**
      Enqueue(S, Dest(Arrow(u)));
    else
      Backtrack();
      Acc=Remove(u,Acc);
  else
    Acc=Merge(u,acc);
    Enqueue(S, LeftChild(u));
Step 4b\
for *(i=0;i<length(F);i++)* **do**
  t=SelectTemplate(i)
  AddToCombinedParametric(t)

8.4.1 Step 1: Initialization

The Class list Q describes the selected scheduling classes for the application and platform domain under study. It is initialized with all the available scheduling classes provided by T. A list S is used to support the efficient search of the tree T and it is initialized with the T root. The Pruned classes list P describes the scheduling classes which are incompatible with the domain constraints and it is initialized with the empty set. To efficiently schedule the domain, the developed techniques should at least have one class per domain constraint.

8.4.2 Step 2: Application & Platform Domain Analysis

The application and platform domain analysis derive the domain constraints which the developed scheduling techniques should satisfy in order to achieve a (near-) optimal schedule. The domain constraints describe relevant information from the application structure which can derive from profiling tools, e.g. the size of the application (internal kernels, loop bounds), the available parallelism due to dependencies, the existence of cross-iteration dependencies, which define the real critical path etc, and the objectives of the problem under study, e.g. optimality in the cost function, low execution time. The developed techniques are described by the class combination of the selected classes, which satisfy and match with the domain constraints. In this way, the developed

techniques have the required characteristics to achieve an efficient schedule.

8.4.3 Step 3: Propagation of Domain Constraints

8.4.3.1 Step 3a: Pruning Incompatible Classes

In this step, the compatible classes are selected based on a systematic propagation of the domain constraints in the tree T, which prunes the incompatible classes. The selection is performed in each level of the tree T by comparing the characteristics of a class with the set of imposed constraints through labels. The characteristics are used to label the classes with the promising domain constraints, i.e. the domains where the techniques described by the class are very promising approaches. When a propagated domain constraint matches with a label, the class is selected. If the characteristics of one class are incompatible with a constraint, the class is removed from the list Q. It is placed in the list P and, thus, it is not further explored. In this way an efficient search in the T is achieved, since the subclasses of the pruned class are also incompatible classes, due to the tree structure. If the characteristics of the class meet the propagated domain constraint, the class will be part of the developed scheduling technique. In this way, some classes are favored against others for a given application domain. The children of each compatible class are further exploited. The class selection is performed for all the domain constraints determined by the previous step of the proposed methodology. The output of the propagation of the domain constraints step is a reduced tree described by the list Q. The domain constraints, which are propagated to the tree, are also propagated to the prestored parametric templates of the selected classes. The propagated constraints partially define the templates, because they provide values for several parameters. For instance, the reduced search time constraint implies that the parameter describing the type of the SA components should be adaptive, since it more efficiently searches the space. The estimation of the bounds in the B&B template is very crucial, as it is responsible for the optimality of the obtained schedule. The optimality domain constraint partially defines the B&B template by using an exact minimum cost for the node bounds. Further illustration of this step is provided in Section 8.5.

8.4.3.2 Step 3b: Partial Tree Flattening

Since several classes have been pruned from the tree due to mismatching with the domain constraints, several of the selected classes have remained with only one compatible child. The tree is further reduced by merging the parent class with the unique child class into one class in the tree described by Q.

8.4.4 Step 4: Propagation of Design Constraints

In this step, the partially defined parametric templates are merged in a systematic way to the combined parametric template, which describes the set of the developed techniques. The systematic

combination follows the uni-directional arrows of the tree T, derived from Axiom 1, to compose an Ordered List (OL) of the selected classes.

8.4.4.1 Step 4a: Full Flattening of the reduced tree

The process starts from the root of the tree Q and it applies an in-depth search. It first selects the left class in every split, which is the source class of the arrow. During the in-depth search, the characteristics of the classes in this path are accumulated. When the search reaches the class in the last level of the first left branch, e.g. N, the accumulated characteristics of the classes of the $(N-1)$ levels and the characteristics of the last class of level N are merged into a flattened class, which fully describes the path from the root to the last class. The flattened class is stored to the ordered list OL. Then, if an arrow exists in this level (i.e. N) from the current class to a right class, the process follows the arrow. If the right class is also at the last level of the right branch, the accumulated characteristics from the classes of the $(N-1)$ levels are merged with the characteristics of the right class, creating a new flattened class, which is stored in OL. If the right class is not the last level, but it is further split (level $N+1$), the in-depth search and the accumulation of characteristics is repeated. If no arrow exists from the left class, the search backtracks one level (i.e. $N-1$) to the parent class. The characteristics of the parent class are removed from the set of the accumulated characteristics (which now describe the characteristics of the classes until $(N-2)$ level). The process checks if an uni-directional arrow exists in $(N-1)$ level. If the arrow is missing, the search backtracks one more layer and the process is repeated until an arrow is found. The process terminates when the root has been reached through backtracking steps. The output is an ordered list OL with the flattened scheduling classes and their ordering.

8.4.4.2 Step 4b: Combination of prestored parametric templates

The combination of the flattened classes is the output of the proposed methodology, i.e. a combined parametric template with the developed techniques which satisfy the domain constraints, thus providing a (near-) optimal solution for this domain. Each flattened class is described by the prestored partially defined parametric template. The combination is performed by selecting the prestored template of the first class in the ordering of OL and propagate the template output and to the template of the next class in OL.

8.4.5 Instantiation of combined parametric template

The output of the proposed methodology can be used to efficiently select a specific scheduling technique. This process provides specific values to the parameters of the template based on the specific application instance and, thus, instantiates it to a specific technique. A similar methodology can be created to guide the template instantiation by refining the principles of the proposed methodology. The instantiation will start from the selection of the values of the parameters and the functions of the prestored template of the first class. The decisions are propagated as a design

constraint to the next templates following the arrows. Then, the parameters of the next template are instantiated under the design constraints of the instantiated parameters of first template. The step is repeated, until all template parameters have specific values.

8.5 Demonstration case studies

This section demonstrates how the proposed methodology leads to a combined parametric template to develop techniques for the main demonstration case study, i.e. the domain characterized by applications with large and complex CDFG with real-time constraints, which is described in Section 8.2. We also present several variations of the objective functions and the domain characteristics to illustrate how the proposed methodology leads to different combination of classes.

8.5.1 Small and uncoupled critical subgraphs

The main demonstration case study is the target domain described in Section 8.2.

8.5.1.1 Step 1: Initialization

The Class list Q is initialized with all T classes, list S with the T root and the Pruned list P with the empty set.

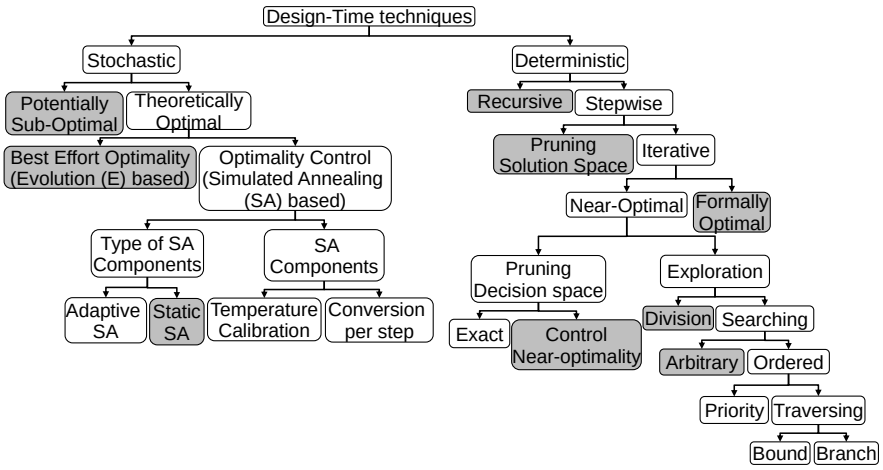
8.5.1.2 Step 2: Application & Platform Domain Analysis

The analysis, which is described in Section 8.2, leads to the domain constraints of: large applications, optimal schedule for the critical subgraphs, relative small and uncoupled subgraphs, linear complexity in terms of problem size in order to find the (near-)optimal schedule within a reasonable search time.

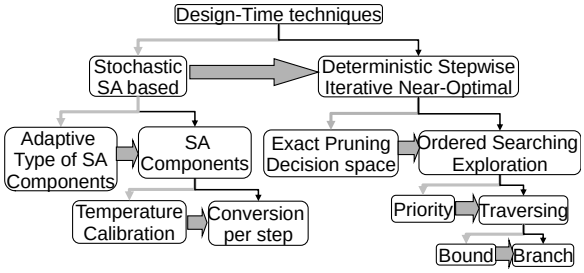
8.5.1.3 Step 3: Propagation of Domain Constraints

8.5.1.3.1 Step 3a: Pruning Incompatible Classes The set of domain constraints is propagated to the tree T. The next paragraphs explain how the propagation of the domain constraints to the initial tree of Fig. 8.3-(a) selects the classes of the scheduling technique. The result is Fig. 8.5(a), where the pruned classes are marked with gray.

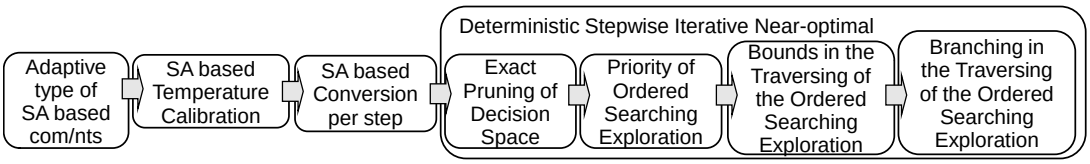
Propagating the domain constraint of large CDFG, the stochastic class (top left part of Fig. 8.3-(a)) matches quite well with this constraint. The characteristics of the stochastic class is that it finds feasible schedules in problems with high dimensions [172] due to the random moves that applies. The probabilistic way of accepting a schedule allows to ``hill climbing'' moves to occur and thus the search can escape from a local minima [211]. The domain constraint of optimal schedule of the critical subgraphs prunes more the stochastic classes which are incompatible with this constraint. In order to have optimality in the subgraphs, an optimality control should be available in the developed scheduling technique, i.e. the scheduling technique should have an mechanism to



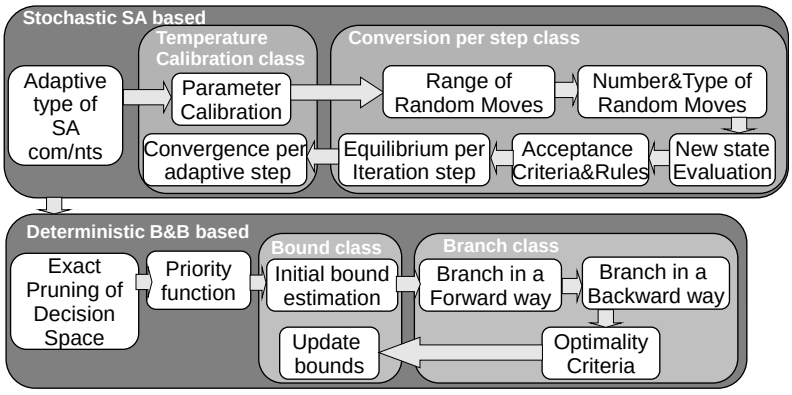
(a)



(b)



(c)



(d)

Figure 8.5: Step Results for the demonstration case study: (a)Domain Constraint Propagation step: the gray classes are pruned due to incompatibility with the propagated constraints, (b) Partial flattening step: the tree with the selected classes is reduced, (c) Full flattening of the tree: the outer box in the deterministic classes is used to describe the common part of the flattened classes to avoid redundancy and (d) Combined Parameterized template

control how optimal is schedule obtained from the scheduling technique, e.g. entropy in SA techniques. Techniques without a schedule optimality control are less efficient in our domain, since they cannot provide guarantee for their convergence process. The potentially sub-optimal stochastic class, e.g. Monte Carlo [144] or tabu search [168], is pruned, because they cannot guarantee the optimality constraint. The Evolution based (E) class is also pruned because it does not have a control mechanism for the optimality of the obtained schedule, e.g. a Genetic Algorithm [112]. The class that meets the optimality domain constraint is the stochastic class with a *control over the schedule optimality* (SA based). The optimality can be achieved, but not in limited time for large scale applications. The constraint of the linear complexity of the scheduling technique in terms of problem size requires a class that is capable of quite fast exploration. However, not all stochastic classes with control over the schedule optimality can reach a quite good solution within a reasonable amount of time. If their stochastic process uses *static type of stochastic components*, they will quite fast reach a good solution only in the specific cases where the parameters of the stochastic components efficiently characterize the solution space, which is quite complex for our domain. Otherwise, small steps and a lot of iterations must be performed by the stochastic class in order to converge [66], increasing the complexity to unacceptable values. Hence, the static type of components is pruned. However, an *adaptive type* is likely to find a schedule with good quality already within a reasonable amount of time. The complexity of the ASA class is proportional to the dimension of the problem by the possibility of having samples in the improved regions [212]. The ASA converges faster when the problem tolerates it and goes (much) slower when the annealing temperature becomes very critical, otherwise the sensitivity of making erroneous decisions becomes too high. The way the annealing temperature is adapted is used to identify the critical subgraphs.

The propagation of the domain constraints matches also with *deterministic* classes (top right part of Fig. 8.3-(a)). They can explore more efficient and faster small areas with several good solution than the stochastic techniques. To ensure the optimality of the critical subgraphs, the tree is pruned to the deterministic classes, which are *optimal* or offer a way to *control the schedule optimality (exact)* of the obtained schedule. The recursive class is also pruned, since it is less useful, when it is combined with an optimal/exact class. The class that prunes the search space in a near-optimal, but controlled way (Pruning Decision Space/Control near-optimality) is removed, because an optimal schedule in the critical subgraphs is required. Because of the domain constraint for reduced search time, the formally optimal class, which meets the optimality constraint, it is pruned since it is incompatible with the linear complexity in terms of problem size constraint. The division class and the arbitrary class are also pruned since they require too much time to reach an optimal schedule. The pruning of the overall solution space in the initial search steps (Pruning Solution Space) class is removed based on the domain constraint of uncoupled subgraphs with weak dependencies. The class which matches with the constraints, i.e. can guarantee schedule optimality and may prune the search space, is the ordered class. It consist of the *priority* function to describe the CDFG nodes sorting, the *bounds estimation* of the schedules and the *branching* during the search. The complexity of a standard $B\&B$ technique heavily depends on the size of the application and the

pruning of the schedules. When it is applied on small subgraphs, the complexity becomes close to linear on the overall problem size and significantly less time is required for convergence comparing with the initial complexity.

8.5.1.3.2 Step 3b: Partial Tree Flattening The output of the pruning of incompatible classes step is a reduced class tree, where several classes have been left with one unique child. The characteristics of these classes are further merged to one class. The stochastic class, the theoretically optimal class and the optimality control (SA based) class are merged into one new flattened class, i.e. the Stochastic SA based class, as depicted in the left part of Fig. 8.5(b). The type of SA components and the adaptive class are combined into the adaptive type of SA components. By applying this process to the pruned tree of Fig. 8.5(a), the tree of Fig. 8.5(b) is composed. Each class is described by a prestored parametric template with the parameters and the functions that implement the class scheduling approach. Due to the propagation of the domain constraints and the flattening step, parameters from the templates take specific values. E.g. the parameter that describes the type of SA components is set to adaptive, thus the exact minimum cost function is selected for the Bounds of the nodes.

8.5.1.4 Step 4: Propagation of Design Constraints

8.5.1.4.1 Step 4a: Full Flattening of the reduced tree Following the in-depth search with priority to the left branch, the characteristics of the left branch are accumulated (stochastic SA based class). In the last level, the characteristics of the adaptive are merged with the accumulated characteristics to compose a flat class, i.e. Adaptive type of Stochastic SA based components of design-time scheduling techniques (left part of Fig. 8.5(c)). Since an uni-directional arrow exists in this level, it is followed to reach the right class. The right class describes the implementation of the SA components, which are further split into the temperature calibration and the convergence per step classes. At this level, a similar merging is applied in order to create the flat classes of the Temperature Calibration of Stochastic SA based component of design-time scheduling techniques and the Stochastic SA Convergence per temperature step, as depicted in Fig. 8.5(c). Since no arrow exists in this level, the search backtracks to the stochastic SA based class, where it follows the arrow to the deterministic branch. In a similar way, the characteristics of deterministic classes are merged and flattened. The result is depicted in Fig. 8.5(c).

8.5.1.4.2 Step 4b: Combination of prestored parametric templates Following the order imposed by the flattened tree, we compose an ordered list of scheduling classes. Based on this ordering, the partially defined parametrized templates of the flattened classes are combined by propagating design constraints to compose the combined parametric template. The partially defined parametric templates of the selected stochastic classes and deterministic classes are depicted by the stochastic and the deterministic box, respectively, in Fig. 8.5(d).

The first is the stochastic template (top part of Fig. 8.5(d)) described by the parameter SA com-

ALGORITHM 14: Combined parametric template.

begin ASA part
  Input: Application,platform domain
  Output: Set of critical subgraphs, Partial scheduling
  $S \leftarrow$ Initial state generation; $Temp \leftarrow$ Init. temp. $Temp_0$;
  while *(Cost(S)>CostThr.)* **do**
    while *(Entropy>Entr.Thr.)* **do**
      R = Sel RM range($Temp$);
      N = Sel RM number($Temp$);
      $Type$ = Sel RM type($Temp$,N);
      S' = Sel random state (Neighborhood(S),R,N,$Type$);
      Cost(S') = evaluate(S');
      if *(Cost(S') < Cost(S))* **then**
        $S = S'$;
      else
        r = random(0,1);
        if $(r < e^{\frac{Cost(S)-Cost(S')}{Temp}})$ **then**
          $S = S'$;
        if $(R \approx R_{min})$ **then** Break;
    Adaptively Update(T);
    Identify Crit.Subgraphs(R,Entropy);

*RM=Random Moves

begin B&B part
  Input: Critical subgraphs, Partial scheduling
  Output: Final scheduling
  for *(i=0;i<Subgraphs;i++)* **do**
    Best $\leftarrow$ Cost(stochastic solution);
    v $\leftarrow$ root of subgraph;
    Node list Q $\leftarrow \emptyset$;
    Enqueue (Q, v);
    for *(j in Subgraph nodes)* **do**
      Compute node bound B_j;
    while $(Q! = \emptyset)$ **do**
      Select n=min(B_j);
      for *(each child u of n)* **do**
        if *(Cost(u) < Best)* **then**
          Best=Cost(u);
          Update(B_j);
        if *(B(u) < Best)* **then** Enqueue(Q,u);
      Dequeue (Q,n);

ponent type, which is set to the adaptive option. The latter is propagated, as a design constraint, to the temperature calibration class, which is described by a parametric function of how the temperature parameter is modified during the search and a parametric function of how the convergence per temperature step is verified. The design constraint propagation imposes to the parametric functions of the temperature calibration class to be adaptive. In this way, when the search enters a smoother area, the temperature modification can be relaxed and the search goes faster. In the critical subgraphs, the temperature change is restricted to continue more carefully. The ratio of the temperature change is used to identify the critical subgraphs (Subsection 8.5.1.5 describes the parametric technique in details). These decisions are propagated as design constraints to the Convergence per step class. The parametric template is described by the parameters that determine the range, the number and the types of the applied random moves to create a new schedule. For instance, one instantiation of the random moves is described in [28], [162]. One parametric function describes the procedure of how to create the new schedule. Another parametric function evaluates the new schedule based on the value of the objective function of the current schedule and decides, in a stochastic way, to maintain it or not. Another parametric function is used to decide if the technique has converged for a given temperature step or another iteration of randomly selected moves is performed for this specific temperature value. Otherwise the convergence criteria determine the overall convergence or decide another adaptive parameter calibration. The adaptive way of modifying the temperature, which is propagated from the previous parametric templates, affect the implementation of the aforementioned parametric functions. The ASA parametric templates pseudocode is shown in Alg. 14.

The remaining parametric templates describe the B&B classes (bottom part of Fig. 8.5(d)).

The parameter that defines the pruning during the search decisions is set to exact. This decision is propagated to the parametric templates of the priority, the bound and the branching class. The bound template is described by the parametric function of estimating the bounds and the parametric function of updating the bounds. The propagated exact pruning constraint implies an exact minimum cost of bounds, i.e. a lower cost than the real lower cost of all the schedules, which pass from each node in the B&B search tree. In this way, it guarantees the schedule optimality in a fully controlled way. The decisions are propagated to the parametric template of the branch class, which includes parametric functions to describe the branching in a forward way, the branching in a backward way and decision over the optimality of the obtained schedule. The forward branching is performed using the estimation of bounds (propagated as design constraint from the previous step) and the current optimal, which is initially the propagated solution. With a good initial solution, i.e. the solution propagated from the ASA class, significant node pruning can be expected. Hence, the complexity of the $B\&B$ is significantly reduced, which also reduces the complexity of the developed techniques. During the forward branching, the node with the smaller cost is expanded. If the new expanded nodes are more costly than the already expanded ones, the search backtracks to the less costly node (function of backward branching). When the real cost of a solution is found, the optimality function verifies the acceptance of the schedule. The process terminates, otherwise a potential update in the estimation of bounds is performed in an exact way and another iteration is applied. The pseudocode of the $B\&B$ part is depicted in second block of Alg. 14.

8.5.1.5 Combined Parameterized Template

The scheduling process described by the combined paramaterized template is illustrated via the example of Fig. 8.6. The light gray boxes describe the main loops of the application. At the left part, the search of the ASA class is described through the temperature reduction and the identification of the Critical Subgraphs (CS). The ASA temperature calibration process is highly benefited from the hierarchical structure of the CDFG of the application. At high temperatures ($Temp$ high), the range of the random ASA moves is large, as depicted by the arrow of $Temp$ high. The schedule of nearly none operation is ``frozen" and the relative position has not been decided yet, i.e. moves applied in the subsequent temperature steps can still easily change the schedule decisions. When the temperature is further decreased (e.g. $Temp$ medium), the range of random moves is also decreased (arrow of $Temp$ medium) and, thus, the impact of the moves in the schedule is decreased.

For illustration purposes, we assume that we have two hierarchical levels: the schedule between the critical subgraphs and the internal schedule of each subgraph (the process can extend further to more hierarchical levels). This hierarchy is typically not known beforehand, so it should be determined on-the-fly during the ASA execution. The ASA temperature calibration process can be performed faster with large temperature steps. The ASA convergence per step process can be performed with fewer iterations of random moves at a given temperature, when the range of the random moves is larger than the typical cost impact of the random moves within one of these

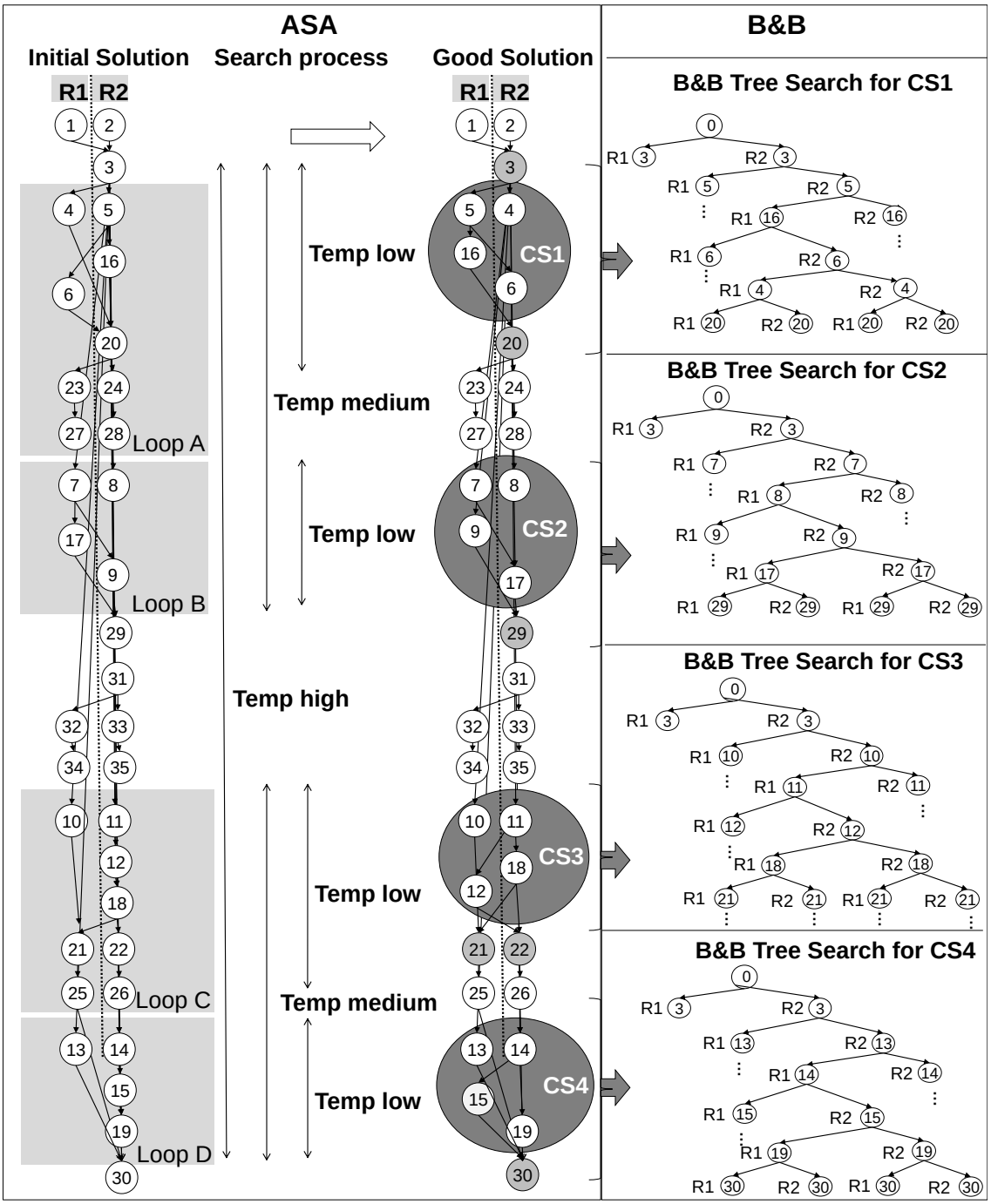


Figure 8.6: The ASA process (left part) partially freezes the schedule during temperature reduction. The range of the applied random moves is defined by the temperature $Temp$. When a level in the structural hierarchy has been explored, the allowed range is lower than the cost impact of reordering of the hierarchical graphs at that level. Then, the ASA proceeds at a finer granularity inside those graphs. It identifies four Critical Subgraphs (CSs), which are then propagated to the $B\&B$ to find the optimal schedule for each CS.

hierarchical levels. When the range comes close to that critical zone, i.e. the search proceeds to a critical subgraphs, the process has to be adapted in order to slow down to a very detailed search, where the equilibrium criterion of the ASA technique is maintained at every iteration. When the equilibrium has been reached, we have frozen the schedule decisions for that subgraph ordering. Then, we can again proceed faster for the schedule inside the subgraph. The random moves with distance larger than this critical range are now prohibited, thus a partial schedule in the higher hierarchy levels is decided. E.g. in the example of Fig. 8.6, the schedule of Loop A group of operations and Loop B group of operations cannot be exchanged with the Loop C group of operations and Loop D group of operations. When the temperature is further decreased, the next schedule decision is the partial schedule inside the group of Loop A and Loop B and the group of Loop C and Loop D. The reduction process continues until the random range is reduced to relative small critical subgraphs in the same hierarchy level ($Temp$ low). When the sensitivity of taking erroneous schedule decisions and the complexity for the remaining temperature range become too high, the ASA approach requires too much time in the detailed search and thus it is terminated. The regions that are still explored in this stage by ASA can then be considered as the only ones that need to be further explored by a more suited method. In this way, the critical subgraphs are localized independently of their type, as they are the ``non-frozen" parts of the CDFG. The ``frozen" are described by the (near-) optimal schedule of ASA of the search space, where the process is smooth and the random moves can still place the operations in optimal positions, i.e. the CDFG nodes between the critical subgraphs.

The critical subgraphs (dark gray circles in Fig. 8.6) and the schedule of the interface CDFG nodes (gray nodes in Fig. 8.6) found by ASA are propagated to the $B\&B$, as illustrated at the right part of Fig. 8.6. The latter applies a pseudoexhaustive search for each critical subgraph to optimally schedule them. The overall solution consists of most of the schedule obtained quite fast by the ASA, besides the critical subgraphs, and the optimal schedule of the critical subgraphs obtained by the $B\&B$. This reduces the complexity of the developed technique, since a large exploration area is pruned and the very detailed search to find the optimal is performed in a much smaller search area. Hence, the developed technique is still scalable. When the operations are increased, the ASA can still identify the position of all critical subgraphs. As only the critical subgraphs and the interfaces are given to the $B\&B$ and the critical subgraphs are uncoupled, the $B\&B$ can be concurrently applied per subgraph.

8.5.1.6 Results for MiBench benchmark suite

The output of the proposed methodology is the combined paramaterized template, which parametrically describes a group of similar in nature scheduling techniques. The narrow instantiation of the combined parametric template in one specific scheduling technique is inappropriate to be used in the evaluation process, because when the parameters have specific values, they affect the final result. Hence, we have to estimate a lower bound on the gains provided by the techniques described in the combined parameterized template. In this way, we evaluate the combined parame-

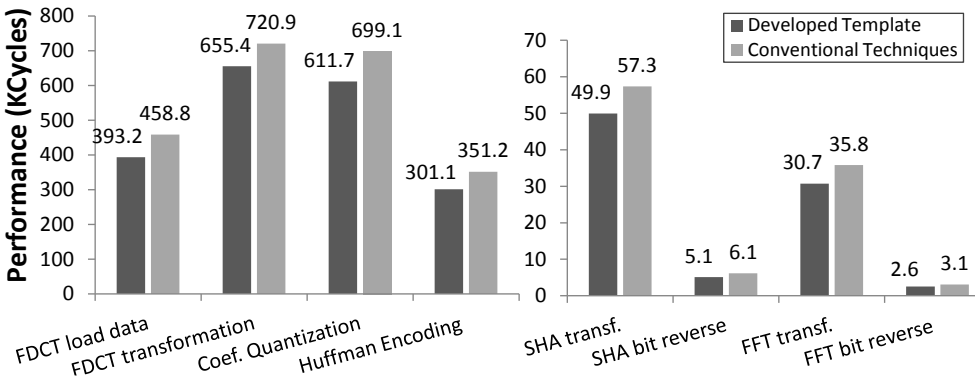


Figure 8.7: Schedule lengths for the demonstration case study domain.

terized template and not only one specific technique. We used several applications of the MiBench benchmark suite [63]. The CDFG of each application has been analyzed to identify the dominant basic blocks and their dominant loops, which are responsible for the large number of execution cycles. For each basic block, the structure of the internal operations, the dependencies between operations and the critical number of available resources in the platform are considered. Based on this information, the worst case critical subgraphs, placed at the end of a basic block, are identified. In these hardest-to-be-optimally scheduled subgraphs, the schedule result of the conventional approaches takes at least one cycle more in one iteration execution of the critical subgraph compared with the optimal schedule. In this way, a pessimistic lower bound of one cycle performance gain is achieved for the scheduling techniques which are capable of finding the optimal schedule, e.g. the techniques of the developed template. In several cases, depending on the type of the resources and the cycles of the operations, the gain can be increased, e.g. two cycles in the example of Section 8.2. The approximated length of the schedule of the proposed methodology is derived by multiplying the length of the optimal schedule for the critical subgraph by the number of loop iterations of the basic block, which contains the critical subgraph. The final length is computed by adding the schedule lengths of all the critical subgraphs occurring in the application, which is substantial as several blocks in loops with one critical subgraph often simultaneously exist. For the conventional techniques we apply the same approximation considering a schedule which has a length equal to the optimal length increased by one cycle due to the pessimistic lower bound on the gain.

The schedule length derived from the conventional approaches and the developed template is depicted in Fig. 8.7 for several dominant basic block of the MiBench benchmark suite. The analyzed dominant blocks are usually parts of other applications, e.g. FDCT jpeg encoding exist in jpeg decoding benchmark, the FFT transformation is applied in signal processing algorithms and SHA transformation in cryptography algorithms e.t.c. For the basic blocks which load the data in the Fast Discrete Cosine Transform (FDCT) application, a gain of 14.3% is expected when the schedule derives from the developed template. When the developed template is used to schedule the FDCT processing a gain of 18.2% is achieved. For the quantization of the coefficients and the Huffman encoding, we have a gain of 12.5% in the schedule length and a gain of 14.3%,

respectively. The schedule of the transformation of the Secure Hash Algorithm (SHA) and the schedule of SHA reversing of bits function based on the developed template has a gain of 12.9% and 16.7%, respectively. Finally, the schedule length of the developed technique for the transformation of the Fast Fourier Transformation (FFT) and the FFT reversing of bits function has a gain of 14.3% and 16.7%, respectively.

8.5.2 High number of critical subgraphs domain

We present the results of applying the proposed methodology to a second domain, i.e. the initial domain, but with high number of critical subgraphs. Step 1, i.e. initialization step, is omitted as it is the same with the demonstration case study.

8.5.2.1 Step 2: Application & Platform Domain Analysis

The domain constraints of the demonstration case study are modified: the number of critical subgraphs is highly increased and thus the optimality of the schedule of the critical subgraphs is not feasible within the available CPU time. However, the loss on the critical subgraphs schedule optimality should be limited and controllable. The remaining domain constraints are: large applications, relative small and uncoupled subgraphs, linear complexity of the scheduling technique in terms of problem size in order to find the (near-)optimal schedule within a reasonable search time. In the remaining steps, we provide the output per step and describe the differences with the result of the demonstration case study steps.

8.5.2.2 Step 3: Propagation of Domain Constraints

By propagating the domain constraint of large CDFG, the control of the optimality of the schedule and the linear complexity of the scheduling technique in terms of problem size requires an adaptive stochastic class. The propagation of the domain constraints matches also with the deterministic classes. The propagation of the control over the optimality constraint to the tree T, the near-optimality control of the pruning of space during the decisions (Pruning Decision Space/Control Optimality) is selected and the exact class is pruned. The recursive class is now compatible with near-optimality domain constraint, which provides the way the selected deterministic classes are applied in each iteration and the interface between consecutive iterations. The result is depicted in Fig. 8.8. After partial tree flattening step, the characteristics of the compatible classes are merged to one class, as depicted in Fig. 8.9.

8.5.2.3 Step 4: Propagation of Design Constraints

The fully flattened tree is depicted in Fig. 8.10. After the flattening, the combined parametric template includes the deterministic recursive class and the control near-optimality pruning decision space class, as depicted in Fig. 8.11. The stochastic template is explained in the demonstration case study. The recursive template consists of the type of the recursion, which can be set to adaptive

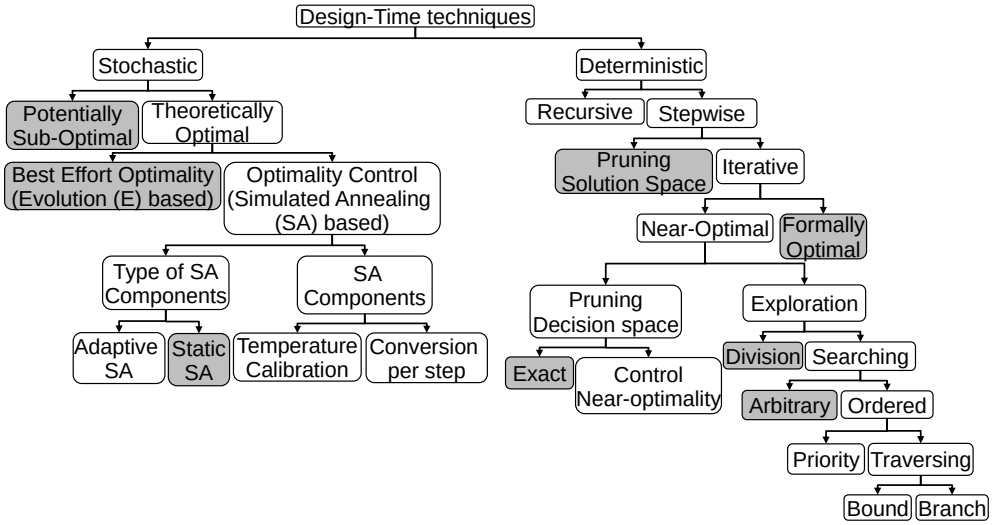


Figure 8.8: Domain Constraint Propagation step result for high number of critical subgraphs.

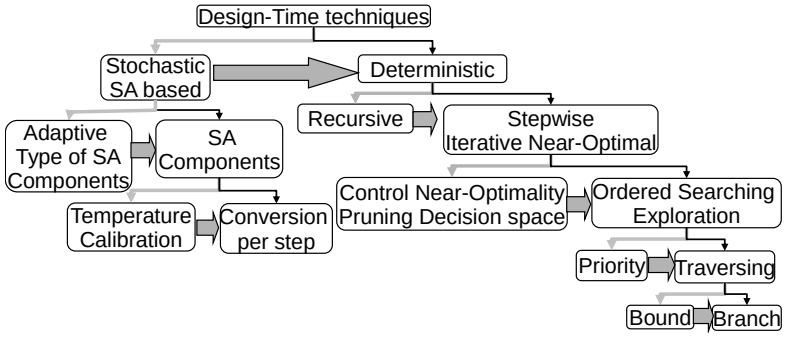


Figure 8.9: Partial flattening step result for high number of critical subgraphs.

or rigid, the calibration of the parameters of the process and the recursion termination criteria. The pruning during the search decisions is set to controlled near-optimal. This design constraint propagation partially defines the B&B template in a different way as in the demonstration case study. The bounds of the B&B class are estimated in a less exact, but tight and still controllable way, and, thus, the branching of the nodes can apply a more aggressive pruning, e.g. based on the average case, to process in a faster way.

8.5.2.4 Combined Parameterized Template & Results for MiBench benchmark suite

The ASA part identifies the critical subgraphs and (near-)optimally schedules the non-critical nodes. The critical subgraph, the initial subgraph schedule and the schedule of the interface nodes are propagated to the deterministic classes. The subgraph size and the number of iterations is used to select the most dominant critical subgraphs. The subgraph size gives an estimate of the CPU time required for the B&B part and the number of iterations describes the impact on the overall schedule. Hence, the graphs with high number of iterations are selected as dominant. The B&B part applies exact minimum cost of the bounds of the nodes to achieve optimal schedule for the dominant subgraphs. For the less dominant subgraphs, it obtains in a fast way a controlled near-optimal schedule for each subgraph using less exact and tight bounds in a controlled way. The

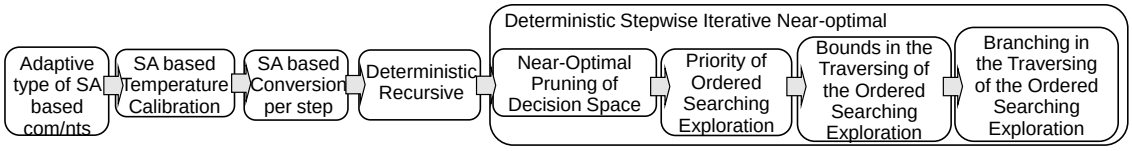


Figure 8.10: Full flattening step result for high number of critical subgraphs.

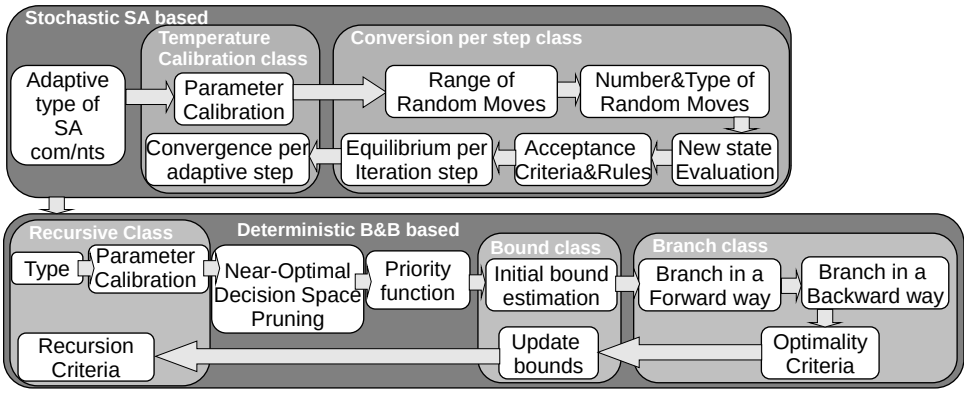


Figure 8.11: Combined Parameterized template for high number of critical subgraphs.

bounds estimation, which gives the control, is performed based on the available time. If enough time is available, closer to exact bounds are used to have a loss of one cycle. If less time is available, a more aggressive pruning is performed, increasing the cycle loss. The information to make the selection is provided by a set of pareto points, which depict the lower bounds on schedule quality and upper bounds on time execution for different bound estimations. Depending on the CPU time available, the B&B for each critical subgraph can be repeated using as an input the solution obtained by the previous iteration. The recursions of the search are selected based on the actual requirements in quality and available time. In this way, the schedule optimality loss of the developed template is inserted only due to the more aggressive B&B in the less dominant critical subgraphs. The estimation on the performance of the developed template derives analyzing the CDFG to identify the basic blocks, the loops and the worst case critical subgraphs, placed at the end of a basic block. In these hardest-to-be-optimally scheduled subgraphs, the dominant subgraphs are optimally scheduled and the less dominant are near-optimally scheduled with one cycle performance loss. The approximated length derives by multiplying the length of the optimal schedule for the dominant subgraphs and the length of the optimal plus one by the corresponding number of surrounding loop iterations, which can vary depending on the depth of the subgraph in the overall loop nest, and adding the results. For the conventional techniques we apply the same approximation to derive a lower bound on the optimality loss. For all critical subgraphs, a schedule which has a length equal to the optimal length increased by one cycle is multiplied by the corresponding number of surrounding loop iterations. Depending on the type of the resources and the cycles of the operations, the cycle loss may be increased, e.g. two cycles in the example of Section II. The estimated gains are depicted in Fig. 8.12.

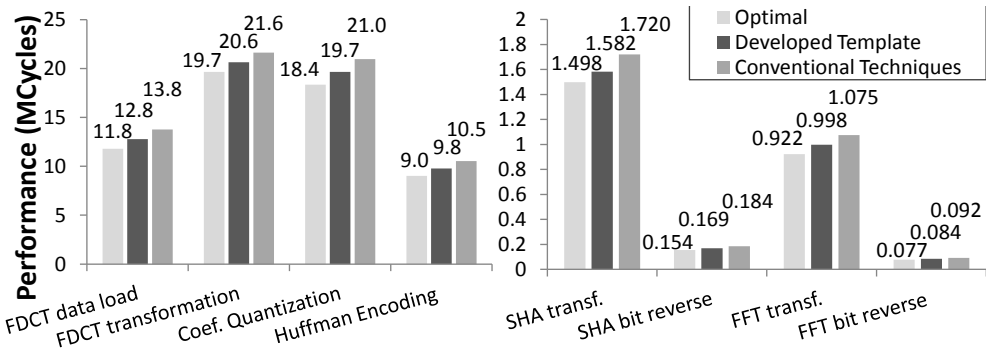


Figure 8.12: Schedule lengths for high number of critical subgraphs domain.

8.5.3 Large and strongly connected subgraphs

The second case study is the initial domain, but with large and strongly connected subgraphs. Step 1, i.e. initialization step, is omitted as it is the same with the demonstration case study.

8.5.3.1 Step 2: Application & Platform Domain Analysis

The domain constraints of the demonstration case study are modified: the CDFG is composed by several basic blocks which have subgraphs with strong dependencies between them. Hence, the CDFG subgraphs are strongly connected and coupled components with increased size. A potential separation of the strongly connected subgraphs will lead to sub-optimal results. The remaining domain constraints are: large applications, optimal schedule for the critical subgraphs, linear complexity of the scheduling technique in terms of problem size in order to find the (near-)optimal schedule within a reasonable search time. In the remaining steps, we provide the output and describe the differences with the result of the demonstration case study.

8.5.3.2 Step 3: Propagation of Domain Constraints

By propagating the domain constraints of large CDFG, of optimal schedule of the critical subgraphs and of the linear complexity of the scheduling technique in terms of problem size, they require an adaptive stochastic class to find a (near-) optimal solution for the operations outside the critical subgraphs and to identify the critical subgraphs position. The propagation of the domain constraints matches also with the deterministic classes, which control the schedule optimality (exact). Again, because of the domain constraint for reduced search time, the formally optimal class is pruned, since it is incompatible with the linear complexity in terms of problem size constraint. By propagating the constraint of coupled subgraphs (instead of weak dependencies in the demonstration case study) the pruning of the overall solution space in the initial search steps (Pruning Solution Space) class is now compatible with the domain constraint. The result is depicted in Fig. 8.13. After partial tree flattening step, the characteristics of the compatible classes are merged, as depicted in Fig. 8.14.

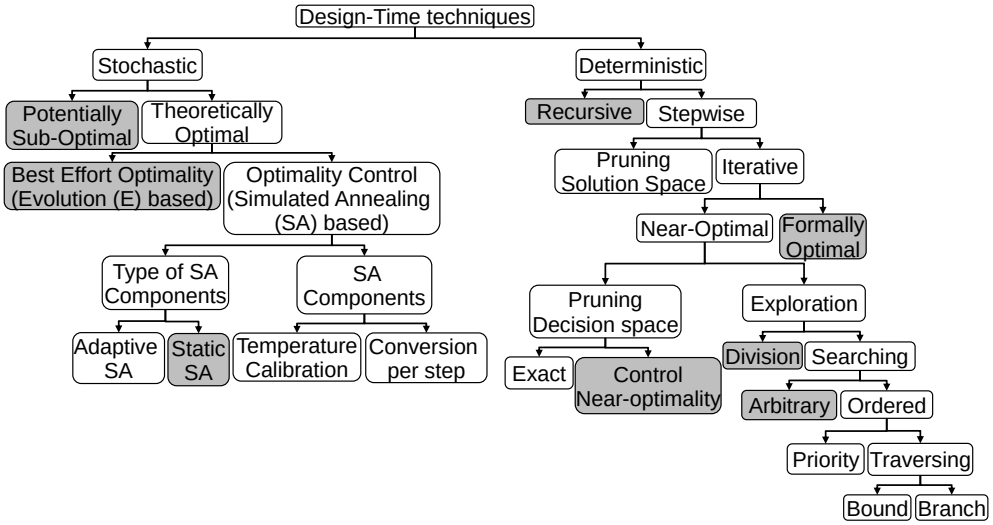


Figure 8.13: Domain Constraint Propagation step result for large, strongly connected subgraphs.

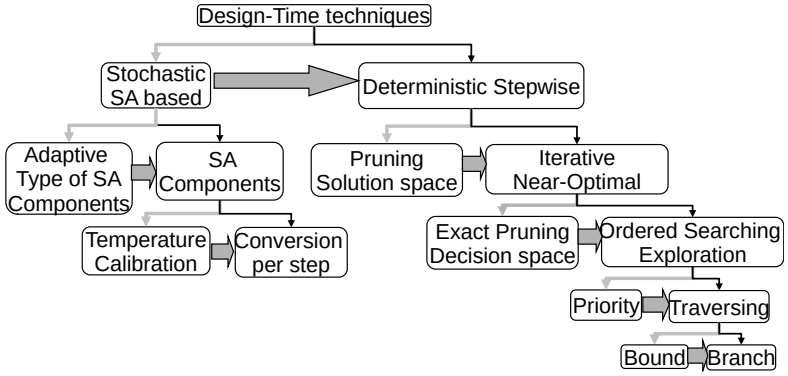


Figure 8.14: Partial flattening step result for large, strongly connected subgraphs.

8.5.3.3 Step 4: Propagation of Design Constraints

Following the in-depth search with priority to the left branch, the partial flattened tree is fully flattened as depicted in Fig. 8.15. The partially defined parametric templates of the selected stochastic classes and deterministic classes are depicted by the stochastic and the deterministic box respectively in Fig. 8.16. The first is the stochastic template, which is explained in the demonstration case study. The pruning of solution space class template is described by a high level function to estimate the quality of the solution and the function for the pruning criteria. The propagation of the optimality constraint to the pruning of the solution space imposes to the high level function to optimistically estimate the quality. Then, the pruning criteria are defined to prune the nodes that have an optimal high level estimation worst than the lower quality threshold. The pruning during the search decisions is also set to exact. The B&B template is described in demonstration case study.

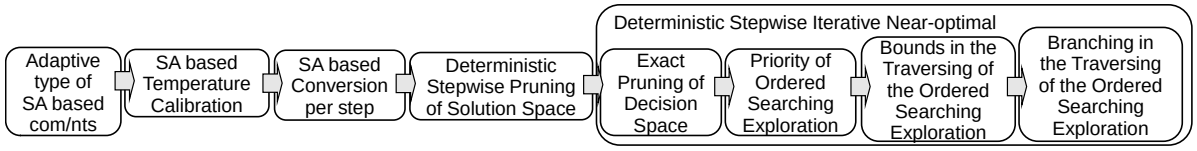


Figure 8.15: Full flattening step result for large, strongly connected subgraphs.

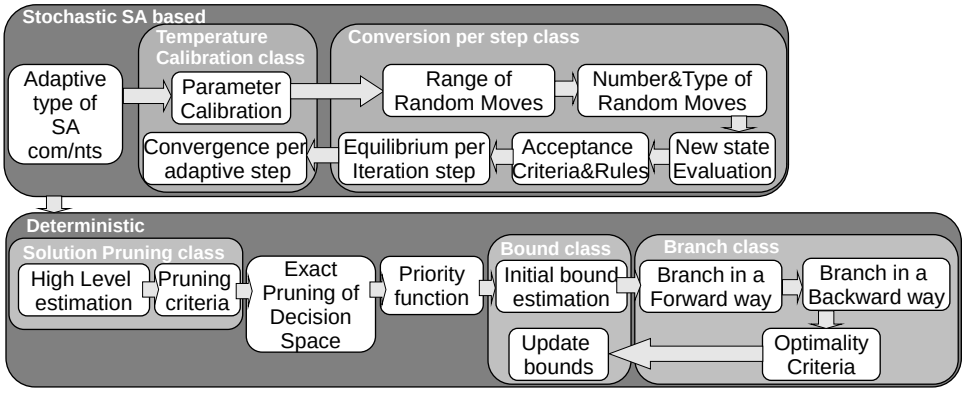


Figure 8.16: Combined Parameterized template for large, strongly connected subgraphs.

8.5.3.4 Combined Parameterized Template & Results for MiBench benchmark suite

The critical subgraphs, their initial schedule and the schedule of the interface CDFG nodes identified by the ASA are propagated to the deterministic classes, where a further high level pruning can be achieved before the B&B is applied. The pruning of solution space explores the hierarchy of the application. It schedules from the end of the large critical graph. During the scheduling, the dependencies, which couple the different parts of the subgraphs, are removed. When these dependencies are removed, the remaining parts are decoupled and can be independently scheduled. The process can be repeated in a nested way and when the subgraphs reach acceptable size, they are propagated to the B&B part. For instance, in the case of a large FFT kernel, the last stage is responsible for the coupling of the subgraphs. By scheduling the last stage in the pruning of solutions space, a relative schedule between the inner FFT stages is computed. Depending on the size of the FFT, the process can be repeated for further decoupling. Then, the remaining early stages can be passed to the B&B part. The schedule length of the developed template is estimated by optimal schedule in the inner critical subgraphs and one cycle loss in the schedule of the intermediate nodes, which couple the subgraphs. The schedule lengths are multiplied by the number of the loop iterations, which include each part, and the summation of the results gives the estimation of the developed template performance. If the large subgraphs were propagated directly to the B&B part, quite significant time may be required to reach the optimal solution. Now the decoupling in the up-front pruning template reduces the required exploration time for the B&B part, leading to overall near-optimal results. The conventional techniques will have at least one cycle loss in the critical subgraphs at the end of the basic blocks, as explained in Section VI-F-1. Hence, the estimated lower bound loss derives by considering the optimal length increased by one for the critical subgraphs and the optimal length for the remaining nodes. The schedule lengths are multiplied by

the corresponding number of surrounding loop iterations and the results are added. The schedule length are depicted in Table II of Section VI-F. The schedule lengths are in Table 8.2.

Table 8.2: Schedule length for Large, strongly connected subgraphs

Algorithm	Optimal (MCycles)	Developed (MCycles)	Conventional (MCycles)
FDCT Transformation	83.902	83.911	92.291
FFT Transformation	29.491	30.081	34.210

8.6 Conclusions

In this chapter, we show how the principles and the process of using the framework with the partitions of all the cases of Chapter 2 is projected to the goal of developing parametric template of design-time scheduling techniques, which near-optimally and in a scalable way schedule the target domain. The systematic methodology, which is derived by the projection, develops a parametric template of design-time scheduling techniques which satisfy the target domain constraints. The target domain constraints are propagated to the framework of the complete set of scheduling classes. Each class is described by prestored parametric template. The vertical constraint propagation principle selects the compatible classes and the combination principle combines the partial parametric templates to the final parametric template of the developed techniques by propagating design constraints.

Chapter 9

Conclusions & Future Directions

9.1 Conclusions

We have proposed a reusable methodology which achieves near-optimal and scalable DSE frameworks. We have applied the proposed DSE into both the storing part of embedded systems and the processing part of the embedded systems.

In the storing part, we have focused on the intra-signal in-place optimization steps of the background memory. In Chapter 3 we have applied the reusable methodology to develop the uni-directionally connected steps of a scalable and near-optimal intra-signal in-place methodology [96], [98] [97]. Per step we have defined a framework with the different possible cases. In Chapter 4 we have presented a scalable and near-optimal representation for irregular access schemes to support the translation step of the developed intra-signal in-place optimization methodology [98]. In Chapter 5 we have proposed the parametric templates which provide the scalable and near-optimal solution processes for each valid option for the non-overlapping [96] and the overlapping case [97] of intra-signal in-place cases for irregular access schemes. We have evaluated the results of our methodologies and we have shown that we achieve near-optimal results with low exploration time, which remains stable when the number of accesses is increased due to increase in the loop bound. The exploration time of existing approaches is highly increased with the increase in the accesses. In addition, part of the PhD contributions on the storing part of the embedded systems has been performed in cooperation with PhD candidate V. Kelefouras. The contributions are not described in the current PhD dissertation, as they are described by the PhD dissertation of the first author. The contributions have focused on methodologies to improve the data locality with performance objectives with focus on global optimization transformations and the exploration of the data reuse in the memory hierarchy of the on-chip background memory [86], [4].

In the processing part, we have applied the reusable methodology to describe the steps of a DSE methodology for an FPGA architecture platform with one microprocessor and several HW accelerators [95], [94]. We have developed the partial pareto curves per step and by following the uni-directional constraint propagation of the steps, we have developed the final pareto curve, as described in Chapter 6. The latter can be used to select the final design based on the specified

requirements of the system each time. The foreground memory management and the datapath mapping steps of the developed DSE methodology have as main task the scheduling and the assignment process. In order to provide a scalable and near-optimal scheduling and assignment process for the domain under study, we have applied the reusable methodology principles to create a framework with the complete set of scheduling and assignment processes in Chapter 7 [99]. Then, in Chapter 8, we have developed a methodology by applying the principles of the reusable DSE methodology to create a scalable and near-optimal parametric template for the target domain under study [100]. The developed parametric templates for large and complex CDFGs achieve near-optimal designs in acceptable CPU time. The developed parametric template can be used to select the solutions for the scheduling and the assignment tasks of the foreground and data-path mapping step of the developed DSE methodology.In addition, part of the PhD contributions on the processing part of the embedded systems has been performed in cooperation with PhD candidate D. Tsitsipis. The contributions have as context the Wireless Sensor Networks and focus on the proposal of clustering scheduling and assignment techniques to improve the energy and the packet loss during transmissions [192], [193], [191]. Another part of the PhD contributions on the processing part of the embedded systems has been performed in cooperation with Dr. H. Michail and focus on the co-design of a SHA256 cryptographic module [129].

In conclusions, applying the principles of the methodology to create and use scalable and near-optimal DSE frameworks in the storing and processing context of the dissertation has lead to near-optimal and scalable solutions. Hence, we believe that by applying the proposed DSE principles to the remaining steps of the design flow will lead to similar results.

9.2 Future directions

In the current PhD dissertation, we have applied the reusable DSE methodology in the context of the processor level data transfer and storage exploration to propose methodologies for the intra-signal in-place optimization step. As a future direction, thorough application of the proposed methodologies through several different realistic case studies can be performed. In addition, more application for several case studies can be also applied in the DSE framework for the processing part and the methodology to develop near-optimal and scalable parametric templates for a target domain.

In the current PhD dissertation we have applied the DSE methodology to create the DSE frameworks for several steps and options in the problems under study. However, we have made several assumptions in order not to open all the possible cases of the DSE framework. For instance, in the intra-signal in-place methodologies we have focused on the affine expressions and provided hints on how to deal with non-affine cases. Another example is the regularity of the application parts in the DSE mapping methodology of the processing part. Hence, as a future direction, the different cases that have not been further refined can be explored by applying the DSE methodology and provide the corresponding parametric templates with the solutions.

For the storing part of the processor level system design, a similar methodology can be developed for the inter-signal in-place optimization step, i.e. the minimum number of resources required to store the elements of all the arrays of the application. The cases for the inter-signal in-place methodology should be defined. The analysis step and the pattern representation of the translation step of the intra-signal in-place optimization step can be partially reused for the inter-signal in-place methodology. Then, parametric templates will be developed to solve in a scalable and near-optimal way the different inter-signal cases. The reusable methodology can be also applied for other mapping steps in the storing part, i.e. memory hierarchy level assignment optimization step, memory allocation and assignment, storage order optimization etc [21]. By propagating the characteristics of the Processor Level heterogeneous ordering and assignment of irregular data to the Design-Time Global Scheduling Classification of Chapter 7, which describes the scheduling and assignment of heterogeneous resources, we can define the techniques which are promising for these steps of the background memory mapping methodology.

For the processing part of the processor level, DSE methodologies can be developed for other partially fixed platform architectures, e.g. FPGA with multiple microprocessors and multiple HW accelerators or other platforms with different but still relevant HW characteristics. The DSE methodologies can be developed by applying and partially re-project the principles of the reusable DSE methodology of Chapter 6. The different valid cases for each step can be defined by creating the corresponding DSE framework following the reusable DSE methodology described in Chapter 2. The methodology to develop near-optimal and scalable scheduling parametric templates, as described in Chapter 8 and illustrated for large and complex CDFGs, is applicable for several domains with different characteristics and constraints. In addition, another future work is to apply the reusable DSE methodology of Chapter 2 in the developed parametric template having as goal to propose a methodology which will fully instantiate the parametric template to a specific scheduling and assignment technique by giving specific values to the parameters of the template, based on the characteristics of the application instance.

Finally, other future directions is the application of the reusable methodology for scalable and near-optimal in the context of the remaining steps of the unified design flow described in Chapter 2. Attempts and early experiments on this direction have been performed in [132] for the task level concurrency management of the thread frame abstraction layer, whereas the data level has not been explored yet.

Appendix A

Publication List

A.1 Journals

1. A.Kritikakou, F. Catthoor, V. Kelefouras and C. Goutis, “A Systematic Approach to Classify Design Time Global Scheduling Techniques”, Journal of ACM Computed Surveys, Vol. 45, No. 2, Feb, 2013, pp. 14:1 -- 14:30, DOI: 10.1145/2431211.2431213

2. A.Kritikakou, F. Catthoor, G.S. Athanasiou, V. Kelefouras and C. Goutis, “Near-optimal Microprocessor & Accelerators Co-Design with Latency & Throughput Constraints”, ACM Trans. Architecture and Code Optimization, Vol.10, No.2, May, 2013, pp. 6:1--6:25, DOI: 10.1145/2459316.2459317

3. A.Kritikakou, F. Catthoor, V. Kelefouras and C. Goutis, “A Scalable & Near-optimal Representation for Storage Size Management”, ACM Trans. Architecture and Code Optimization, conditionally accepted, 2013.

4. A.Kritikakou, F. Catthoor, V. Kelefouras and C. Goutis, “Near-optimal & Scalable Intra-signal In-place for Non-overlapping & Irregular Access Scheme”, ACM Trans. Design Automation of Electronic Systems (TODAES), conditionally accepted, 2013.

5. V.I. Kelefouras, G.S. Athanasiou, N. Alachiotis, H. E. Michail, A. S. Kritikakou and C.E. Goutis, “A memory efficient Fast Fourier Transform (FFT) methodology”, IEEE Transactions on Signal Processing, 2011, Vol.59, No. 12, pp. 6217-6226, DOI: 10.1109/TSP.2011.2168525.

6. N. Alachiotis, V. Kelefouras, G. Athanasiou, H. Michail, A. Kritikakou and C. Goutis, “A Data Locality Methodology for Matrix-Matrix Multiplication Algorithm”, The journal of Supercomputing, Springer, 2012, Vol. 59, No. 2, pp. 830—851, DOI: 10.1007/s11227-010-0474-3.

7. A.Kritikakou, F. Catthoor, V. Kelefouras and C. Goutis, “A Systematic Methodology to Develop New Global Design Time Scheduling Techniques”, Submitted in ACM Transactions on Design Automation of Electronic Systems (TODAES), 2013.

8. A.Kritikakou, F. Catthoor, V. Kelefouras and C. Goutis, “Scalable & Near-Optimal Array Storage Size under Overlapping & Irregular Accesses”, Submitted in IEEE Trans. Computers (TC), 2013.

9. A.Kritikakou, F. Catthoor, V. Kelefouras and C. Goutis, “Scalable & Near-Optimal Fore-Ground Memory Allocation”, to be submitted, 2013.

A.2 Conferences

1. A. Kritikakou, F.Catthoor, G.S. Athanasiou, V. Kelefouras and C. Goutis, “A Template-based Methodology for Efficient Microprocessor and FPGA Accelerator Co-Design”, Proc. Int’l Conf. Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 16-19 July 2012, Samos, Greece.

2. H. Michail, A. Gregoriades, V. Papadopoulou, G. Athanasiou, A. Kritikakou and C. Goutis, “High Throughput Hardware/Software Codesign Methodology For Designing SHA-256 Hashing Cryptographic Module in Hardware”, in Proc. Int'l Conf. Security and Cryptography (SECRYPT), 26-28 July 2010, Athens, Greece.

3. D. Tsitsipis, S.M. Dima, A.Kritikakou, C. Panagiotou, and S.Koubias, “Data Merge: A data aggregation technique for Wireless Sensor Networks”, Proc. Int’l Conf. Emerging Technologies and Factory Automation (ETFA), 5-9 Sep 2011, Toulouse, France.

4. D. Tsitsipis, S.M. Dima, A.Kritikakou, C. Panagiotou, and S.Koubias, “Segmentation and Reassembly Data Merge (SaRDaM) Technique for Wireless Sensor Networks”, Proc. Int’l Conf. Industrial Technology (ICIT), 19-21 Mar 2012, Athens, Greece.

5. D. Tsitsipis, S.M. Dima, A.Kritikakou, C. Panagiotou, J. Gialelis, H. Michail and S.Koubias, “Prioritized Segmentation and Reassembly Data Merge (PSaRDaM) Technique for Wireless Sensor Networks”, Proc. Int’l Conf. Emerging Technologies and Factory Automation (ETFA),17-21 Sep 2012, Cracow, Poland.

Appendix B

Εκτεταμένη Περίληψη στα Ελληνικά

B.1 Εισαγωγή

Τα ενσωματωμένα συστήματα είναι υπολογιστικά συστήματα τα οποία εκτελούν εφαρμογές αφιερωμένες σε ένα συγκεκριμένο σκοπό και δεν είναι υπολογιστές γενικού σκοπού. Τα ενσωματωμένα συστήματα αποτελούνται από μια συλλογή προγραμματιστικών μερών και στοιχείων, η οποία επικοινωνεί με το περιβάλλον. Παραδείγματα ενσωματωμένων συστημάτων αποτελούν οι συσκευές κινητών τηλεφώνων, συσκευές βιοϊατρικής, συσκευές που χρησιμοποιούνται στην ασφάλεια επικοινωνίας και σε πολυμέσα.

Οι εφαρμογές ενσωματωμένων συστημάτων προέρχονται από τομείς όπως τα πολυμέσα, τα γραφικά, τις ασύρματες συνδέσεις, την βιοϊατρική, την επεξεργασία σημάτων. Οι εφαρμογές ενσωματωμένων συστημάτων, εκτός από την πολυπλοκότητα που έχουν, εισάγουν αυστηρούς περιορισμούς (constraints) και ελαστικούς περιορισμούς (trade-offs) που μετρούν την απόδοση του συστήματος. Επιπρόσθετοι περιορισμοί τίθενται από το περιβάλλον, π.χ. την αγορά, την εταιρία σχεδίασης και παραγωγής των ενσωματωμένων συστημάτων και τους χρήστες. Τα constraints περιγράφουν την ελάχιστη ή μέγιστη (ανάλογα με την περίπτωση) αποδεκτή τιμή που μπορεί να έχει ένας παράγοντας σχεδιασμού του συστήματος, ενώ τα trade-offs περιγράφουν ένα εύρος τιμών για τις σχεδιαστικές μετρικές που είναι πάνω από την ελάχιστη αποδεκτή τιμή.

Οι περισσότερες εφαρμογές για ενσωματωμένα συστήματα έχουν constraints στην απόδοση που εκφράζονται μέσω του χρόνου εκτέλεσης της εφαρμογής και του ρυθμού παραγωγής των αποτελεσμάτων της εφαρμογής. Π.χ. όταν η εφαρμογή πρέπει να έχει εκτελεστεί πριν από μια συγκεκριμένη χρονική στιγμή δημιουργεί ένα χρονικό constraint στην απόδοση του ενσωματωμένου συστήματος. Τα περισσότερα ενσωματωμένα συστήματα λειτουργούν με μπαταριές και επομένως η κατανάλωση ενέργειας είναι πολύ κρίσιμη για την διάρκεια που θα λειτουργεί το σύστημα χωρίς επαναφόρτιση. Η ελάχιστη

απαιτούμενη διάρκεια ζωής του συστήματος ανάμεσα στις επαναφορτίσεις καθορίζει το constraint στην κατανάλωση ενέργειας. Επιπρόσθετη μείωση στον χρόνο εκτέλεσης και στην κατανάλωση ενέργειας της εφαρμογής από την ελάχιστη τιμή που ορίζουν τα constraints μπορεί να αποφασιστεί με την εξερεύνηση του πολυδιάστατου trade-offs χώρου λύσεων.

Για να ικανοποιηθούν τα constraints του συστήματος, τα ενσωματωμένα συστήματα αποτελούνται από ποικίλους ετερογενείς πολυεπεξεργαστές, όπως Reduced Instruction Set Computer (RISC), Very Long Instruction Word (VLIW) ή Single Instruction Multiple Data (SIMD),Application Specific Instruction Set Processors (ASIP), Digital Signal Processor (DSP) και Application-Specific Integrated Circuit (ASIC), και χρησιμοποιούν διάφορους τρόπους λειτουργίας των πολυεπεξεργαστών, όπως Dynamic Voltage and Frequency Scaling (DVFS) και Dynamic Power Management (DPM), ενώ περιέχουν επαναπρογραμματιζόμενες μονάδες επεξεργασίας, πολύπλοκες δομές και ιεραρχίες μνημών, πολύπλοκες διεπαφές. Λόγω των απαιτήσεων του συστήματος, οι σχεδιασμοί βασισμένοι μόνο στο λογισμικό (software) δεν μπορούν να πραγματοποιηθούν ανεξάρτητα από το υλικό (hardware). Επομένως, υλικό και λογισμικό πρέπει να αναπτυχθούν παράλληλα κατά την σχεδίαση του συστήματος για να ικανοποιήσουν τις απαιτήσεις του συστήματος [122].

Ο πολυδιάστατος trade-offs χώρος λύσεων δημιουργείται από ένα σύνολό αξόνων που περιγράφουν τις σχεδιαστικές μετρικές και οι οποίες χαρακτηρίζουν την συνολική απόδοση του συστήματος. Οι σχεδιαστικές μετρικές με υψηλό συντελεστή βαρύτητας είναι η απόδοση, η επιφάνεια ολοκλήρωσης και η κατανάλωση ενέργειας του συστήματος. Τα ενσωματωμένα συστήματα είναι συνήθως συστήματα πραγματικού χρόνου και επομένως ο χρόνος εκτέλεσης της εφαρμογής είναι ένας βασικός άξονας στον πολυδιάστατο trade-offs χώρο. Επιπρόσθετα, τα ενσωματωμένα συστήματα είναι φορητά και επομένως η κατανάλωση ενέργειας παίζει πολύ σημαντικό ρόλο. Η leackage κατανάλωση ενέργειας είναι έμμεσα συσχετιζόμενη με την επιφάνεια ολοκλήρωσης, δηλαδή την περιοχή που καταλαμβάνει το ενσωματωμένο σύστημα. Καθώς αυξάνει ο αριθμός των πυλών που χρειάζονται για την ολοκλήρωση του συστήματος, αυξάνει και η κατανάλωση ενέργειας λόγω αύξησης του ρεύματος διαφυγής. Οι σχεδιαστικές μετρικές με υψηλό συντελεστή βαρύτητας δεν μπορούν να ικανοποιηθούν ταυτόχρονα διότι η μία μετρική συνήθως αντικρούει άλλες μετρικές. Για να μειωθεί περαιτέρω ο χρόνος εκτέλεσης της εφαρμογής, υπάρχουν διάφορες επιλογές στην σχεδίαση του συστήματος, όπως η απεικόνιση των κρίσιμων τμημάτων της εφαρμογής σε υλικό ειδικού σκοπού, ο πιθανός παραλληλισμός τμημάτων της αρχιτεκτονικής πλατφόρμας, η επιλογή υψηλότερης συχνότητας στα τμήματα υλικού ή ακόμα πιο πυκνό χρονοπρογραμματισμό των εντολών και τον προσπελάσεων στη μνήμη. Οι προαναφερθείσες σχεδιαστικές επιλογές οδηγούν σε αύξηση της κατανάλωσης ενέργειας και της απαιτούμενης επιφάνειας ολοκλήρωσης. Οι επιπρόσθετες απαιτήσεις σε υλικό αυξάνουν τον αριθμό των πυλών που χρησιμοποιούνται για την σχεδίαση του συστήματος και επομένως την κατανάλωση ενέργειας. Ο πιο πυκνός χρονοπρογραμματισμός βελτιστοποιεί τις προσπελάσεις στην μνήμη, αλλά αυξάνει τον

απαιτούμενο χώρο αποθήκευσης των δεδομένων διότι αυξάνεται η διάρκεια ζωής των δεδομένων. Η αύξηση στον απαιτούμενο χώρο αποθήκευσης οδηγεί σε μεγαλύτερες μνήμες και σε αύξηση της κατανάλωσης ενέργειας ανά προσπέλαση.

Τα ενσωματωμένα συστήματα είναι προϊόντα μεγάλης παραγωγικότητας σε αγορές υψηλού ανταγωνισμού. Επομένως, το τελικό σύστημα, που είναι και το προϊόν προς πώληση, οφείλει να έχει χαμηλό κόστος μονάδας. Το κόστος μονάδας εκφράζει τόσο το κόστος κατασκευής όσο και το κόστος σχεδίασης. Το κόστος κατασκευής εκφράζεται είναι το κόστος κατασκευής ενός συστήματος. Όταν η κατανάλωση της ενέργειας και της ισχύος του συστήματος αυξάνεται, το κόστος μονάδας αυξάνεται λόγω των αυξημένων απαιτήσεων σε παροχή ισχύος αλλά και στο σύστημα ψύξης. Όταν αυξάνεται η αναγκαία επιφάνεια ολοκλήρωσης του συστήματος λόγω αύξησης του μεγέθους στο απαιτούμενο φυσικό χώρο, ο οποίος μετριέται σε Bytes για το λογισμικό και σε πύλες για το υλικό, αυξάνεται το κόστος μονάδας. Το κόστος σχεδίασης εκφράζεται μέσω του Non-Recurring Engineering (NRE) κόστους, το οποίο περιγράφει το κόστος για την σχεδίαση του συστήματος πριν την κατασκευή του και το κόστος για τροποποίηση του συστήματος. Επομένως, ένα επιπρόσθετος παράγοντας είναι η ευελιξία του συστήματος, που περιγράφει την δυνατότητα να μεταβληθεί η λειτουργικότητα του συστήματος χωρίς υψηλό NRE κόστος. Για να αυξηθεί η ευελιξία του συστήματος, απαιτείται επιπρόσθετη επιφάνεια ολοκλήρωσης όταν οι πραγματικού χρόνου περιορισμοί δεν ικανοποιούνται. Όταν το σύστημα παράγεται σε υψηλές ποσότητες, ο συντελεστής βαρύτητας για την ευελιξία του συστήματος είναι υψηλός και επομένως μια μεγαλύτερη επιφάνεια ολοκλήρωσης είναι πιθανόν να είναι αποδεκτή διότι παρέχει υψηλότερο κέρδος.

Η υψηλή ανταγωνιστικότητα στις αγορές απαιτεί μικρό χρόνο μέχρι το προϊόν να είναι έτοιμο στην αγορά, δηλαδή μικρό συνολικό χρόνο για τον σχεδιασμό και την κατασκευή του προϊόντος. Μια μικρή καθυστέρηση στο χρόνο προς την αγορά μπορεί να έχει καταστροφικές οικονομικές συνέπειες. Επομένως ο χρόνος προς την αγορά (time-to-market) έχει υψηλό συντελεστή βαρύτητας στον πολυδιάστατο trade-offs χώρο λύσεων. Ο χρόνος προς την αγορά θα πρέπει να παραμείνει εντός ενός χρονικού παραθύρου που προσδιορίζει τον διαθέσιμο χρόνο για την σχεδίαση του συστήματος και για την εξερεύνηση των πιθανών σχεδιάσεων. Όταν περισσότερος χρόνος είναι διαθέσιμος για την σχεδίαση του συστήματος, τότε μπορεί να επιτευχθεί καλύτερη εξερεύνηση του πολυδιάστατου χώρου λύσεων, η οποία οδηγεί σε σχεδόν βέλτιστους σχεδιασμούς με χαμηλότερο κόστος παραγωγής. Ωστόσο, σε αυτή την περίπτωση το κόστος NRE αυξάνεται. Όταν ο διαθέσιμος χρόνος για την σχεδίαση του προϊόντος ή το NRE κόστος είναι χαμηλότερα, πραγματοποιείται μια πιο επιθετική μείωση του χώρου λύσεων, η οποία οδηγεί σε όχι τόσο βέλτιστες, αλλά ακόμα αποδεκτές, σχεδιάσεις.

Λόγω της αύξησης της πολυπλοκότητας του υλικού και του λογισμικού των ενσωματωμένων συστημάτων, του αυστηρού χρόνου και των περιορισμών κατανάλωσης των εφαρμογών, το χαμηλό κόστος και τον λίγο διαθέσιμο χρόνο προς την αγορά και τον πολυδιάστατο χώρο λύσεων, οι σχεδιάσεις βασισμένες σε τυχαίες επιλογές και στην

προηγούμενη εμπειρία των σχεδιαστών δεν οδηγούν σε σχεδόν βέλτιστες σχεδιάσεις. Η διαδικασία για την σχεδίαση του συστήματος παίρνει πολύ χρόνο και δεν παρέχει εγγύηση ότι θα ικανοποιηθούν οι περιορισμοί του συστήματος και ότι θα σχεδιαστεί ένα σχεδόν βέλτιστο σύστημα λαμβάνοντας υπόψιν όλες τις σχεδιαστικές μετρικές. Επομένως, απαιτούνται μεθοδολογίες για την εξερεύνηση των σχεδιασμών (Design Space Exploration - DSE), οι οποίες εξερευνούν αποδοτικά τις επιλογές σχεδίασης στον πολυδιάστατο χώρο λύσεων του συστήματος και παρέχουν σχεδόν βέλτιστες σχεδιάσεις σε μικρό χρόνο προς την αγορά.

Β.2 Υπάρχουσες μεθοδολογίες και περιορισμοί

Οι DSE μεθοδολογίες εξερευνούν τις επιλογές της αρχιτεκτονικής, των στοιχείων, των διεπαφών και της απεικόνισης των δεδομένων, ώστε να επιτύχουν ένα σχεδόν βέλτιστο σύστημα το οποίο ικανοποιεί τους περιορισμούς και ελαχιστοποιεί το κόστος στον πολυδιάστατο trade-off χώρο λύσεων. Κατά την διάρκεια εξερεύνησης του χώρου λύσεων εξερευνούνται οι απαιτήσεις σε είσοδο και έξοδο των δεδομένων, σε αποθήκευση δεδομένων, σε επεξεργασία δεδομένων και στον έλεγχο του συστήματος των διαφόρων επιλογών.

Η έρευνα από την ακαδημαϊκή κοινότητα έχει περισσότερο επικεντρωθεί σε μεθοδολογίες εξερεύνησης σχεδιασμών οι οποίες παρέχουν σχεδόν βέλτιστες σχεδιάσεις. Η ιδανική μεθοδολογία εξερεύνησης σχεδιασμών αντιμετωπίζει όλα τα προβλήματα απεικόνισης του λογισμικό στο υλικό ταυτόχρονα σε μια μοναδική φάση. Ωστόσο η ιδανική μεθοδολογία εξερεύνησης δεν μπορεί να πραγματοποιηθεί. Η διαδικασία σχεδιασμού ενσωματωμένων συστημάτων είναι πολύ περίπλοκη και αποτελείται από ποικίλες φάσεις και δεν υπάρχει μοναδικός τρόπος να αναπαρασταθεί αποδοτικά το πρόβλημα. Επομένως, η μεθοδολογία εξερεύνησης του χώρου λύσεων πρέπει να διαιρεθεί σε ένα σύνολο βημάτων ούτως ώστε να είναι διαχειρίσιμη [122].

Οι υπάρχουσες μεθοδολογίες για εξερεύνηση των σχεδιασμών χωρίζουν την διαδικασία σε βήματα. Ωστόσο ο διαχωρισμός υλοποιείται με τυχαίο τρόπο και τα βήματα τα οποία προκύπτουν επηρεάζουν με δικατευθυντήριο τρόπο το ένα το άλλο, δημιουργώντας κυκλικές εξαρτήσεις. Λόγω των εξαρτήσεων απαιτούνται επαναλήψεις των βημάτων για την εύρεση σχεδόν βέλτιστων σχεδιασμών, ενώ δεν υπάρχει εγγύηση ότι μια σχεδόν βέλτιστη σχεδίαση θα ευρεθεί εντός του διαθέσιμου χρόνου εξερεύνησης. Όταν ο αριθμός των παραμέτρων του υλικού και του λογισμικού αυξάνεται, οι ακριβές σχεδιαστικές επαναλήψεις των βημάτων που συσχετίζονται με δικαυτευθυντήριο τρόπο οδηγούν σε μη επεκτάσιμες μεθοδολογίες. Για παράδειγμα, μια επαναλήψιμη DSE μεθοδολογία ξεκινά την εξερεύνηση από μια βασική σχεδίαση από τον σχεδιαστή, τροποποιεί την τιμή μιας παραμέτρου κάθε φορά και χρησιμοποιεί τα αποτελέσματα για να προβλέψει τον βέλτιστο σχεδιασμό [175]. Η DSE μεθοδολογία μπορεί να οδηγήσει σε λιγότερους αποδοτικούς

σχεδιασμούς όταν υπάρχει υψηλός αριθμός παραμέτρων και εξαρτήσεων.

Επιπρόσθετα, κάθε βήμα απεικόνισης του λογισμικού στο υλικό αποτελείται από διεργασίες οι οποίες είναι NP-complete προβλήματα. Οι συνήθης τεχνικές που εφαρμόζονται σε ένα βήμα απεικόνισης μπορούν να επιτύχουν σχεδόν βέλτιστα αποτελέσματα και εντός του διαθέσιμου χρόνου μόνο για μικρά σχεδιαστικά προβλήματα. Όταν η πολυπλοκότητα του συστήματος αυξάνεται, οι συνήθης τεχνικές δεν είναι ικανές στην εύρεση της σχεδόν βέλτιστης λύσης εντός του διαθέσιμου χρόνου. Για παράδειγμα, οι στοχαστικές τεχνικές απαιτούν μη αποδεκτό χρόνο εξερεύνησης για να βρουν σχεδόν βέλτιστες λύσεις σε ένα μεγάλο χώρο λύσεων. Οι στοχαστικές τεχνικές ψάχνουν τον χώρο βασιζόμενες σε τυχαίες κινήσεις. Για να επιτύχουν σχεδόν βέλτιστες λύσεις χρειάζονται υψηλό αριθμό τυχαίων κινήσεων σε γειτονικές περιοχές. Για παράδειγμα, η απόδοση του quantum-inspired evolutionary algorithm (QEA) για την απεικόνιση πολυεπεξεργαστών βασίζεται στον αριθμό των γενιών που θα εφαρμοστούν. Η αύξηση του αριθμού των γενιών αυξάνει και τις πιθανότητες να επιτύχουν έναν σχεδόν βέλτιστο σχεδιασμό [3]. Μια μεθοδολογία εξερεύνησης με στοχαστικούς αλγορίθμους προτάθηκε στο [147] και μια simulated annealing DSE τεχνική για την εύρεση συνεπεξεργαστών στο [73]. Ντετερμινιστικές τεχνικές σχεδιασμού, όπως Integer Linear Programming (ILP) τεχνικές, απαιτούν αρκετό χρόνο εξερεύνησης όταν εφαρμόζονται σε μεσαία και μεγάλα προβλήματα σχεδιασμού. Τεχνικές σχεδίασης τύπου branch and bound επίσης χρειάζονται αυξημένο χρόνο εξερεύνησης για να εγγυηθούν σχεδόν βέλτιστους σχεδιασμούς. Για την μείωση του απαιτούμενου χρόνου εξερεύνησης, η branch and bound τεχνικές πρέπει να εφαρμόσουν ένα πιο επιθετικό κλάδεμα λύσεων στο χώρο εξερεύνησης, ο οποίος μειώνει την ποιότητα του τελικού σχεδιασμού. Ευριστικές μέθοδοι σχεδιασμού ψάχνουν το χώρο λύσεων με προκαθορισμένους κανόνες, οι οποίοι δεν μπορούν εγγυηθούν βέλτιστες λύσεις στο στο γενικό πρόβλημα σχεδιασμού [39].

Οι μεταγλωττιστές είναι βασικό τμήμα των μεθοδολογιών εξερεύνησης και έχουν ως στόχο την απεικόνιση της εφαρμογής στους διαφορετικούς σχεδιασμούς. Τα βασικά στάδια ενός μεταγλωττιστή είναι: 1) η επιλογή κώδικα, δηλαδή η απεικόνιση των εντολών μηχανής στον εκάστοτε επεξεργαστή, 2) η επιλογή καταχωρητών, δηλαδή η απεικόνιση των μεταβλητών στους καταχωρητές με στόχο την μείωση των αναφορών στη μνήμη κατά την εκτέλεση της εφαρμογής, 3) η ανάθεση σε καταχωρητές, δηλαδή ο καθορισμός σε ποιον φυσικό καταχωρητή θα αποθηκευτεί μια μεταβλητή, 4) ο χρονοπρογραμματισμός των εντολών, δηλαδή η αναδιοργάνωση των εντολών για να εξερευνήσουν και να εκμεταλλευτούν την πιθανή παραλληλία στις εντολές και η ανάθεση πόρων, δηλαδή ο καθορισμός των επεξεργαστικών μονάδων και των διαύλων στις πράξεις της εφαρμογής. Οι φάσεις που εκτελούν οι μεταγλωττιστές είναι συσχετιζόμενες με δικαυτευθυντήριο τρόπο, διότι οι αποφάσεις μιας φάσης επηρεάζουν και επιβάλουν περιορισμούς σε επόμενες φάσεις και συνολικά οδηγούν σε υποβέλτιστες λύσεις. Για παράδειγμα, σε έναν μεταγλωττιστή για ενσωματωμένα συστήματα η φάση επιλογής κώδικα τοποθετεί εικονικούς καταχωρητές από διάφορες κλάσεις, αλλά η κλάση από την

οποία πρέπει να επιλεγούν οι καταχωρητές είναι γνωστή μόνο μετά την φάση επιλογής καταχωρητών. Η φάση επιλογής καταχωρητών δεν μπορεί να προηγηθεί από την φάση επιλογής κώδικα, διότι οι απαιτούμενοι καταχωρητές είναι γνωστοί μετά τη φάση επιλογής κώδικα [121]. Οι συνήθεις DSE μεθοδολογίες παρέχουν μη βέλτιστο συνδυασμό των μη βέλτιστων αποτελεσμάτων ανά φάση το οποίο οδηγεί σε συνολικά υπο-βέλτιστη λύση. Ερευνά έχει πραγματοποιηθεί και για την εύρεση της βέλτιστης σειράς εκτέλεσης των φάσεων του μεταγλωττιστή. Δεν έχει ευρεθεί συνολικά βέλτιστη σειρά εκτέλεσης των φάσεων του μεταγλωττιστή η οποία να οδηγεί σε βέλτιστο αποτέλεσμα, διότι επηρεάζεται από την εφαρμογή προς μεταγλώττιση, τον μεταγλωττιστή και την αρχιτεκτονική του συστήματος [103].

Για να μειωθεί ο χρόνος εξερεύνησης, οι περιορισμοί ανάμεσα στα βήματα απεικόνισης, τα οποία είναι συσχετιζόμενα με δικατευθυντήριο τρόπο, δεν λαμβάνονται υπόψιν. Για παράδειγμα, οι παράμετροι σχεδίασης ταξινομούνται με βάση την επιρροή την οποία έχουν και η οποία καθορίζεται από την μέγιστη μεταβολή στην τιμή των παραμέτρων [176]. Όλοι οι συνδυασμοί των δύο πρώτων παραμέτρων λαμβάνονται υπόψιν για περαιτέρω εξερεύνηση. Σε άλλες περιπτώσεις η ανεξαρτησία των παραμέτρων χρησιμοποιείται ώστε να μειώσει το χώρο εξερεύνησης η οποία είναι συνήθως περιορισμένη, και να παράγει την Pareto καμπύλη στο Platune [147]. Τεχνικές που ακολουθούν το διαιρεί και βασίλευε δεν μπορούν να εξερευνήσουν αποδοτικά την δομή του προβλήματος προς σχεδίαση και αγνοούν τους περιορισμούς κατά την διαίρεση σε υπο-προβλήματα. Επιλύουν κάθε υπο-πρόβλημα μεμονωμένα και ανεξάρτητα και όταν συνδυάζουν τα μερικά αποτελέσματα καταλήγουν σε υπο-βέλτιστες λύσεις.

Στην βιομηχανία, τα προβλήματα σχεδίασης είναι μεγάλα και περίπλοκα και επομένως οι DSE μεθοδολογίες πρέπει να εγκαταλείψουν την απαίτηση για σχεδόν βέλτιστη λύση, ούτως ώστε να επιτύχουν την επεκτασιμότητα των μεθοδολογιών σε μεγάλα προβλήματα σχεδίασης. Επομένως, τα βήματα σχεδιασμού εκτελούνται με μερικώς ανεξάρτητο τρόπο για να μειώσουν τις εξαρτήσεις και τον χρόνο εξερεύνησης. Τα υπάρχοντα εργαλεία δεν λαμβάνουν υπόψιν τις κυκλικές εξαρτήσεις ανάμεσα στην επεξεργασία, την μνήμη και την επικοινωνία με αποτέλεσμα την δημιουργία λιγότερο αποδοτικών σχεδιάσεων. Όταν συνδυάζονται τα αποτελέσματα από τα εκάστοτε βήματα, η ποιότητα του σχεδιασμού μειώνεται λόγω συγκρουόμενων αποφάσεων στα διαφορετικά βήματα.

Συμπερασματικά, υπάρχει ένα δίλημμα στις υπάρχουσες μεθοδολογίες εξερεύνησης: είτε να εγκαταλείψουν την απαίτηση για σχεδόν βέλτιστη ποιότητα του τελικού σχεδιασμού ή να εγκαταλείψουν την επεκτασιμότητα στην μεθοδολογία εξερεύνησης.

Β.3 Σκοπός και συνεισφορές

Η παρούσα διδακτορική διατριβή παρουσιάζει μια εναλλακτική μεθοδολογία εξερεύνησης σχεδιασμών η οποία να απαντά στο δίλημμα για σχεδόν βέλτιστη σχεδίαση και επεκτάσιμη

μεθοδολογία. Η προτεινόμενη DSE μεθοδολογία χωρίζει τον περίπλοκο πρόβλημα σχεδιασμού σε μικρότερα και λιγότερο περίπλοκα βήματα σχεδιασμού επιτυγχάνοντας επεκτασιμότητα στη μεθοδολογία. Εν αντιθέσει με τις υπάρχουσες μεθοδολογίες εξερεύνησης του χώρου λύσεων, τα βήματα της προτεινόμενης μεθοδολογίας είναι συνδεδεμένα με διάδοση περιορισμών που ακολουθεί μονοκατευθυντήριο και όχι δικατευθυντήριο τρόπο. Με αυτόν τον τρόπο, επιτυγχάνονται σχεδόν βέλτιστοι σχεδιασμοί διότι δεν αγνοούνται περιορισμοί, το οποίο συμβαίνει όταν τα βήματα θεωρηθούν ανεξάρτητα. Οι ακριβές επαναλήψεις στην εξερεύνηση αποφεύγονται. Στην συνήθεις μεθοδολογίες εξερεύνησης σχεδιασμών το πρόβλημα σχεδίασης χωρίζεται σε βήματα, τα οποία είναι συνδεδεμένα με δικατευθυντήριο τρόπο, και απαιτούν σχεδιαστικές επαναλήψεις, οι οποίες αυξάνουν σημαντικά τον χρόνο εξερεύνησης. Στην παρούσα διδακτορική διατριβή εφαρμόζουμε την επαναχρησιμοποιούμενη επεκτάσιμη και σχεδόν βέλτιστη μεθοδολογία εξερεύνησης του χώρου σχεδιάσεων σε διάφορα στάδια στην σχεδίαση ενσωματωμένων συστημάτων σε υψηλό επίπεδο, και ειδικά για την εύρεση του ελάχιστου μεγέθους μνήμης για την τοποθέτηση των δεδομένων ενός πίνακα στην κύρια μνήμη (intra-signal in-place optimization problem), που εξερευνά τα trade-offs απόδοσης, επιφάνειας ολοκλήρωσης και κατανάλωσης ενέργειας, και για την απεικόνιση του επεξεργαστικού τμήματος του ενσωματωμένου συστήματος, που εξερευνά το trade-off μεταξύ χρόνου σχεδίασης, απόδοσης και επιφάνειας ολοκλήρωσης.

Οι συνεισφορές της παρούσας διδακτορικής διατριβής συνοψίζονται ως εξής:

- **Επαναχρησιμοποιούμενη μεθοδολογία για επεκτάσιμα και σχεδόν βέλτιστα DSE πλαίσια (frameworks).**

  Το πρόβλημα σχεδίασης χωρίζεται σε μικρότερα και λιγότερο πολύπλοκα βήματα σχεδίασης, τα οποία είναι συνδεδεμένα μέσω διάδοσης περιορισμών με μονοκατευθυντήριο τρόπο. Σχεδόν βέλτιστοι σχεδιασμοί επιτυγχάνονται διότι οι περιορισμοί δεν αγνοούνται και η επεκτασιμότητα διατηρείται διότι δεν γίνονται σχεδιαστικές επαναλήψεις.

- **Ανάπτυξη επεκτάσιμης και σχεδόν βέλτιστης μεθοδολογίας για την εύρεση του ελάχιστου μεγέθους απαιτούμενης μνήμης για την τοποθέτηση των δεδομένων ενός πίνακα.**

  Οι αρχές της επαναχρησιμοποιύμενης μεθοδολογίας εφαρμόζονται στο πρόβλημα εύρεσης του ελάχιστου μεγέθους απαιτούμενης μνήμης για την τοποθέτηση των δεδομένων ενός πίνακα και δημιουργούν μια επεκτάσιμη και σχεδόν βέλτιστη μεθοδολογία για σχήματα προσπελάσεων που έχουν υψηλό βαθμό ανομοιογένειας.

- **Επεκτάσιμη και σχεδόν βέλτιστη αναπαράσταση των προσπελάσεων στη μνήμη βασισμένη σε μοτίβα και πράξεις μοτίβων.**

  Οι προσπελάσεις στην κύρια μνήμη οι οποίες χαρακτηρίζονται από υψηλό βαθμό ανομοιογένειας οδήγησαν στην ανάγκη για μια αναπαράσταση, η οποία είναι

βασισμένη σε μοτίβα και πράξεις επί τον μοτίβων, ούτως ώστε να περιγράψει με σχεδόν βέλτιστο και επεκτάσιμο τρόπο τις ανομοιογενείς προσπελάσεις. Τα προτεινόμενα μοτίβα περιγράφουν τις προσπελάσεις ανά συνθήκη. Οι πράξεις εφαρμόζονται στα μοτίβα για να τα συνδυάσουν και να περιγράψουν τον συνολικό χώρο επαναλήψεων (Iteration space), όταν ποικίλες συνθήκες υπάρχουν ταυτόχρονα στην εφαρμογή.

- **Επεκτάσιμη και σχεδόν βέλτιστη μεθοδολογία για την εύρεση του ελάχιστου μεγέθους απαιτούμενης μνήμης για την τοποθέτηση των δεδομένων ενός πίνακα για μη επικαλυπτόμενες και επικαλυπτόμενες εντολές εγγραφής και ανάγνωσης.**
  Επεκτάσιμες και σχεδόν βέλτιστες λύσεις προτείνονται σε κλειστού τύπου εξισώσεις και ρουτίνες (παραμετρικά πλαίσια) για να επιλύσουν τις διάφορες περιπτώσεις σε κάθε βήμα της προτεινόμενης μεθοδολογίας για την εύρεση του ελάχιστου μεγέθους απαιτούμενης μνήμης για την τοποθέτηση των δεδομένων ενός πίνακα με ανομοιογενείς προσπελάσεις και μη επικαλυπτόμενες και επικαλυπτόμενες εγγραφές και αναγνώσεις δεδομένων.

- **Ανάπτυξη μεθοδολογίας για το επεξεργαστικό τμήμα για τη απεικόνιση περιοχής εφαρμογών σε αρχιτεκτονική πλατφόρμα με έναν επεξεργαστή και ποικίλους συνεπεξεργαστές στο υλικό.**
  Μια επεκτάσιμη και σχεδόν βέλτιστη μεθοδολογία εξερεύνησης του χώρου σχεδιάσεων προτείνεται για την απεικόνιση εφαρμογών κυριαρχούμενων από δεδομένα και από βρόγχους σε μια μερικώς προ-αποφασισμένη πλατφόρμα με ένα μικροεπεξεργαστή και ποικίλους συνεπεξεργαστές στο υλικό. Παραμετρικά πλαίσια εφαρμόζονται για κάθε βήμα της μεθοδολογίας. Η προτεινόμενη μεθοδολογία εξερεύνησης δημιουργεί μια μερική Pareto καμπύλη με τους διάφορους σχεδιασμούς για κάθε βήμα απεικόνισης. Η μερική καμπύλη του πρώτου βήματος διαδίδεται στα επόμενα βήματα της μεθοδολογίας και κλαδεύει υποβέλτιστες λύσεις. Οι εναπομείναντες λύσεις συνδυάζονται με την καμπύλη που διαδόθηκε από το προηγούμενο βήμα και δημιουργούν μια καινούργια Pareto καμπύλη, η οποία διαδίδεται στα επόμενα βήματα. Η διαδικασία επαναλαμβάνεται για όλα τα βήματα της μεθοδολογίας εξερεύνησης και το αποτέλεσμα είναι η τελική Pareto καμπύλη με τους διάφορους σχεδιασμούς.

- **Πλαίσιο που περιγράφει τις σχεδόν βέλτιστες τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων που εφαρμόζονται κατά την διάρκεια σχεδίασης.**
  Ο διαχωρισμός και η χρονική ταξινόμηση των διαθέσιμων επιλογών στις τεχνικές που λύνουν το πρόβλημα του χρονοπρογραμματισμού και της ανάθεσης των πόρων που περιγράφεται σε μια κατηγοριοποίηση μέσω ενός DSE framework. Οι τεχνικές εφαρμόζονται στην διαχείριση της μνήμης στο επεξεργαστικό τμήμα του συστήματος (foreground) και στην εκτέλεση των εντολών (data path). Η κατηγοριοποίηση

περιγράφει αποδοτικά το σύνολο των επιλογών του χρόνου εξερεύνησης για τις τεχνικές και περιγράφει την διάδοση περιορισμών ανάμεσα στις τεχνικές.

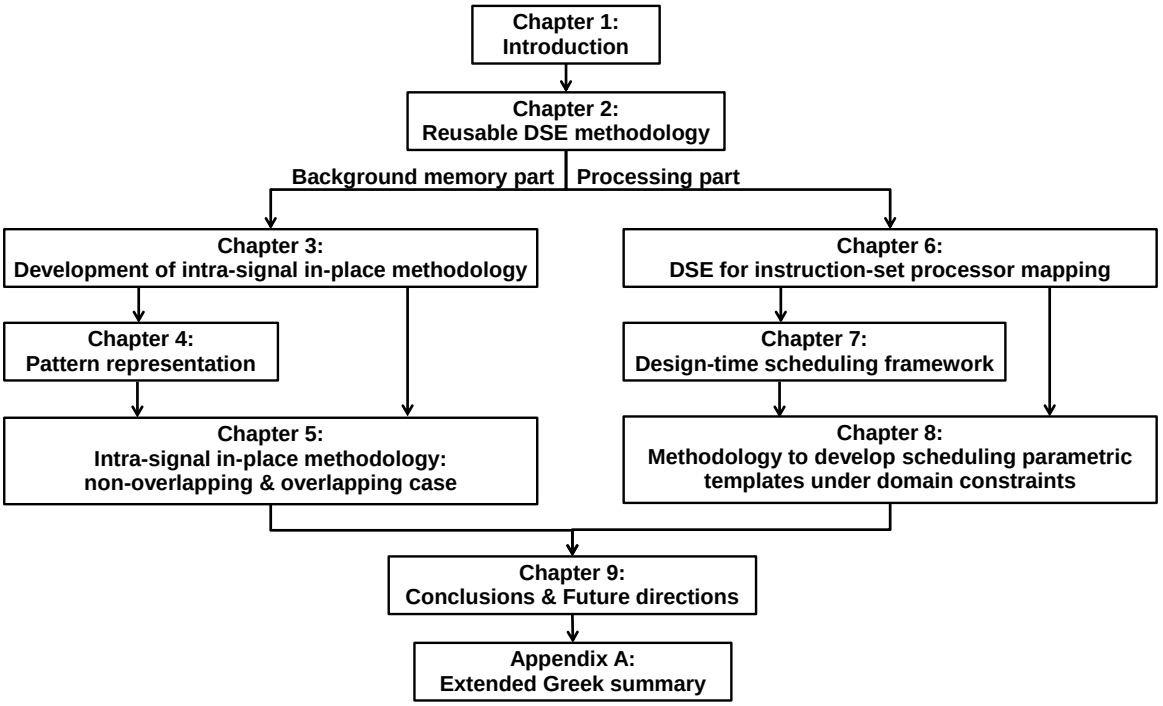
- **Συστηματική μεθοδολογία για την ανάπτυξη παραμετρικών πλαισίων για σχεδόν βέλτιστες και επεκτάσιμες τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων.**
  Προτείνεται μια συστηματική μεθοδολογία εξερεύνησης που χρησιμοποιεί την κατηγοριοποίηση των τεχνικών που λύνουν το πρόβλημα του χρονοπρογραμματισμού και της ανάθεσης των πόρων σαν βάση και παράγει ένα νέο framework στο οποίο λαμβάνονται οι επιπρόσθετοι περιορισμοί της περιοχής που μελετάται κάθε φορά. Οι κλάσεις που περιγράφουν τις τεχνικές που λύνουν το πρόβλημα του χρονοπρογραμματισμού και της ανάθεσης των πόρων του καινούργιου framework περιγράφονται από παραμετρικά πλαίσια, τα οποία συνδυάζονται ακολουθώντας μονοκατευθυντήρια διάδοση περιορισμών. Το αποτέλεσμα είναι το τελικό παραμετρικό πλαίσιο που περιγράφει επεκτάσιμες και σχεδόν βέλτιστες τεχνικές που λύνουν το πρόβλημα του χρονοπρογραμματισμού και της ανάθεσης των πόρων για την υπο μελέτη περιοχή.

Η παρούσα διδακτορική διατριβή μελετά τα υψηλά επίπεδα εξερεύνησης σχεδιασμών και δεν επικεντρώνεται στα χαμηλότερα επίπεδα που περιγράφουν τη λεπτομερή απεικόνιση στην αρχιτεκτονική πλατφόρμα, π.χ. τα επίπεδα χρονοπρογραμματισμού και ανάθεσης πόρων χωρίς να περιλαμβάνουν την δημιουργία τελικού κώδικα και την αρχιτεκτονική σύνθεση. Στην παρούσα διδακτορική διατριβή περιγράφεται ένα διαφορετικό είδος εξερεύνησης σχεδιασμών. Η επιβεβαίωση της σχεδιαστικής διαδικασίας αποτελεί ένα σχετικό ερευνητικό θέμα, το οποίο ωστόσο δεν προσεγγίζεται στην παρούσα διδακτορική διατριβή. Παρεμφερείς μεθοδολογίες μπορούν να δημιουργηθούν ακολουθώντας τις τεχνικές που περιγράφονται στην αναφορά [85] για τις μεθοδολογίες τις παρούσας διδακτορικής διατριβής.

B.4 Οργάνωση κεφαλαίων

Η παρούσα διδακτορική διατριβή αποτελείται από δύο μέρη. Το πρώτο μέρος αποτελείται από τα κεφάλαια 3 έως 5 και πραγματεύεται στο αποθηκευτικό τμήμα των ενσωματωμένων συστημάτων και πιο συγκεκριμένα στην κύρια μνήμη. Το δεύτερο τμήμα αποτελείται από τα κεφάλαια 6 έως 8 και είναι αφιερωμένο στο επεξεργαστικό τμήμα των αρχιτεκτονικών υλικού και λογισμικού των ενσωματωμένων συστημάτων. Οι εξαρτήσεις των κεφαλαίων παρουσιάζονται στο Σχ. B.1.

- Το κεφάλαιο 2 περιγράφει την επεναχρησιμοποιούμενη μεθοδολογία για την δημιουργία και την χρήση επεκτάσιμων και σχεδόν βέλτιστων Design Space Exploration (DSE) frameworks. Η επαναχρησιμοποιούμενη DSE μεθοδολογία του



Σχ. Β.1: Γράφος που απεικονίζει τις εξαρτήσεις των κεφαλαίων

κεφαλαίου 2 εφαρμόζεται σε πολλαπλά και διαφορετικά περιεχόμενα και στάδια σχεδίασης στην παρούσα διδακτορική διατριβή με στόχο την ανάπτυξη επεκτάσιμων και σχεδόν βέλτιστων μεθοδολογιών.

- Το κεφάλαιο 3 περιγράφει το υπο μελέτη πρόβλημα απεικόνισης για την κύρια μνήμη, δηλαδή την εύρεση της ελάχιστης απαιτούμενης μνήμης για την τοποθέτηση των δεδομένων ενός πίνακα με προσπελάσεις οι οποίες έχουν υψηλό βαθμό ανομοιογένειας. Η επαναχρησιμοποιούμενη DSE μεθοδολογία εφαρμόστηκε στο πρόβλημα της εύρεσης του ελάχιστου απαιτούμενου χώρου και δημιούργησε ένα framework που περιγράφει τα γενικά βήματα και την διάδοση περιορισμών ανάμεσα στα βήματα για να παραχθεί μια επεκτάσιμη και σχεδόν βέλτιστη μεθοδολογία. Ο στόχος χωρίζεται σε υπο-στόχους, δηλαδή στον υπολογισμό του ελαχίστου μεγέθους, στη μετάφραση του σχήματος των προσπελάσεων σε μια επεκτάσιμη και σχεδόν βέλτιστη αναπαράσταση και στην ανάλυση της εφαρμογής για την εύρεση της πληροφορίας σχετικά με την δομή του προγράμματος και τις προσπελάσεις στη μνήμη. Η επαναχρησιμοποιούμενη DSE μεθοδολογία εφαρμόζεται ανά υπο-στόχο για να αναπτυχθούν αποδοτικοί διαχωρισμοί των πιθανόν περιπτώσεων και λύσεων ανά υπο-στόχο. Τα παραμετρικά πλαίσια που προκύπτουν και περιγράφουν τις λύσεις των διαφόρων περιπτώσεων παρουσιάζονται στο κεφάλαιο 5.

- Το κεφάλαιο 4 περιγράφει την προτεινόμενη αναπαράσταση για τις προσπελάσεις των δεδομένων των πινάκων, η οποία είναι κατάλληλη για να εκφράζει ανομοιογενείς προσπελάσεις με επεκτάσιμο και σχεδόν βέλτιστο τρόπο. Προτείνεται αναπαράσταση βασισμένη σε μοτίβα τα οποία περιγράφουν τις προσπελάσεις με

συμπαγή και επαναληπτικό τρόπο. Επιπρόσθετα προτείνονται πράξεις που ενεργούν επάνω στα μοτίβα με στόχο να ενώσουν τα μοτίβα με επεκτάσιμο και σχεδόν βέλτιστο τρόπο κάτω από όλες τις πιθανές συνθήκες που μπορούν να δημιουργηθούν από την εφαρμογή. Το σύνολο όλων των πιθανών περιπτώσεων σύνδεσης μοτίβων προκύπτει από την εφαρμογή της επαναχρησιμοποιούμενης DSE μεθοδολογίας του κεφαλαίου 2. Η αναπαράσταση βασισμένη σε μοτίβα χρησιμοποιείται στις λύσεις που προτείνονται στο βήμα της μετάφρασης της μεθοδολογίας για την εύρεση του ελάχιστου απαιτούμενου χώρου.

- Το κεφάλαιο 5 περιγράφει τα παραμετρικά πλαίσια της προτεινόμενης μεθοδολογίας για την εύρεση του ελάχιστου απαιτούμενου χώρου στην κύρια μνήμη για την τοποθέτηση των δεδομένων ενός πίνακα για ανομοιογενείς προσπελάσεις και με μη επικαλυπτόμενες και επικαλυπτόμενες εντολές εγγραφής και ανάγνωσης. Η προτεινόμενη μεθοδολογία αποτελείται από το βήμα της ανάλυσης, το βήμα της μετάφρασης και το βήμα υπολογισμού του απαιτούμενου χώρου. Το βήμα της ανάλυσης περιγράφει τις εφαρμογές υπό μελέτη και τις διαφορετικές συνθήκες που μπορεί να υπάρξουν σε ένα μοναδικό πλαίσιο, το οποίο περιγράφει την εφαρμογή και παρέχει την απαιτούμενη πληροφορία από την εφαρμογή που εξετάζεται. Το βήμα της μετάφρασης περιγράφει τις λύσεις για τις διάφορες συνθήκες μεταφράσεις που μπορεί να υπάρχουν χρησιμοποιώντας μοτίβα και πράξεις επάνω στα μοτίβα, όπως περιγράφηκαν στο κεφάλαιο 4.

- Το κεφάλαιο 6 περιγράφει μια μεθοδολογία εξερεύνησης των διαφόρων σχεδιασμών, η οποία δημιουργεί μια καμπύλη Pareto για την απεικόνιση της εφαρμογής σε μια μερικώς προαποφασισμένη αρχιτεκτονική πλατφόρμα που αποτελείται από επεξεργαστές, οι οποίοι ελέγχονται από εντολές, και περιλαμβάνουν ένα πυρήνα επεξεργαστή και πολλαπλούς συνεπεξεργαστές στο υλικό. Η προτεινόμενη μεθοδολογία προέρχεται από την εφαρμογή των αρχών του κεφαλαίου 2 στην απεικόνιση μερικών προαποφασισμένων πλατφορμών υλικού και λογισμικού με ένα μικροεπεξεργαστή και ποικίλους συνεπεξεργαστές. Τα βήματα της μεθοδολογίας είναι η ανάλυση και η επιβεβαίωση ότι οι περιορισμοί του συστήματος ικανοποιούνται μέσω προσεγγίσεων σε υψηλό επίπεδο, η οργάνωση μεταξύ υλικού και λογισμικού, η οργάνωση της μνήμης στο επεξεργαστικό τμήμα και η εκτέλεση των πράξεων. Για κάθε βήμα χρησιμοποιούνται παραμετρικά πλαίσια με παραμέτρους από το λογισμικό και το υλικό. Στο πρώτο βήμα γίνεται ανάγνωση της εφαρμογής για την παροχή τιμών στις παραμέτρους του πρώτου βήματος. Το αποτέλεσμα διαδίδεται στο επόμενο βήμα, όπου οι επιλογές και οι τιμές των παραμέτρων του δεύτερου παραμετρικού πλαισίου αποκλείονται με βάση τις τιμές που διαδόθηκαν από το πρώτο βήμα. Οι επιλογές που παρέμειναν ενώνονται με τις επιλογές που διαδόθηκαν από το πρώτο βήμα και η διαδικασία επαναλαμβάνεται μέχρι το τελικό βήμα όπου γίνεται η απεικόνιση των πράξεων στις μονάδες εκτέλεσης.

- Το κεφάλαιο 7 περιγράφει το αποτέλεσμα της εφαρμογής των αρχών της επαναχρησιμοποιούμενης μεθοδολογίας εξερεύνησης σε σχεδόν βέλτιστες τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων. Οι τεχνικές εφαρμόζονται στο χρονοπρογραμματισμό και την ανάθεση των μεταβλητών στους καταχωρητές στην διαχείριση της μνήμης στο επεξεργαστικό τμήμα και το χρονοπρογραμματισμός και την ανάθεση των εντολών στις μονάδες εκτέλεσης. Το αποτέλεσμα είναι μια κατηγοριοποίηση με το σύνολο των τεχνικών χρονοπρογραμματισμού και ανάθεσης πόρων και την χρονική ταξινόμηση των τεχνικών. Η κατηγοριοποίηση περιγράφει αποδοτικά τις δυνατές περιπτώσεις του χώρου λύσεων.

- Το κεφάλαιο 8 εφαρμόζει τις αρχές της επαναχρησιμοποιούμενης μεθοδολογίας εξερεύνησης στην κατηγοριοποίηση των σχεδόν βέλτιστων τεχνικών χρονοπρογραμματισμού και ανάθεσης πόρων για να δημιουργήσει μια συστηματική μεθοδολογία που αναπτύσσει παραμετρικά πλαίσια για τον χρονοπρογραμματισμό και την ανάθεση πόρων που ικανοποιούν τους περιορισμούς που τίθενται από την υπό εξερεύνηση περιοχή. Με αυτό τον τρόπο δημιουργούνται επεκτάσιμα και παραμετρικά πλαίσια για τον χρονοπρογραμματισμό και την ανάθεση πόρων τα οποία εγγυούνται σχεδόν βέλτιστα αποτελέσματα για την υπο μελέτη περιοχή. Το αναπτυσσόμενο πλαίσιο μπορεί να χρησιμοποιηθεί από την μνήμη που βρίσκεται στο επεξεργαστικό τμήμα και στην απεικόνιση των εντολών στις μονάδες εκτέλεσης.

- Το κεφάλαιο 9 παρουσιάζει μια σύνοψη με τα συμπεράσματα και τις μελλοντικές κατευθύνσεις.

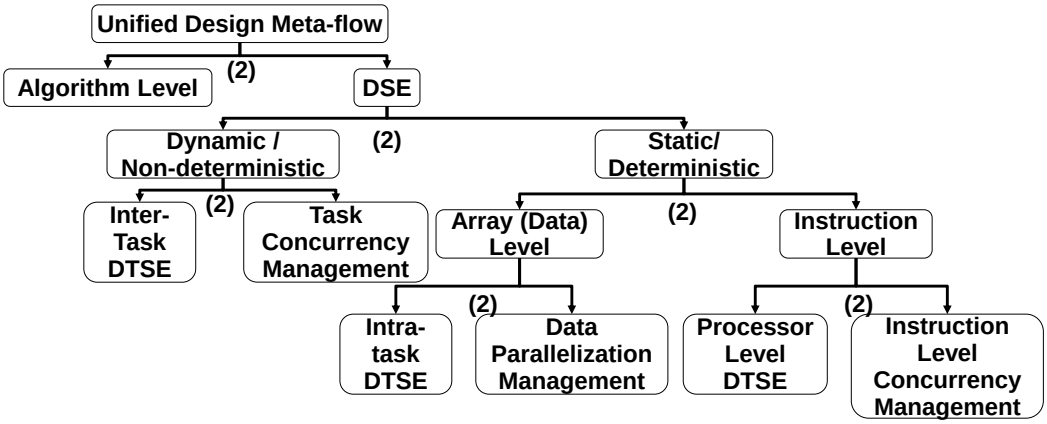
Β.5 Επαναχρησιμοποιούμενη μεθοδολογία για επεκτάσιμα και σχεδόν βέλτιστα DSE πλαίσια (frameworks).

Ο στόχος είναι να δημιουργήσουμε έναν τρόπο ώστε να επιλύονται μεγάλα, περίπλοκα και εξαρτημένα προβλήματα εξερεύνησης των σχεδιάσεων με σχεδόν βέλτιστο και επεκτάσιμο τρόπο. Σε αυτή την υπο μελέτη περιοχή, οι συνήθεις τεχνικές είναι λιγότερο αποδοτικές γιατί βασίζονται σε bottom-up προσεγγίσεις χωρίς διαχωρισμούς που να δημιουργούνται λόγω διάδοσης περιορισμών. Επομένως, οι συνήθεις τεχνικές απαιτούν αυξημένο χρόνο εξερεύνησης για να εντοπίσουν σχεδόν βέλτιστους συνδυασμούς, λόγω του υψηλού αριθμού παραμέτρων και των συσχετίσεων ανάμεσα στις παραμέτρους, ή καταλήγουν με ένα λιγότερο βέλτιστο αποτέλεσμα ούτως ώστε να ολοκληρώσουν την εξερεύνηση των σχεδιασμών εντός του διαθέσιμου χρόνου.

Για να επιτύχουμε σχεδόν βέλτιστες σχεδιάσεις σε αποδεκτό χρόνο εξερεύνησης

απαιτείται μια μεθοδολογία που διαιρεί το πρόβλημα σε υπο-προβλήματα, με αποτέλεσμα την μείωση του χρόνου εξερεύνησης, και να διατηρεί όλη την λειτουργικότητα των υπο-προβλημάτων, για να εγγυηθεί σχεδόν βέλτιστες σχεδιάσεις. Όταν το σχεδιαστικό πρόβλημα έχει διαιρεθεί σε μικρότερα υπο-προβλήματα, οι συνήθεις τεχνικές σχεδιασμού μπορούν να επιτύχουν σχεδόν βέλτιστες σχεδιάσεις σε αποδεκτό χρόνο εξερεύνησης για κάθε υπο-πρόβλημα. Ο διαχωρισμός σε υπο-προβλήματα μπορεί να πραγματοποιηθεί με εύρεση όλων των πιθανών διαχωρισμών και των συνδυασμών τους. Η καταμέτρηση όλων των πιθανών συνδυασμών οδηγεί σε τεράστιο αριθμό και δεν μπορεί να εφαρμοστεί σε μεγάλα σχεδιαστικά προβλήματα. Μια εναλλακτική λύση είναι η εφαρμογή μιας top-down μεθοδολογίας που βασίζεται σε γκρι κουτιά. Αντί για καταμέτρηση όλων των πιθανών επιλογών, οι επιλογές εξερευνούνται σε ιεραρχικά αφαιρετικά επίπεδα και με επαναλήψιμο τρόπο. Σε κάθε επίπεδο, οι επιλογές ομαδοποιούνται σε περιπτώσεις με δεδομένη διεπαφή (προσδιορισμένο το ``τι'' επιλύουν) και σε άγνωστο εσωτερικό τμήμα (απροσδιόριστο το ``πως'' το επιλύουν). Ωστόσο, οι περιπτώσεις δεν είναι ανεξάρτητες. Σε κάθε επίπεδο οι περιπτώσεις συνδέονται μέσω διάδοσης περιορισμών με μονοκατευθυντήριο τρόπο. Η κατεύθυνση διάδοσης περιγράφει ποια περίπτωση επηρεάζει την άλλη. Ακολουθώντας την μονοκατευθυντήρια διάδοση των περιορισμών δημιουργείται σχεδόν βέλτιστος συνδυασμός των διαφόρων περιπτώσεων, διότι οι περιορισμοί που υπάρχουν δεν παραβλέπονται κατά τον συνδυασμό. Η μονοκατευθυντήρια διάδοση περιορισμών περικόπτει συνδυασμούς οι οποίοι είναι υποβέλτιστοι. Το αποτέλεσμα είναι ένα πλαίσιο (framework) με τον διαχωρισμό του χώρου εξερεύνησης σε ένα σύνολο από περιπτώσεις που συνδέονται με μονοκατευθυντήριο τρόπο. Η μεθοδολογία για την δημιουργία του framework εφαρμόζεται επαναληπτικά αρχίζοντας από το πρόβλημα προς επίλυση με στόχο να το περιγράψει με ένα σύνολο υπο-προβλημάτων αρκετά λιγότερο περίπλοκων και που συνδέονται με μονοκατευθυντήρια διάδοση παραμέτρων. Η διαδικασία προσδιορισμού των υπο-περιπτώσεων επαναλαμβάνεται έως ότου οι υπο-περιπτώσεις είναι αρκετά μικρές ώστε να μπορούν να επιλυθούν σχεδόν βέλτιστα και σε αποδεκτό χρόνο από τις συνήθεις τεχνικές. Κάθε υπο-πρόβλημα επιλύεται και το αποτέλεσμα διαδίδεται στο επόμενο υπο-πρόβλημα ακολουθώντας την μονοκατευθυντήρια διάδοση περιορισμών.

Στην παρούσα διδακτορική διατριβή προτείνεται μια επαναχρησιμοποιούμενη μεθοδολογία η οποία βασίζεται στη συστηματική δημιουργία επεκτάσιμων (scalable) και σχεδόν βέλτιστων (near-optimal) πλαισίων (frameworks) για την εξερεύνηση των διαφόρων σχεδιάσεων. Ένα DSE framework περιγράφει όλες τις διαθέσιμες επιλογές στην εξερεύνηση του χώρου σχεδιάσεων με ένα πεπερασμένο σύνολο περιπτώσεων. Παρουσιάζουμε τις βασικές αρχές που διέπουν την δημιουργία και την χρήση των επεκτάσιμων και σχεδόν βέλτιστων DSE frameworks και παρουσιάζουμε επαναχρησιμοποιημένες μεθοδολογίες για την δημιουργία και την χρήση των DSE frameworks στο κεφάλαιο 2. Η εφαρμογή των αρχών στο πρόβλημα σχεδιασμού των ενσωματωμένων συστημάτων παράγει ένα μοναδική σχεδιαστική ροή η οποία



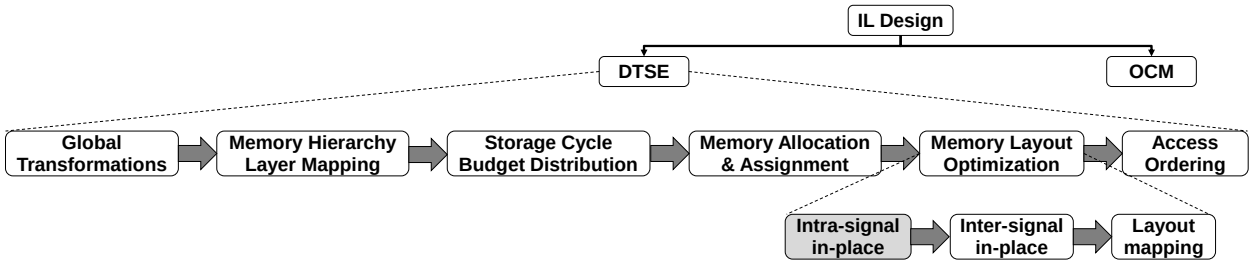
Σχ. Β.2: Ροή για την σχεδίαση ενσωματωμένων συστημάτων που βασίζεται σε μονοκατευθυντήρια διάδοση περιορισμών.

παρουσιάστηκε στα [21] [19] και απεικονίζεται στο Σχ. Β.16

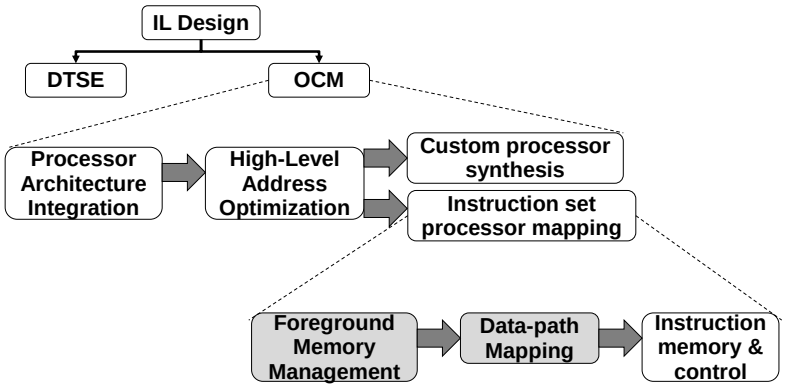
Λόγω των αρχών και των ιδιοτήτων της επαναχρησιμοποιούμενης DSE μεθοδολογίας, η σχεδιαστική ροή δεν περιέχει επικαλυπτόμενα βήματα ή σχεδιαστικές επαναλήψεις. Επομένως, διαχωρίζεται σε διαφορετικά αφαιρετικά επίπεδα που συνδέονται με μονοκατευθυντήρια διάδοση παραμέτρων. Τα επίπεδα δημιουργήθηκαν από την εφαρμογή top-down διαχωρισμών. Η περιγραφή των βημάτων παρουσιάζεται στο κεφάλαιο 2.

Η σχεδιαστική ροή δεν είναι μια πολύ λεπτομερή ροή για μια συγκεκριμένη περιοχή εφαρμογών και αρχιτεκτονικών. Χρειάζεται μεγάλη ερευνητική προσπάθεια για να δημιουργηθεί για ειδική ροή για μία συγκεκριμένη περιοχή εφαρμογών [51]. Για μια συγκεκριμένη περιοχή μπορεί να μην είναι κρίσιμα όλα τα βήματα της γενικής σχεδιαστικής ροής διότι τα χαρακτηριστικά και οι περιορισμοί της ροής μπορεί να μην συσχετίζονται με μερικά από τα βήματα. Τα μη σχετικά βήματα κόβονται όταν δημιουργείται η ειδική σχεδιαστική ροή για μια περιοχή. Για τα υπόλοιπα βήματα που υπάρχουν στην ειδική σχεδιαστική ροή, κατάλληλες τεχνικές θα πρέπει να επιλεγούν ή να δημιουργηθούν για να υλοποιήσουν με σχεδόν βέλτιστο και επεκτάσιμο τρόπο το εκάστοτε βήμα. Η ανάπτυξη και η δημιουργία των απαιτούμενων τεχνικών εξαρτάται από την υπό μελέτη περιοχή κάθε φορά.

Ο στόχος της παρούσας διδακτορικής διατριβής είναι να προσδιορίσει περεταίρω τα βήματα της σχεδιαστικής ροής [21] [19] με την εφαρμογή των αρχών της επαναχρησιμοποιούμενης μεθοδολογίας εξερεύνησης που παρουσιάζεται στο κεφάλαιο 2 και να δημιουργήσει επεκτάσιμα και σχεδόν βέλτιστα DSE frameworks. Η παρούσα διδακτορική διατριβή επικεντρώνεται στο επίπεδο του επεξεργαστή της σχεδιαστικής ροής (Processor Level) και εφαρμόζει τις αρχές της επαναχρησιμοποιούμενης μεθοδολογίας για την δημιουργία επεκτάσιμων και σχεδόν βέλτιστων λύσεων για την οργάνωση της αποθήκευσης και της μεταφοράς των δεδομένων (Processor Level Data Transfer and Storage Exploration) και το επεξεργαστικό τμήμα (Instruction Level Operation Concurrency Manage-



Σχ. Β.3: Σχεδιαστικά βήματα για την οργάνωση της αποθήκευσης και της μεταφοράς των δεδομένων.



Σχ. Β.4: Σχεδιαστικά βήματα για το επεξεργαστικό τμήμα.

ment). Τα σχεδιαστικά βήματα εμφανίζονται στο Σχ. Β.3 και Σχ. Β.4.

Οι προτεινόμενες αρχές και οι επαναχρησιμοποιημένες μεθοδολογίες εφαρμόστηκαν σε στάδια της σχεδίασης των ενσωματωμένων συστημάτων με στόχο την ανάπτυξη επεκτάσιμων και σχεδόν βέλτιστων μεθοδολογιών εξερεύνησης των σχεδιασμών. Η παρούσα διδακτορική διατριβή αναπτύσσει μια μεθοδολογία για σχεδόν βέλτιστη λύση στο πρόβλημα εύρεσης του ελαχίστου μεγέθους μνήμης για την αποθήκευση των δεδομένων ενός πίνακα, η οποία παραμένει επεκτάσιμη και σχεδόν βέλτιστη σε ανομοιογενείς προσπελάσεις δεδομένων, σε αντίθεση με τις υπάρχουσες τεχνικές. Επιπρόσθετα εφαρμόστηκαν οι αρχές της επαναχρησιμοποιούμενης μεθοδολογίας στο επεξεργαστικό τμήμα για την ανάπτυξη μεθοδολογίας εξερεύνησης των σχεδιασμών σε αρχιτεκτονική πλατφόρμα με έναν επεξεργαστή και ποικίλους συνεπεξεργαστές και στις τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων για την διαχείριση της μνήμης στο επεξεργαστικό τμήμα και των εντολών στις μονάδες εκτέλεσης.

Β.6 Ανάπτυξη επεκτάσιμης και σχεδόν βέλτιστης μεθοδολογίας για το ελάχιστο μέγεθος μνήμης για τα δεδομένα ενός πίνακα.

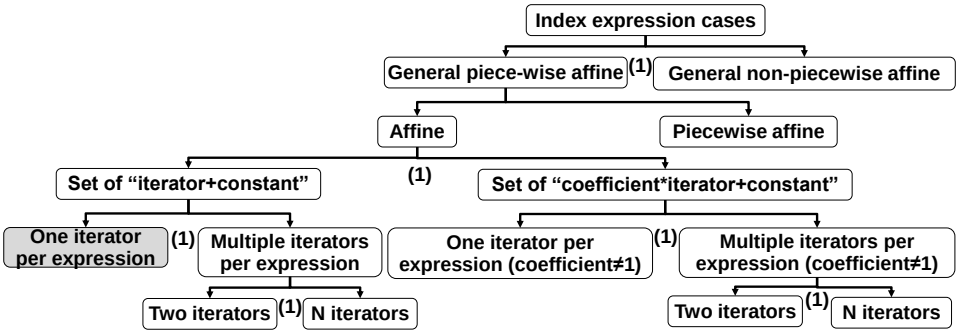
Οι τεχνικές για την διαχείριση του αποθηκευτικού χώρου του συστήματος και της κύριας μνήμης ψάχνουν για το ελάχιστο αριθμό πόρων που απαιτούνται για την αποθήκευση των στοιχείων της εφαρμογής δίχως να δημιουργούν μια μη αποδοτική διευθυνσιοδότηση των στοιχείων κατά την προσπέλαση τους από την κύρια μνήμη. Οι τεχνικές διαχείρισης αποθηκευτικού χώρου του συστήματος εφαρμόζονται σε διάφορες περιοχές όπως στις μνήμες scratch pad των ενσωματωμένων συστημάτων [37], στις μνήμες που ελέγχονται από hardware των συστημάτων γενικού σκοπού [18] και στην διαχείριση του αποθηκευτικού χώρου στην βιομηχανία, όπως τα συστήματα cargo [108]. Η μείωση των αριθμών των πόρων του συστήματος είναι σημαντική, διότι άμεσα συσχετίζεται με το κόστος του συστήματος, την επιφάνεια ολοκλήρωσης και την κατανάλωση ενέργειας [18]. Στα ενσωματωμένα συστήματα, το κόστος της κατανάλωσης ισχύος είναι κυριαρχούμενο από την αποθήκευση των πινάκων, και, επομένως, τα στοιχεία μνήμης καταλαμβάνουν ένα μεγάλο μέρος του συνολικού κόστους [18]. Για συχνά χρησιμοποιούμενες εφαρμογές ενσωματωμένων συστημάτων, όπως εφαρμογές σε εικόνα, βίντεο και επεξεργασία σήματος, όπου τα κυριαρχούμενα δεδομένα είναι πίνακες, η οργάνωση της αποθήκευσης των πινάκων γίνεται ένα πολύ σημαντικό τμήμα της σχεδίασης. Μια λιγότερο αποδοτική οργάνωση οδηγεί σε υπερεκτίμηση των αναγκαίων πόρων του συστήματος, η οποία άμεσα αυξάνει τις απαιτήσεις σε μνήμη και σε επιφάνεια ολοκλήρωσης, αυξάνοντας την κατανάλωση ενέργειας του συστήματος. Για να επιτευχθεί σχεδόν βέλτιστη οργάνωση της διαχείρισης της μνήμης, τόσο η ανεξάρτητη απεικόνιση των πινάκων (intra-signal in-place optimization) όσο και η ταυτόχρονη απεικόνιση όλων των πινάκων της εφαρμογής (inter-signal in-place optimization) είναι κρίσιμα βήματα κατά την σχεδίαση [42]. Η παρούσα διδακτορική διατριβή παρουσιάζει μια μεθοδολογία για την ανεξάρτητη απεικόνιση ενός πίνακα, η οποία είναι επεκτάσιμη και σχεδόν βέλτιστη για προσπελάσεις μνήμης που χαρακτηρίζονται από υψηλή ανομοιογένεια. Μια παρεμφερής διαδικασία μπορεί να χρησιμοποιηθεί για να αναπτύξει μια επεκτάσιμη και σχεδόν βέλτιστη μεθοδολογία για την ταυτόχρονη αποθήκευση των πινάκων της εφαρμογής στην κύρια μνήμη.

Υπάρχουσες τεχνικές για την διαχείριση της μνήμης και για την ανεξάρτητη απεικόνιση των στοιχείων ενός πίνακα στην κύρια μνήμη βασίζονται στην καταμέτρηση, σε συμβολικές/πολύτοπες αναλύσεις και σε προσεγγίσεις με βάση την χειρότερη περίπτωση, όπως περιγράφονται αναλυτικά στο κεφάλαιο 3. Οι τεχνικές καταμέτρησης βρίσκουν το βέλτιστο απαιτούμενο χώρο αποθήκευσης, αλλά δεν είναι επεκτάσιμες. Επομένως, όταν ο αριθμός των προσπελάσεων αυξάνεται, ο χρόνος εξερεύνησης φτάνει σε μη αποδεκτές τιμές. Οι συμβολικές τεχνικές είναι κυρίως πολυεδρικές [42] [31], οι οποίες

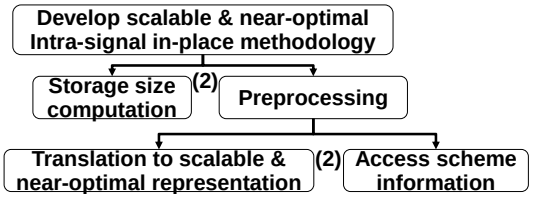
είναι επεκτάσιμες με βάση των αριθμό των προσπελάσεων, αλλά λειτουργούν αποδοτικά μόνο σε χώρους επαναλήψεων οι οποίοι είναι συνεχόμενοι [194], δηλαδή προσπελάσεις χωρίς κενά στον χώρο επαναλήψεων. Με επιπλέον προεπεξεργασία οι συμβολικές τεχνικές μπορούν να εφαρμοστούν σε ομοιογενείς χώρους (piecewise regular spaces), όπως η τεχνική στο [55]) που εφαρμόζεται σε χώρους επαναλήψεων με σχεδόν ομοιογενή κενά. Ένα πολύεδρο ορίζει αποδοτικά τον απαιτούμενο χώρο αποθήκευσης μιας γεωμετρικής περιοχής, δηλαδή προσπελάσεις στον χώρο επαναλήψεων που είναι ομοιογενείς και μπορούν να αναπαρασταθούν από πίνακες ή έχουν πολύ μικρού μεγέθους κενά, τα οποία προσεγγίζονται από συνεχόμενες περιοχές, δηλαδή θεωρούνται ως προσπελάσεις. Όταν υπάρχουν ποικίλες γεωμετρικές περιοχές, οι πολυεδρικές τεχνικές είτε χρειάζονται αυξημένο χρόνο εξερεύνησης είτε πρέπει να προσεγγίσουν αρκετά το χώρο, οδηγώντας σε υπερεκτίμηση των πόρων, διότι η ανομοιογένεια των κενών αυξάνεται λόγων των συνθηκών που καταστρέφουν την συνεχόμενη εκτέλεση των εντολών προσπέλασης. Τότε, οι συμβολικές τεχνικές πρέπει να χρησιμοποιήσουν προσεγγίσεις με βάση την χειρότερη περίπτωση [160]. Οι τεχνικές που πραγματοποιούν τις προσεγγίσεις, π.χ. [160], θεωρούν ότι τα κενά του χώρου επαναλήψεων είναι προσπελάσεις και δημιουργούν ένα συνεχόμενο χώρο επαναλήψεων και υπολογίζουν ένα μη βέλτιστο αποτέλεσμα. Επομένως χρειάζεται μια μεθοδολογία για την ανεξάρτητη απεικόνιση των στοιχείων, που να παραμένει επεκτάσιμη και σχεδόν βέλτιστη σε προσπελάσεις με υψηλό βαθμό ανομοιογένειας.

Η μεθοδολογία για την εύρεση της ελάχιστης απαιτούμενης μνήμης για την αποθήκευση των στοιχείων ενός πίνακα θα πρέπει να λαμβάνει υπόψιν όλες τις εντολές προσπέλασης και τις σχετικές συνθήκες για τον πίνακα με στόχο να ορίσει τον ελάχιστο απαιτούμενο χώρο κατά την διάρκεια εκτέλεσης της εφαρμογής. Οι υπο μελέτη εφαρμογές διαφέρουν και επομένως έχουν διαφορετική δομή, διαφορετικές συνθήκες και εντολές προσπέλασης στον κώδικά τους. Επομένως δημιουργούν διαφορετικές περιπτώσεις για την εύρεση του ελαχίστου μεγέθους μνήμης που απαιτούν διαφορετική διαδικασία για να επιτύχουν σχεδόν βέλτιστο μέγεθος. Δεν είναι δυνατή η δημιουργία μιας γενικής και επεκτάσιμης μεθοδολογίας που να βρίσκει το βέλτιστο μέγεθος για όλες τις εφαρμογές και όλες τις περιπτώσεις. Επομένως, για να επιτευχθεί τόσο η επεκτασιμότητα όσο και η σχεδόν βέλτιστη λύση απαιτείται μια γενική μεθοδολογία που να χωρίζεται σε βήματα με λύσεις για κάθε πιθανή περίπτωση.

Στο κεφάλαιο 3 εφαρμόζεται η επαναχρησιμοποιούμενη DSE μεθοδολογία του κεφαλαίου 2 με σκοπό την ανάπτυξη μιας επεκτάσιμης και σχεδόν βέλτιστης μεθοδολογίας για την εύρεση του ελαχίστου μεγέθους μνήμης για την αποθήκευση των στοιχείων ενός πίνακα σε περιπτώσεις όπου υπάρχουν ανομοιογενείς προσπελάσεις που δημιουργούνται από εντολές προσπέλασης μέσα σε ένα βρόχο με συνθήκες πάνω στην μεταβλητή του βρόχου. Στο κεφάλαιο 3.2 δίνουμε το κίνητρο για την προτεινόμενη μεθοδολογία μέσω ενός χαρακτηριστικού παραδείγματος. Επιπρόσθετα, σε αυτό το κεφάλαιο παρουσιάζονται οι υπάρχουσες τεχνικές για την εύρεση του ελαχίστου μεγέθους μνήμης που απαιτείται για την αποθήκευση των στοιχείων ενός πίνακα και παρουσιάζεται το υπό μελέτη πρόβλημα.



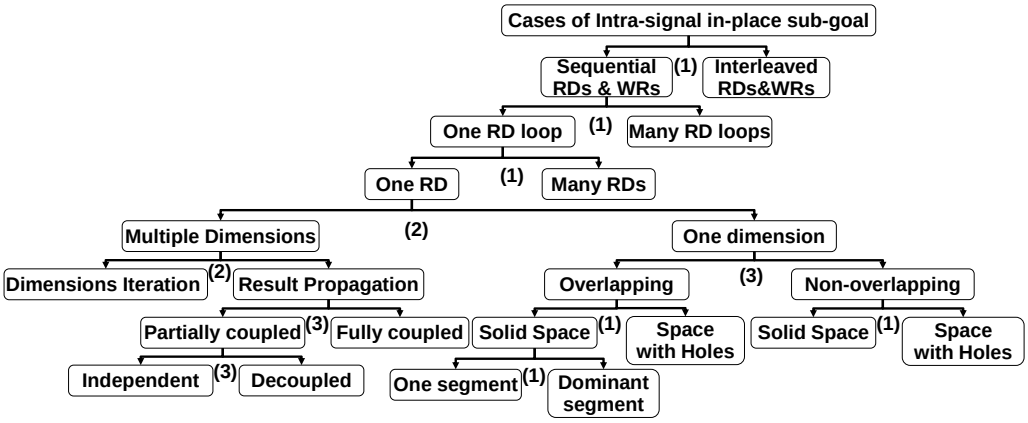
Σχ. Β.5: Το σύνολό των index expressions που προκύπτει μετά την εφαρμογή των αρχών της επαναχρησιμοποιούμενης DSE μεθοδολογίας.



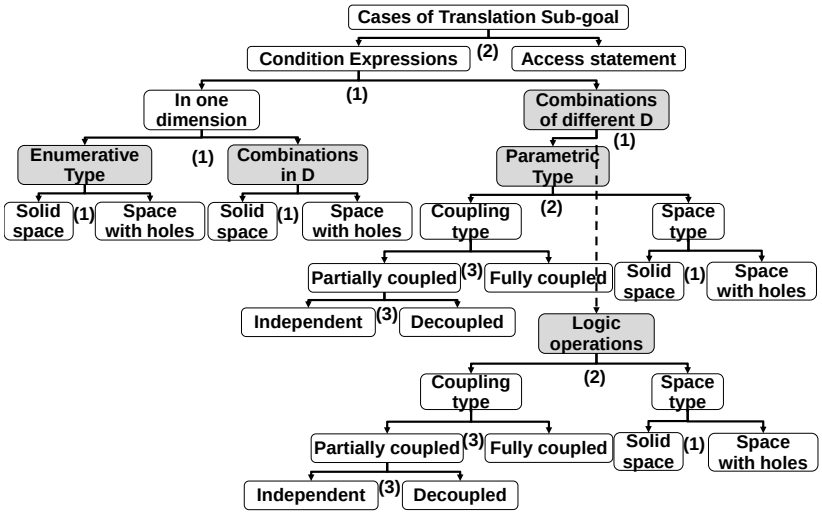
Σχ. Β.6: Οι υπο-στόχοι της intra-signal in-place μεθοδολογίας μετά την εφαρμογή των αρχών της επαναχρησιμοποιούμενης DSE μεθοδολογίας.

Οι αρχές της επαναχρησιμοποιούμενης μεθοδολογίας εφαρμόζονται για να βρεθούν όλες οι πιθανές περιπτώσεις στις εκφράσεις του συντελεστή του πίνακα (index expressions), δηλαδή στην εντολή προσπέλασης A[i] ο συντελεστής πίνακα είναι το i. Από τα πιθανά index expressions που εμφανίζονται στο Σχ. Β.5 επιλέγουμε την αντιπροσωπευτική περίπτωση με βάση την σχετικότητα στο υπο μελέτη πρόβλημα, την συχνότητα χρήσης και τον αριθμό των index expressions που μπορούν να εκφράσει μετά από μετασχηματισμούς. Επιπρόσθετα δείχνουμε πως μπορούμε να απεικονίσουμε τα εναπομείναντα index expressions στην αντιπροσωπευτική περίπτωση, δηλαδή ``iterator+constant''. Η μεθοδολογία αναπτύσσεται με διαχωρισμό των βημάτων σε υπο-στόχους οι οποίοι απεικονίζονται στο Σχ. Β.6.

Ανά υπο-στόχο προσδιορίζονται όλες οι πιθανές περιπτώσεις. Το αποτέλεσμα για τον υπο-στόχο εύρεσης του ελαχίστου απαιτούμενου μεγέθους μνήμης απεικονίζεται στο Σχ. Β.7. Επιλέγουμε δύο αντιπροσωπευτικές περιπτώσεις με βάση την θέση των εντολών ανάγνωσης και εγγραφής του πίνακα και για να αναπτύξουμε το βήμα για την εύρεση του ελάχιστου μεγέθους, το οποίο παρουσιάζεται στο κεφάλαιο 5. Το αποτέλεσμα για τον υποστόχο της μετάφρασης απεικονίζεται στο Σχ. Β.8. Το αποτέλεσμα για τον υπο-στόχο της ανάλυσης απεικονίζεται στο Σχ. Β.9.



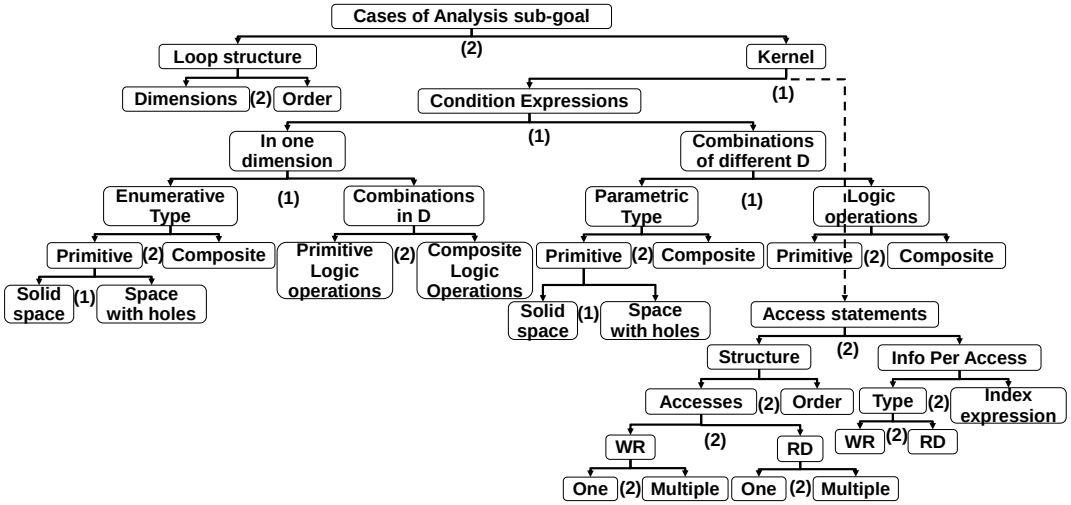
Σχ. Β.7: Το σύνολο των περιπτώσεων του intra-signal in-place υπο-στόχου μετά την εφαρμογή των αρχών της επαναχρησιμοποιούμενης DSE μεθοδολογίας.



Σχ. Β.8: Το σύνολο των περιπτώσεων του υπο-στόχου της μετάφρασης μετά την εφαρμογή των αρχών της επαναχρησιμοποιούμενης DSE μεθοδολογίας.

Β.7 Επεκτάσιμη και σχεδόν βέλτιστη αναπαράσταση των προσπελάσεων στη κύρια μνήμη βασισμένη σε μοτίβα

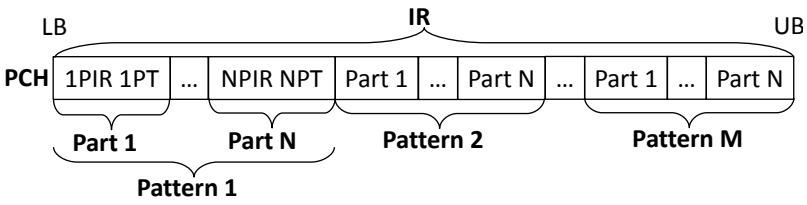
Οι τεχνικές για την διαχείριση της μνήμης και ειδικά για την εύρεση του απαιτούμενου μεγέθους παίρνουν ως είσοδο τις προσπελάσεις στην κύρια μνήμη. Οι προσπελάσεις στη κύρια μνήμη είναι ο ενεργός χώρος επαναλήψεων, ο οποίος καθορίζεται από την δομή της εφαρμογής, τους βρόχους, τις συνθήκες και τις εντολές προσπελάσεων στη κύρια μνήμη. Για παράδειγμα, οι εφαρμογές ενσωματωμένων συστημάτων, όπως η εικόνα, το βίντεο και η επεξεργασία σήματος, έχουν ως κυριαρχούμενα δεδομένα ομοιογενείς προσπελάσεις στους πίνακες μέσω βρόχων που περιέχουν συνθήκες. Οι συνθήκες καταστρέφουν την ομοιογένεια των προσπελάσεων και δημιουργούν κενά στον χώρο επαναλήψεων. Όταν αυξάνεται ο αριθμός των κενών λόγω των συνθηκών και αυξάνεται ο αριθμός εντολών προσπέλασης στην κύρια μνήμη, ο ενεργός χώρος επαναλήψεων γίνεται



Σχ. Β.9: Το σύνολο των περιπτώσεων του υπο-στόχου της ανάλυσης μετά την εφαρμογή των αρχών της επαναχρησιμοποιούμενης DSE μεθοδολογίας.

αρκετά περίπλοκος. Οι υπάρχουσες τεχνικές περιγράφουν τον ενεργό χώρο επαναλήψεων χρησιμοποιώντας καταμέτρηση των προσπελάσεων στην κύρια μνήμη, χρησιμοποιώντας συμβολική αναπαράσταση και χρησιμοποιώντας προσεγγίσεις, οι οποίες βασίζονται στην χειρότερη περίπτωση. Η αναπαράσταση μέσω της καταμέτρησης των προσπελάσεων στην κύρια μνήμη, π.χ. [135], είναι βέλτιστη. Όμως, η αναπαράσταση αυτή δεν είναι επεκτάσιμη, διότι ο χρόνος εξερεύνησης για την εύρεση του ελαχίστου μεγέθους μνήμης για την αποθήκευση των στοιχείων του πίνακα φτάνει μη αποδεκτές τιμές. Οι συμβολικές αναπαραστάσεις είναι επεκτάσιμες και σχεδόν βέλτιστες, αλλά καλύπτουν χώρους επαναλήψεων, οι οποίοι είτε είναι συμπαγής είτε αποτελούνται από κενά τοποθετημένα με ομοιογενή τρόπο, π.χ. [55]. Σε ανομοιογενείς χώρους επαναλήψεων προσεγγίζουν τις περιοχές όπου πραγματοποιούνται προσπελάσεις. Οι αναπαραστάσεις που βασίζονται σε προσεγγίσεις, π.χ. [160], χρησιμοποιούν την χείριστη περίπτωση και θεωρούν περιοχές του χώρου επαναλήψεων που δεν προσπελάσονται σαν πραγματικές προσπελάσεις με αποτέλεσμα υπερεκτίμηση του απαιτούμενου μεγέθους μνήμης. Επομένως, χρειάζεται μια σχεδόν βέλτιστη και ταυτόχρονα επεκτάσιμη αναπαράσταση των προσπελάσεων στην κύρια μνήμη, η οποία θα χρησιμοποιηθεί στο βήμα της μετάφρασης και θα υποστηρίζει την προτεινόμενη μεθοδολογία για την εύρεση του ελαχίστου απαιτούμενου χώρου μνήμης για την αποθήκευση των στοιχείων ενός πίνακα.

Στο κεφάλαιο 4 προτείνεται μια αναπαράσταση των προσπελάσεων στην κύρια μνήμη, η οποία παραμένει επεκτάσιμη και σχεδόν βέλτιστη σε περίπλοκους χώρους επαναλήψεων όπου περιέχουν πολλά ανομοιογενή κενά που δημιουργούνται από προσπελάσεις στην κύρια μνήμη μέσω βρόχων και συνθηκών. Για την επίτευξη μιας σχεδόν βέλτιστης και επεκτάσιμη αναπαράστασης των προσπελάσεων απαιτείται μια αναπαράσταση η οποία να μην καταμετρά τις προσπελάσεις και να μπορεί να περιγράφει έναν υψηλό αριθμό από τμήματα του χώρου επαναλήψεων. Τα τμήματα του χώρου επαναλήψεων κλαδεύονται και γίνονται ανενεργά από ένα συνήθως μικρο σύνολο από συνθήκες που υπάρχουν στον

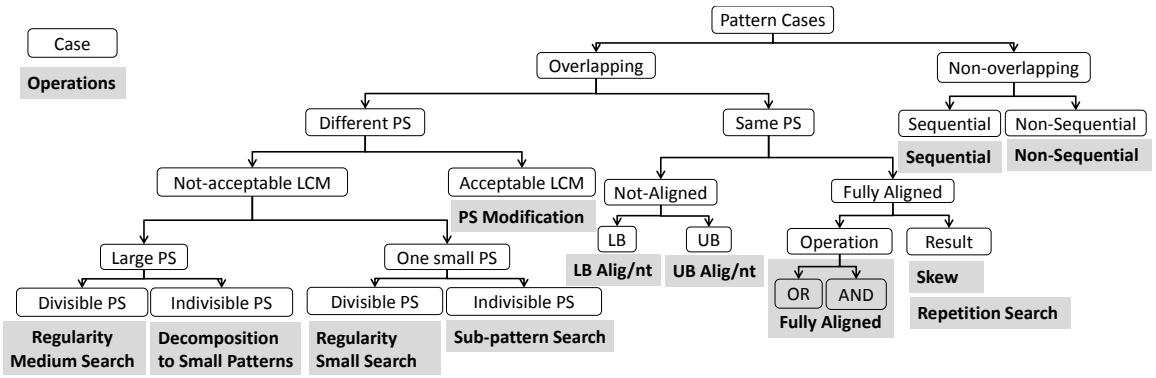


Σχ. Β.10: Μοτίβο που περιγράφει τον ενεργό χώρο επαναλήψεων.

κώδικα της εφαρμογής. Μια συνθήκη καταμέτρησης μπορεί να περιγράψει ένα τμήμα που προσπελάσεται ή όχι. Μια παραμετρική συνθήκη μπορεί να περιγράψει πολλαπλά τμήματα με ομοιογενή τρόπο. Όταν πολλαπλές συνθήκες συνυπάρχουν στο κώδικα, ο ενεργός χώρος επαναλήψεων περιγράφεται από των συνδυασμό των ενεργών χώρων επαναλήψεων που περιγράφεται από κάθε συνθήκη. Το αποτέλεσμα είναι πολλαπλά μη ενεργά τμήματα ανομοιογενώς τοποθετημένα στον χώρο επαναλήψεων που αυξάνουν την πολυπλοκότητα περιγραφής του χώρου επαναλήψεων.

Στην παρούσα διδακτορική διατριβή προτείνουμε τα μοτίβα για να επιτύχουμε μια επεκτάσιμη και σχεδόν βέλτιστη αναπαράσταση του ενεργού χώρου επαναλήψεων. Ένα μοτίβο περιγράφει τον ενεργό χώρο επαναλήψεων μιας συνθήκης ή εντολής προσπέλασης με συμπαγή και επαναλήψιμο τρόπο, αποφεύγοντας την καταμέτρηση των μεμονωμένων προσπελάσεων από την εντολή. Με την χρήση των κενών στην περιγραφή, ο ενεργός χώρος επαναλήψεων μπορεί να περιγραφεί με ομοιογενή και επαναλήψιμο τρόπο λόγω της δομής του βρόχου της εφαρμογής. Το ποσοστό των κενών που θεωρούνται ως προσπελάσεις ελέγχεται από το μέγεθός του κενού που θα αποφασιστεί να μην αναπαρασταθεί από το μοτίβο. Αυτό συμβαίνει για λόγους υψηλής ανομοιογένειας κατά την διευθυνσιοδότηση, δηλαδή κενά με μέγεθος ένα. Με αυτό τον τρόπο ο χώρος επαναλήψεων των παραμετρικών συνθηκών περιγράφεται με επεκτάσιμο και αποδοτικό τρόπο. Ένα μοτίβο ορίζεται σαν μια σειρά από δύο παραμέτρους: 1) τον αριθμό των συνεχόμενων τιμών στην μεταβλητή του βρόχου που η εντολή προσπέλασης ή η συνθήκη έχουν την ίδια συμπεριφορά και 2) την συμπεριφορά της εντολής προσπέλασης ή της συνθήκης, δηλαδή προσπέλαση (Access-A) ή όχι προσπέλαση (Hole-H). Με αυτόν τον τρόπο τα ανενεργά τμήματα του χώρου επαναλήψεων περιγράφονται αποφεύγοντας μη βέλτιστες προσεγγίσεις. Ένα μοτίβο περιγράφεται στο Σχ. Β.10, το οποίο αποτελείται από Ν τμήματα. Κάθε τμήμα έχει μέγεθος PIR και συμπεριφορά PT. Το μοτίβο είναι ενεργό από το χαμηλότερο όριο LB μέχρι το υψηλότερο όριο UB και επαναλαμβάνεται Μ φορές. Για παράδειγμα το μοτίβο {1H 1A} επαναλαμβάνεται 5 φορές και περιγράφει την συμπεριφορά μιας συνθήκης η οποία προσπελάσει μόνο στους περιττούς iterators από το 0 έως το 10.

Η υπο μελέτη εφαρμογή έχει διαφορετικές συνθήκες, δομή βρόχου και εντολές προσπελάσεων που οδηγούν σε διαφορετικά μοτίβα και συνδυασμούς μοτίβων. Ο ενεργός χώρος επαναλήψεων κάθε συνθήκης περιγράφεται από ένα μοτίβο. Για να περιγραφεί ο συνολικός ενεργός χώρος επαναλήψεων ανά εντολή προσπέλασης, τα μοτίβα που προκύπτουν από τις συνθήκες πρέπει να ενωθούν με επεκτάσιμο και σχεδόν βέλτιστο



Σχ. Β.11: Το σύνολο των πιθανών περιπτώσεων ένωσης μοτίβων και οι αντίστοιχες προτεινόμενες πράξεις.

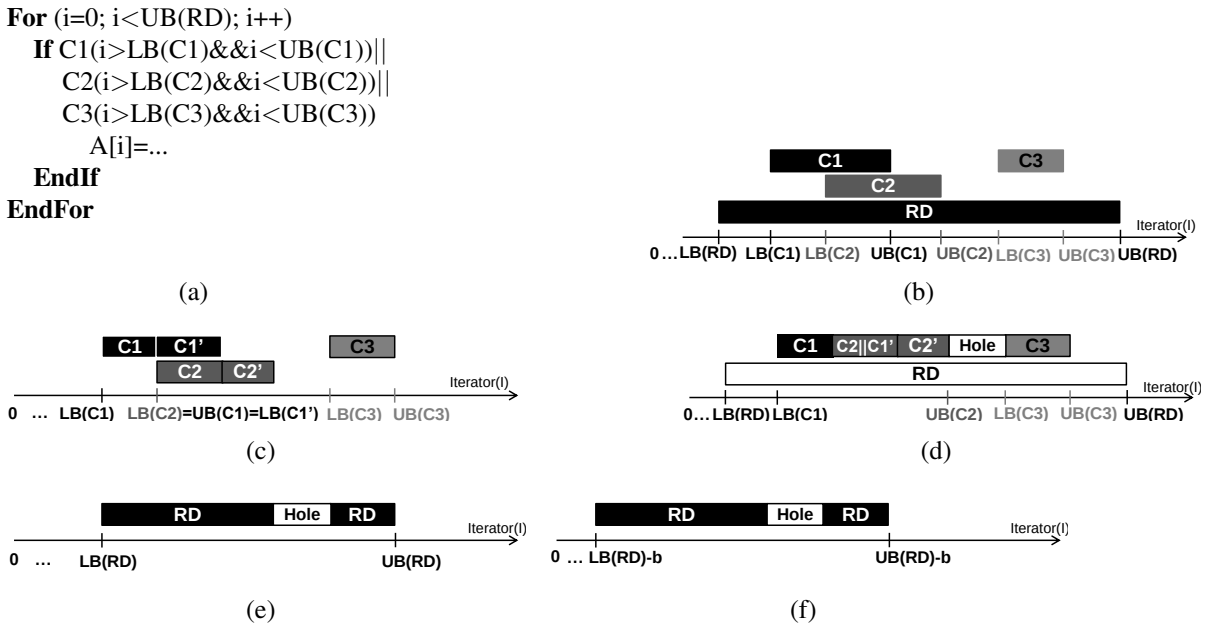
τρόπο. Για το σκοπό αυτό εισάγουμε ένα σύνολο που περιγράφει όλες τις πιθανές περιπτώσεις κάτω από τις οποίες μπορεί να πραγματοποιηθεί μια ένωση ανάμεσα σε μοτίβα. Το σύνολο των περιπτώσεων για την ένωση των μοτίβων προσδιορίζεται με βάση την δομή της εφαρμογής και χρησιμοποιώντας τις αρχές τις επαναχρησιμοποιούμενης DSE μεθοδολογίας του κεφαλαίου 2. Στην συνέχεια, προτείνουμε επεκτάσιμες και σχεδόν βέλτιστες πράξεις που ενεργούν επάνω στα μοτίβα για κάθε περίπτωση ένωσης μοτίβων. Το αποτέλεσμα περιγράφεται στο Σχ. Β.11, όπου στα λευκά κουτιά περιγράφονται οι διάφορες περιπτώσεις ένωσης μοτίβων και στα γκρι κουτιά οι αντίστοιχες προτεινόμενες πράξεις.

Σε πραγματικές περιπτώσεις, η διαδικασία ένωσης των μοτίβων γίνεται σε ένα πεπερασμένο μικρό σύνολο από μοτίβα, το οποίο προσδιορίζεται από το σύνολο των συνθηκών και των εντολών προσπέλασης. Στο κεφάλαιο 4 περιγράφεται αναλυτικά ο τρόπος που δημιουργούνται τα μοτίβα και πως εφαρμόζονται οι πράξεις για την δημιουργία ενός τελικού μοτίβου που περιγράφει την ανάγνωση, από όπου προκύπτει το τελικό απαιτούμενο μέγεθος.

Β.8 Επεκτάσιμη και σχεδόν βέλτιστη μεθοδολογία για το μέγεθος μνήμης για τα δεδομένα ενός πίνακα για μη επικαλυπτόμενες και επικαλυπτόμενες εντολές εγγραφής και ανάγνωσης.

Η αναπαράσταση βασισμένη σε μοτίβα χρησιμοποιείται στο βήμα της μετάφρασης και χρησιμοποιείται από το βήμα εύρεσης του ελαχίστου μεγέθους μνήμης για την αποθήκευση των δεδομένων ενός πίνακα στην μεθοδολογία που προτάθηκε στο κεφάλαιο 3 για τις περιπτώσεις μη επικαλυπτόμενων και επικαλυπτόμενων εντολών εγγραφής και

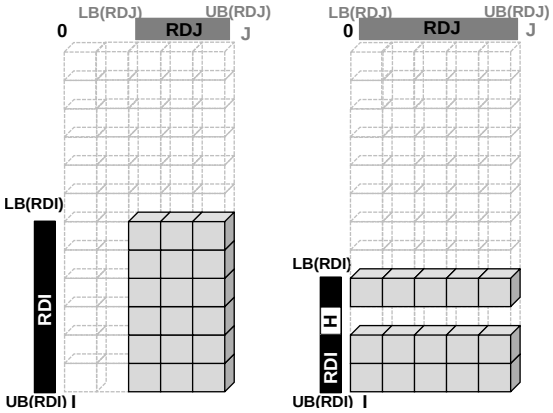
ανάγνωσης του πίνακα. Το πρώτο βήμα της προτεινόμενης μεθοδολογίας είναι η ανάλυση, που απεικονίζει την υπο μελέτη εφαρμογή σε ένα παραμετρικό πλαίσιο και βρίσκει την πληροφορία για τις προσπελάσεις του πίνακα. Το επόμενο βήμα είναι το βήμα της μετάφρασης, που μεταφράζει την πληροφορία σε μια αναπαράσταση χρησιμοποιώντας μοτίβα, η οποία υποστηρίζει επεκτάσιμη και σχεδόν βέλτιστη εύρεση του μεγέθους μνήμης. Τα μοτίβα διαδίδονται στο επόμενο βήμα που βρίσκει το μέγεθος της απαιτούμενης μνήμης. Η προτεινόμενη μεθοδολογία εφαρμόζεται με συστηματικό τρόπο: η ανάλυση της υπο μελέτη εφαρμογής δίνει συγκεκριμένες τιμές στις παραμέτρους του πλαισίου του πρώτου βήματος. Η διάδοση των τιμών αυτών σαν constraints στα επόμενα βήματα επιλέγει την ενεργές περιπτώσεις. Οι λύσεις των ενεργών περιπτώσεων εφαρμόζονται και οι παράμετροι των βημάτων παίρνουν συγκεκριμένες τιμές.



Σχ. Β.12: Σχηματική περιγραφή του βήματος της μετάφρασης για μία διάσταση.

Μερικές από τις περιπτώσεις των βημάτων της μετάφρασης και της εύρεσης του μεγέθους παρουσιάζονται στα παρακάτω σχήματα, ενώ το σύνολό των βημάτων παρουσιάζεται στο κεφάλαιο 5.

Στο Σχ. Β.12 παρουσιάζεται η σχηματική περιγραφή του βήματος της μετάφρασης για μία διάσταση. Στο (a) απεικονίζονται τα αρχικά μοτίβα από τις αρχικές συνθήκες (C1, C2, C3) και μοτίβο της εντολής ανάγνωσης (RD). Στο (b) απεικονίζονται τα νέα μοτίβα μετά την εφαρμογή των πράξεων ευθυγράμμισης του κάτω και του άνω ορίου, ενώ στο (c) παρουσιάζεται το συνδυασμένο μοτίβο μετά την εφαρμογή των πράξεων OR, συνεχόμενα μη επικαλυπτόμενα μοτίβα και μη συνεχόμενα μη επικαλυπτόμενα μοτίβα. Το τελικό μοτίβο για την ανάγνωση μετά την εφαρμογή της πράξης AND ανάμεσα στο μοτίβο ανάγνωσης και στο συνδυασμένο μοτίβο των συνθηκών απεικονίζεται στο (d). Τέλος, το (e) παρουσιάζει το ολισθημένο μοτίβο των στοιχείων που προσπελάσονται λόγω του δείκτη



(a) (b)

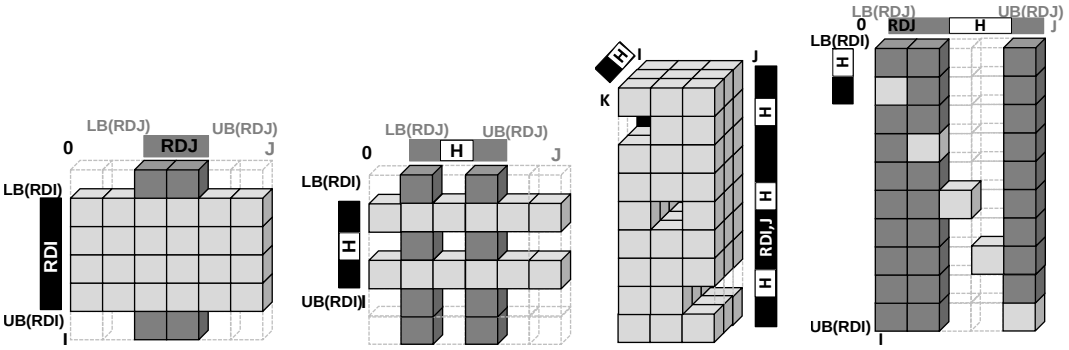
```
For (I=6;I<12;I++)
  For (J=0;J<5;J++)
    If (J>1)
      ...=A[I-5][J-2]
    EndIf
    If (J>2)
      ...=A[I-5][J-2]
    EndIf
  EndFor
EndFor
```

(c)

```
For (I=6;I<12;I++)
  For (J=0;J<5;J++)
    If (I==8)||(I>9)
      ...=A[I-5][J]
    EndIf
  EndFor
EndFor
```

(d)

Σχ. Β.13: Σχηματική αναπαράσταση των μοτίβων ανάγνωσης για τις διαστάσεις I και J, οι οποίες είναι ανεξάρτητες.



(a) (b) (c) (d)

```
For (I=0;I<6;I++)
 For (J=0;J<6;J++)
  If (0<I<5)||(1<J<4)
   ...=A[I][J]
  EndIf
 EndFor
EndFor
```

(e)

```
For (I=0;I<6;I++)
 For (J=0;J<6;J++)
  If (I==1)||(I==3)||
     (J==1)||(J==3)
   ...=A[I][J]
  EndIf
 EndFor
EndFor
```

(f)

```
For (I=0;I<9;I++)
 For (J=0;J<3;J++)
  For (K=0;K<3;K++)
   If (I≠3J+1)||(I==2K)
    ...=A[I][J][K]
   EndIf
  EndFor
 EndFor
EndFor
```

(g)

```
For (I=0;I<10;I++)
 For (J=0;J<5;J++)
  If (I==2J+1)||
     (J<2)||(J>3)
   ...=A[I][J]
  EndIf
 EndFor
EndFor
```

(h)

Σχ. Β.14: Σχηματική αναπαράσταση για τα τελικά μοτίβα ανάγνωσης για 2 decoupled διαστάσεις.

του πίνακα i+b.

Στο Σχ. Β.13 παρουσιάζεται η σχηματική αναπαράσταση των μοτίβων ανάγνωσης για τις διαστάσεις I και J, οι οποίες είναι ανεξάρτητες. Το μέγεθος μνήμης που απαιτείται για την αποθήκευση των στοιχείων προκύπτει από το άθροισμα των γκρι περιοχών για: (a) ενιαίους χώρους επαναλήψεων (Solid Iteration Space - SIS) και (b) για χώρους επαναλήψεων

Πίνακας Β.1: Αποτελέσματα για την περίπτωση με κυρίαρχο τμήμα στην εξωτερική διάσταση για επικαλυπτόμενες εντολές γραφής και ανάγνωσης.

Εφαρμογή: Ρουτίνα: Πίνακας (Αρχ. Όρια)	Λύση	Παράγοντας	Προτεινόμενη Μεθ.		Μεθ. Καταμέτρησης	
			Μέγεθος (στοιχεία)	Χρόνος (ms)	Μέγεθος (στοιχεία)	Χρόνος (ms)
Atax*: A (32)(32)	Eq. 5.31 Sec. 5.4.5.1.1	1	32	0.101	32	14.866
		4	128	0.103	128	816.279
		8	512	0.104	512	52,059.418
		12	2,048	0.101	-	-
Reg_detect*: path (20)(20)	Eq. 5.35 Sec. 5.4.5.1.2	1	21	0.104	21	4.107
		2	61	0.102	61	89.263
		3	201	0.104	201	3,070.95
		4	601	0.102	601	86,142.682
		5	2001	0.100	-	-
Gsm**: Update_residual_ signal P3: drp (100)(3)(40)	Table 5.6: case i Sec. 5.4.5.2.1	1	80	0.100	80	153.295
		2	160	0.102	160	608.821
		3	240	0.104	240	1,341.206
		4	480	0.103	480	5,376.844
		5	920	0.104	920	19,166.318
Jacobi-2D*: A (128)(32)	Eq. 5.37 Sec. 5.4.5.2.2	1	34	0.177	34	31.102
		2	66	0.181	66	113.017
		3	1,002	0.190	1,002	23,439.816
		4	8,002	0.175	8,002	1,577,880.183
Doitgen*: sum (10)(10)(10)	Eq. 5.28 Sec. 5.4.4	1	111	0.108	111	41.545
		2	273	0.104	273	452.183
		3	813	0.103	813	6,585.703
		4	1,057	0.103	1,057	12,954.013

(*) Polybench, (**) MediaBench, (-) Memory Overflow

με κενά (Iteration Space with Holes - ISH) και συνθήκες καταμέτρησης (Enumerative Conditions with Holes - ECH) στην I διάσταση και οι κώδικες για (c) SIS και (d) ISH με συνθήκες καταμέτρησης στην I διάσταση.

Στο Σχ. B.14 παρουσιάζεται η σχηματική αναπαράσταση για τα τελικά μοτίβα ανάγνωσης για 2 decoupled διαστάσεις. Το τελικό μέγεθός προκύπτει από το άθροισμα των γκρι περιοχών: (a) SIS, (b) ISH με ECH που συνδέονται μέσω OR πράξης, (c) ISH με παραμετρικές συνθήκες με κενά (Parametric Condition with Holes - PCH) και (d) ISH με PCH και ECH συνδυασμένα με OR πράξη. Αντίστοιχοι κώδικές παρουσιάζονται για: (e) SIS, (f) ISH με ECH, (g) ISH με PCH and (h) ISH με PCH και ECH.

Στην παρούσα διδακτορική διατριβή παρουσιάζουμε τον τρόπο με τον οποίο εφαρμόζεται η προτεινόμενη μεθοδολογία και συγκρίνουμε την απόδοσή της για διάφορες εφαρμογές από τα PolyBench, MediaBench και Mibench benchmark suites (Πίνακες B.8, B.8, B.3, B.4, B.5).

Στο Σχ. B.19 παρουσιάζεται η σύγκριση του χρόνου εξερεύνησης για την εύρεση του ελαχίστου απαιτούμενου μεγέθους μνήμης για μη επικαλυπτόμενες εντολές γραφής και ανάγνωσης, όταν ο αριθμός των προσπελάσεων αυξάνεται κατά: i) ένα παράγοντα για τα όρια του βρόχου για την εφαρμογή pgp-outdec (a) και την εφαρμογή blowfish-decode/encode (c) και ii) ένα παράγοντα για τον αριθμό των μοτίβων που υπάρχουν σε έναν πυρήνα της εφαρμογής για την εφαρμογή pgp-outdec benchmark (b) και την εφαρμογή blowfish-decode/encode (d). Στο σχήμα (e) παρουσιάζεται ο χρόνος εξερεύνησης για την εύρεση

Πίνακας Β.2: Αποτελέσματα για την περίπτωση χωρίς κυρίαρχο τμήμα στην εξωτερική διάστασή για επικαλυπτόμενες εντολές γραφής και ανάγνωσης.

Εφαρμογή: Ρουτίνα: Πίνακας (Αρχ. Όρια)	Λύση	Παράγοντας	Προτεινόμενη Μεθοδολογία		Μεθ. Καταμέτρησης		Προσέγγιση
			Μέγεθος (στοιχεία)	Χρόνος (ms)	Μέγεθος (στοιχεία)	Χρόνος (ms)	Μέγεθος (στοιχεία)
Jpeg**: idct_2x2: wsptr (32)(256)	Table 5.7: 5.4.6.1: case i Sec. 5.4.6.1.1	1	84	0.901	84	131.853	132
		2	164	0.896	164	2,558.200	260
		3	644	0.902	644	44,243.614	1,028
		4	1,924	0.899	1,924	5,842,417.563	3,076
Jpeg**: decompress_ smooth_data: coef_bits (4)(228)	Eq. 5.41 Sect. 5.4.6.1.2	1	5	0.158	5	18.782	6
		2	10	0.175	10	65.887	12
		3	15	0.183	15	138.009	18
		4	20	0.219	20	234.046	24
		5	25	0.219	25	365.003	30
Jpeg**: idct_2x2: wsptr (un) (32)(256)	Table 5.7: 5.4.6.2: case i Sec. 5.4.6.2.1	1	80	0.798	80	113.259	132
		2	160	0.793	160	2,542.592	260
		3	640	0.795	640	44,142.714	1,028
		4	1,920	0.792	1,920	5,843,373.740	3,076
Gauss-Seidel: A (32)(32)	Eq. 5.44 Sec. 5.4.6.2.2	1	31	0.161	31		32
		2	499	0.162	499	47,432.427	500
		3	999	0.164	999	397,211.781	1,000
		4	1,999	0.163	1,999	3,066,944.504	2,000

(**) MediaBench

του ελαχίστου απαιτούμενου μεγέθους μνήμης για επικαλυπτόμενες εντολές γραφής και ανάγνωσης για τα Benchmarks (a) Jpeg**: xbuf1 benchmark and (b) Pegwit**: roundKeys_e benchmark.

Πίνακας Β.3: Αποτελέσματα για τα MediaBench για μη επικαλυπτόμενες εντολές ανάγνωσης και γραφής. Το σύμβολο '-' χρησιμοποιείται όταν δημιουργείται Memory Error κατά την διάρκεια των προσομοιώσεων.

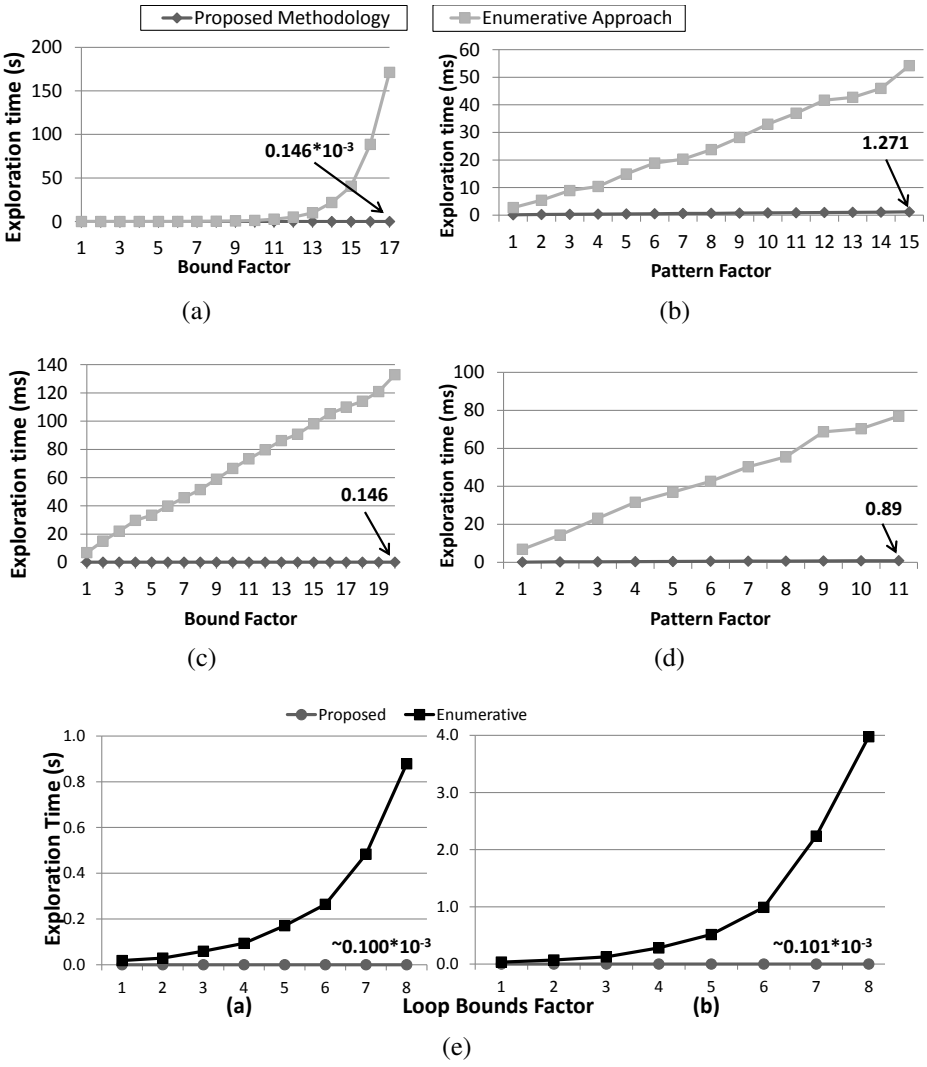
Εφαρμογή: Πίνακας (Αρχ. Όρια)	Παράγοντας Αύξησης	Προτεινόμενη μεθ.		Μεθ. Καταμέτρησης		Προσέγγιση
		Μέγεθος Μνήμης	Χρόνος Ευρ. (ms)	Μέγεθος Μνήμης	Χρόνος Εύρ. (ms)	Μέγεθος Μνήμης
Jpeg-Decode&Quant.: DCTblock (256)	1	7,680	0.148	7,680	34.667	10,239
	3	23,040	0.147	23,040	103.736	30,719
	5	38,400	0.149	38,400	162.533	51,199
	7	53,760	0.145	53,760	228.764	71,679
	9	69,120	0.147	69,120	281.893	92,159
	11	84,480	0.145	84,480	350.931	112,639
	13	99,840	0.145	99,840	413.799	133,119
Jpeg-Decode&Quant.: DCTblock (512)	1	15,360	0.146	15,360	65.987	20,476
	2	30,720	0.150	30,720	128.877	40,959
	3	46,080	0.147	46,080	212.997	61,439
	4	61,440	0.146	61,440	264.078	81,919
	5	76,800	0.149	76,800	318.667	102,399
	6	92,160	0.149	92,160	389.390	122,879
Epic: image (240)	1	4,257	1.313	4,257	101.116	11,520
	2	17,265	1.302	17,265	1,067.983	138,144
	3	34,609	1.301	34,609	3,888.289	522,016
	4	69,297	1.337	69,297	16,329.751	2,027,040
	5	138,673	1.367	138,673	64,824.346	7,986,208
	6	277,425	1.318	277,425	249,110.047	31,701,024
Mpeg: curr_frame (64x32)	1	34,833	1.801	34,833	852.955	65,537
	2	69,649	1.840	69,649	1,703.624	131,073
	3	139,281	1.806	139,281	3,317.909	262,145
	4	278,545	1.813	278,545	7,167.751	524,289
	5	557,073	1.818	557,073	30,241.290	2,097,185
Mpeg: curr_frame (128x32)	1	69,649	1.840	69,649	1,703.624	131,073
	2	139,281	1.806	139,281	3,317.909	262,145
	3	278,545	1.813	278,545	7,167.751	524,289
	4	557,073	1.818	557,073	30,241.290	2,097,185
Motion estimation-full pel: p1/p2 (32)	1	3,576	0.147	3,576	51.922	12,759
	2	11,512	0.146	11,512	232.104	65,991
	3	39,672	0.147	39,672	1,399.778	389,031
	4	145,144	0.147	145,144	9,093.884	2,589,543
	5	552,696	0.146	552,696	66,325.246	18,713,319
	6	2,154,232	0.147	-	-	141,892,071
Motion estimation-full pel: p1/p2 (80)	1	39,672	0.147	39,672	1,399.778	389,031
	2	145,144	0.147	145,144	9,093.884	2,589,543
	3	552,696	0.146	552,696	66,325.246	18,713,319
	4	2,154,232	0.147	-	-	141,892,071
Motion estimation-half pel: p1/p2 (272)	1	40,800	0.149	40,800	2,181.419	657,152
	2	79,200	0.151	79,200	8,034	2,492,160
	3	156,000	0.149	156,000	34,129.439	9,701,120
	4	309,600	0.149	309,600	133,794.569	38,274,816
	5	616,800	0.148	-	-	152,045,312

Πίνακας Β.4: Αποτελέσματα για τα MediaBench για μη επικαλυπτόμενες εντολές εγγραφής και ανάγνωσης. Το σύμβολο '-' χρησιμοποιείται όταν δημιουργείται Memory Error κατά την διάρκεια των προσομοιώσεων.

Εφαρμογή: Πίνακας (Αρχ. Όρια)	Παράγοντας Αύξησης	Προτεινόμενη μεθ.		Μεθ. Καταμέτρησης		Προσέγγιση
		Μέγεθος Μνήμης	Χρόνος Ευρ. (ms)	Μέγεθος Μνήμης	Χρόνος Εύρ. (ms)	Μέγεθος Μνήμης
Motion estimation-half pel: p1/p2 (48)	1	7,200	0.151	7,200	69.523	19,231
	2	12,000	0.153	12,000	193.411	55,071
	3	21,600	0.155	21,600	615.794	182,047
	4	40,800	0.149	40,800	2,181.419	657,152
	5	79,200	0.151	79,200	8,034	2,492,160
	6	156,000	0.149	156,000	34,129.439	9,701,120
	7	309,600	0.149	309,600	133,794.569	38,274,816
	8	616,800	0.148	-	-	152,045,312
Motion estimation-half pel: p1a (272)	1	27,200	0.157	27,200	2,239.512	657,183
	2	52,800	0.154	52,800	8,325.536	2,492,191
	3	104,000	0.159	104,000	36,410.136	9,701,151
	4	206,400	0.156	206,400	129,134.210	38,274,847
	5	411,200	0.165	-	-	152,045,343
Motion estimation-half pel: p1a (48)	1	4,800	0.155	4,800	93.294	19,231
	2	8,000	0.158	8,000	192.457	55,071
	3	14,400	0.156	14,400	640.237	182,047
	4	27,200	0.157	27,200	2,239.512	657,183
	5	52,800	0.154	52,800	8,325.536	2,492,191
	6	104,000	0.159	104,000	36,410.136	9,701,151
	7	206,400	0.156	206,400	129,134.210	38,274,847
	8	411,200	0.165	-	-	152,045,343
Pgp-outdec: p (48)	1	2,048	0.143	3,072	11.911	3,074
	3	8,192	0.148	12,288	43.058	12,290
	6	131,072	0.149	196,608	683.534	196,610
	8	524,288	0.155	786,432	2,732.710	786,434
	11	4,194,304	0.146	6,291,456	21,838.798	6,291,458
	13	16,777,216	0.153	15,165,824	88,559.484	25,165,826
	16	134,217,728	0.150	-	-	201,326,594
Pgp-outdec: p (262,144)	1	4,194,304	0.146	4,194,304	21,838.798	6,291,458
	2	8,388,608	0.147	8,388,608	40,522.5	12,582,914
	3	16,777,216	0.153	16,777,216	88,559.484	25,165,826
	4	33,554,432	0.147	33,554,432	171,213.572	50,331,650
	5	67,108,864	0.150	-	-	100,663,298
Mesa-light: frontcolor & backcolor (10)	1	5	0.147	5	0.131	10
	10	50	0.145	50	0.410	10
	10^2	500	0.146	500	3.290	100
	10^3	5 K	0.149	5 K	34.657	1 K
	10^4	50 K	0.147	50 K	375.118	10 K
	10^5	500 K	0.152	500 K	3,492.222	100 K
Mesa-light: frontcolor & backcolor (10^2)	1	50	0.147	50	0.410	10^2
	10^4	$5*10^3$	0.149	$5*10^3$	34.657	10^4
	10^6	$5*10^5$	0.152	$5*10^5$	3,492.222	10^6
	10^8	$5*10^7$	0.151	$5*10^7$	338,464.414	10^9

Πίνακας Β.5: Αποτελέσματα για τα PolyBench(*1), MiBench(*2) και την εφαρμογή που χρησιμοποιήθηκε για την επίδειξη του βήματος της μετάφρασης για μη επικαλυπτόμενες εντολές γραφής και ανάγνωσης. Το σύμβολο '-' χρησιμοποιείται όταν δημιουργείται Memory Error κατά την διάρκεια των προσομοιώσεων.

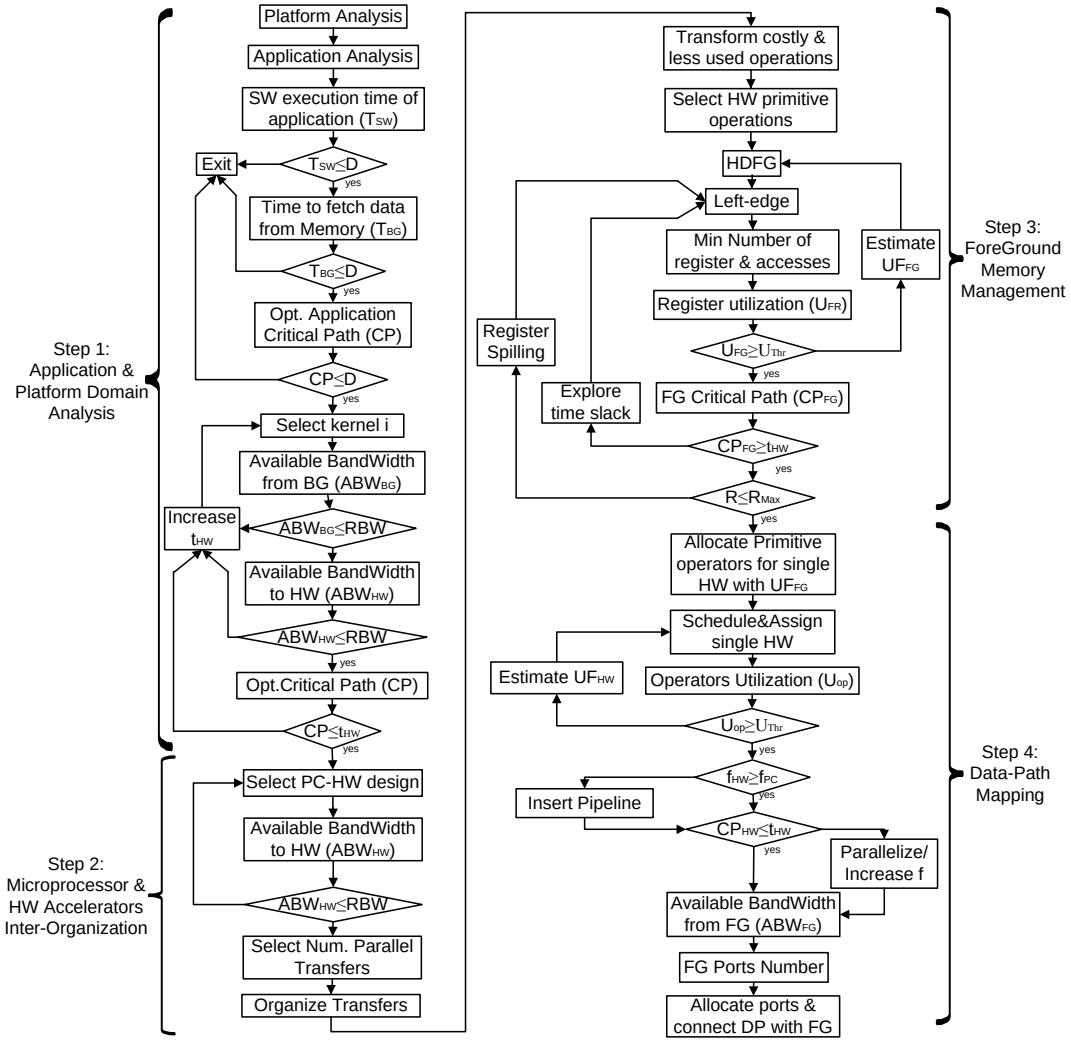
Εφαρμογή: Πίνακας (Αρχ. Όρια)	Παράγοντας Αύξησης	Προτεινόμενη μεθ.		Μεθ. Καταμέτρησης		Προσέγγιση
		Μέγεθος Μνήμης	Χρόνος Ευρ. (ms)	Μέγεθος Μνήμης	Χρόνος Εύρ. (ms)	Μέγεθος Μνήμης
Correlation: data (32) *1	1	0.25 M	0.343	0.25 M	2,886.159	0.25 M
	2	1 M	0.340	1 M	11,732.231	1 M
	3	4 M	0.337	4 M	48,614.943	4 M
	4	16 M	0.347	16 M	188,547.256	16 M
	5	64 M	0.333	64 M	780,152.034	64 M
	6	256 M	0.345	-	-	256 M
Jacobi-1D: A (500) *1	1	1,004	0.674	1,004	5.428	1,004
	2	2,004	0.660	2,004	9.998	2,004
	3	4,004	0.676	4,004	21.646	4,004
	4	8,004	0.664	8,004	40.701	8,004
	5	16,004	0.656	16,004	78.056	16,004
	6	32,004	0.658	32,004	159.161	32,004
Blowfish Decode/Encode: p (2,048) *2	1	256	0.146	256	6.899	2,040
	4	1,024	0.147	1,024	29.753	8,184
	7	1,792	0.149	1,792	45.805	14,328
	10	2,560	0.148	2,560	66.517	20,472
	13	3,328	0.148	3,328	86.228	26,616
	16	4,096	0.147	4,096	105.264	32,760
	19	4,864	0.149	4,864	120.938	38,904
Blowfish Decode/Encode: p (4,096) *2	1	512	0.146	512	15.065	4,088
	2	1,024	0.147	1,024	29.753	8,184
	3	1,536	0.148	1,536	39.888	12,288
	4	2,048	0.148	2,048	51.536	16,376
	5	2,560	0.148	2,560	66.517	20,472
	6	3,072	0.148	3,072	79.722	24,568
	7	3,584	0.149	3,584	90.756	28,664
	8	4,096	0.147	4,096	105.264	32,760
	9	4,608	0.147	4,608	114.122	36,856
	10	5,120	0.146	5,120	132.905	40,952
Εφαρμογή μετάφρασης: A (64)	1	32	0.799	32	0.487	507
	3	128	0.804	128	1.694	2,043
	5	512	0.798	512	6.308	8,187
	7	2,048	0.810	2,048	25.776	32,763
	9	8,192	0.793	8,192	97.852	131,067
	11	32,768	0.796	32,768	395.624	524,283
	13	131,072	0.793	131,072	1,629.237	2,097,147



Σχ. Β.15: Σύγκριση του χρόνου εξερεύνησης για την εύρεση του ελαχίστου απαιτούμενου μεγέθους μνήμης για μη επικαλυπτόμενες εντολές γραφής και ανάγνωσης (a)-(d) και για επικαλυπτόμενες εντολές γραφής και ανάγνωσης (ε).

Β.9 Μεθοδολογία για τη απεικόνιση σε πλατφόρμα με έναν επεξεργαστή ελεγχόμενο από εντολών και ποικίλους συνεπεξεργαστές.

Τα ενσωματωμένα συστήματα έχουν, συνήθως, αυστηρούς χρονικούς περιορισμούς, οι οποίοι απαιτούν ειδικές σχεδιάσεις στο υλικό για να μπορέσουν να ικανοποιηθούν. Οι ειδικές σχεδιάσεις στο υλικό αυξάνουν την απόδοση, αλλά έχουν μεγάλο κόστος στον σχεδιασμό και πολύ περιορισμένη ευελιξία, ακόμα και όταν μπορούν να τροποποιηθούν μερικώς. Οι σχεδιάσεις σε λογισμικό αυξάνουν την ευελιξία για μεγάλο εύρος εφαρμογών, αλλά στο κόστος της μειωμένης απόδοσης. Επομένως, μια υβριδική σχεδίαση σε υλικό και λογισμικό είναι μια υποσχόμενη λύση, η οποία ισορροπεί την ευελιξία των σχεδιάσεων σε



Σχ. Β.16: Η ροή και τα βήματα της προτεινόμενης μεθοδολογίας.

λογισμικό και την απόδοση των σχεδιάσεων σε υλικό [92].

Υπάρχοντα εργαλεία για σχεδιάσεις προσφέρουν μια μερικώς αυτοματοποιημένη διαδικασία για την δημιουργία ειδικών τμημάτων στους μικροεπεξεργαστές. Τα εργαλεία αυτά απαιτούν μεγάλο χρόνο εξερεύνησης, διότι αναλύουν μεγάλο αριθμό σχεδιασμών, οι οποίοι, ωστόσο, επικεντρώνονται σε περιορισμένη περιοχή του χώρου λύσεων. Όταν τα χαρακτηριστικά της εφαρμογής δεν ταιριάζουν με αυτά της περιοχής που εξερευνά το εργαλείο, το εργαλείο οδηγεί σε μη βέλτιστα αποτελέσματα. Μια ευρεία εξερεύνηση του χώρου λύσεων που να καλύπτει όλους τους σχεδιασμούς είναι μια πολύ δύσκολη και χρονοβόρα διαδικασία λόγου του υψηλού αριθμού σχεδιαστικών παραμέτρων τόσο στο υλικό όσο και στο λογισμικό. Επομένως, οι σχεδιαστές δημιουργούν σχεδιασμούς στο υλικό και το λογισμικό ακολουθώντας τυχαίες ή επαναληπτικές διαδικασίες που βασίζονται στην εμπειρία τους [131]. Επομένως, απαιτείται μια συστηματική μεθοδολογία που θα παρέχει εξερεύνηση του χώρου λύσεων με επεκτάσιμο τρόπο και θα οδηγεί σε σχεδόν βέλτιστους σχεδιασμούς [176].

Στην παρούσα διδακτορική διατριβή προτείνεται μια επεκτάσιμη μεθοδολογία

εξερεύνησης των σχεδιασμών (DSE) που απεικονίζει μια περιοχή εφαρμογών σε ένα SW/HW FPGA σχεδιασμό με ένα πυρήνα μικροεπεξεργαστή και ποικίλους HW συνεπεξεργαστές. Η προτεινόμενη μεθοδολογία διαχωρίζει την διαδικασία σχεδίασης σε συνεχόμενα βήματα απεικόνισης τα οποία συνδέονται με διάδοση περιορισμών ακολουθώντας μονοκατευθυντήρια κατεύθυνση. Με αυτόν τον τρόπο αποφεύγονται τα μη επεκτάσιμα βήματα απεικόνισης και οι επαναλήψεις στην διαδικασία της σχεδίασης. Η ροή και τα βήματα της προτεινόμενης μεθοδολογίας απεικονίζονται στο Σχ. B.16

ALGORITHM 15: Βήμα Ανάλυσης.

Step 1.1: Platform Analysis\
Determine HW parameters();\
Step 1.2: Application Analysis\
Identify basic kernels;\
for *(i=0;i<NumOfKernels;i++)* **do**\
  Profile t_i;\
  Identify SW parameters();\
Kernels=Sort based on t_i;\
Identify control flow;\
Step 1.3: Decide SW & HW execution\
Step 1.3.1: Application Constraints\
$f_{PC} = max(Av_{f_{PC}})$;\
t_{SW}=Assign(Application, PC);\
t_{tot}=t_{SW};\
if *(t_{tot} <D)* **then**\
  Exit;\
High level estimate($t_{TR,Opt,CDFG}$);\
if *($t_{TR,Opt,CDFG} > t_{HW}$)* **then**\
  Exit(Change HW parameters);\
High level estimate($CP_{HW,Opt,CDFG}$);\
if *($CP_{HW,Opt,CDFG} > t_{HW}$)* **then**\
  Exit(Change HW parameters);\
High level estimate($t_{TR,Opt,CDFG}$);\
if *($t_{TR,Opt,CDFG} > t_{HW}$)* **then**\
  Exit(Change HW parameters);\
Step 1.3.2: Kernel Constraints\
while *($t_{tot} > D$)* **do**\
  k=select i from Kernels;\
  t_{SW}=t_{tot}-t_i;\
  t_{HW}=(1-slack)*(D-t_{SW});\
  Assign(k,HW);\
  Compute $BandW$, $ABandW_{BG,Opt}$;\
  if *($BandW > ABandW_{BG}$)* **then**\
    Repeat=1; break;\
  Compute $ABandW_{PC-HW,Opt}$;\
  if *($BandW > ABandW_{PC-HW,Opt}$)* **then**\
    Repeat=1; break;\
  High level estimate($CP_{HW,Opt}$);\
  if *($CP_{HW,Opt} > t_{HW}$)* **then**\
    Repeat=1; break;\
  if *(Repeat==1 && SWsolution==1)* **then**\
    t_{tot}=t_{SW}+t_{HW};\
  else\
    Exit(Change HW parameters);

ALGORITHM 16: Βήμα οργάνωσης Microprocessor & HW Accelerators.

Dependent=1;CoProcessor=1;Control=0;\
Compute $ABandW_{PC-HW}$;\
if *($BandW > ABandW_{PC-HW}$)* **then**\
  Dependent=0;NPI=0;\
  Compute $ABandW_{PC-HW_{PLB}}$;\
  if *($t_{TR,PC-HW_{PLB}} > t_{HW}$)* **then**\
    NPI=1;\
    Explore(SCIP-NPI);\
High level estimate of $t_{TR,PC-HW}$;\
if *($t_{TR,PC-HW} << CP_{HW,OPt}$)* **then**\
  $t_{HW} = t_{HW} - t_{TR,PC-HW}$\
Compute Bus_{PC-HW};

Τα βήματα της προτεινόμενης μεθοδολογίας είναι: η ανάλυση της εφαρμογής και της πλατφόρμας (Σχ. 15), η οργάνωση μεταξύ μικροεπεξεργαστή και συνεπεξεργαστών

ALGORITHM 17: Βήμα Διαχείρισης Foreground Μνήμης.

$OPs = \sum_{n=0}^{OPsType_{\text{Arithm}}} OPs_{\text{Arithm}}(n)$
if *(Coprocessor==0)* **then**
    $Size_{\text{FG}} = Size_{\text{RegFile}}$; Exit;
if *(Dependent==0)||(Dependent==1 && Coprocessor==0 && Control=1)* **then**
    $OPs = OPs + \sum_{n=0}^{OPsType_{\text{Cntr}}} OPs_{\text{Cntr}}(n)$
Evaluate(OPs);
Select primitive operations;
UF_{FG}=1;
Evaluate Reg(UF_{FG});
Compute U_{FG};
Num_{Reg}=FG;
Exploration(UF_{FG}, ∞);
if *($Num_{\text{Reg}} < Num_{\text{Reg,Max}}$)* **then**
    Num_{Reg}=FG;
    Exploration($UF_{\text{FG}}, Num_{\text{Reg,Max}}$);
$Accesses_{\text{FG}}$=Count(RD,WR,MLT);
FG Exploration(U_{FG},Max){
while *($U_{\text{FG}} < U_{\text{Thr}}$)* **do**
    Update UF_{FG}; Evaluate Reg(UF_{FG},Max);
    Compute U_{R};
while *($CP_{\text{FG}} < t_{\text{HW}}$)* **do**
    Reduce Num_{Reg};
    Evaluate Reg(UF_{FG},Num_{Reg});
}
Evaluate(OPsType){
for *(j=0;j<OPsType;j++)* **do**
    if *($Cost_{\text{j}}$ >Threshold)* **then**
        Explore for transformation of operation(j);
        Update(SW Parameters);
        Transform(Code);
}
Evaluate Reg(UF,PF,Max){
Compose HDFG(UF,PF);
Compute life time(HDFG);
SLT=Sort(life time);
MLT=Left edge algorithm(SLT,Max);
Num_{Reg}=Count(MLT);
}

ALGORITHM 18: Βήμα απεικόνισης στο Data Path.

Step 4.1: DP Mapping\
Allocate(Primitive Ops,Single HW);
Schedule, Assign(UF_{FG},Single HW);
UF_{HW}=1;
Compute U_{Op};
while *($U_{\text{Op}} < U_{\text{Thr}}$)* **do**
    Update UF_{HW};
    Schedule, Assign(UF_{HW},Single HW);
    Compute U_{R};
if *($f_{\text{HW}} > f_{\text{PC}}$)* **then**
    Estimate PL stages;
    Pipeline(PL);
if *($CP_{\text{HW}} < t_{\text{HW}}$)* **then**
    Parallelize(PF,Single HW);
if *($CP_{\text{HW}} < t_{\text{HW}}$)* **then**
    Pipeline(PL);
Step 4.2: FG Memory Connection\
Estimate $ABandW_{\text{FG}}$;
$Num_{\text{FG,Ports}}$=$\frac{BandW}{ABandW_{\text{FG}}}$;
Allocate ports;
Connect(DP,FG);

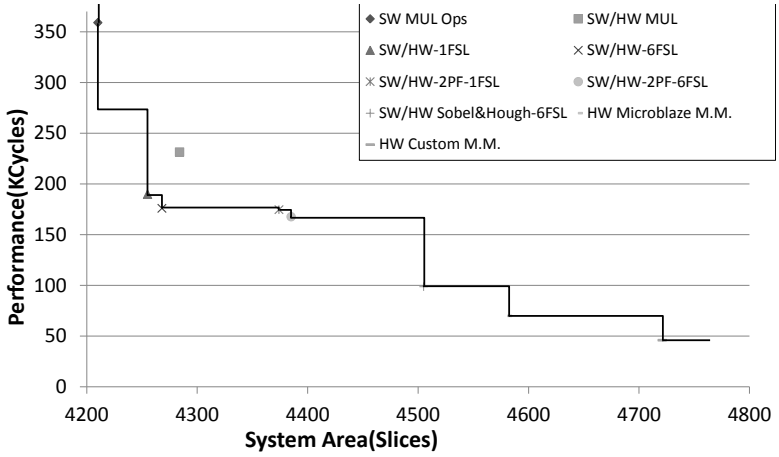
(SW/HW Organization) (Σχ. 16), η διαχείριση της μνήμης στο επεξεργαστικό τμήμα (Foreground (FG) Memory Management) (Σχ. 17) και η απεικόνιση στις μονάδες εκτέλεσης (Data Path (DP) Mapping) (Σχ. 18). Κάθε βήμα περιγράφεται από παραμετρικά πλαίσια, δηλαδή μια επεκτάσιμη δομή με τις σχετικές παραμέτρους, εξισώσεις και ρουτίνες που συνδέονται με μονοκατευθυντήρια διάδοση περιορισμών. Η εισαγωγή τιμών στις παραμέτρους και η εφαρμογή των ρουτινών παράγουν μια Pareto καμπύλη. Κάθε σημείο εξερευνά ενεργές τιμές για τις παραμέτρους και οδηγεί σε διαφορετικές τιμές για την απόδοση και την επιφάνεια ολοκλήρωσης. Οι επιλογές σε ένα βήμα απεικόνισης διαδίδονται σαν σχεδιαστικοί περιορισμοί στα επόμενα βήματα για να αποκόψουν επιλογές οι οποίες δεν είναι συμβατές. Μια επιλογή δεν είναι συμβατή όταν για να μπορεί να επιτευχθεί θα πρέπει

να αγνοηθεί ένας σχεδιαστικός περιορισμός από προηγούμενα βήματα. Για παράδειγμα, οι αποφάσεις για το χρονοπρογραμματισμό για το κρίσιμο τμήμα της εφαρμογής διαδίδονται σαν σχεδιαστικοί περιορισμοί στο μη κρίσιμο τμήμα. Οι επιλογές που απαιτούν την αλλαγή του χρονοπρογραμματισμού του κρίσιμου τμήματος είναι μη συμβατές και επομένως διαγράφονται σαν επιλογές. Με αυτόν τον τρόπο μόνο τα ενεργά και υποσχόμενα τμήματα του χώρου λύσεων εξερευνούνται μέσω ενός αποδοτικού κλαδέματος που βασίζεται στις τιμές των παραμέτρων που έχουν διαδοθεί και των περιπτώσεων απεικόνισης σε κάθε βήμα. Η επεκτασιμότητα της μεθοδολογίας παραμένει, διότι ακόμα και όταν αυξάνει το μέγεθος της εφαρμογής ο χρόνος εξερεύνησης παραμένει γραμμικός.

Πίνακας Β.6: Απόδοση και επιφάνεια ολοκλήρωσης για την πραγματικού χρόνου βιοϊατρική εφαρμογή που βασίζεται σε ένα μικροροικό FPGA.

Σχεδιασμός	Απόδοση 200x16 Window		Απόδοση 300x75 Window		Περ.Ολοκ.	
					HW Συν.	Σύνολο
	Κύκλοι	Χρόνος(ms)	Κύκλοι	Χρόνος(ms)	Slices	Slices
SW MUL Ops	358,996	4.31	1,458,006	17.50	0	4210
SW/HW MUL	231,444*	2.78*	765,901*	9.19*	≈128*	4284*
SW/HW-1FSL	189,864	2.28	439,591	5.27	114	4255
SW/HW-6FSL	176,004	2.11	330,821	3.97	114	4268
SW/HW-2PF-1FSL	174,618	2.09	319,944	3.84	231	4374
SW/HW-2PF-6FSL	167,688	1.98	265,559	3.19	231	4385
SW/HW Sobel&Hough-6FSL	98,705*	1.19*	196,576*	2.36*	314*	4505*
HW & Microblaze M.M.	>70,000*	>0.84*	>182,638*	>2.19*	>375*	>4581*
HW & Custom M.M.	>46,000*	> 0.56*	>142,218*	>1.77*	>550*	>4721*

*estimated



Σχ. Β.17: Pareto καμπύλη για παράθυρο εφαρμογής 200x16 της ρουτίνας για την εύρεση της γωνίας απόκλισης.

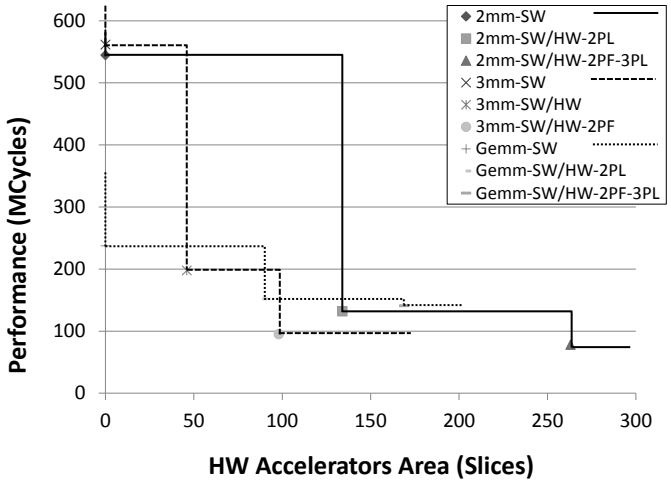
Επιπρόσθετα, περιγράφεται η εφαρμογή της προτεινόμενης μεθοδολογίας και εφαρμόζεται σε ένα σύνολό από benchmarks. Τα πιο λεπτομερή αποτελέσματα παρουσιάζονται για μια πραγματικού χρόνου βιοϊατρική εφαρμογή που βασίζεται σε ένα μικροροικό FPGA. Το κέρδος σχεδίασης είναι 47.11% στην απόδοση και 72.29% στην επιφάνεια ολοκλήρωσης για SW και HW υλοποιήσεις. Τα αποτελέσματα απεικονίζονται στον Πίνακα Β.6 και στο Σχ. Β.17.

Πίνακας Β.7: Απόδοση και επιφάνεια ολοκλήρωσης για το PolyBench Benchmark Suite.

Bench.	Σχεδιασμός	Απόδοση (Κύκλοι)	HW Συν.					
			Αρχικές Πράξεις	CP (ns)	f (MHz)	Slices	DSP Slices	Σύνολο Slices
2mm	SW	6,000,208	-	-	-	-	-	-
	SW/HW	2,155,152	1-2PL	10.17	98.32	38	6	134
			2	10.11	98.87	24	3	
	SW/HW-2PF	1,298,043	1-3PL	10.44	95.8	71	12	263
			2	11.49	87.06	48	6	
3mm	SW	9,067,317	-	-	-	-	-	-
	SW/HW	3,227,592	1	10.13	98.67	22	3	46
	SW/HW-2PF	1,832,904	1	11.48	87.08	50	6	98
Bicg	SW	128,678	-	-	-	-	-	-
	SW/HW	48,256	1	10.11	98.91	42	6	90
Gemm	SW	3,648,404	-	-	-	-	-	-
	SW/HW	2,246,593	1-2PL	10.22	97.81	41	6	89
	SW/HW-3PL	2,190,881	1-2PF	10.65	93.9	73	12	169
Gesummv	SW	143,301	-	-	-	-	-	-
	SW/HW	70,753	1	10.11	98.87	24	3	48
	SW/HW-2PF	65,633	1	11.49	87.06	48	6	96
Atax	SW	139,420	-	-	-	-	-	-
	SW/HW	80,790	1	10.13	98.67	22	3	46
Cholesky	SW	374,334	-	-	-	-	-	-
	SW/HW	70,753	1	10.31	96.99	25	3	49
Gemver	SW	251,550	-	-	-	-	-	-
	SW/HW	96,650	1	11.48	87.08	50	6	203
			2-2PL	10.17	98.32	39	6	
			3	2.67	374.96	18	0	
Jacobi_2d	SW	1,333,510	-	-	-	-	-	-
	SW/HW	303,464	1	6.59	151.86	80	0	80
Correlation	SW	4,413,504	-	-	-	-	-	-
	SW/HW	1,084,299	1	2.65	376.65	16	0	1194
			2	11.91	83.97	32	6	
			3-3PL	44.82	22.31	1028	3	
			4	10.13	98.67	22	3	

SW: Software, SW/HW: Co-design, PF=Parallel Factor, PL=Pipeline stages

Επιπρόσθετα περιγράφονται πειραματικά αποτελέσματα για 10 PolyBench benchmarks [159] που δείχνουν την αποδοτικότητα και την ευρεία εφαρμογή της προτεινόμενης μεθοδολογίας. Τα αποτελέσματα απεικονίζονται στον Πίνακα Β.7. Οι αντίστοιχες καμπύλες απεικονίζονται στο Σχ. Β.18.



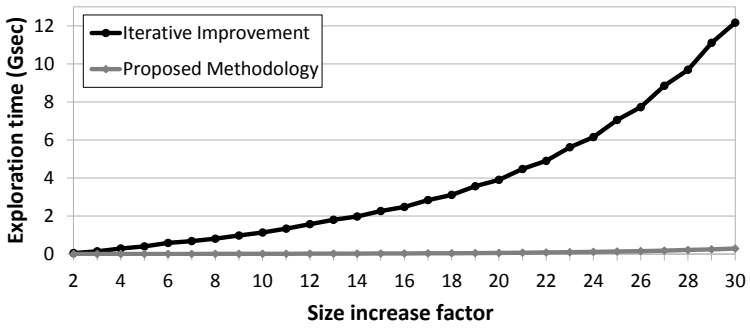
Σχ. Β.18: Pareto καμπύλες για τις εφαρμογές 2mm, 3mm και Gemm για μέγεθος δεδομένων 128.

Η προτεινόμενη μεθοδολογία συγκρίνεται με μια επαναληπτική μεθοδολογία για την απόδοση και τον χρόνο εξερεύνησης. Τα αποτελέσματα παρουσιάζονται στο Σχ. Β.8 και

Σχ. Β.19.

Πίνακας Β.8: Αποτελέσματα για την προτεινομένη μεθοδολογία και για μια επαναληπτική μεθοδολογία (iterative improvement).

Εφαρμογή	Επαναληπτική μεθ.			Προτεινόμενη μεθ.		
	Απόδ. (ms)	Περ.Ολοκ. (LUT)	Χρόνος (ms)	Απόδ. (ms)	Περ.Ολοκ. (LUT)	Χρόνος (ms)
Seidel-2d	0.02020892	72	118,686.089	0.02020892	72	1,125.692
	0.02024000	64		0.02022202	64	
	0.02524778	56		0.02521426	56	
	0.02526088	48		0.02521426	48	
	0.03525846	40		0.03062306	40	
	-	-		0.04059266	32	
Gemm	0.1609017	40	50,451.987	0.1609017	40	256.695
	0.1609075	32		0.1609075	32	
	0.2418297	24		0.2418297	24	
Syr2k	0.3217926	40	55,155.037	0.3217926	40	265.931
	0.3217984	32		0.3217984	32	
	0.48365575	24		0.48365575	24	



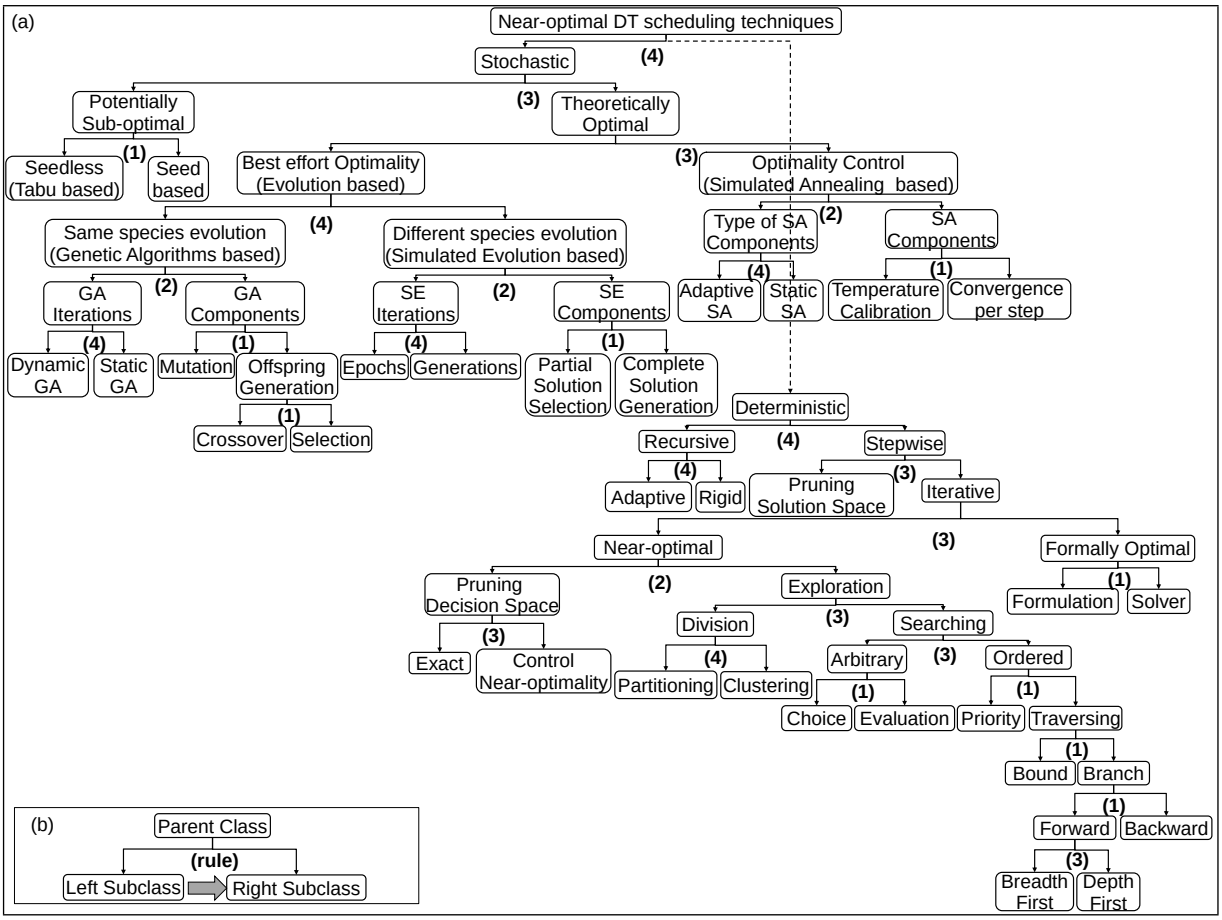
Σχ. Β.19: Χρόνος εξερεύνησης όταν το μέγεθος της εφαρμογής αυξάνεται κατά έναν παράγοντα.

Β.10 Πλαίσιο με τις σχεδόν βέλτιστες τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων.

Το πρόβλημα χρονοπρογραμματισμού και ανάθεσης των πόρων εφαρμόζεται σε πολλές περιοχές [141], όπως υπολογιστές, οικονομικά, χρονοπρογραμματισμός εργασίας, διαχείριση προσωπικού, παραγωγή κτλ. Τόσο η ερευνητική όσο και η βιομηχανική κοινότητα έχει επενδύσει δεκαετίες σε έρευνα και πειράματα στις τεχνικές που επιλύουν το πρόβλημα του χρονοπρογραμματισμού και της ανάθεσης πόρων. Σημαντικές εξελίξεις έχουν δημιουργηθεί στο χώρο απεικόνισης πολυεπεξεργαστικών συστημάτων

που ταξινομούν στο χώρο και στο χρόνο διεργασίες, μεταφορές δεδομένων και πράξεις. Ωστόσο το πρόβλημα του χρονοπρογραμματισμού και της ανάθεσης των πόρων δεν έχει λυθεί επαρκώς [182].

Όπως έχει αναφερθεί στην αναφορά ο [182], η καλύτερη κατανόηση της συμπεριφοράς των αλγορίθμων του χρονοπρογραμματισμού και της ανάθεσης πόρων θα οδηγήσει στην ανάπτυξη πιο αποδοτικών προσεγγίσεων. Η κατανόηση αυτή προκύπτει από μια αποδοτική κατηγοριοποίηση του συνόλου των τεχνικών. Μια κατηγοριοποίηση η οποία είναι ολοκληρωτική παρέχει μια πανοραμική περιγραφή όλων των δυνατών τεχνικών χρονοπρογραμματισμού και ανάθεσης πόρων που μπορεί να υπάρξουν. Η κατηγοριοποίηση χωρίζει αποδοτικά τις τεχνικές σε κλάσεις με μοναδικά χαρακτηριστικά. Στην βιβλιογραφία υπάρχουν λίγες κατηγοριοποιήσεις ([125], [126], [17], [151], [152], [171], [141] και [60]) για τις τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων που εφαρμόζονται κατά την σχεδίαση για πλατφόρμες με έναν ή πολλαπλούς επεξεργαστές. Οι υπάρχουσες κατηγοριοποιήσεις είναι μερικώς ολοκληρωμένες, περιλαμβάνουν πλεονασμούς και δεν παρέχουν αποδοτικότητα. Άλλες μελέτες ([114], [79] και [182]) επικεντρώνονται στην περιγραφή νέων τάσεων.



Σχ. Β.20: Η προτεινόμενη κατηγοριοποίηση των τεχνικών που επιλύουν σχεδόν βέλτιστα το πρόβλημα χρονοπρογραμματισμού και ανάθεσης πόρων. Ο αριθμός κάτω από κάθε διαχωρισμό περιγράφει τον κανόνα που εφαρμόστηκε για να προσδιοριστεί η κατεύθυνση της διάδοσης των περιορισμών.

Στην παρούσα διδακτορική διατριβή προτείνεται μια νέα κατηγοριοποίηση για τεχνικές που βρίσκουν σχεδόν βέλτιστο χρονοπρογραμματισμό και ανάθεση των πόρων σε πλατφόρμες με ένα ή και περισσότερους επεξεργαστές. Η κατηγοριοποίηση δημιουργείται εφαρμόζοντας τις αρχές της επαναχρησιμοποιούμενης DSE μεθοδολογίας του κεφαλαίου 2. Το αποτέλεσμα απεικονίζεται στο Σχ. Β.20. Παρουσιάζεται ένας νέος και συστηματικός τρόπος για να κατηγοριοποιηθούν οι τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων. Επιπρόσθετα στο κεφάλαιο 7 περιγράφεται πως η προτεινόμενη κατηγοριοποίηση καλύπτει αποδοτικά όλες τις υπάρχουσες τεχνικές, αφού στην προτεινόμενη κατηγοριοποίηση μια τεχνική μπορεί να ανήκει σε μια αρχική κλάση ή σε συνδυασμό κλάσεων.

Β.11 Μεθοδολογία για την ανάπτυξη παραμετρικών πλαισίων για σχεδόν βέλτιστες και επεκτάσιμες τεχνικές χρονοπρογραμματισμού και ανάθεσης.

Οι τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων επηρεάζουν τον σχεδιασμό και την απόδοση του συστήματος διότι είναι υπεύθυνες για την πραγματικού χρόνου συμπεριφορά, την ελάχιστη κατανάλωση ενέργειας, την αξιοπιστία του συστήματος κτλ. Οι τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων αναθέτουν πράξεις, ομάδες από πράξεις, αναφορές στην μνήμη ή μεταφορές δεδομένων σε βήματα ελέγχου με βάση το χρόνο και σε πόρους του συστήματος με βάση το χώρο, όπως ομογενή ή ετερογενή επεξεργαστικά στοιχεία, μνήμες, διαύλους κτλ. Οι περιορισμοί, όπως deadlines, πρέπει να ικανοποιούνται και οι κρίσιμες μετρικές του συστήματος, όπως ο απαιτούμενος αριθμός πόρων, πρέπει να μειώνονται ανάλογα με το υπο μελέτη πρόβλημα χρονοπρογραμματισμού και ανάθεσης πόρων.

Το πρόβλημα του χρονοπρογραμματισμού και της ανάθεσης πόρων δεν έχει επιλυθεί πλήρως και υπάρχει χώρος για περαιτέρω εξερεύνηση στις αρχιτεκτονικές πλατφόρμες με έναν ή και πολλαπλούς επεξεργαστές [182]. Τα υπάρχοντα εργαλεία προσφέρουν με αυτοματοποιημένο τρόπο χρονοπρογραμματισμό και ανάθεση πόρων στις αρχιτεκτονικές. Ωστόσο παρέχουν περιορισμένη κάλυψη αφού εφαρμόζουν μόνο συγκεκριμένες τεχνικές τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων για όλες τις περιοχές εφαρμογών, π.χ. Cheddar [181]. Συνήθως τα εργαλεία εξερευνούν μια μεγάλη γκάμα από τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων, οι οποίες επικεντρώνονται σε μια σχετικά περιορισμένη περιοχή. Όταν τα χαρακτηριστικά της περιοχής δεν συμπίπτουν με τα χαρακτηριστικά της εφαρμογής, το αποτέλεσμα είναι συνήθως μη βέλτιστο. Τα εργαλεία δεν μπορούν παρέχουν συμβουλή για το ποια τεχνική χρονοπρογραμματισμού και ανάθεσης πόρων είναι η πιο υποσχόμενη. Μια μεθοδολογία για την εύρεση της

πιο υποσχόμενης τεχνικής χρονοπρογραμματισμού και ανάθεσης πόρων για μια περιοχή εφαρμογών λείπει από την υπάρχουσα βιβλιογραφία [5].

Η εφαρμογή μιας πολύ ευρείας εξερεύνησης για τις τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων είναι χρονοβόρα λόγω του μεγάλου αριθμού από διαφορετικής φύσεως τεχνικές. Επομένως, οι σχεδιαστές αναπτύσσουν τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων, που μπορούν να επιτύχουν σχεδόν βέλτιστο αποτέλεσμα για πολύ παρεμφερής εφαρμογές, βασιζόμενοι στην εμπειρία τους. Ωστόσο, πρακτικά, οι εφαρμογές είναι πολύ διαφορετικές στην φύση τους και περιλαμβάνουν πολύπλοκους και μεγάλους γράφους με χρονικούς περιορισμούς πραγματικού χρόνου. Σε αυτές τις περιπτώσεις οι υπάρχοντες τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων είτε αυξάνουν κατά πολύ τον χρόνο εξερεύνησης είτε μειώνουν την ποιότητα του αποτελέσματος. Π.χ. Integer Linear Programming (ILP) τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων βρίσκουν βέλτιστο αποτέλεσμα αλλά είναι εφαρμόσιμοι σε μικρούς σε μέγεθος γράφους. Επομένως χρειάζεται μια συστηματική μεθοδολογία για την ανάπτυξη τεχνικών χρονοπρογραμματισμού και ανάθεσης πόρων που ικανοποιούν τους περιορισμούς που τίθενται από την υπο μελέτη περιοχή.

Στην παρούσα διδακτορική διατριβή εφαρμόζουμε τις αρχές της επαναχρησιμοποιούμενης μεθοδολογίας στην κατηγοριοποίηση των τεχνικών χρονοπρογραμματισμού και ανάθεσης πόρων για να αναπτύξουμε μια συστηματική μεθοδολογία που δημιουργεί παραμετρικά πλαίσια για τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων κάτω από περιορισμούς για σχεδόν βέλτιστο αποτέλεσμα και για επεκτάσιμη τεχνική που εφαρμόζεται σε περίπλοκους και μεγάλους γράφους εφαρμογών. Η προτεινόμενη μεθοδολογία αποτελείται από τέσσερα βήματα, που απεικονίζονται στο Σχ. 19.

Το πρώτο βήμα αρχικοποιεί τις απαραίτητες δομές που χρησιμοποιούνται στα επόμενα βήματα. Το δεύτερο βήμα αναλύει την εφαρμογή και την αρχιτεκτονική πλατφόρμα για τα βρει τις χρήσιμες ιδιότητες, όπως το μέγεθος και την δομή των γράφων, το πόσο βέλτιστο πρέπει να είναι το αποτέλεσμα, τον διαθέσιμο χρόνο για την εκτέλεση της τεχνικής κτλ. Οι ιδιότητες αυτές χρησιμοποιούνται σαν περιορισμοί που τίθενται από την υπό μελέτη περιοχή και πρέπει να ικανοποιούνται από την τεχνική που θα περιγράφεται από το παραμετρικό πλαίσιο. Στο τρίτο βήμα, οι περιορισμοί διαδίδονται κάθετα στην κατηγοριοποίηση των τεχνικών χρονοπρογραμματισμού και ανάθεσης πόρων. Κάθε κλάση περιγράφει μια διαφορετική κατηγορία τεχνικών χρονοπρογραμματισμού και ανάθεσης πόρων μέσω ενός παραμετρικού πλαισίου. Οι περιορισμοί που διαδίδονται αποκόπτουν τις κλάσεις που δεν είναι συμβατές. Για παράδειγμα, ο περιορισμός για βέλτιστο αποτέλεσμα αποκόπτει την κλάση που περιγράφει σχεδόν βέλτιστες τεχνικές. Το αποτέλεσμα είναι ένα μειωμένο δέντρο με μόνο τις συμβατές κλάσεις. Στο τέταρτο βήμα το δέντρο γίνεται επίπεδο και τα παραμετρικά πλαίσια ενώνονται και συνδυάζονται σε ένα τελικό παραμετρικό πλαίσιο ακολουθώντας την οριζόντια διάδοση περιορισμών, όπως περιγράφεται στην κατηγοριοποίηση των τεχνικών.

ALGORITHM 19: Προτεινόμενη μεθοδολογία για την δημιουργία παραμετρικών πλαισίων που περιγράφουν επεκτάσιμες και σχεδόν βέλτιστες τεχνικές χρονοπρογραμματισμού.

```
Input: Application and Platform Domain
Output: Combined Parameterized Template of
        Developed Techniques
\*Step 1*\
Class list Q ← The set of all T classes
v ← root of T
Search list S ← ∅
Enqueue(S,v);
Pruned Classes list P ← ∅
\*Step 2*\
Constraints = Analysis(App.&Platf.Domain);
\*Step 3a*\
while (S ! = ∅) do
  Dequeue(u,S);
  for (i=0;i<NumOfConstraints;i++) do
    if (Label(u) ≠ constraint[i]) then
      Dequeue(u,Q);
    else
      Enqueue(S,LeftChild(u));
      Enqueue(S,RightChild(u));
\*Step 3b*\
for (i=0;i<length(Q);i++) do
  Dequeue(i,Q);
  while (NumOfChildren(i)==1) do
    Dequeue(Child(i),Q);
    u=Merge(i,Child(i));
    Enqueue(Q,u));
\*Step 4a*\
Search list S ← ∅ ;
v ← root of Q ;
Enqueue(S,v));
while (S! = ∅) do
  Dequeue(u,S);
  if (NumOfChildren(u)==0) then
    n=Merge(u, acc);
    Enqueue(F,n);
    if (Arrow(u) exists) then
      Enqueue(S, Dest(Arrow(u)));
    else
      Backtrack();
      Acc=Remove(u,Acc);
  else
    Acc=Merge(u,acc);
    Enqueue(S, LeftChild(u));
\*Step 4b*\
for (i=0;i<length(F);i++) do
  t=SelectTemplate(i)
  AddToCombinedParametric(t)
```

Το τελικό παραμετρικό πλαίσιο είναι το αποτέλεσμα της προτεινόμενης μεθοδολογίας και περιγράφει τις προτεινόμενες τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων για την υπο μελέτη περιοχή εφαρμογών. Στην παρούσα διδακτορική διατριβή εφαρμόζουμε την προτεινόμενη μεθοδολογία στην περιοχή εφαρμογών με μεγάλου μεγέθους και περίπλοκους γράφους και πραγματικού χρόνου απαιτήσεις. Επιπρόσθετα παρουσιάζονται διαφοροποιήσεις στους περιορισμούς της υπο μελέτη περιοχής εφαρμογών με στόχο να δείξουμε πως οι περιορισμοί επηρεάζουν την εξερεύνηση του χώρου λύσεων. Το τελικό παραμετρικό πλαίσιο που προκύπτει από την προτεινόμενη μεθοδολογία επιτυγχάνει κέρδος 13-18% στην απόδοση για εφαρμογές από το Mibench benchmark suite, π.χ. 18,2% για την εφαρμογή Forward DCT. Το αποτέλεσμα της προτεινόμενης μεθοδολογίας μπορεί να χρησιμοποιηθεί για να την αποδοτική αυτοματοποίηση του χρονοπρογραμματισμού και της ανάθεσης πόρων σε υπάρχοντα εργαλεία: η πιο υποσχόμενη τεχνική για την συγκεκριμένη περιοχή εφαρμογών μπορεί να αποφασιστεί αποδοτικά από τις τεχνικές που περιγράφονται στο τελικό παραμετρικό πλαίσιο δίνοντας συγκεκριμένες τιμές στις παραμέτρους του πλαισίου. Η διαδικασία αυτή θα δημιουργήσει την τελική τεχνική χρονοπρογραμματισμού και ανάθεσης πόρων

η οποία θα υλοποιηθεί από το εργαλείο. Η μεθοδολογία συγκεκριμενοποίησης της τεχνικής χρονοπρογραμματισμού και ανάθεσης πόρων από το παραμετρικό πλαίσιο δεν αντιμετωπίζεται από την παρούσα διδακτορική διατριβή, αλλά αποτελεί μελλοντική ερευνητική κατεύθυνση.

Στο κεφάλαιο 8 περιγράφεται λεπτομερώς η προβολή των αρχών και της χρήσης του framework της επαναχρησιμοποιούμενης μεθοδολογίας εξερεύνησης στο πρόβλημα ανάπτυξης υποσχόμενων τεχνικών χρονοπρογραμματισμού και ανάθεσης πόρων, οι οποίες είναι επεκτάσιμες και παρέχουν σχεδόν βέλτιστο αποτέλεσμα για μια συγκεκριμένη περιοχή εφαρμογών. Παρουσιάζεται ένα λεπτομερές παράδειγμα στο οποίο οι υπάρχουσες τεχνικές χρονοπρογραμματισμού και ανάθεσης πόρων δεν μπορούν να επιτύχουν σχεδόν βέλτιστα αποτελέσματα και δεν είναι επεκτάσιμες και παρουσιάζεται η ανάγκη και δίνεται η λύση για μια μεθοδολογία που να αναπτύσσει τεχνικές οι οποίες να είναι επεκτάσιμες και σχεδόν βέλτιστες για το υπό μελέτη πρόβλημα.

References

[1] T. Vander Aa, B.F. Mei, and B. De Sutter. A backtracking instruction scheduler using predicate-based code hoisting to fill delay slots. In *Proc. Int'l Conf. on Compilers, Architecture and Synthesis for Embedded Systems*, pages 229--237, New York, NY, USA, 2007. ACM. 170

[2] I. Ahmad et al. Automatic parallelization & scheduling of programs on multiprocessors using casch. In *ICPP*, pages 288--291, 1997. 180

[3] Yongjin Ahn et al. Socdal: System-on-chip design accelerator. *Trans. Des. Autom. Electron. Syst.*, **13**[1]:17:1--17:38, Feb 2008. 4, 121, 217

[4] Nicolaos Alachiotis, Vasileios I. Kelefouras, George S. Athanasiou, Harris E. Michail, Angeliki S. Kritikakou, and Costas E. Goutis. A data locality methodology for matrix--matrix multiplication algorithm. *The Journal of Supercomputing*, **59**:830--851, 2012. 207

[5] F. Balarin et al. Scheduling for embedded real-time systems. *Des.Test Comp.*, **15**:71--82, Jan. 1998. 175, 251

[6] Florin Balasa et al. Transformation of nested loops with modulo indexing to affine recurrences. *Let. Parallel Proces.*, **4**:271--280, 1994. 42

[7] V. Balasundaram and K. Kennedy. A technique for summarizing data access and its use in parallelism enhancing transformations. *SIGPLAN Not.*, **24**[7]:41--53, Jun 1989. 36

[8] A. Balboni et al. Partitioning and exploration strategies in the tosca co-design flow. In *Proc. Int'l Work. HW/SW CoDesign*, pages 62--69, USA, 1996. IEEE Computer Society. 36

[9] M.R. Barbacci and D.P. Siewiorek. Automated exploration of the design space for register transfer (rt) systems. In *Proc. 1st annual Int'l Symp. Computer Architecture*, pages 101--106, New York, NY, USA, 1973. ACM. 149

[10] S.J. Beaty. Genetic algorithms and instruction scheduling. In *Proc. Int'l Symp. Microarchitecture*, pages 206--211. ACM, 1991. 182

[11] Tobias Bjerregaard et al. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, **38**[1]:1--51, jun 2006. 135

[12] C. G. E. Boender et al. A stochastic method for global optimization. *Mathematical Programming*, **22**:125--140, 1982. 180

[13] S.W. Bollinger and S.F. Midkif. Processor and link assignment in multicomputers using simulated annealing. In *Proc. IEEE Int'l Conf. Parallel Processing*, Los Alamitos, CA, USA, Jun 1988. IEEE Computer Society. 160

[14] Magnus Broberg et al. A tool for binding threads to processors. In *Proc. Int'l Euro-Par Conf. Par. Processing*, pages 57--61. Springer-Verlag, 2001. 180

[15] T.J. Callahan et al. The garp architecture and c compiler. *J. Computer*, **33**[4]:62--69, Apr. 2000. 121

[16] Andrea Capitanio et al. A hypergraph-based model for port allocation on multiple-register-file vliw architectures. *Int'l J. Parallel Programming*, **23**:499--513, 1995. 135

[17] T.L. Casavant and J.G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, **14**[2]:141--154, Feb 1988. 145, 151, 152, 249

[18] F. Catthoor. Energy-delay efficient data storage and transfer architectures and methodologies: Current solutions and remaining problems. *J. VLSI Signal Processing*, **21**:219--231, 1999. 33, 38, 228

[19] F. Catthoor et al. System-level transformations for low power data transfer and storage. In *Low-Power CMOS Design*, pages 609--618, Los Alamitos, CA, USA, 1998. IEEE Press. 23, 25, 26, 37, 226

[20] Francky Catthoor, Eddy de Greef, and Sven Suytack. *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Kluwer Academic Publishers, Norwell, MA, USA, 1998. 26

[21] Francky Catthoor, Eddy de Greef, and Sven Suytack. *Unified low-power design flow for data-dominated multi-media and telecom application*. Kluwer Academic Publishers, Norwell, MA, USA, 2000. 23, 25, 26, 27, 209, 226

[22] Francky Catthoor et al. Clustered l0 (loop) buffer organization and combination with data clusters. In *Ultra-Low Energy Domain-Specific Instruction-Set Processors*, **0**, pages 115--141. Springer, 2010. 177

[23] N. Chabini and W. Wolf. Unification of scheduling, binding, and retiming to reduce power consumption under timings and resources constraints. *IEEE Trans. Very Large Scale Integr.(VLSI) Syst.*, **13**[10]:1113--1126, 2005. 159, 171, 181

[24] K.S. Chatha and R. Vemuri. A tool for partitioning and pipelined scheduling of hardware-software systems. In *Proc. Int'l Symp. System Synthesis*, pages 145--151, Los Alamitos, CA, USA, 1998. IEEE Computer Society. 170

[25] S. Chaudhuri, S.A. Blthye, and R.A. Walker. A solution methodology for exact design space exploration in a three-dimensional design space. *IEEE Trans. Very Large Scale Integration Systems*, **5**[1]:69--81, Mar 1997. 159, 167

[26] S. Chaudhuri, R.A. Walker, and J.E. Mitchell. Analyzing and exploiting the structure of the constraints in the ilp approach to the scheduling problem. *IEEE Trans. Very Large Scale Integr.(VLSI) Syst.*, **2**[4]:456--471, 1994. 168

[27] S. Che et al. Dymaxion: optimizing memory access patterns for heterogeneous systems. In *Proc. Int'l Conf. High Performance Computing (PC)*, pages 13:1--13:11, USA, 2011. ACM. 37

[28] S. Chen et al. Adaptive simulated annealing for optimization in signal processing applications. *Signal Proc.*, **79**[1]:117--128, Oct. 1999. 194

[29] D. Cho et al. Software controlled memory layout reorganization for irregular array access patterns. In *Proc. Int'l Conf. CASES*, pages 179--188, New York, NY, USA, 2007. ACM. 36

[30] P. Clauss et al. Automatic memory layout transformations to optimize spatial locality in parameterized loop nests. *SIGARCH Comput. Archit. News*, **28**:11--19, Mar. 2000. 37

[31] Albert Cohen et al. Storage mapping optimization for parallel programs. In *Proc. Int'l Euro-Par Conf. Parallel Processing*, pages 375--382, London, UK, UK, 1999. Springer-Verlag. 33, 228

[32] K. Compton et al. Reconfigurable computing: a survey of systems and software. *ACM Comput. Surv.*, **34**:171--210, Jun. 2002. 120

[33] Jason Cong et al. Automatic memory partitioning and scheduling for throughput and power optimization. *ACM Trans. Des. Autom. Electron. Syst.*, **16**[2]:15:1--15:25, Apr 2011. 37

[34] Keith D. Cooper et al. Operator strength reduction. *ACM Trans. Program. Lang. Syst.*, **23**[5]:603--625, Sep. 2001. 126

[35] Béatrice Creusillet and François Irigoin. Exact vs. approximate array region analyses, 1996. 37

[36] Criticalblue. Criticalblue cascade, programmable application coprocessor generation. `http://www.criticalblue.com`, 2012. 120

[37] L. Cucu-Grosjean and O. Buffet. Global multiprocessor real-time scheduling as a constraint satisfaction problem. In *Proc. Int'l Conf. Parallel Processing Workshops*, pages 42--49, Washington, DC, USA, 22-25 2009. IEEE Computer Society. 33, 38, 168, 228

[38] A. Darte et al. Lattice-Based Memory Allocation. *Tran. Computers*, **54**:1242--1257, 2005. 37

[39] T. Davidović. Exhaustive list-scheduling heuristic for dense task graphs. *YUJOR*, **10**[1]:123--36, 2000. 4, 181, 217

[40] Tatjana Davidović, Leo Liberti, Nelson Maculan, and Nenad Mladenović. Towards the optimal solution of the multiprocessor scheduling problem with communication delays. In *Proc. Multi-Disciplinary Int'l Conf. Scheduling Theory and Applications*, Nottingham, UK, 2007. Sherwood Press. 171, 181

[41] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold Company, New York, 1991. 172

[42] Eddy De Greef et al. Memory size reduction through storage order optimization for embedded parallel multimedia applications. In *Parallel Computing*, pages 84--98, 1997. 33, 228

[43] S. Devadas and A.R. Newton. Algorithms for hardware allocation in data path synthesis. *IEEE Trans. Computer-Aided Design of Integr. Circuits Syst.*, **8**[7]:768--781, Jul 1989. 149, 150, 160, 171

[44] M.K. Dhodhi and I. Ahmad. A multiprocessor scheduling scheme using problem-space genetic algorithm. In *IEEE Proc. 1st Int'l Conf. Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 152--157, Los Alamitos, CA, USA, Sep 1995. IEEE Computer Society. 150, 173, 182

[45] M.K. Dhodhi et al. Shemus: Synthesis of heterogeneous multiprocessor systems. *Microproc. & Microsys.*, **19**[6]:311--319, 1995. 182

[46] M.K. Dhodhi, F.H. Hielscher, R.H. Storer, and J. Bhasker. Datapath synthesis using a problem-space genetic algorithm. *IEEE Trans. Computer-Aided Design of Integr. Circuits Syst.*, **14**[8]:934--944, 1995. 172

[47] J. Ph. Diguet et al. A framework for high level estimations of signal processing vlsi implementations. *J. VLSI Signal Process. Syst.*, **25**[3]:261--284, Jul. 2000. 126, 128

[48] R. Dimond et al. Custard - a customisable threaded fpga soft processor and tools. In *Proc. Int'l Conf. Field Progr.Logic&Applic.*, pages 1--6, USA, 2005. IEEE. 121

[49] J. Dongarra et al. A tool to aid in the design, implementation, and understanding of matrix algorithms for parallel processors. *J. Parallel & Distributed Computing*, **9**[2]:185--202, 1990. 36

[50] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Syst., Man, Cybern.B,Cybern.*, **26**:29--41, 1996. 150, 174, 182

[51] F.Catthoor et al. Proposal for unified system design meta flow in task-level and instruction-level design technology research for multi-media applications. In *Proc. Intnl. Symp. System-Level Synthesis*, pages 89--95, Dec. 1998. 11, 26, 226

[52] F. Ferrandi et al. An evolutionary approach to area-time optimization of fpga designs. In *Proc. Int'l Conf. Embedded Computer SAMOS*, pages 145--152, USA, Jul. 2007. IEEE. 121

[53] F. Ferrandi, P.L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo. Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems. *Trans. Computer-Aided Desing Integr. Circuits Syst.*, **29**[6]:911--924, 2010. 150, 174, 181

[54] H. Flatt et al. Mapping of a real-time object detection application to a configurable risc/coprocessor architecture at full hd resolution. In *ReConFig&FPGAs*, pages 452--457, USA, Dec. 2010. IEEE. 121

[55] F.H.M. Franssen et al. Modeling multidimensional data & control flow. *VLSI*, **1**[3]:319--327, Sep. 1993. 34, 35, 42, 55, 229, 232

[56] D. Gajski et al. Specsyn: an environment supporting the specify-explore-refine paradigm for hardware/software system design. *Trans. VLSI*, **6**[1]:84--100, Mar. 1998. 121

[57] C.H. Gebotys. Throughput optimized architectural synthesis. *IEEE Trans. Very Large Scale Integr.(VLSI) Syst.*, **1**[3]:254--261, Sep 1993. 168

[58] C.H. Gebotys and M.I. Elmasry. A global optimization approach for architectural synthesis. In *IEEE Proc. Int'l Conf. Computer-Aided Design: Digest of Technical Papers*, pages 258--261, Los Alamitos, CA, USA, Nov 1990. IEEE Computer Society. 170

[59] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, New York, 1989. 161

[60] S. Govindarajan. Scheduling algorithms for high-level synthesis. Term Paper in Digital Design Environments, 1995. 145, 151, 249

[61] W. Grass. A branch-and-bound method for optimal transformation of data flow graphs for observing hardware constraints. In *Proc. Eur. Conf. Design Automation*, pages 73--77, Los Alamitos, CA, USA, Mar 1990. IEEE Computer Society Press. 150, 152, 168

[62] Z. Guo et al. Efficient hardware code generation for fpgas. *ACM Trans. Archit. Code Optim.*, **5**[1]:6:1--6:26, May 2008. 121

[63] M. R. Guthaus et al. Mibench: A free, commercially representative embedded benchmark suite. In *Proc. Int'l Works. Workload Characterization*, pages 3--14, Washington, DC, USA, 2001. IEEE. 43, 89, 198

[64] L.J. Hafer and A.C. Parker. A formal method for the specification, analysis, and design of register-transfer level digital logic. *IEEE Trans. Computer-Aided Design of Integr. Circuits Syst.*, **2**[1]:4--18, January 1983. 149, 171

[65] Emma Hart, Peter Ross, and David Corne. Evolutionary scheduling: A review. *Genetic Programming and Evolvable Machines*, **6**:191--220, 2005. 151

[66] Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, Cambridge, MA, USA, 1st edition, 1995. 180, 192

[67] M.J.M. Heijligers, L.J.M. Cluitmans, and J.A.G. Jess. High-level synthesis scheduling and allocation using genetic algorithms. In *Proc. Asia and South Pacific Conf. Design Automation*, page 11, New York, NY, USA, 1995. ACM. 150, 172

[68] M.J.M. Heijligers and J.A.G. Jess. High-level synthesis scheduling and allocation using genetic algorithms based on constructive topological scheduling techniques. In *Proc. IEEE Int'l Conf. Evolutionary Computation*, **1**, pages 56--61, Los Alamitos, CA, USA, Nov-1 Dec 1995. IEEE Computer Society. 172

[69] A. Hemani and A. Postula. A neural net based self organising scheduling algorithm. In *IEEE Proc. European Conf. Design Automation*, pages 136--140, Los Alamitos, CA, USA, 1990. IEEE Computer Society. 150, 157, 167

[70] J.L. Hennessy et al. *Computer Architecture, Fourth Edition: A Quantitative Approach*. Mor.Kaufmann Pub., San Francisco, CA, USA, 2006. 123

[71] C.Y. Hitchcock and D.E. Thomas. A method of automatic data path synthesis. In *ACM/IEEE Proc. 20th Design Automation Conf.*, pages 484--489, New York, NY, USA, 1983. ACM. 149

[72] Chao-Ju Hou and K.G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. *Trans. Computers*, **46**[12]:1338--1356, Dec 1997. 170, 186

[73] C. Huang et al. Scalable object detection accelerators on fpgas using custom design space exploration. In *Proc. Symp. Application Specific Processors*, pages 115--121, USA, Jun. 2011. IEEE. 4, 121, 217

[74] C.T. Hwang, J.H. Lee, and Y.C. Hsu. A formal approach to the scheduling problem in high level synthesis. *IEEE Trans. Computer-Aided Design of Integr. Circuits Syst.*, **10**[4]:464--475, apr 1991. 149, 150, 152, 168, 170

[75] B. Jang et al. Exploiting memory access patterns to improve memory performance in data-parallel architectures. *Trans. Parallel & Distributed Systems*, **22**:105--118, 2011. 37

[76] T. Janjusic et al. Gleipnir: A memory analysis tool. In *Proc. ICCS*, pages 2058--2067, 2011. 36

[77] P.K. Jha et al. Library mapping for memories. In *Proc. EDAC*, pages 288--, USA, 1997. IEEE. 39

[78] Shiyuan Jin, Guy Schiavone, and Damla Turgut. A performance study of multiprocessor task scheduling algorithms. *J. Supercomput.*, **43**:77--97, January 2008. 161

[79] A. Jones and L.C. Rabelo. Survey of job shop scheduling techniques, 1998. 146, 152, 180, 249

[80] L. Jozwiak et al. Multi-objective optimal controller synthesis for heterogeneous embedded systems. In *Proc. Int'l Conf. EC-SAMOS*, pages 177--184, USA, Jul. 2006. IEEE. 120

[81] M. Kafil and I. Ahmad. Optimal task assignment in heterogeneous computing systems. In *Proc. 6th Work. Heterogeneous Computing*, pages 135--146, Washington, DC, USA, Apr 1997. IEEE Computer Society. 169, 181

[82] Mahmut Taylan Kandemir. A compiler technique for improving whole-program locality. *SIGPLAN Not.*, **36**[3]:179--192, Jan 2001. 37

[83] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302--311, New York, NY, USA, 1984. ACM. 171

[84] Sanza T. Kazadi. Stochastic search methods, Feb 1995. 180

[85] M.Bruynooghe F.Catthoor G.Janssens K.C.Shashidar. An automatic verification technique for loop and data reuse transformations based on geometric modeling of program. *Compiler Construction meets Compiler Verification*, **3**:248--269, 2003. 7, 221

[86] V.I. Kelefouras, G.S. Athanasiou, N. Alachiotis, H.E. Michail, A.S. Kritikakou, and C.E. Goutis. A methodology for speeding up fast fourier transform focusing on memory architecture utilization. *Signal Processing, IEEE Transactions on*, **59**[12]:6217--6226, dec. 2011. 207

[87] Y. Kim et al. Improving performance of nested loops on reconfigurable array processors. *ACM Trans. Archit. Code Optim.*, **8**[4]:32:1--32:23, Jan. 2012. 127

[88] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, **220**[4598]:671--680, May 1983. 171

[89] P. G. Kjeldsberg et al. Storage requirement estimation for optimized design of data intensive applications. *ACM Trans. Des. Autom. Electron. Syst.*, **9**[2]:133--158, Apr 2004. 37

[90] P.G. Kjeldsberg et al. Data dependency size estimation for use in memory optimization. *Trans. Computer-Aided Design of Integrated Circuits and Systems*, **22**:908--921, 2003. 37

[91] David Ryan Koes et al. Near-optimal instruction selection on dags. In *Proc. Int'l Symp. Code Generation & Optimization*, pages 45--54, New York, NY, USA, 2008. ACM. 131

[92] G. Kornaros. A soft multi-core architecture for edge detection & data analysis of microarray images. *J. Syst. Archit.*, **56**:48--62, Jan. 2010. 119, 121, 243

[93] T.J. Kowalski, D.J. Geiger, W.H. Wolf, and W. Fichtner. The vlsi design automation assistant: From algorithms to silicon. *IEEE Des. Test. Comput.*, **2**[4]:33--43, Aug. 1985. 149

[94] A. Kritikakou, F. Catthoor, G.A. Athanasiou, V. Kelefouras, and C. Goutis. A template-based methodology for efficient microprocessor & fpga accelerator co-design. In *Proc. Int'l Conf. EC-SAMOS*, pages 15--22, Los Alamitos, USA, 2012. IEEE. 207

[95] A. Kritikakou, F. Catthoor, G.A. Athanasiou, V. Kelefouras, and C. Goutis. Near-optimal microprocessor & accelerators co-design with latency & throughput constraints. *ACM Trans. Architecture and Code Optimization*, **10**[1], 2013. 207

[96] A. Kritikakou, F. Catthoor, V. Kelefouras, and C. Goutis. Near-optimal & scalable intra-signal in-place for non-overlapping & irregular access scheme. *Trans. Design Automation of Electronic Systems*, [conditionally accepted], 2013. 207

[97] A. Kritikakou, F. Catthoor, V. Kelefouras, and C. Goutis. Scalable & near-optimal array storage size under overlapping & irregular accesses. *IEEE Trans. Computers*, [Submitted], 2013. 207

[98] A. Kritikakou, F. Catthoor, V. Kelefouras, and C. Goutis. A scalable & near-optimal representation for storage size management. *ACM Trans. Architecture and Code Optimization*, [conditionally accepted], 2013. 207

[99] A. Kritikakou, F. Catthoor, V. Kelefouras, and C. Goutis. A systematic approach to classify global scheduling techniques. *ACM J. Computing Surveys*, **45**[2]:14:1--14:30, 2013. 208

[100] A. Kritikakou, F. Catthoor, V. Kelefouras, and C. Goutis. A systematic methodology to develop new global design time scheduling techniques. *ACM Trans. Design Automation of Electronic Systems*, [Submitted], 2013. 208

[101] A. Kritikakou et al. A systematic approach to classify global scheduling techniques. *ACM J. Computing Surveys*, **(to appear)**, 2012. 134

[102] Pavel Krčál et al. Decidable and undecidable problems in schedulability analysis using timed automata. In *Proc. TACAS*, pages 236--250, 2004. 180

[103] Prasad A. Kulkarni et al. In search of near-optimal optimization phase orderings. In *Proc. Conf. Languages, Compilers, and Tools for Embedded Systems*, pages 83--92, 2006. 4, 218

[104] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, **31**:406--471, Dec. 1999. 151, 180

[105] Yu-Kwong Kwok et al. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *Tran. Parallel and Distributed Systems*, **7**[5]:506--521, may 1996. 178, 181

[106] M. Lam. Software pipelining: an effective scheduling technique for vliw machines. *SIGPLAN Not.*, **23**[7]:318--328, Jun. 1988. 124

[107] Chunho Lee et al. Mediabench: a tool for evaluating & synthesizing multimedia & communicatons systems. In *Proc. Int'l Symp. Microarchitecture*, pages 330--335, Washington, USA, 1997. IEEE. 43, 89, 113

[108] LooHay Lee et al. An optimization model for storage yard management in transshipment hubs. In *Container Terminals and Cargo Systems*, pages 107--129. Springer, Berlin, Heidelberg, 2007. 33, 38, 228

[109] W. Lee, R. Barua, M. Frank, D. Srikrishna, J. Babb, V. Sarkar, and S. Amarasinghe. Space-time scheduling of instruction-level parallelism on a raw machine. *ACM SIGOPS Operaing Systems Review*, **33**[11]:46--57, 1998. 150, 158, 169

[110] V. Lefebvre et al. Automatic storage management for parallel programs. *Parallel Comput.*, **24**[3-4]:649--671, May 1998. 37

[111] T. Lewis et al. Parallax: A tool for parallel program scheduling. *Parallel Distributed Technology*, **1**:62--72, May 1993. 180

[112] Yajun Li, Yuhang Yang, Maode Ma, and Rongbo Zhu. A problem-specific genetic algorithm for multiprocessor real-time task scheduling. In *Proc. 3rd Int'l Conf. Innovative Computing Information and Control*, pages 186--, Washington, DC, USA, 2008. IEEE Computer Society. 150, 173, 182, 192

[113] J. Liao et al. A model for hardware realization of kernel loops. In *Proc. Int'l Conf. FPGA*, **2778**, pages 334--344, Berlin, Germany, 2003. Springer. 121

[114] Y.L. Lin. Recent developments in high-level synthesis. *ACM Trans. Des. Autom. Electron. Syst.*, **2**[1]:2--21, 1997. 146, 152, 249

[115] P. E. R. Lippens, J. L. van Meerbergen, W. F. J. Verhaegh, and A. van der Werf. Allocation of multiport memories for hierarchical data stream. In *IEEE/ACM Proc. Int'l Conf. Computer-Aided Design*, pages 728--735, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press. 37, 169

[116] R. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, **4**[2]:4--22, Apr 1987. 157

[117] Y. Liu, X. Zhang, H. Li, and D. Qian. Allocating tasks in multi-core processor based parallel system. In *Proc. IFIP Int'l Conf. Network and Parallel Computing Workshops*, pages 748--753, Washington, DC, USA, 2007. IEEE Computer Society. 170, 181

[118] Chi-Keung Luk et al. Pin: building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not.*, **40**[6]:190--200, Jun 2005. 36

[119] T.A. Ly and J.T. Mowchenko. Applying simulated evolution to high level synthesis. *IEEE Trans. Computer-Aided Design of Integr. Circuits Syst.*, **12**[3]:389--409, Mar 1993. 149, 150, 162, 173, 182

[120] C. Martin. *BANDBX: An Enumeration Code for Pure and Mixed Zero-One Programming Problems*. Industrial and Systems Engineering Dept., Ohio State University, 1978. 171

[121] P. Marwedel. Building a compiler, 2006. 4, 218

[122] P. Marwedel. *Embedded System Design*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2011. 2, 3, 214, 216

[123] Dror E. Maydan et al. Array-data flow analysis and its use in array privatization. In *Proc. Symp. Principles of programming languages*, pages 2--15, New York, NY, USA, 1993. ACM. 37

[124] M.C. McFarland. Using bottom-up design techniques in the synthesis of digital hardware from abstract behavioral descriptions. In *ACM/IEEE Proc. 23rd Design Automation Conf.*, pages 474--480, New York, NY, USA, 1986. ACM. 149

[125] M.C. McFarland, A.C. Parker, and R. Camposano. Tutorial on high-level synthesis. In *ACM/IEEE Proc. 25th Conf. Design Automation*, pages 330--336, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press. 145, 149, 249

[126] M.C. McFarland, A.C. Parker, and R. Camposano. The high-level synthesis of digital systems. *IEEE Proc.*, **78**[2]:301--318, Feb. 1990. 145, 149, 249

[127] D. Melpignano et al. Platform 2012, a many-core computing accelerator for embedded socs: performance evaluation of visual analytics applications. In *DAC*, pages 1137--1142, USA, 2012. ACM. 120

[128] B. Mesman, A.H. Timmer, J.L. van Meerbergen, and J.A.G. Jess. Constraint analysis for dsp code generation. *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, **18**[1]:44--57, Jan 1999. 150, 152, 157, 168

[129] Harris E. Michail, George Athanasiou, Angeliki Kritikakou, Costas E. Goutis, Andreas Gregoriades, and Vicky G. Papadopoulou. Ultra high speed sha256 hashing cryptographic module for ipsec hardware/software codesign. In *Proc. Int'l Conf. Security and Cryptography (SECRYPT)*, pages 309--313, July 2010. 208

[130] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs (2nd, extended ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1994. 172

[131] P. Milder et al. Computer generation of hardware for linear digital signal processing transforms. *ACM Trans. Des. Autom. Electron. Syst.*, **17**[2]:15:1--15:33, Apr. 2012. 119, 243

[132] Narasinga Rao Miniskarg. *System Scenario Based Resource Management of Processing Elements on MPSoC*. PhD thesis, ESAT/EE Dept., K.U.Leuven, Belgium, Dec. 2012. 209

[133] Li Minqiang and Kou Jisong. Phases-based dynamic genetic strategies for genetic algorithms. In *IEEE Int'l Conf. Systems, Man and Cybernetics*, **1**, pages 343--348, Los Alamitos, CA, USA, Oct. 2003. IEEE Computer Society. 161

[134] N. Muscettola. Scheduling by iterative partition of bottleneck conflicts. In *IEEE Proc. Conf. Artificial Intelligence for Applications*, pages 49--55, Los Alamitos, CA, USA, March 1993. IEEE Computer Society Press. 150, 152, 168

[135] L. Nachtergaele et al. Specification and simulation front-end for hardware synthesis of dsp applications. *Int. J. Comp. Simulation*, **2**:213--2291, 1992. 36, 55, 232

[136] M. Narasimhan et al. A fast approach to computing exact solutions to the resource-constrained scheduling problem. *Trans. Des. Autom. Electron. Syst.*, **6**:490--500, Oct. 2001. 181

[137] G. E. Nasr, A. Harb, and G. Meghabghab. Enhanced simulated annealing techniques for multiprocessor scheduling. In *Proc. 12th Int'l Florida Artificial Intelligence Research Society Conf.*, pages 124--128, California, USA, 1999. AAAI Press. 150, 171, 182, 186

[138] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, New York, 1988. 168

[139] N. Nethercote et al. Building workload characterization tools with valgrind, Oct. 2006. 36

[140] B. Neumann et al. Design flow for embedded fpgas based on a flexible architecture template. In *Proc. Conf. Design, Automation & Test in Europe*, pages 56--61, USA, 2008. ACM. 120

[141] S.J. Noronha and V.V.S. Sarma. Knowledge-based approaches for scheduling problems: A survey. *IEEE Trans. Knowl. Data Eng.*, **3**[2]:160--171, 1991. 145, 151, 248, 249

[142] D. Novo et al. Ultra low energy domain specific instruction-set processor for on-line surveillance. In *Proc. SASP*, pages 30--35, Los Alamitos, CA, USA, Jun.e 2010. IEEE Computer Society Press. 126

[143] Yunheung Paek et al. Efficient & precise array access analysis. *TOPLAS*, **24**[1]:65--109, Jan 2002. 36, 37

[144] E.S. Page. On monte carlo methods in congestion problems: I. searching for an optimum in discrete situations. *Operations Research*, **13**[2]:291--199, 1965. 150, 163, 174, 192

[145] P. Palazzari, L. Baldini, and M. Coli. Synthesis of pipelined systems for the contemporaneous execution of periodic and aperiodic tasks with hard real-time constraints. *Proc. Int'l Symp. Parallel and Distributed Processing*, **3**:121a, 2004. 150, 172, 182

[146] K.V. Palem et al. Design space optimization of embedded memory systems via data remapping. In *Proc. Conf. Languages, Compilers & Tools for Embedded Systems*, pages 28--37, USA, 2002. ACM. 36

[147] G. Palermo et al. Multi-objective design space exploration of embedded systems. *J. Embedded Comput.*, **1**:305--316, Aug. 2005. 4, 5, 121, 217, 218

[148] P. R. Panda et al. Data and memory optimization techniques for embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, **6**[2]:149--206, Apr 2001. 37

[149] P.R. Panda and N.D. Dutt. Low-power memory mapping through reducing address bus activity. *IEEE Trans. Very Large Scale Integr.(VLSI) Syst.*, **7**[3]:309--320, 1999. 169

[150] A.C. Parker, J. Pizarro, and M. Mlinar. Maha: A program for datapath synthesis. In *ACM/IEEE Proc. Conf. Design Automation*, pages 461--466, Piscataway, NJ, USA, June 1986. IEEE Press. 150, 152, 158, 169

[151] P.G. Paulin and J.P. Knight. Force-directed scheduling in automatic data path synthesis. In *ACM/IEEE Proc. 24th Design Automation Conf.*, pages 195--202, New York, NY, USA, 1987. ACM. 145, 149, 151, 249

[152] P.G. Paulin and J.P. Knight. Force-directed scheduling for the behavioral synthesis of asics. *IEEE Trans. Computer-Aided Design of Integr. Circuits Syst.*, **8**[6]:661--679, jun 1989. 145, 151, 249

[153] C.S. Pedamallu et al. Investigating a hybrid simulated annealing and local search algorithm for constrained optimization. *J. Op. Res.*, **185**[3]:1230--1245, 2008. 182

[154] Z. Peng. Synthesis of vlsi systems with the camad design aid. In *ACM/IEEE Proc. 23rd Design Automation Conf.*, pages 278--284, Piscataway, NJ, USA, 1986. IEEE Press. 149

[155] D. Piriyakumar, Paul Levi, and C. Murthy. Optimal scheduling of iterative data-flow programs onto multiprocessors with non-negligible interprocessor communication. In *High-Performance Computing and Networking*, **1593** of *Lecture Notes in Computer Science*, pages 732--743. University of Stuttgart Institute of Parallel and Distributed High-Performance Systems, Stuttgart, Germany, 1999. 170, 181

[156] Douglas Antony Louis Piriyakumar, C. Siva Ram Murthy, and Paul Levi. A new a* based optimal task scheduling in heterogeneous multiprocessor systems applied to computer vision. In *Proc. Int'l Conf. & Exh. High-Performance Computing and Networking*, pages 315--323, London, UK, 1998. Springer-Verlag. 170, 181

[157] Massimiliano Poletto et al. Linear scan register allocation. *ACM Trans. Program. Lang. Syst.*, **21**[5]:895--913, Sep. 1999. 132

[158] C.D. Polychronopoulos et al. The structure of parafrase-2: an advanced parallelizing compiler for c & fortran. In *LCPC*, pages 423--453, 1990. 180

[159] Luis-Noel Pouchet et al. Polybenchmarks benchmark suite. `http://www.cse.ohio-state.edu/~pouchet/software/polybench/`, 2012. 43, 89, 113, 120, 141, 247

[160] J. Ramanujam et al. Reducing memory requirements of nested loops for embedded systems. In *Proc. Conf. Design Automation*, pages 359--364, New York, NY, USA, 2001. ACM. 34, 37, 42, 55, 89, 113, 229, 232

[161] Miniskar Narasinga Rao. *System Scenario Based Resource Management of Processing Elements on MPSoC*. PhD thesis, KULeuven, 2012. 39

[162] H. Edwin Romeijn et al. Simulated annealing and adaptive search in global optimization. *Prob.Eng.&Inf.Sciences*, **8**[1]:571--590, 1994. 194

[163] F. Romeo and A. Sangiovanni-Vincentelli. Probabilistic hill climbing algorithms: Properties and applications. In *Proc. Chapel Hill Conf. Very Large Scale Integration*, pages 393--417, New York, USA, Dec 1985. Computer Science Press. 159

[164] S. Rubin et al. An efficient profile-analysis framework for data-layout optimizations. *SIGPLAN Not.*, **37**:140--153, Jan. 2002. 36

[165] A. Safir and B. Zavidovique. On the synthesis of specific image processing automata by a simulated annealing-based design space search. In *IEEE Proc. Int'l Symp. Circuits and Systems)*, **2**, pages 1374--1377, Los Alamitos, CA, USA, May 1989. IEEE Computer Society. 171

[166] A. Safir and B. Zavidovique. On the synthesis of specific image processing automata from emulation results. In *IEEE Proc. European Conf. Application Specific Integrated Circuits*, pages 104--115, Los Alamitos, CA, USA, Jan 1989. IEEE Computer Society. 171

[167] A. Safir and B. Zavidovique. Towards a global solution to high level synthesis problems. In *IEEE Proc. European Conf. Design Automation*, pages 283--288, Los Alamitos, CA, USA, 1990. IEEE Computer Society. 149, 156, 171

[168] S.M. Sait, S.Ali, and M.S. Benten. Scheduling and allocation in high-level synthesis using stochastic techniques. *Microelectronics Journal*, **27**[8]:693--712, 1996. 150, 160, 161, 173, 182, 192

[169] M. Sami et al. An instruction-level energy model for embedded vliw architectures. *Trans. CAD*, **21**:998--1010, 2002. 177

[170] R.E. Sant'Anna et al. A left-edge algorithm approach for scheduling & allocation of hw contexts in dynamically reconfigurable architectures. In *FPGA*, pages 259--259, USA, 2004. ACM. 132

[171] F. Schoen. Stochastic techniques for global optimization: A survey of recent advances. *Journal of Global Optimization*, **1**:207--228, 1991. 145, 151, 249

[172] Fabio Schoen. Stochastic techniques for global optimization: A survey of recent advances. *J. Global Optimization*, **1**:207--228, 1991. 190

[173] Rachid Seghir et al. Integer affine transformations of parametric polytopes and applications to loop nest optimization. *ACM Trans. Archit. Code Optim.*, **9**[2]:8:1--8:27, Jun 2012. 37

[174] M. Shahzad et al. Image coprocessor: A real-time approach towards object tracking. In *Proc. Int'l Conf. Digital Image Processing*, pages 220--224, USA, Mar. 2009. IEEE. 121

[175] D. Sheldon et al. Application-specific customization of parameterized fpga soft-core processors. In *Proc. Int'l Conf. Computer-Aided Design*, pages 261--268, USA, Nov. 2006. IEEE. 3, 121, 216

[176] D. Sheldon et al. Making good points: application-specific pareto-point generation for dse using statistical methods. In *Proc. Int'l Symp. FPGA*, pages 123--132, NY, USA, 2009. ACM. 5, 119, 121, 218, 243

[177] Z. Shen, Z. Li, and P.C. Yew. An empirical study of fortran programs for parallelizing compilers. *IEEE Transactions on Parallel and Distributed Systems*, **1**:356--364, 1990. 37

[178] V. Shestak et al. A hybrid branch & bound & evolutionary approach for allocating strings of applications to heterogeneous distributed computing systems. *Par. Distr. Comp.*, **68**:410--426, Apr. 2008. 182

[179] H. Shin and N.S. Woo. A cost function based optimization technique for scheduling in data path synthesis. In *IEEE Proc. Int'l Conf. Computer Design: VLSI in Computers and Processors*, pages 424--427, Los Alamitos, CA, USA, Oct 1989. IEEE Computer Society Press. 150, 152, 157, 167

[180] Behrooz Shirazi et al. Parsa: A parallel program scheduling and assessment environment. In *Proc. Parallel Processing*, **2**, pages 68--72, aug. 1993. 180

[181] Frank Singhoff et al. Investigating the usability of real-time scheduling theory with cheddar project. *Real-Time Sys.*, **43**:259--295, 2009. 175, 179, 250

[182] S.F. Smith. Is scheduling a solved problem? In *Proc. Multi-Disciplinary Int'l Conf. Scheduling Theory and Applications*, pages 3--18, Nottingham, UK, 2003. Sherwood Press. 145, 146, 152, 175, 249, 250

[183] B. So et al. Custom data layout for memory parallelism. In *CGO*, pages 291--, USA, 2004. IEEE. 36

[184] R.H. Storer, S.D. Wu, and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *INFORMS Management Science*, **38**[10]:1495--1509, 1992. 150, 172

[185] SUIF. Suif compiler system, 2012. 180

[186] Synopsys. Synopsys synphony -- high level synthesis solution. `http://www.synopsys.com`, 2012. 120

[187] D.E. Thomas, C.Y. III Hitchcock, T.J. Kowalski, J.V. Rajan, and R.A. Walker. Automatic data path synthesis. *IEEE Computer*, **16**[12]:59--70, 1983. 149

[188] Sid-Ahmed-Ali Touati et al. On the decidability of phase ordering problem in optimizing compilation. In *Proc. Conf. Computing frontiers*, pages 147--156, New York, NY, USA, 2006. ACM. 4

[189] E.P.K. Tsang. Scheduling techniques -- a comparative study. *British Telecom Technology Journal*, **13**[1]:16--28, 1995. 156

[190] C.J. Tseng and D.P. Siewiorek. Automated synthesis of data paths in digital systems. *IEEE Trans. Computer-Aided Design of Integr. Circuits Syst.*, **5**[3]:379--395, 1986. 149

[191] D. Tsitsipis, S. Dima, A. Kritikakou, C. Panagiotou, and S. Koubias. Data merge: A data aggregation technique for wireless sensor networks. In *Proc. Int'l Conf. Emerging Technologies Factory Automation (ETFA)*, pages 1--4, sept. 2011. 208

[192] D. Tsitsipis, S.M. Dima, A. Kritikakou, C. Panagiotou, John Gialelis, Harris Michail, and S. Koubias. Priority handling aggregation technique (phat) for wireless sensor networks. In *Proc. Int'l Conf. Emerging Technologies Factory Automation (ETFA)*, Sep. 2012. 208

[193] D. Tsitsipis, S.M. Dima, A. Kritikakou, C. Panagiotou, and S. Koubias. Segmentation and reassembly data merge (sardam) technique for wireless sensor networks. In *Proc. Int'l Conf. Industrial Technology (ICIT)*, pages 1014--1019, march 2012. 208

[194] R. Upadrasta et al. Potential and challenges of two-variable-per-inequality sub-polyhedral compilation. 33, 229

[195] M. van Swaaij et al. Automating high level control flow transformations for dsp memory management. In *Proc. Work. VLSI Signal Processing*, pages 397--406, USA, 1992. IEEE. 37

[196] N. Vassiliadis et al. The arise approach for extending embedded processors with arbitrary hardware accelerators. *Trans. VLSI*, **17**[2]:221--233, Feb. 2009. 120

[197] G. Payá Vayá, J. Martín-Langerwerf, P. Taptimthong, and P. Pirsch. Design space exploration of media processors: A parameterized scheduler. In *IEEE Proc. Int'l Conf. Embedded Computer Systems: Architectures, Modeling and Simulation*, pages 41--49, Los Alamitos, CA, USA, 2007. IEEE Computer Society Press. 170

[198] W.F.J. Verhaegh, P.E.R. Lippens, E.H.L. Aarts, J.H.M. Korst, A. van der Werf, and J.L. van Meerbergen. Efficiency improvements for force-directed scheduling. In *IEEE/ACM Proc. Int'l Conf. Computer-Aided Design*, pages 286--291, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press. 149

[199] R.A. Walker and S. Chaudhuri. Introduction to the scheduling problem. *Design Test of Computers, IEEE*, **12**[2]:60--69, 1995. 181

[200] Gang Wang et al. Operation scheduling: Algorithms & applications high-level synthesis. In *High-Level Synthesis*, pages 231--255. Springer, 2008. 178, 181

[201] N. Wehn, M. Held, and M. Glesner. A novel scheduling and allocation approach for datapath synthesis based on genetic paradigms. In *Proc. IFIP Working Conf. Logic and Architecture Synthesis*, pages 47--56, Washington, DC, USA, 1991. IEEE Computer Society. 172

[202] J. Weidendorfer et al. A tool suite for simulation based analysis of memory access behavior. In *Proc. Int'l Conf. Computational Science*, pages 440--447. Springer, 2004. 36

[203] E. Weisstein et al. Wolfram mathworld: Affine. `http://mathworld.wolfram.com/Affine.html`, 2012. 41

[204] B.E. Wells et al. An augmented approach to task allocation: combining simulated annealing with list-based heuristics. In *Proc.Work.Par.& Distr.Proc.*, pages 508--515, Jan 1993. 182

[205] Min-You Wu et al. Hypertool: A programming aid for message-passing systems. *Trans. Parallel & Distr.Systems*, **1**:330--343, 1990. 180

[206] Sven Wuytack et al. Formalized methodology for data reuse exploration for low-power hierarchical memory mappings. *Trans. Very Large Scale Integr. Syst.*, **6**:529--537, Dec. 1998. 37

[207] Xilinx. Logicore ip multi-port memory controller, Mar. 2011. 129

[208] J. Xu and D. L. Parnas. On satisfying timing constraints in hard-real-time systems. *IEEE Trans. Softw. Eng.*, **19**:70--84, Jan. 1993. 148

[209] Jia Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. Softw. Eng.*, **19**:139--154, 1993. 170, 186

[210] Tao Yang et al. Pyrros:static task scheduling&code generation for message passing multiprocessors. In *Supercomp.*, pages 428--437, 1992. 180

[211] A. Yarkhan and J.J. Dongarra. Experiments with scheduling using simulated annealing in a grid environment, 2002. 172, 182, 190

[212] Z.B. Zabinsky et al. *Complexity Analysis Integrating Pure Adaptive Search & Pure Random Search*, pages 171--182. Kluwer Academic Publ., 1997. 192

[213] R. Zhou and E.A. Hansen. Beam-stack search: Integrating backtracking with beam search. In *Proc. Int'l Conf. Automated Planning and Scheduling*, pages 90--98, California, USA, 2005. AAAI. 152, 170, 181

[214] X. Zhuang et al. A framework for parallelizing load/stores on embedded processors. In *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques*, pages 68--, USA, 2002. IEEE. 37

[This page is intentionally left blank]