OULU 2012
C 435

ACTA UNIVERSITATIS OULUENSIS

Jari Kreku

EARLY-PHASE PERFORMANCE EVALUATION OF COMPUTER SYSTEMS USING WORKLOAD MODELS AND SystemC

UNIVERSITY OF OULU GRADUATE SCHOOL;
UNIVERSITY OF OULU,
FACULTY OF TECHNOLOGY,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ACTA UNIVERSITATIS OULUENSIS
C Technica 435

JARI KREKU

EARLY-PHASE PERFORMANCE EVALUATION OF COMPUTER SYSTEMS USING WORKLOAD MODELS AND SystemC

Academic dissertation to be presented with the assent of the Doctoral Training Committee of Technology and Natural Sciences of the University of Oulu for public defence in Auditorium TS101, Linnanmaa, on 14 December 2012, at 12 noon

UNIVERSITY OF OULU, OULU 2012

Acta Univ. Oul. C 435, 2012

Supervised by
Professor Juha Röning

Reviewed by
Associate Professor Antonio Pimentel
Professor Eugenio Villar

ISBN 978-951-42-9989-6 (Paperback)
ISBN 978-951-42-9990-2 (PDF)

ISSN 0355-3213 (Printed)
ISSN 1796-2226 (Online)

Cover Design
Raimo Ahonen

JUVENES PRINT
TAMPERE 2012

Kreku, Jari, Early-phase performance evaluation of computer systems using workload models and SystemC.
University of Oulu Graduate School; University of Oulu, Faculty of Technology, Department of Computer Science and Engineering, P.O. Box 4500, FI-90014 University of Oulu, Finland
Acta Univ. Oul. C 435, 2012
Oulu, Finland

Abstract

Novel methods and tools are needed for the performance evaluation of future embedded systems due to the increasing system complexity. Systems accommodate a large number of on-terminal and or downloadable applications offering the users with numerous services related to telecommunication, audio and video, digital television, internet and navigation. More flexibility, scalability and modularity is expected from execution platforms to support applications. Digital processing architectures will evolve from the current system-on-chips to massively parallel computers consisting of heterogeneous subsystems connected by a network-on-chip. As a consequence, the overall complexity of system evaluation will increase by orders of magnitude.

The ABSOLUT performance simulation approach presented in this thesis combats evaluation complexity by abstracting the functionality of the applications with workload models consisting of instruction-like primitives. Workload models can be created from application specifications, measurement results, execution traces, or the source code. Complexity of execution platform models is also reduced since the data paths of processing elements need not be modelled in detail and data transfers and storage are simulated only from the performance point of view. The modelling approach enables early evaluation since mature hardware or software is not required for the modelling or simulation of complete systems.

ABSOLUT is applied to a number of case studies including mobile phone usage, MP3 playback, MPEG4 encoding and decoding, 3D gaming, virtual network computing, and parallel software-defined radio applications. The platforms used in the studies represent both embedded systems and personal computers, and at the same time both currently existing platforms and future designs. The results obtained from simulations are compared to measurements from real platforms, which reveals an average difference of 12% in the results. This exceeds the accuracy requirements expected from virtual system-based simulation approaches intended for early evaluation.

Keywords: application, architecture, capacity, computer, embedded, evaluation, hardware, modelling, performance, simulation, software, system, workload

Kreku, Jari, Tietokonejärjestelmien varhaisen vaiheen suorituskykyevaluointi käyttäen työkuormamalleja ja SystemC:tä.
Oulun yliopiston tutkijakoulu; Oulun yliopisto, Teknillinen tiedekunta, Tietotekniikan osasto, PL 4500, 90014 Oulun yliopisto
Acta Univ. Oul. C 435, 2012
Oulu

Tiivistelmä

Sulautettujen tietokonejärjestelmien suorituskyvyn arviointi muuttuu yhä haastavammaksi järjestelmien kasvavan kompleksisuuden vuoksi. Järjestelmissä on suuri määrä sovelluksia, jotka tarjoavat käyttäjälle palveluita liittyen esimerkiksi telekommunikaatioon, äänen ja videokuvan toistoon, internet-selaukseen ja navigaatioon. Tästä johtuen suoritusalustoilta edellytetään yhä enemmän joustavuutta, skaalautuvuutta ja modulaarisuutta. Suoritusarkkitehtuurit kehittyvät nykyisistä System-on-Chip (SoC) -ratkaisuista Network-on-Chip (NoC) -rinnakkaistietokoneiksi, jotka koostuvat heterogeenisistä alijärjestelmistä. Sovellusten ja suoritusalustan muodostaman järjestelmän suorituskyvyn arviointiin tarvitaan uusia menetelmiä ja työkaluja, joilla kompleksisuutta voidaan hallita.

Tässä väitöskirjassa esitettävä ABSOLUT-simulointimenetelmä pienentää suorituskyvyn arvioinnin kompleksisuutta abstrahoimalla sovelluksen toiminnallisuutta työkuormamalleilla, jotka koostuvat kuormaprimitiiveistä suorittimen käskyjen sijaan. Työkuormamalleja voidaan luoda sovellusten spesifikaatioista, mittaustuloksista, suoritusjäljistä tai sovellusten lähdekoodeista. Suoritusalustoista ABSOLUT-menetelmä käyttää yksinkertaisia kapasiteettimalleja toiminnallisten mallien sijaan: suoritinarkkitehtuurit mallinnetaan korkealla tasolla ja tiedonsiirto ja tiedon varastointi mallinnetaan vain suorituskyvyn näkökulmasta. Menetelmä mahdollistaa aikaisen suorituskyvyn arvioinnin, koska malleja voidaan luoda ja simuloida jo ennen valmiin sovelluksen tai suoritusalustan olemassaoloa.

ABSOLUT-menetelmää on käytetty useissa erilaisissa kokeiluissa, jotka sisälsivät esimerkiksi matkapuhelimen käyttöä, äänen ja videokuvan toistoa ja tallennusta, 3D-pelin pelaamista ja digitaalista tiedonsiirtoa. Esimerkeissä käytetiin tyypillisiä suoritusalustoja sekä kotitietokoneiden että sulautettujen järjestelmien maailmasta. Lisäksi osa esimerkeistä pohjautui tuleviin tai keksittyihin suoritusalustoihin. Osa simuloinneista on varmennettu vertaamalla simulointituloksia todellisista järjestelmistä saatuihin mittaustuloksiin. Niiden välillä huomattiin keskimäärin 12 prosentin poikkeama, mikä ylittää aikaisen vaiheen suorituskyvyn simulointimenetelmiltä vaadittavan tarkkuuden.

Asiasanat: arviointi, järjestelmä, kapasiteetti, mallinnus, ohjelmisto, simulointi, sovellus, sulautettu, suorituskyky, tietokone, tietokonearkkitehtuuri, työkuorma

Dedicated to my grandparents

Preface

The research for this thesis was performed at VTT technical research centre of Finland since 2003. The work was started in the BAPPEA2 project and since then has involved several others, namely ASTERIX, WIMPAIN, MARTES and MOSART in the past. The development of the ABSOLUT approach continues in the SMECY and PRESTO projects even after the writing of the thesis. The writing of this thesis was started with the literature review in Chapter 2 in summer 2010 and continued intensively with the other parts during spring and summer 2011.

Several people have provided a considerable amount of assistance during the research work. Research Professor Juha-Pekka Soininen and Mr. Kari Tiensyrjä deserve a special thank you for all the encouragement, guidance and general help they have given me since I started at VTT as a research trainee in 1999. Docent Martti Forsell has steered me to the correct direction during the early phases of the work. Research Professor Aarne Mämmelä has provided invaluable advice on good scientific habits during his lectures in the internal meetings at VTT.

I would like to thank all the scientists that have contributed to ABSOLUT in one way or another: Mr. Matti Eteläperä, Mr. Mika Hoppari, Ms. Tarja Kauppi, Mr. Tuomo Kestilä, Mr. Jani Penttilä, Dr. Yang Qu, Mr. Jukka Saastamoinen and Mr. Tommi Salminen. Without your support in the development of the modelling approach and / or validation of the case studies this thesis would not be complete. I would like to further thank the co-authors of the original papers not yet mentioned: Mr. Janne Kangas and Mr. Geert Vanmeerbeeck.

I would like to thank Professor Juha Röning for supervising this thesis and assisting with the completion of my dissertation. I greatly appreciate the work of the reviewers, Assistant Professor Antonio Pimentel and Professor Eugenio Villar, for providing helpful advice on improving the thesis and giving me ideas for future research work.

VTT has provided me with the research facilities and work environment for the research and has supported me by allowing me to go on a writing leave for the completion of the thesis. I would like to thank everyone involved. Nokia Foundation has provided me with financial support, which has been important for encouragement and motivation.

I would especially like to thank Anu Peräniemi for all the encouragement, love and support and understanding. Finally, I wish to thank my parents, sister and all the other family members for support through the years.

And now for something completely different.

Oulu, 10 August 2012 Jari Kreku

Abbreviations

ABSOLUT	*Abstract Instruction Workload and Execution Platform-based performance simulation*
ABSINTH	*ABStract INstruction exTraction Helper*
ALE	*ABSOLUT Library of Existing models*
ARCHER	*ARCHitecture ExploRation of heterogeneous embedded systems*
Artemis	*Architectures and Methods for Embedded Media Systems*
API	*Application Programming Interface*
ARTS	*Abstract system-level modelling and simulation framework*
ASIC	*Application-Specific Integrated Circuit*
ASIP	*Application-Specific Instruction-Set Processor*
AV	*Architect's View*
BCET	*Best-Case Execution Time*
BEER	*Binary pErformance EvaluatoR*
CAG	*Communication Analysis Graph*
COGNAC	*COnfiguration GeNerator for ABSOLUT performanCe simulation*
CPI	*Cycles Per Instruction*
CPU	*Central Processing Unit*
DCT	*Discrete Cosine Transform*
DESERT	*Design space exploration tool*
DMA	*Direct Memory Access*
DME	*Distributed shared Memory Engine*
DRAM	*Dynamic RAM*
DSP	*Digital Signal Processor*
FIFO	*First In, First Out*
FPS	*Frames Per Second*
FV	*Functional View*
GCC	*GNU Compiler Collection*
GIPS	*Giga Instructions Per Second*
GNU	*GNU's Not Unix*
GPRS	*General Packet Radio Service*
GPS	*Global Positioning System*

GSM	*Global System for Mobile communications*
HDL	*Hardware Description Language*
HiPerE	*High-level Performance Evaluator*
HW	*Hardware*
I/O	*Input / Output*
IP	*Intellectual Property*
ISS	*Instruction Set Simulation*
JPEG	*Joint Picture Experts Group*
LAN	*Local Area Network*
MBD	*Model-Based Design*
MCU	*MicroController Unit*
MESH	*Modelling Environment for Software and Hardware*
MILAN	*Model-based integrated simulation framework*
MMS	*Multimedia Messaging Service*
MP3	*MPEG1 audio layer 3*
MPA	*MPSoC Parallelization Assist*
MPEG	*Moving Picture Experts Group*
MPEG1	*MPEG phase 1*
MPEG4	*MPEG phase 4*
MPSoC	*Multiprocessor SoC*
MVP	*Mobile Video Player*
NoC	*Network-on-Chip*
NoTA	*Network on Terminal Architecture*
OCP	*Open Core Protocol*
OMAP	*Open Multimedia Application Platform*
OS	*Operating System*
OSCI	*Open SystemC Initiative*
OSK	*OMAP Starter Kit*
PC	*Personal Computer*
PHY	*PHYsical layer*
PDA	*Personal Digital Assistant*
PV	*Programmer's View*
QN	*Queuing Networks*
RAM	*Random Access Memory*
ReSP	*Reflective Simulation Platform*

RISC	*Reduced Instruction Set Computer*
RTL	*Register-Transfer Level*
RTL	*Register-Transfer Language*
RTOS	*Real-Time Operating System*
SA	*System Architecting*
SAD	*Sum of Absolute Differences*
SAKE	*abStract externAl library worKload Extractor*
SD	*Secure Digital*
SDL	*Specification and Description Language*
SDR	*Software-Defined Radio*
SDRAM	*Synchronous DRAM*
SESAME	*Simulation of Embedded System Architectures for Multilevel Exploration*
SMS	*Short Message Service*
SoC	*System-on-Chip*
SPADE	*System-level Performance Analysis and Design space Exploration*
SPU	*Symbolic Program Unit*
SRAM	*Static RAM*
SW	*Software*
TAPES	*Trace-based Architecture Performance Evaluation with SystemC*
TBM	*Timed Behavioural Modelling*
TLM	*Transaction-Level Modelling*
UI	*User Interface*
UML	*Unified Modelling Language*
USB	*Universal Serial Bus*
VHDL	*VHSIC HDL*
VHSIC	*Very High Speed Integrated Circuit*
VNC	*Virtual Network Computing*
VODKA	*Viewer Of collecteD Key information for Analysis*
VPU	*Virtual Processing Unit*
VV	*Verification View*
WCET	*Worst-Case Execution Time*
WiMAX	*Worldwide Interoperability for Microwave Access*
WLAN	*Wireless LAN*
XML	*eXtensible Markup Language*

List of original articles

I Kreku J, Hoppari M, Kestilä T, Qu Y, Soininen J-P, Andersson P & Tiensyrjä K (2008) Combining UML2 Application and SystemC Platform Modelling for Performance Evaluation of Real-Time Embedded Systems. EURASIP Journal on Embedded Systems. DOI: 10.1155/2008/712329.

II Kreku J, Hoppari M, Kestilä T, Qu Y, Soininen J-P & Tiensyrjä K (2009) Application Workload and SystemC Platform Modeling for Performance Evaluation. Best of FDL 2009 book.

III Kreku J & Tiensyrjä K (2011) Scalable Multi-core Architectures: Design Methodologies and Tools, chapter System exploration. Springer.

IV Kreku J, Penttilä J, Kangas J & Soininen J-P (2004) Workload simulation method for evaluation of application feasibility in a mobile multiprocessor platform. Proceedings of the Euromicro Symposium on Digital System Design. Rennes, 31 Aug.–3 Sept. 2004 . IEEE Computer Society: 532–539. DOI: 10.1109/DSD.2004.1333322.

V Kreku J, Kauppi T & Soininen J-P (2004) Evaluation of platform architecture performance using abstract instruction-level workload models. International Symposium on System-on-Chip. Tampere, 16–18 Nov. 2004. Tampere University of Technology: 43–48.

VI Kreku J, Tiensyrjä K & Vanmeerbeeck G (2010) Automatic workload generation for system-level exploration based on modified GCC compiler. Proc. Design, Automation and Test in Europe conference and exhibition.

VII Kreku, Eteläperä M & Soininen J-P (2005) Exploitation of UML 2.0-based platform service model and systemC workload simulation in MPEG-4 partitioning. International Symposium on System-on-Chip. Tampere, 15–17 Nov. 2005. Tampere University of Technology: 167–170.

VIII Kreku J, Hoppari M, Tiensyrjä K & Andersson P (2007) SystemC workload model generation from UML for performance simulation. Forum on Specification and Design Languages. FDL, Barcelona, Spain, 15–18 Sep. 2007. ECSI, Grenoble.

IX Kreku J, Hoppari M, Kestilä T, Qu Y, Soininen J-P & Tiensyrjä K (2008) Application – Platform Performance Modeling and Evaluation. Forum on Specification, Verification and Design Languages, 2008. FDL 2008. 23–25 Sept. 2008: 43–48.

Contents

1 Introduction

Only ten years ago, services provided by mobile devices were much more restricted than now. A mobile phone was used to call other people, a portable music player to listen to music, and perhaps a separate PDA was used for electronic calendar and notes. Many of the current smartphones and other embedded mobile devices offer, for example, telephony, music and movie playing, digital television, internet browsing, and GPS navigation capabilities.

The applications in embedded systems are not fixed on system design time anymore. Instead, they accommodate a large number of on-terminal and/or downloadable applications, which offer the user numerous services. The service contents may be provided by servers anywhere on Earth. The sets and types of applications running on the terminal are dependent on the context of the user. To deliver the requested services to the user, some of the applications run sequentially and independently, while many others execute concurrently and interact with each other. The converging functionality trend will continue and bring more and more features from desktops and other home devices to battery-powered handheld devices. These features can include, for example, live TV functionality, 3D surround sound, and electronic money (Weber (2007)).

The increasing complexity and diversification in applications introduces new requirements for execution platforms, including flexibility, scalability and modularity (Suoranta (2006)). The digital processing architectures of terminals will evolve from the current system-on-chips (SoCs) and multiprocessor-SoCs with a few processor cores to massively parallel computers that consist mostly of heterogeneous subsystems but may also contain homogeneous computing subsystems. Some subsystems will be designed by the device vendor, whereas others will be provided by third parties. The network-on-chip (NoC) communication paradigm will replace the bus-based communication to allow scalability, but it will increase uncertainties due to latencies in case large centralised or external memories are required.

The application and platform development trends will increase the overall complexity of system development by orders of magnitude especially from system-level exploration and performance evaluation point of views. Performance, energy consumption and cost of the battery-powered devices must be optimised while not forgetting the expectations of users. The costs, risks, and time of system development requires flexibility, which will

be achieved through programmable and adaptable computing resources, configurable memory and communication architecture, and a design methodology approach. Tools that span from application use cases to implementation design will be needed.

Design methodology approaches for mobile devices have evolved from the application-specific integrated circuit (ASIC) style in the 1980s to platform-based design in late 1990s. During the 2000s model-based design (MBD) has been introduced. In the ASIC style, designers take an architectural specification and create a microarchitecture description. It will be synthesised and optimised for speed (clock frequency), area (gate count), and power (e.g., modes and clock gating).

Platform-based design addresses the challenges of increasing design complexity of SoCs that consist typically of a few processor cores, hardware accelerators, memories, and I/O peripherals communicating through a shared bus. While the emphasis is on intellectual property (IP) design and integration, the function–architecture co-design and microarchitecture exploration already pave the way for the model-based approach that is the research direction today. To alleviate the scalability problems of the shared bus, the NoC architecture paradigm proposes a communication-centric approach for systems requiring multiple processors or integration of multiple SoCs.

Model-based approaches extend the separation of the application and execution platform modelling further. Usually, the specify–explore–refine paradigm following the principles of the Y-chart model (Kienhuis *et al.* (1997)) is applied. In other words, a model of application is mapped onto a model of platform, and the resulting allocated model is analysed. The computation and communication modelling are separated on both the application and platform sides. Service orientation is a recent trend where the end user interactions and the associated applications are modelled in terms of services required from the underlying execution platform (Network on Terminal Architecture). An obvious consequence is that the execution platform also needs to be modelled in terms of services it provides for the applications.

The application and platform designers are facing an abundant number of design alternatives and need systematic approaches for the exploration of the design space. For example, an application designer has to know early whether a new application or a feature is feasible on the target platform. A platform designer must be able to analyse the impacts of next generation applications on the platform even before the applications are implemented. Efficient methods and tools for early system-level performance analysis are necessary to avoid wrong decisions at the critical stage of system development.

1.1 Problem definition

There are three typical use cases for performance evaluation in the design phase of embedded systems. Performance evaluation is used to compare a number of alternative platforms and find the best one, to search for the best application out of several for a given platform, and to design a new system, platform or application. The goal of the designer is to provide the highest achievable performance at the lowest possible cost (Jain (1991)). The most significant problem with the performance evaluation of future embedded systems is the complexity of the evaluation.

First, the applications and supporting software are increasing in size and variety. Linux is a popular choice for an operating system in embedded system devices. For eaxample, Google's Android and Intel's and Nokia's MeeGo are based on the Linux kernel. Several other vendors use Linux in a variety of devices ranging from home cinema amplifiers to coffee makers. Figure 1 displays the size of the desktop and server computer-oriented Debian Linux distribution as a function of time (DistroWatch.com, Debian Counting). The number of software packages, i.e. applications and libraries, has increased tenfold in ten years. The number of source code lines in those applications has experienced a similar increase during the same time frame. The Linux distributions used in embedded systems are more streamlined than Debian but likely to follow the same trend. On the other hand, Deshpande & Riehle (2008) claim that the number of software projects and total lines of source code is increasing at an exponential rate. In addition to the convergence in functionality and the resulting increase in the number of applications, the size of the individual applications is also growing due to the constant need of adaptation (Turski (1996)). According to a large-scale study performed on open source software in Koch (2007), the evolutionary behaviour of software projects follows a quadratic model.

Second, the number of components and number of transistors in each component in the platforms is growing. ITRS predicts that the number of processing engines in platforms is increasing 1.4-fold each year (Figure 2). The underlying fabrics — logic, embedded memory, on-chip switching fabric, and system interconnect — will scale consistently with the increase in the number of cores. More and more performance can be extracted from individual components, thereby providing more capacity for applications to utilise. As ASIC technology progresses, more transistors fit in the same area. Historically, the number of transistors in microprocessors has increased

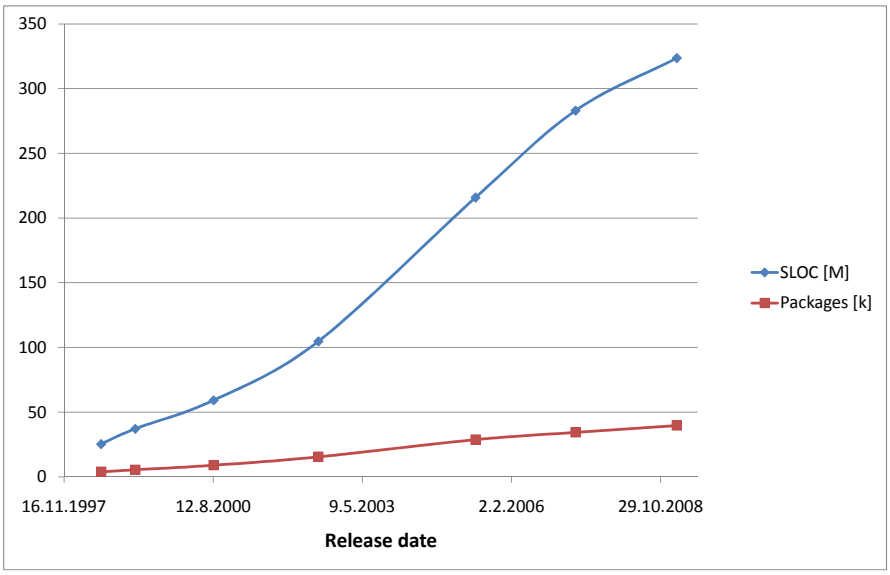

Fig 1. The size of Debian Linux distribution releases from Debian 2.0 to Debian 5.0.

exponentially since the first singe-chip CPU, Intel 4004. Furthermore, the clock frequencies of processor cores and accelerators is increasing by 1.05x per year (ITRS).

Enormous system complexity can be realised on a single die, but exploiting this potential reliably and cost-effectively will require a roughly 50-fold increase in design productivity over what is possible today (ITRS). Performance evaluation models are required to capture both the characteristics of the application functionality and the architectural resources needed for the execution. Using models at a too low level of abstraction, for example, register-transfer level (RTL) or instruction set simulation (ISS) is not feasible. Even though they are able to provide accurate results, the modelling effort is heavy and simulation times long due to the vast amount of details needed. Some high-level abstraction approaches like queuing networks (QNs) and its variants fail to exhibit the characteristics of the execution platforms. New methods and tools are needed to simplify the design of the complex systems and to enable efficient design space exploration. These methods must enable evaluation of performance and power at the system level, in the early phases of the design flow, to avoid costly mistakes and redesigns.

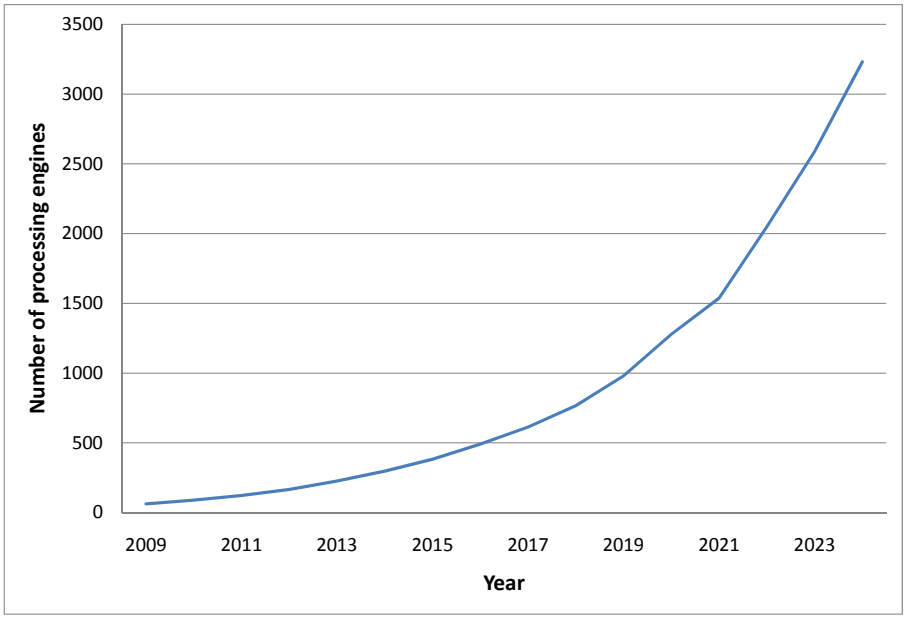

Fig 2. ITRS prediction for the number of processing engines in future platforms.

1.2 Research hypothesis

This reseach work aimed at developing a new method for evaluating the performance of the complex computer systems of the future. The developed method needs to be accurate enough so that it is possible to make design decisions based on the evaluation results. Furthermore, it needs to be usable already in the early phases of design to avoid costly redesigns. Methods were developed to simplify the modelling of applications and platforms in order to reduce the modelling effort. The effort reduction is provided by using proper abstractions and support tools that assist in model development. Methods to improve evaluation speed were also studied. The performance evaluation approach needs to be fast enough for exploring design alternatives without running simulations that last weeks or months.

The hypothesis for the research is that it is possible to perform system-level performance evaluation using abstract, non-functional workload models of applications instead of fully-functional application binaries. The workload models consist of abstract instructions read, write and execute. The use of workload models reduces the complexity of the system model consisting of the workload models and a SystemC-based execution

platform model. This decreases modelling effort, improves simulation speed and maintains accuracy at a reasonable level. It is possible to simulate the system model already in the early phases of design, before the implementations of both the application and the platform exist. The load information for creating the workload models can be extracted from various sources, including algorithm descriptions, traces, measurement data, and the source code.

1.3 Research methods

This research concentrates on designing a novel approach for the performance evaluation of complex, future embedded systems. The abstractions needed to lower modelling effort and enhance simulation speed while maintaining accuracy will be developed intuitively and experimentally. Several methods for creating workload models from various sources of information will be researched. The models can be created analytically from algorithm descriptions or by transformation from the source code, measurement data, or execution trace. Automatic or semi-automatic tools for assisting the designer with the workload modelling methods will be developed where applicable.

The developed performance evaluation approach will be applied to several case studies, which represent both state-of-the-art and future embedded systems. Simulations of the systems will be performed and data representing the performance and power and energy consumption of the system will be extracted. The complexity of even the state-of-the-art systems is such that it is not feasible to verify all simulations with measurements. The visibility of the internal performance of real systems is typically restricted and the measurements are often limited to the execution time and throughput of application or applications. However, the simulation results obtained with the performance evaluation approach will be verified with measurements in real systems while taking the above limitations into account.

1.4 Original papers

This thesis is built on the foundation laid in nine original scientific papers, which are presented in more detail in Chapter 6. Paper I describes the overall performance evaluation approach presented in this thesis in the form it was at the beginning of 2008. It contains detailed descriptions of how applications and platforms are modelled and of the supporting tools. Paper II elaborates on the interfaces between models of

applications and platforms and on service modelling. Paper III expands the approach for system-level power and energy consumption simulation.

Load extraction techniques for workload modelling of applications are described in Papers I, IV, V and VI. Paper I describes an analytical technique based on application specifications. Paper IV shows, how to extract load information from measurements of individual applications and how to combine them to a workload model of a multi-application use case. Paper V presents an initial, manual source code-based technique, which is replaced with an automatic compiler-based technique in Paper VI. Paper VII contains an early technique for utilising the Unified Modelling Language (UML) in the creation of workload models. This technique is then improved in Papers VIII and IX.

All of the papers from I to IX contain experiments with case studies. Where possible, the simulation results obtained from the experiments have been validated with measurements from real applications executed on real platforms.

2 System-level performance evaluation

Performance evaluation approaches can be divided into three categories: analytical approach, simulations, and measurements (Heidelberg & Lavenberg (1984), Jain (1991)). The analytical approach is suitable for early performance evaluation, since results are available quickly and evaluating trade-offs is easy. Analysis provides the best insight into the effects of various system parameters and their interactions (Jain (1991)). On the other hand, the accuracy of results is low because the analytical approach requires many simplifications and assumptions. As a consequence, the results provided by the analytical approaches are not convincing. Analytical methods are typical in capacity planning — especially the determination of computing, caching and buffering capacity — I/O design and helping in resource organisation design. Examples of analytical approaches are presented in Raghavan *et al.* (2004), Marcon *et al.* (2005), Sangiovanni-Vincentelli & Di Natale (2007).

In simulation approaches, the execution of an application is simulated using a computer program. In co-simulation, this consists of two parts: simulation of software execution and simulation of hardware. Simulation provides more accurate results than analysis in general since it is possible to incorporate more details of the system than in analysis (Jain (1991)). The level of detail is limited only by the time that is available for model development. More detailed simulations require more time to develop and are harder to debug and slower to simulate. Thus it is better to start with less detailed models and increase the detail level as the design progresses. Like the analytical approaches, simulations are suitable for early evaluation, since they can be performed before implementations of hardware and or software are available. Evaluation of design trade-offs is still possible with simulations, but due to slower evaluation speed, it is less practical than with analysis.

Performance measurements can be done with real applications, prototypes of real applications, or benchmark programs that mimic the real software. An implementation of the execution platform is, however, required in all cases, and therefore measurements are not suitable for early evaluation. Evaluation speed with measurements varies, but it is slower than analysis and faster than simulation in general (Jain (1991)). The measurement results are obtained from a real system and thus convincing; however, the instrumentation required to perform the measurements can affect the results.

Performance data that can be obtained with measurements is typically limited by the system. For example, measuring processor utilisation or shared memory access latency may not be possible without special HW/SW support in the system. Examples of measurement-based performance evaluation approaches can be found from Calvez & Pasquier (1998), Stewart & Arora (2003), Suresh *et al.* (2003).

Measurements are out of the scope of this thesis since they cannot be used during system design phases. Analytical approach is discarded due to low accuracy, which leaves only simulation. The simulation speed problem needs to be solved with proper abstraction level while maintaining result accuracy at a useful level. Only such simulation methods that are capable of evaluating the performance of a complete system as opposed to an individual application or hardware component are considered in the following.

2.1 Virtual system, virtual platform and virtual prototype approaches

The following categorisation of performance simulation approaches has been adapted from European EDA Roadmap. The approaches are divided into virtual system, virtual platform and virtual prototype categories based on the detail level in the execution platform model and whether real applications or models of applications are simulated.

Virtual system approaches combine abstract application models with an abstract execution platform model. The applications are represented using e.g. workload models, traces or task graphs, but not as real instructions of processors. The platform model typically has a high abstraction level and capacity models of components instead of instruction set simulators. Virtual system approaches are used for system-level exploration, where they fill the gap between system specification and virtual platforms. They can be used to make decisions regarding architecture, performance, price, and power consumption in addition to verifying design expectations in terms of functional and non-functional system properties (European EDA Roadmap). Due to the abstraction level, virtual system approaches are both instruction and cycle approximate. Simulation speed is typically faster than with lower-level simulation approaches, but the results are less accurate compared to virtual prototype simulations.

Virtual platform approaches use real application software compiled to binary form in simulations. The execution of the applications is simulated on top of a virtual platform model, which contains one or more instruction set simulators. The platform models

need to be functionally complete and use accurate memory maps to be able to execute the application binaries. Therefore, the models are complex and the approaches rely on the availability of IP models. The SystemC library is often used for modelling the interconnects and peripherals, which are then linked to custom instruction set simulators. Virtual platform approaches are instruction accurate, but accuracy with respect to timing can vary a lot — some approaches or tools can even have two sets of platform component models, one for slow and accurate simulation and another for fast and inaccurate. Virtual platforms are mostly used for embedded software development, where it is necessary to verify that the functionality of the software works correctly and the accuracy of the performance evaluation results is less important.

Virtual prototype approaches also use real application binaries with instruction set simulators like the virtual platform approaches. However, the virtual prototype approaches have a much lower level of abstraction in their execution platform model. They model the full functionality that is required for simulating low-level details of the hardware, which emphasises the problems of model complexity and IP availability. The models are created using a hardware description language like VHDL or Verilog. Virtual prototype approaches are both instruction and clock accurate and provide more convincing results than either virtual system or virtual platform simulations. However, they are too slow for early performance evaluation from both modelling time and simulation time point of view due to the level of details in the models.

2.2 Transaction-level modelling use cases

Another categorisation is presented in Kogel *et al.* (2005), which defines four transaction-level modelling (TLM) use cases for different design tasks: Functional View (FV) for executable specification design task, Architect's View (AV) for architecture exploration, Programmer's View (PV) for embedded software design and development, and Verification View (VV) for system verification. Transaction-level modelling is a technique for raising the abstraction level of the modelling of communication to the level of communication transactions. It strives for simulation speed, accuracy and lightweight modelling (Maillet-Contoz & Ghenassia (2005)).

The functional view model is an accurate TLM model of the functionality of an application. The focus of FV models is on the algorithms, and the models are independent of any implementation. Thus the platform is not modelled at all and the

models are untimed. As a consequence, FV simulation is not suitable for performance evaluation.

The architect's view model is a functional model of all elements in the system. The models of the elements provide sufficient timing information to analyse the performance of different architectures with respect to overall performance and identification of potential bottlenecks. The application part of an AV system model may be a FV model, or represented as non-functional workload models or traffic generators (Kogel *et al.* (2005)). The platform elements — communication nodes, memories, peripherals, etc. — are performance models focusing on communication timing and resource sharing. A sufficiently accurate model of the communication and/or memory requirements needs to be derived from the system requirements.

AV models are used in architecture exploration, where it can support the evaluation of architecture trade-offs in system design — how many processors to use, what interconnect approach, how large a cache, etc. — and identification and resolving of bottlenecks in the applications and the execution platform. They are also used in hardware–software partitioning and mapping of the applications to the execution platform (Vanthournout *et al.* (2004)). AV models have enough detail to achieve approximate timing. The modelling style enables easy modelling of resource contention and arbitration (Montoreano (2007)) and enough speed to explore a very large solution space (Vanthournout *et al.* (2004), Kogel *et al.* (2005)). 100% accuracy is not needed from AV models, since the impact of architecture changes are on a much bigger scope than a single clock cycle. However, an accuracy level of 70–80% must be reached to ensure the quality of result analysis (Kogel *et al.* (2005)).

In embedded software development, large amounts of software need to be run for verification while most hardware-related aspects can be ignored. Simulation speed and development time can be greatly reduced if untimed or loosely-timed models are used. The programmer's view (PV) level extends instruction set simulators with a functional view of a system to enable SW developers to develop and debug embedded software for a SoC. A PV model contains a functionally correct model of the SoC, but other parts of the system may be excluded or abstracted to save modelling and simulation time (Kogel *et al.* (2005)). A typical PV system is composed of a processor model (instruction-set simulator), interconnection and a functional description of peripherals. A PV model is more architecture-specific than a FV model. Compared to architect's view, PV provides a different abstraction of the same platform as AV, for a different purpose. Where AV requires more timing accuracy, PV needs more functional accuracy.

PV places many requirements on the level of detail in an execution platform model. The models need to be register accurate and bit true (Donlin (2004), Vanthournout *et al.* (2004), Kogel *et al.* (2005)), capable of interrupt handling, and the memory map of the system and synchronisation between processing elements must be correctly modelled to ensure correct operation. PV is not suitable for the development of timing-critical software. However, simulation speed can be very high due to the low timing accuracy — an operating system can be booted and test code run in seconds (Montoreano (2007)).

All cycle timing details of a protocol become important as design refinement goes down at the abstraction level. Detailed HW design requires more accurate simulations than what is provided by the AV level. Verification view (VV) models include details of microarchitecture and bit-level interfaces. They have fully protocol-compliant arbitration of the communication infrastructure, and timing annotations are accurate to the level of individual clock cycles (Donlin (2004)). The detail level of the models is high enough to enable accurate HW–SW development and verification. VV simulation requires implementation models of hardware components, an accurate memory map, and functionally accurate peripherals (Kogel *et al.* (2005)). It provides more accurate results than either PV or AV simulation, but it has also lower simulation speed than both PV and AV, since the models are clocked and highly detailed.

The following two sections present system-level simulation approaches existing both in research and industry. The purpose of each approach and the modelling and simulation technique used in it is described. The approaches are also categorised according to the criteria presented in Sections 2.1 and 2.2. If the simulation approach is a part of a larger system exploration or design flow, the emphasis is on the simulation part of the approach in the following.

2.3 Simulation approaches in research

SPADE

System-level Performance Analysis and Design space Exploration (SPADE) (Lieverse *et al.* (2001a,b)) is a method and tool which implements trace-driven system-level co-simulation of application and architecture. The applications are described with Kahn process networks using YAPI (de Kock *et al.* (2000)). The Kahn process networks allow applications to be modelled relatively independently of the target platform. However,

they also limit the scope of SPADE to signal processing systems and other fields where communication can be modelled using unbounded FIFO buffers.

The application models generate symbolic instruction traces, one trace per each process in the application. The traces do not contain control expressions at all and can not model time-dependent behaviour (Pimentel *et al.* (2001)). The traces are transformed from the application level to the architecture level for performance simulation. Abstract performance models are used for describing the resources in platforms constructed from generic building blocks in a model library. The resources are divided into processing, communication and memory resources. The resources of the platform interpret the trace operations as the load to be executed, account time, and report performance data.

ARCHER

ARCHitecture ExploRation of heterogeneous embedded systems (ARCHER) (Zivkovic *et al.* (2002)) presents an extension to SPADE, which adds hybrid symbolic programs to application modelling. A symbolic program is like a symbolic instruction trace in SPADE, but which has been augmented with control information obtained from the execution of an annotated application model (Zivkovic *et al.* (2003a)). The control trace is valid only for a single set of data. The platform model supporting performance evaluation of the symbolic programs consists of Symbolic Program Units (SPUs), read write interfaces and FIFO buffers. SPUs model instruction-level parallellism and reuse of available processing resources.

Artemis

Architectures and Methods for Embedded Media Systems (Artemis) is a modelling and simulation environment focusing on embedded multimedia systems (Pimentel *et al.* (2001)). It supports two separate simulation frameworks: SPADE and Simulation of Embedded-System Architectures for Multilevel Exploration (SESAME) (Terpstra *et al.* (2001)).

Like SPADE, SESAME is also based on Kahn process networks. In SESAME, the designer first selects candidate architectures using analytical modelling and multiobjective optimisation. High-level and architecture-independent application specifications are maintained by applying dataflow graphs in its intermediate mapping layer. The dataflow graphs take care of run-time transformation of coarse-grained application-level events

into finer grained architecture-level events. The load of an application is captured partly manually by instrumenting the code of each Kahn process with annotations describing the application's computational and communication actions (Pimentel *et al.* (2006)).

SESAME allows simulation at multiple levels of abstraction (Pimentel *et al.* (2002)) and supports mixed-level models, which enable more detailed evaluation of a specific platform component. At the highest abstraction level, the components are black boxes with processing capacity, power consumption, and cost parameters. The refinement of the models can continue until RTL-level simulation (Pimentel *et al.* (2001)). The low-level simulations can be used to calibrate higher-level architecture models for improved accuracy (Pimentel (2008)). Models are created using the Pearl discrete-event simulation language, or SystemC with an add-on library extending SystemC with Pearl's message-passing paradigm (Pimentel *et al.* (2006)).

Originally only single-application simulations were possible in SESAME, but Thompson & Pimentel (2007) present an extension for simulating a combination of a primary application and stochastic secondary applications concurrently. Furthermore, Van Stralen & Pimentel (2010) add support for intra- and inter-application scenarios. The inter-application scenarios describe the behaviour of multiple applications, which can run concurrently.

ARTS

Abstract system-level modelling and simulation framework (ARTS) (Mahadevan *et al.* (2005a,b)) is intended for the modelling and simulation of multiprocessor SoCs and focuses on multimedia streaming applications. Application designers can develop new application models and study their performance for the available configurations of the platform with ARTS, whereas platform developers can develop new platform models and study their impact on a given application domain.

Applications are modelled using static data flow task graphs. The tasks can be periodic or sporadic. Tasks have a set of parameters, which define e.g. best and worst case execution time in cycles and the memory requirement of the task (Mahadevan *et al.* (2007)). Pre-emption, synchronisation, allocation of shared resources and task scheduling is supported through a real-time operating system (RTOS) model. A system model is formed by mapping tasks of the application model onto computing components of the platform model. The platform model includes communication and memory models in addition. ARTS supports modelling of different communication topologies

from shared bus to 2d mesh NoC. The application, platform and mapping models are expressed using a custom ARTS scripting language. A SystemC model is then generated and simulated in the ARTS framework.

Baghdadi *et al.*

Baghdadi *et al.* (2000, 2002) present a design space exploration methodology based on the co-design tool MUSIC and system-level simulator GEODESIM. In the first stage, the system functionality is specified in SDL language. In the second stage of the methodology, the designer targets the SDL specification for one specific system architecture. Finally, the architecture-annotated SDL specification is simulated at high level using GEODESIM.

Each SDL process may have a realisation in software or hardware. The SDL code will be translated to C or RTL code for software and hardware, respectively. For all available HW and SW realisations, the basic blocks are identified from the code, and their execution times in terms of clock cycles are calculated from low-level simulation. The SDL specification is back-annotated with the execution times. The architecture is also modelled with SDL for the GEODESIM simulation. The architecture model contains node, priority and delay directives, where nodes represent execution resources, generally processors. Priority and delay directives are used to assign priorities to SDL processes to model multitasking and delays to specify the execution time of SDL actions obtained from the low-level simulation.

Jaber *et al.*

Jaber *et al.* (2009) presents a high-level architecture exploration methodology for investigating the influence of shared resources on the performance of a system. The methodology is based on the DIPLODOCUS design space exploration UML profile, and uses the Y-chart approach, where applications, architecture and mapping are modelled separately. UML models are given as input to the simulation environment, which generates SystemC code for simulation.

Abstract application models are created using class diagrams for modelling tasks and activity diagrams for the behaviour of those tasks (Apvrille *et al.* (2006). Abstract architecture models are compositions of instances of generic components. The list of generic components includes a processor, bus, memory, hardware accelerator, and

peripheral. Each component has a small set of parameters. In addition to SystemC, DIPLODOCUS supports transformation to a LOTOS specification for formal analysis.

Applications are modelled as a network of communicating tasks (Waseem *et al.* (2006)), which can exchange data samples and signals and request the execution of another task. Abstract data exchange and data processing instructions are used to characterise the behaviour. The architecture is modelled as a network of physical resources, including computation, communication and storage nodes. All resources have parameters like processing capacity in millions of cycles per second or memory size in bytes. The nodes are instantiated from a library implementing a set of predefined models. In the mapping phase, tasks issue computation and communication requests to shared resources. Simulation is able to produce e.g. average, worst and best case execution times for each task and the utilisation of each resource.

M3-SCoPE

M3-SCoPE proposed by Posadas *et al.* (2011) is a design space exploration framework based on the virtual system simulation approach. Instead of an ISS, it uses native co-simulation of the embbeded software in the simulation host computer. It provides performance estimation of the entire system by including facilities to model the temporal behaviour of application software, to emulate the behaviour of an operating system (Posadas *et al.* (2005)), and to enable realistic modelling of hardware software communication.

Performance modelling of the embedded software is performed by adding static execution time estimates as annotations to the code. Next, the SW code is executed and the annotated estimates are used to achieve timed simulation. The estimates are obtained by cross-compiling the application to the target hardware and then analysing the basic blocks of the generated assembly code. Compiler optimisations and caching in processors can also be taken into account. In hardware platform modelling, PV models of components including processors, buses, memories, and network interfaces are used. Furthermore, it is possible to define static and dynamic power consumption for each component to estimate the power consumption of the system.

M3-SCoPE requires mature application software, models of hardware components and an XML-based description of the system as input. The XML file describes the instantiation of applications and hardware components and the allocation of the

applications on the hardware. The output of the simulator gives estimates on execution time and number of cache misses in XML form.

MESH

Modelling Environment for Software and Hardware (MESH) (Paul *et al.* (2003)) is a simulation framework based on a layered model composed of dynamic logical threads on top of a physical thread layer. The logical threads model application software and are expressed by annotating C code with consume calls. The calls indicate computational complexity of a software region. The complexity values are derived by profiling or designer experience. The trace annotations dictate the finest timing resolution provided by MESH. The physical threads model hardware and describe the computational power of processors, memories, HW devices, and networks. The modelling technique is intended mainly for RISC style processors (Paul *et al.* (2005)). The scheduling layer of MESH can use the computational power figures to determine, when and where the logical threads should execute.

MESH uses a hybrid of performance analysis and simulation (Bobrek *et al.* (2004)). Analytic contention models are applied to groups of shared resource accesses, which are derived from simulation. The threads are simulated for a period of time determined by the thread annotations, while contention is ignored. Accesses to shared resources within that period are grouped and sent to the analytical model to calculate time penalties due to competition for the shared resources of the system. Complex systems may have a large number of small timeslices, which has an adverse effect on simulation speed. The designer can specify a minimum timeslice to reduce the effect at the cost of accuracy.

MILAN

The Model-Based Integrated Simulation (MILAN) framework (Mohanty *et al.* (2002)) includes the DESERT design space exploration and High-level Performance Evaluator (HiPerE) (Mohanty & Prasanna (2002)) rapid performance estimation tools. DESERT is used to eliminate designs, which do not meet user-defined constraints and HiPerE to estimate the performance of the remaining designs at the system level. HiPerE evaluates the system-wide performance based on component-specific performance parameters, which are provided by the designer initially. However, MILAN also supports component-specific performance evaluation with analytical and cycle-accurate simulators for

providing more accurate performance parameters to HiPerE. The component-specific evaluation estimates computation, storage and communication cost.

Applications in HiPerE are modelled using trace files, which consist of ordered lists of communication and computation operations. First, the application source code is annotated at task level. Next, the trace is obtained from the functional simulation during which the task is executed and certain amount of data is transferred. HiPerE uses a Generic Model (GenM) for modelling the SoC architectures. GenM consists of a processor, some reconfigurable logic, and memory connected via an interconnect. Each resource in GenM has timing and energy parameters. The user must model, which tasks can be implemented on which available resources. For application–platform mapping, HiPerE needs additional performance parameters, including the time for executing a task on a specific processor. The designer provides initial values for the performance parameters, but component-specific performance simulation can be used to provide more accurate values.

TAPES

The Trace-based Architecture Performance Evaluation with SystemC (TAPES) (Wild *et al.* (2006)) performance evaluation approach uses traces to model applications at a high abstraction level. The functionality of applications is replaced with execution latencies of subfunctions consisting of a processing delay and external transactions on the respective resources. Interaction between subfunctions is represented with inter-SoC-module transactions, but no real data exchange is performed during simulation. The architecture resources are modelled as black boxes, whose internal structure and processing is disregarded during simulation. The effect of caches in processors is taken into account manually during trace definition. Contention and arbitration in the communication architecture is modelled, and the contention results in stretching of the traces with respect to time.

TAPES is based on SystemC, and the simulation model is dynamically generated from an XML-based configuration and a model library. The library includes abstract resources for processors, memories, accelerators, and communication architectures. The configuration and model generation system enables easy reconfiguration to different resource and mapping configurations. However, the specification of the traces is a manual process, and the approach needs to evaluate all possible patterns of conditions for branches since control dependencies cannot be resolved during simulation. It is

also limited to bus-based SoC architectures with a single system bus and point-to-point connections.

Fornaciari *et al.*

Fornaciari *et al.* (2001) presents a design space exploration framework focusing on processor-to-memory communication through the memory hierarchy. The simulation technique of the framework is based on a software execution profiler for cycle-accurate instruction set simulation of the application and a dynamic tracer to generate data and address bus streams. The technique includes configurable bus and memory models, with the latter having behavioural models of on- and off-chip level 1 and 2 caches and main memory. The bus and memory models use the bus traces from the software execution profiler as input.

The framework aims to find the best platform configuration for the application without performing exhaustive analysis of the parameter space. The parameters for the exploration include cache size, block size, and data and instruction cache associativity (Fornaciari *et al.* (2001, 2002)). Sensitivity analysis is used to perform optimisation of exploration parameters and to discard less promising configurations from evaluation. Analytical energy models are used to evaluate overall energy-delay cost.

Lahiri *et al.*

Lahiri *et al.* (2001b) presents a hybrid trace-based performance analysis technique and tool for the design of custom communication architectures for SoCs utilising the POLIS (Balarin (1997)) and PTOLEMY (Eker *et al.* (2003)) frameworks. First, an initial co-simulation with abstract communication, where the communication is modelled as exchange of events or tokens, is performed. The architecture implementing the communication is not considered. The output from the simulation is a timing-inaccurate system execution trace, which is represented using communication analysis graphs (CAGs). The CAGs represent computation, communication and synchronisation without details like values of internal variables or data. The first step of the technique is slow and intended to be performed only once per each use case.

Second, the CAGs are analysed to estimate the performance of the system by taking the effects of the communication architecture into account. The designer lays down the communication architecture to implement communication events and the tool

manipulates the CAGs based on the communication architecture to generate a timing-accurate trace. The manipulation includes adding handshaking to initiate communication, resizing of communication according to maximum block transfer sizes of channels, and dynamic delays due to contention and arbitration. A clustering algorithm is used to choose an initial mapping of the communication to the network topology, which is then improved iteratively. The second step of the technique supports design space exploration and optimisation by being able to evaluate alternative communication architectures rapidly (Lahiri *et al.* (2004)).

Schnerr *et al.*

Schnerr *et al.* (2008) presents an approach for the simulation of embedded software, which integrates cycle-accurate analysis with an abstract SystemC model. The first step of the approach consists of static cycle-accurate execution time analysis for the application. A cross-compiler is used to create an executable binary for the target processor. Execution time in clock cycles is calculated statically for each basic block based on the instructions in the binary and a pipeline description of the processor. Abstract SystemC models using TLM communication are generated in the second step of the approach. Finally, the SystemC models are back-annotated with WCET and BCET timing from the analysis step. A dynamic correction is included in the models to adjust for branch prediction and caching techniques in modern processors. The architecture simulation technique of the approach is limited to the simulation of processors and caches.

StepNP

StepNP is a simulation environment developed at ST Microelectronics, which concentrates on network processors (Paulin *et al.* (2002)). The main components of the environment are a high-level multiprocessor architecture simulation model, a network router application framework and a control, debugging and analysis toolset. The architecture simulation platform is SystemC-based and encapsulates instruction-set simulators inside a SystemC wrapper. It contains models for processor engines (a RISC-style processor with multithreading), NoC communication channel using the OCP protocol, and specialised co-processors. StepNP supports functional, transaction and cycle-based abstraction levels.

Beltrame *et al.* (2006) proposes an application-platform mapping methodology based on StepNP to achieve co-exploration of the application design space. Mapping begins with the consideration of a software-only implementation. The C source code of the application is profiled natively in a workstation and parallelised if possible. Then the application and a platform model is co-simulated with StepNP. The simulation results are analysed and the application and or architecture models can be modified by adjusting the hardware software partitioning. The mapping cycle continues until all application constraints are met.

A technique for multi-accuracy power and performance modelling with StepNP is presented in Beltrame *et al.* (2007). It provides StepNP with the capability of dynamic switching of functionality, communication and power consumption models at different accuracy levels. A refined, accurate model is used for the interesting parts of the simulation and they are replaced with less accurate but faster models for the other sections. A library of power consumption models is used to measure power consumption and perform energy-delay tradeoffs.

ReSP

Reflective Simulation Platform (ReSP) (Beltrame *et al.* (2008)) is a TLM-based multi-processor simulation platform, which provides a wrapper for the Python scripting language around the SystemC kernel. The Python extension adds a simplified way to specify the architecture of the system and simulate the given configuration, and it provides support for automated analysis.

ReSP has a set of primitives to build a complex platform from components chosen from a database of SystemC modules. The database contains functional, cycle-accurate processor cores written using the ArchC language, interconnect models for buses and NoCs, memory hierarchy models, and miscellaneous components like interrupt controllers. Additional components can be created with SystemC. Since an instruction set simulator is used for processors, there are no models of applications. ReSP provides two operation modes: interactive and automatic. In the interactive mode, the architecture is built using commands exported by the user interface. In the automatic mode, the architecture description is provided in an XML file, from which the architecture model is generated, which can be used to support design space exploration algorithms.

Metropolis

Metropolis (Balarin *et al.* (2003)) is a system design framework based on a metamodel supporting functionality capture and analysis, architecture description, and mapping. It can support synthesis and formal analysis tools in addition to simulation and is able to generate verification models automatically. Metropolis includes a parser that reads metamodel designs and an API for developers to modify it and augment additional information. The API is also used by tool-specific backends to generate the input required by the tool from the metamodel. Simulator backend translates the metamodel specification into executable SystemC code (Sangiovanni-Vincentelli (2007)).

Application functionality is modelled as a network of processes communicating through ports with defined interfaces. The network's behaviour is defined as a set of executions, where the executions are a sequence of events, i.e. program entries or exits. Architecture functionality is modelled as a set of services provided by the arhicture to the application. Efficiency of the service implementation is taken into account with a cost value representing either time or energy. The architecture contains both media (processors, buses and memories) and processes (software tasks on a processor). A mapping network encapsulates function and architecture networks and defines, which part of the architecture serves which application functionality.

Koski

Koski (Kangas *et al.* (2006)) is an automated flow for multiprocessor SoC design based on the TUT UML profile. It covers all design phases from specification to implementation. First, the requirements and constraints of both application and architecture are captured. Next, the functionality of the system is described with an UML application model, which can be verified with functional simulation. UML application and architecture models are transformed to abstract architecture exploration models, and an automatic exploration is performed using static and dynamic exploration models. Finally, an implementation is produced by passing the UML descriptions to automatic code generation.

The UML application model uses class diagrams to describe the class hierarchy of the application and composite structure diagrams to describe the connections between the class instances. Signals and ports are used for communication. The UML architecture model is described with a composite structure diagram, which instantiates processing element and communication network components from a library with a set of parameters

like clock frequency, cache size, data width, etc. The design constraints are defined in a UML system model with architecture specific attributes such as area, power and execution time. For architecture exploration, the UML application models are abstracted to Kahn process network models, where the complexity of each application task is obtained from profiling in a reference platform or in the final platform. The processing elements of the architecture are characterised with performance, area, power and available memory properties. A coarse model of communication is used for static exploration, whereas dynamic exploration adds a PV-level or cycle-accurate model of the communication network. Dynamic exploration uses models at different levels of abstraction, so a transaction generator (Kangas *et al.* (2003)) is needed to combine the models. Performance information obtained from the exploration is back-annotated to the UML environment after the exploration is finished.

2.4 Commercial system-level simulation tools

CoFluent Studio

CoFluent Studio is a commercial simulation framework from CoFluent Design, which contains a graphical modelling tool and a SystemC library extending the OSCI SystemC and TLM libraries (CoFluent). CoFluent Studio contains two packages, one for timed behavioural modelling (TBM) and another for system architecting (SA). The TBM package is used for obtaining timed-executable specifications of the application for further system architecting or implementation. The SA package is used for exploring system architectures, analysing their performance, and obtaining executable specifications for futher HW/SW co-design and implementation.

TBM captures the behaviour and time of applications with a proprietary presentation using the GUI tool and/or C++ code. The execution time for parts of the software is given by the designer as input. The TBM package is able to generate SystemC code from the graphical model automatically for simulation. SA adds a proprietary representation of the execution platform in the GUI tool and enables mapping of the application model created in TBM to the platform model. The platform model consists of layers of processor, interconnect and memory resources, to which the designer can add timing annotations.

Synopsys Platform Architect and Processor Designer

Synopsys Platform Architect (Synopsys Platform Architect) is a tool for the design, performance analysis and optimisation of multi-core SoC architectures. It is based on SystemC and TLM and provides a graphical modelling tool and visualisation of simulation results. Platform Architect supports early hardware–software partitioning for optimising system performance through the virtual system modelling approach. Generic task graphs are used to create SystemC performance models of application tasks. The graphs are mapped onto Virtual Processing Units (VPUs), which are task-driven traffic generators. After exploration the models can be refined to create a virtual platform capable of executing the application software by replacing the VPUs with instruction set simulators. Mixed SystemC / HDL co-simulation is supported in combination with other Synopsys tools. A component library provides models of VPUs, processors, interconnects, and memories. Processor models are available in loosely-timed and cycle-accurate form, whereas the interconnect and memory models can be cycle-accurate or cycle-approximate.

Synopsys Processor Designer (Synopsys Processor Designer) is another tool for developing custom application-specific instruction set processors (ASIPs). It is able to produce an ISS, SW tools including a compiler, and an RTL implementation model from the ASIP specification given as input. The ISS models from Processor Designer can be combined with the system-level models in Platform Architect (Wieferink *et al.* (2005)).

Cadence Virtual System Platform

Cadence Virtual System Platform (Cadence Virtual System Platform) is a commercial software development, functional verification, and system analysis and optimisation tool. Hardware software co-simulation and virtual prototype verification is supported through other Cadence tools. Virtual System Platform is able to produce SystemC TLM 2.0-compatible virtual platform models automatically from IP-XACT or text input specifications. The platform is built from components in a SystemC TLM IP model library, which includes fast instruction set simulators of processors for software development and peripheral models. Abstraction level can be lowered to RTL by refining the TLM models and using high-level synthesis.

2.5 Summary

Table 1 summarises the purpose of each performance evaluation approach presented in this chapter and categorises them to simulation and hybrid simulation–analysis approaches. The simulation technique used in each approach is further categorised to virtual system, virtual platform or virtual prototype simulation according to European EDA Roadmap and to FV, AV, PV or VV simulation based on the TLM use cases presented in Kogel *et al.* (2005). It is also noted, if the approach supports RTL simulation.

Section 1.1 stated that the most significant problem with the performance evaluation of future embedded systems is complexity. The requirement for early evaluation precludes virtual platform and virtual prototype approaches, since it is not possible to wait until the applications have been developed. For the same reason, PV and VV use cases are not suitable for the task. Furthermore, the detail level in the platform model would be very high with VV, and it would require considerable modelling effort especially if several alternative designs were to be explored. Thus, virtual system and architect's view based simulation were selected for the ABSOLUT approach presented in Chapter 3 of this thesis to achieve early evaluation, low modelling effort, and reasonable accuracy.

The ABSOLUT approach is naturally closest to the other virtual system and AV-based approaches. ABSOLUT is not limited to a specific application field or model of computation. However, SPADE and its descendants, ARCHER and Artemis/SESAME, use Kahn process networks for application models and concentrate on signal processing systems. In TAPES, the applications are modelled as traces, whose specification is a manual process. TAPES is also limited to bus-based SoC architectures with a single system bus. ARTS uses data flow graphs as application models and focuses on multimedia streaming. A common problem of the ARTS, Baghdadi *et al.*, MESH and MILAN approaches is that they require execution times of (parts of) applications as input to the models. The approaches rely on designer experience, profiling in a workstation or low-level simulation support to obtain the execution times. Jaber *et al.* and CoFluent Studio use UML and proprietary models of applications, respectively. Creating the graphical models requires effort and it is very difficult to utilise existing data, e.g. source code, measurements or traces, to reduce the modelling effort.

Table 1. Categorisation of surveyed performance simulation approaches.

Approach	Purpose			Type		Simulation							
						EER			TLM				RTL
	Ev[1]	Ex[2]	F[3]	A[4]	S[5]	S[6]	Pl[7]	Pr[8]	FV	AV	PV	VV	
ABSOLUT	x				x	x				x			
ARCHER		x			x	x				x			
Artemis		x			x	x				x		x	x
ARTS	x				x	x				x			
Baghdadi *et al.*	x[9]	x	x[10]		x	x[9]		x[10]		x[9]			x[10]
Cadence VSP					x		x	x			x		x
CoFluent Studio	x				x	x			x[11]	x[12]			
Fornaciari *et al.*		x		x	x[13]		x[13]				x[13]		
Jaber *et al.*		x		x[14]	x	x			x[14]	x			
Koski			x	x	x	x		x[15]	x		x[16]	x[16]	x[15]
Lahiri *et al.*		x		x	x[17]		x[17]				x[17]		

[1] Performance evaluation
[2] Design space exploration
[3] Design flow
[4] Analysis
[5] Simulation
[6] Virtual system
[7] Virtual platform
[8] Virtual prototype
[9] GEODESIM
[10] MUSIC
[11] TBM package
[12] SA package
[13] Software execution profiler (instruction set simulator) part
[14] Provided by DIPLODOCUS
[15] Real FPGA prototype
[16] For communication network only
[17] HW/SW co-simulation part of the approach

Table 1. Categorisation of surveyed performance simulation approaches (continued).

Approach	Purpose			Type		Simulation							
						EER			TLM				RTL
	Ev[1]	Ex[2]	F[3]	A[4]	S[5]	S[6]	Pl[7]	Pr[8]	FV	AV	PV	VV	
M3-SCoPE	x	x			x	x[18]					x		
MESH	x				x	x				x			
Metropolis			x	x	x	x			x	x			
MILAN	x	x		x[20]	x	x[19]	x[20]			x[19]		x[20]	
ReSP	x				x		x				x		
Schnerr *et al.*	x			x	x		x					x[21]	
SPADE		x			x	x				x			
StepNP	x	x[22]			x		x		x		x	x	
Synopsys PA		x			x	x	x			x	x	x	
TAPES	x				x	x				x			

[1] Performance evaluation
[2] Design space exploration
[3] Design flow
[4] Analysis
[5] Simulation
[6] Virtual system
[7] Virtual platform
[8] Virtual prototype
[9] HW/SW co-simulation part of the approach
[18] Requires application SW for simulation but does not use an ISS
[19] System-wide performance estimation with HiPerE
[20] Component-specific performance estimation
[21] Architecture simulation limited to only processors and caches
[22] With the methodology presented in Beltrame *et al.* (2006)

3 ABSOLUT

This chapter presents the system-level performance modelling and simulation approach developed in the work. Abstract Instruction Workload and Execution Platform-based performance simulation (ABSOLUT) is a model-based layered approach for the evaluation of embedded computer system performance in early phases of development. ABSOLUT supports early decision making by giving answers to questions like:

- Is this new application or feature feasible on that platform?
- Does this platform provide enough capacity for this set of applications?
- What kind of platform architecture and computing and/or communication resources would be needed for these foreseen applications and/or services?

The ABSOLUT approach aims at early evaluation with low modelling effort. It does not require that the final software or hardware exists before the models can be created and simulated, which is shown in the case examples of Chapter 4. The amount of work required for simulating use cases is far less than comparable implementation efforts. A set of support tools exists for generating complex workload and platform models and for filtering and visualising simulation results.

Figure 3 illustrates the ABSOLUT approach in Y-chart form (Kreku *et al.* (2008b)). Applications and their input are modelled as workloads, which consist of abstract primitive instructions. The workload models control the progress of the simulation and utilise the resources of the execution platform model. However, data processing of the applications is abstracted with the primitive instructions, so for example an ABSOLUT model of a video playback application is not able to play video files.

The second arm of the Y-chart is formed by a capacity model of the execution platform. The platform model provides execution resources to the workload models and models performance or timing characteristics of the real platform. It is the role of the platform model to establish how long it takes to execute the workloads. The execution platform model provides its services to the workload models via pre-defined interfaces, which enables separation of application and platform concerns. The workload models can be allocated to platform resources, which provide the services required by the model. It is possible to replace individual resources or the entire platform without modifications to the workload models as long as the service requirements are satisfied.

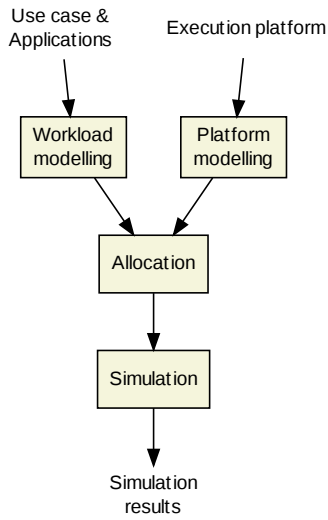

Fig 3. The ABSOLUT approach presented according to the Y-chart model.

Both the workload and platform models are implemented in SystemC (Grötker (2002)) and simulated using the OSCI SystemC (Open SystemC Initiative website) kernel. The models are instrumented with performance probes for extracting resource utilisation, latencies, and other relevant performance data out of the simulations. Several tools have been implemented so that the user can create workload and platform models without writing new SystemC code or porting existing code to SystemC.

3.1 Application modelling

Applications are modelled as abstract workload models in the ABSOLUT approach (Kreku *et al.* (2004b, 2008b)). The purpose of the workload models is to illustrate the load an application causes to an execution platform when it is executed. They characterise the control flow and the effect of the data processing and communication on the execution platform. Abstraction is achieved by not modelling the calculations or operations of the original application in detail. Due to the abstraction method, the workload models can be created and simulated before the applications are finalised, which enables early performance evaluation. Abstracting the functionality of the application reduces application modelling effort and improves simulation speed. Furthermore, the workload

modelling technique has a beneficial impact also on the platform model complexity, which will be elaborated on in Section 3.2.

It is straightforward to modify the models, which facilitates easier evaluation of various use cases with minor differences. For example, the models can be parameterised to scale the complexity or change the execution order of applications from one use case to another. As opposed to typical virtual system-based simulation approaches, the workload models themselves do not contain execution time information. It is left for the platform model to find out how long it takes to process the workloads in ABSOLUT.

3.1.1 Layers

Workload models have a hierarchical structure (Kreku *et al.* (2006), Kreku & Tiensyrjä (2011)) depicted in Table 3, where application workload model A is constructed of one or more process workload models P_i:

$$A = \{C_p, P_1, P_2, \ldots, P_n\}, \tag{1}$$

where C_p denotes the common control between the process workload models and n is the total number of processes and threads in the application. There is no disctinction between processes and threads in ABSOLUT due to the abstraction — both are modelled using process workload models. The control may be implemented using, for example, standard C++ control structures in SystemC-based workload models.

The processes are comprised of function workloads F_i:

$$P_i = \{C_f, F_1, F_2, \ldots, F_n\}, \tag{2}$$

where C_f is control and describes the relations of the functions, for example, branches and loops. The operating system models of the execution platform model handle workload scheduling at the process level.

Function workload models are basically control flow graphs:

$$F_i = (V, G), \tag{3}$$

where nodes $v_i \in V$ are basic blocks and arcs $g_i \in G$ are branches. Basic blocks are ordered sets of load primitives used for load characterisation. The load primitives in ABSOLUT are abstract instructions read and write for modelling memory accesses and execute for modelling data processing (Table 2). Reads and writes take target

Table 2. Read, write and execute are the low-level workload primitives in ABSOLUT.

Primitive	Parameters	Description
Read	$addr$, w, b	Read w words of b bits from address $addr$
Write	$addr$, w, b	Write w words of b bits to address $addr$
Execute	n	Execute n data processing instructions

Table 3. The workload model hierarchy has four layers.

Layer	Content
Application layer	Process workload models of processes and threads
Process layer	One or more function workload models; control
Function layer	Basic blocks; control
Basic block layer	Abstract instructions read, write and execute; service requests

address, number of words and number of bits as parameters, where the address is not exact but directs the read or write to a specific component in the system. The workload models are platform independent since the primitives are abstract and not real processor instructions.

Control at the function layer can be either deterministic or statistical. Deterministic control follows the modelled use case accurately, and it is written explicitly in the workload model using C++ control structures. It can also be read from an external control trace (Saastamoinen & Kreku (2011)). With statistical control, the arcs are associated with branching probabilities, which the simulator will use to select the next basic block to be executed (Kreku *et al.* (2010)). Statistical control results to smaller models, but at the cost of accuracy.

3.1.2 Load extraction for application models

Workload models capture the control behaviour of applications in the hierarchically layered structures. On the other hand, they abstract the details of data processing and communication as workload primitives. To obtain the control and workload primitives for the models, four different techniques are used in ABSOLUT: analytical, measurement-based, trace-based, and compiler-based (Kreku *et al.* (2008b)). These techniques can be

used separately or in combination depending on what kind of descriptions of application algorithms are available.

Analytical workload modelling

The analytical workload modelling technique (Kreku *et al.* (2004a)) is currently the simplest way to create workload models, but also generally the least accurate. As the name suggests, it is based on analysis of the number of operations from an algorithm specification or other suitable description.

Analytical modelling consists of three phases. First, the number of data processing instructions and the amount of memory traffic is analysed from the specification. In principle, after this point we could write a simple three-line workload model that would consist of one — potentially large — read, write and execute primitives. However, the ordering and the block size of the operations usually affects performance notably. Therefore, in the second phase the total operation numbers are divided into smaller blocks. In the best case, this can also be done based on the algorithm description. In the worst case, we can only create a number of uniformly distributed blocks of workload primitives.

The analytical technique typically produces the most compact workload models. It also requires the least effort, if support tools are not taken into account. However, the quality of the workload models depends a great deal on the use case (algorithm) being modelled.

Measurement-based workload model generation

The second technique for creating the workload models is to extract data from partial measurements or traces made of the use case (Kreku *et al.* (2004b)) being modelled. This method can easily be automated, which allows rapid modelling of complex use cases with minimal manual work. The complex use case in this context means a workload, which cannot be measured in its entirety in an existing system and which consists of several programs that can be measured or estimated alone.

A method has been developed for the generation of workload models from processor utilisation data in three steps. The first step is to measure the processor utilisation data of the individual programs. The second step is to merge and concatenate the data according to the state sequence model of the complex use case. The merged utilisation curve is

likely to have data points where the utilisation is more than 100%. This is, however, not a problem because the value will only be used as a basis for estimating the number of load primitives for the workload model. The third step is to generate the workload models from the samples of the monitoring tool.

Trace-based workload model generation

The automatic trace-based workload model generation technique takes an application execution trace as input. It is assumed that the trace describes the instructions executed by a processor or a hardware accelerator while it is running the application to be modelled. The trace can be directly converted to a workload model by generating a read primitive for each load instruction, a write primitive for each store, and execute primitives for all the other instructions. Another possibility is to create a statistical workload model from the trace by sampling the probabilities of the load, store and data processing instructions periodically. The benefit of the statistical trace-based models is that they are smaller and faster to simulate. It is also straightforward to create variants of the models by scaling the complexity upwards or downwards.

Compiler-based workload model generation: ABSINTH and ABSINTH2

ABSINTH (ABStract INstruction exTraction Helper) is a tool for generating workload models automatically from the application source code (Kreku *et al.* (2010)). ABSINTH has been implemented by extending GNU Compiler Collection (GCC) version 4.5.1 with two additional passes (Figure 4). The first ABSINTH pass, pass_absinth_cf, is responsible for constructing the function layer of the workload model, i.e. the control flow between basic blocks in each source code function. The second pass, pass_absinth_bbs, will traverse the Register Transfer Language (RTL, GCC's low-level intermediate format) representation to extract workload primitives for each basic block. ABSINTH generates workload models in a late phase of compilation after most optimisation passes. Thus the resulting models are somewhat target dependent. For best accuracy, one should use the same compiler target architecture in model generation and performance simulation (e.g. a cross-compiler).

There are three phases in the model generation process to obtain a proper model of control for the models:

1. First, the source code must be compiled with profiling (`-fprofile-generate`)

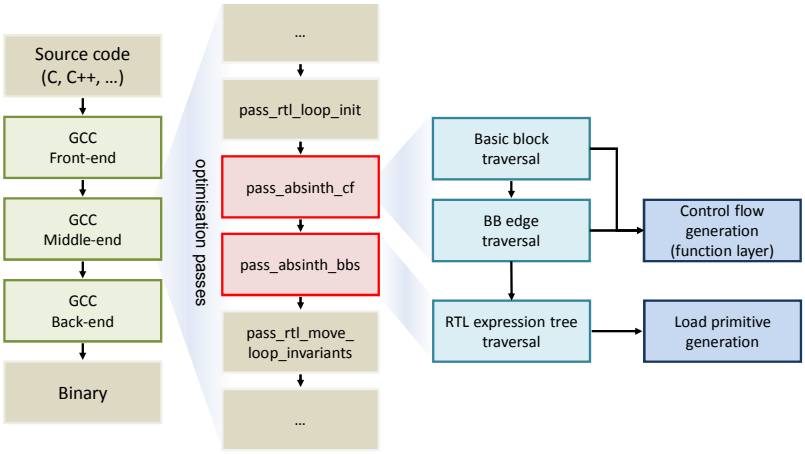

Fig 4. ABSINTH as a part of GCC (Kreku *et al.* (2010)) (©2010 IEEE).

2. then, the compiled binary must be executed with a data set corresponding to the use case
3. finally, the source code must be compiled again with both profile-guided optimisation and ABSINTH-enabled (`-fprofile-use -fabsinth`).

ABSINTH uses the profiling data during model generation for the probabilities of branches, which are modelled statistically. It is also used to extract the number of iterations for loops.

ABSINTH2 (Saastamoinen & Kreku (2011)) is a second-generation tool for compiler-based workload model generation. Unlike ABSINTH, it produces workload models, which are deterministic at the function layer. ABSINTH2 is an alternative to ABSINTH, not a replacement. It is able to produce more accurate workload models due to the deterministic function-layer control. However, the resulting models are also larger and slower to simulate.

There are two phases in workload model generation with ABSINTH2:

1. First, the application is compiled with ABSINTH2-modified GCC with flags `-fabsinth2 -labsinth-tracer`. ABSINTH2 modifies the application binary by inserting a function call to a tracing library at the beginning of each basic block. An XML file containing the workload primitives of each basic block is also written.
2. Next, the application is executed. The tracing library writes a zlib-compressed trace of the control.

A special ABSINTH2 SystemC process workload model has been implemented. The process workload model parses the control trace and primitives on-the-fly during simulation.

An extension to ABSINTH2 called SAKE, or abStract externAl library worKload Extractor, is presented in Saastamoinen & Kreku (2011). ABSINTH and ABSINTH2 by themselves do not generate workload models for library functions, since they are not normally compiled at the same time as the application source code. However, SAKE uses the Valgrind memory debugging, memory leak detection and profiling tool to automatically generate workload models for the library functions. Neither ABSINTH nor ABSINTH2 have dependencies on any particular source code language, so they should be able to generate workload models from any language supported by GCC. However, only the C and C++ front-ends have been used so far.

Summary

The selection of the load extraction method (Table 4) depends on:

– What kind of information is available of the application or applications
– How accurate the resulting models need to be
– How much effort the designer is willing to use to create the models.

In general, you will want to use as accurate method as possible for the best results. On the other hand, if you have only limited information available of the applications, you may be forced to use the analytical method. If you are modelling a small background load with only a minor impact on the overall performance, it makes sense to use one of the faster extraction methods.

3.2 Platform modelling

The execution platform model in ABSOLUT models both hardware and platform software components and interconnects (Kreku *et al.* (2004b, 2008b)). It is an abstracted hierarchical representation of the actual platform architecture. The models are created with the SystemC language using the transaction-level modelling (TLM2 Whitepaper) style. The role of the execution platform model is to process the workload primitives in the workload models and consume time in a cycle-approximate manner. The platform

Table 4. Comparison of the relative strengths and weaknesses of the load extraction techniques. The modelling effort column reflects the fact that support tools exist for measurement-, trace-, and compiler-based techniques.

Technique	Requirements	Modelling effort	Simulation speed	Accuracy
Analytical	Coarse-grained information of number of memory accesses and data processing operations	Average	Fast	Low
Measurement-based	Processor utilisation data	Average	Average	Average
Trace-based deterministic	Instruction trace	Low	Slow	High
Trace-based statistical	Instruction trace	Average	Average	Average
Compiler-based deterministic	Source code	Low	Slow	High
Compiler-based statistical	Source code	Average	Average	Average[1]

[1] With good profiling data

model also provides high-level services for the workload models implemented on top of the low-level primitives.

Abstractions used in ABSOLUT platform modelling approach aim at obtaining a sophisticated compromise between the platform models in typical high-level virtual system approaches and low-level instruction and cycle-accurate virtual platform approaches. The approach strives for reducing complexity to decrease modelling effort and enhance simulation speed while maintaining most of the accuracy.

From the abstractions in application modelling (Section 3.1) follows that the platform model need not contain full functionality for executing compiled application binaries. The workload models of applications are constructed of workload primitives instead of real instructions. Relieved of the burden of simulating the application functionality, the components in the ABSOLUT platform model can be pure performance models. They model just the component's execution capacity, which is consumed by the primitives in the workload models.

3.2.1 Layers

The ABSOLUT platform model is composed of three layers (Kreku *et al.* (2006)): platform architecture layer, subsystem layer and component layer. Each layer provides its own services, which are abstraction views of the architecture models. They describe the behaviour of the platform and related attributes — performance, for example — but hide other details. High-level services are built on top of low-level services, and they can also use the services implemented at the same layer. Each service may have many different implementations. This makes the design space exploration process easier, because replacing components or platforms with others can be easily done as long as they implement the same services.

Platform architecture layer

The platform architecture layer is the topmost layer, which incorporates platform software and serves as the portal that link the workload models and the platforms to the mapping process. The models at the platform architecture layer contain high-level services in addition to one or more subsystems. The platform-layer services consist of service declaration and instantiation information. The service declaration describes the functionalities that the platform can provide. Because a platform can provide the same service with several different ways, the instantiation information describes how a service is instantiated in a platform.

Application workloads typically call platform- or subsystem-level services, process workloads call subsystem services, and function workloads call component-level services. Ideally, all services required by the application are provided by the execution platform, and there is a 1:1 mapping between the requirements and provisions. However, often this is not the case, and the workloads need to use several lower level services in combination to produce the desired effect.

Subsystem layer

The subsystem layer is the next layer below the platform architecture and describes the components of the subsystem and how they are connected. The services used at this layer could include e.g. video preprocessing, decoding and postprocessing for a video acceleration subsystem.

The model can be presented as a composition of structure diagrams that instantiates the elements taken from the library. The load of the application is executed on processing elements. The communication network connects the processing elements with each other. The processing elements are connected to the communication network via interfaces.

Component layer

This layer consists of processing (e.g. processors, DSPs, dedicated hardware, and reconfigurable logic), storage, and interconnection (e.g. bus and network structure) elements. An element must implement one or more types of component-layer services. For example, a network interface component should implement both master and slave services. In addition, some elements need to implement services that are not explicitly defined in component-layer services, e.g. a bus must support arbitration and a network must support routing.

The component-layer read, write and execute services are the primitive services, based on which higher-level services are built. The processing elements in the component layer realise the low-level workload-platform interface, through which the workload primitives are transferred from the workload side. The processing element models will then generate accesses to the interconnections and slaves as appropriate. The platform model may contain several processing elements, which have different instruction sets in real life, i.e. simulations of heterogeneous computer systems are supported. However, the ABSOLUT models of all processing elements have the same set of abstract instructions; only the set of higher-level services may differ.

All the component models contain cycle-approximate or cycle-accurate timing information. The data path of processing units is not modelled in detail in contrast to cycle-accurate virtual platform models. Instead, the processor models have a cycles per instruction (CPI) parameter, which is used in estimating the execution time of the workloads. For example, the execution time E for data processing instructions is:

$$E = nC, \tag{4}$$

where n is the number of instructions to be executed and C is the value of the CPI parameter. It is the responsibility of the user to provide the values for the model parameters.

Interconnects are modelled at a nearly cycle-accurate level, if possible, to maintain simulation accuracy. TLM2-style local time offsets (TLM2 Whitepaper) can be used

to speed up simulation. Memory models in ABSOLUT do not need to provide data storage due to the abstraction decisions in workload modelling. Instead, they receive requests from processing elements, simulate the memory access latency and, finally, send responses back to the processing elements. No data is moved in the request or response transactions at all.

Processor caches and SDRAM memory page misses are currently modelled statistically since the workload models do not contain accurate address information. A potential future development of ABSOLUT would be to extend ABSINTH (Section 3.1.2) to output at least memory-addressing patterns in the workload models. The patterns could enable high-level deterministic simulation of caches and page misses.

3.2.2 Services

The services of the platform can model either hardware or software services. In ABSOLUT, software services are modeled as workload models, but unlike application models, they are integrated in the platform model and easily reusable by the applications (Kreku *et al.* (2009)). If the service is provided by a process or a set of processes running in the system, the service model consists of application- or process-layer workloads. If the service is implemented as a library, the model will be at the function layer. Service models can utilise other services, but eventually they consist of the same workload primitives as the application models.

There are two alternatives in how to implement a HW service: It can be implemented simply as a delay in the associated component, if the processing of the service does not affect the other parts of the system at all. In this case, the service must not perform I/O operations or request other services. The second alternative is to implement the service as workload primitives, which are executed inside the HW component and not inside a process workload running on one of the processor models.

3.2.3 Operating system

In simple cases, the execution of workload models can be scheduled manually by hard-coding it to the models. However, it is not recommended. Typically, the platform model includes one or more operating system (OS) models, which control access to the processing unit models of the platform by scheduling the execution of process workload models (Kreku & Tiensyrjä (2011)). The OS model provides both low-level workload

primitive and high-level services interfaces to the workloads and relays interface function calls to the processor or other models which realise those interfaces. The OS model will allow only those process workload models which have been scheduled for execution to call the interface functions. Re-scheduling of process workload models is performed periodically according to the scheduling policy implemented in the model.

The OS model extends the workload primitive interface with a number of methods for registering, starting, stopping, and killing process workload models. For mapping a process workload model to an OS model, the process must be registered. For the OS model to schedule the process for execution, the process must be started. Processes can also be stopped to temporarily disable them — while they are waiting for an external event, for example – and killed once they are not needed anymore.

If all the processors in the platform model have the same instruction set in real life, the operating system model can distribute the process workload models across all the processor models. However, if the system is heterogeneous, multiple operating system models are needed in the platform model: one for each different instruction set. The current implementation includes a round-robin scheduler, which makes its decisions based solely on the availability of the process workload and processor models in addition to the previous time the workload models were scheduled for execution. The algorithm can be replaced without changes to the rest of the OS model to simulate more complex scheduling decisions, which could take power consumption into account, for example.

High-level services are registered to OS models so that workload models can utilise them through a generic service interface. In the registration phase, the service provider, service name and service attribute types are informed to the OS model. The service provider can be any model, which realises the generic service interface, e.g. a platform component or a process workload model.

3.2.4 Model library

ABSOLUT Library of Existing models (ALE) is a collection of performance models for supporting model re-use and rapid construction of complex platform models. ALE contains low-level component (abstract base class), master, slave, and shared bus models. There are also generic processor, hardware accelerator, and memory models, whose purpose is to support creating new component models by extending the generic models. Finally, the set of fully-fledged models includes ARM and Intel Core i7 processor

models, SDRAM and SRAM memory models, a DMA controller, a display controller, network interfaces and routers, and a distributed shared memory engine (DME).

COGNAC, or COnfiguration GeNerator for ABSOLUT performanCe simulation, is an semi-automatic tool for generating complex platform models from building blocks. It takes two text-based files as input: The first one is provided by ALE and denotes, what kind of components exist in the model library and what kind of interfaces each component has. The second one is provided by the user and defines the structure of the platform at block level, i.e. which components should be instantiated and how the components should be connected. There is also additional support for constructing repetitive structures like the subsystems in a homogeneous NoC system. COGNAC generates the platform model, except for subsystem- and platform-layer services, which have to be added manually by the user.

3.2.5 Allocation of workloads on the platform

In order to facilitate simulation of the system model consisting of both the application and platform models, the applications need to be mapped to the platform (Kreku & Tiensyrjä (2011)). Mapping involves choosing the part of the platform, which will host (execute) each particular workload model, and which instruction and data memory or memories they will utilise. Mapping the execution is done during the initialisation of the workload model, i.e. each workload model receives a pointer to its host as a parameter. The mapping is done in several layers in such a way that

– Application workload models are mapped to subsystem models
– Process workload models are mapped to operating system models inside subsystems
– Function workload models are mapped to processing unit models.

The operating system model is already mapped to a fixed set of one or more processors in its initialisation. Thus, the process model will just distribute its pointer to the OS model when mapping the functions to the processing units.

Memory mapping is done via memory addresses, which are used as parameters to high- and or low-level interface calls. These addresses could, for example, be hard-coded in the models. However, the recommended way is to set these up as model parameters in the process workloads, and the process workloads then distribute the addresses to function workloads during their initialisation. The addresses used during simulation

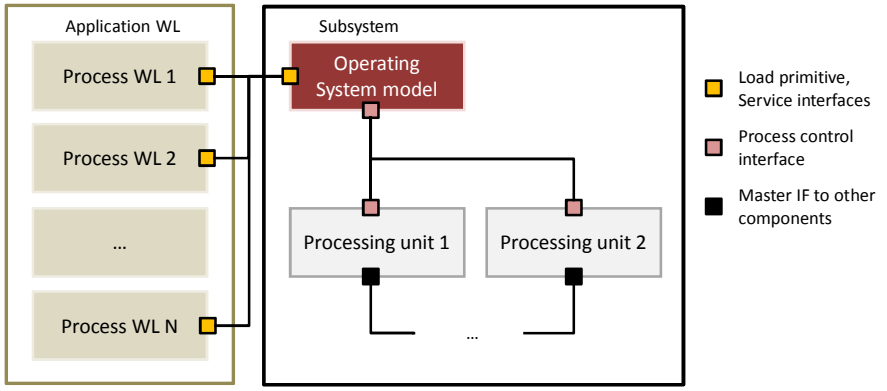

Fig 5. Process workloads send load primitives and service calls to the platform model during simulation (Kreku *et al.* (2008b)).

are then defined in the configuration file of the system model. Thus adjusting memory mapping does not force the user to recompile the models.

3.3 Performance simulation

Workload models utilise the resources of platform models with workload primitives and high-level service requests (Figure 5). The operating system models receive the requests, propagate the requests from the processes scheduled for execution, and block the others. The requests are passed through to the processing elements and other service providers in the platform model.

The processing elements simulate the execution time of data processing instructions. Memory accesses are also processed by the processing elements, but they may propagate to the other components of the system, e.g. interconnects and memories, if the element does not have cache memory or if the cache misses. The platform model advances simulation time while it is processing the workload primitives and service requests. Simulation will continue until it is explicitly stopped by one of the workload models when the use case has been completed.

3.3.1 Performance probes

The platform model is instrumented with counters, timers, and probes, which record the status of the components during the simulation. These performance probes are manually

inserted in the component models where appropriate and are flexible so that they can be used to gather information about platform performance as needed. Typically,

1. Status probes collect information about utilisation of components and scheduling of processes performed by the operating system models. They can provide average utilisation percentage of any component for each performance and power state. They are also able to periodically write temporary utilisation to a file. Viewer Of collecteD Key information for Analysis (VODKA) is a tool, which can visualise the files and show utilisation, power and energy consumption curves as a function of time.
2. Counters are used to e.g. calculate the number of load primitives, service calls, requests, and responses performed by the components.
3. Timers are used to e.g. keep track of the task switch times of the OS models and processing times of services.

The probes support the setting of soft and hard constraints. If a soft constraint is not met during simulation, a warning message will be displayed. An unmet hard constraint will stop the simulation.

The models in the ALE library (Section 3.2.4) contain a default set of probes for extracting the most commonly required performance and power consumption data from the system. The probes output the performance data to the standard output once the simulation is complete. The data can be analysed and feedback can be given to application or platform designers. For example, if the utilisation of components is low, lowering the clock frequency can be proposed for decreasing power consumption.

3.3.2 Simulation environment

ABSOLUT uses the IEEE standard OSCI SystemC 2.2 (Open SystemC Initiative website) library for simulation. The system model is built in a GNU/Linux environment using GNU Compiler Collection (GCC) (GCC) and CMake (CMake) cross-platform build system. The simulator, BEER, is a command line program executed in a terminal window. During simulation, it prints progress information to standard output, and once the simulation is completed, it displays the collected performance results.

In a mobile video player case (Kreku *et al.* (2007)), the simulation speed was one tenth of real time. In other words, simulating the system for one second took about ten seconds in a Linux PC with Intel Xeon processor. ABSOLUT is fast enough for performing early-phase performance evaluation and for simulating multiple, alternative

use cases for architecture exploration, which will be shown in Chapter 4. However, the simulation speed could still be improved by optimising the models: the existing models are proof-of-concept models, where ease of debugging has been more important than simulation time. Furthermore, the models are based on OCP-IP protocol models using TLM1 library and lack the simulation time improvements brought by TLM2 (TLM2 Whitepaper).

4 Case studies

ABSOLUT has been applied to a number of case studies, where the goal of the study has been to evaluate the performance capabilities of existing and future platforms, explore platform sizing and parallelisation, and explore application partitioning, to name a few. The following sections describe the applications and platforms, results obtained from the simulations, and how the results were validated, if possible.

4.1 OMAP1-based platforms

Several case studies have used the Texas Instruments' OMAP1 (Chaoui *et al.* (2001)) family of mobile multimedia processor SoCs as their execution platforms. The OMAP1 consists of ARM9 MCU, Texas Instruments C55x DSP, SDRAM and SRAM memories as shown in Figure 6. There is also a traffic controller, which allows the CPU and DSP to access the memories.

4.1.1 Mobile phone usage scenarios

Mobile phone usage scenario 1 (Figure 7) consists of simultaneous bluetooth download and application list browsing. The length of the case is about 54 seconds in total and everything is executed on the MCU. Mobile phone usage scenario 2 has a lot of activity on the MCU, including but not limited to a GPRS connection, SMS and MMS

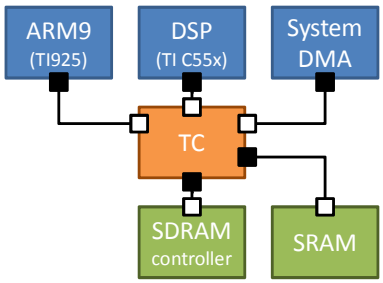

Fig 6. Block diagram of the structure of OMAP1 SoC. Black and white boxes in the connections represent master and slave ports, respectively.

messaging and video capturing in addition to MP3 decoding on the DSP. The length of this case is slightly over six minutes in total. The workload models were modelled using the measurement-based approach (Section 3.1.2) from separate measurements of individual programs. The performance measurements of the programs were made in a real prototype platform consisting of the OMAP1510 processor and Symbian operating system.

The use cases could not be measured in their entirety due to software maturity problems on the prototype platform. For comparison purposes, the average utilisation values of the individual measurements were summed, although this induces a systematic error, because task switching, scheduler time slices and other operating system effects are not taken into account. Thus, the evidence with respect to accuracy of ABSOLUT simulation results provided by this case is less reliable than those provided by the other case studies.

Table 5 displays the accuracy of ABSOLUT simulation results in comparison to measurements in both mobile phone usage scenarios. The error percentages E have been calculated using the following formula:

$$E = |1 - \frac{S}{M}|, \tag{5}$$

where S is the value obtained from simulation and M from measurements. In this case, the values represented the ARM MCU and DSP utilisation in the OMAP platform (Kreku *et al.* (2004b)). The maximum and minimum error percentages were 25% for use scenario 1 MCU load and 13% for use scenario 2 DSP load, respectively. The comparison is limited to processor utilisation because it was not possible to measure anything else on the prototype platform. However, ABSOLUT is able to provide much more information about the system with the performance probes presented in Section 3.3.1.

4.1.2 MP3 playback

This use case consists of only MPEG1 audio layer 3 (MP3) playback, which is decoded by the DSP of the OMAP1510 SoC for one second (Kreku *et al.* (2004b)). The decoder is using both the internal memory of the DSP and external SDRAM. The workload models for this case were based on the source code of a freely available mpg123 MP3 decoder (MPG123) and modelled manually using the source code- or compiler-based approach

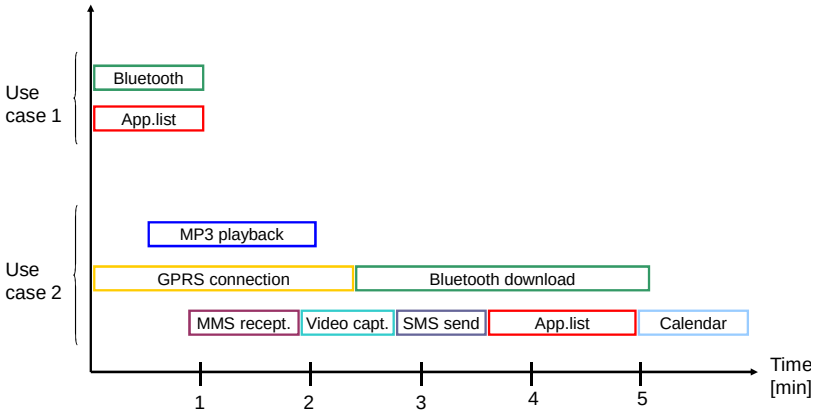

Fig 7. Use cases 1 and 2 of the mobile phone usage scenarios.

Table 5. Comparison of simulation and measured results for OMAP1510-based platforms.

	Simulation, S	Measurement, M	Error, E
Mobile phone use case 1 MCU load	35%	28%	25%
Mobile phone use case 2 MCU load	43%	36%	19%
Mobile phone use case 2 DSP load	9%	8%	13%
MP3 playback DSP load	19%	19%	0%

of Section 3.1.2, but without the ABSINTH tool. Table 5 provides a comparison between the DSP load obtained from simulations and measurements made using the same prototype OMAP platform as in Section 4.1.1. The DSP load in the simulation results is exactly the same as in the measurement results, which results to an error percentage of 0%.

4.1.3 Partitioning of MPEG4 encoding

The partitioning of an MPEG4 encoder (Figure 8) for the OMAP5912 platform was studied in this case example (Kreku *et al.* (2005)). The video encoder used in the task was the open source libavcodec library with ffmpeg front-end, which was partitioned to

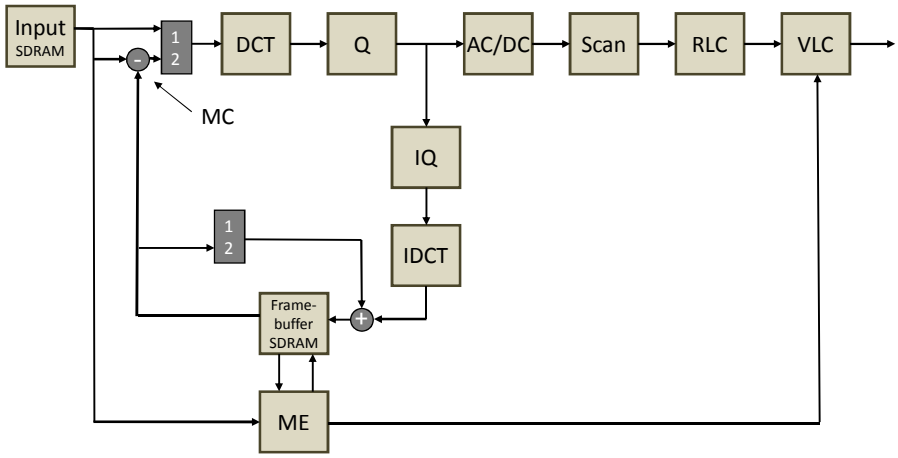

Fig 8. Block diagram of the MPEG-4 encoder (Kreku *et al.* (2005)) (©2005 IEEE).

utilise both the DSP and the ARM sides of the OMAP5912 SoC. Based on the hardware accelerators available in the DSP and on the profiling results of the MPEG-4 encoder, two partitioning alternatives were considered for the discrete cosine transformation (DCT) and sum of absolute differences (SAD) calculation inside motion estimation. The DCT and SAD were mapped to either the ARM or the DSP independently of each other. All other parts of the encoder were executed on the ARM processor.

The workload models were created using a variant of the source code- or compiler-based approach without the ABSINTH tool. During the simulation runs we measured the achieved frame rate and utilisation of all the components of OMAP. For validating the simulation results, the ffmpeg encoder was executed on an OMAP5912 development kit. The encoding frame rate and ARM and DSP processor utilisation were measured. During the course of the work, it was noticed that the communication between the processor included a notable overhead on single macro block size data transfers. With ABSOLUT, it was also possible to estimate the performance potential of the architecture with a lower communication overhead to explore, how it would affect the partitioning decision.

There was virtually no difference between the obtained DSP load values resulting to the minimum error percentage of 2%. On the other hand, the frame rate achieved with the OSK5912 kit was 24% lower than in the simulator (Table 6). Simulations with the low communication overhead showed a clear improvement in the frame rate, though the performance still was not clearly better than in the all-MPU case.

Table 6. Comparison of simulation and measured results for OMAP5912-based platforms.

	Simulation, S	Measurement, M	Error, E
All-MCU MPEG4 encoding FPS	15.9	17.8	11%
DSP-accelerated MPEG4 encoding FPS	8.3	6.7	24%
DSP-accelerated MPEG4 encoding DSP load	18.9%	18.5%	2%

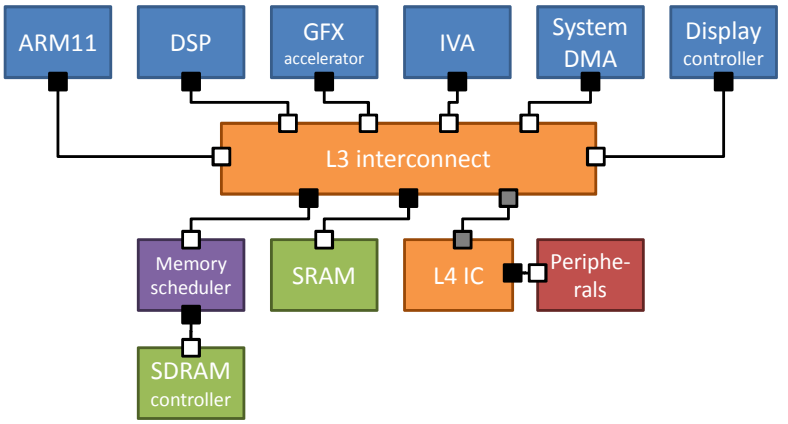

Fig 9. Block diagram of the structure of OMAP2 SoC. Gray boxes in the connections denote that the link has both master and slave ports.

4.2 OMAP2-based platforms

OMAP2 is a newer, more complex multiprocessor SoC than OMAP1 and contains a faster ARM CPU and DSP. It has a hierarchical multi-level interconnect and several hardware accelerators, including 2D and 3D graphics (GFX) and video (IVA) accelerators (Figure 9). It has been used as the execution platform for two ABSOLUT case studies about Quake 2 gaming and virtual network computing (VNC).

4.2.1 Quake 2 gameplay

The Quake 2 3D game (Quake 2) was modelled in this case study. The purpose of the study was to evaluate the capability of a future mobile device based on the OMAP2

platform in executing a contemporary 3D PC game. The Quake2 game was modelled and simulated with two different display sizes and frame rates.

Quake2 consists of three main modules: game core, audio and OpenGL graphics functionality, which were mapped to the ARM, DSP and graphics accelerator, respectively. The workload models were mostly based on the source code of Quake2; however, the models of OpenGL API functions were based on profiling data.

The utilisation of each component in OMAP2 was obtained with both display resolutions and frame rates. The simulations indicated that only the high resolution and high frame rate combination would be problematic for the platform with the graphics accelerator and SDRAM memory limiting the system performance. However, the clock frequencies of the components used in the simulations turned out to be lower than what was available in final OMAP2420 and OMAP2430 SoCs. It was not possible to measure the performance of Quake 2 on OMAP2 for comparison purposes since OMAP2 was still in development at the time of the study.

4.2.2 Virtual network computing

Virtual Network Computing (VNC) is a remote controlling software which allows to view and fully interact with a VNC server. In this use case, the VNC viewer program is executed on a mobile terminal (Figure 10) for internet browsing. The platform consists of an OMAP2-based computer with Linux OS, touch screen, buttons, sound, USB 2.0, microSD, WLAN wireless networking, and bluetooth connectivity. Workload models were created by extracting the load of the RealVNC application using analytical and source code-based techniques.

The system model was instrumented to extract the ARM CPU load, average frame rate, and average network traffic achieved in the simulations. For validation purposes a VNC application was executed on TI OMAP2430 SDP development board and corresponding information was measured from the platform. The simulation results were quite close to the results from the measurements: The load of the ARM11 CPU was underestimated in the simulations by about 15%, whereas the frame rate and network traffic were overestimated by about 11% and 14%, respectively (Table 7).

The WLAN connection to the server was found to be the main reason for the relatively low utilisation of the components in the platform. In order to find out, how the platform would cope in the same use case with an ideal network, the models were modified accordingly. According to the simulations, the platform would have achieved

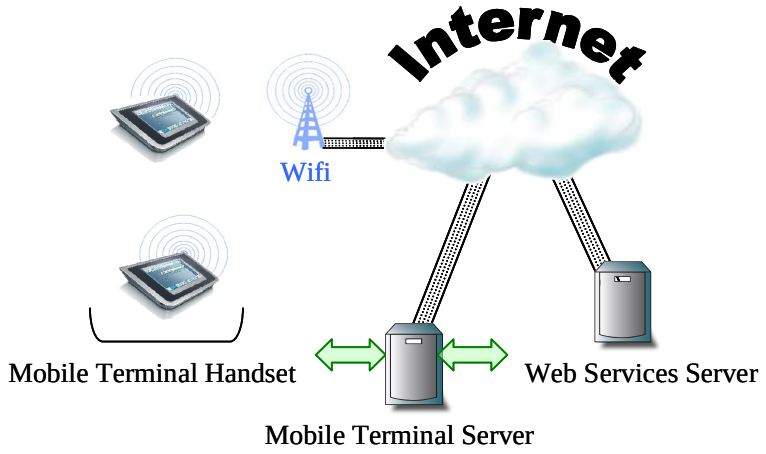

Fig 10. Mobile VNC terminal concept. (Kreku *et al.* (2008a))(©2008 IEEE).

Table 7. Comparison of simulation and measured results for OMAP2-based platforms.

	Simulation, S	Measurement, M	Error, E
VNC use case CPU load	9.2 %	10.8 %	15%
VNC use case FPS	6.0	5.4	11%
VNC use case network traffic	179 kB/s	157 kB/s	14%

ten-fold increase in frame rate, at which point the CPU and display controller would have been at the limit of their performance.

4.3 Intel Core i7-based personal computer platforms

Intel Core i7 975 (Intel Nehalem Microarchitecture) is a 3.33 GHz CPU for high-end personal computers and workstations. It contains eight logical processor cores divided into four physical and four virtual cores. Each physical core contains independent L1 and L2 caches, and the processor has a shared L3 cache in addition. Integrated memory controller handles accesses to external SDRAM memories.

Saastamoinen *et al.* (2011) models a network traffic analysis application, which monitors internet traffic to ensure quality of service and to detect intrusions and other malicious activities. The application is multi-threaded with POSIX threads and runs as a

deamon in the Linux operating system. The ABSINTH tool was used to model the application from the source code and the pthread threading library was modelled as services on the Intel Core i7 platform model. Processor utilisation for each virtual and physical core was obtained from simulations. It was noticed that mutex locking between the application threads was causing one of the threads to be starved for execution time.

Saastamoinen & Kreku (2011) uses the ABSOLUT approach and the ABSINTH1, ABSINTH2 and SAKE tools to model parallel application kernels for the same Core i7 platform. The kernels include fibonacci sequence calculation, sparse matrix manipulation, and multi-threaded matrix multiplication.

4.4 Future platforms

These case studies represent examples, where ABSOLUT has been used to evaluate the performance of complex future platforms.

4.4.1 Distributed mobile video player

In the mobile video player (MVP) case example, a mobile terminal user wants to view a movie on the device and selects one from a list of movies available on the mobile terminal (Kreku *et al.* (2009)). The execution platform provides services for storing and playing of movie files.

The platform in the mobile video player case consists of four heterogeneous subsystems (Figure 11):

1. General purpose (GP) subsystem, which is used for executing an operating system and generic applications and services
2. Image (IM) subsystem, which accelerates image and video processing
3. Storage (ST) subsystem, which contains a repository for video clips
4. Display (DP) subsystem, which takes care of displaying the device UI and video.

The subsystems are interconnected by a network using a ring topology. Major parts of the activity in the MVP case study are modelled as services in the platform model. The workload model of the MVP application utilises those services to display a list of video files to the user of the system and then initiates playback of a user-selected file. A generic background load was added to the GP subsystem to model the execution of other applications in the system.

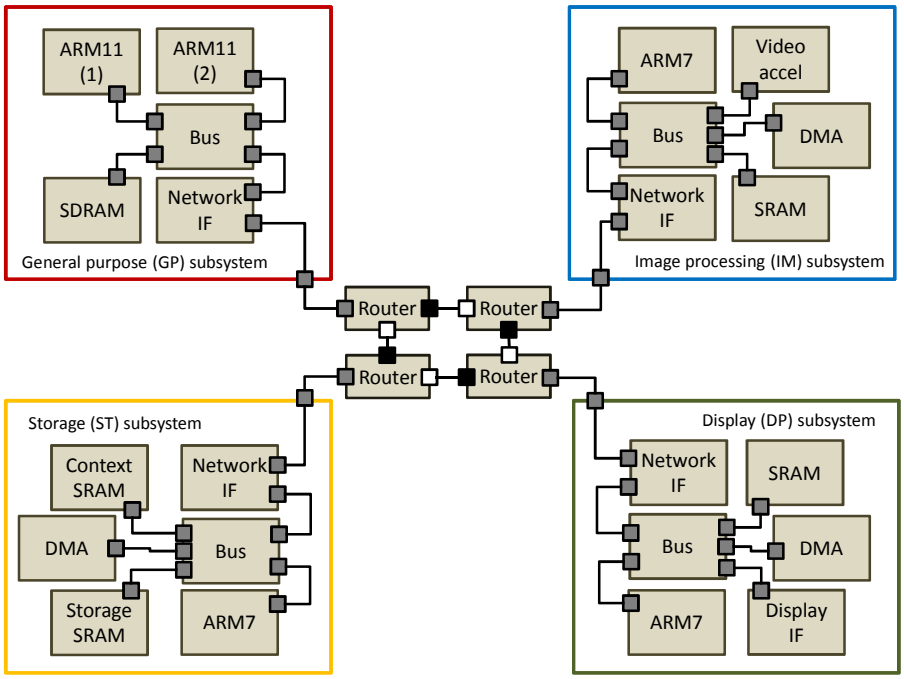

Fig 11. The Mobile Video Player platform consists of four subsystems connected via ring network (Modified from Kreku *et al.* (2007)).

The MVP application and the platform consisting of 27 components were modelled with the ABSOLUT approach in less than a week. The system was successfully simulated and potential performance bottlenecks were identified. The platform was able to perform video playback without problems according to the simulations. However, it was estimated that the video accelerator of the IM subsystem might not be powerful enough for video recording, and increasing its clock frequency was proposed as a solution.

4.4.2 Parallel WiMax SDR

This case study (Ieromnimon *et al.* (2011)) concentrated on the parallelisation of IEEE 802.16e PHY software implementation. Several parallelisations of sequential source code were explored with IMEC MPA parallelisation tool (Mignolet *et al.* (2009)), and their speed-ups were simulated with MPA high-level simulator. Based on the high-level

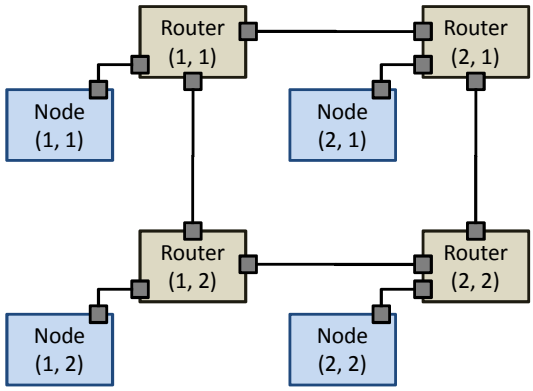

Fig 12. 2x2 mesh execution platform for the parallel WiMAX SDR application.

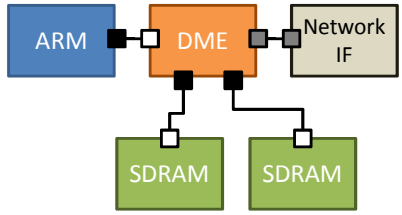

Fig 13. Block diagram of a node in the execution platform for the parallel WiMAX SDR application.

simulator results the best parallelisations were chosen for more detailed evaluation with ABSOLUT.

The platform for the application was a KTH McNoC (Millberg *et al.* (2004)) Network-on-Chip platform with either a 2x2 or 2x3 2D mesh configuration, the smaller of which is displayed in Figure 12. All nodes in the mesh are alike and consist of an ARM11 processor, Distributed shared Memory Engine (DME), SDRAM memory, and a network interface (Figure 13). The DME (Lu *et al.* (2010)) provides distributed shared memory accessing to the system by controlling the processors' accesses to local private, local shared and remote shared memory regions.

Execution time, utilisation, speedup, power consumption, and GIPS per Watt figures were extracted from the system with ABSOLUT simulations. The obtained speedups were lower than anticipated, and the utilisation values indicated that it resulted from uneven load balancing. New parallelisations were created and simulated according to

feedback from ABSOLUT to solve the load balancing issue. According to Ieromnimon *et al.* (2011), ABSOLUT and MPA together provided a 5-fold productivity improvement compared to traditional design methods in the case study.

4.4.3 Parallel SDR sensing application

The application in this case study is the software implementation of the sensing algorithm of cognitive radio in a Software Defined Radio (SDR) terminal (Aguirre & Candaele (2011)). The terminal analyses the spectrum based upon 40 MHz bandwidth slices and looks for a GSM transmission, which means that the parameters of 200 channels of 200 kHz each have to be extracted. Again, the sequential source code was parallelised with MPA and the efficiency of the parallelisation was initially estimated with the MPA high-level simulator.

The platform is related to the WiMAX study platform, but with a number of key differences. The network is exactly the same KTH McNoC (Figure 12), but the number of nodes varies from 2 to 54 depending on the parallelisation of the application. At most, there were more than 300 components in the system. Each node has a custom ASIP processor with two parallel execution units, the DME, two SRAM memory blocks, and a network interface.

The execution time of the sensing algorithm, speedup compared to the sequential version, GIPS / Watt and node utilisation figures were extracted from ABSOLUT simulations. Next, energy-delay product was calculated from ABSOLUT results to identify the most efficient parallelisation. Finally, the simulation results were validated with instruction set simulation of the application using the Synopsys Processor Designer. The comparison was performed only for the sequential version of the application since the Processor Designer is not able to simulate the parallel systems. The execution time given by ABSOLUT was 3.2% higher than the one given by Processor Designer.

5 Discussion

The goal of this thesis was to develop a novel approach for the performance evaluation of future embedded systems. The requirements for the approach included that it needs to be usable already in the early phases of the design flow, and that the results are accurate enough for making design decisions based on them. The research hypothesis was that by using workload models consisting of abstract instructions, the above goals could be achieved. Furthermore, it was proposed that the load information for creating the workload models could be extracted from several different sources, including application specifications, traces, measurement data, and the source code.

5.1 Analysis of results

The results from the case studies in Chapter 4 prove that it is possible to do early-phase system-level performance evaluation with non-functional workload models of applications. This thesis presented nine case studies, where the ABSOLUT approach was used to evaluate the performance of an embedded computer system. Many of the case studies explored several application, execution platform or mapping alternatives. The Quake 2, VNC, MVP, WiMax, and sensing application case studies were completed before the execution platform of the case study had been developed. The mobile phone usage case studies were performed before the applications were mature, and the MVP case study without any application software at all.

Analytical, measurement-, trace- and source code-based workload modelling techniques were developed for ABSOLUT and used in the case studies. Further techniques could be developed based on what kind of data is available of the applications for creating the models.

The analytical technique turned out to be useful for the early simulations since it can be used once there is some kind of specification available about the work the application does. This was demonstrated in the MVP case study. The analytical technique is best suited for applications without an extensive amount of control, e.g. signal processing applications, so that the modelling effort remains low. The models created with the analytical technique can be replaced with more detailed measurement-, trace-, or source code-based models for later simulations. They require that the application

or a reasonably similar application exists and its performance can be measured in a resembling architecture, its execution can be traced, or it can be compiled in a workstation. Those techniques can utilise a previous generation of the same application or a different application developed for the same purpose as input. The complexity of the workload models obtained can then be easily scaled upwards or downwards as necessary.

The MVP case study also showed that the control part of the workload models can be generated from existing UML application models, which enables reuse of design models for performance evaluation and further reduces modelling effort. The workload primitives could also be obtained if source code were first generated from the UML models and then ABSINTH was used to create the workload models from the source code. Another possibility would be to explicitly model the primitives in UML using e.g. statechart diagrams.

The results of the case studies show that the complexity of the workload models varies a lot depending on the technique used for creating the models. However, the model complexity can be very low especially for models created using the analytical technique. On the other hand, the complexity of models created from the application source code approaches that of compiled, executable binaries. The abstractions used in modelling the applications also reduce the complexity of the execution platform models: The data paths of processing elements need not be modelled, hardware accelerators can be modelled as black boxes providing services to the rest of the system, and the memory models do not need to provide data storage. No data is moved in the transactions during simulations.

The required modelling effort was low. A component model for the execution platform can be created in one day from scratch, assuming that the user is familiar with the ABSOLUT approach and the component to be modelled. In other words, it takes more time to study the architecture of the component and obtain values for model parameters than it takes to create the model itself. The subsystem and platform layers of the execution platform models can be created in minutes or hours depending on the complexity of the platform with the assistance of the COGNAC tool. However, that does not include the time required for modelling subsystem- or platform-layer services. The effort required by workload modelling depends on the chosen technique. The analytical technique turned out to have the highest manual effort despite the lowest detail level because automatic tools and scripts were developed for the other techniques.

For example, the ABSINTH tool was able to produce workload models as a part of normal source code compilation flow.

Simulation speed depends on the complexity of the simulated system and the simulation hardware. For example, the simulation of the mobile video player application on a four-subsystem platform consisting of about 30 components took ten seconds for one second of video playback. The workstation running the simulation was a modern Linux PC with a Intel Core i7 processor. The workload models in the MVP case study were hand-written models using the analytical technique. Automatically generated models in other case studies were 2–5 times slower, partially due to higher level of detail and partially due to the inefficiencies of the generated code. Simulation speed was not compared to other simulation approaches due to the effort it would have required. However, simulation speed is improved compared to cycle-accurate or cycle-approximate instruction set simulation of a similar platform on the grounds of the higher abstraction level used in ABSOLUT.

The absolute error percentages from the comparisons between ABSOLUT simulations and measurements or instruction set simulations in Chapter 4 are collected in Table 8. The largest error was in the MCU load of the mobile phone use case 1, 25%. The smallest errors were with MP3 playback DSP load, DSP-accelerated MPEG4 encoding DSP load, and sensing algorithm execution time, which were all 3% or less. The arithmetic average error of all cases is $12\% \pm 8\%$. According to Kogel *et al.* (2005) an accuracy of 70–80% is enough for an architect's view simulation approach, which has been achieved.

Fidelity refers to the degree to which a model or simulation reproduces the state and behaviour of a real application and/or platform. ABSOLUT concentrates on performance only and does not simulate the functionality of the applications. Thus, the only potential source of poor fidelity would be the statistical modelling of control on the application side. Statistical control works well for applications, where data processing is dominating the execution, which was shown in the parallel JPEG encoder case study. However, for control-dominated applications, the control in the workload models may not always follow the desired path, resulting to poor fidelity unless the simulation is repeated a large number of times. Therefore, it is recommended to use deterministic control in the workload models for best results.

Table 8. Error percentages of simulations in the case studies, which were validated with measurements in real systems.

Measurement	Error, E
Mobile phone use case 1 MCU load	25%
Mobile phone use case 2 MCU load	19%
Mobile phone use case 2 DSP load	13%
MP3 playback DSP load	0%
All-MCU MPEG4 encoding FPS	11%
DSP-accelerated MPEG4 encoding FPS	24%
DSP-accelerated MPEG4 encoding DSP load	2%
VNC use case CPU load	15%
VNC use case FPS	11%
VNC use case network traffic	14%
Sensing algorithm execution time	3%
Average	12%
Standard deviation	8%

5.2 Theoretical and practical implications

ABSOLUT has proven to be an efficient approach for early phase performance evaluation due to the various reasons referred to in the previous section. The abstractions used in the approach work and ABSOLUT is a more general purpose approach than the other virtual system or architect's view approaches presented in Chapter 2. It is not limited to any specific application field or platform type. It does not require the execution time for (parts of) the application models as input, which would result to the problem of how to obtain those execution times. Instead of proprietary models, ABSOLUT uses an IEEE standard modelling language, SystemC, and models based on the TLM standard and common OCP protocols. Even if interoperability with other modelling approaches is not straightforward as a result of different abstractions, due to compliance with standard interfaces, it is at least possible to achieve.

The case studies performed with ABSOLUT confirm the claims of Kogel *et al.* (2005) with respect to architect's view: More accurate results are produced than by fast, i.e. loosely-timed, programmer's view approaches. Furthermore, less modelling effort is required compared to PV or VV approaches due to lower level of detail in the models. ABSOLUT, however, is not the Swiss army knife of simulation approaches. It is capable

of filling a slot between more coarse-grained early evaluation techniques and more accurate methods able to simulate the functionality of applications. It is a useful part of a design flow, not the entire design space exploration or design flow by itself.

One very beneficial property of a virtual system-based approach turned out to be the ease of constructing models of very large platforms and getting the simulations running. For example, the largest variant of the platform used in the sensing application case study had 54 subsystems with more than 300 components in total. The extra effort required by this platform in comparison to the smaller variants consisted of writing larger configuration files for the COGNAC platform model generator and for the parameters of each component in the platform. A much larger effort is required to set up a functionally-correct virtual platform model of a similar platform. Evaluation speed has been fast enough in the case studies presented in this thesis. However, it needs to be further improved to combat the ever-increasing complexity of the future systems. Partially, it can be achieved with optimisations of models, but also the abstraction level still needs to increase in the future.

Creating a platform model with ABSOLUT requires only proper documentation of the components and subsystems of the platform. A block diagram of the architecture showing the components and their connections is needed as well as values for capacity parameters like clock frequency, data width, etc. In practice, however, obtaining the documentation can be very difficult. Often, the documentation, which reveals the required amount of detail, is not publicly available, and some kind of leverage is needed to obtain the documentation from the vendor of the component or platform. One partial solution to the problem is to have a large model library so that extensive design space exploration can be performed without resorting to creating new component models. The values for statistical performance parameters, which include CPI and cache hit probability parameters for processors, can be obtained by executing the application in an instruction set or cache simulator, respectively. In the worst case, typical values for the processor for a generic workload can be used instead at the cost of accuracy.

Even if both application and platform models are simple, there is a relatively high learning curve with ABSOLUT. The concept of simulating non-functional workload models may be difficult to grasp initially. People expect to use ABSOLUT simulations for software development even if it was designed for performance evaluation purposes. It was also noticed that people that only run simulations without developing models themselves may not trust the simulation results when they do not concretely see what is going on in the simulations. This problem could be alleviated by developing

service models that provide a partial, selective functionality for the workload models of applications. For example, a printf service model could be developed to print the same strings to standard output as the real C library implementation would.

There are two important drawbacks with ABSOLUT currently, one theoretical and one practical. The workload modelling techniques presented in Section 3.1.2 are not able to produce accurate memory addresses to workload models currently. Thus, it is difficult to model address-dependent behaviour in the memory hierarchy accurately, and the platform models in the case studies used statistical models of caches, for example. This is an interesting topic for future work, which will be elaborated on in Section 5.4.

The second drawback is the lack of special support for debugging. The normal means for debugging provided by SystemC, C++ and software tools are of course usable with ABSOLUT. However, earlier in this chapter it was mentioned that studying the architecture of a component typically takes more time than modelling it. The same applies also for the debugging of the model. Even if a model works correctly in one case study, it does not mean that it will do so in all the corner cases. If a simulation stops unexpectedly, it can be very hard to find out, what was the original cause for the stop. For example, additional support for tracking of transactions in the models and visualisation of the inner workings of the platform model would help alleviate the problem.

The target user for ABSOLUT is a platform developer interested for developing new platforms or extracting more performance from existing platforms, an application developer concentrating on performance issues with ABSOLUT and using other means for evaluating functionality, or a system developer exploring applications, platforms or mappings. For adoption by the target users, ABSOLUT needs improvement also in ease of use. The existing tools are at proof-of-concept level, and there are no graphical user interfaces for modelling, configuration or simulation. More automation is needed so that performance evaluation with ABSOLUT can be better integrated with the normal design flow instead of being a separate effort.

5.3 Reliability and validity

An extensive set of case studies have been modelled to expirement with the ABSOLUT approach. The applications in the case studies are from different fields including multimedia, gaming and networking. Two of the case studies, WiMax software radio and GSM sensing application, were performed by third parties not involved in ABSOLUT

development. The studies showed that differences between simulation results and measurements made with the same applications in a real platform were rather low, reaching an average of 12%. The error percentages were in the same ballpark in each case, resulting to a standard deviation of 8%. However, the case studies are only snapshots that show a trend. They do not rule out that the error could not be higher than what can be considered acceptable for early evaluation in a particular case study. Much of the data obtained from simulations can not be verified simulations due to immaturity of software and hardware for performing the measurements and lack of visibility in the hardware.

5.4 Recommendations for future work

Currently, ABSOLUT operates at a single abstraction level on the platform side, whereas on the workload side, the analytical technique can be considered to produce models at a higher abstraction level than the other techniques. Incorporation of consistent support for higher abstraction level simulation would be useful for even earlier evaluation and for assisting with the evaluation of even more complex systems. At the same time, improved accuracy would be beneficial for the later stages of the design flow. This could be achieved by taking the data paths of processing elements into account with static estimation techniques like in Kreku & Soininen (2003), or by supporting also instruction set simulation. ABSOLUT could implement a full design exploration flow with support for models at various abstraction levels and additional tools implementing design space exploration algorithms like a number of the approaches presented in Chapter 2. Integrated support for the calibration (Pimentel *et al.* (2008)) of the statistical parameters of the platform model would help increase the accuracy of the results.

A more difficult research problem is how to get accurate address information to workload models, which would enable more accurate modelling of the memory hierarchy. Currently, the various workload modelling techniques do not provide means for obtaining the addresses automatically. The designer has to map parts (code and data for basic blocks, functions, processes, etc) of application to one of the memory components of the platform. In other words, memory addresses are considered only at the block level. ABSINTH could be extended to produce addresses during compilation or run-time; however, an open question is what could be done with the other workload modelling techniques to resolve the problem.

Improving the accuracy of the approach is difficult if it is not known, where the errors come from. The case studies presented in Chapter 4 are not able to quantify, whether the majority of the errors results from the abstractions in the workload or platform models. More experiments where combinations of applications and platforms are simulated and measured are required in the future.

Simulation speed has been adequate for the case studies but could be improved for faster exploration and evaluation of very complex systems. Transforming the models in the ALE library from TLM 1.0 to 2.0 is one potential way to enhance evaluation speed. Another is to eliminate bottlenecks from the models. For example, currently the workload primitives are implemented as function calls as shown in Section 3.1.1, which results to a considerable number of function calls during simulation. This could be optimised by passing basic blocks or entire functions at once from the workload models to the platform model. There are further bottlenecks especially in the ABSINTH2-generated models, which utilise run-time parsing of trace and XML files. This could be moved offline to speed up simulation. Adoption of a parallel SystemC kernel, which has been research for example in Cox (2005), Chopard *et al.* (2006), Ezudheen *et al.* (2009), would allow utilising the performance of contemporary workstations more efficiently. A more research-oriented problem would be to explore how to reach higher abstraction levels without sacrificing accuracy of the results.

6 Introduction to papers

This thesis includes nine scientific publications about the ABSOLUT performance evaluation approach. Paper I is a research arcticle published in an open access journal, whereas Papers II and III are book chapters. The rest of the papers have been published in conference proceedings. Jari Kreku is the first author of each included publication and they are mostly based on his research work.

Table 9 shows, how the description of the modelling approach is distributed to the publications. Kreku *et al.* (2008b) presents the overall ABSOLUT approach as it was at the beginning of 2008. Kreku *et al.* (2009) adds more details to the interfaces used in the workload and platform models and to the modelling of services. Kreku & Tiensyrjä (2011) expands ABSOLUT for system-level power simulation by utilising the performance probes in combination with component-level power consumption input. The analytical workload modelling technique is first presented in Kreku *et al.* (2008b), the measurement-based technique in Kreku *et al.* (2004b), manual source code-based technique in Kreku *et al.* (2004a) and automatic source code- / compiler-based technique in Kreku *et al.* (2010). Kreku *et al.* (2005) presents an initial technique to create workload models from UML application models, which is then improved on in Kreku *et al.* (2007, 2008a).

Table 10 reveals, which case studies are presented in each publication. The case studies are used to demonstrate that the ABSOLUT approach can be used to evaluate performance at the system level before implementations of applications and/or platform exist. The results from a number of the case studies are verified by comparing them to measurements obtained by executing the same applications in a real platform. The mobile video player case study is shown in Kreku *et al.* (2008b, 2009, 2007, 2008a). The parallel JPEG encoding study is depicted in Kreku *et al.* (2010), Kreku & Tiensyrjä (2011). Mobile phone usage scenarios can be found from Kreku *et al.* (2004b), Quake 2 gaming case study from Kreku *et al.* (2004a), and MPEG4 encoding case study from Kreku *et al.* (2005). Finally, Kreku *et al.* (2008a) contains the virtual network computing case study.

Table 9. Description of the ABSOLUT modelling approach in the publications.

No	Publication	Modelling approach						
		Platform		Workload				
		Perf[1]	Pwr[2]	A[3]	M[4]	T[5]	S[6]	U[7]
I	Kreku *et al.* (2008b)	x		x	x		x	x
II	Kreku *et al.* (2009)	x						x
III	Kreku & Tiensyrjä (2011)	x	x				x	
IV	Kreku *et al.* (2004b)	x			x			
V	Kreku *et al.* (2004a)	x					x	
VI	Kreku *et al.* (2010)						x	
VII	Kreku *et al.* (2005)							x
VIII	Kreku *et al.* (2007)							x
IX	Kreku *et al.* (2008a)							x

[1] Performance modelling

[2] Power modelling

[3] Analytical technique

[4] Measurement-based technique

[5] Trace-based technique

[6] Source code-based technique

[7] UML front-end for workload modelling

6.1 Combining UML2 Application and SystemC Platform Modelling for Performance Evaluation of Real-Time Embedded Systems

Paper I presents a detailed overview on the ABSOLUT performance evaluation approach. The modelling approach is presented starting from the Y chart and continuing to the workload modelling of applications with UML and SystemC and capacity modelling of platforms with SystemC. UML to SystemC workload model transformation utilising Telelogic Tau and the SystemC generator developed by the Lund University is also presented, which results to a pure SystemC-based system model. The layering of the workload and platform models is shown in addition to the primitive and service interfaces between the workload and platform models. Of the load extraction techniques presented in this thesis, the analytical and measurement-based techniques are included in Paper I. It also presents the initial developments towards the automatic source

Table 10. Case studies presented in the publications.

No	Publication	Case studies					
		M[8]	J[9]	P[10]	Q[11]	M[12]	V[13]
I	Kreku *et al.* (2008b)	x					
II	Kreku *et al.* (2009)	x					
III	Kreku & Tiensyrjä (2011)		x				
IV	Kreku *et al.* (2004b)			x			
V	Kreku *et al.* (2004a)				x		
VI	Kreku *et al.* (2010)		x				
VII	Kreku *et al.* (2005)					x	
VIII	Kreku *et al.* (2007)	x					
IX	Kreku *et al.* (2008a)	x					x

[8] Mobile video player
[9] Parallel JPEG encoding
[10] Mobile phone usage
[11] Quake 2 gaming
[12] MPEG4 encoding
[13] Virtual network computing

code-based workload model generation. Finally, the Mobile Video Player case study is used as an example.

Jari Kreku designed the overall ABSOLUT workload and platform modelling approach, developed the platform models used in the paper, and performed the MVP case study simulations. The layering of the models was created in collaboration with Kari Tiensyrjä. The measurement-based load extraction technique was developed in collaboration with Jani Penttilä and Tarja Kauppi.

6.2 Application Workload and SystemC Platform Modeling for Performance Evaluation

Paper II extends Paper I with a more detailed description of how the workload and platform models are created through a thorough presentation of the MVP case study. The modelling of services is emphasised, and the utilisation of components in a model library for the platform model is shown. Finally, more detailed results from the MVP case study are displayed. The main contributions by Jari Kreku to this paper are the

design of the ABSOLUT approach, implementation of the platform models for the case study, performing the simulation, and analysing the results.

6.3 Scalable Multi-core Architectures: Design Methodologies and Tools, chapter System Exploration

Paper III presents ABSOLUT as the system-level performance and power evaluation tool of the Mapping Optimisation of Scalable Multi-core Architectures (MOSART) design flow. The application modelling is presented in detail, including layers, base classes for application, process and function workload models of the ALE model library, and methods of the base classes. In the MOSART flow, input to the workload modelling is the parallel C/C++ source code from the IMEC MPA parallelisation tool. Thus the ABSINTH workload model generator and the interface between MPA and ABSOLUT are also presented.

The main extension presented by the Paper III to ABSOLUT is the power and energy consumption modelling and simulation at the system level. It is based on component-level power consumption input and extension of the performance probes to support recording of power consumption states of components. Mapping of workload models to platform models, interfaces provided by the operating system model for process control and service registration and model parameters are also shown in such a detail not found in the other publications. The link between MPA and ABSOLUT and the performance and power simulation is demostrated with the JPEG case study in this paper.

Jari Kreku developed the ABSOLUT approach, the ABSINTH tool, the link between the MPA and ABSOLUT tools and the JPEG case study simulation models. The power and energy consumption modelling and simulation was developed in collaboration with Kari Tiensyrjä.

6.4 Workload simulation method for evaluation of application feasibility in a mobile multiprocessor platform

Paper IV is the first publication of the ABSOLUT approach. It describes the contents of workload and platform models at a conceptual level. A systematical method for creating workload models from existing measurement data of several independent applications and combining them for the simulation of a complex use case is presented.

The ABSOLUT approach is applied to three case examples: The first one consists of MP3 playback as performed by the DSP coprocessor of the OMAP1510 application platform engine. An MP3 decoder is modelled manually from the source code, and a capacity model of the OMAP1510 is created. The combined system model is simulated, and then the results are compared to measurements made from the same application running in an OMAP prototype. The second and third case examples utilise several typical mobile handset applications sequentially and concurrently, including text and multimedia messaging, MP3 playback, and bluetooth download, among others. These are modelled using the presented measurement-based approach from individual applications. Finally, the results obtained from the simulations are compared to the measurements made of the entire use cases.

Jari Kreku designed the overall ABSOLUT workload and platform modelling approach, developed the source code-based load extraction technique, implemented the OMAP1510 and MP3 player models, and performed the simulations. The measurement-based load extraction technique was developed in collaboration with Jani Penttilä and Tarja Kauppi.

6.5 Evaluation of Platform Architecture Performance using Abstract Instruction-level Workload Models

Paper V extends Paper IV by providing more detailed descriptions of both workload and platform models. It elaborates how time is simulated in the different component types of the platform side, i.e. processing unit, interconnection, and memory or peripheral models. An initial hierarchy for workloads is also presented. In this paper they consist of

1. workloads for each individual processing unit
2. workload subsets

3. workload functions
4. load primitives.

The hierarchy and naming of the upper layers is refined in later papers. Furthermore, a manual method for modelling of applications from the source code is presented. This method is applied for the modelling of the Quake 2 3D game for a modified OMAP2410 platform.

Jari Kreku designed the overall ABSOLUT workload and platform modelling approach, developed the source code-based load extraction technique, implemented the OMAP2410 model, and performed the simulations. The original layering of the workload models was created in collaboration with Tarja Kauppi.

6.6 Automatic workload generation for system-level exploration based on modified GCC compiler

Paper VI contributes the ABSINTH automatic source code-based workload modelling to the ABSOLUT approach. It is based on two compilation passes inserted in the open source GCC compiler, one for generating the control flow of workload models and another for extracting the load primitives. The version presented in the paper generates only statistical control for the models with the help of profiling data from application execution. ABSINTH generates SystemC code, which can be used as workload models in ABSOLUT simulations after a postprocessing step. The first version of the JPEG encoder case study consisting only of the performance simulation is presented in this paper. The simulation results provided by ABSOLUT are compared to those obtained from MPA high-level simulation. ABSINTH is further applied to an x264 video encoder case study. The workload model generation technique used in ABSINTH and the implementation of the tools were done by Jari Kreku.

6.7 Exploitation of UML 2.0-Based Platform Service Model and SystemC Workload Simulation in MPEG-4 Partitioning

This publication contains the first experiments with modelling applications with UML. As a prerequisite the workload platform interfaces of the ABSOLUT approach were described in UML. In the modelling approach presented in this paper, the structure of

the workload, including functions, control flow, and load primitives, is expressed in the UML-based model. The load primitives are implemented as calls to the aforementioned interfaces. Then, a UML to C++ generator is used to transform it to SystemC. Unfortunately this requires that essential parts of the SystemC like the sc_module class are also expressed in the UML model for the C++ generator to work correctly. The case example in the paper consists of evaluating the partitioning of an MPEG4 encoder to OMAP5912 platform with ABSOLUT. Three alternatives were examined: In the first one, everything is executed on the OMAP5912's ARM CPU. In the second one, the discrete cosine transform (DCT) is accelerated by the DSP. Finally, in the last one, both the DCT and the sum of absolute differences (SAD) algorithms are accelerated by the DSP.

Jari Kreku developed the overall ABSOLUT approach, the technique for UML-based workload modelling used in the paper, and the OMAP platform model. Validation of the results obtained from simulations was performed by Matti Eteläperä for his diploma thesis. This effort consisted of porting the open source FFMPEG encoder to OMAP5912 Starter Kit and executing it while monitoring its performance. Simulations, comparisons to measurements and analysis were performed by Jari Kreku.

6.8 SystemC Workload Model Generation from UML for Performance Simulation

Paper VIII presents an improved UML front-end for workload modelling compared to Paper VII. Existing, unmodified UML class and statechart diagrams are automatically transformed to SystemC by utilising Telelogic Tau and a SystemC generator developed by the Lund University. A flow to get a combined SystemC-based system model from UML-based workload models and a SystemC-based platform model is presented. The mobile video player case study is used as an example to demonstrate the concept. The main contributions by Jari Kreku to the research in this paper include the overall ABSOLUT approach and the linking of the SystemC code generated from UML with the platform models, the latter of which was done in collaboration with Mika Hoppari at VTT and Per Andersson at the Lund University. Jari Kreku implemented the platform models, performed the simulations and analysed the results.

6.9 Application–Platform Performance Modeling and Evaluation

Paper IX extends Paper VIII with the description of the low-level workload primitive interface and high-level service interface between application and platform models. Furthermore, the UML-based workload modelling front-end is experimented with the VNC case study, where a mobile handset is used for internet browsing through a VNC server. Jari Kreku contributed the overall ABSOLUT approach and supported Tuomo Kestilä with the modelling and simulation of the VNC case for his diploma thesis.

7 Conclusions

Performance evaluation is used in the design of embedded systems to compare alternative platforms and find the best one, to search for the best application out of several for a given platform, and to design a new system, platform, or application. Evaluation aims at obtaining the highest possible performance of the system under development with the least amount of cost. However, studies have shown that application complexity has a superlinear, quadratic growth, and also the number of components in execution platforms and the number of transistors in each component is increasing. As a consequence, enormous system complexity can be realised on a single die, but novel methods and tools are needed to alleviate the resulting system evaluation complexity problem.

Performance evaluation approaches are divided into analytical, simulation and measurement categories. Analytical approaches require simplifications and assumptions resulting to low accuracy. On the other hand, measurements are not suitable for early evaluation since they require that the system is available and mature enough for the execution of the applications. Simulation approaches can be further divided into virtual system, virtual platform, and virtual prototype approaches. The virtual platform and prototype approaches simulate executable application binaries using instruction set simulators in a platform model with a high and low abstraction level, respectively. The virtual system approaches combine abstract application and platform models making them particularly well suited for early evaluation.

The ABSOLUT performance evaluation approach presented in this thesis aims at low modelling effort through proper abstractions to solve the complexity problem while maintaining enough accuracy for reliable evaluation results. Applications are modelled as layered workload models ultimately consisting of abstract, instruction-like workload primitives. Several techniques have been developed to create the workload models from information sources such as application specifications, execution traces, or source code. As application functionality is abstracted, the complexity of execution platform models is also reduced, especially with respect to the processing elements. A platform model can be rapidly constructed from components in a model library with the help of a platform generation tool. The virtual system model constructed by allocating workload models on top of the platform components is simulated using the SystemC simulation kernel and models based on the TLM standard. Performance data, for example processor

utilisation, is extracted from simulations by instrumenting the workload and or platform models with custom performance probes.

ABSOLUT has been applied to nine case studies in this thesis. Applications have been modelled from different fields including audio video playback and recording, communication and gaming. All the workload modelling techniques presented in the thesis have been utilised in the case studies. The modelled execution platforms have included contemporary embedded system and workstation platforms, future embedded platforms under development, and even custom, complex future platforms have been invented and then explored with ABSOLUT. Whenever possible, the simulation results have been validated with measurements from real applications in a real platform. The experiences and results from the case studies prove that ABSOLUT is suitable for early performance evaluation and that low modelling effort, reasonable simulation speed and accurate results have been achieved.

The workload modelling techniques presented in the thesis are not able to extract accurate address information for the models in their current form. Thus, certain parts of the memory hierarchy, for example caches, need to be modelled statistically in the execution platform models. Finding a solution to the address problem is seen as one challenging research problem for the future work. ABSOLUT could be further expanded with support for multiple abstraction levels both in application and in platform sides. Higher-level models would provide faster evaluation for design space exploration, whereas the more detailed lower-level models would provide more accuracy and reliability for the later phases of the design flow.

References

Aguirre S & Candaele B (2011) Scalable Multi-core Architectures: Design Methodologies and Tools, chapter MPSoC Performance Analysis for Cognitive Radio Applications. Springer.

Apvrille L, Muhammad W, Ameur-Boulifa R, Coudert S & Pacalet R (2006) A uml-based environment for system design space exploration. Proc. Electronics, Circuits and Systems, 2006. ICECS'06. 13th IEEE International Conference on, IEEE, 1272–1275.

Baghdadi A, Zergainoh N, Cesario W & Jerraya AA (2002) Combining a performance estimation methodology with a hardware/software codesign flow supporting multiprocessor systems. IEEE Transactions on software engineering 28(9): 822–831.

Baghdadi A, Zergainoh N, Cesario W, Roudier T & Jerraya AA (2000) Design space exploration for hardware/software codesign of multiprocessor systems. Proc. 11th International Workshop on Rapid System Prototyping (RSP), 8–13.

Balarin F (1997) Hardware-software co-design of embedded systems: the POLIS approach. Springer Netherlands.

Balarin F, Watanabe Y, Hsieh H, Lavagno L & Passerone C (2003) Metrpolis: an integrated electronic system design environment. IEEE Computer 36(4): 45–52.

Beltrame G, Bolchini C, Fossati L, Miele A & Sciuto D (2008) Resp: A non-intrusive transaction-level reflective mpsoc simulation platform for design space exploration. Proc. Proceedings of the 2008 Asia and South Pacific Design Automation Conference.

Beltrame G, Sciuto D & Silvano C (2007) Multi-accuracy power and performance transaction-level modeling. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26(10): 1830–1842.

Beltrame G, Sciuto D, Silvano C, Paulin P & Bensoudane E (2006) An application mapping methodology and case study for multi-processor on-chip architectures. Proc. 2006 IFIP International Conference on Very Large Scale Integration, Citeseer, 16–18.

Bobrek A, Pieper J, Nelson J, Paul J & Thomas D (2004) Modeling shared resource contention using a hybrid simulation/analytical approach. Proc. Proceedings of the Design, Automation and Test in Europe, 1144–1149.

Buck J, Ha S, Lee EA & Messerschmitt DG (1992) Ptolemy: A framework for simulating and prototyping heterogeneous systems. Technical report, University of California, Department of Electrical Engineering and Computer Science.

Cadence Virtual System Platform (2011) Available at http://www.cadence.com/products/sd/virtual_system/pages/default.aspx.

Calvez J, Heller D & Pasquier O (1996) Uninterpreted co-simulation for performance evaluation of hw/sw systems. Proc. 4th International Workshop on Hardware/Software Co-Design, Pittsburgh, Pennsylvania, USA, 132–139.

Calvez J & Pasquier O (1998) Performance monitoring and assessment of embedded hw/sw systems. Design Automation for Embedded Systems 3(1): 5–22.

Calvez J, Pasquier O, Isidoro D & Jeuland D (1994) Codesign with the mcse methodology. Proc. Proceedings of the 20th EUROMICRO Conference, 19–26.

Chaoui J, Cyr K, de Gregorio S, Giacalone JP, Webb J & Masse Y (2001) Open multimedia application platform: Enabling multimedia applications in third generation wireless terminals through a combined risc/dsp architeture. Proc. Proceedings of the Acoustics, Speech and

Signal Processing (ICASSP) conference, 2: 1009–1012.
Chopard B, Combes P & Zory J (2006) A conservative approach to systemc parallelization. Computational Science–ICCS 2006 653–660.
CMake (2011) Cross platform make website. Available at http://www.cmake.org.
CoFluent (2011) Cofluent design website. Available at http://www.cofluentdesign.com.
Cox D (2005) Ritsim: Distributed systemc simulation. Master's thesis, Rochester Institute of Technology .
de Kock EA, Essink G & Smits WJM (2000) Yapi: application modeling for signal processing systems. Proc. Proceedings of the 37th Design Automation Conference (DAC'00), Los Angeles, California, USA, 402–405.
Debian Counting (2011) Available at http://libresoft.dat.escet.urjc.es/debian-counting/.
Deshpande A & Riehle D (2008) The total growth of open source. Proc. Proceedings of the Fourth Conference on Open Source Systems, Springer-Verlag, 197–209.
DistroWatch.com (2011) Debian gnu/linux. Available at http://distrowatch.com/table.php?distribution=debian.
Donlin A (2004) Transaction level modeling: flows and use models. Proc. Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, ACM, 75–80.
Eker J, Janneck J, Lee E, Liu J, Liu X, Ludvig J, Neuendorffer S, Sachs S & Xiong Y (2003) Taming heterogeneity-the ptolemy approach. Proceedings of the IEEE 91(1): 127–144.
European EDA Roadmap (2009) CATRENE. 352 p.
Ezudheen P, Chandran P, Chandra J, Simon B & Ravi D (2009) Parallelizing systemc kernel for fast hardware simulation on smp machines. Proc. Principles of Advanced and Distributed Simulation, 2009. PADS'09. ACM/IEEE/SCS 23rd Workshop on, IEEE, 80–87.
Fornaciari W, Sciuto D, Silvano C & Zaccaria V (2001) A design framework to efficiently explore energy-delay tradeoffs. Proc. Ninth International Symposium on Hardware/Software Codesign (CODES), 260–265.
Fornaciari W, Sciuto D, Silvano C & Zaccaria V (2002) A sensitivity-based design space exploration methodology for embedded systems. Design Automation for Embedded Systems 7(1–2).
GCC (2011) The gnu compiler collection website. Available at http://gcc.gnu.org.
Grötker T (2002) System Design with SystemC. Kluwer Academic Publishers. 240 p.
Heidelberg P & Lavenberg S (1984) Computer performance evaluation methodology. IEEE Transactions on Computers .
Hylands C, Lee E, Liu J, Liu X, Neuendorffer S, Xiong Y, Zhao Y & Zheng H (2003) Overview of the ptolemy project, techreport ucb/erl m03/25, department of electrical engineering and computer science. University of California, Berkeley .
Ieromnimon F, Kritharidis D & Voros NS (2011) Scalable Multi-core Architectures: Design Methodologies and Tools, chapter Application of the MOSART Flow on the WiMAX (802.16e) PHY Layer. Springer.
Intel Nehalem Microarchitecture (2009) First the tick, now the tock: Intel microarchitecture (nehalem) white paper. Available at http://www.intel.com/technology/architecture-silicon/next-gen/319724.pdf.
ITRS (2009) International technology roadmap for semiconductors. Available at http://www.itrs.net/Links/2010ITRS/Home2010.htm.

Jaber C, Kanstein A, Apvrille L, Baghdadi A, Moenner PL & Pacalet R (2009) High-level system modeling for rapid hw/sw architecture exploration. Proc. IEEE/IFIP International Symposium on Rapid System Prototyping (RSP '09), Paris, France, 88–94.

Jain R (1991) The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling. John Wiley & Sons, Inc. 685 p.

Kangas T, Kukkala P & Orsila H (2006) Uml-based multiprocessor soc design framework. ACM Transactions on Embedded Computing Systems 5(2): 281–320.

Kangas T, Riihimaki J, Salminen E, Kuusilinna K & Hamalainen T (2003) Using a communication generator in soc architecture exploration. Proc. System-on-Chip, 2003. Proceedings. International Symposium on, IEEE, 105–108.

Kienhuis B, Deprettere E, Vissers K & Van Der Wolf P (1997) An approach for quantitative analysis of application-specific dataflow architectures. Proc. Application-Specific Systems, Architectures and Processors, 1997. Proceedings., IEEE International Conference on, IEEE, 338–349.

Koch S (2007) Software evolution in open source projects — a large-scale investigation. Journal of Software Maintenance and Evolution: Research and Practice 361–382.

Kogel T, Haverinen A & Aldis J (2005) Ocp tlm for architectural modeling. Technical report, OCP-IP.

Kreku J, Eteläperä M & Soininen JP (2005) Exploitation of UML 2.0-based platform service model and SystemC workload simulation in MPEG-4 partitioning. Proc. International Symposium on System-on-Chip Proceedings, 167–170.

Kreku J, Hoppari M, Kestila T, Qu Y, Soininen J & Tiensyrja K (2008a) Application-platform performance modeling and evaluation. Proc. Specification, Verification and Design Languages, 2008. FDL 2008. Forum on, IEEE, 43–48.

Kreku J, Hoppari M, Kestilä T, Qu Y, Soininen JP, Andersson P & Tiensyrjä K (2008b) Combining uml2 application and systemc platform modelling for performance evaluation of real-time embedded systems. EURASIP Journal on Embedded Systems .

Kreku J, Hoppari M, Kestilä T, Qu Y, Soininen JP & Tiensyrjä K (2009) Languages for Embedded Systems and their Applications, volume 36 of *Lecture Notes in Electrical Engineering*, chapter Application Workload and SystemC Platform Modeling for Performance Evaluation, 131–148. Springer.

Kreku J, Hoppari M, Tiensyrjä K & Andersson P (2007) Systemc workload model generation from uml for performance simulation. Proc. Forum on Specification and Design Languages.

Kreku J, Kauppi T & Soininen JP (2004a) Evaluation of platform architecture performance using abstract instruction-level workload models. Proc. International Symposium on System-on-Chip Proceedings, 43–48.

Kreku J, Penttilä J, Kangas J & Soininen JP (2004b) Workload simulation method for evaluation of application feasibility in a mobile multiprocessor platform. Proc. Proceedings of the Euromicro Symposium on Digital System Design, 532–539.

Kreku J, Qu Y, Soininen JP & Tiensyrjä K (2006) Layered uml workload and systemc platform models for performance simulation. Proc. International Forum on Specification and Design Languages (FDL), 223–228.

Kreku J & Soininen J (2003) Mappability estimate: a measure of the goodness of a processor-algorithm pair. Proc. System-on-Chip, 2003. Proceedings. International Symposium on, IEEE, 119–122.

Kreku J & Tiensyrjä K (2011) Scalable Multi-core Architectures: Design Methodologies and Tools, chapter System exploration. Springer.

Kreku J, Tiensyrjä K & Vanmeerbeeck G (2010) Automatic workload generation for system-level exploration based on modified gcc compiler. Proc. Design, Automation and Test in Europe conference and exhibition.

Lahiri K, Dey S & Ragunathan A (2001a) Evaluation of the traffic-performance characteristics of system-on-chip communication architectures. Proc. Proceedings of 14th International Conference on VLSI Design, 29–35.

Lahiri K, Ragunathan A & Dey S (2000a) Efficient exploration of the soc communication architecture design space. Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD), 424–430.

Lahiri K, Ragunathan A & Dey S (2000b) Performance analysis of systems with multi-channel communication architectures. Proc. Proceedings of the 13th International Conference on VLSI Design, Calcutta, India, 530–537.

Lahiri K, Ragunathan A & Dey S (2001b) System-level performance analysis for designing on-chip communication architectures. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 20(6): 768–783.

Lahiri K, Ragunathan A & Dey S (2004) Design space exploration for optimizing on-chip communication architectures. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 23(6): 952–961.

Lieverse P, van der Wolf P & Deprettere E (2001a) A trace transformation technique for communication refinement. Proc. 9th International Symposium on Hardware/Software Codesign (CODES), 134–139.

Lieverse P, van der Wolf P, Vissers K & Deprettere E (2001b) A methodology for architecture exploration of heteregeneous signal processing systems. Kluwer Journal of VLSI Signal Processing 29(3): 197–207.

Lu Z, Chen X, Zhang Y, Naeem A, Jantsch A, Anagnostopoulos I, Xydis S, Bartzas A, Soudris D & Baloukas C (2010) Final version of nostrum extensions for distributed memory and abstract data types. Technical report, Mapping Optimisation for Scalable multi-core ARchiTecture (MOSART). Http://www.mosart-project.org.

Mahadevan S, Angiolini F, Storgaard M, Olsen R, Sparso J & Madsen J (2005a) A network traffic geenrator model for fast network-on-chip simulation. Proc. Proceedings of the Design, Automation and Test in Europe, 780–785.

Mahadevan S, Storgaard M, Madsen J & Virk K (2005b) Arts: a system-level framework for modeling mpsoc components and analysis of their causality. Proc. Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 480–483.

Mahadevan S, Virk K & Madsen J (2007) Design Automation for Embedded Systems, chapter ARTS: A SystemC-based framework for multiprocessor Systems-on-Chip modelling. Springer.

Maillet-Contoz L & Ghenassia F (2005) Transaction level modeling. Transaction Level Modeling with SystemC 23–55.

Marcon C, Kreutz M, Susin A & Calazans N (2005) Models for embedded application mapping onto nocs: timing analysis. Proc. Rapid System Prototyping, 2005.(RSP 2005). The 16th IEEE International Workshop on, IEEE, 17–23.

Mignolet JY, Baert R, Ashby T, Avasare P, Jang HO & Son JC (2009) Mpa: Parallelizing an application onto a multicore platform made easy. IEEE Micro 29(3): 31–39.

Millberg M, Nilsson E, Thid R, Kumar S & Jantsch A (2004) The nostrum backbone - a communication protocol stack for networks on chip. Proc. 17th International Conference on VLSI Design, Mumbai, India. DOI: http://doi.ieeecomputersociety.org/10.1109/ICVD.2004.1261005.

Mohanty S & Prasanna V (2002) Rapid system-level performance evaluation and optimization for application mapping onto soc architectures. Proc. Proceedings of the IEEE International ASIC/SOC Conference, 160–167.

Mohanty S, Prasanna V, Neema S & Davis J (2002) Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. ACM SIGPLAN Notices 37(7): 18–27.

Montoreano M (2007) Transaction level modeling using osci tlm 2.0. Open SystemC Initiative (OSCI) .

MPG123 (2011) Fast console mpeg audio player and decoder library. Available at http://www.mpg123.de.

Network on Terminal Architecture (2011) Nota world open architecture initiative.

OMAP2 (2005) Omap2 architecture: Omap2420 processor product bulletin. Available at http://focus.ti.com/pdfs/wtbu/TI_omap2420.pdf.

Open SystemC Initiative website (2011) Available at http://www.systemc.org/.

Paul J, Bobrek A, Nelson J, Pieper J & Thomas D (2003) Schedulers as model-based design elements in programmable heterogeneous multiprocessors .

Paul JM, Thomas DE & Cassidy AS (2005) High-level modeling and simulation of single-chip programmable heterogeneous multiprocessors. ACM Transactions on Design Automation of Electronic Systems 10(3): 431–461.

Paulin P, Pilkington C & Bensoudane E (2002) Stepnp: A system-level exploration platform for network processors. Design & Test of Computers, IEEE 19(6): 17–26.

Pimentel A (2008) The artemis workbench for system-level performance evaluation of embedded systems. International Journal of Embedded Systems 3(3).

Pimentel A, Erbas C & Polstra S (2006) A systematic approach to exploring embedded system architectures at multiple abstraction levels. IEEE Transactions on Computers 55(2): 99–112.

Pimentel A, Hertzberger L, Lieverse P, van der Wolf P & Deprettere E (2001) Exploring embedded systems architectures with artemis. IEEE Computer 34(11): 57–63.

Pimentel A, Polstra S, Terpstra F, van Halderen A, Coffland J & Hertzberger L (2002) Embedded processor design challenges., volume 2268 of *Systems, architectures, modeling, and simulation — SAMOS*, chapter Towards efficient design space exploration of heterogeneous embedded media systems, 57–73. Springer-Verlag.

Pimentel A, Thompson M, Polstra S & Erbas C (2008) Calibration of abstract performance models for system-level design space exploration. Journal of Signal Processing Systems (50): 99–114. DOI: 10.0007/s11265-007-0085-2.

Posadas H, Adamez J, Villar E, Blasco F & Escuder F (2005) Rtos modeling in systemc for real-time embedded sw simulation: A posix model. Design Automation for Embedded Systems 10: 209–227.

Posadas H, Herrera F, Sanchez P, Villar E & Blasco F (2004) System-level performance analysis in systemc. Proc. Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE âĂŹ04), Paris, France, 378–383.

Posadas H, Real S & Villar E (2011) M3-scope: Performance modeling of multi-processor embedded systems for fast design space exploration. In: Silvano C, Fornaciari W & Villar E (eds) Multi-objective Design Space Exploration of Multiprocessor SoC Architectures, 19–50.

Springer New York.
Ptolemy II (2011) Ptolemy ii website. Available at http://ptolemy.berkeley.edu/ptolemyII.
Quake 2 (2011) id software quake 2 website. Available at http://www.idsoftware.com/gate/?uri=%2Fgames%2Fquake%2Fquake2%2F.
Raghavan G, Salomäki A & Lencevicius R (2004) Model based estimation and verification of mobile device performance. Proc. Proceedings of the 4th ACM international conference on Embedded software, ACM, 34–43.
Saastamoinen J, Khan S, Tiensyrjä K & Taipale T (2011) Multi-threading support for system-level performance simulation of multi-core architectures. Proc. Proceedings of the 24th International Conference on Architecture of Computing Systems, Como, Italy, 169–177.
Saastamoinen J & Kreku J (2011) Workload model generation for system-level design exploration. Proc. Tbd.
Sangiovanni-Vincentelli A (2007) Quo vadis sld: reasoning about trends and challenges of system-level design. Proceedings of the IEEE 95(3): 467–506.
Sangiovanni-Vincentelli A & Di Natale M (2007) Embedded system design for automotive applications. Computer 40(10): 42–51.
Schnerr J, Bringmann O, Viehl A & Rosenstiel W (2008) High-performance timing simulation of embedded software. Proc. Proceedings of the 45th annual Design Automation Conference, New York, NY, USA.
Sciuto D, Salice F & Fornaciari W (2002) Metrics for design space exploration of heterogeneous multiprocessor embedded systems. Proc. Proceedings of the tenth international symposium on Hardware/software codesign.
Stewart D (2001) Measuring execution time and real-time performance. Proc. Embedded Systems Conference (ESC), Citeseer.
Stewart D & Arora G (2003) A tool for analyzing and fine tuning the real-time properties of an embedded system. IEEE Transactions on Software Engineering 311–326.
Suoranta R (2006) New directions in mobile device architectures. Proc. 9th EUROMICRO conference on Digital System Design: Architectures, Methods and Tools, 17–26.
Suresh D, Najjar W, Vahid F, Villarreal J & Stitt G (2003) Profiling tools for hardware/software partitioning of embedded applications. Proc. Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems, ACM, 189–198.
Synopsys Platform Architect (2011) Available at http://www.synopsys.com/Systems/ArchitectureDesign/pages/PlatformArchitect.aspx.
Synopsys Processor Designer (2011) Available at http://www.synopsys.com/Systems/BlockDesign/ProcessorDev/Pages/default.aspx.
Terpstra F, Polstra S, Pimentel A & Hertzberger B (2001) Rapid evaluation of instantiations of embedded systems architectures: A case study. Proc. In Proc. of the Progress workshop on Embedded Systems, Citeseer.
Thompson M & Pimentel A (2007) Systems, architectures, modeling, and simulation — SAMOS, volume 4599, chapter Towards Multiapplication Workload Modeling in Sesame for System-Level Design Space Exploration, 222–232. Springer-Verlag.
TLM2 Whitepaper (2007) Available at http://www.systemc.org.
Turski W (1996) Reference model for smooth growth of software systems. IEEE Transactions on Software Engineering 22(8): 599–600.
Van Stralen P & Pimentel A (2010) Scenario-based design space exploration of mpsocs. Proc. Proc. of the IEEE International Conference on Computer Design (ICCDâĂŹ10), Citeseer.

Vanthournout B, Goossens S & Kogel T (2004) Developing transaction-level models in systemc. Technical report, Coware.

Waseem M, Apvrille L, Ameur-Boulifa R, Coudert S & Pacalet R (2006) Abstract application modeling for system design space exploration. Proc. Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on, IEEE, 331–337.

Weber A (2007) The convergence of mobile data phones, consumer electronics, and wallets: Lessons from japan. Telematics and informatics 24(3): 180–191.

Wieferink A (2004) System level processor/communication co-exploration methodology for multi-processor system-on-chip platforms. Proc. Proceedings of Design Automation and Test in Europe, Paris, France, 1256–1261.

Wieferink A, Dörper M, Leupers R, Ascheid G, Meyr H, Kogel T, Braun G & Nohl A (2005) System level processor/communication co-exploration methodology for multiprocessor system-on-chip platforms. IEE Proceedings of Computers and Digital Techniques 152(1): 3–11.

Wild T, Herkersdorf A & Lee GY (2006) Tapes — trace-based architecture performance evaluation with systemc. Design Automation for Embedded Systems 10(2–3): 157–179. Special Issue on SystemC-based System Modeling, Verification and Synthesis.

Zivkovic V, de Kock EA, van der Wolf P & Deprettere E (2003a) Fast and accurate multiprocessor architecture exploration with symbolic programs. Proc. Proceedings of the conference on Design, Automation and Test in Europe.

Zivkovic V, Deprettere E, van der Wolf P & de Kock EA (2002) Design space exploration of streaming multiprocessor architectures. Proc. IEEE Workshop on Signal Processing Systems (SIPS), 228–234.

Zivkovic V, Deprettere V, van der Wolf P & De Kock E (2003b) From high level application specification to system-level architecture definition: Exploration, design and compilation .

Original articles

I Kreku J, Hoppari M, Kestilä T, Qu Y, Soininen J-P, Andersson P & Tiensyrjä K (2008) Combining UML2 Application and SystemC Platform Modelling for Performance Evaluation of Real-Time Embedded Systems. EURASIP Journal on Embedded Systems. DOI: 10.1155/2008/712329.

II Kreku J, Hoppari M, Kestilä T, Qu Y, Soininen J-P & Tiensyrjä K (2009) Application Workload and SystemC Platform Modeling for Performance Evaluation. Best of FDL 2009 book.

III Kreku J & Tiensyrjä K (2011) Scalable Multi-core Architectures: Design Methodologies and Tools, chapter System exploration. Springer.

IV Kreku J, Penttilä J, Kangas J & Soininen J-P (2004) Workload simulation method for evaluation of application feasibility in a mobile multiprocessor platform. Proceedings of the Euromicro Symposium on Digital System Design. Rennes, 31 Aug.–3 Sept. 2004 . IEEE Computer Society: 532–539. DOI: 10.1109/DSD.2004.1333322.

V Kreku J, Kauppi T & Soininen J-P (2004) Evaluation of platform architecture performance using abstract instruction-level workload models. International Symposium on System-on-Chip. Tampere, 16–18 Nov. 2004. Tampere University of Technology: 43–48.

VI Kreku J, Tiensyrjä K & Vanmeerbeeck G (2010) Automatic workload generation for system-level exploration based on modified GCC compiler. Proc. Design, Automation and Test in Europe conference and exhibition.

VII Kreku, Eteläperä M & Soininen J-P (2005) Exploitation of UML 2.0-based platform service model and systemC workload simulation in MPEG-4 partitioning. International Symposium on System-on-Chip. Tampere, 15–17 Nov. 2005. Tampere University of Technology: 167–170.

VIII Kreku J, Hoppari M, Tiensyrjä K & Andersson P (2007) SystemC workload model generation from UML for performance simulation. Forum on Specification and Design Languages. FDL, Barcelona, Spain, 15–18 Sep. 2007. ECSI, Grenoble.

IX Kreku J, Hoppari M, Kestilä T, Qu Y, Soininen J-P & Tiensyrjä K (2008) Application – Platform Performance Modeling and Evaluation. Forum on Specification, Verification and Design Languages, 2008. FDL 2008. 23–25 Sept. 2008: 43–48.

Book orders:
Granum: Virtual book store
http://granum.uta.fi/granum/

UNIVERSITY OF OULU P.O.B. 7500 FI-90014 UNIVERSITY OF OULU FINLAND

ACTA UNIVERSITATIS OULUENSIS

ISBN 978-951-42-9989-6 (Paperback)
ISBN 978-951-42-9990-2 (PDF)
ISSN 0355-3213 (Print)
ISSN 1796-2226 (Online)

UNIVERSITY of OULU
OULUN YLIOPISTO