GLOBAL ADDRESS SPACES FOR EFFICIENT RESOURCE PROVISIONING IN THE DATA CENTER

A Thesis
Presented to
The Academic Faculty

by

Jeffrey Young

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
December 2013

GLOBAL ADDRESS SPACES FOR EFFICIENT RESOURCE PROVISIONING IN THE DATA CENTER

ACKNOWLEDGEMENTS

This thesis would not have been possible without the assistance, guidance, and inspiration of many individuals. My adviser, Sudhakar Yalamanchili, provided the spark for moving from FPGA-based research to networking-related topics, and his influence can be seen in the many papers and complementary research ideas that make up this thesis. I am grateful to him for sharing his expertise and for providing me with the opportunity to become the self-sufficient researcher that I am today.

I also am appreciative of the efforts of my committee, many of whom I have had the great honor of working with over the past few years on research projects and several of whom have influenced my research thought process and skills through their classes. I would like to acknowledge Dr. George Riley and Dr. David Schimmel for their selfless service as part of my Master's and PhD committees and also Dr. Karsten Schwan and Dr. Ada Gavrilovska for their consistent feedback through our joint Heterogeneous Virtual Machine (HVM) work, which helped to mold the design of the Oncilla runtime and the last portion of my thesis. In addition to these Georgia Tech professors, I would also like to thank Dr. Ron Sass, who I worked with at Clemson University and at the University of Kansas. His optimism and enthusiasm for research showed me the positive impact that advisors can have on their graduate students, and my initial work with him was a powerful motivator towards pursuing a degree and career in research.

Also, many thanks go to our research colleagues at the University of Heidelberg and the University of Valencia. The initial idea for the DPGAS model was fleshed out with assistance from Dr. Federico Silla and Dr. José Duato of the University of Valencia. Dr. Holger Fröning and Dr. Mondrian Nüssle of the University of Heidelberg provided extraordinary hardware and software support related to the use of their EXTOLL networking technology, and I am still in their debt for many useful research ideas and timely software patches. I am also indebted to the many students at the University of Heidelberg including but not limited to Alexander Giese, Lena Oden and Benjamin Klenk.

Work on the HyperTransport over Ethernet and HyperTransport over InfiniBand specifications was highly dependent on the feedback of the members of the HyperTransport Consortium. I am thankful for the opportunity to contribute to these and other specifications, and I appreciate the efforts and insights provided by Mario Cavalli, Brian Holden, Emilio Billi, and Paul Miranda.

I would also like to thank my many friends and colleagues at Georgia Tech who have provided needed motivation, programming knowledge, and in some cases substantial contributions to my thesis research. From our lab, CASL, I would especially like to thank Greg Diamos, Andrew Kerr, Mani Ramaswamy, Nawaf Almoosa, Chad Kersey, Se Hoon Shon, and Mitchelle Rasquinha for their insight and for contributing to a positive environment in the lab. Also, I am deeply grateful to Alex Merritt for his persistence in working with the Oncilla runtime and his assistance with wrangling servers and networking hardware.

Finally, I must thank my family and especially my wife, Jessica, for their patience and support as I have undertaken this long path to finishing my dissertation. To my parents and my brother Chris and his wife, Sarah, thank you for never forgetting about my endeavour and for your continued interest in my sometimes long-winded research explanations. And to Jessica - thank you for your extraordinary support and love as I have worked long nights and weekends to achieve this goal. I couldn’t have done this without you, and I look forward to life “post-PhD” as we explore what lies ahead for the both of us.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The rise of large data sets, or “Big Data”, has coincided with the rise of clusters with large amounts of memory and Graphics Processing Unit, or GPU, accelerators that can be used to process rapidly growing data footprints. However, the complexity and performance limitations of sharing memory and accelerators in a cluster limits the options for efficient management and allocation of resources for applications. The global address space model (GAS), and specifically hardware-supported GAS, is proposed as a means to provide a high-performance resource management platform upon which resource sharing between nodes and resource aggregation across nodes can take place. This thesis builds on the initial concept of GAS with a model that is matched to “Big Data” computing and data transfer requirements. Contributions from this thesis include:

- A new global address space model, Dynamic Partitioned Global Address Spaces (DPGAS), that utilizes common networking hardware
- The proposal of architectural support for DPGAS in a commodity fabric - Hyper-Transport over Ethernet (HToE). This has led to the creation and adoption of the HToE commercial standard
- Resource partitioning techniques that use the DPGAS model to save memory power and cost in data centers
- Resource aggregation for efficiently using GPU-accelerated clusters
- Design, development, and evaluation of a software runtime and Application Programming Interface (API), Oncilla, for realizing the DPGAS model

The evaluation of this new global address space model is performed using a hardware reference model, detailed timing simulations, and a software framework with typical data center workloads. This thesis illustrates that the global address space model, when properly matched to applications, can help to reduce overall power usage and cost, and in many cases this model can be used to improve the existing performance of applications by keeping their working sets in memory rather than swapping to disk.

CHAPTER I

INTRODUCTION

Recent growth in the size of data sets in the business world [79] has led to an increased need for computational power and large amounts of memory to support high-performance processing of what has become known as “Big Data.” In-core applications can be used to satisfy large memory footprints while new accelerator hardware, such as graphics processing units (GPUs), provide additional computation power to process large data sets.

However, developers for cluster-based applications are constrained by current blade designs that present barriers to low overhead, dynamic sharing of memory and accelerator resources across nodes. If such sharing were seamless, memory and accelerator resources could be aggregated across nodes in response to the requirements of applications rather than being provisioned for the worst case usage scenario. Even during periods of high demand, dynamic resource sharing could then be used to provision nodes for the average case (e.g., in terms of memory footprints), which would result in significant power and cost savings. Currently, existing techniques assume that applications can only efficiently access resources on the same node and that accelerators and memory on remote nodes must be accessed through a protected software interface or message-based buffers. These constraints illustrate a fundamental problem for managing memory resources in data centers: *Efficient management of host and accelerator memory resources in data centers is difficult due to the cost of existing hardware solutions, the limitations of related software stacks, and the lack of a flexible resource aggregation and resource sharing model amenable to acceleration of “Big Data”.*

This problem statement indicates that there is a need for an approach that supports simplified application programming and that provides high-performance data movement using scalable remote memory accesses. The global address space (GAS) model [27] satisfies these requirements because it provides a simple mechanism for accessing remote memory (one-sided put/get operations) and because it scales well as the size of a cluster grows. However, previous implementations of the GAS model have typically required custom interconnects [157], which can be expensive in terms of cost, or complicated software layers, which can reduce the performance of data movement between nodes [57]. Due to these existing limitations, a new GAS-based model, Dynamic Partitioned Global Address Spaces (DPGAS), is proposed as a suitable memory management scheme that can be constructed using low-cost and high-performance commodity hardware. This leads to the following thesis statement: *Global address spaces built using commodity parts can achieve efficient memory provisioning for data center applications by supporting high-performance and low-cost data movement.*

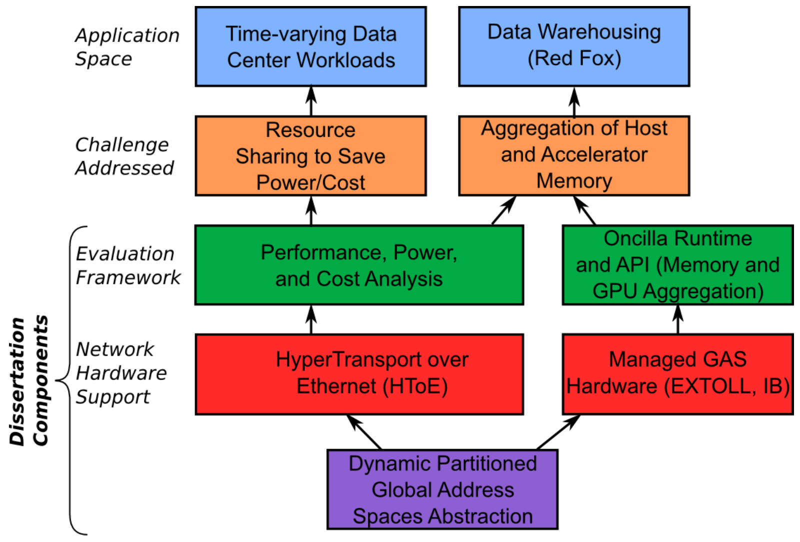

Application Space
Time-varying Data Center Workloads
Data Warehousing (Red Fox)
Challenge Addressed
Resource Sharing to Save Power/Cost
Aggregation of Host and Accelerator Memory
Evaluation Framework
Performance, Power, and Cost Analysis
Oncilla Runtime and API (Memory and GPU Aggregation)
Network Hardware Support
HyperTransport over Ethernet (HToE)
Managed GAS Hardware (EXTOLL, IB)
Dynamic Partitioned Global Address Spaces Abstraction
Dissertation Components

Figure 1: Challenges Addressed by the DPGAS Model

This thesis demonstrates the design of the DPGAS model as well as architectural and software support that can be used to efficiently manage varied resources (memory and accelerators) in the data center for Big Data workloads. At a high level, this thesis can be thought of as addressing the following challenges: 1) How can hardware-supported GAS be designed in a manner that uses commodity parts, and thus is cost-effective and easy to build? 2) How can GAS be used to reduce memory overprovisioning and thereby to reduce hardware power and cost in data centers? 3) How can GAS enable efficient aggregation of host and accelerator memory resources in a programmer-friendly manner? The DPGAS abstraction is proposed to address the first challenge, and HyperTransport over Ethernet (HToE) is proposed as an implementation of this model using commodity parts (HyperTransport and Ethernet). As Figure 1 shows, challenge 2) is addressed using network hardware support in the form of the "converged" fabric, HyperTransport over Ethernet and power, cost, and performance analysis. Challenge 3) is addressed in part by performance analysis of an HToE-based accelerator cloud and in part by the development of the Oncilla software framework that is demonstrated by taking advantage of two existing network layers, InfiniBand and EXTOLL.

1.1 Contributions

This thesis makes contributions in the following areas.

1.1.1 Research contributions

DPGAS Model: Hardware-supported, Dynamic GAS This work proposes that a model built on a dynamically composed partitioned global address spaces should be used to provide high performance sharing of and aggregation of resources. Unlike previous GAS models, DPGAS is built around the concept of extending existing physical node addressing (typically using 40 to 48-bit addresses) to a 64-bit GAS that can be remapped at run-time via specialized hardware support. As applications are mapped to different address ranges in the 64-bit global address space, the physical location of each application's memory can be migrated to local or remote memory using specialized network hardware and non-coherent put/get operations. This abstraction allows for extremely performant partitioning and/or aggregation of accelerator and host memory.

Converged, commodity fabrics: HyperTransport over Ethernet (HToE) The DPGAS model is implemented with the creation of the HyperTransport over Ethernet hardware layer, which was the focus of research and an industry-related specification. HyperTransport over Ethernet is based around non-coherent data transfers, and it focuses on creating a low-latency path through hardware that is then used to investigate two problem spaces. Research using this fabric shows that a on-board interconnect (HyperTransport) typically used for I/O device traffic and an off-board interconnect (Ethernet) can be combined to create a low-latency network layer that can provide performance benefits to two different application spaces, time-varying data center workloads and applications that rely on aggregated memory and accelerators.

Reducing Dynamic Random Access Memory (DRAM) Overprovisioning in Data Centers One challenge addressed by this thesis is how to provision memory in data centers for applications that may have significantly varied memory footprints with respect to time. This thesis proposes that average provisioning of memory should be used in data centers and that the DPGAS model and its spill/receive memory allocation algorithm should be used to satisfy temporary memory requirements for applications. Offline and online analysis is used to determine the performance characteristics of using HToE to implement average-case provisioning as well as to determine potential cost and power savings from reductions in installed DRAM.

Supporting Aggregation of Accelerator and Host Memories The DPGAS model is also used to investigate the challenge of how GAS can be used to aggregate host and accelerator memory efficiently for applications executing in accelerator clouds (clusters with a limited number of accelerators). Further research using HyperTransport over Ethernet to support accelerator clouds demonstrates the importance of hardware design in supporting aggregation. Experiments showed that network adapters that are built to support the DPGAS model need to be matched closely with the application(s) that a specific cluster is planning to support. For instance, the data warehousing application investigated for this research requires high bandwidth streaming to a few accelerators, which puts more network pressure on a small number of "receiver" adapters. This work demonstrates both the benefit of using dynamic hardware updates via DPGAS but also the necessity to match hardware characteristics with the desired application's communication characteristics.

Insights into Programmer-Accessible Software to support DPGAS The Oncilla software framework provides the basis for using multiple different network layers to aggregate local and remote accelerators and memory. This portion of thesis research used Oncilla to investigate managed GAS network fabrics (EXTOLL and InfiniBand) and to evaluate high-performance data transfer with a sample business application (data warehousing) and HPC application (breadth first search). Results showed that EXTOLL provides better latency behavior for applications with many small packets while InfiniBand is best suited for high-bandwidth applications including data warehousing acceleration, and this chapter further enumerates the causes of this behavior.

This work also demonstrated that a software framework could be used with non-coherent GAS to provide a high-performance in-memory implementation of the target data warehousing application. Finally, Oncilla was used to compare the programmability of normal message passing paradigms (MPI) with GAS-based models for applications with GPU accelerators and large data sets. This last insight that Oncilla can be used to make accelerator-based clusters more accessible to programmers is discussed with respect to other potential application spaces that were not evaluated in this thesis, including Hadoop, large graph-based applications, and large HPC codes.

1.1.2 Engineering contributions

There are two main engineering contributions that were the result of research for this thesis:

HyperTransport over Ethernet Hardware Reference Design As part of the research performed before the approval process of the HyperTransport over Ethernet specification, a hardware design was created that was targeted towards a network adapter with an on-board Field Programmable Gate Array (FPGA). This design was then used in conjunction with the completed HToE specification to create a detailed timing solution that was used to evaluate the characteristics of an HyperTransport Ethernet Adapter (HTEA). Optimizations incorporated as part of the reference design (e.g., tag remapping) and further refinements for the simulation (parallelized hardware-based flow control) would be useful to future implementers in terms of providing a real-world baseline for performance and logic usage.

Oncilla Runtime and Application Programming Interface (API) The Oncilla runtime and API is a two-part software framework that allows for the realization of the DPGAS model using "managed" GAS and Remote Direct Memory Access (RDMA) based interconnects, such as InfiniBand and EXTOLL. Performance is enhanced through optimized implementations using the Oncilla runtime, and programmability is enabled through the use of a simple API that supports allocation, aggregation, and high-bandwidth data movement for remote and local host and accelerator memory. This software framework is evaluated in this thesis using a data warehousing application, TPC-H, and two clusters with InfiniBand and EXTOLL networking, respectively.

1.2 Organization

This thesis is organized as follows.

Chapter 2 addresses the origin of global address spaces as well as current implementations of this model. In addition, resource management is discussed as it relates to memory aggregation and disaggregation (the use of memory blades or a shared pool of memory) and the management of memory and accelerator memory resources for recent heterogeneous clusters based on GPUs. Additional related work is included in each chapter as it relates to each problem space that is evaluated.

Chapter 3 presents details on the Dynamic Partitioned Global Address Space (DPGAS) model. First, this chapter addresses relevant trends for memory and interconnects that provide motivation for implementation of the DPGAS model in data centers. Next, DPGAS is presented as an architecture and memory model that is built around a dynamically-composed 64-bit global address space. As applications are mapped and remapped into this GAS, the location of the application's physical memory may change. This chapter also discusses DPGAS control path support, which defines OS support for DPGAS mappings. Finally, the challenges that are addressed in this thesis using the DPGAS model are discussed in detail.

Chapter 4 discusses architectural support for the DPGAS model through the implementation of a network adapter for a newly proposed converged fabric, HyperTransport over Ethernet. This hardware design is implemented using Verilog to support efficient data movement between nodes in an AMD- and Ethernet-based cluster. A hardware prototype is evaluated and offline and online simulations are used to evaluate HToE's suitability for addressing the challenge of sharing memory resources in data centers when application footprints are highly variable.

Chapter 5 discusses the design and ratification of the HyperTransport over Ethernet specification. This specification was influenced by initial research performed as part of this thesis, and HToE was further fleshed out through a collaborative effort between AMD, other companies that make up the HyperTransport Consortium, and this author. Support for some of the most unique features are discussed including tag remapping, HToE retry and recovery, and support for hardware-based flow control.

Chapter 6 builds on the DPGAS model and new features suggested by the HToE specification, and it demonstrates how HToE can be used to support an accelerator cloud, a cluster with a limited number of accelerators that are shared using GAS-based semantics. This accelerator cloud system model is used to address a new application space, data warehousing. The design of a specification-compliant HToE adapter is discussed, and several performance optimizations for this adapter are evaluated for the target data warehousing application.

Chapter 7 details a system-level approach to implement the DPGAS model using “managed” GAS fabrics (InfiniBand and EXTOLL) for the data warehousing application. A software runtime and API called Oncilla is proposed to enable the use of high-performance networking layers with put/get style programming for remote memories and accelerators. Oncilla is evaluated using EXTOLL and InfiniBand with a data warehousing application and with a small HPC-related application, and programmability is briefly discussed for a sample CUDA kernel.

Finally, Chapter 8 presents future areas of research for the DPGAS model and the Oncilla software framework as well as a brief summary of the contributions of this thesis.

CHAPTER II

RELATED WORK

The related background work for this thesis can be broken down into three distinct areas: (1) the development of hardware (interconnects) and software to support global address spaces, (2) techniques for aggregating and disaggregating (or partitioning) memory across nodes in a cluster, and (3) the development of General Purpose Graphics Processing Units (GPGPUs) for use with data warehousing applications and related developments in data movement for accelerators. In addition to this overview, more specific related work for certain problem spaces is included in each chapter as each space is further evaluated in relation to the thesis hypothesis.

2.1 The Global Address Space Model and Related Interconnects

While some of the earliest multiprocessors (multiple processors with shared resources, such as memory) used the concept of a flat address space, the concept of a global address space largely has developed from research in multicomputers (multiple nodes each with their own processors and memory) and the related field of parallel computing. One of the earliest references to a flat 64-bit GAS was published in 1992 with the development of the Opal 64-bit address space system for use with a MIPS R3000-based machine. Also in the early 1990's, custom machines like the J-Machine [116] started to take advantage of fast interconnects to implement message passing schemes that allowed reads and writes to remote memory on distinct cluster nodes. Other machines, such as the CM-5 [64], Princeton's SHRIMP [19], AP1000+ [61], Stanford's FLASH [92], and Wisconsin's Typhoon [136] focused on using combinations of message passing and a new "virtual shared memory" model [94] (later known as the Distributed Shared Memory (DSM) model) to implement efficient sharing of a global memory. While each of these projects suggested different schemes to achieve high performance, they all focused on the use of a custom hardware interconnection network along with a custom software layer to help manage sharing across nodes and to handle memory coherence.

Hardware-based global address space support continued to evolve throughout the 1990's but complete support for this model was limited mainly to custom-built machines and super computers, including Cray's T3D and T3E. The T3D had hardware support for GAS as well as caching of remote memory accesses, and the T3E had expanded global address support that used high-performance "e-registers" and the shared memory (SHMEM) application programming interface (API) to support applications without the need for caching of remote memory accesses. This hardware support continued in later products developed in the 2000's, including the SeaStar [22] and Gemini interconnects [143] that were built around a commodity interconnect for AMD's Opteron processor, HyperTransport.

In addition to increasingly advanced memory sharing hardware, two software-only approaches for memory "sharing" were first proposed and widely studied in the early 1990's. The Message Passing Interface 1.0 (MPI) [109] and Parallel Virtual Machine 2.0 (PVM) were both introduced in 1991 and continue to be used today for parallel systems. MPI has gained slightly more traction in the scientific community due to its feature-rich set of communication primitives, but PVM remains relevant due to its support for heterogeneous platforms, resource allocation, and fault tolerance [56]. These two specifications are important in that they represent some of the first software to address the problem of implementing high-performance data movement without requiring custom hardware to implement data transfer.

Other software that focused on managing shared memory from this time period included the Active Messages platform [44], Split-C [33], SHMEM [144], Co-Array Fortran [29], and Global Arrays [114]. Of these languages, SHMEM was used mainly in the context of Cray super computers, such as the X1 [13], and Global Arrays and Split-C led to the development of the aggregate remote copy interface (ARMCI) library [113] and Unified

Parallel C (UPC) [155]. UPC is one of many partitioned global address space (PGAS) languages (along with Co-Array Fortran and Titanium [168]) developed in the late 1990's that combine the idea of a globally shared memory model with local "private" memory that is reserved using specific language keywords. PGAS languages are intended to increase programmer productivity (i.e., to enable easier use of reads/writes to remote memory) for shared memory systems, so they tend to focus on programming constructs that enable features such as (1) one-sided communication via remote memory "put" or "get" operations, (2) consistency structures such as barriers and fences, and (3) collective communication support like broadcast across all processes. These languages also are intended to supplement APIs like MPI to improve the speed of data movement between cluster nodes.

In addition to these languages, one research group has focused on building a low-level run-time called GASNet [21] that is based off the Active Messages specification. GASNet provides one-sided communication primitives that are mapped to specific hardware via "conduits", and higher-level primitives and PGAS languages can then run on top of the GASNet framework. This run-time and the UPC language continue to be heavily investigated for use with traditional HPC applications [115].

Global address space hardware and related languages evolved again with DARPA's High Productivity Computing Systems (HPCS) program in 2002, which led to further development of Chapel, X10, and Fortress [99]. These languages are important for their focus on programmer productivity (similar to PGAS languages), but the combined Cray hardware and Chapel programming language that were selected as part of the HPCS program has so far not made significant in-roads on PGAS- or MPI-based programming due to its development as a revolutionary code base rather than an incremental update to existing languages like C and Fortran. This approach offers the possibility of much higher performance but also requires significant effort to port existing application code.

At the same time that research groups were developing hardware and novel programming languages for handling shared memory, new custom interconnects started showing up in super computers and parallel computers. Quadrics [128] released its first QSNet cards for the Meiko CS-2 machine, and Myrinet [20] began developing products that had support for the GAS model through their GM message passing library. Both of these products offered high performance but were eventually surpassed in market share by an industry-sponsored interconnect, InfiniBand [54].

InfiniBand was initially developed from two competing efforts to implement the Virtual Interface Architecture (VIA) Specification [43], Future I/O and Next Generation I/O. The combined industry effort for InfiniBand (initially called System I/O) developed a standard methodology for implementing Remote Direct Memory Access (RDMA) and kernel bypass operations that could be used to decrease the involvement of the CPU in initiating data transfers to remote nodes and to improve the performance of these transfers with a standard, high-speed interconnect. The development of this specification meant that for the first time, high-performance data movement between nodes could be realized with an industry standard interconnect that could be developed by competing companies. Since then, InfiniBand has made significant in-roads in the HPC space due to its high bandwidth and low latency, and it currently is used in 41.8 % of the SuperComputing Top500 [70].

Although Ethernet has been one of the most important and common interconnects in smaller HPC clusters and data centers over the past 20 years, the introduction of InfiniBand, Myrinet, and Quadrics in the mid-2000's showed that one Gigabit per second (Gbps) Ethernet did not have high enough bandwidth or low enough latency to compete in the HPC space. A standard for 10 Gigabit Ethernet (10GE) in 2002 [1] and the more recent IEEE 802.3ba standard for 40 and 100 Gbps Ethernet provided the opportunity for Ethernet to compete with InfiniBand in terms of bandwidth, but Ethernet's typical coupling with a TCP/IP software stack meant that concerns about latency still existed [9] [98]. However, switch manufacturers, such as Woven Systems (now Fortinet) in 2007 demonstrated that 10 Gigabit Ethernet with TCP offloading can compete in terms of performance with single data rate (SDR) InfiniBand; both fabrics demonstrated latencies in the low microseconds during a Sandia test [164]. In addition, switch manufacturers have built 10 Gigabit Ethernet devices with latencies in the low hundreds of nanoseconds [34] [169], which has helped mitigate some concerns about Ethernet latency.

On the software side, the Internet wide area RDMA protocol (iWARP) stack has been designed to offer RDMA support similar to what the InfiniBand Verbs layer provides. 10GE iWARP adapters have shown latencies that are on the order of eight to ten microseconds, as compared to similar InfiniBand adapters with latencies of four to six microseconds [35]. More recent tests have confirmed that 10 Gigabit Ethernet latency for MPI with iWARP is in the range of eight microseconds [89].

Ethernet has further been enhanced to compete with InfiniBand through the release of specifications to support Data Center Ethernet, aka Converged Enhanced Ethernet (CEE) [90]. These specifications effectively provide "lossless" Ethernet channels and other connection-oriented features that can be implemented at the link-level (L2) layer. One of the motivating developments for these new specifications was the desire to have "converged" interconnects that could take advantage of Ethernet's ubiquity and ease of use in data centers. FibreChannel over Ethernet (FCoE) [151] was the first converged interconnect to take advantage of Ethernet encapsulation, but newer specifications have since proposed encapsulating more specialized interconnects over common interconnects (such as Ethernet and InfiniBand) to reduce overall cabling and take advantage of the higher bandwidths available in newer interconnect standards. For instance, a recent specification has proposed the use of InfiniBand's Verbs layer over Ethernet, also known as RDMA over Converged Enhanced Ethernet (RoCEE) [30], and some companies have shown interest in using PCI Express over Ethernet [150]. In fact, the new PCI Express 3.0 (PCIe 3.0) standard alludes to this future of converged fabrics with the addition of a "PMux" feature that supports sending packets from other protocols over PCIe links.

This recent proliferation of low-latency commodity interconnects means that both traditional HPC clusters and newer data centers can easily incorporate dedicated hardware support for GAS-based memory sharing via one-sided put/get operations. In addition to software-oriented stacks such as GASNet, several recent specifications support dedicated hardware for commodity GAS. The HyperShare platform includes three such specifications: native HyperTransport (HT) [42], encapsulation of HT over Ethernet [170], and HT over InfiniBand [17]. Other groups have designed custom off-chip fabrics, such as the EXTOLL group and associated corporation [52] and NumaScale, which provides coherent HyperTransport support between nodes with its NumaConnect adapter [118].

2.2 Memory Aggregation and Disaggregation

This thesis deals with resource provisioning, and it specifically focuses on techniques to better enable the aggregation and disaggregation (or partitioning) of scarce memory resources. Chapter 4 focuses on how disaggregation can be used to more efficiently provision memory for time-varying applications, so the related work focuses on this aspect of memory partitioning.

2.2.1 Memory Aggregation

Memory aggregation has been recently investigated in the data center world for large "in-core" applications that run faster when they do not have to swap out to disk from DRAM. While early multiprocessors provided a physically shared memory to run large jobs, current software and hardware approaches focus on aggregating memory across a cluster built using the DSM model.

Several research groups have investigated implementations of in-core databases, most notably the RAMcloud project [126], vNUMA [26], and the MEMSCALE project for MySQL-based databases [108]. Oracle, SAP, ScaleMP, [138] and others also have commercial solutions, typically based on TCP/IP implementations. However, this this is some of the first research to focus on the specific problem of using a limited number of accelerators with a GAS model to improve the performance of in-core databases.

2.2.2 Memory Disaggregation

Disaggregation, or partitioning of memory, has recently become an important topic for data centers due to power concerns related to servers that are typically idle [11] and are over-provisioned according to their average workloads [96]. The dynamic, partitioned global address space (DPGAS) [167] [176] model discussed in Chapter 3 is a type of disaggregation that has its roots in the concept of "virtualized" resources that was furthered by projects including Cellular Disco [59] and global memory [49] or page swapping [95] schemes for page caching.

The closest related work to DPGAS is HP's disaggregated memory research [96] that was later incorporated as part of the design of their low-power Project Moonshot server blades [68]. While DPGAS advocates using the "idle" memory on existing blades in a data center, the disaggregated memory approach specifies that DRAM should reside in a separate blade for cost and power reasons. These memory blades increase available memory bandwidth and enable coherent sharing between client nodes, but the DPGAS research differs in that it focuses on using existing DRAM resources more efficiently to reduce overprovisioning. While disaggregated memory provides better bandwidth than DPGAS, it may also prove cost-prohibitive for small to medium data centers due to extra hardware requirements.

2.3 Accelerators and Data Movement

APIs like the compute unified device architecture (CUDA) and open computing language (OpenCL) have been used with HPC applications for the past few years, but more recent research has focused on using these accelerator-oriented APIs to handle traditional data center tasks, including using GPUs to perform database queries [8]. However, there has been relatively little focus on merging traditional in-core databases with data transfers to and from remote accelerator memory.

Support for accessing remote accelerators has been greatly improved through the efforts of the high-performance community and more specifically by NVIDIA and the MPI community. NVIDIA has greatly simplified device memory addressing through the use of their Unified Virtual Addressing (UVA) in CUDA 4.0, the inclusion of GPUDirect support that can facilitate peer-to-peer transfers between GPUs (GPUDirect 2.0) [140], and efficient transfers of data from the GPU to the NIC using pinned pages in the host operating system (GPUDirect 1.0) [141]. More recent improvements in CUDA include CUDA inter-process communication (cuIPC) which allows for the transfer of data between distinct processes that can exist on separate nodes [85].

The MPI community has built on top of these improvements in CUDA to provide a seamless layer that integrates CUDA and MPI two-sided and one-sided transfer functionality. MVAPICH [162] and OpenMPI [156] have both provided implementations for using MPI to efficiently transfer data between remote and local GPUs and remote memory. However, due to the current limitations of CUDA and GPUDirect, both APIs must copy data through host memory when doing a transfer between a local and a remote GPU. The MVAPICH group also has started work on supporting standard data center applications by creating an interposer library that hides the complexity of the InfiniBand Verbs stack from the application developer [127]. A related project, MPI-ACC [2] provides similar support for using MPI with GPU memory, but it supports the OpenCL runtime in addition to the CUDA runtime.

While GASNet provides full-featured support for implementations of PGAS languages and data movement, there is currently little work on using it with CUDA. Two projects that have recently started to investigate using GASNet with accelerators is the Stanford Legion project [106] and NVIDIA's Phalanx project, which combines GASNet with CUDA support and task scheduling across multiple nodes [55]. Various other approaches have focused on using virtualization to share remote GPUs [105] [133]. Despite these many advances in data movement between accelerators, most of these projects are still focused on the needs of the high-performance computing community. This focus means that most prior research has advocated the use of MPI and other techniques that may not be easily applied to existing in-core data warehousing applications. The MVAPICH group [127] is one of the exceptions to this trend as they are focused on both HPC applications [37] and business applications [83].

2.4 Concluding Remarks

The current landscape for clusters illustrates a long history of improvements in concepts related to global address space models and more importantly to performance, cluster interconnects. At the same time, GPUs have become a larger part of HPC clusters and to some extent, data centers, but data movement for these new accelerator devices is still tied to HPC-style communication paradigms like message passing. Business applications that have typically been designed with a single multi-threaded node as the target hardware are not easily mapped to traditional HPC methods of communication. This thesis targets this unfulfilled area of data movement and associated resource management using a high-performance implementation of GAS as discussed in Chapter 3

CHAPTER III

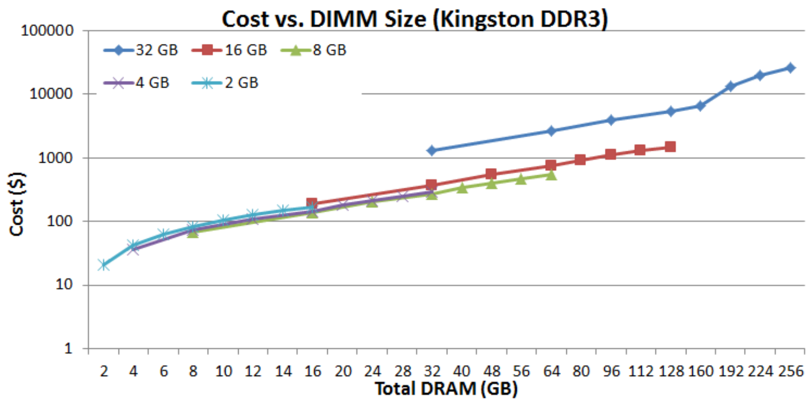
DYNAMIC PARTITIONED GLOBAL ADDRESS SPACES

3.1 Motivation

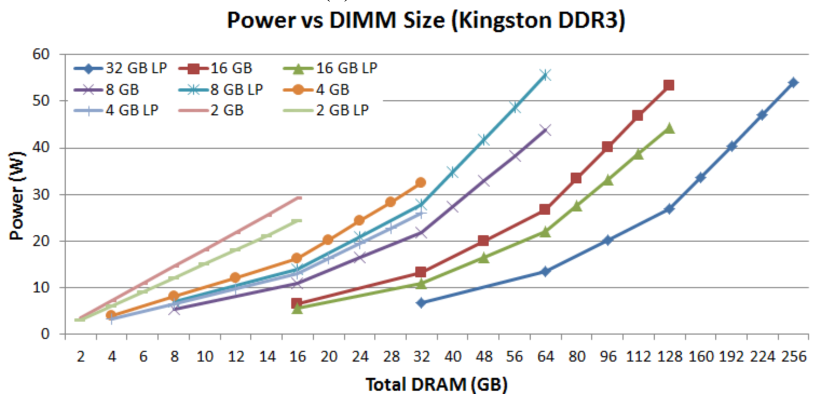
As discussed in Chapter 2, the past 20 years have presented several new communication models for data movement in clusters, including message passing, shared memory, cache-coherent NUMA, and the one-sided put-get model. Each of these models has distinct advantages and disadvantages that impact application performance. However, there are also other ramifications of using a specific communication model with respect to its impact on total power and cost usage in a cluster or data center.

3.1.1 Memory cost and power motivations

While many architectural improvements in data center servers have focused on reducing power consumed by CPU and disk, several studies [66] [93] indicate that DRAM power can consume up to 30% of the total hardware power budget and thus should not be overlooked. One of the challenges in optimizing memory power usage is the time-varying nature of data center workloads [58]. Servers within the data center are usually overprovisioned to handle memory footprints associated with peak workloads, which leads to excess static power dissipation. However, Figures 2a and 2b show that there is a trade-off between the initial purchase price of DRAM and the power consumption of certain sizes and numbers of dual-inline memory modules (DIMMs). While larger DIMMs have lower power consumption, they are also dramatically more expensive than using more common DIMM sizes that benefit from manufacturing economy of scale. For instance, provisioning a server with 128 GB of RAM using four 32 GB DIMMs uses about 60% of the power of using eight 16 GB DIMMs (26.96 W vs. 44.28 W) but also costs almost four times as much ($5280 vs. $1480).



(a) Cost Trends



(b) Power Trends

Figure 2: DDR3 DRAM Power and Cost Trends (2012)

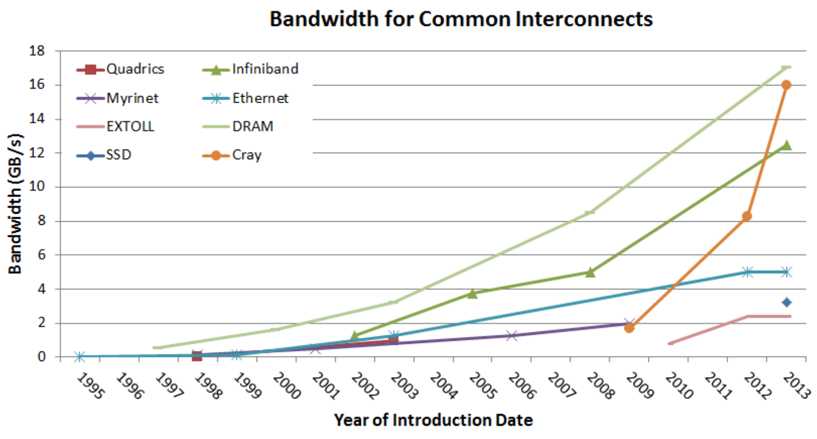
Furthermore, commodity server blades are packaged in a manner such that remote blades can share only via heavyweight, OS-coordinated mechanisms between blades. Remote Direct Memory Access (RDMA) and custom NUMA interconnects have provided two approaches to make DRAM sharing easier, but they are limited by OS page registration overhead and the requirement for non-commodity parts, respectively. This limited model of sharing leads directly to overprovisioning of memory resources and also can limit the performance of cluster resource aggregation techniques (as discussed in further detail in Ch. 6).

3.1.2 Interconnect motivations

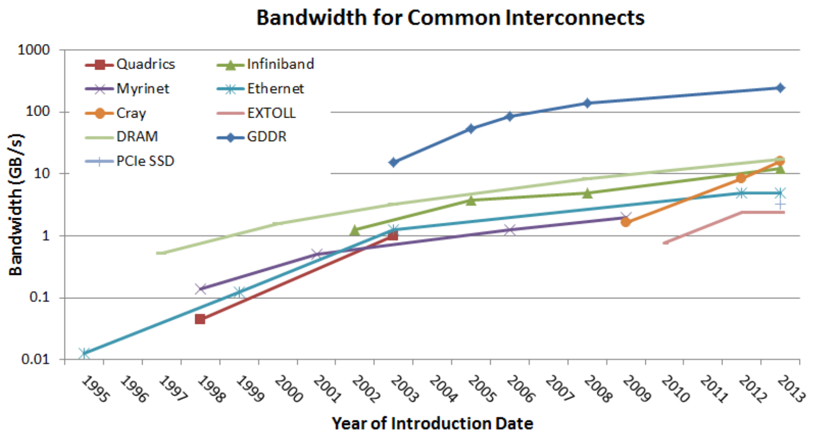
The trade-off between overprovisioning and average provisioning of memory resources in data centers might seem like a problem that is solely based on the number of applications that can fit within one blade's memory, but recent trends in interconnects have emphasized that inter-node memory sharing can be considered as another suitable optimization for cluster resource management.

As Figure 3a shows, maximum DRAM bandwidth has risen from .53 GB/s in 1997 to 17.10 GB/s (single-channel DDR3-2133 MHz) in 2013. During the same time frame, interconnect bandwidth has risen even more dramatically from 0.01 GB/s to 16 GB/s for the Aries interconnect included as part of Cray's new Cascade system [3], shrinking the bandwidth gap between memory and interconnect from 40x to $\sim$0.06x. On the other hand, Figure 3b illustrates a divergence between the memory bandwidth of normal systems and the memory bandwidth of stream-based processors, like GPUs. Memory bandwidth of systems using graphics double data rate, or GDDR memory (higher-frequency DRAM geared towards GPUs), has gone from being $\sim$4.75x higher than that of standard DRAM in 2003 to $\sim$14.65x higher in 2013 with the introduction of Tesla's K20X GPU [121]. This disparity is due to the higher clocking frequency of GDDR and the use of 384-bit-wide memory buses compared to 128-bit buses used with dual-channel DRAM on most desktop systems. Taking both of these figures together, interconnect bandwidth has largely matched DRAM bandwidth improvements but accelerator memory bandwidths have grown at a much faster rate to match recent improvements in GPU compute architecture.

Figure 4 shows the trend in latency improvements for interconnects as compared to DRAM latency (measured for t_{CAS}). The latency of various interconnects is measured for one small message "ping", or one-sided write, from a local to remote node. In general, interconnect latency has decreased from being $\sim$240,000x higher than DRAM latency to $\sim$100x higher for low-latency fabrics like InfiniBand and EXTOLL (see Ch. 7 for more information on EXTOLL). The dramatic convergence of on-board latency (DRAM) and off-board latency (inter-node interconnects) provides a powerful motivation for accessing remote memory, and this thesis builds around this concept of using high-performance interconnects to provide easier sharing of memory resources while also providing suitable



(a) DRAM and Interconnect Bandwidth



(b) GDDR and Interconnect BW

Figure 3: Interconnect Bandwidth Trends

performance to applications.

Many of the applications that could benefit from using a high-performance interconnect to share cluster resources are traditionally served by a model built on a limited amount of memory that is backed up by a much larger data store on disk, or increasingly by PCI Express-based SSDs. Figures 3a and 4 show the latest bandwidth and latency numbers for a “top-end” PCIe SSD [122]. These data points demonstrate that PCI Express-based disks provide reasonable bandwidth (3.2 GB/s vs. 12 GB/s for high-end interconnects), but latency still lags much farther behind other interconnects (50 μs vs. 1 μs for InfiniBand). When combined with other performance penalties related to swapping pages to disk, this disparity in bandwidth and latency for disk-based storage provides another motivator to

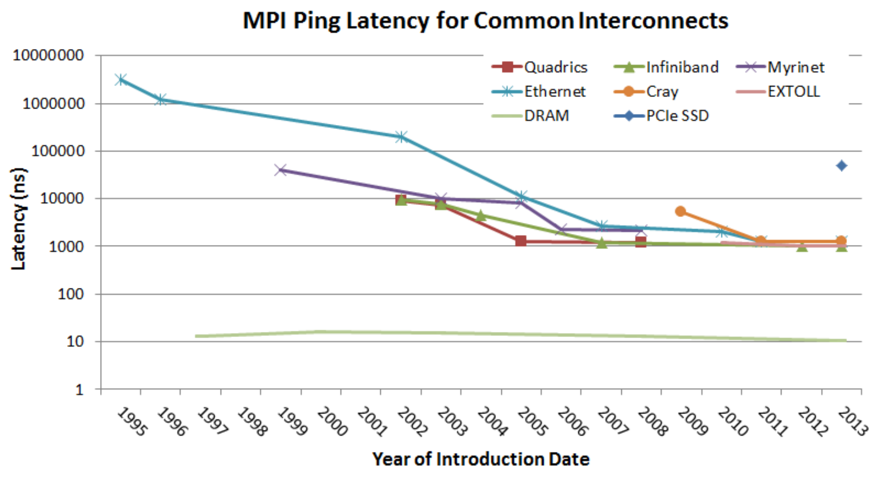


Figure 4: DRAM and Interconnect Latency Trends

keep applications in-memory or "in-core".

3.2 The Dynamic Partitioned Global Address Space Model

To provide efficient resource sharing, a data movement technique is needed that handles peak memory workloads while also providing benefits that cannot be efficiently obtained by traditional techniques, such as virtual machine (VM) consolidation [159] or memory overcommitment [65]. The Dynamic Partitioned Global Address Space (DPGAS) model is proposed as an abstraction that builds on the traditional software-based Partitioned Global Address Space (PGAS) [24] model while also allowing for customizable, hardware-based remote memory put/get operations. As the hardware-based implementation of DPGAS in Chapter 4 demonstrates, this model supports dynamic updates for remote memory mappings that can also be used to address challenges such as memory overprovisioning or resource aggregation.

Figure 5 shows a simplified overview of the DPGAS model. At the most basic level, the virtual address space of an application can be mapped into the global address space, and the global address space mapping determines where the physical memory for that application is placed. By changing the global address space mapping, the application can be placed in either local or remote memory. While PGAS languages do a similar type of mapping in

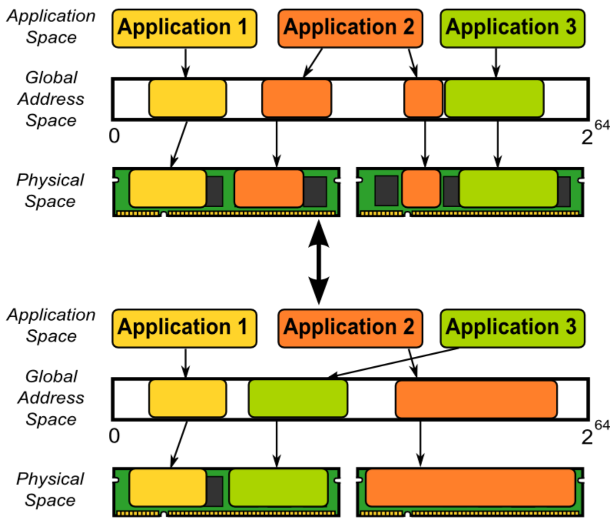


Figure 5: Dynamic Partitioned Global Address Space Overview

software using private and shared compiler flags, the main difference in the DPGAS model is that the mapping can be made at run-time and is manipulated at the hardware level.

The DPGAS model is split into two components: the architecture model, which details a high-level overview of the hardware that is required to implement this dynamic partitioned address space, and the memory model which details how remote memory is accessed and data movement is performed when using DPGAS.

3.3 Architecture model

The DPGAS architecture model assumes that a number of processors exist, and they can all access a distributed, global 64-bit physical address space. A set of cores on a chip will share one or more memory controllers and low-latency link interfaces integrated onto the die, such as those offered with the HyperTransport or QuickPath interface. All of the cores will also share access to a memory management function that will examine a physical address and route a request (read or write) to the correct memory controller—either local or remote. For example, in the recent Opteron systems, such a memory management function resides in the System Request Queue (SRQ) or System Request Interface (SRI) (both are names for the same queue-based structure), which is integrated on-chip with the northbridge [31].

3.4 Memory model

The memory model is built on a 64-bit partitioned global physical address space. Each partition corresponds to a contiguous physical memory region controlled by a single memory controller, where all partitions are assumed to be of the same size. The AMD Opteron uses partitions that can handle up to one terabyte (TB), which corresponds to the 40-bit Opteron physical address. Thus, a system can have 2^{24} partitions with a physical address space of 2^{40} bytes for each partition. Although large local partitions would be desirable for many applications, such as databases, there are non-intuitive trade-offs between partition size, network diameter, packaging constraints, and end-to-end latency that may motivate smaller partitions. For instance, applications may want to use small partitions as a non-coherent message buffer to easily share state between processes on different nodes. Due to these various requirements, the DPGAS model incorporates a view of the system as a dynamically changing network of memory controllers accessed from cores, accelerators, and I/O devices.

The programming model for DPGAS provides *get/put* operations that correspond to one-sided read/write operations on memory locations in remote partitions [114]. A sample *get* transaction on a memory location in a remote partition must be forwarded over the network to the target memory controller (as shown in Figure 6), and a read response is transmitted back over the same network. Once the DPGAS memory model is enabled, an application's virtual address space can be allocated a physical address space that may span multiple local and remote partitions. From an application's point of view, remote physical memory acts as a slightly higher latency "virtual DIMM", and the addition of a fast interconnect allows the use of this remote memory with latency that is still much lower than OS page faults to a standard hard drive.

The set of physical pages allocated to a process can be static (compile-time) or dynamic (run-time). Multiple physical address spaces can be overlapped to facilitate sharing and communication, although coherence is not guaranteed. A visual representation of this

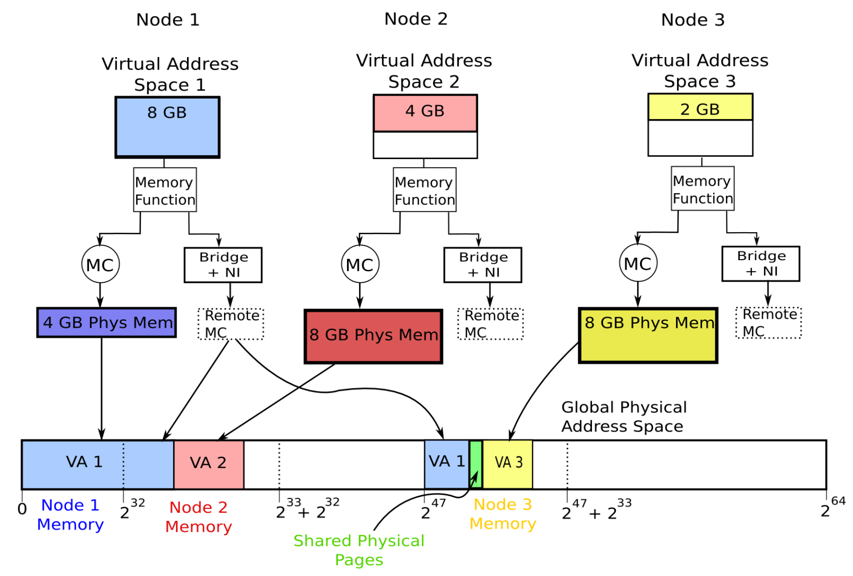


Figure 6: Detailed DPGAS Overview with Memory Functions

overlapping of global addresses to share pages is shown in Figure 6.

Islands of Coherence The DPGAS model operates over a non-coherent global physical address space. However, if blades implement coherence, one can view the DPGAS model as dynamically increasing the size of physical memory accessible by a blade within a coherence domain, as shown in Figure 7. These "islands" of coherence can be maintained using standard protocols for intra- and inter-socket coherence but require additional hardware or software constructs to support coherence between nodes. The specific size of a coherence island is dependent on the coherence requirements of the desired application and on hardware and software that supports the DPGAS model.

Most applications that are designed to run on top of the DPGAS model can benefit from limited consistency models, such as eventual consistency [160]. For instance, several business applications discussed in Section 7.8.1 rely either on eventual consistency or do not have any strict consistency or coherence requirements.

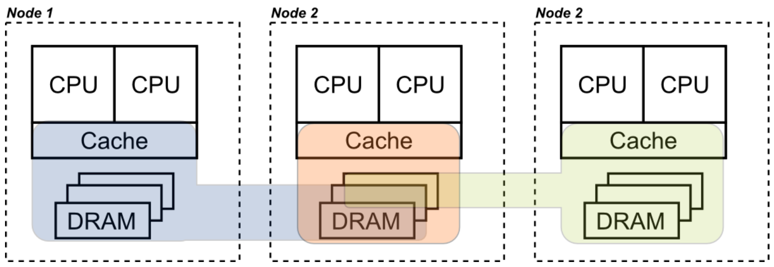


Figure 7: Islands of Coherence

3.5 Required Control Path Support for DPGAS

While data transfer with DPGAS is based on non-coherent put/get operations that utilize hardware support, initial allocations for DPGAS must include some kernel-level support to initiate and manage remote partitions.

3.5.1 Spill/Receive Model

DPGAS control path support builds on existing work using spill-receive for cache models [131] and extends this concept for system-level allocations. Figure 8 shows that one or more node can "spill" some memory requests to a remote memory using the non-coherent put/get access method described in Section 3.4. One or more nodes can "receive" spill requests by partitioning its physical resources to be accessed by remote nodes. Although this figure shows two spill nodes and one receive node, the actual ratio of spill and receive nodes is dependent on the algorithm that is used for allocating and sharing remote memory.

In principle, nodes can also spill and receive memory requests at the same time. For example, if a remote node has one large, contiguous chunk of physical memory but only one application with a small footprint and short runtime, it could in theory "spill" requests for the small application to remote memory and fulfill "receive" requests for long-running applications with very large footprints.

3.5.2 Memory Allocation Algorithms

Memory allocation algorithms for DPGAS are envisioned to be implemented at the operating system level using a simple kernel device driver that communicates via MPI or

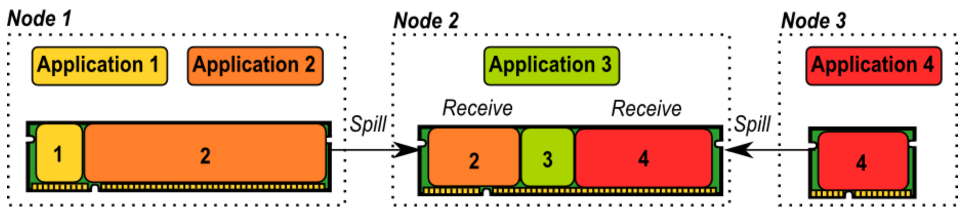


Figure 8: Example of Spill/Receive Memory Allocation

by standard networking sockets. The OS is notified of changes to its available physical memory using the capabilities of libraries like Linux's libnuma or, in the case of a virtual OS, updates to the hypervisor's page tables. The memory allocation daemon negotiates for changes to the global physical address space with daemons on other nodes and then provides dynamic mapping updates to the requesting node's hardware address mapping registers and hardware that implements the DPGAS model. Updates to the underlying hardware and operating system allow for transparent remapping of remote memory accesses without requiring application involvement.

For simplicity, this section uses an AMD Opteron-based system and the HyperTransport over Ethernet (HToE) hardware implementation of the DPGAS model to discuss allocation. For more information on HToE. please see Ch. 4.

The envisioned process for memory allocation with DPGAS is illustrated in Figure 9. On system boot, DPGAS daemons (OS- or hypervisor-level process) are started on each DPGAS-enabled machine, and each blade enumerates its closest (one-hop) neighbors. The DPGAS daemon monitors current DRAM allocations and workloads, keeping track of average load or major page faults (to disk) over a long period of time (similar to running "ps" in the background on a Linux OS). 1) When the average allocation exceeds a preset threshold (e.g. 80% of DRAM or 100 page faults to disk in S seconds), 2) the DPGAS daemon issues a request to a neighboring node (or a dedicated "receive" node) via TCP/IP for M MB of memory. 3) The daemon at neighboring Node B checks to see if it can satisfy the DPGAS memory request and then decides how much of the requested M MB of memory can be allocated. 4) Node A is notified whether its spill request is rejected or

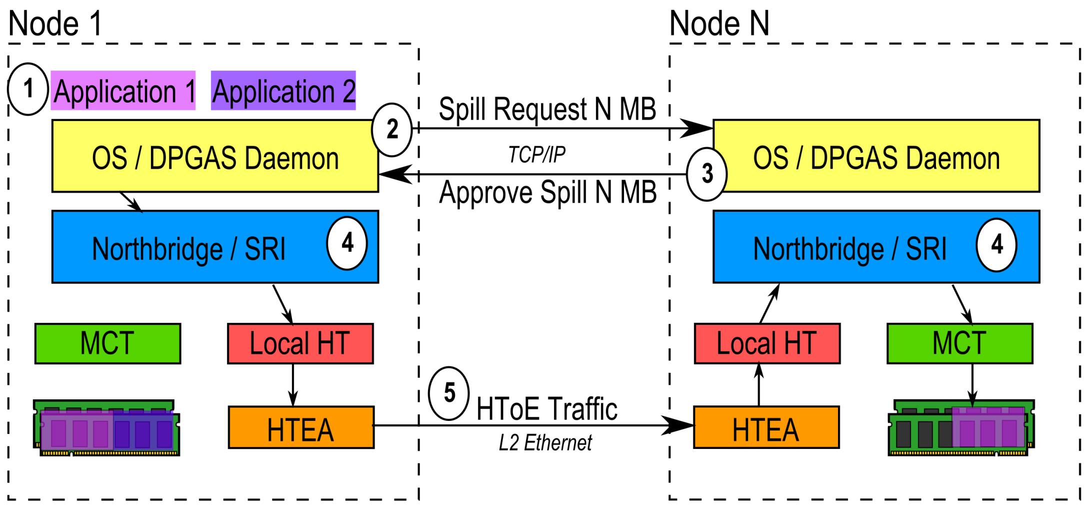


Figure 9: DPGAS Control Path and Memory Allocation

fulfilled. Then both nodes update hardware-based mapping tables in their DPGAS-enabled network adapter (the HTEA) as well as their hardware-based mapping tables in the SRI, and they notify the local VMM or OS of the change in the mapped physical address space. 5) Once this setup is complete, reads or writes to virtual address ranges that are mapped to the DPGAS-based network adapter (HTEA) result in reads and writes (via HToE) to remote memory.

This allocation model is implemented in the simulation-based experiments discussed further in Section 4.7.

3.5.3 Memory Deallocation

Memory deallocation takes place in three situations: 1) the requesting node has finished using the remote memory it requested. Typically, this takes place when its load has decreased below a certain threshold or a hard timeout (set globally for all nodes at start up) occurs. 2) The node that is receiving, i.e. sharing memory via DPGAS, experiences high load (typical when all nodes have equal amounts of memory). 3) Another Node C with better fairness characteristics (fewer requests or a smaller amount of memory requested) needs memory that is currently being used by the original requesting Node A. Obviously, the manner in which memory is forcefully deallocated depends on the overall implementation goals and required QoS metrics of the system, but one possible deallocation technique is as follows: 1) Node B notifies node A that it is revoking its shared memory. Node A must then provide an acknowledgement to confirm that it is done using this remote memory. If Node A does not acknowledge within a preset time limit, Node B will change its hardware-based mapping tables, thus ensuring that requests from Node A to Node B are rejected. 2) Both nodes update their DPGAS-enabled NIC mappings, VMM status, and SRQ tables. 3) Node B proceeds to allocate memory to the next highest-priority node, as requested.

3.6 *Challenges Addressed by the DPGAS Model*

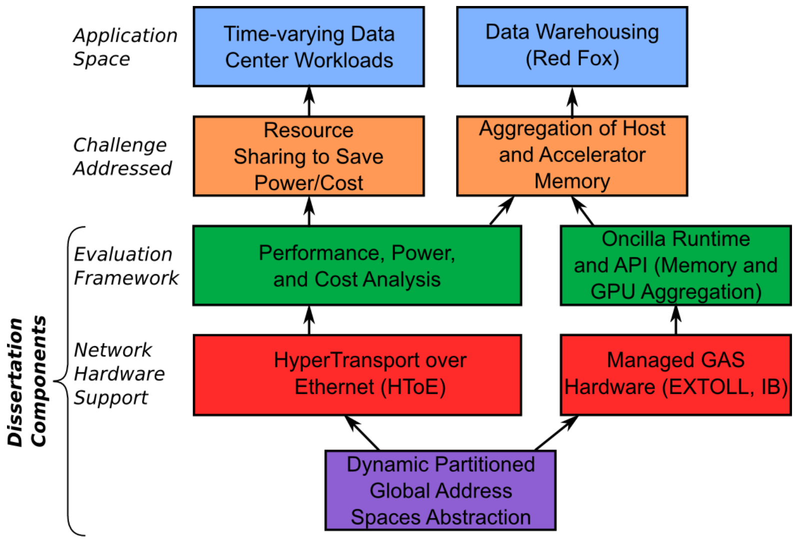

Application Space
Time-varying Data Center Workloads
Data Warehousing (Red Fox)
Challenge Addressed
Resource Sharing to Save Power/Cost
Aggregation of Host and Accelerator Memory
Evaluation Framework
Performance, Power, and Cost Analysis
Oncilla Runtime and API (Memory and GPU Aggregation)
Network Hardware Support
HyperTransport over Ethernet (HToE)
Managed GAS Hardware (EXTOLL, IB)
Dissertation Components
Dynamic Partitioned Global Address Spaces Abstraction

Figure 10: Challenges Addressed by the DPGAS Model

The DPGAS model is used in this thesis to address two main challenges: 1) resource sharing to reduce memory overprovisioning and to reduce hardware power and cost and 2) efficient aggregation of host and accelerator memory resources in a programmer-friendly manner. Figure 10 shows how support for each problem space is implemented using slightly different hardware and software support. Resource sharing is examined using detailed timing simulations and time-varying data center workloads in Sections 4.6 and 4.7, and hardware support for this workload is implemented using HyperTransport over Ethernet (Section 4.4) and a related specification (Ch. 5). This concept of resource sharing is extended to include the aggregation of host and accelerator memory, and is represented by the data warehousing workload (Section 6.1.1) and an in-house data warehouse software framework called Red Fox (Section 6.1.2). Detailed timing simulations are again used to investigate how resources can be efficiently aggregated with the HyperTransport over Ethernet-based implementation of the DPGAS model (Ch. 6). Finally, a full system is evaluated that uses existing network hardware under a managed GAS software layer along with a software runtime and API, Oncilla, that supports remote resource aggregation of memory and accelerators (Ch. 7).

Thesis Contributions This thesis aims to solve the problem of how to efficiently manage varied resources (memory and accelerators) in the data center for time-varying and Big Data workloads, and it uses the DPGAS abstraction along with GAS hardware support to provide efficient management of these resources.

As previously mentioned in Section 1.1, the contributions of this thesis are as follows:

1. A new global address space model, Dynamic Partitioned Global Address Space (DPGAS), that utilizes common networking hardware [167] [176]
2. The proposal of architectural support for DPGAS in a commodity fabric - HyperTransport over Ethernet (HToE) [176]
3. An industry specification for HyperTransport over Ethernet [170] and HyperTransport over InfiniBand [17]
4. Resource partitioning techniques that use the DPGAS model to save memory power and cost in data centers [173]
5. Resource aggregation for exploiting GPU-accelerated clusters [174]
6. Design, development, and evaluation of a software runtime and Application Programming Interface (API), Oncilla, for realizing the DPGAS model [172]

3.7 Concluding Remarks

The Dynamic Partitioned Global Address Spaces, or DPGAS, model presents an abstraction that allows for hardware-based mapping of remote memory partitions to enable efficient sharing and aggregation of host and accelerator memory resources. As Chapter 4 demonstrates, this model requires a tightly integrated networking layer to implement both the architectural model and the memory model that is detailed here.

CHAPTER IV

ARCHITECTURAL SUPPORT FOR THE DPGAS MODEL

4.1 Introduction

While there are several existing interconnect technologies that could be used to implement the high-performance data movement needed for the DPGAS model, this thesis focuses on one standardized on-chip interconnect, HyperTransport, and one off-chip interconnect, Ethernet, due to each interconnect's performance, open specifications, and commodity nature. This chapter details the implementation of hardware to support HyperTransport over Ethernet (HToE) via an adapter called a HyperTransport Ethernet Adapter (HTEA), and it demonstrates offline and online (simulation-based) analysis of this adapter for reducing overprovisioning of memory for data centers.

As previously discussed in Section 3.1.1, servers in data centers are typically overprovisioned in terms of memory resources due to two factors: 1) the time-varying nature of workloads that leads to peaks and valleys of demand for server resources and which makes average provisioning difficult [58] [96] and 2) limited mechanisms for sharing memory resources, with many existing solutions requiring custom hardware or depending on high-overhead software stacks or drivers [18]. The DPGAS model provides a means to perform more efficient sharing and usage of remote memory resources, and the HToE implementation of this model is used to evaluate how more efficient sharing can reduce overprovisioning of memory resources.

4.2 Motivation for Using HT to Support DPGAS

HyperTransport has specific benefits when compared to related protocols, such as PCI Express and Intel's QuickPath Interconnect (QPI), and these benefits were the motivation for this work's implementation of DPGAS using HyperTransport over Ethernet. HyperTransport typically has had slightly lower bandwidth than PCI Express but also lower latency, especially for smaller messages [74]. QPI is more of a layered protocol than HT (similar to PCIe), but it has similar bandwidth characteristics [177].

Despite its similarities to QPI, HyperTransport has been an open standard for high-performance, non-coherent implementations as recently as the early 2000's. The HT standard for non-coherent traffic and HT's early entry into the consumer market (via the Athlon XP in 2003 [73] vs. QPI in 2008 [145]) were two main reasons that HyperTransport was considered as a candidate to support DPGAS. In addition, HyperTransport also played a key role in AMD's Torrenza initiative (announced in 2006 [31]), which aimed to use HyperTransport as an interconnect fabric to support co-processors that were manufactured as CPU socket replacements on multi-socket server boards. At the time that this research was performed, the IBM RoadRunner [10] was widely thought to be the first wave of Torrenza-based servers with HT-enabled co-processors, such as FPGAs, Cell processors, or custom ASICs. This ecosystem did not materialize as more emphasis in the HPC space was placed on GPUs [47] and focus for later HPC interconnects moved towards PCI Express [3]. Despite recent market trends, many of the issues examined with this work's HyperTransport-based implementation are still relevant to other point-to-point interconnects including QPI and PCI Express.

4.3 The HyperTransport Interconnect

HyperTransport (HT) is a point-to-point interconnect standard that was initially defined in 2001 to replace the on-package front-side bus (FSB) for AMD CPUs. There is both a coherent HyperTransport specification meant for traffic between CPU sockets and a non-coherent specification for use with I/O devices. The HyperTransport 3.1 specification [75] defines requirements for a low-latency, high-bandwidth interconnect for coherent memory access between CPU sockets with a maximum one-way bandwidth of 25.6 GB/s (using 32-bit links). The non-coherent variant of HyperTransport can also interact with I/O devices connected to a motherboard by an HyperTransport eXpansion (HTX) connector, such as with the network adapter discussed in this chapter.

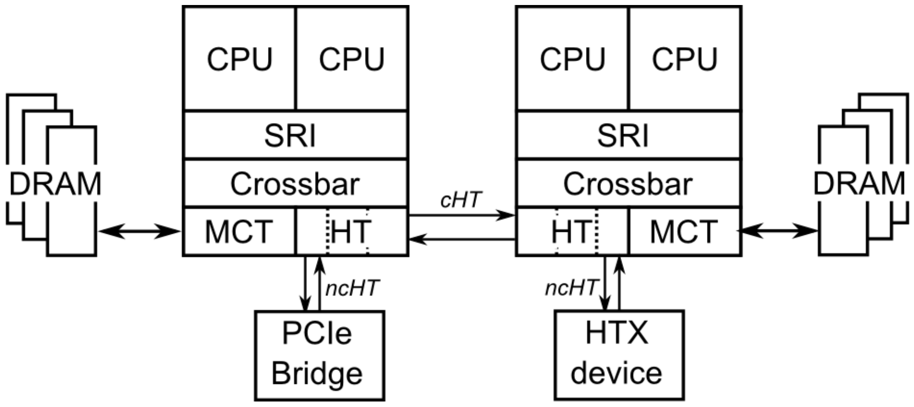


Figure 11: AMD Opteron CPU and Northbridge Overview

Figure 11 shows the role that HT plays in a standard AMD Opteron northbridge, as discussed further in [31]. The HyperTransport links receive packets from the CPU via a queue called the System Request Interface (SRI) or System Request Queue (SRQ) that sits on the backside of the L2 cache. The SRI, memory controller, and HT links are connected together via a high-speed crossbar. Coherence protocol-related requests travel over coherent HT (cHT) links to up to seven neighboring CPU sockets, and non-coherent HT (ncHT) packets are used to interact with devices connected via the HTX connector or with PCI Express devices by using a host bridge to translate between the two packet protocols. DRAM read and write requests are handled by a separate memory controller that communicates to the cHT links and SRI using the crossbar. It should also be noted that the HT links are all connected to a host bridge (not shown) that helps to direct point-to-point HT messages to the correct location, i.e., memory, another HT device, or back to the cache via the SRI.

At the highest level, HyperTransport links are composed of two unidirectional, point-to-point links that are implemented using a 2- to 32-bit control and data (CAD) wire along with a one-bit wire to indicate whether the signal on CAD is a control or data signal. A high-level overview of the HT link between two devices is shown in Figure 12. The specific packet type currently being transmitted on an HT link is indicated by the CTL line. The advantage of this design is the ability to interleave packets - an optimization referred to as priority request interleaving. When a large data packet is being transmitted along a link, a control packet can be injected into the middle of the data transmission and can then be distinguished by the state of the CTL line so that the packet is steered to the correct control buffer.

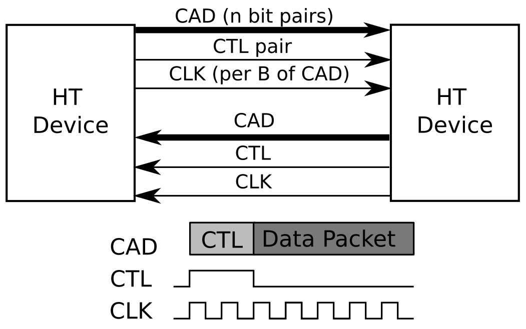


Figure 12: HT Link Overview

HT links operate at double data rate and run at speeds of up to 3.2 GHz and up to 6.4 GTransactions/sec in each direction. With 16-bit links this translates to 12.8 GB/s in each direction and 25.6 GB/s aggregate across a single link. While 32-bit links are not common in current systems, the HT standard defines the maximum aggregated link bandwidth as 51.2 GB/s, as compared to the maximum bandwidth of a PCI Express 3.0 16x link, which is 32 GB/s or 16 GB/s in each direction. Since most current implementations of HT use 16x links, the maximum bandwidth for HT transfers is typically limited to 25.6 GB/s

HT packets contain either a four or eight byte command word and up to 64 bytes of data. Each HT packet is classified according to one of three virtual channels: posted (no response needed), non-posted (response needed), and response packets (used to return data or notifications). Figure 13 shows a HT read request packet along with fields that are used to address and route packets between local I/O devices and main memory in a HyperTransport-based system. The UnitID specifies the source or destination device

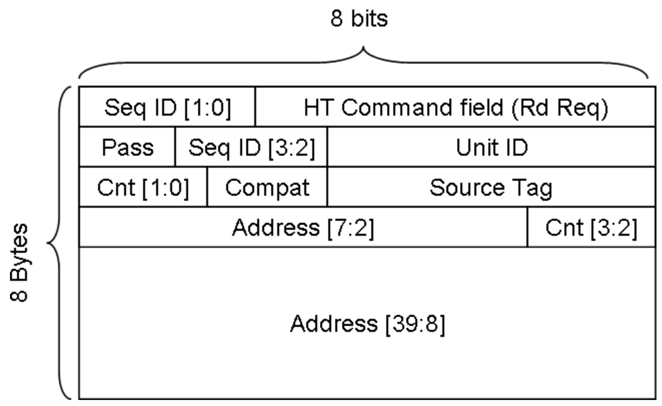
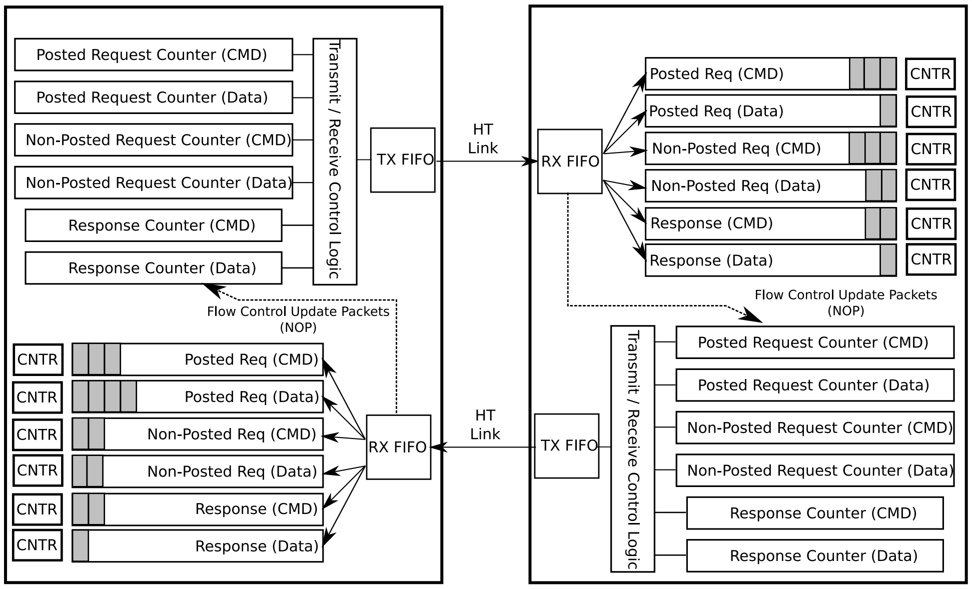


Figure 13: HT Read Request Packet

and allows the local host bridge to direct requests/responses to main memory or up to 32 devices. The Source Tag (SrcTag) and Sequence ID (SeqID) are used to specify ordering constraints between requests from a device, for example, ordering between outstanding, distinct transactions. SrcTag typically indicates the source device for a packet and is used to direct responses which lack address fields, and SeqID is used to specify ordering for up to 32 packets from a particular source to a particular destination. The count field is used to indicate either how many words should be read, or in some cases which bytes should be read from a specific address range. Finally, the address field is used to access memory that is mapped to either main memory or HT-connected devices. An extended HT packet can be used that builds on this format to specify 64-bit addresses [75]. It should also be noted that HT responses do not include address fields and instead use the SrcTag field to route responses to the correct device.

Command and data packets are transmitted between HT devices using a credit-based flow control, as shown in Figure 14, where one credit matches up with one physical buffer for either a command or data packet. Credits and changes in the link status are transmitted using special HT no-operation (NOP) packets that do not convey any memory or I/O-related data. Due to the usage of HT as an I/O protocol that requires deadlock-free messaging, the specification defines an ordering protocol that allows for a semi-relaxed ordering between packets in different virtual channels. The exception is for non-posted packets that are defined to be part of a ordered sequence. Packet transmission must then maintain ordering for this sequence's packets from source to destination. However, transmissions from different sources or destinations do not have any dependencies on ordering and may be reordered at will. This HT ordering requirement is relevant to encapsulation protocols that use HT (such as the HToE protocol) in that packets from the same source node must following HyperTransport's ordering constraints for local transmission, but packets from different sources can be reordered with respect to each other. The proposed receiver-based optimizations in Ch. 6 take advantage of this feature.



Posted Request Counter (CMD)
Posted Request Counter (Data)
Non-Posted Request Counter (CMD)
Non-Posted Request Counter (Data)
Response Counter (CMD)
Response Counter (Data)
Transmit / Receive Control Logic
TX FIFO
HT Link
RX FIFO
Posted Req (CMD)
Posted Req (Data)
Non-Posted Req (CMD)
Non-Posted Req (Data)
Response (CMD)
Response (Data)
CNTR
Flow Control Update Packets (NOP)

Figure 14: HT Link and Credit Buffers

4.4 *HyperTransport over Ethernet Reference Design*

HyperTransport was selected as the on-board interconnect to support the DPGAS model, but at the time of the initial work with the DPGAS model there were few viable options for connecting server blades with HyperTransport interconnects, mainly due to a lack of standardized cabling. In addition, the use of the DPGAS model for data center applications necessitated interoperability with existing data center infrastructure. For this reason,

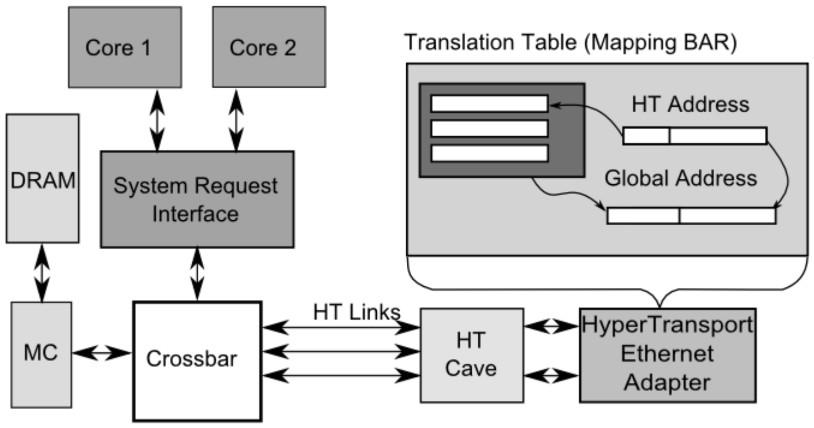


Figure 15: HToE Block Diagram

1 Gbps Ethernet was chosen as the inter-node interconnect, and experimentation with DPGAS was designed based around the encapsulation of HyperTransport packets into Layer 2 (L2) Ethernet frames, that is raw Ethernet packets without any additional layering support from the OS, such as TCP/IP. This encapsulation protocol was referred to as HyperTransport over Ethernet, or HToE.

Initial requirements for the implementation of an HToE network adapter, or HyperTransport Ethernet Adapter (HTEA), were defined as follows: 1) The adapter should support encapsulation and transmission of HT packets at a fast enough rate to meet the 1 Gbps transmission rate of an Ethernet NIC. 2) Packets should be properly routed on remote nodes without needing to modify the local HyperTransport protocol or physical links. 3) Packets from multiple remote nodes should be handled within one pipelined adapter. 4) Remote accesses to memory use the DPGAS model's addressing scheme where N bits are used to specify the remote node (and match to a remote Ethernet address) while B - N bits are used to address memory locations within the node.

The HToE implementation, as shown in Figure 15, is based on a system with AMD Opteron nodes where each node has an Ethernet-enabled FPGA card available in the HTX connector slot, similar to the HTX card built by the University of Heidelberg [23]. Several nodes are connected via an inexpensive Ethernet switch, and it is assumed that HyperTransport messages sent to remote addresses via the HTEA are routed using one of two methods:

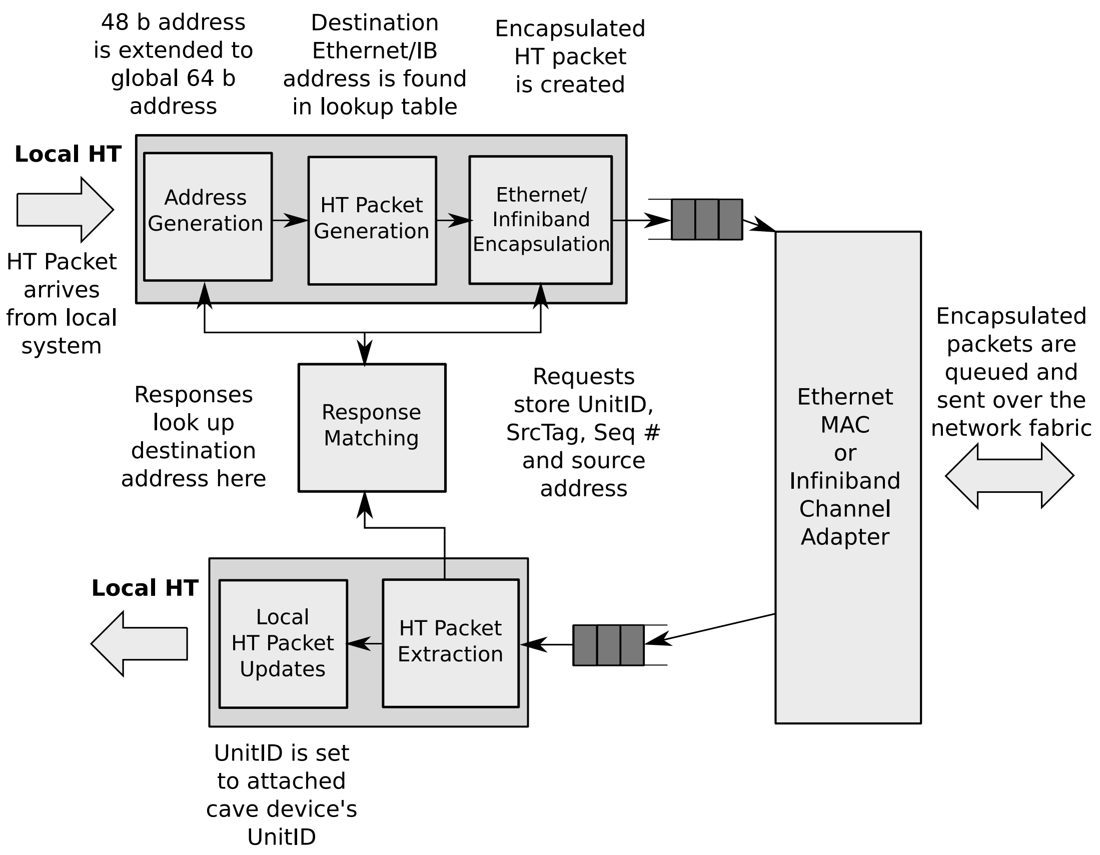

48 b address is extended to global 64 b address
Destination Ethernet/IB address is found in lookup table
Encapsulated HT packet is created
Local HT
Address Generation
HT Packet Generation
Ethernet/ Infiniband Encapsulation
HT Packet arrives from local system
Responses look up destination address here
Response Matching
Requests store UnitID, SrcTag, Seq # and source address
Ethernet MAC or Infiniband Channel Adapter
Encapsulated packets are queued and sent over the network fabric
Local HT
Local HT Packet Updates
HT Packet Extraction
UnitID is set to attached cave device's UnitID

Figure 16: HToE/HToIB Two-Node Packet Send and Receive

1) access to the northbridge address mapping tables (via the BIOS) in order to specify the physical address space mappings for the HTEA device, or 2) an intelligent MMU that distinguishes between accesses to the local memory and the I/O address space and HT packets that are sent for non-local addresses through the HTEA.

For a properly initialized system, there are three stages in each individual communication operation (e.g., a read request command) at a given source host and attached devices: 1) extension from the 40-bit physical address used by Opteron CPUs to a 64-bit physical address, 2) creation of a HT packet that includes a 64-bit extended address, and 3) mapping the most significant 24 bits of the destination address to a 48-bit MAC address and encapsulation of the HT packet into an Ethernet frame. These steps are shown in detail for the two-node case in Figure 16. InfiniBand encapsulation (known as HToIB) is also shown here but is not discussed in detail due to its similarities with Ethernet encapsulation.

An efficient implementation could further pipeline the stages to minimize latency, but retaining the three stages has the following advantages: 1) It separates the issue of current processor core addressing limitations from the rest of the system, which offers a clean, global shared address space, thus allowing implementations with other true 64-bit processors. 2) It is easy to port to other platforms that do not encapsulate by using Ethernet frames, but use other link layer formats such as InfiniBand (as shown in Figure 16). Thus, some efficiency was sacrificed for initial ease of implementation and for a cleaner, modular design.

For outgoing request packets in the HTEA, the two most significant bits of the 40-bit address are decoded to select one of four partition registers to access the 24-bit partition address—the two most significant bits in the 40-bit address used to address the partition register are reset in parallel with the access to the partition register. Next, three pieces of information are needed: 1) the extended 24-bit address to form an HT read request packet with extended address, 2) the MAC address of the destination HTEA to encapsulate the extended HT packet into Ethernet, and 3) the local MAC address, which is used according to Ethernet frame format to enable the response. Item 3 has been set during initialization, so access to the source MAC address is not in the critical path. Items 1 and 2 have a direct correspondence among them—given a destination node ID or the remote partition address, there is a unique MAC address associated with both data fields. Therefore, the partition register can store both the 24-bit partition address and the destination MAC address together, thus reducing access time when forming the Ethernet frame. Once the remote MAC address and the 64-bit address have been found in the partition table, the new HT packet is constructed and encapsulated in a standard Ethernet packet, illustrated in Figure 16 as the Ethernet encapsulation module. The encapsulated packet is then buffered until it can be sent using the local node's Ethernet MAC and the physical Ethernet interface. For packets that send a set amount of data, the control and data packets must be buffered until all the data has been encapsulated into Ethernet frames.

The receive behavior of the HTEA on the remote node requires a "response matching" table where it stores, for every non-posted HT request (request that requires a response), all the information required to route the response back to the source when it arrives. This table is required since HT is strictly a local interconnect and response packets have no notion of a destination 40-bit (or extended 64-bit) address. Since the formats of HT request and response packets differ and the implementation cannot change the local HT link's operation, the SrcTag field of each packet is used to match MAC addresses from an incoming request packet with an outgoing response packet. Note that each request packet contains the source MAC address, and this is the address that is stored in the "response matching" table and that is later used as the destination MAC address for the corresponding response. Encapsulation and buffering occur once again until the response and data can be transmitted over Ethernet. For this reason, the structure that does response matching is shared between incoming and outgoing data paths in the HTEA.

Also, it should be noted that because HT SrcTags are five bits, a maximum of 32 outstanding requests can be handled concurrently by the response matching structure. If two request packets arrive with the same SrcTag (e.g., two read requests from different nodes in the cluster), then the latter packet is remapped before being stored in the table. When the corresponding response leaves the HTEA, the SrcTag is mapped back to its original value to ensure proper HT routing on the requesting local node. This "tag remapping" optimization and an associated UnitID clumping optimization derived from this thesis is discussed in greater detail as it relates to the HyperTransport over Ethernet specification in Section 5.4.2.

Once a response reaches the local HTEA that initiated a read request, the HT packet is removed from its Ethernet encapsulation. The UnitID is changed again to that of the local host bridge and the bridge bit is set to send the packet upstream. This allows the local host bridge to route responses to the originating HT device. Other transactions, such as a posted write or a non-posted write, involve similar sequences of events. The differences in these transactions are that for posted writes, no data is stored to create a response; for non-posted writes, only a “TargetDone” response is returned, and no data needs to be buffered before the response is sent over Ethernet. Similarly, atomic Read Modify Write commands can be treated as non-posted write commands for the purposes of this model.

4.5 HTEA - Hardware Implementation

The hardware implementation of a network adapter for HToE, or HTEA, was designed using Verilog and was built to interface with standard Xilinx IP cores for 1 Gbps Ethernet and with a custom non-coherent HyperTransport “cave” (endpoint) device designed by the University of Heidelberg [146].

Xilinx’s ISE tool was used to synthesize, map, and place and route the HToE Verilog design for a Virtex 4 FX140 FPGA. Synthesis tests using Xilinx software indicated that the four modules that make up the HTEA were individually capable of speeds in excess of 160 MHz—combined, unoptimized results showed that the HTEA was more than capable of feeding a 1 Gbps or faster Ethernet adapter with a 125 MHz (1 Gbps) clock speed. Evaluations for each of the request and reply critical paths demonstrated that the latency overhead of this implementation is on the order of 24 to 72 ns (for a control packet with no data and a read request response with eight double words of data, respectively). In a Xilinx Virtex 4 FX140 FPGA, an unoptimized placement of the bridge used approximately 1,300 to 1,500 slices, or approximately 5% to 6% of the chip.

Overheads that reduced performance included the use of a serial Gigabit Ethernet MAC interface and the use of only one pipeline to handle packets for each of the three available virtual channels. These latency results are listed in Table 1 along with the associated latency of the Heidelberg cave device from [146]. Total read and write latencies include latency components that are discussed in relation to Table 2. All bridge operations assume a 125 MHz clock and discount any serialization latency normally associated with Xilinx Ethernet MAC interfaces.

Table 1: Latency Results for HTEA (Two-Node System)

DPGAS operation	**Latency (ns)**
Heidelberg HT Core (input)	55
Heidelberg HT Core (output)	35
HTEA Read (no data)	24
HTEA Response (8 B data)	32
HTEA Write (8 B data)	32
Total Read (64 B)	1692
Total Write (8 B)	944

4.5.1 Memory Subsystem Latency Performance Penalties

The transfer of data between two nodes with the HTEA is low-latency at one to two μs, but it is also important to understand the overall latency penalty that the memory subsystem contributes to remote memory accesses. The latency statistics for the HTEA and related Ethernet and memory subsystem components were obtained from other studies [31] [146] [81] and from the above place and route timing statistics for the HTEA reference implementation. An overview is presented in Table 2. This HToE implementation was based on a 1 Gbps Ethernet MAC included with the Virtex 4 FPGA, but latency numbers were only available for a 10 Gbps Ethernet IP core, which is shown below.

Table 2: Latency Numbers Used for Evaluation of Performance Penalties (Offline)

Interconnect	**Latency (ns)**
AMD Northbridge	40
CPU to on-chip memory	60
Heidelberg HT Cave Device	35 - 55
HTEA	24 - 72
10 Gbps Ethernet MAC	500
10 Gbps Ethernet Switch	200

Using the values from Table 1 for a request to remote memory, the performance penalty of remote memory access can be calculated using the formula:

$$\mathrm{t}_{rem\_req} = \mathrm{t}_{northbridge} + \mathrm{t}_{HTEA} + \mathrm{t}_{MAC} + \mathrm{t}_{transmit}$$

where the remote request latency is equal to the time for an AMD northbridge request to

DRAM, the HTEA latency (including the Heidelberg HT interface core latency), and the Ethernet MAC encapsulation and transmission latency. This general form can be used to determine the latency of a read request that receives a response:

$$t_{rem\_read\_req} = 2*(t_{HToE\_req} + t_{HToE\_resp} + t_{MAC} + t_{transmit}) + t_{northbridge} + t_{rem\_mem\_access}$$

These latency penalties compare favorably to other technologies, including the cut-through latency for the 10 Gbps Ethernet switch, which is close to 200 ns [130]; the fastest MPI latency, which is 1.2 μs [103]; and disk latency, which is on the order of 6 to 13 ms for hard drives, such as those in one of the server configurations used in Section 4.6 to evaluate resource sharing with HToE [148]. Additionally, this unoptimized version of the HTEA is fast enough to feed a 1 Gbps Ethernet MAC without any delay due to the encapsulation of packets. As Chapter 6 shows, improvements for a 10 or 40 Gbps-compatible version of the HTEA can include optimizations such as multiple pipelines and buffering of packets destined for the same destination in order to reduce the overhead of Ethernet encapsulation.

These results indicate that an HToE-enabled cluster would be a viable means for implementing the DPGAS model due to its low-latency data path. However, Sections 4.6 and 4.7 also consider HToE's impact on total system power since additional accesses to remote memory can lead to slower overall execution and can consume more static power from other system components.

4.6 Reducing Overprovisioning with DPGAS - Offline Analysis

In the absence of a full hardware testbed, a trace-driven analysis was employed to determine potential power and cost savings offered by the DPGAS model [176]. Virtual address traces were acquired using an instrumented SIMICS model [100] and fed through a page table simulator to determine the number of page faults as a function of physical memory footprints ranging from 32 MB to 1 GB. Five benchmarks were selected: Spec CPU 2006's MCF, MILC, and LBM [63]; the HPCS SSCA graph benchmark [7]; and the DIS Transitive Closure benchmark [39]. These benchmarks had maximum memory footprints ranging

Table 3: HP Proliant Server Configurations

Model	CPU Cores	Max. Memory	Base Cost/Power
HP DL785 G5	eight quad-core, 2.4 GHz Opterons	512 GB	∼\$42,000/1110 W
HP DL165 G5	two quad-core, 2.1 GHz Opterons	64 GB	∼\$2,000/197 W

from 275 MB to 1600 MB. A 2.1 billion address trace (with 100 million addresses to warm the page table) was sampled from memory intensive program regions of each benchmark. These traces were used in conjunction with application and system models to assess potential power and cost savings when using DPGAS.

4.6.1 Experimental Setup

Two HP Proliant server configurations were selected as models for the offline analysis, where the DL785 represented a high-end server with 32 threads and 512 GB of DRAM, and the DL164 represented a low-end server with eight threads and 64 GB of DRAM (Table 3). Both configurations were expected to execute at least two VMs per core where a VM instance was modeled as a benchmark application trace. All associated system and memory costs and power statistics were derived from the HP Power Calculator [71] and online data sheets [69], respectively.

Three different scenarios were investigated: 1) Using a fixed workload of VMs (benchmarks), the DPGAS model was applied to reduce the "scale out" factor of data centers. 2) Using memory throttling to reduce the power consumption from servers that have been overprovisioned in terms of memory to handle added applications. 3) Using the DPGAS model to try and most efficiently map a set of applications to a fixed, global memory footprint in the data center. These experiments were chosen as representative strategies that might be used in a data center to handle large time-varying workloads but that would likely lead to memory overprovisioning and higher power usage and cost when compared with using a DPGAS-based approach.

4.6.1.1 Reducing Scale Out for a Fixed Workload

Scale out refers to a typical data center method for increasing application capacity, which is to add more machines or bring idle machines online. This experiment started with a scaled-out data center configuration of 250 servers, which consolidated a static number of applications onto 225 servers and then onto 200 servers. The high- and low-end server configurations both contained a certain number of applications (as recommended by an Intel virtualization study [25]) that was then extrapolated to a data center with 250 servers (translating to 2,000 or 500 processor sockets for the high- and low-end server configurations). Using these statistics, this data center could then support either 19,500 applications using high-end servers or 4,700 applications using low-end servers.

In this set of experiments, the baseline memory allocation started out with enough memory to comfortably support the maximum memory footprint for every application allocated, while a simple DPGAS-based allocation algorithm (spill/receive) took into account the amount of unallocated free memory and decreased the provisioned amount of memory on half of the servers by that amount. This reduction in memory is analogous to designing a data center with half of the servers overprovisioned (receive nodes in the DPGAS model) and half of the servers minimally provisioned (spill nodes).

4.6.1.2 Memory Throttling

When system designers overprovision servers, much of the installed memory sits idle on each node. This experiment investigated the the power, cost, and performance implications of memory throttling—that is, reducing the allocation for each application below its desired level, resulting in additional cost and power savings at the expense of performance, i.e., page faults. This technique was used with the spill/receive model to see how DPGAS could be combined with memory throttling to create a minimally provisioned data center with optimal memory allocation.

Two strategies were tested with fixed applications to compare with an initial application allocation on 250 servers: 1) Each server had 50% of the desired memory and each application was given 50% of its maximum memory footprint. 2) Each server had 25% as much memory, and each application received 25% of its max footprint. These scenarios were used to simulate the effects of memory throttling where an application might suffer from performance loss due to lack of memory and memory fragmentation in the date center.

4.6.1.3 Fixed Memory Analysis

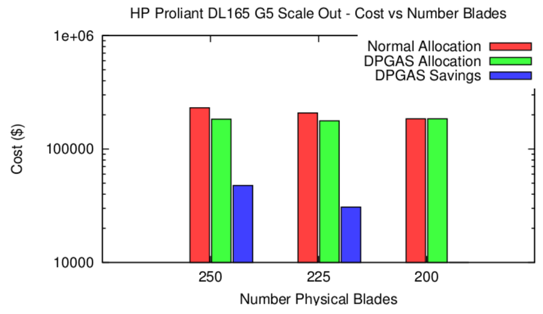
This experiment investigated the effects of using DPGAS's spill/receive model for allocation of a random number of applications on servers with a fixed amount of memory. Applications were allocated until 250 servers had no available memory for new applications using both baseline (no spill/receive or memory sharing) and DPGAS allocation algorithms.

4.6.2 Offline Analysis - Results

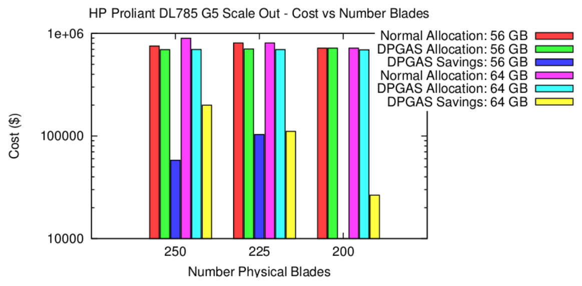
Results from three sets of experiments are shown here, based on a memory allocation simulation of random application workloads using both baseline (no sharing) and DPGAS (spill/receive) memory allocation. Results for all experiments were averaged over 50 iterations.

4.6.2.1 Reducing Scale Out for a Fixed Workload

The total cost savings for the low- and high-end server configurations are shown in Figures 17a and 17b with savings between normal and DPGAS allocation graphed as the third column of each group. In the baseline (250-server) case, DPGAS has the potential to save 22% to 26% in memory cost, which translates into a $60,000 savings for the low-end servers and $200,000 for the high-end servers. However, it is also important to notice that savings with DPGAS allocation drop as applications are consolidated onto fewer servers. This is likely due to the fact that there is less free memory unallocated when using normal allocation, and fewer nodes have memory left to act as receiving nodes for remote spilled allocations.



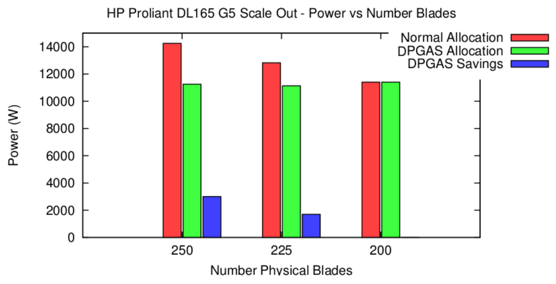
(a) Proliant DL165 G5



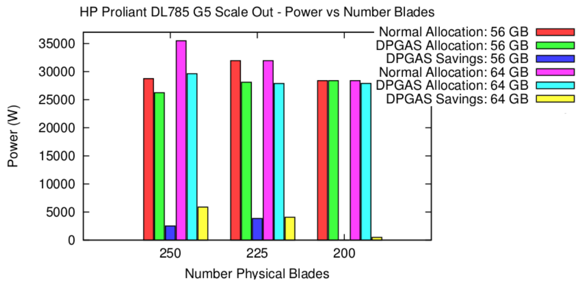
(b) Proliant DL785 G5

Figure 17: Scale Out Cost

Similarly, the power savings using DPGAS allocation (Figures 18a and 18b) is substantial in the base case, with a savings of 3,625 (25%) and 5,875 (22%) watts of input power for the low-end and high-end server configurations, respectively. When server consolidation onto 200 servers is used, this power savings drops to 800 and 500 watts for the same configurations. Both the cost and power results indicate that DPGAS memory allocation is best suited for an environment that has a reasonable amount of fragmented memory that can be utilized and for workloads that have a high amount of variability in their memory footprints.



(a) Proliant DL165 G5

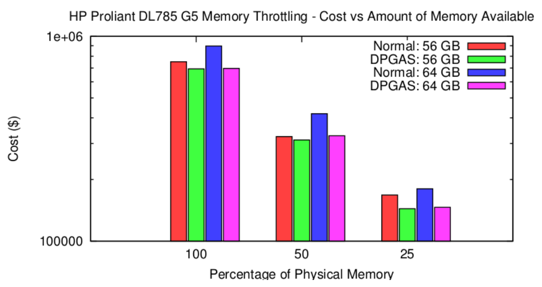


(b) Proliant DL785 G5

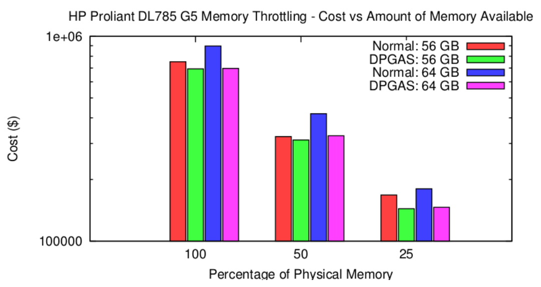
Figure 18: Scale Out Power

4.6.2.2 *Memory Throttling*

The results for cost and power in the high-end server are shown in Figures 19a and 19b. The effects of memory throttling are dramatic, and even when less memory is allocated to a server, DPGAS can be used to improve memory cost and power efficiency. For instance, reducing memory from 64 GB to 32 GB in each server reduces memory cost by \$478,000 and memory power by 17,750 watts (from a base cost of \$897,000 and base power of 35,500 watts). The usage of DPGAS allocation with 50% memory throttling with the high-end server configuration can reduce the total memory cost by \$570,000 and total memory power by 21,125 watts.



(a) Proliant DL785 G5 (Cost)



(b) Proliant DL785 G5 (Power)

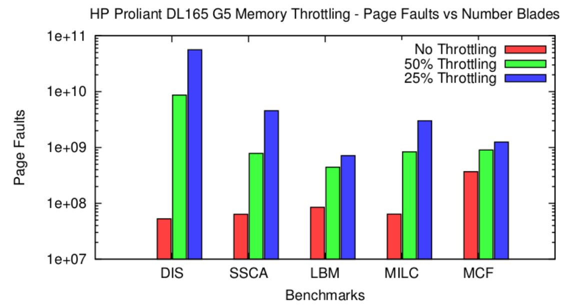
Figure 19: Memory Throttling Effects on Power and Cost, High-end Server

Additional statistics for the low-end server configuration are shown in Table 4. These experimental results concur with the high-end server configuration, except that power and cost savings are smaller due to less memory fragmentation and less memory overall for remote sharing. Overall, DPGAS enables a 10% to 22% reduction in memory cost and a 16% to 25% reduction in memory power when compared to normal allocation for the low- and high-end servers.

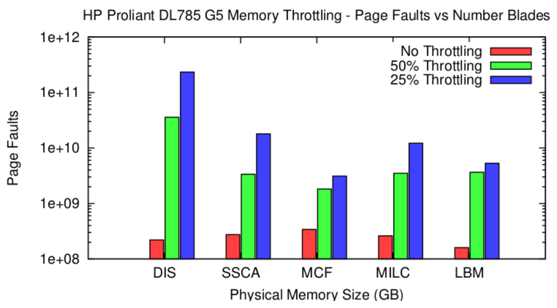
Table 4: Low-end Server Cost and Power with Memory Throttling

Allocation	No Throttling	50% Throttling	25% Throttling
Normal (\$)	\$230,750	\$111,250	\$51,500
DPGAS (\$)	\$171,000	\$93,000	\$46,250
Normal (W)	14,250	7,000	3,500
DPGAS (W)	10,625	5,875	2,625

When using memory throttling, performance must also be taken into account. The results from trace-driven analysis of the benchmark applications provide data on page fault rates that directly correspond to the amount of memory a benchmark is given. These results



(a) Proliant DL165 G5



(b) Proliant DL785 G5

Figure 20: Memory Throttling Performance

were used to generate Figures 20a and 20b that demonstrate the effects of memory throttling on random allocations of each of the benchmark applications. In general, the usage of memory throttling leads to an order-of magnitude increase in the number of page faults for all applications, but some applications with small memory footprints or random access patterns (poor spatial reuse) are affected much more by using memory throttling with normal allocation. However, additional results discussed in [171] indicate that DPGAS can also be used to improve the performance of applications with throttled memory footprints. If all servers are provisioned with an equal amount of throttled memory (i.e., no cost and power savings), the DPGAS model can be used to give unallocated memory to applications that would benefit the most from additional memory. By trading some of the power and cost savings, page faults can be reduced system-wide while still garnering the benefits of memory throttling.

4.6.2.3 Fixed Memory Analysis

Table 5 shows the results of random allocations for a fixed memory size. In both the low- and high-end cases, DPGAS allocation of memory for an application failed one to two orders of magnitude fewer times, due to the use of spill/receive allocations that allowed the use of remote memory sharing. Additionally, fixed memory experiments showed that DPGAS can support a small number of additional applications over standard allocation (results not shown in table), typically on the order of 30 to 40 more than a baseline allocation.

Table 5: Offline Analysis - Fixed Memory Workloads

Server Config	Alloc. Method	Workloads Allocated	Alloc. Failures
Proliant 785	Normal	24,875	133
Proliant 785	DPGAS	24,875	3
Proliant 165	Normal	6,163	33
Proliant 165	DPGAS	6,163	2

4.6.3 Discussion

For these three experiments, the offline analysis indicates that using the DPGAS model (with a suitable hardware implementation like HToE) can help to reduce the effects of memory fragmentation and reduce overall cost and power for servers when combined with other techniques like application/VM compaction (reduced scale out) or with memory throttling.

4.7 Reducing Overprovisioning with DPGAS - Online Analysis

Initial results from offline analysis indicate the suitability of the DPGAS model and the HTEA for high-performance data movement and subsequently for providing a temporary remote "spill" buffer to time-varying workloads with large memory footprints [176]. This section discusses further work that used a trace-based simulation to provide an online analysis of HToE performance in the context of a more complete system model for a 16 node cluster [173].

4.7.1 Online Simulation Infrastructure

Figure 21 shows an illustration of the overall simulation environment, which takes advantage of the open-source NS-3 network simulator project [117] and University of Maryland's DRAMSIM [161]. Scalable simulation is supported by the event-driven scheduling classes used within NS-3 as well as updates that allow for distributed (MPI-based) simulations. Detailed models in DRAMSIM and NS-3 allow the modeling of power and latency used by DDR2 DRAM as well as per-hop latencies associated with different topologies of switched Ethernet.

In addition, custom code was created to provide a detailed model of the HTEA and the

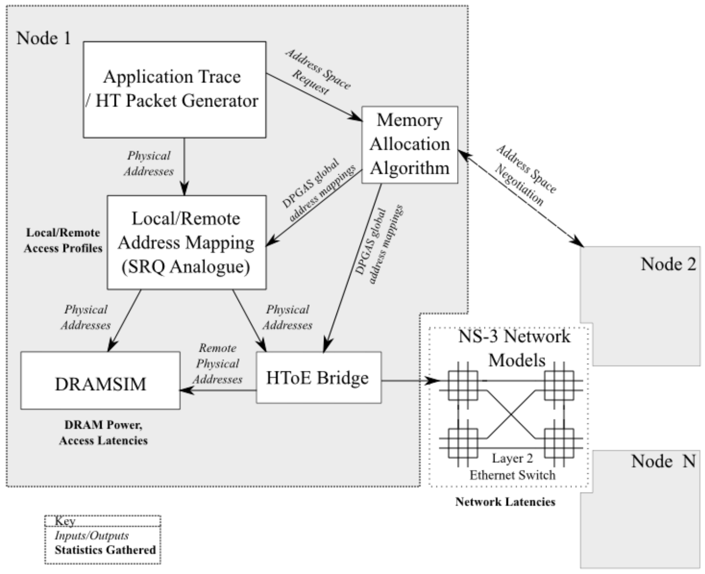

Node 1
Application Trace / HT Packet Generator
Address Space Request
Memory Allocation Algorithm
Address Space Negotiation
Physical Addresses
DPGAS global address mappings
Local/Remote Address Mapping (SRQ Analogue)
Local/Remote Access Profiles
DPGAS global address mappings
Node 2
Physical Addresses
Physical Addresses
NS-3 Network Models
Remote Physical Addresses
DRAMSIM
HToE Bridge
DRAM Power, Access Latencies
Layer 2 Ethernet Switch
Node N
Network Latencies
Key
Inputs/Outputs
Statistics Gathered

Figure 21: HToE-based Simulation Infrastructure

Opteron Northbridge's System Request Queue (SRQ) that can be used to map physical addresses to local or remote DRAM [4] [31]. The SRQ is programmed to determine whether a physical address is mapped to the local DRAM and memory controller or to another device attached to the crossbar in the northbridge. By using the address mapping tables, a certain subset of physical addresses can be mapped to the HTEA and are thus satisfied by remote memory controllers. This infrastructure uses a C++ mapping function that simulates the operation and address range updates in the Opteron (SRQ).

4.7.2 Experimental Setup

The NS-3 and DRAMSIM-based simulation was used to evaluate four key metrics: 1) The effect of using DPGAS spill/receive on DRAM latency, 2) the power impact of using DPGAS in conjunction with average-case memory provisioning, 3) a more fine-grained analysis of packet latency with DPGAS, and 4) the impact of using spill/receive on overall network bandwidth utilization for a shared 10 Gbps Ethernet switch.

These experiments tracked several different metrics including: 1) dynamic power for DRAM (W), 2) memory access latency for DRAM (ns), 3) link and buffering latency for the 10 Gbps network (ns), and 4) network utilization of the 10 Gbps network (B/s).

4.7.2.1 Application Workloads

Synthetic traces were generated based on application access patterns and inter-reference timing from other studies that investigated benchmark suites such as Spec 2006 [84]. These studies were used to specify the DRAM access inter-reference time for synthetic benchmarks. The Spec 2006 benchmark represents applications that are worst-case application workloads in terms of L2 cache misses, and some of the most intensive of these applications, such as MCF and Omnetpp, have an L2 cache miss every 4000-4500 cycles for a 2 MB L2 cache size. Enterprise applications and general-purpose applications likely have better spatial and temporal locality than the SPEC suite, so these experiments use an inter-reference time ranging from 10,000 to 20,000 cycles between L2 cache misses.

Three different synthetic benchmarks were used to represent different access patterns: 1) Random - a worst case scenario where there is no spatial locality within an address stream; 2) Clustered Random - accesses are clustered around a mean address, and locality is similar to an application repeatedly accessing several pages in the application's address space; 3) Strided - the access stream is specified by a stride and is related to applications that perform large numbers of sequential array element accesses. Application footprints were randomly assigned from 500-1500 MB, although specific tests were designed with larger memory footprints to have one or more applications that would require spilling to a remote node via DPGAS. The sizing of these memory footprints was drawn from insights found in [25]. The number of read and write operations was split equally between reads and writes.

4.7.2.2 Simulation Setup

DRAMSIM was initialized using a 4 GB module that corresponds to Micron's MT47H512M8 TwinDie 4 GB DDR2 module [107]. Timing and power parameters were selected for the DDR2-800 part, and one and four ranks were used to simulate 4 and 16 GB of installed memory, respectively. The associated CPU speed was set at 3000 MHz.

Table 6: Online Simulation - DPGAS Application and Memory Parameters

Num Nodes	Apps / node	Spill Nodes	Receive Nodes	Mem Size (GB)	Apps / Receive Node
1	4,16	0	0	4,16	N/A
2	6	0	0	4/8	2
2	6,16	1	1	4,16	2,8
4	6,16	2	2	4,16	2,8
8	6,16	4	4	4,16	2,8
16	16	0	0	16/20	8
16	6,16	8	8	4,16	2,8
8	6,16	6	2	4,16 (8 for Recv)	2,8
16	6,16	14	2	4,16 (8 for Recv)	2,8

Table 6 shows the different test cases for the online simulation. NS-3 was initialized with between one and 16 nodes, with each node connected to a single switch via a 10 Gbps channel. NS-3's's raw socket layer was used to represent the transfer of HT packets inside Ethernet packets. Simulations were run for 100 real-world seconds, and each node had between one and 16 applications that each generated accesses for 500,000 synthetic addresses. For applications with a miss rate of every 20,000 cycles, this would mean that the synthetic applications would run for about three seconds of wall-clock time. Tests with 4 GB were run with six applications per "spill" node and two applications per "receive" node, while tests with 16 GB of memory were run with at least 16 applications for the "spilling" node and eight applications for the "receiving" node. In each test a certain number of applications were specified to spill to other nodes, usually one to three applications that exceeded the amount of memory for a minimally provisioned blade. This setup assumed that an overprovisioned blade would have 8 GB and 20 GB of memory, while this simulation used 4 GB and 16 GB, respectively.

Two tests were also devised to test scenarios similar to those tested by proponents of memory blades [96] where a greater number of "client" servers could access remote memory on one central blade to support peak workloads. The first simulation used eight nodes, with six nodes containing 4 GB of installed memory spilling to two nodes with 8 GB of DRAM installed. The second test used 16 nodes with fourteen spill nodes and two receive nodes with 4 and 8 GB of installed DRAM, respectively.

4.7.3 DPGAS's Impact on Memory Latency

Table 7 shows the average memory access latency across all simulations. The use of DPGAS spilling does not dramatically change the average latency for a memory access, although it can cause a slight increase in average latency of a lightly loaded node that receives remote memory requests. Changes in access patterns from DPGAS spill operations and increases in the number of applications also affect the number of conflicts on open rows in the simulated DRAM, requiring additional latency to access a new row. In most simulations, access latencies varied between 60 and 70 nanoseconds. Variations between nodes are expressed in the standard deviation calculations in Table 7.

Table 7: DRAM Access Latency with DPGAS Spill/Receive Algorithm

Simulation, DRAM Size	**Ave. Mem Latency (ns)**	**Std. Dev.**
2 node, 4/8GB (no DPGAS)	54.28	6.21
2 node, 4GB	53.06	2.5
4 node, 4GB	69.42	5.58
8 node, 4GB	66.29	9.5
16 node, 4GB	64.35	11.5
8 node, 4GB,8GB	67.74	10.89
16 node, 4GB,8GB	69.98	10.65
2 node, 16GB	68.11	12.51
4 node, 16GB	68.27	13.28
8 node, 16GB	68.72	14.21
16 node, 16/20GB (no DPGAS)	68.17	9.11
16 node, 16GB	68.84	7.2

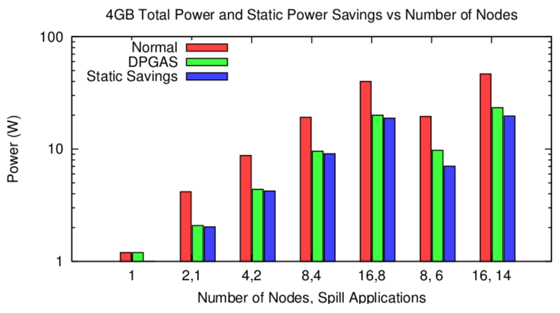
Further tests showed that the latency difference for DPGAS versus non-DPGAS simulations (two nodes with 4/8 GB and 16 nodes with 16/20 GB) led to an increase (16 node case) or decrease (two node case) of less than 2 ns and a smaller standard deviation. While the amount of traffic to the "receiving" DRAM increased, this DRAM was already underutilized due to lighter application loads. The decrease in standard deviation reflects that the average latency of memory accesses on the “spill nodes” was reduced as remote accesses took place.

4.7.4 DPGAS’s Impact on Power Usage

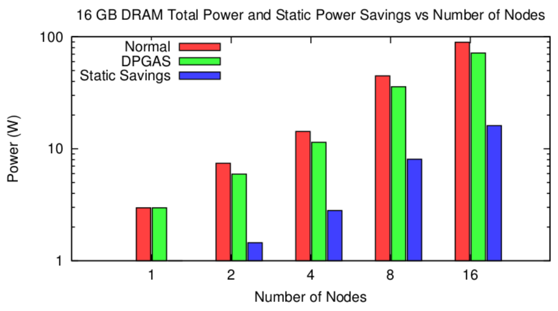
DRAMSIM tracks the average power for one activation-to-activation cycle, meaning that in this simulation it was used to track the power needed to perform one operation, whether it be a read, write, precharge, or refresh operation. The calculated power was then averaged across all the accesses that take place in the memory simulation to get a power value for each type of operation the DRAM performs. Due to overprovisioning, DIMMs that are underutilized by application workload must still be refreshed, and they still consume static leakage energy.

This static component of DRAM power includes the power used when a DIMM is in power-down or standby states for either active (when data is stored in the sense amplifiers) or precharge (all data is restored to a row) modes. Additionally, static power takes into account the power for refresh cycles that occur at regular intervals every 64 ms (for the selected model DIMM). Figures 22a and 22b show the average static power savings achieved by reducing the DRAM in each node by one DIMM. For the 4 GB case, power savings are calculated based on provisioning each DPGAS-enabled node with 4 GB instead of an 8 GB overprovisioned server, and for the 16 GB case, power savings are calculated in relation to a 20 GB overprovisioned server. In the “memory blade” case, there are two receive nodes with an extra 4 GB of memory, leading to slightly higher total power usage.

The power savings are illustrated with respect to the total power (including power for activations, read, write, and termination commands) to give a proper sense of the total power consumed versus the static power that can be saved by provisioning fewer DIMMs. For the 4 GB case, each server receives half as much memory, so power savings are between two and 19 Watts or 47 to 49% for the normal spill/receive tests. For the 16 GB case, the overall percentage of savings is lower since overprovisioning means adding only one more



(a) Low-end Servers (4 GB DRAM)



(b) High-end Servers (16 GB DRAM)

Figure 22: Total Power and Power Savings with DPGAS Model

similar (4 GB) DIMM rather than adding multiple DIMMs. The savings for the 16 GB experiment are between 1.5 and 16 Watts or 18% to 20%.

The memory blade experiments show similar savings for high-load environments. Compared to the medium-load experiments (with an equal number of spill/receive nodes), the memory blade tests reduce static power by 36% and 42%, compared to 47% for the medium-load simulations.

It should also be noted that removing DIMMs from the system can increase the overall system power if this reduction increases the system utilization. The HP Power Advisor Calculator [72] was used to build a 10,000 core data center with similar DRAM characteristics to those used in experiments based on the Proliant DL160 G6 server blade running at 60% utilization. For the 16 GB case, removing 4 GB of DRAM from half of the blades would result in a power savings of 3,540 Watts or 2.8%. However, if removing DIMMs increased utilization (CPU and memory) by 5%, the power cost would *increase* by 3,040 Watts or 2.5%. It is unlikely that the DPGAS-based approach would increase CPU utilization because modern operating systems are already geared toward hiding the latency of accesses to storage devices like DRAM and hard disks, and previous research indicates that most servers are somewhat underutilized [11].

4.7.5 DPGAS's Impact on Packet Latency

To find a more refined value for the latency of a remote DIMM access, data was gathered to model the "static" components that were not measured with the NS-3 simulation. The latency values for the related Ethernet and memory subsystem components were obtained from statistics from other studies [31] [146] [81] and from the place and route timing statistics described in Section 4.5. An overview is presented along with measured results for packet latency from the online simulation in Table 8. The average of an HT packet being encapsulated by the HTEA is again based on an average processing time of six cycles at 125 MHz.

Table 8: Latency Numbers Used for Evaluation of Performance Penalties (Online)

Component	**Latency (ns)**
AMD Northbridge	40
On-chip memory access	60
Heidelberg HT Cave Device	45
HTEA	48
10 Gbps Ethernet MAC	500
10 Gbps Ethernet Switch	200
Average Component Delay	893
Measured Transmission and Buffering Delay (NS-3)	185 - 939

Using the formulas from Section 4.5, the performance penalty of a remote memory access can again be calculated as follows:

$$\mathrm{t}_{rem\_req} = \mathrm{t}_{northbridge} + \mathrm{t}_{HTEA} + \mathrm{t}_{MAC} + \mathrm{t}_{transmit}$$

Figure 23 shows the total average one-way latency using the NS-3 transmission latency values for $\mathrm{t}_{transmit}$ and the average component delay in Table 8 for the other delay values. Note that the 200 ns switching delay [130] is included because NS-3 does not accurately model all the internal components of a data center-capable Ethernet switch. This addition likely overestimates the effect switching delay has on latency.

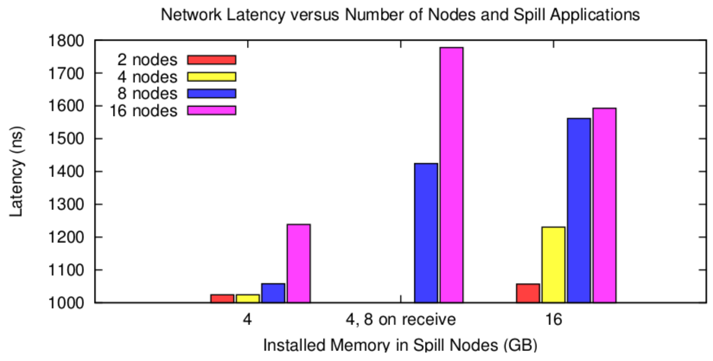


Figure 23: HW Latency for HToE Remote Operation

The one-way latency varies from 1024 ns to 1238 ns for the 4 GB simulations with six synthetic applications on each node and from 1057 ns to 1593 ns for the 16 GB simulations with 16 applications on each node. Also, the addition of more spilling nodes doubles the latency of simulations with eight and 16 nodes to 1478 and 1832 nanoseconds.

Despite the overall increase in latency due to buffering delay, latency still remains low enough that remote, virtualized DRAM can be accessed without causing noticeable delay to applications. The latency penalties again compare favorably to other technologies and earlier offline results. The formula for one-way latency (from Section 4.5) can again be used to determine the total latency of a read request that receives a response:

$$t_{rem\_read\_req} = 2*(t_{HToE\_req} + t_{HToE\_resp} + t_{MAC} + t_{transmit}) + t_{northbridge} + t_{rem\_mem\_access}$$

Using these refined figures and the online simulation, the read latency for a cache line is 2,242 ns.

4.7.6 DPGAS's Impact on Network Utilization

In addition to tracking packet latency, it is also important to analyze the overall impact of using HToE on a shared 10 Gbps Ethernet link. Figure 24 shows the link utilization in MB per second during the simulation time when DPGAS was being used to spill memory requests to remote nodes. Utilization varies from 31.3 MB/s for the two-node, 4 GB test case to 756 MB/s for the heavily loaded test case with 16 nodes and 14 nodes spilling to the two “memory blade” nodes. The 16 GB test cases show similar utilization to the normal 4 GB test cases, with utilization ranging from 31.3 to 250.65 MB/s. Although it would be expected that utilization would be higher for larger numbers of spilling applications, the synthetic benchmarks in the 16 GB tests transferred more data over a much larger interval of time (about four seconds of real-world time versus two seconds in the 4 GB tests).

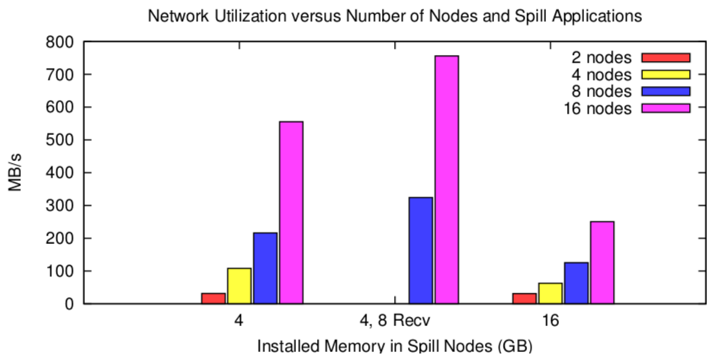


Figure 24: Network Utilization with DPGAS

The addition of more nodes and applications per node increases the utilization, but the 10,000 to 20,000 cycle intervals between memory requests keep packets spaced out for medium-load cases, reducing link utilization. The memory blade tests for 8 and 16 nodes show a dramatic increase in the amount of traffic with utilizations of 555 and 756 MB/s, respectively. For the 16 node case, this translates to a utilization of more than 6 Gbps for the combined traffic of 56 applications sending HToE requests in the same period of time (a relatively heavy workload). This high utilization indicates that large memory blades may need to be placed on a separate 10 Gbps link to prevent delays for other network traffic.

4.7.7 Further Discussion

The online analysis demonstrates a distinct point in time when multiple applications have overlapping requests to spill memory via DPGAS. In real-world situations, multiple servers may experience peak workload at the same time, but this experimental setup, specifically for the memory blade tests, represents a heavy load or worst-case scenario. While these simulations do not currently incorporate a full system performance model, DPGAS accesses should have much better performance than accesses to disk.

Additionally, these experiments did not take into account other existing optimizations for sharing memory, such as page migration. This penalizes the reported results because pages that are most commonly used could be swapped into local memory, reducing the number of remote accesses. On the other hand, the simulations also did not take into account the HT transaction limitations (based on SrcTag and UnitIDs). This omission would increase the remote access latency in heavily loaded situations unless optimizations are designed to address this bottleneck.

4.8 Related Work

Results discussed in this chapter from both offline analysis and online simulations indicate that the DPGAS technique has potential promise as an alternative to virtualization-related strategies and dynamic throttling to reduce overprovisioning and power usage in data centers. Recent work in this area has included three main areas: 1) Dynamic placement and management of applications to allow for throttling or idling of extra servers; 2) Better support for virtualization-related techniques and VM management to reduce power usage; 3) General OS-related and architectural techniques for reducing memory-related power.

Recent work into application and VM placement for power management has started to focus on using multiple resources, including memory and disk as inputs for when to migrate VMs and idle servers [14]. However, much of this research still focuses on memory throttling or memory compression [91], which can negatively impact performance of existing workloads.

Most important in the area of virtualization is improved support for VM migration and strategies such as memory overcommitment (dynamic swapping of hot pages to support VMs with overlapping memory footprints) [65], transparent page sharing for related VMs, VM consolidation, and improved VM balloon drivers [159]. Each of these techniques aims to reduce the overall impact of a VM's memory footprint, which may include a large memory heap but a relatively small and dynamic working set size. Managing the VM footprint in this manner allows for idling of unused servers to conserve power, and the DPGAS model would likely be useful in supplementing existing techniques rather than totally replacing them. DPGAS may provide a performance boost when compared to standard memory overcommitment as it redirects some page faults to remote memory as opposed to disk. In this manner, the "virtual DIMM" concept could be combined with standard VM balloon drivers to coordinate DPGAS allocations as working set sizes for VMs balloon and shrink.

Furthermore, The DPGAS model can also be used to further manage a VM's memory footprint by allowing remote memory accesses to replace less performant techniques, such as VM migration. Despite improvements in live migration techniques (where the VM is still available for use during migration) [112], VM migration is still a heavy-weight operation that takes several seconds [153] and can have downtimes in the range of milliseconds [86]. By providing VMs with low-latency access to remote memory segments, the DPGAS model provides a powerful tool for reducing overprovisioning and increasing the effectiveness of server idling techniques.

4.8.1 Power-based optimizations for DRAM

Other work has also focused on the power implications of overprovisioning DRAM with data movement- or throttling-based solutions. An evaluation of power trends in the data center was conducted in [97], and a design for a memory blade approach to sharing memory, disaggregated memory, was discussed in [96]. The disaggregated memory approach had very similar goals to this research work because it also sought to reduce overprovisioning of DRAM. The main difference between this implementation of disaggregation and the related work is that the DPGAS model focuses on using existing memory that is fragmented (due to uneven workloads in the data center) and the disaggregated memory approach focused on creating additional memory blades that can be shared between multiple nodes.

The approach for resource sharing proposed in this chapter is related to RDMA-based approaches, although DPGAS is focused on reducing registration overhead and supporting more fine-grained access. One company that has approached using RDMA for memory virtualization is RNA Networks (now bought out by Dell), whose Memory Virtualization Platform uses RDMA with InfiniBand or Ethernet to allow high-bandwidth and low-latency memory sharing [137]. This technique also helps to restrict memory power by reducing overprovisioning.

Other higher-level approaches have addressed operating system support and data center level support for reducing DRAM power. At the operating system level, Tolentino [154] suggested a software-driven mechanism using control theory to limit application working set sizes and to reduce the need for DRAM overprovisioning. Additionally, many researchers have proposed the concept of ensemble-level power management [132] to manage power at the server enclosure levels.

4.9 Concluding Remarks

This chapter introduces the hardware that is used to support the Dynamic Partitioned Global Address Space model, the HyperTransport over Ethernet Adapter (HTEA), and it demonstrates an offline and an online analysis of using HToE to reduce DRAM overprovisioning in data centers. It provides a first-order performance analysis for HToE hardware, which is later improved by the definition of the HToE specification (Ch. 5) and detailed timing models used for aggregation-based applications (Ch. 6).

CHAPTER V

THE HYPERTRANSPORT OVER ETHERNET SPECIFICATION

Shortly after the initial research into using HyperTransport over Ethernet to support DP-GAS, the HyperTransport Consortium, or HTC, (and Georgia Tech as a research member of this consortium) decided to pursue a full specification to support HyperTransport over Ethernet for modern data centers. While the initial FPGA implementation of HToE focused on creating a low-latency solution for performing put/get operations, it did not include support for features that are common in modern network adapters, such as support for failure recovery, initialization, and optional security measures. The resulting HyperTransport over Ethernet specification [170] is described here in terms of how it differs from earlier research performed in Chapter 4 and in relation to the detailed simulation that is used in Chapter 6 to investigate an aggregation-related use case with accelerator-based clusters. The format of this chapter is as follows: 1) The motivation for using Ethernet is briefly discussed, 2) the requirements for an HToE specification are then detailed, and 3) the unique features of this specification are demonstrated.

5.1 Logistics

The final HyperTransport over Ethernet specification was approved by the HyperTransport Consortium in September of 2010, and its completion involved contributions from multiple member groups and individuals. This author contributed initial research [176] that discussed concepts that were further refined by the specification, such as the use of a DPGAS-like model for remote memory allocation and Tag Remapping and UnitID clumping in the HTEA (Section 5.4.2). In addition, the two main coauthors (Brian Holden and this author) went through multiple rounds of phone conversations with HTC members from other businesses and from AMD to fully develop recommendations for HToE flow control,

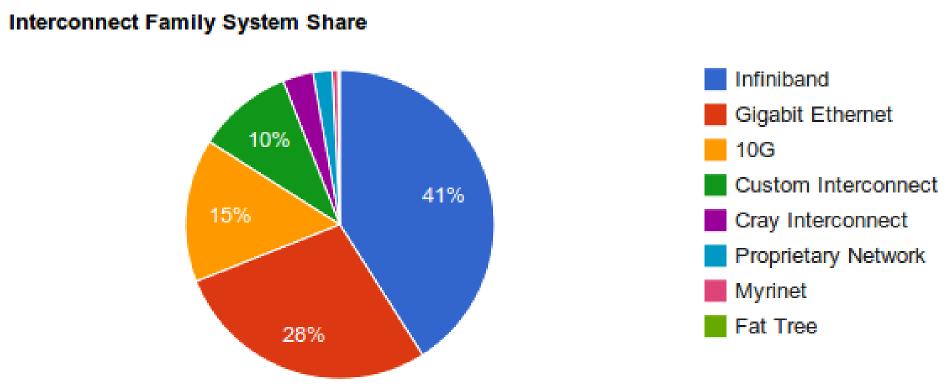


Figure 25: Top 500 Interconnect Share (June 2013) [70]

retry and resets, and data security. As these discussions developed, HToE's feature set was improved and multiple figures were drawn to illustrate the correct implementation of the specification. Although the complete HToE specification is not posted openly (except to HTC members), a version of the specification for public consumption was written and presented by this author [175].

5.2 Motivation for an Ethernet-based Specification

While InfiniBand has made great strides in the HPC space due to its high bandwidth and low latency, Ethernet remains a valued interconnect for both low-end data centers and for high-performance clusters. Figure 25 shows that despite InfiniBand's performance advantages, Ethernet is used in 43% of the Top 500 high-performance clusters in the world as compared to InfiniBand's 41%. While many of these systems currently use 1 Gbps Ethernet, newer Ethernet standards including the 10 Gbps [1] and 40 Gbps Ethernet standards are providing performance-related updates that aim to make Ethernet a more high-performance networking fabric. At the same time, Ethernet has evolved as a lower-cost and "software-friendly" alternative to IB. Thus, the ability to integrate HT over Ethernet would enjoy significant infrastructure and operating cost advantages in data center applications and certain segments of the high-performance marketplace.

5.2.1 Performance

Woven Systems (now Fortinet) in 2007 demonstrated that 10 Gigabit Ethernet with TCP offloading can compete in terms of performance with SDR InfiniBand, with both fabrics demonstrating latencies in the low microseconds during a Sandia test [164]. In addition, switch manufacturers have built 10 Gigabit Ethernet devices with latencies in the low hundreds of nanoseconds [34] [169]. Recent tests with iWARP-enabled 10GE adapters have shown latencies that are on the order of eight to ten microseconds, as compared to similar InfiniBand adapters with latencies of four to six microseconds [35]. More recent tests have confirmed that 10 Gigabit Ethernet latency for MPI with iWARP is in the range of eight microseconds [89]. In more recent years 10GE adapters with hardware acceleration have latency in the range of six microseconds [147] and in some cases, one to two microseconds, [28] which is directly comparable to the best InfiniBand implementations for small messages.

These latencies already are low enough to support the needs of many high-throughput applications, such as retail forecasting and many forms of financial analysis which typically require end-to-end packet latencies in the range of a few microseconds. Additionally, the new IEEE 802.3ba standard for 40 and 100 Gbps Ethernet aims to make Ethernet more competitive with InfiniBand in terms of bandwidth. Although full-scale adoption is likely to take several years, there are already several products focused on the core router market that support 40 and 100 Gbps Ethernet. [88].

The challenge with using these lower-latency fabrics is in making their lower hardware latencies accessible to the application software layers without having to engage higher overhead legacy software protocol stacks that can add microseconds of latency [9] [98]. The HToE specification is a step towards that goal, since it focuses on using Layer 2 (L2) packets and a global address space memory model to reduce dependencies on software and OS-level techniques when performing remote memory accesses.

5.2.2 Cost and Market Share

The continued use of low-bandwidth Ethernet (1 Gbps and 10 Gbps) indicates that cost plays an important role in the construction of computational clusters (for example, for market analysis and geological data analysis in the mineral and natural resource industries). Part of the reason for this widespread market share is due to the low cost of Gigabit Ethernet and the falling cost of 10 Gigabit Ethernet, as well as the management and operational simplicity of Ethernet networks.

However, it should also be noted that InfiniBand still enjoys a price and power advantage over 10 and 40 Gbps Ethernet due to being first to arrive in the HPC market. In 2011, a 40 Gbps, 36 port InfiniBand switch cost around \$6,500 and had a typical power dissipation of 226 Watts [76], while a 10 Gbps, 48 port Ethernet switch cost around \$20,900 and had a power dissipation of 360 Watts. As of 2013, a 10 Gbps, 52 port Ethernet switch still costs $\sim$\$22,000 but now has an average power dissipation of 191 watts [5].

One of the strongest factors for using Ethernet is the trend toward converged networks, driven in large part by the need to lower the total cost of ownership (TCO) of data centers. For example, Fibre Channel (FC) has been the de facto high-performance standard for SANs for the past 15 years. The technical committee behind FC has been a major proponent of convergence in the data center with their introduction of the Fibre Channel over Ethernet (FCoE) standard [48]. This standard relies on several new IEEE Ethernet standards that are collectively referred to as either Data Center Bridging (DCB) or Converged Enhanced Ethernet (CEE) and are described in more detail in Section 5.3.3. The approval of this standard and subsequent adoption by hardware vendors bodes well for the continued usage of Ethernet in data centers and smaller high-performance clusters.

Possibly one of the best indicators of the future market share for Ethernet as a high-performance data center and cluster fabric is the willingness of competitors to embrace and extend Ethernet technologies. Two examples are the creation of high-performance Ethernet switches [111] and the development of RDMA over Converged Ethernet (RoCE) [6], which has been referred to by some as “InfiniBand over Ethernet” since it utilizes the InfiniBand verbs and transport layer with a DCB Ethernet link layer and physical network.

5.2.3 Scalability

As the most prevalent commodity interconnect technology in previous generation data centers, there has been considerable effort devoted to constructing scalable Ethernet fabrics for data centers. For instance, highly scalable fat tree networks have been incorporated into data centers using 10 Gigabit Ethernet [139], while network vendors have already embraced the in-progress standards for Data Center Bridging as a way to create converged SANs and a high-performance cluster fabric [90]. Other recent studies have demonstrated techniques for active congestion management to enable further scaling of topologies constructed around Ethernet [164] [87].

5.3 HToE Specification Requirements

Due to the differences between the point-to-point communication of HyperTransport and the switched, many-to-many communication of Ethernet, the HyperTransport over Ethernet specification needs to address several key requirements to ensure correct functionality. To manage the traversal of packets between these fabrics, the HToE specification focuses on a bridged implementation using encapsulation of HT packets (typically up to 64 Bytes of data) in larger Ethernet packets (up to 1500 Bytes or larger in some cases). In order to remain faithful to end-to-end HT transparency at the software level, the requirements of the HT protocol now translate into requirements for Ethernet transport that are realized in Layer 2 switches.

Furthermore, to productively harness the capabilities of HToE, it must be implemented in the context of a global system model that defines how the system-wide memory address space is deployed and utilized. Toward this end, this specification was designed around the use of global address space models and specifically the Partitioned Global Address Space (PGAS) model as illustrated by previous research into the Dynamic PGAS model

[176]. In particular, the use of a PGAS-related model increases the portability of the model and application/system software across future generations of processors with increasing physical address ranges.

5.3.1 On-package and Off-package Addressing

As previously discussed in Section 4.3, HyperTransport address mapping allows for I/O devices and local DRAM to be mapped to physical addresses that are interpreted by the processor for read and write operations. This physical address mapping is hidden from applications using standard virtual addressing techniques in the operating system.

HToE supports a global, system-wide, noncoherent address space. Addresses must be transparently recognized as either local or remote, and the latter must be mapped to memory or device addresses on a remote node. Implicitly, this mapping must translate between address spaces and Ethernet MAC addresses and vice versa. Consequently, this mapping between local HT addresses and the global HToE address space is necessary to encapsulate and transmit HT packets from a local node to a remote node's memory. Additionally, the remote node must not require modification to its local HyperTransport links in order to route packets that have been sent from a remote node – that is, any remote requests must appear to the local HT link as an access by a local device to a local address. More details on the specific mapping used by HToE are discussed in Sections 5.4.1 and 5.4.2.

5.3.2 Scaling HyperTransport Ordering and Flow Control for Cluster Environments

HyperTransport is a point-to-point protocol that uses three virtual channels to send and receive command and data packets. The HT protocol has been designed to ensure that packet ordering on these channels is preserved on local links via the HT Section 6 Ordering algorithm [75]. This algorithm ensures not only that packets arrive in a logical order but also that deadlock freedom is maintained. In a switched Ethernet environment with the possibility of packet loss, preservation of ordering becomes a much more difficult problem. Thus, any HToE solution must guarantee that packets remain ordered correctly within their virtual channels. For more information on maintaining packet ordering, see Section 5.4.3.

In addition to packet ordering, the HT 3.1 specification also defines a multi-channel, credit-based flow control algorithm. Credits typically flow between two point-to-point links based on the receipt and processing of packets within each virtual channel. In a scalable, switched Ethernet environment, packets could conceivably flow from multiple sources to one destination. Furthermore, since HyperTransport packets are much smaller than Ethernet packets, another requirement is that multiple HyperTransport packets can be encapsulated in one Ethernet packet to reduce the overhead of encapsulation. Both of these requirements indicate the need for a careful rethinking of how to send HyperTransport credits and packets when using HToE. The main flow control requirement is that the sender must possess credits for all HyperTransport packets that it encapsulates. Also, HyperTransport packets that are encapsulated in a single Ethernet packet must be of the same virtual circuit and headed for the same destination.

5.3.3 The Benefit of a Congestion-Managed Ethernet Network for Flow Control

One recent development that was investigated for this specification was the introduction of several IEEE specifications, collectively known as Data Center Bridged (DCB) Ethernet or sometimes Converged Enhanced Ethernet (CEE), depending on the company promoting it.

Data Center Bridged Ethernet aims to provide a congestion-managed Ethernet environment to support converged fabrics in the data center and was motivated by the convergence of the Fibre Channel standard onto Ethernet fabric, aka FCoE [151]. These fabrics aim to prevent packet loss due to congestion but do not prevent packet loss due to bit errors or other sources such as equipment failure or fail-over. Data Center Bridged Ethernet incorporates several specifications including per-priority flow control (IEEE 802.1Qbb), congestion notification (IEEE 802.1Qau), and Data Center Bridging Capabilities Exchange Protocol and Enhanced Transmission Selection (IEEE 802.1Qaz) [78]. These congestion-management algorithms are especially helpful in high-performance computing because of the intensely self-similar nature of HPC traffic.

HyperTransport over Ethernet is intended to be used with switches that have been designed for Data Center Bridging environments, such as those explicitly created to support Fibre Channel over Ethernet. However, some of the DCB specifications would interfere with the normal ordering and priority requirements specified by the HT Section 6 Ordering Requirements. For this reason, many of the solutions specified for ordering and flow control do not explicitly require features like per-flow flow control. This means that HToE could likely be supported on normal 10 GE hardware, but it could also be enhanced by allowing for the usage of the DCBX protocol, per-flow priorities (for packet flows between different sources and destinations), and Enhanced Transmission Selection for usage with other types of network traffic.

5.3.4 Recovery from Failures

HyperTransport 3.1 has several methods for recovering from errors. A special "poison" bit can be set in HT response packets to indicate to the source processor or device that an operation failed (e.g., a read failed to complete). This error notification typically is passed upstream to the initial requesting device without any notion of the initial request's address. In addition, HyperTransport can use the HT 3.1 retry mechanism to resend packets between source and destination HT devices based on a Go-Back-N algorithm that relies on sequence numbers included in HToE packets. If this mechanism should fail to recover from errors, the host processor has the option to issue a reset using a warm or cold reset that is communicated to devices via separate physical signals.

In the HToE environment, these requirements for recovery from errors become more complex due to the nature of HyperTransport transactions and due to the fact that Ethernet does not support the HyperTransport physical signals. Thus the HyperTransport over Ethernet specification must ensure that 1) errors can be appropriately reported to the requesting remote node, 2) resets can be accurately communicated to remote nodes when otherwise unrecoverable failures occur, and 3) resets for traffic between one source and destination HTEA does not affect traffic from other HTEAs.

5.3.5 Requirements for Retry in HToE

The HT specification defines a retry mechanism that resends packets when errors are discovered using a Go-Back-N algorithm and sequence numbers for HyperTransport packets. This mechanism must be extended to function over Ethernet and thereby becomes part of the HToE specification. Ethernet does not define a Layer 2 error retry protocol, so the HToE spec uses a variant of HyperTransport 3.1's retry algorithm that can function across an Ethernet fabric in the presence of packet loss due to congestion or due to bit errors.

5.4 The HyperTransport Over Ethernet Specification

The HyperTransport over Ethernet specification outlines the basic functionality of the HToE bridge device, or HyperTransport Ethernet Adapter (HTEA), that is used to encapsulate HyperTransport 3.1 packets into Ethernet packets. A normal Ethernet MAC can be shared for both HToE traffic and TCP/IP traffic, although the implementer should decide on how to prioritize each traffic type. Alternatively, one can use different adapters to separate HToE and TCP/IP traffic - this approach may be useful for web servers that would like to use HToE with the spill/receive model with maximum bandwidth available for both application and memory traffic.

To assist with the implementation of each of the specification's requirements, functionality in the HTEA is divided into separate "layers" that are implemented in the hardware of the HTEA and that communicate with other layers when processing incoming or outgoing HT packets. Some of the more interesting aspects of the "mapping" layer, the "ordering and flow control" layer, and the "encapsulation" layer (Figure 26) are described in the following sections.

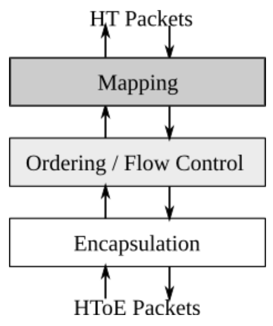


Figure 26: HyperTransport Over Ethernet Layers

5.4.1 Mapping HT Addresses into the Global Address Space

HyperTransport over Ethernet assumes that the range of memory addresses on each node form a subset of a global, 64-bit physical address space. In order to map the local HyperTransport address to a global memory address, such as those used with some PGAS models [176], and to a destination Ethernet address for remote nodes, a few of the upper bits from the physical address are used to select among potential remote nodes in the mapping layer of the HTEA as previously shown in Figure 15 in Section 4.4. This mapping allows for a processor on a local node to make a non-coherent put or get operation to the memory of a remote node.

While the creation of a mapping table is left up to implementers of the HTEA, the selection of global addresses for a particular HTEA and node can be defined using OS-level communication and subsequent PCI-style Programmed I/O commands to write to the HTEA or by using the new functions of the Data Center Bridging Capabilities Exchange Protocol (DCBX) [78] to communicate mapping parameters at the link layer level between DCB-enabled switches.

This mapping of local HT packets to remote nodes also requires the creation of a logical organization scheme to keep track of distinct source and destination pairs, known as a Virtual Link (VL) in the specification. As shown in Figure 27, a Virtual Link couples information such as available credits and buffers for the three virtual channels on the local link as well as information like the destination MAC address. Once the mapping layer of

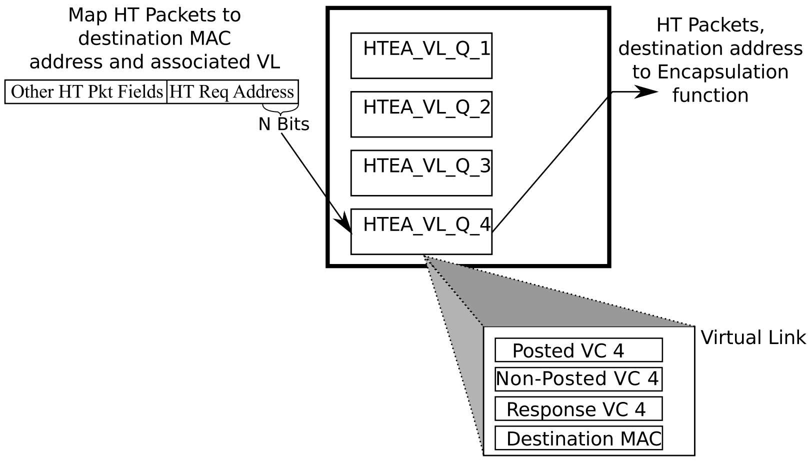


Figure 27: HyperTransport Ethernet Adapter Virtual Link

the HTEA decides which destination MAC address a particular HT request address maps to, the HyperTransport packet is queued according to available credits and associated buffer space at the remote HTEA. These credits are discussed more in Section 5.4.3.

5.4.2 Tag Remapping for Higher Performance

In addition to mapping local HT requests into the global address space supported by HToE, the HToE specification also supports mapping optimizations for the HTEA that allow for increased scalability while still preserving the local link's ability to transparently handle remote HT packets without needing knowledge of their source. This particular optimization is directly derived from previous research done to support the DPGAS model [176].

One of the limits to scalability in an HToE implementation is related to the number of outstanding Non-Posted requests that can be issued by a HyperTransport device at one time. Since the HTEA interface with the HyperTransport link follows all the normal protocols of a HyperTransport device, it is limited to sending a relatively small number of Non-Posted requests (that require a response packet) to the local link using unique Source Tag (SrcTag) bits. Furthermore, packets that are received at a HTEA may have their own Source Tag bits that conflict with requests from other source HTEAs. For this reason, the HToE standard implements a technique called tag remapping [176] to maximize the number of Non-Posted

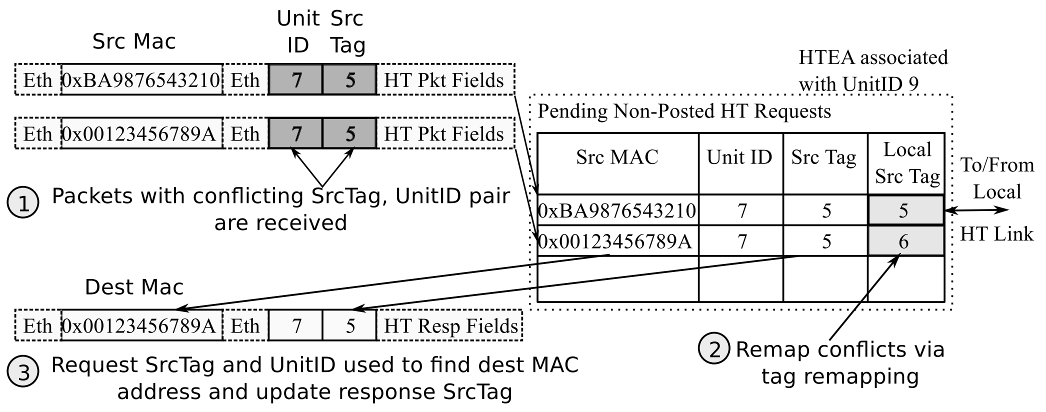

Src Mac
Unit ID
Src Tag
Eth
0xBA9876543210
Eth
7
5
HT Pkt Fields
Eth
0x00123456789A
Eth
7
5
HT Pkt Fields
1 Packets with conflicting SrcTag, UnitID pair are received
HTEA associated with UnitID 9
Pending Non-Posted HT Requests
Src MAC
Unit ID
Src Tag
Local Src Tag
0xBA9876543210
7
5
5
0x00123456789A
7
5
6
To/From Local HT Link
Dest Mac
Eth
0x00123456789A
Eth
7
5
HT Resp Fields
2 Remap conflicts via tag remapping
3 Request SrcTag and UnitID used to find dest MAC address and update response SrcTag

Figure 28: Tag Remapping in the HTEA

requests that can be sent to the local HT link. Figure 28 shows how tag remapping works with two conflicting incoming requests. The original SrcTag, Unit ID, and source MAC address are stored in a pending request table on receipt. If a newly arrived request conflicts with a pending request, its SrcTag and Unit ID bits are remapped and the mapping is maintained in the pending request table. When a response returns to the HTEA, the response's fields are matched up against this table to restore the SrcTag and Unit ID fields as well as to determine the correct destination HTEA for a response.

The HyperTransport specification also specifies an optional technique called Unit ID Clumping that can be used with tag remapping to give the HTEA additional Source Tags for use with the local HyperTransport link. Unit ID Clumping is not a requirement for HToE implementations, but it provides an example of how HToE can be scaled to handle additional sending HTEAs while conforming to the requirements of the original HyperTransport specification.

5.4.3 HToE Ordering and Flow Control for Multiple Senders, Single Receivers

HToE ordering relies on the HT 3.1 ordering requirements, also known as HyperTransport Section 6 Ordering Requirements. Although there are no requirements for packets going to different destinations (from different VLs), ordering of packets within a VL is preserved by the HToE retry algorithm and by sending all Ethernet packets for a specific source/destination pair on the same Ethernet priority level.

In contrast to point-to-point communication, a HTEA must receive packets from multiple source HTEAs. To handle this many-to-many communication pattern, the HToE specification uses a very simple credit-based principle for end-to-end buffer management – any HyperTransport packets that are sent to a remote node must have a HyperTransport credit for the Virtual Link before they can be encapsulated into an Ethernet packet. Additionally, each HT credit is equal to one buffer in the receiving HTEA.

Unlike HT links where HT credit-carrying NOP packets continuously flow on the physical link, credits are passed in the HToE environment only when the receiving HTEA has available buffers for incoming HT packets. A certain number of buffers must be reserved to allow sending HTEAs to initiate new connections, but additional buffers and credits are allocated by the receiving HTEA as its flow control and credit allocation schemes specify.

As buffers are filled in a receiving HTEA, the lack of available credits introduces backpressure on the sending HTEAs. Figure 29 shows how this backpressure causes buffers in the sending HTEA at Node 1 to become full, pausing transactions until more credits are available. Note that since each Virtual Link has its own set of credits, lack of credits for one source-destination pair should not affect the traffic for another VL.

The HToE specification defines the minimum required flow control mechanism, which is to support HT Section 6 Ordering for HT packets that map to the same HTEA-VL. However, this definition leaves open many opportunities to optimize the allocation of credits and buffers to multiple senders. One such optimization is discussed in further detail in Section 6.2.1.

5.4.4 Encapsulation and Support for Recovery and Resets

In addition to specifying how HyperTransport packets are packed into Ethernet packets, the encapsulation layer also interacts with recovery and reset mechanisms that have been

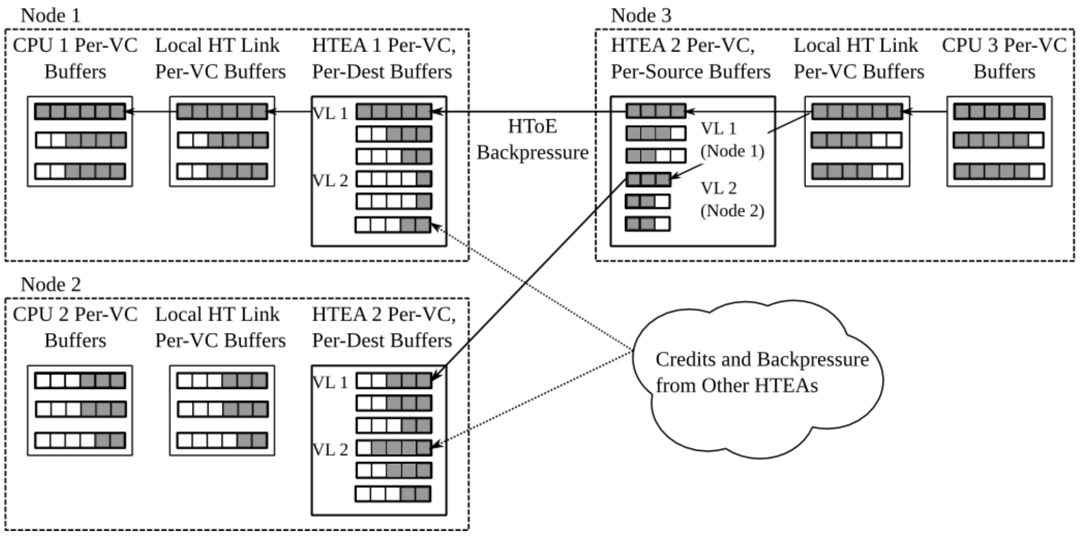

Node 1
CPU 1 Per-VC Buffers
Local HT Link Per-VC Buffers
HTEA 1 Per-VC, Per-Dest Buffers
VL 1
VL 2
HToE Backpressure
Node 3
HTEA 2 Per-VC, Per-Source Buffers
Local HT Link Per-VC Buffers
CPU 3 Per-VC Buffers
VL 1 (Node 1)
VL 2 (Node 2)
Node 2
CPU 2 Per-VC Buffers
Local HT Link Per-VC Buffers
HTEA 2 Per-VC, Per-Dest Buffers
VL 1
VL 2
Credits and Backpressure from Other HTEAs

Figure 29: HToE Backpressure-based Flow Control

adapted from HT 3.1 to handle HToE packets. Each HToE packet contains a special sequence number that is used by the HToE retry algorithm to determine if HToE packets are received in order. This sequence number and retry algorithm are very similar to the 3.1 Go-Back-N algorithm, but each sequence number refers to an entire HToE packet, not just one HT packet. Further error checking is provided by CRCs on both the HToE Payload and the use of the normal Ethernet CRC.

In the case of an unrecoverable error that leads to reset, the encapsulation layer specifies a method for performing link-level resets of one or more Virtual Links that is similar to HyperTransport's concept of cold and warm resets. Since HyperTransport over Ethernet does not include the additional physical sideband signals that HyperTransport devices normally include (such as the power and reset signals), resets must be passed using packets or using OS-level communication. A special encapsulation packet header defines fields for these selective resets, limits their scope, and keeps the entire HTEA from having to reset due to an error between one source and one destination.

While some errors lead to reset, many errors just require a response to notify the original requesting processor that a request packet has not received a valid response. Similar to how HT specifies a method for sending responses with error bits to notify of errors, HToE allows for remote transactions to be terminated, and the HTEA can handle error notifications. To do this the HTEA must keep track of sent HyperTransport packets that require a response (Non-Posted packets). If the HTEA receives a notification that the response has been lost or the remote node has been reset, it can then send a HT packet to the original requester with the "poison" or error bit set. This additional state for remote requests allows for easier error detection and detection of request timeouts.

5.5 Concluding Remarks

The design of HToE in support of the DPGAS model contributed to the further investigation of HyperTransport over Ethernet by the HyperTransport Consortium, and the HToE specification provided a full implementation designed to support encapsulation of standard HT packets over a lossy, L2 Ethernet fabric, with features including HT credit-based flow control, HT-based retry and recovery, and optimizations such as tag remapping with optional UnitID clumping. The ratification of this specification in 2010 led to a similar specification being created for HyperTransport over InfiniBand [17]. This specification contained many of the same features with regards to encapsulation of HT packets into IB packets, but there were several differences due to added features that are found in standard IB fabrics. For example, addressing was modified to support IB Local IDs and Global IDs rather than Ethernet MAC addresses, and many of the retry and recovery features were suggested only for unreliable connections due to IB's support for paired connections.

Both of these specifications provided a relatively complete guide to help hardware implementers to create adapters based on the HToE and HToIB specifications, and they were later combined with the High Node Count specification under the umbrella of the HyperShare marketing terminology currently used by the HTC [77]. At the same time, these specifications did not address essential hardware questions, such as how to design an adapter to satisfy a specific application with strict flow control and bandwidth requirements. Chapter 6 takes a closer look at a large-scale simulation that was designed to address this problem for a specific application domain - accelerator clouds.

CHAPTER VI

A CONVERGED FABRIC MODEL FOR ACCELERATOR CLOUDS

The HToE specification discussed in Chapter 5 provides a complete hardware implementation of the DPGAS model that can be used in data centers to provide high-performance data movement between local and remote memory. At the time of this specification's ratification in 2010, many data center operators were starting to focus on the use of new accelerators, such as GPUs, to speed up common parallel tasks like database queries. The addition of these new accelerators created a new challenge: How can both host and accelerator memory resources be aggregated in a way that is high-performance and programmer-friendly?

This chapter applies the DPGAS model and the HToE network layer to solve this challenge for a specific data center configuration referred to as an "accelerator cloud" – a cluster with a limited number of high-end accelerators that are provisioned to reduce Total Cost of Ownership (TCO) and power [41]. The use of HToE for accelerator clouds is discussed in relation to a system model where accelerator and memory resources are shared via a converged commodity fabric that provides a low-latency aggregation layer. On top of this system model, a new application, data warehousing, is examined and evaluated using performance-related features from the HToE specification [174]. In addition, this research includes a detailed investigation of the HToE network adapter, or HTEA, and specific optimizations that can improve the network performance for the target data warehousing application.

6.1 Motivation for Accelerating Data Warehousing

Recent growth in the size of data sets in the business world [79] has led to an increased need for computational power and large amounts of memory to support high-performance processing of what has become known as "Big Data". Graphics processing units (GPUs) can provide added computational power, but data must first be transferred to these devices for processing. In addition, data center applications cannot easily benefit from expensive hardware and specialized software (e.g., message passing) that enables data movement for other domains, such as high-performance computing (HPC), because using these frameworks would require significant changes to the application's design. Thus, the challenge of efficiently using accelerator and host resources is not just a performance question but also one of programmability.

Many business applications, such as the read-only databases used in data warehousing, do not actually require complex solutions to ensure memory coherence between nodes; in some cases they can take advantage of relaxed consistency requirements such as eventual consistency [160] or on-demand consistency [51]. For this reason, the DPGAS model can be used to provide a memory management scheme that supports simplified application programming and that provides high-performance data movement using non-coherent, scalable remote memory accesses. HyperTransport over Ethernet provides the ideal hardware support for these types of Big Data applications in that it is low-cost, commodity, and has minimal dependence on software interference and coherence protocols. This chapter focuses on the use of HToE as a high-performance network layer while Chapter 7 discusses Oncilla, a software layer that addresses both performance and the programmability aspect of using accelerator clouds for data warehousing.

6.1.1 Data Warehousing

Data warehousing, as used in this thesis, refers to a specific database model and access method that is represented by an industry standard benchmark, TPC-H [152]. While TPC-H is listed as a "decision support" benchmark, many businesses commonly refer to this type of database model as a "data warehouse" due to its emphasis on large numbers of entries that are typically based on customer orders or existing inventory. The TPC-H specification details a common online transaction processing (OLTP) database schema that is filled with pre-generated data and that is then modified via refresh functions and accessed via decision support query functions. For the purposes of data warehousing acceleration, this work focuses on optimizing the query functions, which typically are read-only operations. Figure 30 shows the schema of TPC-H's database, which is the target for query functions.

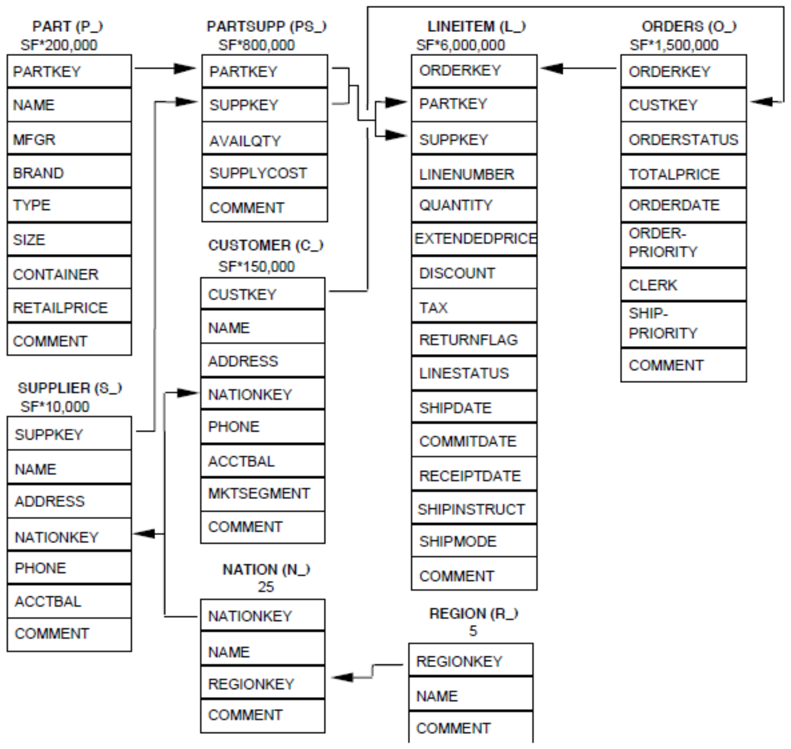


Figure 30: TPC-H Database Schema [152]

The TPC-H benchmark suite contains 21 queries, where Query 1 (Q1) is the simplest and Q21 is the most complex. Listing 6.1 shows TPC-H query 1, which performs a SQL select, group by, and order by operation to return the number and pricing of a certain item in the data warehouse over a certain time period. As discussed in [165], this simple query can be resolved into 15 relational algebra operators including sort, join, select, and aggregate. These relational algebra operators can then be represented using 107 GPU kernels. The creation and optimization of the GPU implementations of relation algebra primitives is the focus of the Red Fox compiler framework.

```
select
  l_returnflag ,
  l_linestatus ,
  sum(l_quantity) as sum_qty ,
  sum(l_extendedprice) as sum_base_price ,
  sum(l_extendedprice*(1-l_discount)) as sum_disc_price ,
  sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as
     sum_charge ,
  avg(l_quantity) as avg_qty ,
  avg(l_extendedprice) as avg_price ,
  avg(l_discount) as avg_disc ,
  count(*) as count_order
from
  lineitem
where
  l_shipdate <= date '1998-12-01' - interval '[DELTA]' day
     (3)
group by
  l_returnflag , l_linestatus
order by
  l_returnflag , l_linestatus;
```

Listing 6.1: TPC-H Query 1

6.1.2 Red Fox compiler framework

The Red Fox compiler is designed to run complex queries with large amounts of relational data for a heterogeneous cluster and is focused on optimizing computation kernels

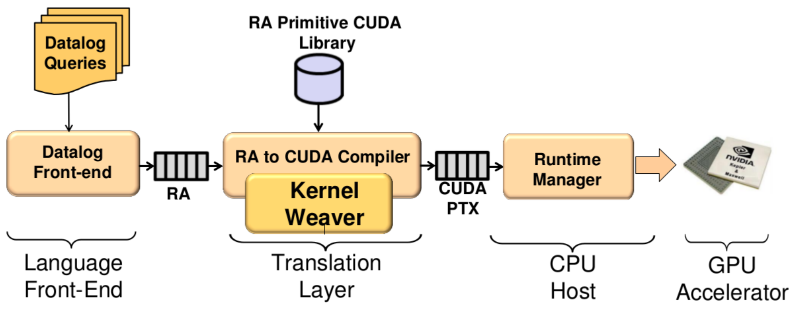


Figure 31: Red Fox Compiler Flow

that run well on highly parallel architectures, such as GPUs. As Figure 31 shows, Red Fox is comprised of: 1) A front-end to parse Datalog, a declarative language, and to create an optimized query plan in the form of a graph of relational algebra (RA) primitives, such as select, project, join, etc., 2) a compiler to map these RA primitives to their corresponding GPU kernels using NVIDIA's CUDA language [38], and 3) a kernel optimization component called Kernel Weaver that applies Kernel Fusion/Fission (KFF) optimizations to reduce the amount of data transferred to the GPU and to overlap data movement with computation [165] [166].

Previous work has shown that database computation maps well to GPUs due the parallel nature of database-related operations [62], and recent work with Red Fox's Kernel Fusion optimization has helped to reduce the overhead of transferring data over the local PCI Express (PCIe) bus by combining computations that operate over the same input set. While Kernel Fusion can be used to limit the number of PCIe data transfers, the transfer of large in-core data sets to smaller GPU global memory is still limited by the efficiency of data transfer between cluster nodes. The system model described in this chapter and the Oncilla framework in Chapter 7 both aim to support the efficient multi-node data transfer mechanism needed to use Red Fox with large data sets.

6.2 *System Model for Future Accelerator Clouds*

This work defines the concept of a commodity converged fabric, or a tightly integrated network layer built around commodity parts, such as HToE, HToIB, and FCoE. As shown in Figure 32, the goal of using commodity converged fabrics for accelerator clouds is to change the ratio of host memory available to GPUs by aggregating memory across cluster nodes while also enabling a simplified memory model for applications that use remote memory or GPUs. Operationally, this model's usage of a non-coherent, system-wide global address space means that transfers of data from remote host memory to GPU memory now appear as a non-uniform memory access (NUMA). This allows data center designers to be able to size accelerator clouds for an appropriate, domain-specific acceleration capacity and to scale these clouds without worrying about coherency traffic constraints.

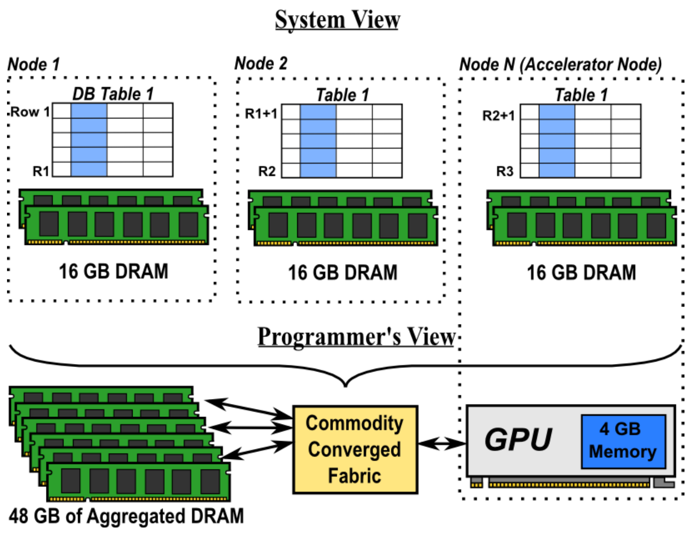


Figure 32: Logical Model of Accelerator Cloud Using Converged Fabrics.

The proposed system model for an accelerator cloud includes N nodes which each have some amount of host memory on each node. Of these N nodes, a smaller subset will have a high-end GPU. While all nodes could be provisioned with GPUs, equipment cost and power as part of TCO means that many data center operators are unlikely to be able to provision GPUs with large amounts of on-board memory, such as NVIDIA Tesla GPUs, for each node. Every node has a fast on-chip network, such as HyperTransport, that allows for non-coherent data transfers as well as a standard off-chip network, such as 10 or 40 Gbps Ethernet. The implementation discussed here assumes that a standard switched topology is used with 24 - 32 nodes connected by a single Ethernet switch. In addition, data transfers to GPU memory are performed using HT posted writes exclusively and any data notifications, such as communicating whether the transfer finished, are handled by software protocols, as needed.

The data warehousing application in this model places data tables in host memory, and these tables can be either segregated to one table per node or striped across multiple nodes. A query application runs on one node, presumably close to the node containing a high-end GPU, and it initiates the transfer of input data for the query using GAS put/get commands to access remote memory.

In addition to these network characteristics, two changes are proposed for the handling of data at the receiving node. To preserve the host memory (DRAM) to be used for additional in-core database space with the data warehousing application, each "accelerator" node can use peer-to-peer communication with an enhanced NIC that implements converged fabric specification (i.e., an HTEA for HToE). This allows the NIC to copy data directly from its buffers to the GPU's memory without needing to pin pages in the host and without needing to interact with the operating system (except to set up the transfer). This optimization is currently supported by NVIDIA's GPUDirect RDMA solution [37]. To further optimize the movement of data for the data warehousing application, an asynchronous streaming model [12] can be used so that the GPU can execute the kernel using input data as it becomes available in the GPU's global memory. Since the output of the TPC-H queries tend to be small, this allows computation and data movement to overlap, especially for queries which are reasonably trivial to execute (e.g., select on a large data set).

A four-node example of the complete system model is shown in Figure 33. Note that

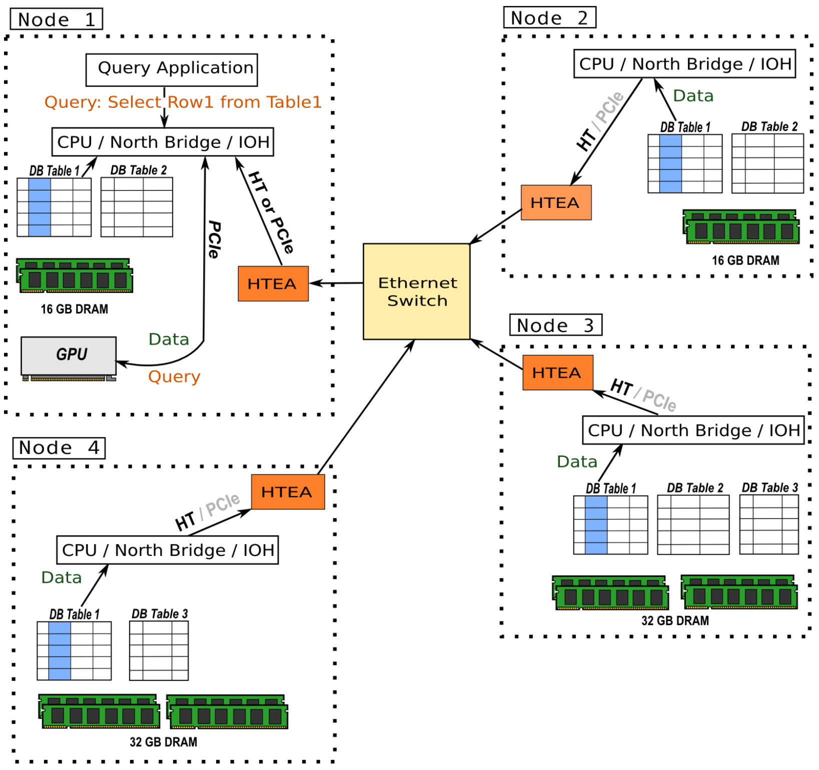


Figure 33: Four-Node Accelerator Cloud System Model.

this figure presents two alternate interconnects connected to the HTEA, one for HyperTransport which connects directly to a northbridge based on the AMD Opteron, and one that routes data through an I/O Hub (IOH) that supports the PCI Express protocol. Future adapters could feasibly support either the HTX (HyperTransport) motherboard slot or a standard PCI Express slot by performing the needed HT to PCIe transformation in the HTEA. However, this work only addressed the former configuration (HTX-based).

6.2.1 Potential Bottlenecks and HTEA Optimizations

A key part of implementing this system model is the investigation of specific bottlenecks that result from sharing high-end GPUs between multiple nodes. A simple diagram of bottlenecks for the system model is shown in Figure 34. Several possible bottlenecks include 1) the link bandwidth of the Ethernet, link 2) the incoming processing speed of the adapter and the size of the buffers in the HTEA, 3) the return rate of credits to “sending” nodes and 4) the transfer bandwidth of PCI Express to the GPU on the “accelerator” node. The total rate of data flowing through each of these links must not exceed the size of the GPU's global memory and 5) the rate of data consumption by the GPU once a query kernel is launched. The experiments in Section 6.4 use measurements from current implementations on GPU hardware for 4) and 5) and a combination of published numbers and detailed simulation numbers to investigate bandwidth- and latency-limiting factors for accelerator clouds.

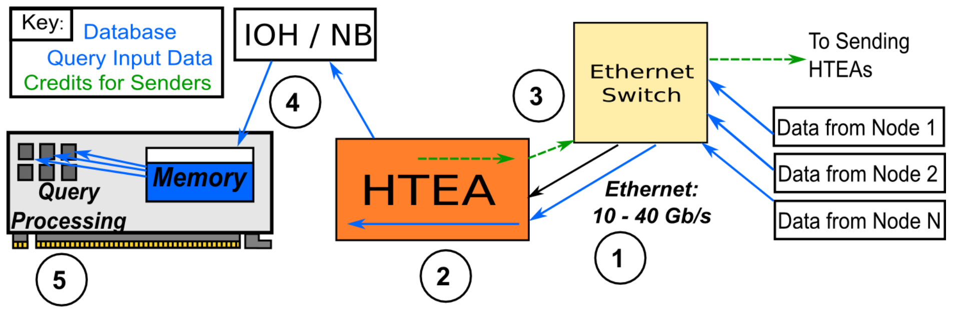


Figure 34: Potential Bottlenecks in a Shared Accelerator Cloud System Model.

To address these potential bottlenecks, several optimizations were tested in the design of the HTEA. As Figure 35 shows, the HTEA has three components that could be optimized in a soft-core implementation (as might be found on an FPGA). The packet transmission protocol is as follows: 1) Outgoing HT packets are encapsulated in as credits are available for a specific source-destination pair (a VL in the HToE spec). Additionally, HT credits (aka NOP packets) are encapsulated with HT data packets for a particular source-destination pair. 2) Encapsulated HToE packets are sent over the Ethernet network to a remote HTEA. 3) At the remote HTEA, HToE packets are deencapsulated. Returned credits for remote buffers are passed to the outgoing path of the HTEA, and incoming HT packets are enqueued until they can be transmitted onto the local HT link, or HT system interface. Once these HT packets are passed to the local HT link, the local buffers are freed, and credits are released to be placed in outgoing HToE packets back to the originating node.

This interdependence of credits and data packets presents several points for optimization. Outgoing HToE packets can be buffered to create a larger Ethernet payload, which increases bandwidth but also increases the latency of each packet and the potential for credit starvation. Also, the width of outgoing pipelines for encapsulating and sending

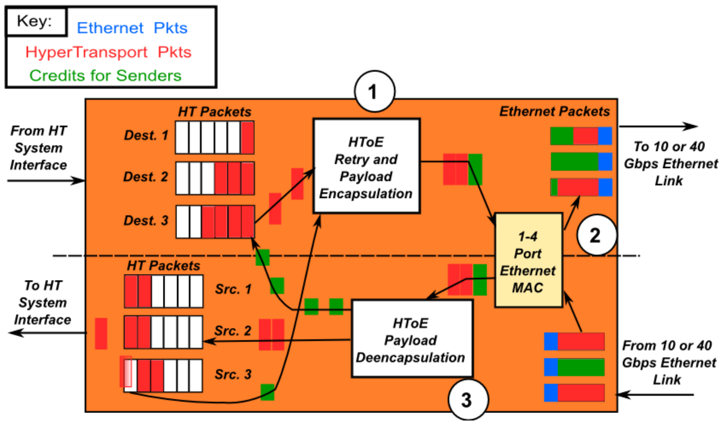


Figure 35: HTEA Design and Potential Optimizations

HToE packets can be increased, increasing the rate at which HToE packets are sent to a remote accelerator node. Similarly, the incoming pipeline for an accelerator node can be parallelized to reduce the time needed to check an HToE packet's checksum and to parse incoming HT packets and credits. Parallel encapsulation and deencapsulation pipelines can be matched by using multiple-port Ethernet adapters, similar to those found in current dual-port Ethernet adapters, such as Intel's 82599 10 Gbps Ethernet adapter, which supports two full-duplex 10 Gbps ports [80].

6.3 Experimental Setup

As with the work in Section 4.7, NS-3 was again used as a synthetic trace-driven experimental platform. However, this set of experiments did not use DRAMSIM and instead focused on more detailed timing characteristics of the HTEA based on previous hardware designs [120] [46] and publicly available numbers [82]. GPU timing numbers were gathered from real experiments using TCP-H data warehousing benchmark queries run using Red Fox [166].

The simulations used two, four, and eight nodes, with one "accelerator" node in each simulation and a one GB data set (about 16 million HT packets, each with a payload of 64 B) striped over the remaining nodes. For example, this meant that the two-node case had the entire one GB data set on one node while the eight node case spread the data set across seven nodes equally.

6.4 Results

Packet latency is shown below for a small HToE message as a baseline for the other investigated optimization scenarios. Results for link utilization (the aggregate bandwidth across the shared link to the receiving HTEA) was measured for three different scenarios: 1) no optimization to the HTEA and varying payload sizes (credits and data), 2) optimization of the outgoing path of each HTEA by using multi-port Ethernet adapters as well as four separate pipelines for encapsulation, and 3) optimization of the incoming path of each HTEA by using multi-port Ethernet adapters as well as four separate pipelines for deencapsulation and credit processing. Finally the aggregate results were used to evaluate how HTEA bandwidth compares to other potential bottlenecks in the accelerator node.

6.4.1 HTEA Packet Latency

Table 9 shows the combined timing for one HT packet passing through a sending and receiving HTEA based on the NS-3 simulations and the Ethernet MAC timing from [46]. The latency is reasonably low for an encapsulation-based adapter with an end-to-end latency of about 1.5 μs. It should also be noted that the most timing-intensive functions are on the receiving path. This delay is due to the need for comparing headers and validating checksums on each received Ethernet frame and HToE payload.

Table 9: Estimated Latency for HTEA Handling of One HT packet

Module	**Latency (ns)**
HToE mapping, queuing	192
HToE credits, retry, encap	244
Ethernet MAC (out)	122
Eth switching and link	244
Ethernet MAC (in)	298
HToE deencapsulation	222
HToE queuing (in)	156
Total	**$\sim$1480**

6.4.2 Effects of Payload Sizing

In Figure 36, the link utilization is shown for one sending HTEA and one receiving HTEA with varying HT packet payload sizes: one to 20 data packets in an Ethernet frame and one to 50 NOP packets in a frame. The low utilization for the base case (one HT packet) and the other one NOP experiments result from the same phenomenon. In both cases, HT packets are encapsulated and sent as fast as possible. When the size of the sender's HT packet payload exceeds the size of bulk credit acknowledgments from the receiver, the sender quickly runs out of credits.

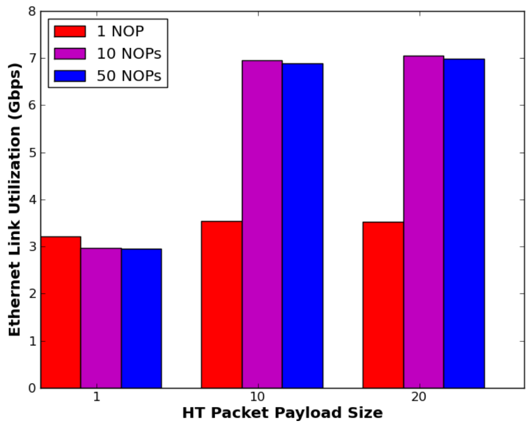


Figure 36: Link Utilization versus NOP Payload Size for Two Nodes.

As the size of the bulk credit acknowledgments and the size of the HT data payload increases, the average number of available credits for the sending HTEA increases, as shown in Figure 37.

Both 10 and 50 NOPs in an acknowledgment payload result in the same average number of credits available at the sender. This indicates that the encapsulation process for HT data packets takes slightly longer than processing received credits, and that the bulk NOP size doesn't need to be dramatically larger than HT data payload size. However, it should be noted that use of the bulk NOPs decreases link bandwidth slightly (Figure 36) because it reduces the total number of Ethernet frames that are sent in the same amount of time.

Figure 38 demonstrates that using smaller or larger payloads can also affect delay within

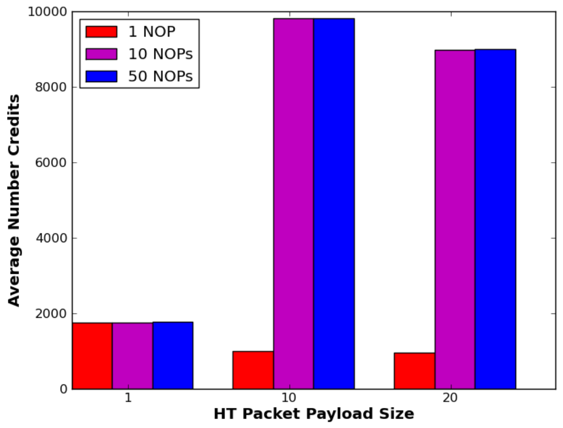


Figure 37: Average Number of HT Credits versus HT Payload Size for Two Nodes.

the network adapter. The outgoing path represents the delay that results when an HT packet has a credit but must wait to be encapsulated due to the first-in, first-out (FIFO) nature of an unoptimized HTEA. Similarly, the incoming path delay results from the latency required to deencapsulate Ethernet frames and HToE payloads. Delay for outgoing HT data payloads ranges from a low of 390.84 ns (20 HT packet payload) to 48.26 ms (one HT packet payload), and delay for incoming payloads ranges from 154.74 ns (one HT packet payload) to 17.90 ms (20 HT packet payload). Small packet payloads have higher delay on the outgoing path due to head-of-line blocking, but they also are processed faster at the receiver.

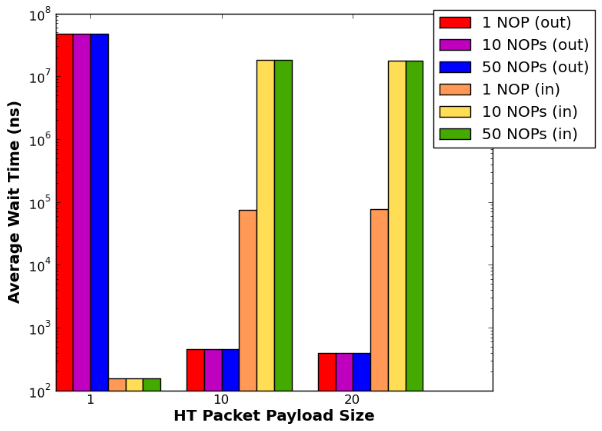
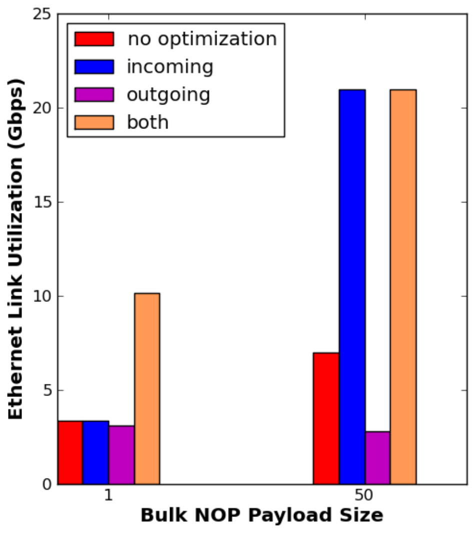


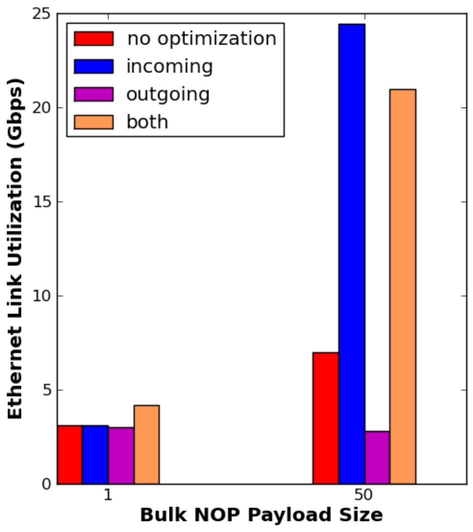
Figure 38: Average HTEA Delays versus HT Payload Size for Two Nodes.

6.4.3 HTEA Pipeline Optimizations

Figures 39a and 39b show link utilization with the use of the proposed HTEA (four parallel processing stages) pipeline optimizations for either the incoming path (deencapsulation), outgoing path (encapsulation), or for both paths.



(a) Four Node Accelerator Cloud



(b) Eight Node Accelerator Cloud

Figure 39: Link Utilization versus NOP Payload Size

These graphs for 20 HT packet payloads and varying bulk NOP payload sizes illustrate two important concepts: 1) Pipeline width optimizations can't make up for a mismatch in the speed of NOPs returning from the receiver to senders, even though the wider outgoing pipeline reduces the average wait time for encapsulation of NOPs from 42.29 ms to 1774 ns. 2) Deencapsulation pipelining affects performance much more than encapsulation pipelining, potentially boosting bandwidth up from three Gbps to 24.45 Gbps. This improvement results from optimizing deencapsulation, which is a much more time-intensive operation (Table 9). In some cases, aggressive encapsulation pipelines can actually reduce bandwidth as shown by the unoptimized, 50 NOP case (6.99 Gbps) and the outgoing optimization, 50 NOP case (2.80 Gbps). This reduction in bandwidth seems to be due to an unusual case where the optimized sender fills the receiving HTEA's buffers and must wait until the receiver processes some of the incoming Ethernet frames before it can accept more frames from the sender.

In summary, the use of wider pipelines can definitely provide much better performance for the HTEA but only in conjunction with the use of bulk NOPs. Bulk NOP acknowledgments allow the rate of HT packets from senders to increase (assisted in part by wider pipelines) while making sure that the overhead of processing credits does not delay the encapsulation process.

6.4.4 Potential Bottlenecks in the HTEA

Finally, the HTEA design and optimizations are checked against the potential bottlenecks described in Figure 34. As seen in Figure 40, the maximum link bandwidth was about 24 Gbps (labeled as 8Nd) over a 40 Ethernet Gbps link, which means that the "optimized" implementation would not be limited by the available Ethernet bandwidth. At the receiving HTEA, the measured incoming bandwidth onto the local HT system interface maxed out at 3.69 Gbps, meaning that the incoming packet stream is more limited by packet processing in the HTEA than by the HT system interface. Finally, the query processing bandwidth of the TCP-H Q1 and Q21 queries was much higher (2,747 Gbps and 112 Gbps) than the PCIe transfer bandwidth (44.55 and 23.57 Gbps). Overall, these results indicate that the network adapter is a potential bottleneck and a good focus for optimization. As the previously discussed optimizations show, there is still room for performance enhancements in the development of future converged commodity fabrics.

6.5 Concluding Remarks

This chapter discusses the application of the DPGAS model to solve challenges related to the efficient use of accelerator resources in a cluster. The data warehousing application was introduced as a new workload that is dependent on high-bandwidth transfer to accelerators and large pieces of host memory to store data sets "in-core". New features from the HyperTransport over Ethernet specification were incorporated into a model of a high-performance HTEA, and further optimizations were used to increase link utilization from

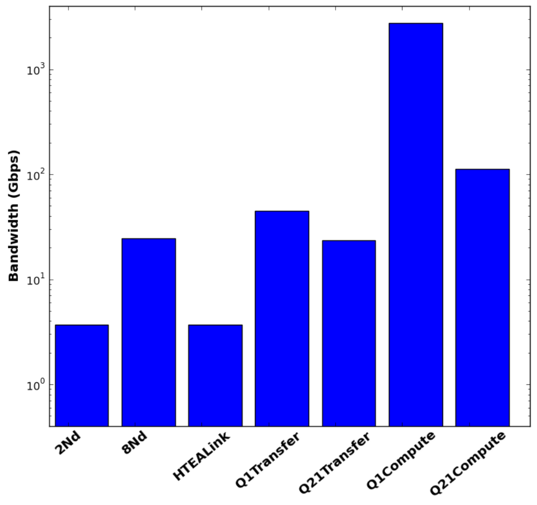


Figure 40: Relative Bandwidths of HTEA, Networks, and GPU.

$\sim$3 Gbps to $\sim$24 Gbps.

Chapter 7 further investigates the data warehousing workload using the concept of a "managed" GAS network layer that underpins the Oncilla software framework. This combined network layer and software is used to provide performance benefits to data warehousing applications even with other commodity interconnects that are not as tightly coupled as the commodity converged fabrics that were discussed in this chapter.

CHAPTER VII

ONCILLA - A SOFTWARE FRAMEWORK FOR MANAGED GAS

This thesis has commented on the convergence of interconnect and memory technologies and has proposed the DPGAS model as an abstraction that can be used to implement aggregation of and high-performance data movement between accelerator and host memories within a cluster. The preceding chapters have addressed the architectural challenges and proposed a hardware implementation used to implement high-performance data transfers and resource sharing, but such hardware components also require a software stack that can expose the DPGAS abstraction to applications.

This chapter describes the design, implementation, and evaluation of Oncilla – a runtime system and API that encapsulates the functionality necessary for dynamic management of global address spaces in a cluster. Oncilla demonstrates a key contribution of this thesis identified in Chapter 3 - the ability to effectively aggregate physical memory and accelerators to meet the demands of applications.

The Oncilla software completes a real-world implementation and demonstration of the DPGAS model and helps to validate the initial hypothesis that commodity global address spaces can be used to create efficient and high-performance memory provisioning. The design and evaluation of Oncilla is focused on a typical data warehousing application, which is demanding in its use of aggregated memory and of throughput via aggregated compute accelerators. Because there is no commercial implementation of HToE at the moment, Oncilla is implemented over a small cluster with an existing interconnection fabric that supports remote memory access. However, the design of the Oncilla runtime is such that it could easily be ported to a native HToE, InfiniBand, or Ethernet layer. This network layer portability is combined with a simplified user-level library that allows for reduced implementation complexity across a variety of networks and accelerators.

7.1 Current State of the Art

Before creating a new software layer oriented around GAS-based data movement, it is important to investigate existing solutions and determine their relative strengths and weaknesses for solving the challenge of efficient memory and accelerator aggregation. Other existing frameworks, such as GASNet [21], MEMSCALE [108], and rCUDA [40] support some form of GAS-based data movement, so a new runtime should be designed to complement this existing work. NVIDIA's Unified Virtual Architecture, or UVA, shown in Figure 41, currently has some GAS-like characteristics in that it supports a common virtual address space across all the host and device memory on one node. However, this capability is currently limited to one node.

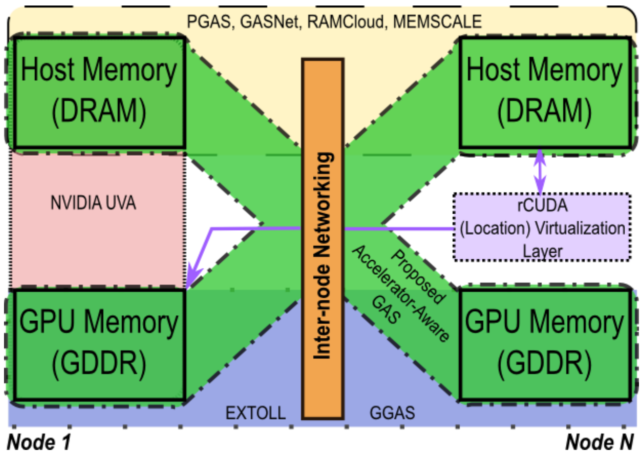


Figure 41: Address Space Models for Accelerator Clusters

GASNet supports the partitioned global address space (PGAS) model of programming with distinct shared and private memory regions (as drawn in yellow in Figure 41) and is primarily focused on improved programmability across a wide variety of cluster interconnects. However, GASNet currently does not focus on data movement between accelerator and host memory. Research into the Phalanx model [55] has extended GASNet with a programming model approach for using GPUs and doing GAS-oriented data transfer, but this project is currently in the prototype stage. Also, both GASNet and Phalanx are limited by their programming model that requires the use of UPC-style primitives to fully take advantage of the underlying GAS model. The use of these primitives may require substantial rewrites for business applications.

From a virtualization perspective, the rCUDA project (represented as a software virtualization layer in Figure 41) implements a solution for virtualizing remote GPUs and handling data transfers using TCP/IP, InfiniBand, and NVIDIA's GPUDirect [141], but it focuses more on the computation aspect of GPU virtualization rather than a cluster-wide aggregation of resources. A closer analogue to the ideal GAS model for host memory and accelerator memory can be found in the data transfer support found in MEMSCALE (also shown in yellow in Figure 41), which shares some characteristics of PGAS models. The MEMSCALE project includes support for using GAS across host memory for standard transactional databases, and the hardware-supported GAS provided by the EXTOLL network is used both by MEMSCALE and by Oncilla. Additionally, GGAS [123], also from the University of Heidelberg, supports GAS across GPU memories using the EXTOLL interconnect and NVIDIA's GPUDirect.

All of these projects provide different solutions to support data movement for multi-node applications, but they are typically focused on the high-performance community and on developers who already have a detailed understanding of data movement APIs (e.g., InfiniBand Verbs, MPI, TCP/IP sockets). The ideal model would enable the coordination of cluster I/O, memory, and network resources with an API that provides the simplicity of NVIDIA's UVA, as currently supported on one node. This proposed model is shown in Figure 41.

7.2 Oncilla System Model

As shown in Figure 42, the generalized system model used with an Oncilla-supported cluster combines the host memory (DRAM resources) *and* accelerator memory (GDDR) into a large, logical partition that can be used by one or more applications. Access to this aggregated memory in a high-performance manner is supported by low-latency and high-bandwidth hardware that supports the one-sided put/get data transfer model. While overall application performance may be affected by the location of remotely allocated memory, application scaling is not restricted by where the memory is physically located.

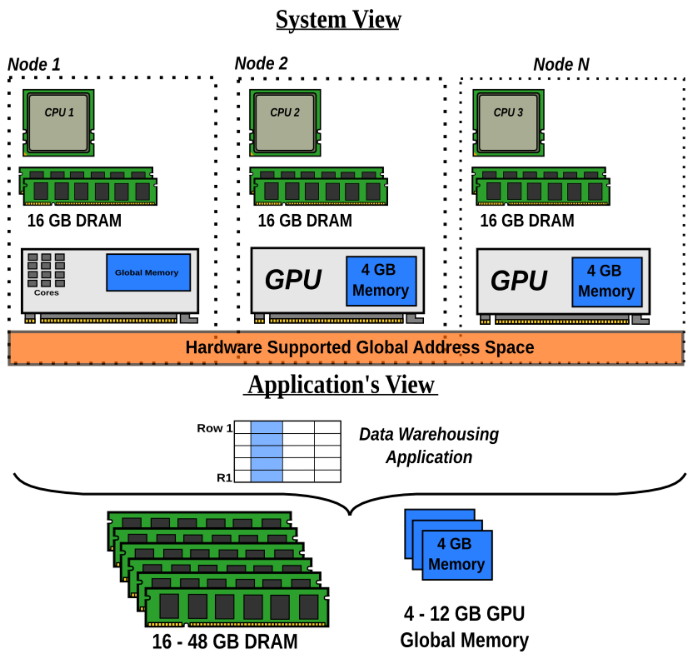


Figure 42: GAS-based GPU Cluster System Model

Oncilla's implementation of this system model combines several different hardware and software components to create a high-performance, commodity solution for allocating remote memory and sharing it more efficiently between cluster nodes. The hardware-supported GAS in this model is built around EXTOLL network interface cards (NICs) or InfiniBand HCAs, and the software layer is composed of the Oncilla runtime for data allocation and movement and the Red Fox compiler for compilation and optimization of data warehousing application kernels for the GPU. Red Fox is discussed further in Section 6.1.2.

7.2.1 The EXTOLL Networking Fabric

EXTOLL [53] is a network fabric that enables the optimization of network performance to support specialized networking tasks, including the consolidation of resources and low-latency data transfer between nodes. The two most relevant aspects of EXTOLL are its support of global address spaces (GAS) for consolidation purposes and low-overhead put and get semantics.

While put/get operations are provided by many different networks, the one-sided communication operations in EXTOLL are optimized for fine-grained communication schemes and high-core counts. Features that have proven to be beneficial for HPC applications also are useful for resource consolidation purposes: the need to keep memory ranges registered is low, as the registration process is very fast ($\sim$2 μs vs. $\sim$50 μs initial overhead for small memory ranges when compared to IB [119]). As registration requires pinning pages in main memory, too many registrations can reduce the number of page swap candidates and significantly impact demand paging and system-level performance. Also, while many put/get semantics provide few possibilities for notification about completed operations, EXTOLL provides notifications on both sides (origin and target) that can signal completion to supervising processes with very little overhead. This feature helps to maximize overlap among the various communication flows.

The EXTOLL NIC currently supports three styles of data transfer: 1) The Shared Memory Functional Unit (SMFU) [50] exposes remote memory via Linux's mmap mechanism and allows for direct manipulation of remote memory using put/get operations on pointers. 2) The Remote Memory Access (RMA) unit supports RDMA-style operations using low-latency pinning of pages, as shown in Figure 43. The low-overhead put/get operations used with RMA are based on a single-instruction dispatch of work requests [119] and are helpful to minimize communication overhead for fine-grained messaging applications, thus maximizing overlap between computation and communication. 3) The VELO unit can be used to send small messages using hardware "mailboxes" and is used to support MPI over the

EXTOLL network. This work makes use of the RMA unit to implement a GAS-supported accelerator clusters, but SMFU or VELO could also be used to perform low-latency put/get operations with GAS.

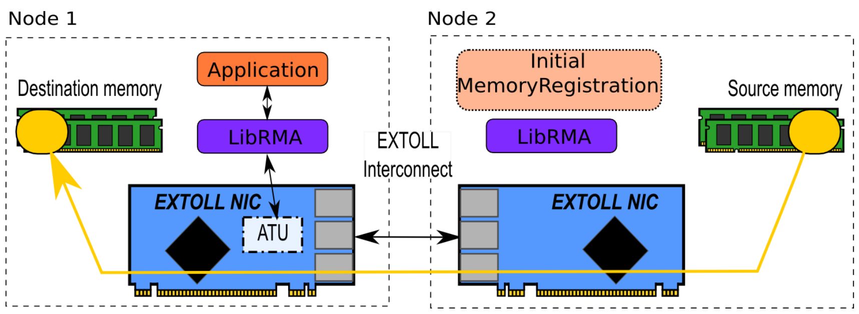


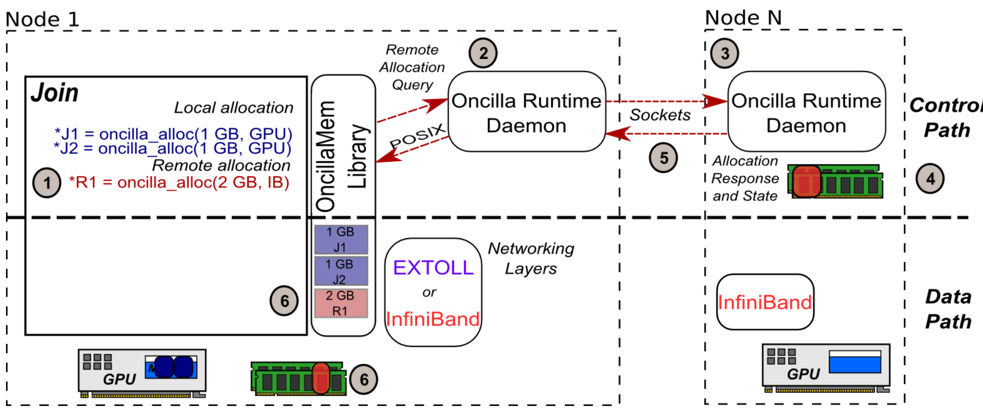
Figure 43: EXTOLL Remote Memory Access Dataflow

Due to the API-based nature of one-sided transfers using the EXTOLL RMA unit as well as the InfiniBand Verbs API to indirectly access memory regions, this model is referred to as a “managed” global address space. This indirect access is mostly due to the limitations of current hardware and software stacks, but the aggregation of multiple, non-coherent regions across the cluster and the use of put/get operations both provide similar functionality to traditional global address spaces.

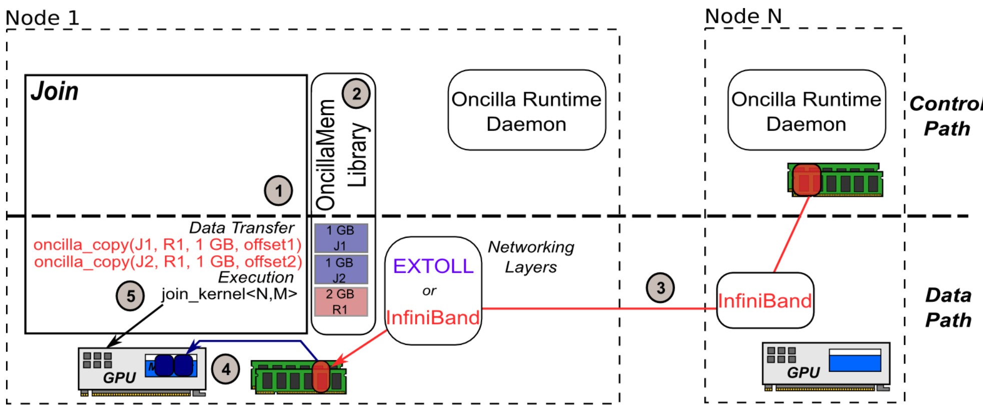
7.3 Oncilla Runtime and API

The Oncilla runtime is designed to abstract away the complexity of remote memory allocation and data movement while also providing optimized data transfer between remote and local host memory and I/O memory (on GPUs). As shown in Figure 44, the runtime consists of a control path and a separate data path. The control path performs the request, allocation, and freeing of local and remote memory and is accessed by an application through the “OncillaMem” library.

The example in Figure 44 illustrates a GPU-based join operation where the input data is stored in a table located on a remote node. Local allocations are serviced easily through standard memory allocation techniques (malloc, or new), but remote allocation (in this case



(a) Control Path



(b) Data Path

Figure 44: Oncilla Runtime and Use of API

via IB) requires the allocation of pinned DRAM pages on the remote node. Once an allocation takes place, the runtime keeps track of the state of the allocation and also any processes needed on each node to set up and tear down memory buffers. This information is made available to the application via the library, and subsequent *oncilla_copy* (aka, *ocm_copy*) calls use this state to transfer data over the desired network layer.

The allocation of GPU and remote memory for a GPU-based join has the following steps that are illustrated in Figure 44a: 1) Local GPU allocations are created using the Oncilla API and *oncilla_alloc* commands. The library calls *cudaMalloc* for each buffer. When an IB (remote) memory buffer is specified, the Oncilla library 2) contacts the Oncilla runtime daemon using a POSIX inter-process message. This daemon then uses a TCP/IP socket connection to request a remote memory allocation from Node N. 3) The request is accepted by Node N, and this node then 4) proceeds to set up a server thread to host a IB buffer pinned to local memory. 5) At the same time, Node N acknowledges the request and notifies Node 1 that it has reserved a buffer for non-coherent data transfer. 6) The Oncilla library on Node 1 adds the remote allocation to its list of existing memory regions, and the runtime daemon forks a client thread to host the local memory endpoint for an IB connection. At this point, there are two GPU buffers and an active IB connection that are all managed by the Oncilla library for the join application.

The data path could consist of any high-performance network, but this work focuses on the EXTOLL RMA network interface due to its low memory registration costs, high bandwidth, and low latency for small messages. InfiniBand RDMA is a similar high-performance networking fabric that is evaluated alongside EXTOLL in Section 7.5.1. Regardless of the selected fabric, once the setup phase is complete, the application relies solely on the library and network layer to transfer data from remote memory to local host memory or local GPU global memory.

As shown in Figure 44b, the data transfer for and execution of a join operation on the GPU proceeds as follows: 1) The Oncilla API command *oncilla_copy* is called for each of the inputs to the join. This diagram shows both inputs stored in a common remote host memory buffer (with offsets specifying the total size of each input set) for simplicity - two buffers could be used. 2) The Oncilla library checks its list of allocations and finds that the source is remote, IB-based memory, and the destination is CUDA-allocated GPU memory. 3) The library executes an IB read into the local buffer, and then 4) a *cudaMemcpy* operation is used to copy data to the GPU. If the local IB buffer is much smaller than the GPU kernel input size and the remote IB buffer, steps 3 and 4 are repeated until all input data has been copied to the GPU. Once the data transfer phase is complete, 5) execution of the join kernel can proceed. In this manner, data transfer can be implemented via different network layers while the application developer is just aware of an abstract, aggregated memory and accelerator space that crosses multiple nodes.

This idea of abstraction is further defined by the Oncilla API which has the concept of “opaque” and “transparent” operators that support either naïve or explicit hints about remote memory allocation and data placement. These library interfaces allow a developer to specify whether a large chunk of memory is needed or whether a chunk of memory on a specific device on a certain node is needed. For example, a naïve allocation might specify that 20 GB of host memory is needed but not the location of this memory.

oncilla_alloc(20 GB, DRAM)

This type of operation could allow for 20 GB of DRAM to be allocated on the local node, across two nodes, or across four nodes. The developer in this case accepts the access penalty for remote DRAM rather than using disk-based operations. Alternatively, a developer could specify allocation in a fashion that is closer to traditional NUMA commands.

oncilla_alloc(20 GB, node1_ddr, node2_ddr, equal_alloc)

This type of “transparent” operator can be extended to GPU memory, and the Oncilla runtime helps to allocate and keep track of more advanced allocation and data movement operations. While scheduling of computation and fair memory allocation decisions can also be incorporated into the runtime, this work does not currently attempt to solve this problem. Future work includes evaluating the Oncilla resource management framework alongside existing scheduling frameworks, such as those found in [134] and [105].

7.3.1 Using Oncilla for Existing Single-node Applications

As Section 7.5 will show, Oncilla can be used to enable multi-node resource allocation and data movement for existing single-node applications. This work has demonstrated a few key steps that are useful to consider when using Oncilla:

- **Plan data partitioning:** The most important step for modifying applications to work with remote memory is to consider how input data should be partitioned. The applications discussed in this chapter depend on aggregation of large amounts of remote memory for one application and the use of one accelerator, but the use of multiple accelerators on multiple nodes would require placement of data sets as close as possible to the desired accelerator.

- **Support non-coherent memory accesses (or not):** Applications that take advantage of Oncilla must either conform to a model that supports non-coherent put/get operations or include additional software layers to provide consistency between shared regions. TPC-H is an application that performs few data updates, so it conforms well to this standard.

- **Use managed allocations** Oncilla's *ocm_alloc* can currently be used to replace GPU allocations, local DRAM allocations and remote host memory allocations with EXTOLL or InfiniBand. Besides having simplified copy semantics with the *ocm_copy* API call, allocation with Oncilla allows the master node to keep track of existing allocations for more informed resource management across the cluster. An *ocm_localbuf* API call provides a standard C pointer to the Oncilla allocated buffer, minimizing code modifications.

- **Define the data transfer model** For aggregated data workloads like data warehousing, data transfer with Oncilla is performed using multiple *ocm_copy* calls to stream inputs from remote host memory to local accelerators. Other applications may require small reads and writes that could affect data placement strategies and the placement of data transfer calls.

7.4 Experimental Setup

Two different two-node systems are used to evaluate the Oncilla framework. Each blade in the EXTOLL cluster includes a single-socket AMD CPU, a solid-state drive (SSD), 16 GB of DRAM, an EXTOLL2 network interface, and a NVIDIA GTX 480 GPU with 1.5 GB of GDDR. Each node in the InfiniBand cluster contains dual-socket Intel CPUs, a standard

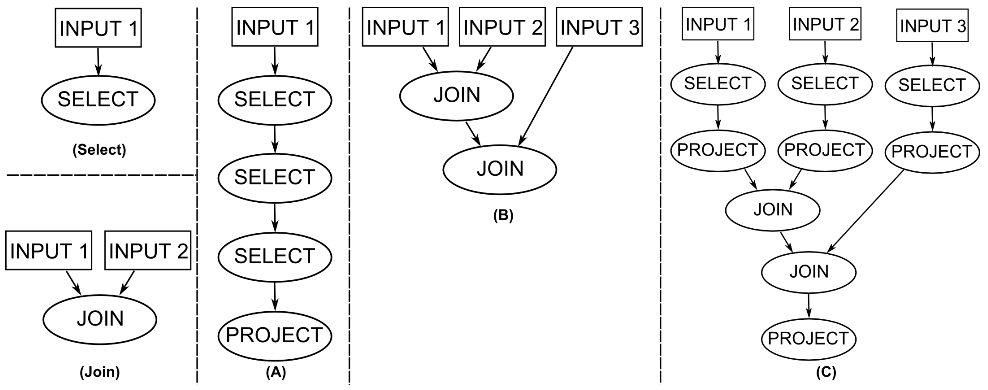


Figure 45: Relational Algebra Benchmarks

5400 RPM hard drive, 12 GB of DRAM, an InfiniBand ConnectX-2 VPI QDR adapter, and a GTX 670 GPU with 4 GB of GDDR.

The main application workload used to simulate large data warehousing applications is based on relational algebra primitives like those that compose the queries represented in the TPC-H benchmark suite [152] (discussed in Section 6.1.1). These queries are also representative of the types of operations that Red Fox aims to support for GPU acceleration. These primitives, including *project*, *select*, and *join*, have been optimized for the GPU to perform efficiently when compared to CPU-only primitives [38]. However, the performance of these GPU primitives is limited by two factors: 1) the size of GPU global memory required for inputs, outputs, and temporary data and 2) the cost of data transfer across the PCI Express bus.

As shown in Figure 45, the tests include two simple operations, select and join, and three more complex operations that incorporate select, project, and join primitives. The more complex microbenchmarks also have two variations that can take advantage of Kernel Fusion [165] to reduce the impact of the PCI Express data transfer costs and to reduce each kernel's GPU global memory footprint. These variations are represented as nonfused or fused (not pictured in Fig. 45) microbenchmarks in the results. The selected patterns for each of the microbenchmarks were chosen because they use optimized versions of relational algebra primitives (designed for Red Fox), and because they represent important subsets of the computation that is performed by TPC-H queries.

The input sets for each of these microbenchmarks are transferred to the GPU either directly from a hard drive using standard C++ file operators (fread) or from local and remote memory using the RMA or RDMA hardware. Due to the limitations of the hardware setup, input files are limited to 24 GB, and each node can allocate up to 12 GB of data to be accessed from local memory, RMA, or disk. The IB cluster is limited to 11 GB, so inputs are pulled from an 11 GB allocation, where 1 GB of input is reused. Data files are represented using binary format and are read sequentially from disk or remote memory.

As a secondary application, the SHOC BFS benchmark [36] is used to represent a simple graph algorithm and to demonstrate how Oncilla can be extended for HPC applications. The benchmark implementation is currently limited to the size of the workload that fits on one GPU, so one million vertices are used as input for each test. Two tests are conducted to demonstrate the overhead of using Oncilla for remote memory allocation and data transfer - one where the input graph resides solely in local memory and one where the input graph resides in remote memory. An ideal implementation of this benchmark would use a multi-node, partitioned graph with Oncilla for data movement instead of other software layers, such as MPI, but the standard for distributed graph implementations, Graph 500 [110], currently lacks suitable support for GPU-based algorithms. For this reason, evaluation of this application focuses on a simple evaluation of the performance aspects and programmability of using the Oncilla model for this type of application.

7.5 *Results*

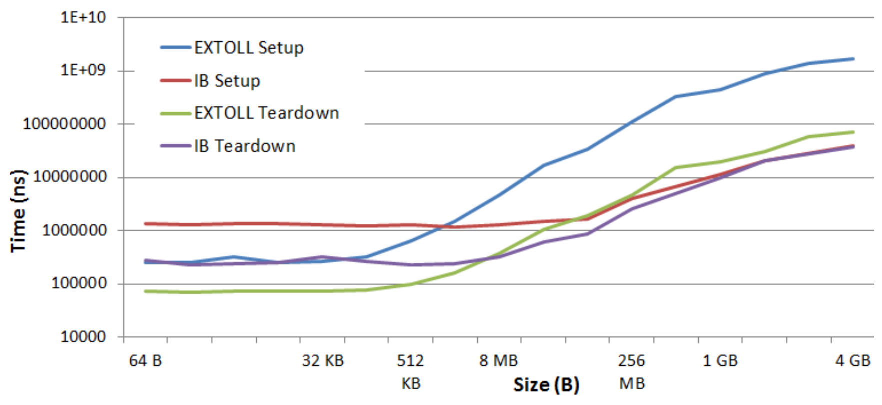
Before evaluating the applications, the Oncilla framework was used to perform a characterization of the allocation and deallocation characteristics of the EXTOLL and InfiniBand networks. These allocation and deallocation tests looked at timing statistics for the creation of symmetric local and remote buffers, and they give insight into which fabric might be best suited for a certain type of allocation.

7.5.1 Oncilla Control Path

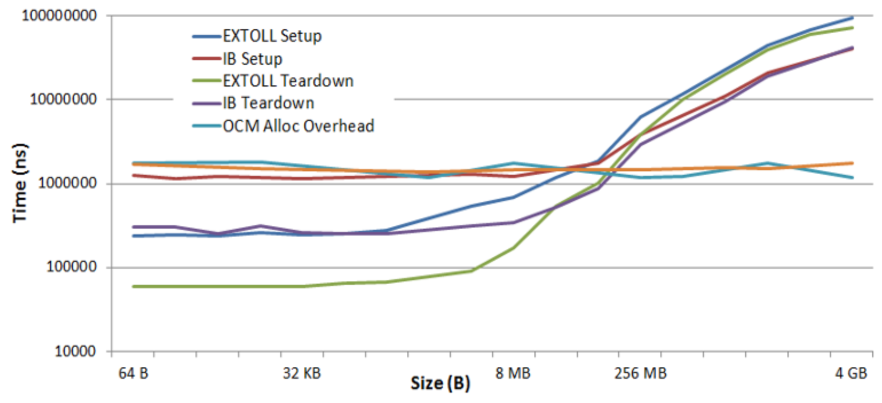
As Figure 46a shows, the EXTOLL network excels at small allocations while the InfiniBand adapter provides quicker allocations of large memory regions. For the smallest allocation, 64 B, EXTOLL takes 250.75 μs versus 1.35 ms in the IB case. However, for a 4 GB allocation, EXTOLL takes 1.67 s while InfiniBand takes 40.54 ms. Since both adapters must register or pin pages in the OS to provide non-coherent buffers for one-sided put/get operations, the difference in performance can be found in the design and optimization of the respective kernel drivers, including support for Linux features like huge pages. For instance, the EXTOLL 4 GB allocation takes 94.89 ms to be registered with the OS while the IB allocation completes the registration process in 38.88 ms. However, for a 64 B allocation EXTOLL requires just 3.10 μs as compared to 37.68 μs for IB. These results indicate that EXTOLL is faster for allocations smaller than 4-8 MB while IB is faster for larger allocations.

Figure 46b shows timing for EXTOLL and IB allocate and free operations on the client, as well as the overhead for making an Oncilla allocation for varying sizes of memory regions. EXTOLL takes from 242.66 μs to 95.47 ms to allocate a local buffer and connect to a remote allocation while InfiniBand takes between 1.26 ms and 40.80 ms for the same memory region sizes. Likewise, teardown for EXTOLL takes between 59.12 μs and 73.28 ms to tear down a 64 B or 4 GB region while InfiniBand takes between 301.42 μs and 42.15 ms.

Oncilla overhead for the allocation of remote memory regions is consistently around 1.18 ms for the two-node case while deallocation of memory regions takes from 1.45 to 1.55 ms. This overhead is due to the time taken for transmission of POSIX and TCP/IP messages that are sent between the Oncilla daemons, so it is not affected by the one-sided setup and teardown that takes place for a server and client allocation. While this overhead is substantial for very small allocations, the use of memory regions larger than 8 MB greatly amortizes the Oncilla setup and teardown overhead. Local host and GPU allocations that



(a) Server



(b) Client

Figure 46: EXTOLL and IB Allocation Time versus Size

use the Oncilla framework also incur a small overhead penalty, on the order of 1 ms, but this overhead allows for tracking of cluster-wide resource allocations by a master daemon that can then make informed decisions about allocation based on the available resources in the cluster.

The use of the Oncilla framework also allows for easy comparison of hardware bandwidths for put and get operations, as shown in Figure 47. The EXTOLL network adapter is based on an FPGA running at 156 MHz, so its realized bandwidth is limited to ∼7 Gbps (9-10 Gbps, theoretical) while the ASIC-based IB QDR adapter has a maximum realized bandwidth of ∼20 Gbps (32 Gbps, theoretical). Despite the frequency limitations of the EXTOLL adapter, this diagram also illustrates its suitability for transmission of messages smaller than 8 KB, which is discussed in more detail in [53].

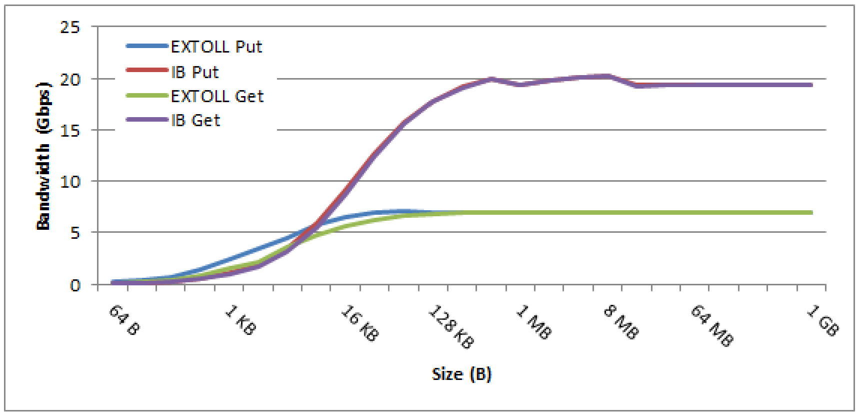
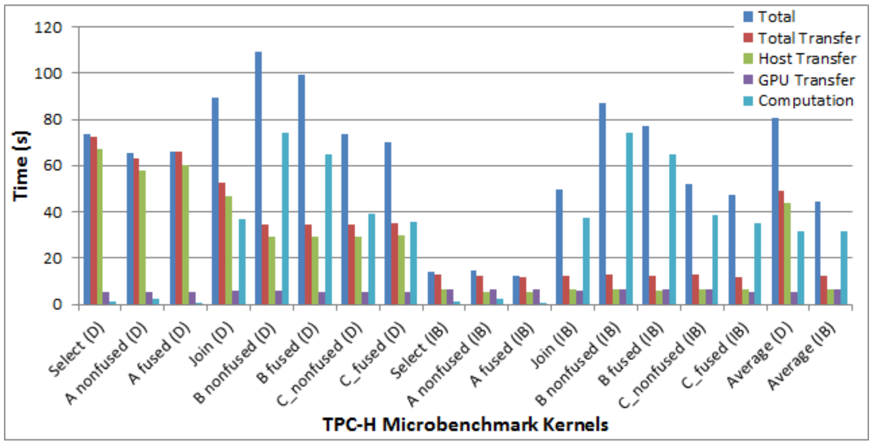


Figure 47: IB and EXTOLL Bandwidth Comparison

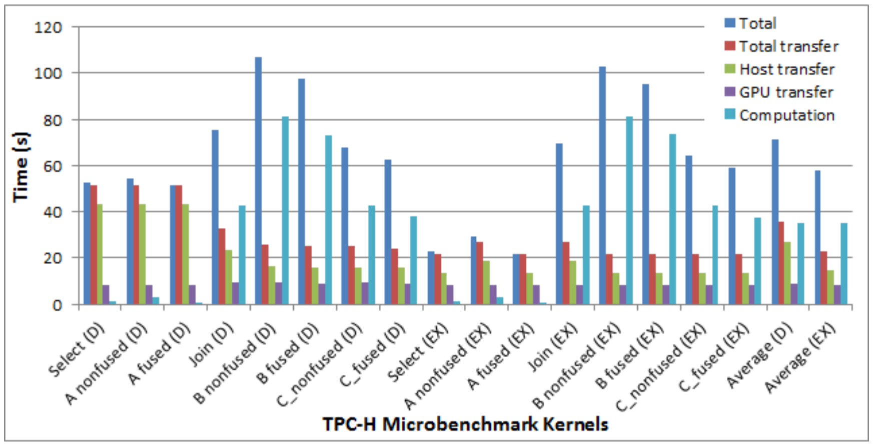
7.5.2 Oncilla as an Aggregation Fabric - TPC-H Microbenchmarks

Figures 48a and 48b show results for using Oncilla to aggregate host memory for a data warehousing application, such as TPC-H. Computation time for the kernels on the GTX 670 ranges from 0.17 seconds (A_Fused) up to 74.2 seconds (B_Nonfused), and the fused variants require between 2.21 (A) and 9.36 seconds (C) less computation time to process 24 GB of input data. The use of remote IB memory, as shown in Figure 48a for input data transfer, is between 4.6x to 10.4x faster than reading input data from disk, and the use of EXTOLL is 1.2x to 3.2x faster than transfer from an SSD disk. The use of InfiniBand decreases the total runtime on average by 81% over standard disk (80.88 s vs. 44.20 s) while the use of EXTOLL decreases runtime on average by 22% (71.06 s vs. 58.02 s) when compared to reading from an SSD.

Both figures demonstrate the computational difficulty of doing join operations on a GPU - this is due to the small input size that can be used for inner join on the GPU, and it results in more iterations for the join, B, and C kernels. For the B_nonfused kernel, performing two joins requires 74.2 seconds (GTX 670) to 81.3 seconds (GTX 480) while the data transfer requires just 29.0 seconds and 15.9 seconds, respectively. Benchmarks like A_Fused greatly benefit from the use of GPU shared memory and small output sizes and can be processed with input sizes of 368 MB (GTX 480) or 1000 MB (GTX 670), but



(a) HDD vs. InfiniBand



(b) SSD vs. EXTOLL

Figure 48: TPC-H with Oncilla- and Disk-Based Transfer

complex join-based operations (B_Nonfused) require small input sizes of either 10 or 28 MB.

The realized bandwidth for each of the networking protocols ranges from 5.15 to 7.2 Gbps for EXTOLL and 14.8 to 17.5 Gbps for InfiniBand while disk transfer bandwidths range from 1.5 (HDD) to 5.84 Gbps (SSD), depending on the input chunk size for each benchmark. The high bandwidth performance in the SSD case is representative of the sequential nature of these microbenchmarks where large input sets, representing large data tables, are transferred first to host memory and then to the GPU. The use of random access patterns to select small pieces of a database table would likely favor the in-core implementations using IB and EXTOLL, despite SSD's benefits over standard rotational HDDs.

GPU transfer bandwidths are relatively consistent at between 2.5 to 3 GB/s (GTX 480) and 3.6 to 4.5 GB/s (GTX 670).

7.5.3 Oncilla for HPC Applications - BFS

As a simple example of how Oncilla can be used to enable multi-node resource support for existing applications, the SHOC BFS benchmark (based on the algorithm described in [67]) was migrated to use inputs from remote memory. This migration required just three steps: 1) the replacement of local GPU allocations with Oncilla-managed GPU allocations, 2) the addition of two remote allocations using the Oncilla API, 3) and the replacement of cudaMemcpy calls with calls to *ocm_copy*.

Figure 49 shows the performance overhead of the Oncilla framework when it is used to hold an in-core data store for a standard graph of one million vertices for the SHOC BFS benchmark. The allocation of the edge array and initial adjacency list on a remote node takes about 5.7x (EXTOLL) to 6.1x (IB) longer for the initialization phase and 1.7 to 5.8x longer for teardown. Input data transfer takes 2.3x longer than the baseline case with IB due to its high bandwidth, but EXTOLL in this case takes 5.42x longer due to its lower bandwidth. Overall, BFS with IB and Oncilla is 1.96x slower than the baseline, and EXTOLL is 2.10x slower than its relative baseline. However, comparison of data transfer with an MPI-1 implementation shows that Oncilla data transfer is on par with traditional message passing techniques.

Currently, this benchmark is limited to one iteration with input sizes of 4 MB and 8 MB respectively for the edge array and adjacency list. This small input size makes it difficult to recommend using Oncilla for this particular BFS implementation, but the ease of implementation with the Oncilla framework and reasonable performance characteristics would likely make it more suitable for larger graphs. The poor performance shown in Figure 49 is a direct result of the BFS algorithm being optimized for one node, so implementations that depend on very large graphs (and would be based on MPI) would provide a better point

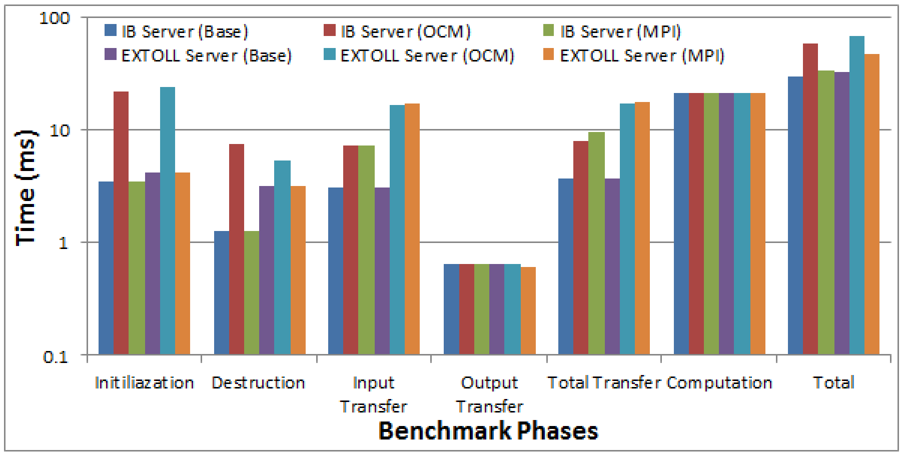


Figure 49: SHOC BFS - Oncilla Overhead

for comparison between Oncilla and a message passing implementation. Future work will focus on creating a multi-node version of BFS based on recent single-node work performed in [104].

7.5.4 Programmability With Oncilla

Figure 50 shows the core functionality of the original SHOC BFS CUDA kernel implemented in the previous section. GPU memory is allocated, data is transferred to the GPU, computation is performed and results are read back (not shown), and then all allocated memory is freed. While any implementation to use remote memory with this kernel would require additional code, the goal of the Oncilla approach is to abstract away some of the programming complexity into the C-oriented library (while the complexity of how to allocate memory resources is handled by the runtime). This section aims to give a brief insight into how Oncilla compares to a related approach, MPI-3's Remote Memory Access API [129], which allocates "private" regions for each MPI process that can then be exposed to local or remote processes using region-specific "windows" for sharing.

Both versions require more setup than the original code, but they differ somewhat in the type of setup that is performed as shown in Figure 51. MPI-RMA uses base data types (unsigned int, int) while Oncilla allocations are considered byte arrays that can be cast to a variety of data types. In addition, Oncilla requires the explicit definition of the type of

```
// Get graph info
unsigned int *edgeArray=G->GetEdgeOffsets();
unsigned int *edgeArrayAux=G->GetEdgeList();
// Get adjacency list length and number of vertices, edges
unsigned int adj_list_length=G->GetAdjacencyListLength();
unsigned int numVerts = G->GetNumVertices();
unsigned int numEdges = G->GetNumEdges();
int sz_edgeArray = sizeof(unsigned int)*(numVerts+1);
int sz_edgeArrayAux = sizeof(unsigned int)*adj_list_length;

// Variables for GPU memory
unsigned int *d_edgeArray=NULL,*d_edgeArrayAux=NULL;

//Allocate GPU input buffers
CUDA_SAFE_CALL(cudaMalloc(&d_edgeArray,sz_edgeArray));
CUDA_SAFE_CALL(cudaMalloc(&d_edgeArrayAux,
                          sz_edgeArrayAux));

//Transfer initial graph information to GPU
CUDA_SAFE_CALL(cudaMemcpy(d_edgeArray, edgeArray,
                          sz_edgeArray,cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy(d_edgeArrayAux,edgeArrayAux,
                          sz_edgeArrayAux,cudaMemcpyHostToDevice));

//Run the benchmark
//===========CUDA kernel call===============
//Copy back the results from the GPU

//Destroy allocated objects
CUDA_SAFE_CALL(cudaFree(d_edgeArray));
CUDA_SAFE_CALL(cudaFree(d_edgeArrayAux));
```

Figure 50: SHOC BFS Single-node Implementation

location of an allocation, for instance “local GPU”. Finally, note that MPI makes use of the standard MPI_Barrier function to synchronize the two processes that are required to set up a remote allocation. While Oncilla creates multiple processes underneath the runtime, these are not explicitly available to the application, so all memory accesses are necessarily blocking. Also, the specific interconnect used with Oncilla is selected before each remote allocation takes place. This enables use cases where one remote memory location might be shared using IB or Ethernet while another is shared using EXTOLL for performance or reliability reasons.

For the transfer of data from remote memory to GPU (Figure 52), Oncilla greatly simplifies the transfer between remote memory (IB, EXTOLL, or another interconnect) and local GPU memory since the library incorporates common functionality underneath its *ocm_copy* API call. Alternatively, MPI-RMA performs more synchronization for transfers and provides better protection of remote memory through the use of *MPI_Win_Lock*.

Finally both Oncilla and MPI-RMA use their respective API calls to free local GPU

```
// MPI-related objects
MPI_Win      win, win2;
unsigned int *edgeArrayMpi = NULL, *edgeArrayAuxMpi = NULL;
// Get the MPI rank of this process
int rank = op.getOptionInt("rank");

// Allocate the MPI-related buffers on each node
MPI_Alloc_mem(sz_edgeArray, MPI_INFO_NULL, &edgeArrayMpi);
MPI_Alloc_mem(sz_edgeArrayAux, MPI_INFO_NULL, &edgeArrayAuxMpi);

// Create two RMA windows to allow access to allocated memory
MPI_Win_create(edgeArrayMpi, sz_edgeArray, sizeof(unsigned int),
                MPI_INFO_NULL, MPI_COMM_WORLD, &win);
MPI_Win_create(edgeArrayAuxMpi, sz_edgeArrayAux, sizeof(unsigned int),
                MPI_INFO_NULL, MPI_COMM_WORLD, &win2);

// Initialize remote memory with Graph data

// Wait for both ranks to reach the barrier before
// operating on the created RMA window
MPI_Barrier(MPI_COMM_WORLD);
```

(a) MPI initialization

```
// Oncilla remote allocations
ocm_alloc_t ocm_edgeArray, ocm_edgeArrayAux;
// Oncilla GPU allocations
ocm_alloc_t d_edgeArray, d_edgeArrayAux;

//Struct used for allocation options
ocm_alloc_param_t alloc_params;
alloc_params = (ocm_alloc_param_t)calloc(1,
                sizeof(struct ocm_alloc_params));

//Struct used for copy options
ocm_param_t copy_params;
copy_params = (ocm_param_t)calloc(1,
                sizeof(struct ocm_params));

//Create two remote allocations with Oncilla
alloc_params->local_alloc_bytes = sz_edgeArray;
alloc_params->rem_alloc_bytes = sz_edgeArray;
alloc_params->kind = OCM_REMOTE_RDMA;
ocm_edgeArray = ocm_alloc(alloc_params);

//Create GPU allocation 1
alloc_params->kind= OCM_LOCAL_GPU;
d_edgeArray=ocm_alloc(alloc_params);

//Create remote allocation 2
alloc_params->local_alloc_bytes = sz_edgeArrayAux;
alloc_params->rem_alloc_bytes = sz_edgeArrayAux;
ocm_edgeArrayAux = ocm_alloc(alloc_params);

//Create GPU allocation 2
alloc_params->kind= OCM_LOCAL_GPU;
d_edgeArrayAux=ocm_alloc(alloc_params);
```

(b) Oncilla initialization

Figure 51: SHOC Breadth First Search Initialization

```
// Initialize remote memory with Graph data

// Wait for both ranks to reach the barrier before
// operating on the created RMA window
MPI_Barrier(MPI_COMM_WORLD);

if(rank == 1)
{
  //Rank 1 proceeds to the barrier
}
else if (rank == 0)
{
  // Allocate GPU input buffers
  CUDA_SAFE_CALL(cudaMalloc(&d_edgeArray,sz_edgeArray));
  CUDA_SAFE_CALL(cudaMalloc(&d_edgeArrayAux,
        sz_edgeArrayAux));

  // Transfer graph information to GPU from remote memory,
  //making sure to read exclusively
  MPI_Win_lock(MPI_LOCK_EXCLUSIVE, rank, 0, win);
  MPI_Get(edgeArrayMpi, sz_edgeArray, MPI_UNSIGNED, 1, 0,
                   sz_edgeArray, MPI_UNSIGNED, win);
  MPI_Win_unlock(rank, win);

  MPI_Win_lock(MPI_LOCK_EXCLUSIVE, rank, 0, win2);
  MPI_Get(edgeArrayAuxMpi, sz_edgeArrayAux, MPI_UNSIGNED, 1, 0,
                   sz_edgeArrayAux, MPI_UNSIGNED, win2);
  MPI_Win_unlock(rank, win2);

  // Standard cudaMemcpy
  CUDA_SAFE_CALL(cudaMemcpy(d_edgeArray, edgeArrayMpi,
        sz_edgeArray,cudaMemcpyHostToDevice));
  CUDA_SAFE_CALL(cudaMemcpy(d_edgeArrayAux,edgeArrayAuxMpi,
        sz_edgeArrayAux,cudaMemcpyHostToDevice));

  //Run the benchmark
  //===========CUDA kernel call===============
  //Copy back the results from the GPU
```

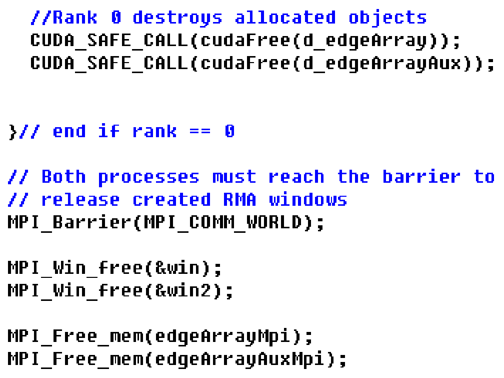
(a) MPI transfer

```
// Initialize remote memory with Graph data

// Transfer graph information to GPU from remote memory
copy_params->src_offset = 0;
copy_params->dest_offset = 0;
//Write operation to destination pointer
copy_params->op_flag = 1;

//Write to the GPU from remote memory
copy_params->bytes = sz_edgeArray;
ocm_copy(d_edgeArray, ocm_edgeArray, copy_params);

copy_params->bytes = sz_edgeArrayAux;
ocm_copy(d_edgeArrayAux, ocm_edgeArrayAux, copy_params);

//To maintain existing CUDA code, we can use "ocm_localbuf" to
//get C-style pointers to the buffer inside the Oncilla object
void *d_edgeArray_buf, *d_edgeArrayAux_buf;
size_t buf_len;
ocm_localbuf(d_edgeArray, &d_edgeArray_buf, &buf_len);
ocm_localbuf(d_edgeArrayAux, &d_edgeArrayAux_buf, &buf_len);

//Run the benchmark
//===========CUDA kernel call===============
//Copy back the results from the GPU
```

(b) Oncilla transfer

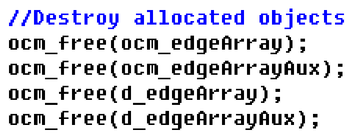
Figure 52: SHOC Breadth First Search Data Transfer

memory and remote memory (Fig. 53). The core functionality of this kernel takes 44 lines to implement with Oncilla and 49 to implement with MPI. In conclusion, Oncilla requires more lines of code for initialization, but provides easier transfer semantics, especially between memories of different types (e.g., host vs. GPU vs. remote memory). Oncilla also uses blocking semantics for data transfer while MPI-RMA relies on barriers to synchronize multiple processes.



```
  //Rank 0 destroys allocated objects
  CUDA_SAFE_CALL(cudaFree(d_edgeArray));
  CUDA_SAFE_CALL(cudaFree(d_edgeArrayAux));


}// end if rank == 0

// Both processes must reach the barrier to
// release created RMA windows
MPI_Barrier(MPI_COMM_WORLD);

MPI_Win_free(&win);
MPI_Win_free(&win2);

MPI_Free_mem(edgeArrayMpi);
MPI_Free_mem(edgeArrayAuxMpi);
```

(a) MPI teardown



```
//Destroy allocated objects
ocm_free(ocm_edgeArray);
ocm_free(ocm_edgeArrayAux);
ocm_free(d_edgeArray);
ocm_free(d_edgeArrayAux);
```

(b) Oncilla teardown

Figure 53: SHOC Breadth First Search Teardown

This particular CUDA kernel is too small to provide a detailed analysis of code complexity between the Oncilla and MPI approach. Even so, an examination of each code implementation can be used to speculate on when each framework should be used. MPI-RMA is suitable for applications that already are designed for the message passing (i.e., multi-process) paradigm and that have strict coherence requirements amongst nodes. MPI provides more fine-grained control over who can access remote memory - good for multiple processes but not really necessary for a single, aggregated application like TPC-H or this particular implementation of BFS. Also, Oncilla provides a similar model to GASNet where the data transfer model is abstracted away from the interconnect that is used.

For instance, MPI-RMA could be used as one of the high-performance "interconnects" for Oncilla's data path.

7.6 Further optimizations and cluster scalability

Important to the discussion of resource management is the use of Oncilla as a portability framework. The same application code that is used to enable multi-node memory aggregation on an IB-based cluster can then be moved to an EXTOLL-based cluster and run without any further modifications. Similar to how open-source frameworks like OpenStack [32] focus on running virtualized applications on a variety of heterogeneous clusters, Oncilla provides a path for application developers to design high-performance applications that can operate on clusters with heterogeneous network architectures. One example would be to use the Oncilla stack to choose, at run-time, the desired networking fabric on a per-allocation basis, dependent on application characteristics and on available networking fabrics. For instance, small allocations and put/get operations might work best with the EXTOLL RMA or VELO protocols, while large, bandwidth-intensive transfers might be more suited for InfiniBand.

To this end, the Oncilla software stack must also be capable of optimizing data movement across host-to-host, host-to-GPU, and GPU-to-GPU data transfers as shown in Figure 54. Future work with Oncilla will support CUDA 5 and related hardware, which can take advantage of the Peer-to-Peer API (aka GPUDirect RDMA) to transfer data directly from a NIC to the GPU and to reduce inter-node overheads.

Experimental results demonstrate that Oncilla provides a high-performance abstraction for allocating memory on remote nodes and transferring it to accelerators, but many data warehousing applications may scale to several terabytes of data. While the test infrastructure is limited in its installed memory, typical servers can be provisioned with between 256 and 512 GB of host memory. For clusters that are built around a 3D torus interconnect, a node might have up to six nodes one hop away and up to 30 nodes one to two hops

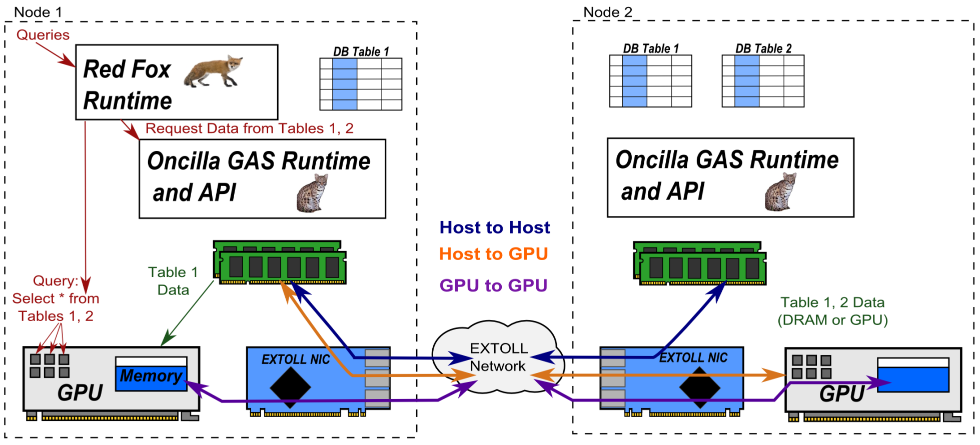


Figure 54: Oncilla Optimized Data Transfer for Accelerators

away. This translates to potentially having 3 - 15 TB of DRAM that can be incorporated into the global address space and that could contain many data warehousing applications in-core [79]. Oncilla further improves the performance and ease of use of these clusters through low-overhead allocations and high-performance data movement.

7.7 *Related Work*

Other recent work has also focused on memory management for in-core business applications. For example, the MVAPICH group has pushed forward in implementing RDMA support for core applications, including HDFS [83]. The Oncilla framework differs from this work in that it focuses on providing a low overhead but network agnostic path for doing remote memory allocations and data transfer. This approach has performance penalties with respect to a straight-forward use of the IB Verbs API, but it also offers a simpler conversion path for using a variety of networking substrates and additional support for remote resource management.

As mentioned previously in Section 7.1, other projects that have focused on using global address spaces to share cluster resources include MEMSCALE [108], rCUDA [40], and APENET+ [15]. Much of the work for each of these projects is focused on providing support for high-performance computing applications, which means that potential applications can be rewritten to take advantage of the high-performance aspects of InfiniBand [135] and CUDA optimizations [16]. Related projects in the GPU virtualization space include DS-CUDA [124], which uses a middleware to launch HPC jobs across many GPUs in a cluster. Oncilla aims to provide similar high-performance data movement while also focusing on a better allocation of resources and better support for a wide variety of accelerator and network resources, as opposed to optimizing for one specific networking fabric.

Finally, projects like RAMCloud [125] are focused on providing large, in-core memory allocations using standardized protocols. Currently, RAMCloud focuses only on host memory and does not incorporate accelerator resources.

7.8 Oncilla and Other Application Domains

The Oncilla framework was demonstrated for a specific type of application, data warehousing, which benefits from one (possibly multi-threaded) application having a large amount of aggregated memory available to store data “in-core”. However, this framework builds on the DPGAS model, and both Oncilla and the spill/receive model of memory sharing are dependent on low-latency access to remote memory and the extension of an application’s virtual memory via “virtual DIMMs” or more explicit aggregation of accelerators or memory.

While the discussed TPC-H and BFS applications did not involve time-varying workloads, it would be relatively easy to add a cost-based allocation model to Oncilla’s runtime to enable a dynamic sharing of memory and accelerator resources. One caveat is that memory sharing in this model would either be restricted or require additional software constraints to enforce consistency. In addition, providing dynamic sharing of accelerator resources would likely require coordinated scheduling of tasks for using these accelerators as examined in recent research [134].

Next, several other common applications are evaluated and discussed with respect to how Oncilla can be extended to support workloads in the business and HPC arenas.

7.8.1 Business Applications

In the business space, there are at least two large systems that could possibly benefit from Oncilla's focus on high-performance data movement, a Hadoop implementation and Facebook's TAO. Both of these applications have limited consistency requirements and require high-performance read operations from the memories of nearby nodes.

Hadoop and other software that supports the MapReduce algorithm typically use repeated "merge" operations as part of the reduce step, and this operation can restrict overall performance [163]. Recent work on an RDMA-enabled version of Hadoop, Hadoop-A, uses RDMA to replace TCP/IP-based requests and as the basis for a more efficient merge algorithm. By using the Oncilla API, Hadoop-A could be modified to work with interconnects other than native InfiniBand, and Oncilla's awareness of accelerator memory could make Hadoop-A easier to integrate with existing GPU-based techniques for Hadoop queries, such as those in [149].

Facebook's TAO [158] is an alternative to the Memcache lookaside-caching architecture that is optimized to store a persistent copy of the social graph and optimize read operations according to customized parameters (e.g., privacy settings for a certain Facebook post). TAO uses a multi-level caching hierarchy of "leaders" and "followers" with eventually consistent semantics where writes are forwarded to a "leader" but reads can be satisfied by any servers in a nearby "follower" tier. The Oncilla framework would not be very useful for the traditional Facebook architecture, due to its focus on somewhat independent memcached servers, but the performance of the graph search algorithm used for TAO could benefit from having short-lived, low-latency connections between servers in co-located "follower" tiers. At the same time, the Oncilla runtime would likely have to be modified to match up more closely with TAO's master and shared regions that might span multiple data centers. Assuming that basic location information is available, one approach could be to allocate 10GE connections between servers in the same data center and use TCP/IP for inter-data center requests. One benefit of using Oncilla with TAO is that its eventually consistent design relies on the API to filter update messages down to followers and to update in-core graph shards, so Oncilla's lack of consistency support does not affect TAO performance.

7.8.2 HPC Applications

HPC applications pose different technical challenges since many are currently based on the message-passing paradigm. Some applications like NWChem [60] have been migrated to use software built on top of global address models like Global Arrays, but other applications like the NAS parallel benchmarks have proved more difficult to port to GAS models using software like UPC. This difficulty stems from the need for either all-to-all exchanges or several of the included kernels require large-scale synchronization via barriers [45]. For instance, in the Block Tri-diagonal solver (BT) benchmark data is partitioned into blocks that must be exchanged with neighbors before the next time step can proceed. However, a framework like Oncilla can also build on recent work [142] that shows that many of these applications can be as performant as MPI variations if they incorporate relaxed semantics and make use of point-to-point synchronizations, i.e., synchronizing only between neighbors exchanging data. Again, the use of a DPGAS implementation like Oncilla would require the addition of some sort of synchronization mechanism to enforce a relaxed consistency model.

While transitioning these business and HPC applications to Oncilla would not necessarily be a trivial process, recent work in using related implementations based on RDMA and PGAS-based languages indicates that migration to Oncilla's simpler programming model is possible to do in a manner that preserves performance. Although many multi-node business and HPC applications are in the early stages of incorporating accelerators, Oncilla's focus on solving the problem of aggregating both accelerators and memory would provide benefits that might not be available to standard PGAS implementations.

7.9 Concluding Remarks

The Oncilla framework and associated managed GAS system model provide an important addition to existing research into scheduling and data placement for emerging heterogeneous clusters. Due to the growth of large data sets for Big Data applications, like data warehousing, and the increasing availability of accelerator clouds, such as EC2, it is important to be able to manage both host and accelerator resources across a variety of cluster infrastructures. This work focuses on two such networking infrastructures, EXTOLL and InfiniBand, and the use of CUDA-based accelerators, but future heterogeneous clusters might include multiple networks and OpenCL, CUDA, or custom accelerators. Regardless of the accelerators or interconnects that are incorporated into future data centers, Oncilla provides a flexible software framework that allows for the aggregation of host and accelerator memories and support forks new technologies with minimal changes to existing software.

CHAPTER VIII

CONCLUSION

In this chapter, future extensions to this thesis are proposed and a brief summary of the work completed is presented.

8.1 Future Work

Future extensions for this work can be divided into two main categories: hardware- and software-related implementations of the DPGAS model.

8.1.1 Hardware Extensions

HyperTransport over Ethernet as discussed in this work provides a high-performance hardware implementation of the DPGAS model, but its ultimate adoption as an industry standard may be limited by AMD's market share and the lack of HTX-based motherboards. Alternative interconnects like Cray's Aries [3] provide a tightly integrated PCI Express network with low-latency capabilities, but this type of hardware is likely too expensive to gain traction in data center designs. The most likely future extension that could provide similar hardware characteristics to the HToE implementation is the combined use of switched PCI Express and PCI Express over Ethernet in the data center. Along with continued integration with current CPU northbridges, PCI Express 3.0 has introduced new features for better support of virtualization and latency-specific optimizations like TLP processing hints [102] that could be used to provide non-coherent memory-to-memory transactions. The addition of a PCI Express over Ethernet component would improve PCIe's scalability and provide for easier integration into existing Ethernet-based data centers [101].

The research performed in this thesis to support HyperTransport over Ethernet illustrates that any converged hardware solution (PCI Express over Ethernet or otherwise) would need to provide the following: 1) Hardware or software support that implements the required consistency model for target applications. 2) An adapter that can be reprogrammed to support changes to the DPGAS mapping of remote memory and that possibly can support different packet processing profiles for different applications (e.g., with accelerator clouds). 3) A customized flow control and retry protocol that leaves the on-board interconnect unchanged but that works seamlessly with existing off-board interconnects. This thesis presented the use of unmodified HT links with standard, "lossy", L2 Ethernet, and similar features would be vital in bringing about the adoption of new converged fabrics in existing data centers.

8.1.2 Software Extensions

In terms of software extensions, the Oncilla framework provides a good starting point for future research using either managed GAS or other hardware built to support the DPGAS model.

The most important research area that has currently been under-explored relates to scheduling of tasks across a large number of accelerators with an emphasis on mapping aggregated resources to scheduled tasks. For instance, many schedulers assume complete control over resources on nodes where computation is being placed, but the DPGAS model enables the opportunity for sharing underutilized resources (e.g., DRAM on a node scheduled with many GPU-intensive tasks) on a time-limited basis. The availability of easier sharing of resources also means that scheduling and resource allocation must be tied together much more closely. Oncilla currently does not focus on scheduling computation, but this area of research is one that would greatly benefit from integration with the resource allocation and data movement capabilities offered by the Oncilla runtime.

In addition to scheduling, another interesting area of research relates to how applications are placed in cloud computing environments. Current research focuses on how to place computation [32] in clusters where the relative locations of nodes are difficult to determine, but there has been little research in applying this concept to resource aggregation. From an application standpoint, aggregated resources from nodes that are many hops away (or possibly even served by a different data center) can change the performance dynamics and in some cases can negatively affect an application's performance. Future work in this area could be used to determine nearest neighbors for dynamically allocated nodes and to manage application placement in a fashion that optimizes the latency of data movement between applications with memory resources mapped across multiple nodes.

8.2 Summary

In this thesis, a new model for cluster-based resource management was proposed to test the following thesis statement: *Global address spaces built using commodity parts can achieve efficient memory provisioning for data center applications by supporting high-performance and low-cost data movement*. Through the definition of the Dynamic Partitioned Global Address Space model and implementations using HyperTransport over Ethernet and managed GAS, this thesis has demonstrated that high-performance GAS support for data centers can be built using commodity interconnects. In addition, these implementations have demonstrated good performance characteristics and have been shown to have potential for power and cost savings when used in a data center.

To summarize, this thesis has presented the following contributions:

- A dynamic, hardware-based GAS model for resource sharing (DPGAS)
- Hardware implementations of this model based on commodity interconnects (HToE, managed GAS)
- A technique for reduce memory overprovisioning in data centers (DPGAS spill/receive)
- A combined mechanism for aggregating both accelerator and host memory

- Software support for applying high-performance GAS to business applications

These contributions provide an abstract model that future data center designers and developers can take advantage of, and this abstraction is supplemented by concrete examples of hardware and software support for the DPGAS model. The evaluation in this thesis is useful in that it provides examples of how real-world applications can have a performance and power impact not just on one blade in a cluster but beyond the boundaries of a single node's compute and memory resources. The related focus on fully utilizing interconnects to manage the aggregate resources of the cluster is one of the most interesting contributions of this work and hopefully one that will be further improved by future research.

REFERENCES

[1] 802.3AE 10GB/S ETHERNET TASK FORCE, I., "10 gigabit ethernet 802.3ae standard," 2002. `http://grouper.ieee.org/groups/802/3/ae/index.html`.

[2] AJI, A. M., DINAN, J., BUNTINAS, D., BALAJI, P., FENG, W.-C., BISSET, K. R., and THAKUR, R., "MPI-ACC: an integrated and extensible approach to data movement in accelerator-based systems," in *14th IEEE International Conference on High Performance Computing and Communications*, (Liverpool, UK), June 2012.

[3] ALVERSON, B., FROESE, E., KAPLAN, L., and ROWETH, D., "Cray XC series network (white paper)," 2012.

[4] *BIOS and Kernel Developer's Guide (BKDG) For AMD Family 11h Processors*. 2008. `http://support.amd.com`.

[5] ARISTA NETWORKS, "Arista Networks 7150S-52 10GE router (data sheet)," 2013. `http://www.aristanetworks.com/en/products/7150-series/7150-datasheet`.

[6] ASSOCIATION, I. T., "RDMA over Converged Ethernet specification," 2010. `http://www.infinibandta.org`.

[7] BADER, D. A. and MADDURI, K., "Design and implementation of the HPCS graph analysis benchmark on symmetric multiprocessors," in *HiPC*, pp. 465–476, 2005.

[8] BAKKUM, P. and SKADRON, K., "Accelerating SQL database operations on a GPU with CUDA," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU '10, (New York, NY, USA), pp. 94–103, ACM, 2010.

[9] BALAJI, P., FENG, W.-C., and PANDA, D. K., "Bridging the Ethernet-Ethernot performance gap," *IEEE Micro*, vol. 26, pp. 24–40, May 2006.

[10] BARKER, K. J., DAVIS, K., HOISIE, A., KERBYSON, D. J., LANG, M., PAKIN, S., and SANCHO, J. C., "Entering the petaflop era: The architecture and performance of Roadrunner," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, 2008.

[11] BARROSO, L. and HOLZLE, U., "The case for energy-proportional computing," *Computer*, vol. 40, pp. 33 –37, Dec. 2007.

[12] BAUER, M., COOK, H., and KHAILANY, B., "CudaDMA: Optimizing GPU memory bandwidth via warp specialization," SuperComputing 2011.

[13] BELL, C., CHEN, W.-Y., BONACHEA, D., and YELICK, K., "Evaluating support for global address space languages on the Cray X1," ICS '04, (New York, NY, USA), pp. 184–195, ACM, 2004.

[14] BELOGLAZOV, A., ABAWAJY, J., and BUYYA, R., "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Gener. Comput. Syst.*, vol. 28, pp. 755–768, May 2012.

[15] BERNASCHI, M., BISSON, M., and ROSSETTI, D., "Benchmarking of communication techniques for GPUs," *J. Parallel Distrib. Comput.*, 2013.

[16] BERNASCHI, M., BISSON, M., MASTROSTEFANO, E., and ROSSETTI, D., "Breadth first search on APEnet+," in *SC Companion (SCC)*, 2012.

[17] BILLI, E., YOUNG, J., and HOLDEN, B., "HyperTransport over InfiniBand specification, 1.0," 2011. `http://www.hypertransport.org`.

[18] BINKERT, N. L., HSU, L. R., SAIDI, A. G., DRESLINSKI, R. G., SCHULTZ, A. L., and REINHARDT, S. K., "Performance analysis of system overheads in TCP/IP workloads," in *Parallel Architectures and Compilation Techniques, 14th International Conference on*, pp. 218–228, 2005.

[19] BLUMRICH, M. A., ALPERT, R. D., CHEN, Y., CLARK, D. W., DAMIANAKIS, S. N., DUBNICKI, C., FELTEN, E. W., IFTODE, L., LI, K., MARTONOSI, M., and SHILLNER, R. A., "Design choices in the SHRIMP system: An empirical study," *SIGARCH Comput. Archit. News*, vol. 26, Apr. 1998.

[20] BODEN, N., COHEN, D., FELDERMAN, R., KULAWIK, A., SEITZ, C., SEIZOVIC, J., and SU, W.-K., "Myrinet: A gigabit-per-second local area network," *Micro, IEEE*, vol. 15, pp. 29 –36, Feb. 1995.

[21] BONACHEA, D., "GASNet specification, v1.1," tech. rep., 2002.

[22] BRIGHTWELL, R., PEDRETTI, K., and UNDERWOOD, K., "Initial performance evaluation of the Cray SeaStar interconnect," pp. 51 – 57, Aug. 2005.

[23] BRUENING, U., "HTX board: Universal HTX test platform (presentation at ISC)," 2006. `http://ra.ziti.uni-heidelberg.de/pages/projects/htx/isc2006_HTC_website.pdf`.

[24] CARLSON, B., EL-GHAZAWI, T., NUMRICH, R., and YELICK, K., "Programming in the Partitioned Global Address Space model tutorial," 2003.

[25] CHALAL, S. and GLASGOW, T., "Memory sizing for server virtualization (white paper)," 2007. `http://communities.intel.com/docs/`.

[26] CHAPMAN, M. and HEISER, G., "vNUMA: a virtual shared-memory multiprocessor," in *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09, (Berkeley, CA, USA), p. 22, USENIX Association, 2009.

[27] CHASE, J. S., LEVY, H. M., FEELEY, M. J., and LAZOWSKA, E. D., "Sharing and protection in a single-address-space operating system," *ACM Trans. Comput. Syst.*, vol. 12, pp. 271–307, Nov. 1994.

[28] CHELSIO COMMUNICATIONS, "Preliminary ultra low latency report (white paper)," 2013. `http://www.chelsio.com/wp-content/uploads/2011/05/Ultra-Low-Latency-Report-0408131.pdf`.

[29] COARFA, C., DOTSENKO, Y., MELLOR-CRUMMEY, J., CANTONNET, F., EL-GHAZAWI, T., MOHANTI, A., YAO, Y., and CHAVARRÍA-MIRANDA, D., "An evaluation of global address space languages: Co-array Fortran and Unified Parallel C," in *PPoPP '05*, (New York, NY, USA), pp. 36–47, ACM, 2005.

[30] COHEN, D., TALPEY, T., KANEVSKY, A., CUMMINGS, U., KRAUSE, M., RECIO, R., CRUPNICOFF, D., DICKMAN, L., and GRUN, P., "Remote Direct Memory Access over the Converged Enhanced Ethernet fabric: Evaluating the options," in *Hot Interconnects*, 2009.

[31] CONWAY, P. and HUGHES, B., "The AMD Opteron Northbridge architecture," *IEEE Micro*, vol. 27, no. 2, pp. 10–21, 2007.

[32] CRAGO, S., DUNN, K., EADS, P., HOCHSTEIN, L., KANG, D.-I., KANG, M., MODIUM, D., SINGH, K., SUH, J., and WALTERS, J., "Heterogeneous cloud computing," in *CLUSTER*, 2011.

[33] CULLER, D., DUSSEAU, A., GOLDSTEIN, S., KRISHNAMURTHY, A., LUMETTA, S., VON EICKEN, T., and YELICK, K., "Parallel programming in Split-C," in *Supercomputing '93. Proceedings*, pp. 262 – 273, Nov. 1993.

[34] CUMMINGS, U., "Focalpoint: A low-latency, high-bandwidth ethernet switch chip," in *Hot Chips 18*, 2006. `http://www.hotchips.org/archives/hc18/3_Tues/HC18.S8/HC18.S8T1.pdf`.

[35] DALESSANDRO, D., WYCKOFF, P., and MONTRY, G., "Initial performance evaluation of the NetEffect 10 Gigabit iWARP adapter," in *Cluster Computing*, 2006.

[36] DANALIS, A., MARIN, G., MCCURDY, C., MEREDITH, J. S., ROTH, P. C., SPAFFORD, K., TIPPARAJU, V., and VETTER, J. S., "The Scalable HeterOgeneous Computing (SHOC) benchmark suite," in *GPGPU 3*, 2010.

[37] DHABALESWAR K. (DK) PANDA, H. S. and POTLURI, S., "MVAPICH2 and GPUDirect RDMA (presentation)," *HPC Advisory Council, June 2013*, 2013.

[38] DIAMOS, G., WU, H., WANG, J., LELE, A., and YALAMANCHILI, S., "Relational algorithms for multi-bulk-synchronous processors," *PPoPP*, 2013.

[39] "Dis stressmark suite, updated by uc irvine," 2001. `http://www.ics.uci.edu/~amrm/hdu/DIS_Stressmark/DIS_stressmark.html`.

[40] DUATO, J., PENA, A., SILLA, F., FERNANDEZ, J., MAYO, R., and QUINTANA-ORTI, E., "Enabling CUDA acceleration within virtual machines using rCUDA," *HiPC*, 2011.

[41] DUATO, J., PENA, A., SILLA, F., MAYO, R., and QUINTANA-ORTI, E., "rCUDA: Reducing the number of GPU-based accelerators in high performance clusters," in *HPCS*, July 2010.

[42] DUATO, J. and OTHERS, "Scalable computing: Why and how (white paper)," 2010. `http://www.hypertransport.org`.

[43] DUNNING, D., REGNIER, G., MCALPINE, G., CAMERON, D., SHUBERT, B., BERRY, F., MERRITT, A., GRONKE, E., and DODD, C., "The Virtual Interface Architecture," *Micro, IEEE*, vol. 18, pp. 66 –76, Apr. 1998.

[44] EICKEN, T., CULLER, D., GOLDSTEIN, S., and SCHAUSER, K., "Active messages: A mechanism for integrated communication and computation," in *Computer Architecture, 1992. Proceedings., The 19th Annual International Symposium on*, pp. 256 –266, 1992.

[45] EL-GHAZAWI, T. and CANTONNET, F., "UPC performance and potential: A NPB experimental study," in *Supercomputing, ACM/IEEE 2002 Conference*, IEEE, 2002.

[46] "Xilinx FPGA simulation with 10 Gbps Ethernet MAC - obtained from tests run by Emilio Billi of EBE," 2012. `http://www.emiliobilli.com/`.

[47] FAN, Z., QIU, F., KAUFMAN, A., and YOAKUM-STOVER, S., "GPU cluster for high performance computing," in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, p. 47, 2004.

[48] "Fibre Channel over Ethernet FC-BB-5 standard," 2010. `http://www.t11.org/fcoe`.

[49] FEELEY, M. J., MORGAN, W. E., PIGHIN, E. P., KARLIN, A. R., LEVY, H. M., and THEKKATH, C. A., "Implementing global memory management in a workstation cluster," *SIGOPS Oper. Syst. Rev.*, vol. 29, pp. 201–212, Dec. 1995.

[50] FRÖNING, H. and LITZ, H., "Efficient hardware support for the Partitioned Global Address Space," *Parallel and Distributed Processing Workshops and PhD Forum, 2011 IEEE International Symposium on*, vol. 0, pp. 1–6, 2010.

[51] FRÖNING, H., MONTANER, H., SILLA, F., and DUATO, J., "On memory relaxations for extreme manycore system scalability," *NDCA-2 Workshop*, 2011.

[52] FRÖNING, H., NÜSSLE, M., LITZ, H., and BRÜNING, U., "A case for FPGA based accelerated communication," in *ICN*, 2010.

[53] FRÖNING, H., NÜSSLE, M., LITZ, H., LEBER, C., and BRÜNING, U., "On achieving high message rates," *CCGRID*, 2013.

[54] G. F. PFISTER, "Introduction to the InfiniBand architecture," in *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, pp. 617–632, Wiley Press, 2001.

[55] GARLAND, M., KUDLUR, M., and ZHENG, Y., "Designing a unified programming model for heterogeneous machines," in *SC*, 2012.

[56] GEIST, G. A., KOHL, J. A., and PAPADOPOULOS, P. M., "PVM and MPI: a comparison of features," *Calculateurs Paralleles*, vol. 8, pp. 137–150, 1996.

[57] GIANNINI, L. A. and CHIEN, A. A., "A software architecture for global address space communication on clusters: put/get on Fast Messages," in *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, HPDC '98, (Washington, DC, USA), IEEE Computer Society, 1998.

[58] GMACH, D., ROLIA, J., CHERKASOVA, L., and KEMPER, A., "Workload analysis and demand prediction of enterprise data center applications," in *Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization*, IISWC '07, pp. 171–180, 2007.

[59] GOVIL, K., TEODOSIU, D., HUANG, Y., and ROSENBLUM, M., "Cellular Disco: Resource management using virtual clusters on shared-memory multiprocessors," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 229–262, 2000.

[60] HAMMOND, J. R., KRISHNAMOORTHY, S., SHENDE, S., ROMERO, N. A., and MALONY, A. D., "Performance characterization of global address space applications: a case study with NWChem," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 2, pp. 135–154, 2012.

[61] HAYASHI, K., DOI, T., HORIE, T., KOYANAGI, Y., SHIRAKI, O., IMAMURA, N., SHIMIZU, T., ISHIHATA, H., and SHINDO, T., "AP1000+: architectural support of PUT/GET interface for parallelizing compiler," *SIGPLAN Not.*, vol. 29, pp. 196–207, Nov. 1994.

[62] HE, B., LU, M., YANG, K., FANG, R., GOVINDARAJU, N. K., LUO, Q., and SANDER, P. V., "Relational query coprocessing on graphics processors," *ACM Trans. Database Syst.*, 2009.

[63] HENNING, J. L., "SPEC CPU2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, 2006.

[64] HILLIS, W. D. and TUCKER, L. W., "The CM-5 connection machine: A scalable supercomputer," *Commun. ACM*, vol. 36, pp. 31–40, Nov. 1993.

[65] HINES, M. R., GORDON, A., SILVA, M., DA SILVA, D., RYU, K., and BEN-YEHUDA, M., "Applications know best: Performance-driven memory overcommit with Ginkgo," in *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, 2011.

[66] HOELZLE, U. and BARROSO, L. A., *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[67] HONG, S., KIM, S. K., OGUNTEBI, T., and OLUKOTUN, K., "Accelerating CUDA graph algorithms at maximum warp," PPoPP '11, 2011.

[68] HP, "HP Project Moonshot (white paper)," 2012.

[69] "HP Proliant DL servers - cost specifications," 2008. http://h18004.www1.hp.com/products/servers/platforms/.

[70] "Interconnect family share of HPC Top 500," 2013. http://www.top500.org.

[71] "HP power calculator utility: A tool for estimating power requirements for HP ProLiant rack-mounted systems," 2008. http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00881066/c00881066.pdf.

[72] "HP power advisor utility: A tool for estimating power requirements for HP ProLiant server systems," 2010. http://h20000.www2.hp.com/bc/docs/support/SupportManual/c01861599/c01861599.pdf.

[73] HUYNH, J., "The AMD Athlon XP processor with 512KB L2 cache (white paper)," 2003.

[74] HYPERTRANSPORT CONSORTIUM, "HTX - PCI Express compared (presentation)," 2008. http://www.hypertransport.org/docs/uploads/HTX3_vs_PCIe_Gen2.pdf.

[75] HYPERTRANSPORT CONSORTIUM, "HyperTransport specification, 3.10," 2008. http://www.hypertransport.org.

[76] HYPERTRANSPORT CONSORTIUM, "Clustering 360 market analysis," 2010. http://www.hypertransport.org/default.cfm?page=Clustering360.

[77] HYPERTRANSPORT CONSORTIUM, "HyperShare technology overview," 2011. http://www.hypertransport.org.

[78] IEEE 802.1 WORKING GROUP, "IEEE 802.1Qaz standards page," http://www.ieee802.org/1/pages/802.1az.html.

[79] INDEPENDENT ORACLE USERS GROUP, "A new dimension to data warehousing: 2011 IOUG data warehousing survey," http://www.oracle.com/us/solutions/datawarehousing/2011-ioug-data-warehousing-survey-522175.pdf.

[80] "Intel 8255 10 Gigabit Ethernet controller datasheet." http://www.intel.com/content/www/us/en/ethernet-controllers/82599-10-gbe-controller-datasheet.html.

[81] "Intel 82541er gigabit Ethernet controller." `http://download.intel.com`.

[82] "Hardware specification for Intel FM2224 10 GE 24 port switch," 2012. `http://ark.intel.com/products/64419/Intel-Ethernet-Switch-FM2224`.

[83] ISLAM, N. S., RAHMAN, M. W., JOSE, J., RAJACHANDRASEKAR, R., WANG, H., SUBRAMONI, H., MURTHY, C., and PANDA, D. K., "High performance RDMA-based design of HDFS over InfiniBand," in *SC 2012*.

[84] JALEEL, A., "Memory characterization of workloads using instrumentation-driven simulation: A Pin-based memory characterization of the SPEC CPU2000 and SPEC CPU2006 benchmark suites," *VSSAD Technical Report 2007*, 2007. `http://www.glue.umd.edu/~ajaleel/workload/`.

[85] JI, F., AJI, A. M., DINAN, J., BUNTINAS, D., BALAJI, P., THAKUR, R., FENG, W.-C., and MA, X., "DMA-assisted, intranode communication in GPU accelerated systems," in *14th IEEE International Conference on High Performance Computing and Communications*, (Liverpool, UK), June 2012.

[86] JIN, H., DENG, L., WU, S., SHI, X., and PAN, X., "Live virtual machine migration with adaptive memory compression," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, 2009.

[87] JUNIPER NETWORKS, "The QFabric architecture: Implementing a flat data center network," 2013. `http://www.juniper.net/us/en/local/pdf/whitepapers/2000443-en.pdf`.

[88] "Juniper Networks T1600 datasheet," 2013. `http://www.juniper.net/us/en/local/pdf/datasheets/1000051-en.pdf`.

[89] KANDADAI, S. N. and HE, X., "Performance of HPC applications over InfiniBand, 10 gb, and 1 gb Ethernet (white paper)," 2010. `http://www.chelsio.com/assetlibrary/whitepapers/HPC-APPS-PERF-IBM.pdf`.

[90] KO, M., EISENHAUER, D., and RECIO, R., "A case for Convergence Enhanced Ethernet: Requirements and applications," in *Communications, 2008. ICC '08. IEEE International Conference on*, May 2008.

[91] KUMAR, K., DOSHI, K., DIMITROV, M., and LU, Y.-H., "Memory energy management for an enterprise decision support system," in *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, pp. 277–282, 2011.

[92] KUSKIN, J., OFELT, D., HEINRICH, M., HEINLEIN, J., SIMONI, R., GHARACHORLOO, K., CHAPIN, J., NAKAHIRA, D., BAXTER, J., HOROWITZ, M., GUPTA, A., ROSENBLUM, M., and HENNESSY, J., "The stanford FLASH multiprocessor," in *Computer Architecture, 1994., Proceedings the 21st Annual International Symposium on*, pp. 302 –313, Apr. 1994.

[93] LEFURGY, C., RAJAMANI, K., RAWSON, F., FELTER, W., KISTLER, M., and KELLER, T. W., "Energy management for commercial servers," *Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[94] LI, K. and HUDAK, P., "Memory coherence in shared virtual memory systems," *ACM Trans. Comput. Syst.*, vol. 7, pp. 321–359, Nov. 1989.

[95] LIANG, S., NORONHA, R., and PANDA, D. K., "Swapping to remote memory over infiniband: An approach using a high performance network block device.," September 2005.

[96] LIM, K. and OTHERS, "Disaggregated memory for expansion and sharing in blade servers," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 267–278, 2009.

[97] LIM, K., RANGANATHAN, P., CHANG, J., PATEL, C., MUDGE, T., and REINHARDT, S., "Understanding and designing new server architectures for emerging warehouse-computing environments," in *ISCA '08*, (Washington, DC, USA), pp. 315–326, IEEE Computer Society, 2008.

[98] LIU, J., WU, J., KINI, S. P., WYCKOFF, P., and PANDA, D. K., "High performance RDMA-based MPI implementation over InfiniBand," in *Proceedings of the 17th annual international conference on Supercomputing*, ICS '03, (New York, NY, USA), pp. 295–304, ACM, 2003.

[99] LUSK, E. and YELICK, K., "Languages for high-productivity computing: The DARPA HPCS language project," *Parallel Processing Letters*, vol. 17, no. 1, pp. 89–102, 2007.

[100] MAGNUSSON, P. S., CHRISTENSSON, M., ESKILSON, J., FORSGREN, D., HÅLLBERG, G., HÖGBERG, J., LARSSON, F., MOESTEDT, A., and WERNER, B., "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[101] MEDURI, V., "A case for PCI Express as a high-performance cluster interconnect (white paper)," 2011. `http://www.hpcwire.com/hpcwire/2011-01-24/a_case_for_pci_express_as_a_high-performance_cluster_interconnect.html`.

[102] MEDURI, V., "PCIe 3.0/2.1 protocol update (presentation)," 2011. Found at `http://www.pcisig.com`.

[103] "Mellanox ConnectX IB specification sheet," 2008. `http://www.mellanox.com`.

[104] MERRILL, D., GARLAND, M., and GRIMSHAW, A., "Scalable gpu graph traversal," in *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pp. 117–128, 2012.

[105] MERRITT, A. M., GUPTA, V., VERMA, A., GAVRILOVSKA, A., and SCHWAN, K., “Shadowfax: Scaling in heterogeneous cluster systems via GPGPU assemblies,” in *Proceedings of the 5th international workshop on Virtualization technologies in distributed computing*, VTDC ’11, p. 310, ACM, 2011.

[106] MICHAEL BAUER, “Legion: Expressing locality and independence with logical regions (tech report),” 2012.

[107] “Micron DDR2 part catalogs - TwinDie 4 GB,” 2010. `http://www.micron.com/products/dram/ddr2/`.

[108] MONTANER, H., SILLA, F., FRÖNING, H., and DUATO, J., “MEMSCALE: A scalable environment for databases,” in *Proceedings of the 2011 IEEE International Conference on High Performance Computing and Communications*, 2011.

[109] MPI FORUM, “MPI specification 1.0,” 1994.

[110] MURPHY, R. C. and OTHERS, “Introducing the Graph 500,” in *Cray User’s Group (CUG)*, 2010.

[111] “Myricom’s Myri-10g 10-Gigabit Ethernet solutions,” 2010. `http://www.myri.com/Myri-10G/10gbe_solutions.html`.

[112] NELSON, M., LIM, B.-H., and HUTCHINS, G., “Fast transparent migration for virtual machines,” in *Proceedings of the annual conference on USENIX Annual Technical Conference*, 2005.

[113] NIEPLOCHA, J. and CARPENTER, B., “ARMCI: a portable remote memory copy libray for distributed array libraries and compiler run-time systems,” in *Proceedings of the IPPS/SPDP’99 Workshops*, pp. 533–546, Springer-Verlag, 1999.

[114] NIEPLOCHA, J., HARRISON, R. J., and LITTLEFIELD, R. J., “Global Arrays: A portable ”shared-memory” programming model for distributed memory computers,” in *Supercomputing ’94*, (New York, NY, USA), pp. 340–349, ACM, 1994.

[115] NISHTALA, R. N., HARGROVE, P. H., BONACHEA, D. O., and YELICK, K. A., “Scaling communication-intensive applications on BlueGene/P using one-sided communication and overlap,” in *23rd International Parallel & Distributed Processing Symposium*, 2009. Rome, Italy.

[116] NOAKES M., WALLACH D., D. W., “The J-Machine multicomputer: An architectural evaluation,” 1993.

[117] “NS-3 webpage,” `http://www.nsnam.org`.

[118] “NumaConnect (white paper),” `http:www.//numascale.com/numa_WP_NumaConnect.html`.

[119] NÜSSLE, M., SCHERER, M., and BRUNING, U., “A resource optimized remote-memory-access architecture for low-latency communication,” in *ICPP 2009*.

[120] NÜSSLE, M., "High performance interconnect leveraging HyperTransport - EXTOLL," 2010. http://www.extoll.de/images/extoll_slides.pdf.

[121] NVIDIA, "Tesla K20X GPU accelerator (board specification)," 2012. http://www.nvidia.com/content/PDF/kepler/Tesla-K20X-BD-06397-001-v05.pdf.

[122] OB́RIEN, K., "Micron P320h 2.5 PCIe application accelerator review (website)," 2013. http://www.storagereview.com/micron_p320h_25_pcie_application_accelerator_review.

[123] ODEN, L. and FRÖNING, H., "GGAS: Global GPU address spaces for efficient communication in heterogeneous clusters," in *Proceedings of the 2013 IEEE Cluster conference*, 2013.

[124] OIKAWA, M., KAWAI, A., NOMURA, K., YASUOKA, K., YOSHIKAWA, K., and NARUMI, T., "DS-CUDA: A middleware to use many GPUs in the cloud environment," in *SC Companion (SCC)*, 2012.

[125] OUSTERHOUT, J., AGRAWAL, P., ERICKSON, D., KOZYRAKIS, C., LEVERICH, J., MAZIÈRES, D., MITRA, S., NARAYANAN, A., ONGARO, D., PARULKAR, G., ROSENBLUM, M., RUMBLE, S. M., STRATMANN, E., and STUTSMAN, R., "The case for RAMCloud," *Commun. ACM*, 2011.

[126] OUSTERHOUT, J. and OTHERS, "The case for RAMClouds: Scalable high-performance storage entirely in DRAM," 2011.

[127] PANDA, D. K., "Acceleration for Big Data, Hadoop and Memcached (presentation)," *HPC Advisory Council Workshop*, Mar. 2012.

[128] PETRINI, F., FENG, W.-C., HOISIE, A., COLL, S., and FRACHTENBERG, E., "The Quadrics network: High-performance clustering technology," *Micro, IEEE*, vol. 22, pp. 46 –57, Feb. 2002.

[129] POTLURI, S., SUR, S., BUREDDY, D., and PANDA, D. K., "Design and implementation of key proposed MPI-3 one-sided communication semantics on InfiniBand," in *Proceedings of the 18th European MPI Users' Group conference on Recent advances in the message passing interface*, EuroMPI'11, 2011.

[130] "Quadrics Qs Ten G for HPC interconnect product family," 2008. http://www.quadrics.com/.

[131] QURESHI, M. K., "Adaptive spill-receive for robust high-performance caching in CMPs," in *In High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pp. 45–54, 2009.

[132] RANGANATHAN, P., LEECH, P., IRWIN, D., and CHASE, J., "Ensemble-level power management for dense blade servers," in *ISCA '06*, (Washington, DC, USA), pp. 66–77, IEEE Computer Society, 2006.

[133] RAVI, V. T., BECCHI, M., AGRAWAL, G., and CHAKRADHAR, S., "Supporting GPU sharing in cloud environments with a transparent runtime consolidation framework," in *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, (New York, NY, USA), p. 217228, ACM, 2011.

[134] RAVI, V. and OTHERS, "Scheduling concurrent applications on a cluster of CPU-GPU nodes," in *CCGrid 2012*, 2012.

[135] REANO, C., PEA, A., SILLA, F., DUATO, J., MAYO, R., and QUINTANA-ORTI, E., "Cu2rcu: Towards the complete rCUDA remote GPU virtualization and sharing solution," in *HiPC*, 2012.

[136] REINHARDT, S. K., LARUS, J. R., and WOOD, D. A., "Tempest and Typhoon: User-level shared memory," *SIGARCH Comput. Archit. News*, vol. 22, p. 325336, Apr. 1994.

[137] RNA NETWORKS, "RNA Networks memory virtualization solution using RNA MVX and Mellanox ConnectX-2 EN with RoCE accelerates business analytics by 10 times (white paper)," 2010.

[138] "Virtualization for aggregation and the vSMP architecture," `http://www.scalemp.com/uploads/whitepaper/vSMP_Foundation_Technical.pdf`.

[139] SCHLANSKER, M., TOURRILHES, J., TURNER, Y., and SANTOS, J., "Killer fabrics for scalable datacenters," in *Communications (ICC), 2010 IEEE International Conference on*, pp. 1 –6, May 2010.

[140] SCHROEDER, T. C., "Peer-to-Peer and Unified Virtual Addressing," *Cuda Webinar*, 2011.

[141] SHAINER, G., AYOUB, A., LUI, P., LIU, T., KAGAN, M., TROTT, C. R., SCANTLEN, G., and CROZIER, P. S., "The development of Mellanox/NVIDIA GPUDirect over InfiniBand-a new model for GPU to GPU communications," *Journal of Computer Science - Research and Development*, June 2011.

[142] SHAN, H., BLAGOJEVIĆ, F., MIN, S.-J., HARGROVE, P., JIN, H., FUERLINGER, K., KONIGES, A., and WRIGHT, N. J., "A programming model performance study using the NAS parallel benchmarks," *Sci. Program.*, vol. 18, pp. 153–167, 2010.

[143] SHAN, H. and OTHERS, "A preliminary evaluation of the hardware acceleration of the Cray Gemini interconnect for PGAS languages and comparison with MPI," in *PMBS*, 2011.

[144] SHAN, H. and SINGH, J. P., "A comparison of MPI, SHMEM and cache-coherent shared address space programming models on the SGI Origin2000," in *Proceedings of the 13th international conference on Supercomputing*, ICS '99, (New York, NY, USA), p. 329338, ACM, 1999.

[145] SHARMA, D. D., "Intel 5520 chipset: An I/O hub chipset for server, workstation, and high end desktop," in *Hot Chips*, vol. 21, 2009.

[146] SLOGSNAT, D., GIESE, A., NÜSSLE, M., and BRÜNING, U., "An open-source hypertransport core," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 1, no. 3, pp. 1–21, 2008.

[147] SOLARFLARE, "10G Ethernet: Now ready for low-latency HPC applications (white paper)," 2011. `http://www.solarflare.com/content/userfiles/documents/solarflare_10gbe_hpc_whitepaper.pdf`.

[148] "Storagereview.com drive performance resource center," 2008. `http://www.storagereview.com/`.

[149] STUART, J. and OWENS, J., "Multi-GPU MapReduce on GPU clusters," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pp. 1068–1079, 2011.

[150] SUZUKI, J., BABA, T., HIDAKA, Y., HIGUCHI, J., KAMI, N., UCHIDA, S., TAKAHASHI, M., SUGAWARA, T., and YOSHIKAWA, T., "Adaptive memory system over Ethernet," in *Proceedings of the 2nd USENIX conference on Hot topics in storage and file systems*, HotStorage'10, (Berkeley, CA, USA), 2010.

[151] TATE, J., "An introduction to Fibre Channel over Ethernet, and Fibre Channel over Convergence Enhanced Ethernet," 2009. `http://www.redbooks.ibm.com/redpapers/pdfs/redp4493.pdf`.

[152] "TCP-H benchmark specification," `http://www.tpc.org/tpch/`.

[153] TECHNOLOGIES, P., "VMWare vSphere vMotion: 5.4 times faster than Hyper-V live migration (test report)," 2011.

[154] TOLENTINO, M. E., TURNER, J., and CAMERON, K. W., "Memory MISER: Improving main memory energy efficiency in servers," *IEEE Trans. Comput.*, vol. 58, pp. 336–350, Mar. 2009.

[155] "UPC language specifications, v1.2," tech. rep., 2005.

[156] VANDEVAART, R., "NVIDIA and OpenMPI (presentation)," *Open MPI State of the Union Community Meeting SC11*, 2011. `www.open-mpi.org/papers/sc-2011/Open-MPI-SC11-BOF-1up.pdf`.

[157] VAUGHAN, C. and OTHERS, "Investigating the impact of the Cielo Cray XE6 architecture on scientific application codes," in *IPDPSW*, May 2011.

[158] VENKATARAMANI, V., AMSDEN, Z., BRONSON, N., CABRERA III, G., CHAKKA, P., DIMOV, P., DING, H., FERRIS, J., GIARDULLO, A., HOON, J., KULKARNI, S., LAWRENCE, N., MARCHUKOV, M., PETROV, D., and PUZAR, L., "TAO: How facebook serves the social graph," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pp. 791–792, 2012.

[159] VMWare, "Understanding memory resource management in VMware ESX4.1," 2006. `http://www.vmware.com/pdf/esx3_memory.pdf`.

[160] Vogels, W., "Eventually consistent," *Commun. ACM*, vol. 52, Jan. 2009.

[161] Wang, D., Ganesh, B., Tuaycharoen, N., Baynes, K., Jaleel, A., and Jacob, B., "DRAMsim: a memory system simulator," *SIGARCH Comput. Archit. News*, vol. 33, p. 100107, Nov. 2005.

[162] Wang, H., Potluri, S., Luo, M., Singh, A. K., Sur, S., and Panda, D. K., "MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters," *Journal of Computer Science - Research and Development*, vol. 26, p. 257266, June 2011.

[163] Wang, Y., Que, X., Yu, W., Goldenberg, D., and Sehgal, D., "Hadoop acceleration through network levitated merge," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, 2011.

[164] Woven Systems, "10 GE fabric delivers consistent high performance for computing clusters at Sandia National Labs," 2007. `http://www.chelsio.com/assetlibrary/pdf/Sandia_Benchmark_Tech_Note.pdf`.

[165] Wu, H., Diamos, G., Cadambi, S., and Yalamanchili, S., "Kernel Weaver: Automatically fusing database primitives for efficient GPU computation," *MICRO*, 2012.

[166] Wu, H., Diamos, G., Wang, J., Cadambi, S., Yalamanchili, S., and Chakradhar, S., "Optimizing data warehousing applications for GPUs using kernel fusion/fission," in *Multicore and GPU Programming Models, Languages and Compilers Workshop, IPDPS-PLC*, May 2012.

[167] Yalamanchili, S., Duato, J., Young, J., and Silla, F., "A dynamic, partitioned global address space model for high performance clusters (white paper)," tech. rep., 2008. `http://www.cercs.gatech.edu/tech-reports/tr2008/git-cercs-08-01.pdf`.

[168] Yelick, K., Semenzato, L., Pike, G., Miyamoto, C., Liblit, B., Krishnamurthy, A., Hilfinger, P., Graham, S., Gay, D., Colella, P., and Aiken, A., "Titanium: A high-performance Java dialect," in *In ACM*, p. 1011, 1998.

[169] Yoichi Koyanagi, Tadafusa Niinomi, Y. U., "10 Gigabit Ethernet switch blade for large-scale blade servers," *Fujitsu Scientific and Technical Journal*, vol. 46, no. 1, pp. 56–62, 2010.

[170] Young, J. and Holden, B., "HyperTransport over Ethernet specification, 1.0," 2010. `http://www.hypertransport.org`.

[171] YOUNG, J., "Dynamic partitioned global address spaces for high-efficiency computing (MS thesis)," 2008. `http://etd.gatech.edu`.

[172] YOUNG, J., SHON, S. H., YALAMANCHILI, S., MERRITT, A., SCHWAN, K., and FRÖNING, H., "Oncilla: A GAS runtime for efficient resource allocation and data movement in accelerated clusters," in *IEEE Cluster*, 2013.

[173] YOUNG, J. and YALAMANCHILI, S., "Dynamic Partitioned Global Address Spaces for power efficient DRAM virtualization," *International Conference on Green Computing*, 2010.

[174] YOUNG, J. and YALAMANCHILI, S., "Commodity converged fabrics for global address spaces in accelerator clouds," in *14th IEEE International Conference on High Performance Computing and Communications*, (Liverpool, UK), June 2012.

[175] YOUNG, J., YALAMANCHILI, S., HOLDEN, B., CAVALLI, M., and MIRANDA, P., "HyperTransport over Ethernet - a scalable, commodity standard for resource sharing in the data center," *Proceedings of the Second International Workshop on HyperTransport Research and Applications (WHTRA)*, 2011.

[176] YOUNG, J., YALAMANCHILI, S., SILLA, F., and DUATO, J., "A HyperTransport-enabled global memory model for improved memory efficiency," *WHTRA*, 2009.

[177] ZIAKAS, D., BAUM, A., MADDOX, R. A., and SAFRANEK, R. J., "Intel QuickPath interconnect architectural features supporting scalable system architectures," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*, pp. 1–6, IEEE, 2010.