Second International Workshop on
HyperTransport™ Research and Applications

Proceedings

# of the

2nd International Workshop on HyperTransport Research and Applications

WHTRA 2011

Editors
Holger Fröning
Mondrian Nüssle
Pedro Javier García García

February 9th, 2011, Mannheim, Germany
University of Heidelberg, Computer Architecture Group

Proceedings of the **Second International Workshop on HyperTransport Research and Applications (WHTRA2011)**, held Feb. 9th, 2011, Mannheim, Germany

EDITORS

Holger Fröning — Universität Heidelberg, Germany
Mondrian Nüssle — Universität Heidelberg, Germany
Pedro Javier García García — Universidad de Castilla-La Mancha, Spain

KEYNOTE SPEAKER

Prof. Olav Lysne — Director of Basic Research at Simula Group, Norway
Dr. Rainer Buchty — Eberhard-Karls-Universität Tübingen, Germany

PROGRAM COMMITTEE

INFORMATION ON PUBLICATION

To ensure a high level of academic content, a peer review process has been used. Each submission has been reviewed by a minimum of two separate reviewers on the Program Committee list.

The proceedings are available electronically at the website of the HyperTransport Center of Excellence as well as on HeiDOK, the Open Access document server of the University of Heidelberg (see links below). This publication platform offers free access to full-text documents and adheres to the principles of OpenAccess as well as the goals of the Budapest Open Access Initiative (BOAI). The papers are accessible through a special sub-portal and are fully citable.

The Open Access Document Server of the University library of Heidelberg also offers the possibility to order hardcopies of the proceedings.

Open Access Document Server:
http://archiv.ub.uni-heidelberg.de/volltextserver/portal/whtra11

Workshop Website:
http://ra.ziti.uni-heidelberg.de/coeht/index.php?page=events&id=20110209

HyperTransport Center of Excellence:
http://htce.uni-hd.de

WELCOME MESSAGE FROM THE EDITORS

As organizers of the Second International Workshop on HyperTransport Research and Applications (WHTRA), we would like to especially thank Prof. Olav Lysne and Dr. Rainer Buchty for accepting to deliver the two keynotes of the workshop.

We also thank all the members of the Program Committee for the time and effort devoted to this second edition of WHTRA. All submissions were reviewed by at least two members of the committee to ensure an ongoing high quality of the workshop contributions.

For this year's workshop we have, again, to thank the University of Heidelberg and the HyperTransport Center of Excellence for hosting this event, and the University Library of Heidelberg for publishing the proceedings.

Finally we would like to thank all the attendees and especially the authors for their interest and effort.

We hope WHTRA2011 will again be a stimulating and interesting meeting for us all!

Holger Fröning[*], Mondrian Nüssle[*] and Pedro Javier García García[†]

[*] Universität Heidelberg, Germany
[†] Universidad de Castilla-La Mancha, Spain

CONTENTS

Analysis of Inter-Chip Communication Patterns on Multi-Core Distributed Shared-Memory Computers

Manfred Mücke, Wilfried N. Gansterer
University of Vienna
Research Lab Computational Technologies and Applications

Abstract

Multi-core multi-socket distributed shared-memory computers (DSM computers, for short) have become an important node architecture in scientific computing as they provide substantial computational capacity with relatively low space and power requirements. Compared to conventional computer networks, inter-chip networks used in DSM computers feature higher bandwidth, lower latency and tighter integration with the CPU.

The inter-chip network is a shared resource among the user application and many other services, which can lead to considerable variation of execution times of identical communication tasks.

In this work, we explore traffic patterns resulting from MPI collective communication primitives and investigate the question whether inter-chip link load is a reliable indicator and predictor for the execution time of collective communication primitives on a DSM computer. Our experiments on a Sun Fire X4600 M2 DSM computer with 32 cores (eight quad-core CPUs) indicate that specific single link loads are positively correlated with the execution time of MPI_ALLREDUCE. Observing patterns over multiple links allows refinement of the single-link observation.

1. Motivation

Multi-core multi-socket distributed shared-memory (DSM) computers are a viable option to consolidate cluster infrastructure and to improve communication performance by reducing inter-node communication. One can think of a DSM computer as a small cluster with very high bandwith and low latency point-to-point interconnect.

In a cluster environment (many interconnected independent nodes), the overall performance is usually limited by the inter-node communication which is typically slow compared to local computation. Yet, recent work has shown that an unexpectedly high percentage of communication time is spent *within* multi-core nodes [3]. As a result, the node-internal communication performance – although faster than inter-node communication – is becoming more important for distributed applications' performance in a conventional cluster setting.

With current DSM computers integrating up to 48 cores in a single chassis, there is an increasing set of distributed applications which can run efficiently on a single DSM computer, thereby removing the need for a conventional cluster environment. To improve such an application's performance usually requires optimising the intra-node communication performance.

The situation for distributed applications executed on a single DSM computer changes considerably compared to a cluster environment as dedicated communication times among CPUs and memory access times become potentially identical. Additionally – and also in contrast to clusters – both computation (via memory access and/or cache coherency) and communication access the inter-chip communication network, which makes it a shared resource. Consequently, execution times of communication and computation can no longer be considered independent of each other but potentially heavily influence each other.

MPI collective communication functions [5] are powerful communication primitives whose optimisation is key to maximising performance of many parallel scientific computing applications. Collective communication can be seen as a parametriseable collection of point-to-point communications with only a few defined synchronisation points and the specific schedule being left to the implementation. We believe that a static schedule (or a set of several static schedules) is inadequate to efficiently exploit the available bandwidth in a contemporary multi-core DSM computer. Dynamic schedules might guarantee a more consistent performance over a wide range of network traffic scenarios. Dynamic schedules require, however, a cheap, yet reliable performance predictor, which is the motivation of our work.

MPI blocking communication provides function calls which return only when communication has finished (i.e., communication and computation is mutually exclusive for a single MPI process). There is an ongoing discussion on integrating non-blocking collective communication primitives into future versions of MPI. Non-blocking communication allows for overlapping communication and computation. However, when communication and computation overlap on DSM computers, usage patterns of shared resources become highly dynamic. In the worst case, this could lead to lower performance compared to blocking communication. Non-blocking collective communication implementations can, however, devise an efficient dynamic communication strategy, subject to available performance indicators. Therefore delivering the performance promises of non-blocking collective communication on DSM computers requires reliable communication performance predictors.

While DSM computers have existed for a long time, only recent developments have made them an almost ubiquitous computing platform. First, AMD integrated high-bandwidth low-latency inter-chip network interfaces (HyperTransport) into its mainstream server CPU family (Opteron), thereby removing the need for dedicated inter-chip communication circuits and simplifying the design of multi-socket computers considerably. Second, the integration of memory interfaces into CPUs enabled low-latency access to memory via inter-chip network thereby allowing very-low-latency non-uniform memory access (NUMA) computers. Third, multi-core CPUs have mitigated the scaling limitations of integrated inter-chip networks (for example AMD Opterons only support up to eight-socket configurations) by providing more cores per socket. Currently, systems with 48 cores (eight quad-core CPUs) are available. Finally, the evolution of communication technology has led to inter-chip point-to-point interface specifications matching typical internal bandwidths of CPUs (HyperTransport 3.1: 16 bit@3.2 GHz, max. 16 bit bi-directional bandwidth of 25.6 GB/s), leading to a communication performance which is *at par* with computation performance.

2. Problem Formulation

Inter-chip networks of contemporary DSM computers are typically used by multiple system services, they are a shared resource. Most prominently, remote memory access, the cache coherency protocol and system I/O usually use the same inter-chip network as dedicated communication between CPUs. Consequently, identical user-triggered communication can meet very different resource usage scenarios leading to variations in execution times.

Dynamic communication schedules can mitigate this effect. To choose the most efficient schedule for a communication operation at any given time, a performance model is required, taking the load on all relevant shared resources into account. The fastest schedule is then derived from the model by extrapolating current usage on all relevant shared resources.

Our aim in this paper is to identify the relevant observables necessary to implement dynamic schedules for MPI collective communication functions on DSM computers at the lowest possible cost (i.e., observation should be feasible on standard hardware and should cause only little overhead).

We hypothesise that on DSM computers the respective bandwidth available on each link of the inter-chip network is the single most relevant parameter influencing the execution time of a collective communication function. If this hypothesis can be verified, observing the inter-chip network bandwidth would provide sufficient information for optimizing dynamic communication schedules. Contemporary CPUs feature hardware performance counters which provide detailed information on the link traffic with high accuracy and at low cost, therefore on existing CPU architectures, monitoring inter-chip network bandwidth is possible for user applications at basically no extra cost.

3. Related Work

Scogland et al. [12] describe in a more general setting than our MPI-centric one that although multi-core hardware is mostly symmetric (i.e. cores have equivalent raw performance and bandwidth available), resulting workload per core is highly asymmetric due to the interaction of communication and computation.

Kayi et al. [7] report performance figures for large-scale simulations on a hybrid cluster consisting of nodes with 2 sockets (4 cores) and 8 sockets (16 cores), respectively. They found that application performance was *poorer* on the more powerful nodes. Only when applications employed some kind of node-internal load balancing, improvements could be observed. Core binding was found to improve the situation, too.

Porterfield et al. [11] conducted a detailed performance study of a variety of AMD quad-core multi-socket systems over a set of memory benchmarks. They found that performance models characterising memory by maximum bandwidth and average latency parameters are not sufficient to model the deep memory hierarchies found in modern ccNUMA architectures. Specifically, they found performance variability for memory-bound benchmarks to be a serious obstacle to load balancing and performance tuning [10]. Binding threads and data to specific sockets and carefully selecting the sockets they are bound to both reduced variability and improved overall performance of the benchmarks.

Underwood [17] discussed the mismatch between frequently used MPI microbenchmarks and the setting which MPI functions encounter in real-world applications, reporting an execution time difference up to a factor of four in extreme cases.

Mamidala et al. [9] investigated performance of MPI collectives on contemporary multi-core architectures. They concentrate on exploiting features of modern multi-core architectures (e.g. shared caches) for improving *average* performance of selected collectives. Their work does not consider execution time deviations of identical function calls. Mamidala et al. show more efficient ways to implement collectives while our work demonstrates the behaviour of a given implementation in the dynamic setting inherent to multi-core distributed shared-memory computers. Our work is complementary, as Mamidala et al. try to understand and reduce average execution time while we try to understand and improve execution time variability.

Hoefler and Lumsdaine investigated the performance of non-blocking MPI collectives on Infiniband and suggested measures for improving overlap of communication and computation [6]. They showed that performance can be improved considerably. They do, however, not consider inter-chip networks but only inter-node networks (Infiniband).

AMD provides a technical report ”Performance Guidelines for AMD Athlon and AMD Opteron cc-NUMA Multiprocessor Systems” [2] which summarises detailed measurements performed on a system with four dual-core AMD Opteron CPUs. A synthetic benchmark is used which comprises two tasks reading/writing data from/to independent memory locations. Execution times for all possible combinations of task and data placement are measured. Additional tasks read data from local memory to simulate background activity. The benchmark chosen explores how (remote) memory access translates into HyperTransport activity under varying task and data placement scenarios. In contrast to the data presented by AMD, we consider collective communication instead of point-to-point communication. Furthermore, while AMD creates a synthetic background activity, our goal is to infer unknown background activity patterns and its impact on execution time on a known collective communication.

In summary, existing work concentrates on cluster settings when evaluating overall application performance. In contrast, we argue that the performance of existing DSM computers is sufficient to run distributed applications entirely on a single DSM computer. Detailed performance analysis of DSM computers exist in literature but mostly focuses on the relative placement of tasks and data. Where communication functions are investigated, the aim is at reducing the *average* performance. To the best of our knowledge, our work is the first one investigating the execution time variance of collective communication due to background activity on a DSM computer’s HyperTransport inter-chip network.

4. Experimental Setup

Our aim is to better understand execution times of MPI collective communication primitives on DSM computer inter-chip networks. To make insights attractive to as many distributed applications as possible, we have chosen a DSM computer with many cores and a complex inter-chip network.

The Sun Fire X4600 M2 server [15] fulfils these requirements by supporting up to eight quad-core CPUs, which results in an inter-chip network of the maximum size currently supported by the AMD Opteron architecture (8 sockets) and a maximum worst-case traffic pressure per link (up to four cores sharing a single link). The X4600’s inter-chip network fully relies on functionality (cache coherency protocol, . . .) and interfaces (HyperTransport) integrated in the AMD Opteron architecture. However, Opteron-internal tables specifying routing and hardware buffer sizes can be set at system start-up potentially leading to physically identical DSM systems executing identical applications yet exhibiting varying appliation performance.

The general findings of our experiments will therefore apply to a wide range of servers with similar architecture while the exact results of our measurement are obviously specific to the system used.

4.1 Hardware

Our prototypical DSM computer is a Sun Fire X4600 M2 server [15] by Sun Microsystems, which we will refer to as ”X4600” in the following. The X4600 is designed to accept up to eight CPU/memory modules and can therefore exploit the maximum number of CPUs currently supported by AMD’s Opteron 8000 CPU family [1]. The motherboard itself provides no memory or computing facilities but only module interconnect, power and I/O.

Each CPU/memory module carries local memory. The total of all local memory present on all modules is mapped by the operating system into a uniform address space (8×4 GB = 32 GB for our system).

Every CPU/memory module features a single CPU socket, which can be fitted with a single-core, dual-

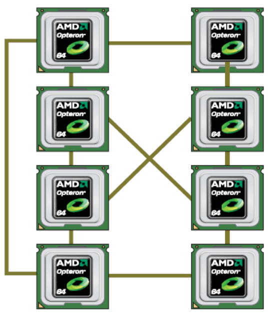
Figure 1. HyperTransport socket interconnect topology of a Sun Fire X4600 M2 server equipped with eight CPU modules.

core or quad-core AMD Opteron. The CPU used in our X4600 configuration is an AMD quad-core Opteron 8356. Each 8356 core features 2 MB private L2 cache while a 2 MB L3 cache is shared among all four cores. The cache-coherency protocol guarantees that all existing cache copies of data in memory are refreshed when data is modified anywhere in the system.

The AMD Opteron architecture integrates all memory controller functionality and three HyperTransport interfaces on-die [4]. The latter makes it possible to build servers with a very dense inter-socket communication network [8]. The AMD Opteron 8356 HyperTransport interfaces comply to HyperTransport 1.0, specifying 16 bit wide links with a clock frequency of 1 GHz. The links work in double-data-rate mode which results in a total bandwidth of 4 GB/s per direction.

Our system is equipped with eight CPU/memory modules. Sockets 0 and 7 dedicate one of their three links to connect the inter-socket network to system I/O. Figure 1 shows the X4600's inter-socket network topology (”twisted ladder”). Our X4600's inter-chip network therefore consists of 22 unidirectional HyperTransport links, while the two remaining links connect the network to system I/O facilities (hard disk drive, network, ..).

4.2 Operating System, Middleware

The used operating system is OpenSolaris 10 5/09. OpenSolaris features memory placement optimisation (MPO) which attempts to allocate memory as near to a process as possible [13, 14]. While the Solaris scheduler is able to move threads between all available cores (and therefore also between sockets), data remains by default on the CPU/memory module where it was first allocated.

The MPI distribution used is OpenMPI 1.3. OpenMPI provides support for core binding, i.e. manually assigning an MPI process to a core. We always bind all processes to distinct cores with the root process being assigned to core 4 (i.e. the first core on the second socket, thereby avoiding socket 0 through which I/O access is routed).

The AMD Opteron architecture provides hardware event counters to measure link load [1]. We have used the Solaris `lcpc(3CPC)` library for setting up and reading out hardware event counter values.

5 Experiments

We have chosen the `MPI_Allreduce` function as a prototypical MPI collective communication function. In this operation, all processes send arrays of identical size and type to the root process. There, entries of the same index are reduced using a specified arithmetic function.

In terms of communication performance, it would suffice to consider `MPI_Allgather`, as `MPI_Allreduce` can be assembled from an all-gather operation followed by some local computation. `MPI_Allreduce`, however, natively integrates this computation following communication and therefore provides better workload characteristics in terms of possible interference between communication and computation.

Each process is bound to a specific core. No specific measures are taken to guarantee placement of data in local memory.

There is no explicit waiting between consecutive calls of `MPI_Allreduce`. While this might be unrealistic in most application settings, it maximises stress on the inter-chip network and therefore allows observation of effects which might only be visible sporadically otherwise.

Using hardware counters accessible via libcpc, we measure the link load (i.e. sent/received data words on the observed link in the given time interval) in both directions on all links during execution of a given communication function (48 measurements). Specifically, we monitor the Opteron's ”Link Event” registers (0F6h, 0F7h, 0F8h, 1F9h, ”HyperTransport Link x Transmit Bandwidth”, see [1] for full details).

5.1 `MPI_Allreduce` with 8x4 processes

We measure the execution time of an `MPI_Allreduce` function call (using `hrtimer()`) collecting and processing messages of 16kB each from 32 MPI processes. Additionally, we monitor the traffic on all HyperTransport links during execution of `MPI_Allreduce`. The measurements are repeated for consecutive 2000 calls of `MPI_Allreduce`.

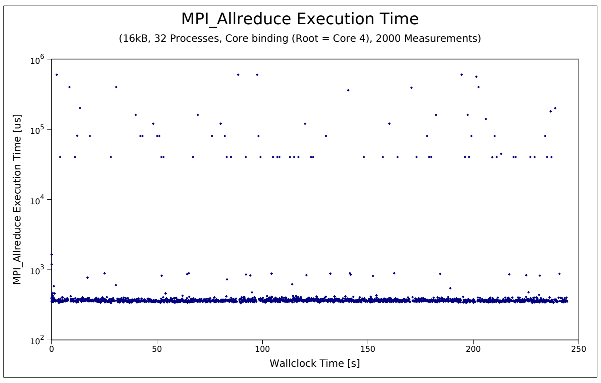

Figure 2. Execution times of 2000 consecutive `MPI_Allreduce` calls.

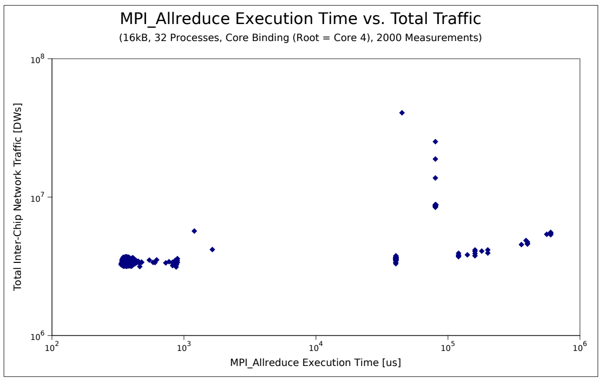

Figure 4. Execution time of `MPI_Allreduce` versus total inter-chip network traffic.

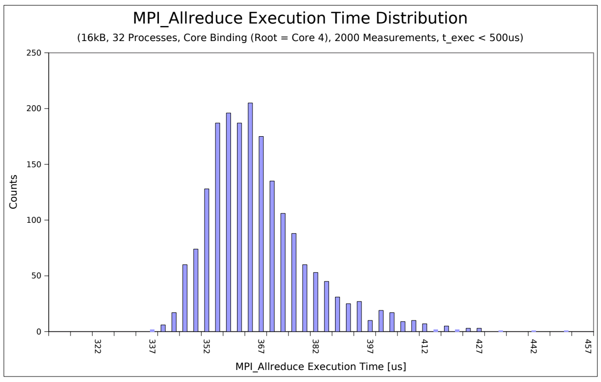

Figure 3. Execution time distribution of `MPI_Allreduce` calls executed in less than $500\mu s$.

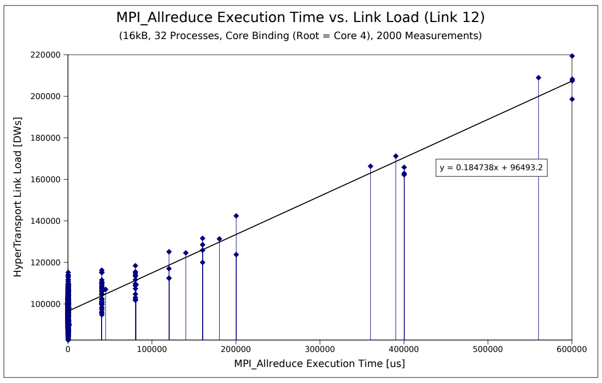

Figure 5. Execution time of `MPI_Allreduce` versus traffic on HyperTransport link 12.

Figure 2 shows the execution time of each `MPI_Allreduce` call over wallclock time (i.e. the x-axis corresponds to time progress during experiment). Most calls take less than 1000 μs (the median of all measurements is 363 μs). However, some execution times deviate considerably with maximum execution times up to 600 ms!

More than 95% of all measurements result in an execution time smaller than 500 μs. Figure 3 shows the distribution of these measurements.

While the majority of calls is very fast, the remaining calls consume an disproportional amount of time. The accumulated execution time of the 100 slowest calls (5%) consumes 93% of the overall sum of all execution times.

We hypothesise that the longer execution times can be explained by activity on the inter-chip network resulting in reduced available bandwidth on some HyperTransport links. In the following, we focus on relating `MPI_Allreduce` execution time with HyperTransport link load.

A first naive approach could be to relate execution time to the overall traffic on the inter-chip network during execution of each call as shown in Figure 4. No obvious correlation can be identified.

We use GGobi [16] to interactively explore the 23-dimensional space spanned by our measurements (22x HyperTransport outgoing link traffic, 1x MPI_Allreduce execution time) and find that some link traffic data is positively correlated with the `MPI_Allreduce` execution time. Figure 5 shows the traffic on link 12 over the execution time of `MPI_Allreduce`. The positive correlation is obvious. Similar correlation exists for data from several links.

The correlation observed is sufficient to distinguish short, medium and long execution times by single link load observations.

The observed link load stems from at least one collective communication (initiated by our foreground task) and multiple additional (point-to-point and maybe col-

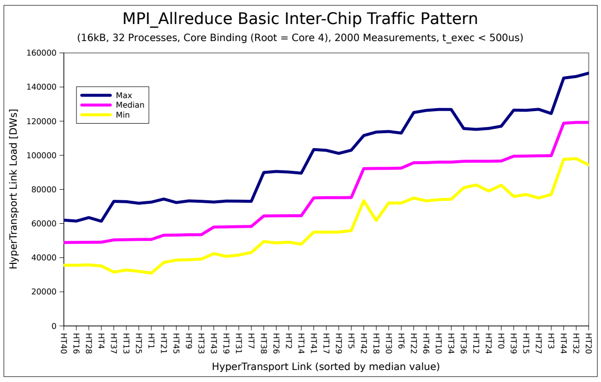

Figure 6. Min/max/median HyperTransport link load for calls of `MPI_Allreduce` with an execution time smaller than 500μs.

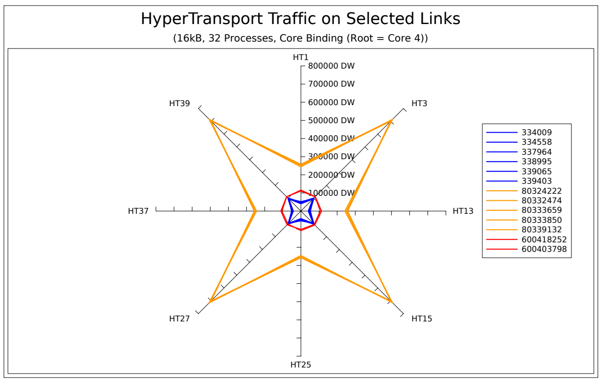

Figure 7. HyperTransport link load for selected calls of `MPI_Allreduce` on Links 1, 3, 13, 15, 25, 27, 37 and 39.

lective) communication triggered by background tasks (scheduler, cache coherency protocol, I/O activity, ...). According to Figure 1, messages exchanged between cores on different sockets can lead to routing of the message through up to three HyperTransport links. We therefore try to identify traffic patterns rather than simple link load to explain execution times.

Figure 6 shows the observed minimum, maximum and median HyperTransport link load for all links when inspecting data for all calls of `MPI_Allreduce` which result in an execution time smaller than 500μs. The links are ordered by their median load.

To identify distinct traffic patterns being related with specific execution time levels, we use GGobi's automatic brushing tool which allows colouring of data in all plots according to an additional given parameter (execution time in our case). Inspection reveals that increased activity on Links 1, 3, 13 and 15 corresponds to an execution time of about 80ms (the third cluster from left in Figure 5) . Figure 7 shows the activity on each of the links for some measurements resulting in high (red), low (blue) and moderate (about 80ms, orange) execution times respectively. The large star pattern is formed by measurements resulting in execution times of around 80ms exclusively.

5.2 Discussion of measurements

Our measurements of 2000 consecutive `MPI_Allreduce` calls reveal that while most calls (95%) finish within a very short time (less than 500 μs, median is 362 μs), the remaining 5% consume 93% of the experiment's run time.

It is possible to identify HyperTransport links whose load is positively correlated with `MPI_Allreduce` execution time. While not a very accurate indicator, single link load data of selected links seems sufficient to separate typical execution times from pathological cases.

The distribution of execution times is not continuous but shows strong clustering. Mining the measurements for correlations between clusters of similar execution time has revealed that increased traffic on a set of links directly corresponds to execution times within the cluster. This insight can be used to improve accuracy of the above indicator.

6 Conclusions and Outlook

We have shown that `MPI_Allreduce` execution time is correlated with HyperTransport link load. This is an important observation as a multitude of root causes might originally be involved, leading ultimately to the varying execution times observed. Relying on the correlation identified, we can focus on a much smaller set of observables. Current CPU architectures provide on-chip hardware performance counters for monitoring of inter-chip network traffic which allows link loads to be observed easily from a user application at run-time.

Which links need to be observed is a function of the full set of communication triggered by both foreground and background tasks. Our method of identifying relevant links relies on visual inspection of data which implies a big overhead in case substantial changes to the set of tasks are made. It would therefore be most desirable to partially automate the process of identifying relevant links.

Different implementations of `MPI_Allreduce` lead to different communication patterns. Therefore our findings only apply to the specific implementation of`MPI_Allreduce` in the used OpenMPI version.

We have considered a single foreground traffic pattern (`MPI_Allreduce`). Further work will investigate other MPI collective communication functions and the effects they will encounter when being executed on an inter-chip network with varying load.

We have not actively triggered any background communication activity. The varying execution times observed show that symmetric multi-core architectures in use today sporadically exhibit extremely asymmetric performance behaviour. This is due to the asymmetry of the communication infrastructure (see Figure 1) as well as conflicting resource usage by competing user and system tasks and communication stack deficiencies (see [12]).

There are two ways how our findings could be applied: First, it could be used to construct a predictor for execution times of selected communication functions under dynamic load situations. Second, it could also be used as a bottom-up analysis tool for system activity affecting the execution time of communication.

We plan to extend our work by identifying relevant background tasks and reducing their activity if possible. We will as well equip our benchmark with the cheap predictor proposed in this work. A simple measure to show the viability of our predictor in a conventional MPI setting would be to postpone execution of communication calls if the predictor suggests very long execution times.

During preparation of this work, the maximum number of cores available on an AMD Opteron CPU has increased from four to twelve. As a consequence, larger distributed applications can be run on a single server, increasing the complexity of traffic patterns on the inter-chip network while relying heavily on its performance.

7 Acknowledgements

We thank the X4600's system administrator Martin Paul for his support and Richard Smith of Sun Microsystems for his helpful comments and pointers on Opteron architecture and hardware event counters under Solaris.

This work was partially supported by the CPAMMS project of the University of Vienna (FS397001) and by the NFN S106 (SISE) of the Austrian Science Fund FWF.

References

[1] AMD. *BIOS and Kernel Developer's Guide for AMD Athlon and AMD Opteron Processors*, February 2006.

[2] AMD. *Performance Guidelines for AMD Athlon and AMD Opteron ccNUMA Multiprocessor Systems*. Advanced Micro Devices, Inc., 3.00 edition, June 2006.

[3] L. Chai, Q. Gao, and D. K. Panda. Understanding the impact of multi-core architecture in cluster computing: A case study with intel dual-core system. In *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 471–478, 2007.

[4] P. Conway and B. Hughes. The AMD Opteron northbridge architecture. *Micro, IEEE*, 27(2):10–21, 2007.

[5] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir. *MPI: The Complete Reference*. The MIT Press, September 1998.

[6] T. Hoefler, P. Gottschling, and A. Lumsdaine. Leveraging non-blocking collective communication in high-performance applications. In *SPAA '08: Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, pages 113–115, New York, NY, USA, 2008. ACM.

[7] A. Kayi, E. Kornkven, T. E. Ghazawi, and G. Newby. Application performance tuning for clusters with cc-NUMA nodes. In *CSE '08: Proceedings of the 2008 11th IEEE International Conference on Computational Science and Engineering*, pages 245–252, Washington, DC, USA, 2008. IEEE Computer Society.

[8] C. N. Keltcher, K. J. Mcgrath, A. Ahmed, and P. Conway. The AMD Opteron processor for multiprocessor servers. *Micro, IEEE*, 23(2):66–76, 2003.

[9] A. R. Mamidala, R. Kumar, D. De, and D. K. Panda. MPI collectives on modern multicore clusters: Performance optimizations and communication characteristics. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pages 130–137, 2008.

[10] A. Mandal, A. Porterfield, R. J. Fowler, and M. Y. Lim. Performance consistency on multi-socket AMD Opteron systems. Technical Report TR-08-07, RENCI, North Carolina, 2008.

[11] A. Porterfield, R. Fowler, A. Mandal, and M. Y. Lim. Empirical evaluation of multi-core memory concurrency. Technical Report TR-09-01, RENCI, North Carolina, January 2009.

[12] T. Scogland, P. Balaji, W. Feng, and G. Narayanaswamy. Asymmetric interactions in symmetric multi-core systems: analysis, enhancements and evaluation. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[13] Sun Microsystems. Solaris memory placement optimization and Sun Fire servers. Technical report, Sun Microsystems, March 2003.

[14] Sun Microsystems. *Memory and Thread Placement Optimization Developer's Guide*, June 2007.

[15] Sun Microsystems. Sun Fire X4600 M2 server architecture. Technical report, Sun Microsystems, June 2008.

[16] D. F. Swayne, D. Temple Lang, A. Buja, and D. Cook. GGobi: evolving from XGobi into an extensible framework for interactive data visualization. *Computational Statistics & Data Analysis*, 43:423–444, 2003.

[17] K. Underwood. Challenges and issues in benchmarking MPI. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 339–346. Springer, 2006.

HyperTransport Over Ethernet - A Scalable, Commodity Standard for Resource Sharing in the Data Center

Jeffrey Young, Sudhakar Yalamanchili[*]
Georgia Institute of Technology
jyoung9@gatech.edu, sudha@ece.gatech.edu

Brian Holden, Mario Cavalli
HyperTransport Consortium
{brian.holden, mario.cavalli}@hypertransport.org

Paul Miranda
AMD
paul.miranda@amd.com

Abstract

Future data center configurations are driven by total cost of ownership (TCO) for specific performance capabilities. Low-latency interconnects are central to performance, while the use of commodity interconnects is central to cost. This paper reports on an effort to combine a very high-performance, commodity interconnect (HyperTransport) with a high-volume interconnect (Ethernet). Previous approaches to extending HyperTransport (HT) over a cluster used custom FPGA cards [5] and proprietary extensions to coherence schemes [22], but these solutions mainly have been adopted for use in research-oriented clusters. The new HyperShare strategy from the HyperTransport Consortium proposes several new ways to create low-cost, commodity clusters that can support scalable high performance computing in either clusters or in the data center.

HyperTransport over Ethernet (HToE) is the newest specification in the HyperShare strategy that aims to combine favorable market trends with a high-bandwidth and low-latency hardware solution for non-coherent sharing of resources in a cluster. This paper illustrates the motivation behind using 10, 40, or 100 Gigabit Ethernet as an encapsulation layer for HyperTransport, the requirements for the HToE specification, and engineering solutions for implementing key portions of the specification.

[*]This research was supported in part by NSF grant CCF-0874991, and Jeffrey Young was supported by a NSF Graduate Research Fellowship

1. Introduction

HyperTransport interconnect technology has been in use for several years as a low-latency interconnect for processors and peripherals [9] [7] and more recently as an off-chip interconnect using the HTX card [5]. However, HyperTransport adoption for scalable cluster solutions has typically been limited by the number of available coherent connections between AMD processors (8 sockets) and by the need for custom HyperTransport connectors between nodes.

The HyperTransport Consortium's new HyperShare market strategy has presented three new options for building scalable, low-cost cluster solutions using HyperTransport technology: 1) HyperTransport-native torus-based network fabric using PCI Express-enabled network interface cards implementing the HyperTransport High Node Count specification [14], 2) HyperTransport encapsulated into InfiniBand physical layer packets, and 3) HyperTransport encapsulated into Ethernet physical layer packets. These three approaches provide different levels of advantages and trade-offs across the spectrum of cost and performance. This paper describes the encapsulation of HyperTransport packets into Ethernet, thereby leveraging the cost and performance advantages of Ethernet to enable sharing of resources and (noncoherent) memory across future data centers. More specifically, this paper describes key aspects of the HyperTransport over Ethernet (HToE) specification that is part of the HyperShare strategy.

In the following sections, we describe 1) the motivation for using HToE in both the HPC and data center arenas, 2) challenges facing the encapsulation of HT packets over Ethernet, 3) an overview of the major components of this specification, and 4) use cases that demonstrate how this new specification can be utilized for resource sharing in high node count environments.

2. The Motivation for HToE: Trends in Interconnects

The past ten years in the high-performance computing world have seen dramatic decreases in off-chip latency along with increases in available off-chip bandwidth, due largely to the introduction of commodity networking technologies like InfiniBand and 10 Gigabit Ethernet (10GE) from companies such as Myrinet and Quadrics. Arguably, InfiniBand has made the most inroads in the high-performance computing space, with InfiniBand composing 42.6% of the fabrics for clusters on the current Top 500 Supercomputing list [18].

At the same time, Ethernet has evolved as a lower-cost and "software-friendly" alternative that enjoys higher volumes. 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.

2.1. Performance

The ratification of the 10 Gigabit Ethernet standard in 2002 [1] has led to its adoption in data centers and the high-performance community. 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 [28]. In addition, switch manufacturers have built 10 Gigabit Ethernet devices with latencies in the low hundreds of nanoseconds [11] [31]. Recent tests with iWARP-enabled 10GE adapters have shown latencies that are on the order of 8-10 microseconds, as compared to similar InfiniBand adapters with latencies of 4-6 microseconds [12]. More recent tests have confirmed that 10 Gigabit Ethernet latency for MPI with iWARP is in the range of 8 microseconds [20].

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. The new IEEE 802.3ba standard for 40 and 100 Gbps Ethernet also aims to make Ethernet more competitive with InfiniBand. Although full-scale adoption is likely to take several years, there are already some early products that support 100 Gigabit Ethernet [25].

The challenge with using these lower-latency fabrics is in making these 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 [4] [23]. The HToE specification described here 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 in performing remote memory accesses.

2.2. Cost and Market Share

While 10 Gigabit Ethernet has had a relatively slow adoption rate in the past few years, it should be noted that 1 Gigabit and 10 Gigabit Ethernet still have a 45.6% share of the Top 500 Supercomputing list [18], with a majority of these installations still using 1 Gigabit Ethernet. This indicates that cost plays an important role in the construction of computational clusters on this list (for example, for market analysis and geological data analysis in the mineral and natural resource industries). Additionally, networks composed of 1 and 10 Gigabit Ethernet also have a dominant position in high-performance web server farms. Part of this widespread market share is due to the low cost of Gigabit Ethernet and 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 market. A 40 Gbps, 36 port InfiniBand switch now costs around $6,500 and has a typical power dissipation of 226 Watts [8], while a 10 Gbps, 48 port Ethernet switch costs around $20,900 and has a power dissipation of 360 Watts.

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). 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 [15]. 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 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 [24] and the development of RDMA over Converged Ethernet (RoCE) [3], 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.

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, consider the use of highly scalable fat tree networks for data centers using 10 Gigabit Ethernet [27], 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 [21]. Other recent studies have demonstrated techniques for active congestion management to enable further scaling of topologies constructed around Ethernet [28]. We can expect to see continued efforts toward expanding the use of Ethernet in an effort to leverage legacy software, existing expertise in the networking-related workforce, and volume cost-related advantages.

2.4. The Case for HToE

As the previous sections have shown, Ethernet has significant benefits in the areas of cost, market share, and competitive performance. HyperTransport over Ethernet shares these benefits while adding the advantage of a transparent on-package to off-package encapsulation using 10, 40, and 100 Gbps Ethernet. The IEEE 802.3ba standard also includes support for short-reach (7 meter) copper cable physical layers for 40 and 100 Gigabit Ethernet, which should allow for more cost-effective implementations of 40 and 100 Gigabit Ethernet. As the penetration of these new flavors of Ethernet grows, the potential for HyperTransport over Ethernet also grows as a high-performance hardware communication and sharing mechanism. In fact, this capability for improved resource sharing is one of the best motivators for using HToE and is discussed in more detail in Section 5.

HyperTransport over Ethernet also addresses a different market space than that served by the HyperTransport High Node Count specification and HyperTransport over InfiniBand. Specifically, HToE is well suited for creating scalable, low cost clusters that rely on a converged Ethernet fabric to share resources in a non-coherent fashion. Ethernet's market share ensures that

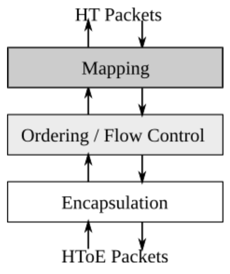

Figure 1. HyperTransport Over Ethernet Layers

the barrier to entry in using HyperTransport over Ethernet is low in most cases, and using converged Ethernet negates the need for a custom sharing fabric like NUMAlink [16] or additional cabling for an InfiniBand or other custom network.

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, we focus 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). If we are 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, we advocate the use of global address space models and specifically the Partitioned Global Address Space (PGAS) model [30]. In particular, we are concerned about the portability of the model and application/system software across future generations of processors with increasing physical address ranges.

To illustrate the differences between HyperTransport and HToE and to help illustrate how HToE supports global address models, we have divided the core functionality of HToE into three "layers": the "mapping" layer, the "ordering and flow control" layer, and the "encapsulation" layer, as shown in Figure 1.

3.1. On-package and Off-package Addressing

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. For more details on the specific mapping used by HToE, see Sections 4.2 and 4.3.

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 [7]. This algorithm ensures not only that packets arrive in a logical order but also that deadlock freedom is ensured. In a switched Ethernet environment with the possibility of packet loss, preservation of ordering becomes a much more difficult problem. Thus, our HToE solution must ensure that packets remain ordered correctly within their virtual channels. For more information on maintaining order, see Section 4.4.

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 requirement is that the sender must possess credits for all HyperTransport packets that it encapsulates. HyperTransport packets that are encapsulated in a single Ethernet packet must be of the same virtual circuit and headed for the same destination.

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 [29]. 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) [17]. These congestion-management algorithms are especially helpful in high-performance computing because of the intensely self-similar nature of HPC traffic.

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 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

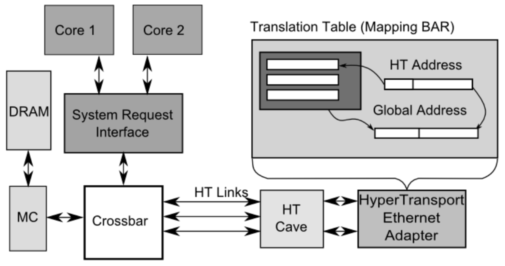

Figure 2. HyperTransport Ethernet Adapter with Opteron Memory Subsystem

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.

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. We did not want to rely on TCP's retry algorithm, but Ethernet does not define a Layer 2 error retry protocol. Therefore, we created a variant of HyperTransport 3.1's retry algorithm that would function across an Ethernet fabric in the presence of packet loss due to congestion or due to bit errors.

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. The location of this device in relation to a typical Opteron system is shown in Figure 2. Note that 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.

To assist with the implementation of each of the specification's requirements, functionality in the HTEA

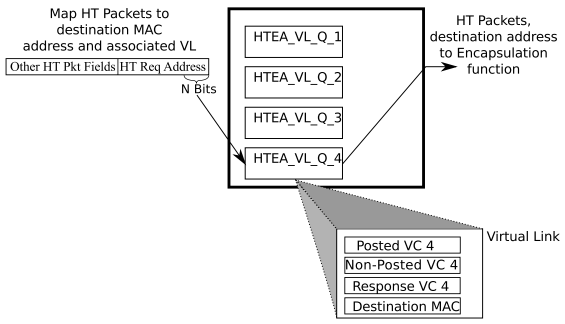

Figure 3. HyperTransport Ethernet Adapter Virtual Link

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. Here we describe some of the more interesting aspects of the "mapping" layer, the "ordering" layer, and the "encapsulation" layer. Full details are available in the HToE specification [32].

4.1. HToE's Relationship with DCB

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 with Enhanced Transmission Selection for usage with other types of network traffic.

4.2. 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 [30], 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 shown in Figure 2. This mapping allows for a processor on a local node to make a remote noncoherent "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 capabilities of the Data Center Bridging Capabilities Exchange Protocol (DCBX) [17] 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 in the specification. As shown in Figure 3, 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 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 4.4.

4.3. 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.

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 [30] to maximize the number of Non-Posted requests that can be sent to the local HT link. Figure 4 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. On completion of the servicing of a request, the corresponding responses 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.

4.4. 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 are 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 standard 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 5 shows how this backpressure causes buffers in the sending HTEA at Node 1 to become full, pausing transactions until more credits are

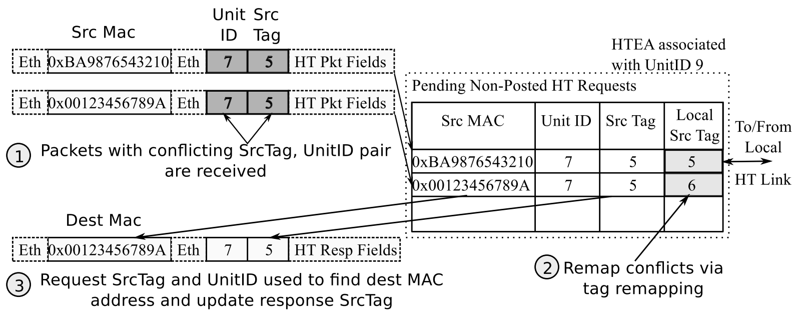
Src Mac
Unit ID
Src Tag
Eth
0xBA9876543210
Eth
7
5
HT Pkt Fields
Eth
0x00123456789A
Eth
7
5
HT Pkt Fields
1 Packets with conflicting SrcTag, UnitID pair are received
HTEA associated with UnitID 9
Pending Non-Posted HT Requests
Src MAC
Unit ID
Src Tag
Local Src Tag
0xBA9876543210
7
5
5
0x00123456789A
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 4. Tag Remapping in the HTEA

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. However, it mentions and leaves open many opportunities to optimize the allocation of credits and buffers to multiple senders.

4.5. 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 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 3.1 specifies a method for sending responses with error bits to notify of errors, HToE allows for remote transactions to be terminated and handles error notification. To do this the HTEA must keep track of sent HyperTransport packets that require a response (Non-Posted packets), and if it receives a notification that the response has been lost or the remote node has been reset, it can then reply with a normal HT 3.1 packet with the "poison" or error bit set. This additional state for remote requests allows for easier error detection and detection of request timeouts.

4.6. Security in HToE-enabled Data Centers

Since HyperTransport over Ethernet enables easy, transparent (OS interaction is not necessarily needed) hardware sharing of noncoherent memory between nodes, more care must be taken to make sure that malicious HyperTransport packets are not inserted into an Ethernet packet and sent to a remote node. While certain HPC-oriented clusters that are not used to handle web-related data may not have as high security requirements, networks exposed to the Internet may require additional security measures. Fortunately, HToE defines the use of IEEE 802.1ae MACsec to provide for encrypted 10 Gigabit Ethernet traffic between nodes.

5. Resource Sharing with HToE - Use Cases

The creation of a high-performance, scalable, commodity network using HyperTransport over Ethernet opens up the possibility of many application models that are based on low-latency noncoherent communication. Here we present two potential usages of this commodity standard to promote resource sharing within a data center or HPC environment. Both are predicated on the assumption that future clusters will be limited not nec-

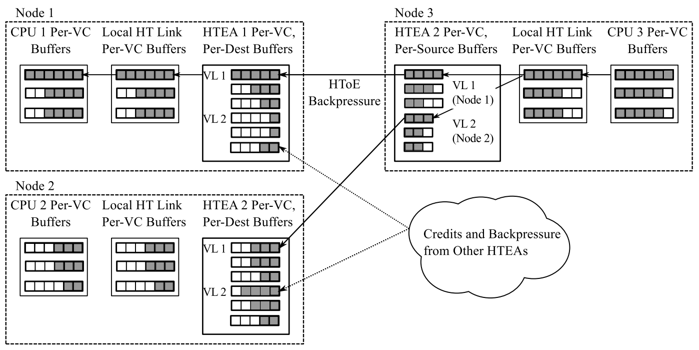
Node 1
CPU 1 Per-VC Buffers
Local HT Link Per-VC Buffers
HTEA 1 Per-VC, Per-Dest Buffers
VL 1
VL 2
HToE Backpressure
Node 3
HTEA 2 Per-VC, Per-Source Buffers
Local HT Link Per-VC Buffers
CPU 3 Per-VC Buffers
VL 1 (Node 1)
VL 2 (Node 2)
Node 2
CPU 2 Per-VC Buffers
Local HT Link Per-VC Buffers
HTEA 2 Per-VC,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

Figure 5. HToE Backpressure-based Flow Control

essarily by processing power but rather by factors like TCO and power usage.

5.1. PGAS Support for Virtualizing DIMMs and DRAM Power Efficiency

Previous research has examined the use of HyperTransport over Ethernet as the hardware support for a PGAS implementation that can be used to reduce DRAM overprovisioning in servers in data centers [33]. DRAM in data centers is typically overprovisioned to handle infrequent peaks in workloads, but low-latency memory transfers can help reduce the need for overprovisioning while also providing much lower latency than swapping data out to disk.

These low-latency remote memory accesses provide an alternative to existing RDMA models and also allow for the "virtualizing" of DIMMs on remote nodes. This means that a node could request the use of part of a remote DIMM for noncoherent accesses to grow its own available memory temporarily. At the same time, applications running on the local node are unaware of the DIMM's actual location due to the transparent address mapping of a local HT request address into the global address space, the transmission of a low-latency HToE packet, and traditional CPU techniques that are used to hide normal memory access latency.

DIMM virtualization can provide opportunities for reducing the amount of installed DRAM in a data center, based on average memory requirements rather than peak requirements. For instance, a 10,000 core data center might currently consist of 625 individual blades, each with 4 sockets and quad-core CPUs. Based on previous estimates of memory requirements for data center workloads [6], each blade would require anywhere from 32 to 64 GB of DRAM in an overprovisioned scenario. The current retail price of a registered 8 gigabyte DDR3-1333 DIMM is around $300 [10], so reducing the amount of memory by 50% (from 64 GB to 32 GB) would save $750,000 over the entire data center. A 75% reduction would save $1,125,000 in memory costs alone, not to mention TCO related to cooling and power. Using HP's online power calculator, we can also estimate that this reduction in memory would save either 8,500 Watts (50% reduction) or 12,750 Watts (75%) due to related reductions in idle memory power [19].

5.2. Pooled Accelerators to Reduce Cluster TCO and Power Usage

In addition to virtualizing DRAM, there are also several researchers interested in virtualizing and sharing accelerators, such as GPUs. Provisioning an entire cluster with GPU cards can prove to be cost- and power-inefficient, especially in situations where only a few applications can take advantage of the benefits of better performance on these accelerators. In the same vein as other approaches that utilize MPI or sockets to access remote accelerators [13], HToE can be used as an enabling technology to allow for pooling accelerators (i.e., sharing a few accelerators between a larger number of general purpose nodes) and reducing cost and power inefficiency in the cluster.

While current approaches to use remote accelerator access would likely rely on using HToE packets to perform remote reads and writes to shared CPU-GPU memory pages, it is foreseeable that GPUs could be accessed directly using HyperTransport packets either natively or after being translated over the PCI Express bus. The availability of direct access to GPUs using HyperTransport packets would allow remote nodes to be able to directly read or write GPU DRAM and would provide a much higher performing model for sharing remote accelerators between nodes in a cluster.

Using our example cluster from Section 5.1 with mid-range GPUs, we can give a simplistic approximation of how pooled accelerators could be used to reduce overall cost and power usage. We assume that a blade could potentially house two PCIe-based GPU cards and that these GPUs are not typically fully utilized. The Fermi-branded, NVIDIA GeForce GTX 570 GPU currently retails for around $350 and has a maximum power dissipation of 220 Watts [26] and an idle power dissipation of around 30 Watts [2]. In our pooled accelerator scenario, one GPU could be shared between two adjacent blades, providing a 75% reduction in cost ($328,125 for the entire data center). More importantly, the power consumption due to idle GPUs would be reduced by at least 28,125 Watts (assuming each GPU uses 30 Watts when inactive).

These savings are highly dependent on the expected workload, but the existence of pooled accelerators would allow for much greater flexibility in the initial provisioning and upgrading of clusters to meet computational, power, and TCO requirements.

6. Conclusions

As part of the new HyperShare strategy, HyperTransport over Ethernet (HToE) provides a low-cost, commodity standard that can be used to enable new higher performance models of resource sharing in clusters and data centers. This specification proposes several engineering solutions for encapsulating HyperTransport packets over a highly scalable, many-to-many interconnect, and it provides cost- and performance-related motivation for using HToE in environments where 10 Gigabit Ethernet is already deployed and where 40 or 100 Gigabit Ethernet is likely to gain future market share.

Additionally, we have proposed several usage cases to demonstrate how HToE can be utilized to dramatically improve resource sharing for overprovisioned hardware such as DRAM and expensive accelerators such as GPUs. The HToE standard can enable these sharing techniques in data centers while taking advantage of the cost, scalability, and management benefits associated with Ethernet interconnect technology.

References

[1] IEEE 802.3ae 10Gb/s Ethernet Task Force. 10 gigabit ethernet 802.3ae standard. 2002. http://grouper.ieee.org/groups/802/3/ae/index.html.

[2] Nvidia's geforce gtx 570: Filling in the gaps - power, temperature, and noise. 2011. http://www.anandtech.com/show/4051/nvidias-geforce-gtx-570-filling-in-the-gaps/15.

[3] InfiniBand Trade Association. Rdma over converged ethernet specification. 2010. http://www.infinibandta.org.

[4] Pavan Balaji, Wu-chun Feng, and Dhabaleswar K. Panda. Bridging the ethernet-ethernot performance gap. *IEEE Micro*, 26:24–40, May 2006.

[5] Ulrich Bruening. The htx board: The universal htx test platform. http://www.hypertransport.org/members/u_of_man/htx_board_data_sheet_UoH.pdf.

[6] S. Chalal and T. Glasgow. Memory sizing for server virtualization. 2007. http://communities.intel.com/docs/.

[7] HyperTransport Consortium. Hypertransport specification, 3.10. 2008. http://www.hypertransport.org.

[8] HyperTransport Consortium. Clustering 360 market analysis. 2010. http://www.hypertransport.org/default.cfm?page=Clustering360.

[9] Pat Conway and Bill Hughes. The amd opteron northbridge architecture. *IEEE Micro*, 27(2):10–21, 2007.

[10] Crucial memory 8 gb, ddr3 pc3-10600 memory module pricing. 2011. http://www.crucial.com/server/index.aspx.

[11] Uri Cummings. 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.

[12] D. Dalessandro, P. Wyckoff, and G. Montry. Initial performance evaluation of the neteffect 10 gigabit iwarp adapter. In *Cluster Computing, 2006 IEEE International Conference on*, pages 1–7, 2006.

[13] J. Duato, A.J. Pea, F. Silla, R. Mayo, and E.S. Quintana-Orti. rcuda: Reducing the number of gpu-based accelerators in high performance clusters. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, pages 224 –231, July 2010.

[14] J. Duato, F. Silla, S. Yalamanchili, B. Holden, P. Miranda, J. Underhill, M. Cavalli, and U. Bruning. Extending hypertransport protocol for improved scalability. In *First International Workshop on HyperTransport Research and Applications*, 2009. http://ra.ziti.uni-heidelberg.de/coeht/pages/events/20090212/whtra09-paper16.pdf.

[15] Fibre channel over ethernet fc-bb-5 standard. 2010. http://www.t11.org/fcoe.

[16] Silicon Graphics. Sgi numalink: Industry leading interconnect technology (white paper). 2005. http://www.sgi.com.

[17] IEEE 802.1 Working Group. Ieee 802.1qaz standards page (in progress). http://www.ieee802.org/1/pages/802.1az.html.

[18] Interconnect share of top 500 for november 2010 - hpc top 500. 2010. http://www.top500.org.

[19] Hp power advisor. 2011. http://h18000.www1.hp.com/products/solutions/power/advisor-online/HPPowerAdvisor.html.

[20] Swamy N. Kandadai and Xinghong He. Performance of hpc applications over infiniband, 10 gb and 1 gb ethernet. 2010. http://www.chelsio.com/assetlibrary/whitepapers/HPC-APPS-PERF-IBM.pdf.

[21] M. Ko, D. Eisenhauer, and R. Recio. A case for convergence enhanced ethernet: Requirements and applications. In *Communications, 2008. ICC '08. IEEE International Conference on*, pages 5702 –5707, May 2008.

[22] Rajesh Kota and Rich Oehler. Horus: Large-scale symmetric multiprocessing for opteron systems. *IEEE Micro*, 25(2):30–40, 2005.

[23] Jiuxing Liu, Jiesheng Wu, Sushmitha P. Kini, Pete Wyckoff, and Dhabaleswar K. Panda. High performance rdma-based mpi implementation over infiniband. In *Proceedings of the 17th annual international conference on Supercomputing*, ICS '03, pages 295–304, New York, NY, USA, 2003. ACM.

[24] Myricom's myri-10g 10-gigabit ethernet solutions. 2010. http://www.myri.com/Myri-10G/10gbe_solutions.html.

[25] Juniper Networks. Press release for juniper network's t1600 100 ge core router. 2009. http://www.juniper.net/us/en/company/press-center/press-releases/2009/pr_2009_06_08-09_00.html.

[26] Nvidia geforce gtx 570 specification. 2011. http://www.nvidia.com/object/product-geforce-gtx-570-us.html.

[27] M. Schlansker, J. Tourrilhes, Y. Turner, and J.R. Santos. Killer fabrics for scalable datacenters. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1 –6, May 2010.

[28] 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.

[29] Jon Tate. An introduction to fibre channel over ethernet, and fibre channel over convergence enhanced ethernet. 2009. http://www.redbooks.ibm.com/redpapers/pdfs/redp4493.pdf.

[30] Sudhakar Yalamanchili, Jose Duato, Jeffrey Young, and Federico Silla. A dynamic, partitioned global address space model for high performance clusters. Technical report, 2008. http://www.cercs.gatech.edu/tech-reports/tr2008/git-cercs-08-01.pdf.

[31] Yasushi Umezawa Yoichi Koyanagi, Tadafusa Niinomi. 10 gigabit ethernet switch blade for large-scale blade servers. *Fujitsu Scientific and Technical Journal*, 46(1):56–62, 2010.

[32] Jeff Young and Brian Holden. Hypertransport over ethernet specification, 1.0. 2010. http://www.hypertransport.org.

[33] Jeffrey Young and Sudhakar Yalamanchili. Dynamic partitioned global address spaces for power efficient dram virtualization. In *Works in Progress in Green Computing, 2010 International Green Computing Conference*, 2010.

System-level Prototyping with HyperTransport

Myles Watson and Kelly Flanagan
Computer Science Department
Brigham Young University
Provo, Utah, USA
myles@byu.edu kelly@cs.byu.edu

Abstract— **The complexity of computer systems continues to increase. Emulation of proposed subsystems is one way to manage this growing complexity when evaluating the performance of proposed architectures. HyperTransport allows researchers to connect directly to microprocessors with FPGAs. This enables the emulation of novel memory hierarchies, non-volatile memory designs, coprocessors, and other architectural changes, combined with an existing system.**

Keywords-HyperTransport; FPGA; prototype; emulation;

I. INTRODUCTION

In accordance with Moore's Law, the number of transistors available to chip designers has continued to double every 18 months. For many years, this transistor scaling also enabled increasing central processing unit (CPU) frequencies. Although CPU frequencies and performance increased rapidly, memory and I/O performance increased much more slowly. This disparity increased the importance of I/O and memory performance in computer systems design [1].

In the last few years, power consumption and cooling have caused CPU manufacturers to shift the focus from frequency scaling to scaling the number of processor cores per die [2]. This has exacerbated the pressure on, and the importance of, the memory and I/O subsystems [3].

The increase in importance of memory and I/O subsystems increases the need for understanding system-level design changes, and their impact on performance. Unfortunately, system-level simulation is error prone and costly. One alternative is to emulate part of the system to be studied using field-programmable gate arrays (FPGAs). Connecting the FPGAs to commercial CPUs enables the study of a portion of the I/O subsystem or memory hierarchy, while eliminating the need to faithfully model the CPUs and their internal components.

Designing and implementing an emulation system from scratch would be a costly endeavor, however in-socket accelerators are commercially available at a much lower cost [4]. In-socket accelerators are FPGA boards designed to fit into a CPU socket, and are marketed as flexible application accelerators. They provide low-latency and low-power computational resources for applications such as bioinformatics, data-mining, real-time financial analysis, and oil and gas exploration.

This work describes how an XtremeData XD1000 FPGA board in an AMD Opteron socket can serve as part of a flexible emulation platform. Since the XD1000 tightly couples an Altera Stratix II FPGA with the CPU and other system resources, such as the DRAM sockets on the motherboard, this platform is useful for exploring the design of I/O subsystems and memory hierarchies. Two emulation platforms incorporating the XD1000 are described, each of which is useful for emulating different system designs. Both of these platforms have been implemented, and preliminary performance results in terms of latency and bandwidth for reads and writes are presented for one of the systems.

The remainder of the paper is divided into sections. Section II presents the design of two emulation platforms using the XD1000, along with some of the implementation concerns. Section III describes three target application areas. Section IV presents preliminary performance measurements and discusses the importance of relative performance as an analysis tool. Section V discusses related work. Section VI is the conclusion.

II. SYSTEM DESIGN

An important characteristic of an emulation system is the connection point to the system, which determines the latency and bandwidth of accesses to the emulated device. Two possible locations are a peripheral bus (e.g., PCIe) and the system bus (e.g., HyperTransport or QuickPath Interconnect).

Connecting the emulation platform to a peripheral bus is a flexible and relatively low-cost way to emulate I/O devices and interfaces. Often, an application-specific integrated circuit (ASIC) can be used to connect to the bus, allowing the designer to use the FPGA entirely for the emulated device.

Using an FPGA to connect directly to the processor via the system bus allows lower-latency access to the device. In general, each bus or device through which memory accesses must pass increases the access latency. The option of using coherent (cache-coherent) memory is another benefit of connecting to the system bus.

Coherent memory provides more flexibility in the memory organizations that can be studied, since it can be cached and paged by the microprocessor. From the perspective of the operating system (OS) and applications, this makes it indistinguishable from DRAM connected to a remote processor. Coherent memory allows the study of caching and buffering schemes.

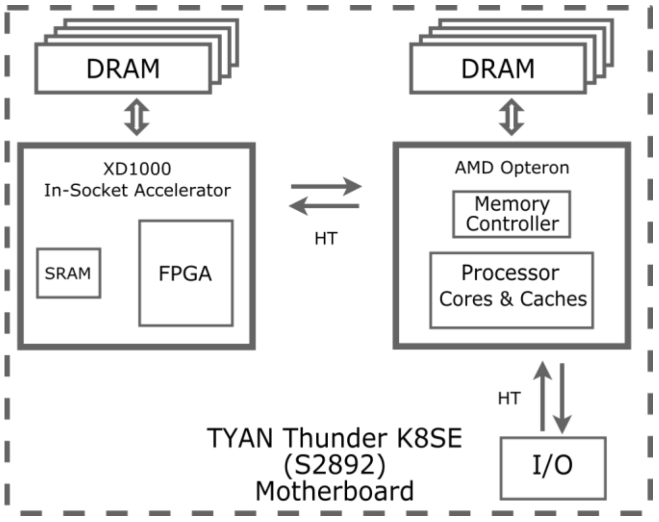

Figure 1. XD1000 in a cave configuration.

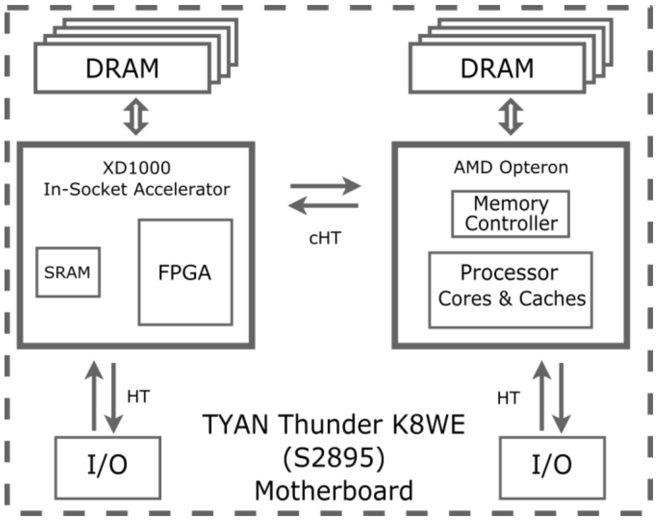

Figure 2. XD1000 in an I/O host configuration.

A. *Coherent HyperTransport*

The coherent HyperTransport (cHT) specification is a superset of the HyperTransport (HT) specification. The HT specification is open, but the cHT specification is only available under NDA with AMD [5,6]. The University of Heidelberg's Center Of Excellence for HyperTransport (CoEHT) has developed HT and cHT cores which can be deployed in FPGAs to connect to AMD Opteron processors through processor-socket interposers (e.g, the XtremeData XD1000) or HyperTransport Extension (HTX) boards (e.g., the CoEHT HTX board [7]).

B. *Architectural Variations*

Opterons and XD1000 modules have three HT links, allowing some flexibility in the configuration of a system. The cHT core adds another option to each configuration. Figures 1 and 2 show two of the configurations available using one or two links. In each case, the link between the Opteron and the XD1000 can be HT or cHT, yielding two additional configurations.

In this work, the XD1000 module is deployed in two Tyan motherboards, the Thunder K8WE (S2895) and Thunder K8SE (S2892). These motherboards were chosen because they are very similar and are supported by coreboot (open-source firmware) [8]. Using coreboot with BIOS emulation routines allows unmodified OSs to be booted, which eases application and driver development [9]. The S2895 has two chipsets, which allows the XD1000 to function as a coherent I/O host. Both configurations have four 1GB DDR DIMMs directly attached to the XD1000.

If main memory is part of the emulated system, cHT is chosen as the connection between the XD1000 and the Opteron. The DRAM connected to the Opteron can then be removed from the system, requiring all memory accesses to be serviced by the XD1000 and the DRAM connected to it. If more than 4 GB of emulated storage is required, the I/O host can connect to I/O devices (PCIe) on the motherboard through a second HT link.

In the configurations shown in Figure 2, where there are multiple HT links, care must be taken to avoid deadlock. HT specifies that no transactions should depend on the completion of other transactions, and transactions should not create new transactions. These guarantees are easily broken by a system which changes the integration level of components, so any new packets must be isolated from the rest of the system. The method of choice is to separate the traffic controlling the I/O devices from the read and write requests from the Opteron. The HT specification requires all packets from devices to traverse the complete chain to the host. This allows the packets to be routed based on their address by the I/O host. In this work, packets are filtered based on their source and destination to make sure that traffic that is part of the emulated system does not reach the CPU.

The XD1000 HT links can run at 200 or 400 MHz using the serializer/deserializer (SERDES) hardware in the FPGA, or at 200 MHz when implemented with DDR registers. When the XD1000 is used as an I/O host on the S2895, at least one of the links is limited to 200 MHz. This is due to a combination of the HT link connecting the XD1000 to the chipset, and limited FPGA resources. Since the links are 16 bits wide and HT is DDR, this provides 800 MB/s of theoretical peak bandwidth in each direction.

C. *Firmware Modifications*

In order to use the XD1000 to emulate multiple system configurations, the firmware which initializes the system must be modified. The modifications can be grouped into three types: XD1000 initialization, address space allocation, and resource reporting. The modifications are more extensive for the I/O host than for the cave.

When used as a cave, the XD1000 initialization is minimal. It consists of an extra hard reset if the HT link is not active. This is necessary to allow the clock generation circuitry of the FPGA sufficient time to stabilize. The resource allocation process must be circumvented for the 4 GB of DRAM, which is allocated above main memory. The Advanced Configuration and Power Interface (ACPI) tables must then be modified so that the XD1000's bus is visible to the OS.

When the XD1000 is an I/O host, it appears to software to be an Opteron processor. It must be programmed with the correct routing values and included in the routing table so that memory accesses reach it correctly. Since the DRAM controller is implemented in the FPGA fabric, the DRAM initialization code needs to be skipped as well. The size of the address space occupied by the emulated storage must be specified, and some ACPI tables must be modified in order for the memory to appear to be attached to node 0. Since there are no processor cores, the code which initializes the Opteron processor cores must be skipped so that the cores appear to be disabled. As a final step, the devices connected to the HT link of the I/O controller, which will be part of the emulated system, must be initialized and hidden from the OS.

D. *Bandwidth and Write Buffering*

The basic unit of transfer in the HyperTransport protocol is the thirty-two bit (four-byte) word. The most efficient transfers (with the lowest overhead) are transfers of 64 bytes. Transfer sizes depend on the Opteron's memory type and page attributes. When the address space is write-back, reads transfer 64 bytes at a time, but writes are performed according to the data size of the store instruction. When the address space is write-combining, the opposite is true.

In order to maximize bandwidth in both directions, the XD1000 example application makes use of DMA engines in the FPGA to transfer data to and from the host memory. This works well when the emulated device is accessed only through a driver, which can set up the transfers. When any size of transfer may be used, this asymmetric performance must be taken into account.

Even with 64-byte transfers, write buffering must be used, since the DRAM controller has a width of 128 bytes. This means that 128 bytes must be read from DRAM before 64 bytes can be written. Much of the complexity involved in creating an application with HyperTransport is a product of the different widths. The 32-bit HT bus protocol is converted by the core to 64-bit data for processing on the FPGA, since FPGAs make better use of wide widths than high clock rates. These data words must be assembled for the DRAM controller. In order to manage this complexity, all writes to RAM are handled by the write buffer, as are any reads that are smaller than 64-bytes.

III. Applications

Many areas of system design can be explored using emulation. Three of the areas that seem most promising are: adding non-volatile memory (NVRAM), adding an application-specific coprocessor (or changing the way one is integrated with the system), and changing the memory hierarchy.

A. *Non-volatile Memories*

Nonvolatile memory technology is advancing. Flash memory is being used as a disk replacement in performance-critical applications. Other technologies, such as phase-change memory (PCM) and spin-torque transfer memory (STTM), are also being developed. Their densities are increasing, and they may be included in future computing systems.

These technologies differ from the DRAM in several important ways, which will influence their integration into computer systems. The two most obvious differences are asymmetric access times for writes and reads, and the need for wear leveling. Both of these factors will influence the design of memory controllers and the resulting performance of applications.

Building prototype systems is prohibitively expensive for exploring the design space, and cannot be done before devices are produced. In order to explore the design space, tools must be developed that will allow accurate performance comparisons for different organizations, block sizes, and wear-leveling and buffering algorithms.

The emulation system of Figure 1 can be used to explore design choices and the interactions of applications with up to 4 GB of NVRAM connected to the system. Programmable delays can be added to the DRAM controller [10] and/or the write buffer in order to more accurately model the access latencies of each technology.

B. *Coprocessors*

One way to increase the time and power efficiency of computation is to use application-specific processors. Many applications have abundant available parallelism. This parallelism can be efficiently exploited by architectures combining many simple, low-power processing elements. General-purpose computing on graphics processing units (GPGPU) is an example of this. The connections between the GPU, the CPU, and memory affect the performance of the application. This could affect how the work is divided among processing units.

The same architectural questions can be explored for general graphics processing. AMD's Fusion architecture more tightly couples the GPU and the CPU in order to achieve higher performance, lower power consumption, or both. An emulated system can be used to explore the design space and performance benefits of such a system before it is built.

C. Memory Hierarchies

The increasing gap between main memory and CPU speed has increased the importance of the memory hierarchy in system performance. Much of the area on recent CPU dies is dedicated to caches. There is a large design space to be explored, and its complexity is increasing with the number of processor cores. Structures such as coherence directories are good candidates for emulation, since they can be implemented with the RAM resources of the FPGA.

One extension to the memory hierarchy which can be explored using emulation is a hardware single-level store, which moves control of swapping pages of memory from the OS into hardware. Swapping is a feature of virtual memory when the virtual memory space is larger than physical RAM. Memory pages are swapped when pages of data are transferred to and from the secondary store to maintain the illusion of large memory space. If a page is chosen for replacement that will be used again soon, its next access will cause another swap. Since secondary storage is much slower than RAM, minimizing swapping is essential to performance. Some related features, such as file caching, can also be controlled by the same hardware, since the files reside in the secondary store and get moved to RAM for faster access.

Hardware paging support is interesting because there is limited information available to the OS about page usage. Usage bits are only updated during page table walks, which occur on TLB misses. In order for an OS to collect more usage information, it must invalidate TLB entries to cause misses, which is expensive. With more information, paging algorithms make better replacement decisions, increasing performance [11]. A hardware paging implementation would be aware of all memory accesses that miss the last level of cache, and therefore have more information on which to base page replacement decisions.

Moving paging support out of the OS is not a new idea. The IBM AS/400 and its predecessor, the IBM System/38, implement paging in virtual machines. This simplifies software development, since from the perspective of the OS and applications, memory is flat and uniform [12]. A virtual machine implementation of paging suffers the same performance penalties as other software implementations, due to limited usage information,.

IV. PERFORMANCE

Performance measurements and comparisons are two of the most compelling reasons to emulate modifications to computer systems. Although the most straightforward way to measure system performance is by measuring wall clock time, it is not the most helpful metric for comparing emulated systems. Although the FPGAs used for emulation continue to improve in speed, they are not as fast as a final implementation.

TABLE I. READ AND WRITE BANDWIDTH MEASUREMENTS.

Transaction Type	**Bandwidth (MB/s)**
32-bit writes	60
64-bit writes	90
64-byte writes (write-combining)	120
32-bit reads	5.5
32-bit reads (two threads)	11
64-byte reads (cacheable 32-bit)	50
64-byte reads (two threads)	92

A. Preliminary Performance Measurements

In order to understand the performance characteristics of a system, simple latency and bandwidth measurements are taken. The system shown in Figure 1 is booted into Linux, and a modified device driver based on the example XD1000 driver is loaded. A simple application is then run, which calls mmap to obtain a pointer to the 4GB of memory on the XD1000. Once the program has a pointer, it is straightforward to write timing loops which measure the average latency and bandwidth of memory accesses. The measured latencies can be verified using Altera SignalTap to view the HT requests.

The latency for each read or write targeting the DRAM is around 850 ns, with the write buffer implemented, but no workload-specific optimizations. This yields varying bandwidths depending on the transaction types and sizes, as shown in Table 1. Because the write buffer is organized as a cache, each write to a new line causes a line fill from the DRAM, and possibly a write back for dirty data. An obvious performance optimization is to bypass the write buffer when multiple consecutive writes are received, and write a full 128 bytes directly to DRAM. Avoiding the write buffer in this way would substantially increase the write bandwidth. Note that read bandwidth is significantly lower than write bandwidth because each read must complete before software can issue another read; writes have no such restriction. Running two threads nearly doubles the read bandwidth because the two processor cores can issue reads in parallel, but it has no effect on write bandwidth.

B. Relative Performance Comparisons

Using absolute performance numbers with emulated architectures can be misleading. The solution is to use relative performance comparisons. Some of the factors that make relative performance comparisons more appropriate than using absolute performance include: the lower frequency of an FPGA implementation of HyperTransport, the fact that the emulated prototype may not be fully optimized, and even restrictions with the NDA in publishing performance numbers for the coherent core.

In order to compare the performance of multiple non-volatile memory technologies and their controllers, the path for each access should be equivalent. This means that a comparison between the delayed RAM on the XD1000 and the RAM attached to the host Opteron would be much less informative than a comparison between two delay settings on the XD1000.

For the case of an emulated single-level store, the only DRAM in the system is attached to the XD1000, and all requests must traverse the same path. The difference being measured can then be attributed to the difference in the paging algorithm, and the information available to it. The latency of a memory access in this scenario is the sum of the latencies due to: the HT link, the write buffer access, the DRAM access, and in the case of a miss, a page transfer from the backing store to DRAM.

When making the baseline measurements, the Opteron is initialized to access 4 GB of RAM with the XD1000 as the only memory controller. Memory needs beyond 4 GB must be supplied by OS-controlled paging to the secondary storage. The baseline is then compared to the same configuration, but hardware paging is enabled and the XD1000 is initialized as a memory controller with up to 1 TB of storage addressable as RAM. The 1 TB limit is a hard limit dictated by the 40 physical address bits available to the processors. Newer Opterons have 48 physical address bits, expanding their addressing capabilities to 256 TB.

V. Related Work

There are many system-level simulators, but there are relatively few systems which add emulation to an existing system using FPGAs. In this section, a case is presented for using emulation in place of full-system simulation. This analysis is followed by a discussion of three related emulation systems, and two FPGA prototype systems that use HT to enable low-latency cluster interconnects.

A. *Emulation vs. Simulation*

Several factors make system-level simulation time consuming, expensive, and error-prone. These include the asynchronous interactions among multiple devices, the closed nature of many CPUs, the complexity of these CPUs and their interconnects, and the increasing sizes of caching structures and translation look-aside buffers (TLBs).

Since modern computer systems incorporate many diverse components, modeling their interactions faithfully can be difficult. Computer systems include devices ranging from PCI Express (PCIe) graphics cards to hard drives to serial ports, with widely varying performance characteristics and latencies. Modeling the system at a sufficient level of detail to accurately reflect system performance is a challenge.

Modern CPUs have complex performance characteristics, which can be difficult to model [13]. Although some high-level details of CPU architectures are available, many of the details needed for accurately simulating their performance are not. Even if all the design parameters are available, the complexity of faithful modeling slows simulations significantly, and it is difficult to assure the correctness of the final model. This also applies to the interconnections among CPU cores and the connections to other subsystems. Multi-core architectures exacerbate this problem.

As storage structures such as caches and TLBs increase in size, the amount of simulated run time needed in order to characterize their performance increases. Measuring the benefit of another level of cache, for example, will require the benchmark to generate many misses in the previous levels.

Emulation is a promising way to reduce the complexity involved in understanding the effects on performance of modifications to an existing system. FPGAs combine programmable logic and I/O interfaces, and some contain implementations of simple microprocessors. This makes them suited to implement a wide variety of functions for experimentation. Their performance is limited in terms of maximum clock frequency, but many times that can be mitigated by the high degree of fine-grained parallelism available in them.

Emulated subsystems implemented in an FPGA run fast enough to allow multiple benchmark runs. These multiple runs add statistical significance to performance measurements of the emulated systems and minimize the effect of performance variability of the other system components.

B. *Emulation Systems*

Three related FPGA emulation systems are Flexible Architecture Research Machine (FARM) [14], Research Accelerator for Multiple Processors (RAMP) [13, 15], and High-performance Advanced Storage Technology Emulator (HASTE) [10].

FARM is similar to this work, in that it modifies and repurposes an existing FPGA and Opteron system in order to explore system architecture. FARM differs from using an in-socket accelerator because the original system is much more expensive, and the FPGAs are not directly connected to the DDR or chipset on the motherboard.

RAMP is a collaborative effort by a number of researchers to enable comparable architectural research and bring down the costs associated with FPGA emulation, specifically for many simple cores and their interconnects. In order to achieve this goal, RAMP specifies FPGA boards, and encourages the sharing and reuse of design components for the FPGA designs. RAMP focuses on the challenges of multi-core architectures and the software which runs on them.

HASTE is a system constructed by UCSD to evaluate NVRAM technologies in supercomputing applications. HASTE connects DRAM with an FPGA controller on a PCIe card, and is compared with the system DRAM and solid-state disks to explore the performance of storage devices built from emerging NVRAM technologies.

C. Low-Latency Cluster Interconnects

Two systems which use FPGAs with HT to prototype low-latency cluster interconnects are the Virtualized Engine for Low Overhead (VELO) [16], and the Hyper Parallel Processing (HPP) architecture [17].

VELO is an implementation of a network engine using an HTX card. The resulting network exhibits latencies of just over 1 μs, including routing.

HPP connects multiple motherboards with an HT backplane and a switch implemented with an FPGA. The HPP prototype demonstrates low-latency, high-bandwidth connections between motherboards in a prototype high-performance, low-cost cluster.

Both VELO and HPP are specifically designed to prototype connections between systems, whereas systems using in-socket emulators are better suited for emulating and prototyping modifications to parts of a single system.

VI. Conclusion

This work demonstrated how HT and FPGAs can be used in commodity systems to emulate and evaluate the performance of proposed system modifications. The ability of the XD1000 to connect directly to the motherboard HT links was shown to allow the exploration of many system configurations. Two of these configurations were presented, along with preliminary performance results from one of them. These emulation systems were presented as a viable way to evaluate new technologies such as NVRAM, and the many ways that they can be incorporated into computer systems.

Acknowledgments

Thanks to Heiner Litz, Maya Gokhale, and the anonymous reviewers for their comments and helpful suggestions.

References

[1] W. A. Wulf and S. A. McKee. 1995. “Hitting the memory wall: implications of the obvious,” SIGARCH Comput. Archit. News 23, 1 (March 1995), 20-24.

[2] K. Asanović, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick. “A view of the parallel computing landscape,” Commun. ACM 52, 10 (October 2009), 56-67.

[3] P. Conway and B. Hughes. “The AMD Opteron northbridge architecture,” IEEE Micro 27, 2 (March 2007), 10-21.

[4] XtremeData web site, http://www.xtremedata.com/.

[5] HyperTransport Center of Excellence web site, http://ra.ziti.uni-heidelberg.de/coeht/

[6] HyperTransport Consortium web site, http://www.hypertransport.org/.

[7] H. Fröning, M. Nüssle, D. Slogsnat, H. Litz, U. Brüning, “The HTX-board: a rapid prototyping station,” Proc. Of 3rd annual FPGAworld Conference, Nov. 16, 2006, Stockholm, Sweden.

[8] Coreboot web site, http://www.coreboot.org/.

[9] A. Agnew, A. Sulmicki, R. Minnich, W. A. Arbaugh: “Flexibility in ROM: a stackable open source BIOS,” USENIX Annual Technical Conference, FREENIX Track 2003: 115-124.

[10] A. M. Caulfield, J. Coburn, T. I. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R. K. Gupta, A. Snavely, and S. Swanson, “Understanding the impact of emerging non-volatile memories on high-performance, IO-intensive computing,” SC'10: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, New Orleans, Louisiana, Nov. 2010.

[11] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar. “Dynamic tracking of page miss ratio curve for memory management,” In Proceedings of the 11th international conference on Architectural support for programming languages and operating systems (ASPLOS-XI). ACM, New York, NY, USA, 177-188.

[12] F. G. Soltis, Inside the AS/400, second ed. Duke Communications, Loveland, CO, 1997.

[13] J. Wawrzynek, D. Patterson, M. Oskin, S. Lu, C. Kozyrakis, J. C. Hoe, D. Chiou, K. Asanović. “RAMP: Research Accelerator for Multiple Processors,” IEEE Micro, 27(2):46–57, 2007.

[14] T. Oguntebi, S. Hong, J. Casper, N. Bronson, C. Kozyrakis, K. Olukotun, “FARM: a prototyping environment for tightly-coupled, heterogeneous architectures,” FCCM '10: The 18th Annual International IEEE Symposium on Field-Programmable Custom Computing Machines, May 2010.

[15] Z. Tan, A. Waterman, R. Avizienis, Y. Lee, H. Cook, D. Patterson, and K. Asanović, “RAMP gold: an FPGA-based architecture simulator for multiprocessors,” In Proceedings of the 47th Design Automation Conference (DAC '10). ACM, New York, NY, USA, 463-468.

[16] M. Nüssle, B. Geib, H. Fröning, and U. Brüning, “An FPGA-based custom high performance interconnection network,” In Proceedings of the 2009 International Conference on Reconfigurable Computing and FPGAs (RECONFIG '09). IEEE Computer Society, Washington, DC, USA, 113-118.

[17] X. Yang, F. Chen, H. Cheng, N. Sun, “A HyperTransport-based personal parallel computer,” Cluster Computing, 2008 IEEE International Conference on, pp.126-132, Sept. 2008.

A HT3 Platform for Rapid Prototyping and High Performance Reconfigurable Computing

Frank Lemke, Sven Kapferer, Alexander Giese, Holger Fröning, Ulrich Brüning
Computer Architecture Group
University of Heidelberg
Mannheim, Germany

{frank.lemke,sven.kapferer,alexander.giese,holger.froening,ulrich.bruening}
@ziti.uni-heidelberg.de

***Abstract* — FPGAs as reconfigurable devices play an important role in both rapid prototyping and high performance reconfigurable computing. Usually, FPGA vendors help the users with pre-designed cores, for instance for various communication protocols. However, this is only true for widely used protocols. In the use case described here, the target application may benefit from a tight integration of the FPGA in a computing system. Typical commodity protocols like PCI Express may not fulfill these demands. HyperTransport (HT), on the other hand, allows connecting directly and without intermediate bridges or protocol conversion to a processor interface. As a result, communication costs between the FPGA unit and both processor and main memory are minimal. In this paper we present an HT3 interface for Stratix IV based FPGAs, which allows for minimal latencies and high bandwidths between processor and device and main memory and device. Designs targeting a HT connection can now be prototyped in real world systems. Furthermore, this design can be leveraged for acceleration tasks, with the minimal communication costs allowing fine-grain work deployment and the use of cost-efficient main memory instead of size-limited and costly on-device memory.**

Hyper Transport, FPGA, High Performance Reconfigurable Computing

I. INTRODUCTION

In the area of accelerated computing the vast amount of research and development focuses on using GPUs [1] [2] [3]. Compared to this, FPGAs are very sparely used. The main reasons for this are certainly the cost advantage of GPUs (with a mass market behind), and the easier way of programming. FPGAs are for most users difficult to program, and due to their small volume they have approximately one order of magnitude higher costs.

However, GPUs are very limited in their usage. Only if the application to be ported to the accelerator has characteristics similar to graphical processing, it can be successfully accelerated [4]. Additionally, a recent report by Intel [5] shows that the speedup between CPUs and GPUs is only about 2.5 in average. Also, the limited amount of graphics memory is preventing a broad use, because the stream processors of a GPU can only operate on this memory.

FPGAs, on the other hand, are much more flexible due to their completely reconfigurable architecture. In particular for applications which are not suitable for GPUs they play an important role [6] [7] [8]. It is also possible to attach a large amount of memory to the FPGA, making it suitable for data-intensive applications.

GPUs with their stream based processing do not rely on a close coupling between accelerator and host system, thus they cannot offer applications the possibility of fine grain accesses to and from the host system. However, many applications rely on such a tight integration. Again, this demand can be fulfilled by FPGAs, in particular if a system interface like HT is used and not a peripheral interface like PCIe. If the interface to the host system is lean enough, the costs for accessing main memory are not higher than accessing memory attached to the FPGA. Then, it is possible for the FPGA to operate directly on main memory, making arbitrary amounts of memory possible.

Last, as more and more performance computing systems are facing the power wall, the GFLOPs achieved per Watt are of paramount importance. FPGAs are certainly one of the best architectures for high GFLOPs/Watt. By equipping installations with FPGA based accelerators, the power consumption can be significantly reduced while maintaining the computing performance.

As hardware platform enabling above described features a Stratix IV HTX3 Board was used. Based on an existing first version prototype it was enhanced by placing additional components onto the board and some refinements resulting in the version presented here providing all required basic functionalities. For using it as fully capable HT3 device in a system the HT3 core [9] had to be ported onto the Altera FPGA.

Also to ensure the usability and reliability of communication between the device and the processors in HT3 systems HW simulations had to be performed. Additionally a physical interface (PHY) had to be created to deliver an interface for the HT3 core to be compatible with the provided hardware environment. This work will enable the Stratix IV HTX3 Board being used as a unique single FPGA HT3 solution which supports all the required features for HT3 and therefore representing an efficient platform for Rapid Prototyping and High Performance Reconfigurable Computing.

The next section presents the HyperTransport protocol as base technology for low latency communication. The architecture of the Stratix IV HTX3 Board serving as rapid prototype platform is specified in section 3 followed by the description of the HT3 implementation enabling high performance reconfigurable computing on top of it in section 4. The fifth section presents measurement results. Finally a conclusion and an outlook are given in section 6.

II. HyperTransport

HT is a unique possibility to easily connect a device directly to a processor. As it is the only public specification [10] available to do so, it is the perfect vehicle for a low latency communication as there are no unnecessary protocol conversions or bridges involved. With the HTX3 connector which is defined by the HyperTransport-Consortium (HTC) [11] and the availability of Opteron mainboards a system can be easily set up [12].

HT allows a broad variation of link widths and frequencies from a 2 bit link at 200 MHz DDR (HT200) up to a 32 bit link at 3.2 GHz DDR (HT3200). Current Opteron architectures support link widths and frequencies from 8 bit at 200 MHz DDR up to 16 bit at 3.2 GHz DDR. This results in a theoretically maximum unidirectional bandwidth of 12.8 GB/s. The signal lines carry the control-, data- and info-packets and are called CAD. Depending on the link width those signals are grouped into independent byte lanes. Every byte lane is accompanied by a single signal lane of additional control information called CTL and a clock signal. As HT is doubleword (32 bit) aligned every doubleword of CAD comes along with 4 bit of CTL which contains additional information about what kind of data is transported.

Three types of the specification exists HT1, HT2 and HT3. HT1 and HT2 only differ in the maximum link frequency. The functionality of the first two versions is described in [13]. HT3, which is realized in state of the art Opteron processors, begins at a link speed of HT1200 and requires features to be implemented such as link training, link deskew, a retry protocol and stomping which were in earlier versions optional or not defined.

To realize link training, each bit lane has to support a mechanism to align its logic with the help of a special training pattern sequence. After link initialization the single bit lanes are deskewed to ensure proper data alignment. Therefore the receiving fifos must be able to handle an amount of 8 bit-times of misalignment from one lane to another. Compared to HT1 and HT2, the higher frequencies of HT3 result in an increased possibility of bit errors on the physical level. A retry mechanism is introduced to handle those errors. The error detection is enhanced due to changing the periodic CRC from HT1/HT2 every 512 bit times to a per packet CRC. Thereby latency and the needed buffer space for retransmission are reduced and a better performance can be achieved. Each packet which CRC is checked correctly increments an acknowledgement counter. If a NOP packet is sent it contains the counter value of the last correctly checked packet. The retry buffers on the receiver side of the NOP packet can then be released. If an error occurred during a transmission the retry handshake is initiated and the data from the last correctly received packet is retransmitted. Stomping is an additional feature to reduce latency. It is used to speculatively forward a packet without CRC being checked. If later the CRC shows an error the CRC is inverted to show the final endpoint that the packet has to be invalidated. A block diagram of the HT3 implementation is shown in figure 1.

HT3 leverages the possibility of higher link speeds by introducing fault detection and recovery mechanism to the HyperTransport protocol. But it requires changes on the physical layer as well which will be described in section 4.

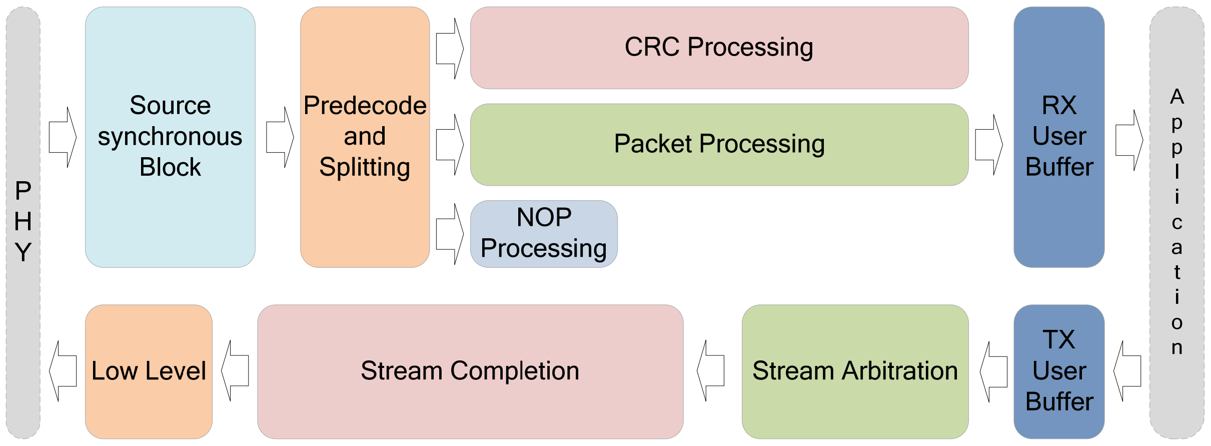

Figure 1. HT3 Blockdiagram

III. Board Architecture

The board design is based on a PCI normal sized card using a HTX3 connector slot for a low latency HT link connection to the system. The main board component is a Stratix IV GX device family FPGA [14]. The selected FPGA uses a F1517 footprint enabling EP4SGX 180, 230, 290, 360 and 530 variants. The used device provides an adequate number of LVDS and I/O pins to enable numerous prototyping features and 36 transceivers giving the capability to use HT3 and two CX4 links with up to 6.375 Gbps per lane for network connections. The CX4 links do support implementations for Infiniband DDR. There are also standard interfaces and components available to use the board in different environments.

For prototyping purposes using extension cards or user defined connectors extension adapters have been placed onto the board. The primary used adapter is a SEAF connector from Samtec with 500 pins supporting single-ended signaling up to 9.5 GHz and differential pair signaling up to 10.5 GHz. Thus speed restriction is primarily defined by the FPGA. The pins used within the connector are shielded considering the suggestions of Samtec. This resulted in 114 single-ended and 55 differential pair connects together with the FPGA.

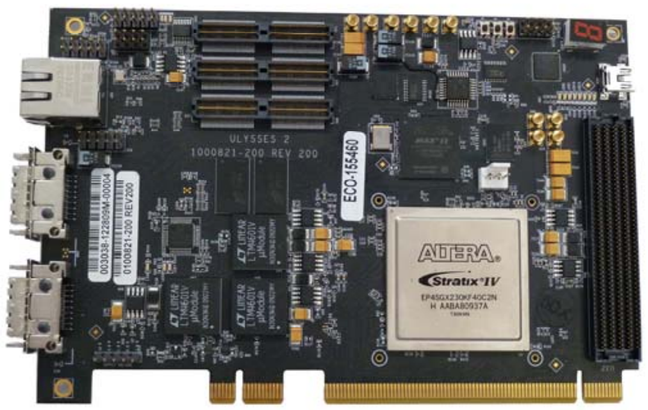

Figure 2. Stratix IV HTX3 Board

Further three QTH series Samtec connectors with 120 pins each organized in two banks with integrated metal plane used as ground are assembled. These connectors provide at least 9GHz single-ended and 8 GHz differential pair capability. The connections to the FPGA are designed to provide up to 108 differential pairs plus sideband signals.

The board was enhanced and upgraded in several design steps. Figure 2 shows the latest revision. All components are tested. It can be used as a prototyping platform or directly for high performance reconfigurable computing needs.

IV. HT3 Implementation

Before porting the HT3 core onto the Altera device an implementation of a PHY had to be realized. Therefore the high frequency traces had to be simulated ensuring that all parameters were within the specification.

A. Simulation

During simulation of the HT link all HT tracks between the Opteron processor and the Stratix IV GX were analyzed. All simulations were performed using IBIS and HSpice models. For the FPGA high speed serial transceiver an HSpice model and for the Opteron processor IBIS models were available. For the HTX connector, which is identical to the PCIe connector, the Samtec Spice model has been used. The required S-Parameter files among others for the vias are generated by the Cadence Allegro PCB design suite. HT3 starts with a minimum of HT1200 with a frequency of 1200MHz and a data rate of 2.4Gbps. This was also the simulation target for the first simulations. Figure 3 shows a representation of the simulated tracks at HT1200 for a HTX3 CADOUT signal. There are three different measure points available, the signal after the Stratix IV GX package, on the receiver pin, and after equalization through the Stratix IV GX Clock Data Recovery (CDR) unit. Depending on the measure point the eye height is in the range from 531 mV to 998mV and the eye width is around 374ps. According to the HT physical specification [11] the eye height must be over 140mV and the minimal eye width must be 0.55 unit intervals (UI), the UI for 2.4Gbps is 416ps. All simulated tracks at HT1200 were clearly within the specification.

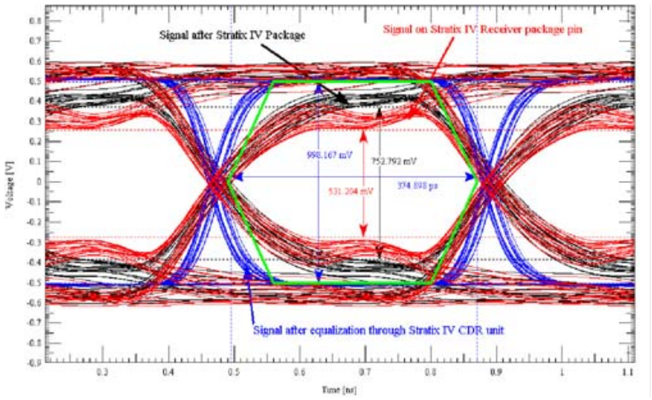

Figure 3. HTX Track Simulated at HT1200

Also simulations using the maximum frequency of the high speed Stratix IV GX transceivers at 6.4Gbps were performed. The HT specification for this frequency requires a minimum eye width of 0.65 UI, which results in 100ps and a minimum eye height of 170mv. One of the most critical extracted tracks is depicted in figure 4. Its eye width is 107ps and the height is 224mV. All simulations show, that the hardware is capable of HTX3 usage.

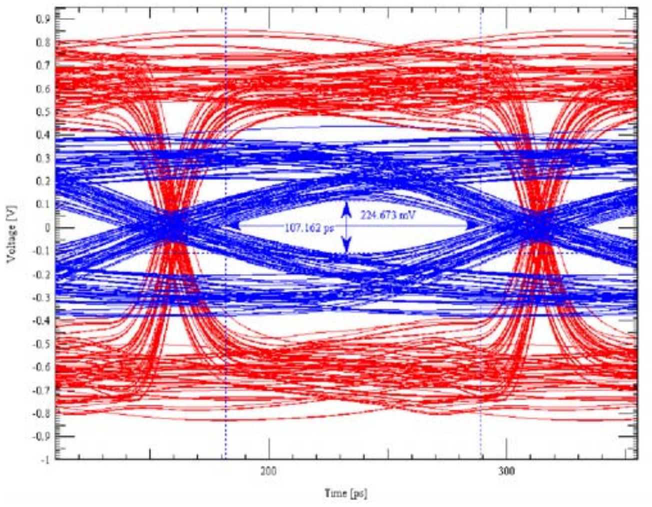

Figure 4. Eye Diagram at 6.4Gbps Link Speed.

B. HT3 PHY

A PHY for HyperTransport 3 must also support HT1 operation because the HyperTransport protocol is backwards compatible. However, this means that the PHY must support two inherently different operation modes. HT1 is working in a source synchronous mode and transmits a link clock in addition to the data lanes which is used to sample the incoming data. Since HT3 operation starts at a link frequency of 1.2 GHz and can go up to 3.2 GHz a different technique must be used. Because the skew requirements between clock and data would be in the range of picoseconds if the same source synchronous mode was used for HT3 frequencies the clock is now recovered at the receiver side by using CDR. In order to ensure enough transitions for a reliable clock reconstruction scrambling is mandatory for HT3 operation. The HyperTransport protocol specification defines several line rates for HT3 operation in the range of 2.4 Gbps to 6.4 Gbps. Because these line rates exceed the maximum supported data rate of LVDS transmitter / receivers by far, high speed serializers must be used to work in HT3 mode. In order to implement proprietary protocols the Stratix IV GX transceivers support an operating range from 600 to 3750 Mbps in single width mode using an 1:16 serialization factor and from 1000 to 6500 Mbps in double width mode with an 1:32 ratio for the -2 speed grade we used for our board. In Stratix IV devices transceivers are grouped in blocks consisting of 6 transceivers as shown in figure 5. Four of those channels support both physical coding sublayer (PCS) and physical medium attachment (PMA), the other two channels are clock multiplier unit (CMU) channels that can be configured either as a normal data channel without PCS support or as a clocking block that provides both the serial and the parallel clock to the other channels.

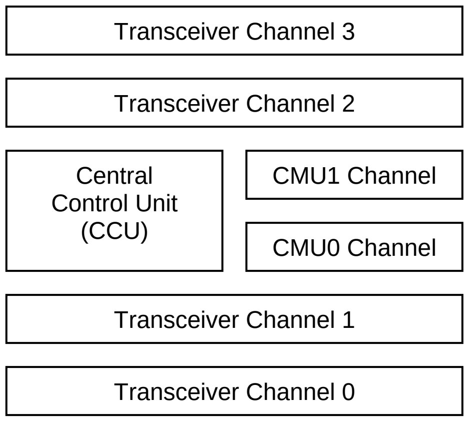

Figure 5. Stratix IV Transceiver Block Architecture [14]

For each sublink, 9 lanes (8 CAD + 1 CTL) are connected to the fully featured serializers and both the incoming and outgoing link clock are connected to separate CMU channels. In order to provide a deterministic latency across all channels the transceivers are configured in PMA direct mode. All required PCS functionality is provided inside the FPGA, the existing transmitter and receiver PCS blocks in the hardware are completely bypassed. Since all HyperTransport clocks are derived from the 200 MHz HT reference clock this clock is also connected to a global clock pin of the Stratix IV FPGA after it was jitter cleaned to improve the transceiver performance.

Although the HyperTransport 3 specification specifies an AC coupled operation mode as well as a DC coupled mode, AC coupled operation is not supported by AMD’s Opteron processor and therefore irrelevant for all practical purposes. Since the HyperTransport specification and the Stratix IV datasheets define different common modes electrical compatibility between the Opteron and Altera’s transceivers had to be verified by HSPICE simulations as described in the previous paragraph and also confirmed by Altera engineers.

All HyperTransport systems start in Gen1 mode running with a 200 MHz link clock (HT200). The resulting data rate of 400 Mbps is below the minimum supported rate of the Stratix IV transceivers. In order to overcome this limitation, the PHY runs five times faster than actually required by the data rate and uses a 5 time oversampling mode for incoming data. In the same way, for the outgoing direction each bit is just replicated 5 times to emulate a link running at HT200.

Since HT1 employs neither scrambling nor 8b/10b encoding clock recovery from the data stream cannot be used, therefore the transceivers are configured in lock-to-reference (LTR) mode and use the HT reference clock to create the sampling points. The link clock on the receiving side is not used to sample the incoming data. In order to create the transmit clock that must be shifted by 90 degrees in relation to the data stream as defined in the HT specification the clock data pattern is padded accordingly so that the clock is driven one half of a bit-time after a data transition. As described above all PCS handling is done in the FPGA fabric. This means that the PHY only handles the basic serialization and deserialization, data word boundaries are not detected at all by the PHY. All alignment is done later inside the HT3 core.

In order to switch to HT3 mode, starting at 1.2 GHz, several things must happen inside the PHY. The oversampling path that was used for HT200 must be bypassed; the data is now processed directly as the link data rate is now natively supported by the transceiver. The transceivers also switch from LTR to CDR and the clock recovery circuitry must lock to the data stream. This means that each lane has its own recovered clock that is used to sample the data. Although each of these lanes will run at the same frequency there will be a phase difference between the different lanes. Elastic buffers are used in order to transfer all the lanes in a single clock domain to process the data stream in parallel. Unlike in HT1 mode this can also lead to inter-lane skew which will also be removed in the HT3 core. The link clock in HT3 is not used at all and requires no special handling since there is no relation between clock and data and each lane has its own embedded clock.

The PHY also supports the LDTSTOP signal defined by HyperTransport specification that is used to disconnect the links. During this time no data is transmitted over the link and the link is idle. Because the CDR circuitry does not recover reliably from this condition after the link restarts the PHY switches back to LTR mode during LDTSTOP and goes back to CDR after the link resumes normal operation and scrambled data patterns are transmitted again.

The PHY does not include any error detection mechanisms. All signal integrity issues are caught by the HT3 core using the reliability features defined in the HT3 protocol.

V. Measurements

The measured latency of our HT3 core together with the HT3 PHY at a HyperTransport link frequency of 1600 MHz running in 8 bit mode in a Tyan 2912-E motherboard with two Opteron processors running at 2800 MHz was 655ns round trip for a single PIO access to the device. This is much higher than the latency measured for our HT1 core [15] in an older system with a slower processor. There are several reasons for this large difference. The higher complexity of the HT3 protocol forced us to implement more pipeline stages to decode the incoming packets. The most prominent factor, however, is the usage of serializer technology inside the FPGA instead of normal LVDS IO cells and the crossing of several clock domains inside the PHY.

The first bandwidth measurements showed rather disappointing results that were not even in the range of half the available bandwidth offered by the link. This was caused by credit starvation [9] because the default BIOS configuration of the link did not allocate enough credits for the posted VC inside the processor. After redistributing the credits to achieve a better link utilization bandwidth measurements using data packets with the maximum allowed payload of 64 bytes showed a write performance of about 2000 MB/s for a DMA Write operation and an average bandwidth of about 1600 MB/s for a DMA Read operation. These numbers, albeit being a huge improvement, show that full utilization of a HT link can only be reached by a device with a fast internal clock speed that can release credits almost instantaneously as soon as new packet is received. The performance that can be reached by an FPGA suffers mainly from the credit starvation that occurs during operation that is caused by the latency added by the serializers and the many pipeline stages in the core.

The consumption of resources within the FPGA shows that there is enough space left to add user logic for prototyping and high performance computing. The synthesis results for the Stratix IV GX 230 device depict resource usage of combinational ALUTs 42,534 / 182,400 (23 %), memory ALUTs 49 / 91,200 (< 1 %), dedicated logic registers 40,009 / 182,400 (22 %), a logic utilization of 34 %, and a total block memory bits 739,154 / 14,625,792 (5 %).

VI. Conclusion and Outlook

Both the HT3 PHY in conjunction with the HT3 core and the developed Altera FPGA based card work reliably in our Tyan test system. Sporadic bit errors that were encountered during operation were easily caught and recovered by the reliability features defined in the HT3 protocol and had no impact on the functionality.

Both HT1200 and HT1600 implementations are stable and work as expected. Unfortunately, the core speed directly scales in relation to the HT link speed as there is no flow control between the PHY and the HT3 core. Thus, reaching higher HT link speeds is currently limited by the HT3 protocol that leads to a complex hardware architecture for the HT3 core and makes internal core frequencies larger than 200 MHz rather difficult to achieve.

The HT3 platform for rapid prototyping and high performance reconfigurable computing was a successful development. It represents the first single FPGA HT3 implementation in comparison to the 3 FPGA solution developed in [16]. Due to the provided low latency high bandwidth connection directly to the processor this platform delivers an ideal environment for developments and research in the areas of coprocessors or FPGA accelerators. Also its numerous extension connectors enable the usage of extender cards such as a card with a Content-Addressable Memory (CAM) and a reasonable amount of RAM to realize a network search engine (NSE).

Acknowledgment

We would like to express our thanks to Elmar Greulich for the work he brought into the first revision of the Stratix IV HTX3 Board. We also thank Altera for the PHY development and AMD for their excellent support.

References

[1] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., and Purcell, T. 2007. A Survey of General-Purpose Computation on Graphics Hardware. Computer Graphics Forum, volume 26, number 1, 2007, 80-113.

[2] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E. and Phillips, J. C. 2008. GPU Computing. In Proceedings of the IEEE, 96, 5 (May 2008), 879–899.

[3] Alerstam, E., Svensson, T., and Andersson-Engels, S. 2008. Parallel computing with graphics processing units for high-speed Monte Carlo simulation of photon migration. In *Journal of Biomedical Optics, vol. 13, issue 6*, Nov. 2008.

[4] Khailany, B., Dally, W. J., Kapasi, U. J., Mattson, P., Namkoong, J., Owens, J. D., Towles, B., Chang, A., and Rixner, S. 2001. Imagine: Media Processing with Streams. *IEEE Micro 21, 2* (Mar. 2001), 35-46.

[5] Lee, V. W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R. and Dubey, P. 2010. Debunking the 100X GPU vs. CPU Myth: an Evaluation of Throughput Computing on CPU and GPU. *SIGARCH Comput. Archit. News* 38, 3 (June 2010), 451-460.

[6] Das, S., Agrawal, D., and Abbadi, A. E. 2008. CAM conscious integrated answering of frequent elements and top-k queries over data streams. In *Proceedings of the 4th International Workshop on Data Management on New Hardware*, Vancouver, Canada, June 2008.

[7] Bandi, N., Metwally, A., Agrawal, D., and El Abbadi, A. 2007. Fast data stream algorithms using associative memories. In *Proceedings of the 2007 ACM International Conference on Management of Data (SIGMOD '07)*, Beijing, China, June 2007.

[8] Fröning, H., Nüssle, M., Litz, H., and Brüning, U. 2010. A Case for FPGA based Accelerated Communication. In *Proceedings of 9th International Conference on Networks (ICN 2010)*, Menuires, France, April 2010.

[9] B. Kalisch, A. Giese, H. Litz and U. Bruening, “Hypertransport 3 Core: A Next Generation Host Interface with Extremely High Bandwidth“, First International Workshop on Hyper Transport Research and Applications (WHTRA), Mannheim, Germany, Feb. 2009.

[10] HyperTransport™ Consortium: HyperTransport™ I/O Link Specification, http://www.hypertransport.org, June 2010

[11] HyperTransport™ Consortium: HTX3™ Specification for HyperTransport™ 3.0 Daughtercards and ATX/EATX Motherboards, http://www.hypertransport.org, Jun. 2008.

[12] Hypertransport Consortium: The Future of High-Performance Computing: Direct Low Latency CPU-to-Subsystem Interconnect, http://www.hypertransport.org, 2008.

[13] D. Anderson, and J. Trodden, HyperTransport System Architecture, Addison-Wesley, 2003.

[14] Altera: Stratix IV Device Handbook, SIV5V1-4.1, http://www.altera.com, July 2010.

[15] David Slogsnat, Alexander Giese, Mondrian Nüssle, Ulrich Brüning, “An Open-Source HyperTransport Core”, ACM Transactions on Reconfigurable Technology and Systems (TRETS), Vol. 1, Issue 3, p. 1-21, Sept 2008.

[16] Heiner Litz, Holger Fröning, Maximilian Thürmer, Ulrich Brüning, “An FPGA based Verification Platform for HyperTransport 3.x”, 19th International Conference on Field Programmable Logic and Applications (FPL2009), Prag, Czech Republic, Aug. 31-Sept. 2, 2009.