DISSERTATION

submitted to the

Combined Faculty of Natural Sciences and Mathematics

of the

Ruprecht–Karls University
Heidelberg

for the degree of

Doctor of Natural Sciences

put forward by

M.Sc. Juri Schmidt

born in
Alexejewka, Kazakhstan

Mannheim, 2017

Accelerating Checkpoint/Restart Application Performance in Large-Scale Systems with Network Attached Memory

Advisor: Professor Dr. Ulrich Brüning

Date of the oral examination: .....

To my beloved family

my parents

Vera and Georg Schmidt

and my sister

Ludmila Schmidt

Abstract

Technology scaling and a continual increase in operating frequency have been the main driver of processor performance for several decades. A recent slowdown in this evolution is compensated by multi-core architectures, which challenge application developers and also increase the disparity between the processor and memory performance. The increasing core count and growing scale of computing systems furthermore turn attention to communication as a significant contributor on application run-times.

Larger systems also comprise many more components which are subject to failures. In order to mitigate the effects of these failures, fault tolerance techniques such as Checkpoint/Restart are used. These techniques often rely on message-based communication and data transport stresses the local memory interface. In order to reduce communication overhead it is desirable to either decrease the number of messages, or otherwise to accelerate the execution of commonly used global operations. Finally, power consumption of large-scale systems has become a major concern and the efficiency of such systems must considerably improve to allow future Exascale systems to operate within a reasonable power budget.

This work addresses the topics memory interface, communication, fault tolerance, and energy efficiency in large-scale systems. It presents Network Attached Memory (NAM), an FPGA-based hardware prototype that can be directly connected to a common high-performance interconnection network in large-scale systems. It provides access to the emerging memory technology Hybrid Memory Cube (HMC) as shared memory resource, tightly integrated with processing elements.

The first part introduces the HMC memory architecture and serial interface, and thoroughly evaluates it in an FPGA using a custom-developed host controller, which has become an open-source initiative.

The next part describes the hardware architecture of the NAM design and prototype, and theoretically evaluates the expected performance and bottlenecks. The NAM design

was fully prototyped in an FPGA and the contribution also comprises a corresponding software stack.

As a first use case NAM serves as Checkpoint/Restart target, aiming to reduce inter-node communication and to accelerate the creation of checkpoint parity information. Reducing checkpointing overhead improves application run-times and energy efficiency likewise.

The final part of this work evaluates the NAM performance in a 16 node test system. It shows a good read/write scaling behavior for an increasing number of nodes. For Checkpoint/Restart with a real application, a 2.1X improvement over a standard approach is a remarkable result. It proves the successful concept of a dedicated hardware component to reduce communication and fault tolerance overhead for current and future large-scale systems.

Zusammenfassung

Der kontinuierliche Anstieg der Mikroprozessorleistung wurde über Jahrzehnte hinweg getrieben von immer feiner werdenden Halbleiterstrukturen sowie steigenden Taktraten. Die kürzlich beobachtete Verlangsamung dieser Entwicklung wird durch Multi-core Architekturen kompensiert. Diese erfordern parallelisierte Anwendungen und stellen Anwendungsentwickler und die Prozessor-Hauptspeicher Schnittstelle gleichermaßen vor große Herausforderungen. Der weiterhin fortwährende Trend zu immer größeren verteilten Systemen und die damit einhergehende Zunahme an Einzelkomponenten stellt insbesondere Anforderungen an das Verbindungsnetzwerk, sodass viele Anwendungen bereits heute viel Zeit mit reiner Kommunikation verbringen.

Größere Systeme erhöhen zugleich die Wahrscheinlichkeit für Defekte. Um deren negative Auswirkungen zu reduzieren und Defekte zu tolerieren, werden üblicherweise Checkpoint/Restart Mechanismen eingesetzt. Da diese zumeist auf Kommunikation zwischen einzelnen Knoten basieren und zusätzlich die Prozessor-Hauptspeicher Schnittstelle belasten, ist es sinnvoll entweder den Umfang der benötigten Kommunikation zu reduzieren oder deren Einfluss zu minimieren. Zu guter Letzt gewinnt auch die Leistungsaufnahme verteilter Systeme immer mehr an Bedeutung. Im Hinblick auf die Exascale-Ära ist es daher zwingend notwendig die Energieeffizienz bedeutend zu verbessern um den Leistungsverbrauch dieser Systeme in einem vertretbaren Rahmen zu halten.

Diese Arbeit geht auf die oben genannten Problematiken Speicherschnittstelle, Kommunikation, Fehlertoleranz und Energieeffizienz ein und stellt Network Attached Memory (NAM) vor. NAM ist ein Hardware Prototyp, der direkt an ein gängiges Hochleistungs-Verbindungsnetzwerk in verteilten Systemen angebunden werden kann. Es bietet Zugriff auf gemeinsamen Speicher, der durch die aufstrebende Hybrid Memory Cube (HMC) Technologie realisiert ist.

Zusammenfassung

Der erste Beitrag umfasst die Vorstellung, Technologieanalyse und HMC Evaluation in einem FPGA mithilfe einer eigens entwickelten Zugriffseinheit, die als Open-Source Initiative frei zugänglich ist.

Der nächste Beitrag erläutert den Entwicklungsprozess und die Hardwarearchitektur des NAM Designs und Prototypen und ermittelt die Leistung theoretisch. Das NAM Design wurde hierfür vollständig in einem FPGA implementiert und durch die für den Zugriff notwendigen Softwarekomponenten ergänzt.

In einem ersten Anwendungsfall dient der NAM als Beschleuniger für Checkpoint/Restart Prozesse mit dem Ziel, Kommunikation zwischen Knoten zu verringern und die benötigte Paritätsinformation schneller zu berechnen. Dies wird sich Vorteilhaft auf Anwendungslaufzeiten und Energieeffizienz auswirken.

Der letzte Beitrag beinhaltet verschiedene Leistungsmessungen in einem realen 16 Knoten System. Diese zeigen optimale Skalierbarkeit für Lese- und Schreibzugriff. Für Checkpoint/Restart wird eine bemerkenswerte, 2.1-fache Beschleunigung erreicht. Dieses Resultat belegt das erfolgreiche NAM Konzept zur Reduktion von Kommunikation und des Berechnungsaufwands für Fehlertoleranz in aktuellen und zukünftigen Systemen.

Contents

CHAPTER 1

Introduction

For many years, the increase in processor and system performance was driven by technology scaling which allowed to pack more transistors per area at a fixed power budget. Increasing operating frequencies supported the improvements of single thread performance accordingly. Since the early 2000's, however, this continual increase has slowed down due to excessive power dissipation, which is also caused by the leakage current of today's tiny transistor feature sizes. Multi-core architectures were developed to keep up with the traditional growth rate, and system performance was scaled by adding more and more components and nodes. Although these new architectures pose challenges to application developers as it requires carefully parallelized codes, the overall system performance kept increasing at a moderate rate. This is documented in Figure 1.1 which shows the evolution of the number 1 systems of the TOP500 list of supercomputers.

At a first glance the current trend gives no indication that there is something wrong at all, especially not with the memory system. This is because the TOP500 LINPACK benchmark in large parts is insensitive to memory performance and re-uses data that remains in registers and caches instead [1]. In reality, however, memory access times and bandwidth lag behind the historical evolution of CPU performance. This disparity is well known as the memory wall [2] and the gap is widening with the recent and ongoing increasing number of CPU cores per socket that operate on the same memory interface.

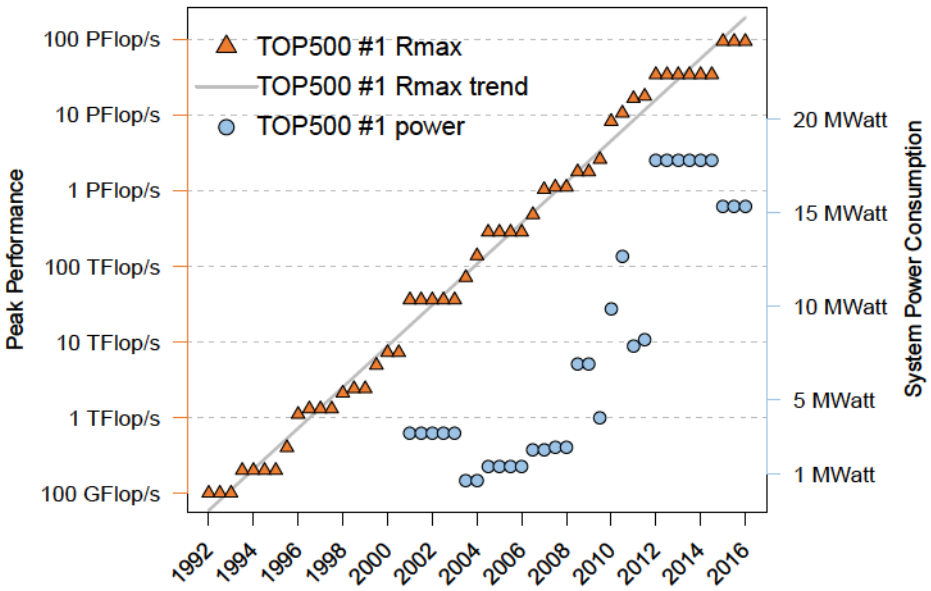


Fig. 1.1 TOP500 number 1 system performance and power development[1]

Despite the lag in memory evolution, large-scale systems continually achieved extraordinary performance gains. Fundamental for this development has been a steady increase in component and node count, with the current number 1 TOP500 system comprising more than 10 million cores[1]. Communication between two or more cores is typically realized via message passing and unless an application is perfectly parallel, inter-process communication will take place. Hence, whenever a message is sent or received across node boundaries it also involves the interconnection network. Work in [3] shows that current large-scale applications spend an average of 36% of their runtime with point-to-point communication and waiting for collective operations to complete. It will be shown that the interconnection networks for their part substantially improved over the last decade and other approaches to mitigate the overhead through inter-process communication must be developed.

One additional and underestimated drawback of system scaling is its impact on the system's error rates, or Mean Time Between Failure (MTBF). Hardware faults are among the most common causes for system crashes and every single additional component potentially decreases the average time between two failures. For an Exascale machine it is predicted that it will comprise about 260 thousand nodes (134 million cores) [4], that is about 6 times more nodes (12 times more cores) than in the current

[1] Data collected via the TOP500 statistics sublist generator (www.top500.org/statistics/sublist).

number 1 TOP500 system. Unless the per-component MTBF will significantly drop, systems will continue to fail even more frequently in the future. In order to properly recover from failures, resilience and fault tolerance mechanisms play an important part in modern large-scale systems. These features are typically implemented by periodically storing the application's or system state to disk. The checkpoint may then be restored upon a system failure in order to reduce the amount of work lost. The obvious disadvantage is that checkpoints are created whether or not there is an actual failure. This process requires application time as well as memory and network bandwidth, and can cause applications to execute more than 10 times slower [5].

Finally, power has become a main concern of today's and future large-scale systems. Main memory in particular is one of the largest consumers with up to 40 % for a current system [6] and a projected 65 % at Exascale [7]. Inter-process communication and fault tolerance mechanisms additionally reduce the actual work that can be done within a given time period, which negatively impacts the system's energy efficiency. To allow future Exascale systems to operate within an economically and practically reasonable power budget it was suggested to limit the power consumption of such a system to 20 MWatt [7, 8]. As can be seen in Figure 1.1, the current number 1 system already consumes more than 15 MWatt at less than 100 PFLOP/s peak performance. The mandatory need for a change of the system architecture becomes clear when this system was scaled to Exascale. At 1 Exaflop per second it would consume about 150 MWatt which exceeds the 20 MWatt goal by a factor of 7.5.

In summary, the challenges on the road to Exascale machines are best described by the following quote:

> *The architectural challenges for reaching Exascale are dominated by power, memory, interconnection networks, and resilience.*
>
> — Richard C. Murphy et. al. (2010) [9]

1.1 Motivation

The motivation for this work is based on the following three key observations:

The Processor-Memory Gap

The memory interface is one of the last parallel buses and probably the most

critical bottleneck in modern computing systems. The disparity between processor and memory performance is ever increasing and the situation got worse with the introduction of multi-core architectures. Since no technological breakthroughs are expected in the near future, it is time to revisit the memory interface and evaluate alternatives.

Inter-Node Communication

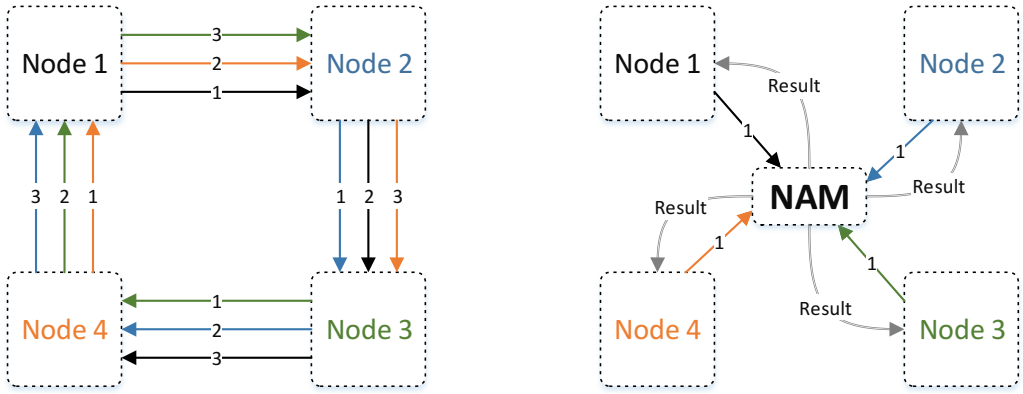
Large-scale systems typically communicate via message passing and data must be transported between two or more nodes whenever the communicating processes are spread across distinct nodes. While some applications mainly rely on point-to-point communication, others spend a lot of time in processing and waiting for the completion of collective operations. Most often the memory interface is involved in collective operations as it holds the data elements that are placed and retrieved by processors and the interconnection network. It is desirable to either reduce the number of messages that are sent or otherwise to increase the performance of point-to-point and collective communication.

Fault Tolerance

With an increasing number of components in large-scale systems, and without a significant improvement in component reliability, the MTBF will continue to drop and the frequency of catastrophic failures will increase. To mitigate the effects of such failures, to reduce the amount of work lost, and to allow rapid system recovery, today's systems deploy fault tolerance techniques using Checkpoint/Restart. Unfortunately, checkpointing introduces additional overhead and can take up a large amount of the application runtimes. Parity checkpoints were introduced to lower the overhead at the expense of computation and communication, which heavily utilizes processors and the memory and storage system. It is therefore necessary to investigate in innovative approaches to reduce the overhead in order to speed-up the parity creation process.

1.2 Vision

As the types of hardware and interfaces in computing systems are standardized and systems are built from commercial off-the-shelf components, the only way to achieve improvements in the areas mentioned above is a dedicated hardware component. The ideal candidate avoids slow memories, offloads processors from computing collective



(a) Example traditional scheme: A collective operation is implemented as a ring

(b) Envisioned scheme: NAM as central instance to execute collective operations

Fig. 1.2 NAM Vision: Reduce communication and offload processor computation

operations, reduces communication and synchronization effort, and provides sufficient bandwidth to serve as target for as many processes and nodes as possible. Figure 1.2 envisions how such a component, integrated with an existing interconnection fabric, is meant to improve collective operations by reducing inter-node communication and associated memory accesses. This general approach can be transferred to Checkpoint/Restart which in large parts relies on these patterns.

1.3 Contributions

This work presents the implementation of the Network Attached Memory, a dedicated component to serve as a global shared storage and to carry out collective operations in large-scale systems. It therefore employs network interfaces that provide the ability to connect it to available links within the EXTOLL high-performance interconnection network.

Based on the motivation to replace the current parallel memory interfaces with a flexible and serial one, the NAM prototype implements the emerging Hybrid Memory Cube memory interface. The HMC performance and power efficiency is analyzed and evaluated in an FPGA (Field Programmable Gate Array) using a custom host controller. This contribution comprises conference publications [10, 11] and the well adopted open-source initiative openHMC [12].

The developed FPGA design implements links to the EXTOLL network and the HMC. It provides modules for read and write operations from remote hosts from and to the HMC memory on the NAM. A checkpointing module improves Checkpoint/Restart mechanisms that are typically deployed in today's systems. It aims to reduce the communication and synchronization overhead between participating nodes and shall offload processors from calculating the corresponding parity information. This contribution has led to news articles [13, 14] and a conference poster [15].

Finally, the performance of the NAM for reading and writing, and for the Checkpoint/Restart (CR) use case is evaluated in a 16 node test system. The results show that CR with the NAM outperforms a current approach by a factor of 2.1.

1.4 Outline

The remainder of this work is organized as follows: The next chapter covers the three relevant topics memory, inter-node communication, and fault tolerance. The discussion supports the need to revisit the memory interface and indicates that a dedicated hardware may be able to mitigate the excessive overhead in communication and fault tolerance. Chapter 3 presents the Hybrid Memory Cube (HMC) interface and technology in detail. Using a self-developed host controller, the HMC performance is characterized with real system measurements. Chapter 4 describes the development of the Network Attached Memory (NAM) hardware prototype. It provides network interfaces and integrates an HMC. The implemented FPGA design units are presented and the theoretical NAM performance is evaluated. As a first use case, the NAM improves the creation of parity checkpoints in the DEEP-ER (Dynamical Exascale Entry Platform - Extended Reach) project. The chapter is concluded by a description of the Checkpoint/Restart process and the developed software components. Chapter 5 evaluates the NAM in a 16 node real system setup with microbenchmarks and a DEEP-ER application mockup. The last chapter summarizes and reflects the obtained results and suggests improvements for a future NAM implementation.

CHAPTER 2

State of the Art

Today's large-scale systems suffer from various limitations often caused by only very few components. As this trend is expected to intensify in the future, and in order to develop potential solutions, it is necessary to understand the reasons behind these limitations.

The first section of this chapter describes the historical evolution and current trends in the main memory development, and motivates the adoption of serial interfaces as one solution for most of the issues presented. Next, based on the prevalent software and hardware components, the communication in High Performance Computing (HPC) systems is analyzed. The third section presents currently deployed fault tolerance techniques which will gain even more importance with increasing system sizes. A final summary that puts these three topics into context concludes this chapter.

2.1 Memory: Technologies and Interfaces

For many years the increase in CPU (Central Processing Unit) performance was driven by Moore's law which was initially formulated in 1965 [16]. It predicts that the transistor count in microprocessors will double every 18 to 24 months, and this prediction remained true for about four decades. Although recently a slowdown can be

observed, device manufacturers found ways to keep increasing the transistor count at a moderate rate.

One processor characteristic that has stopped scaling, however, is the internal operating frequency. This is due to the reason that transistor power consumption is proportional to frequency and the power density increases as more transistors are packed per area. Also, with smaller transistors, leakage current becomes significant which causes the processor to dissipate power and heat at an increasing rate. This has led to the end of the well-known *Dennard scaling* [17] which states that power consumption remains proportional to the chip area.

To keep up with the traditional performance growth rate of CPUs, multi-core architectures were developed and current devices integrate as many as 72 cores on a single die [18].

One component that historically lags behind processor performance is the Dynamic Random-Access Memory (DRAM)-based main memory. Although it was formulated more than 20 years ago, the current situation is very well summarized by the following quote [19]:

> *Across the industry, today's chips are largely able to execute code faster than we can feed them with instructions and data. There are no longer performance bottlenecks in the floating point multiplier or in having only a single integer unit. The real design action is in memory subsystems—caches, buses, bandwidth, and latency.*
>
> — Richard Sites: It's the Memory, Stupid! (1996)

Similar to CPUs, DRAM obeyed Moore's law for a long time and only recently a slowdown in capacity growth is observed. Much more critical than the capacity, however, is the access time for a memory reference to the off-chip main memory. While the relative single core CPU performance increased by a factor of 10.000 in 30 years, the vast increase in peak memory accesses outperforms the capabilities of the memory interface. More specific, the DRAM access latency relative to the number of CPU cycles it takes to serve a memory reference only improved by a factor of eight. Within the same time period access times decreased from 250 ns in 1980 to 31 ns in 2012 [20]. Figure 2.1 illustrates this disparity in performance which is well-known as the *memory wall* [2]. Although the historical development and current trend for the performance of the number 1 system in TOP500 list of supercomputers gives only small indication that memory performance may be critical at all, it is and will remain a serious matter.

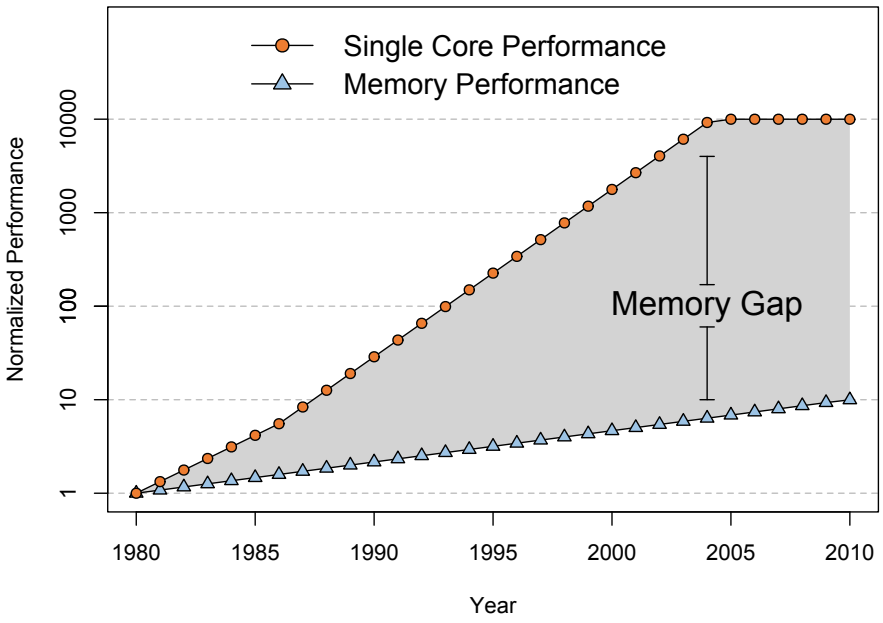


Fig. 2.1 Historical trend of the processor-memory gap [20]

The reason why in particular the TOP500 list is not suitable to discover a memory bottleneck is because of the benchmark it uses to characterize systems. The LINPACK benchmark is very insensitive to memory performance [1] which certainly does not accurately reflect the majority of HPC applications. It was also shown that future application codes will be much more memory sensitive [21].

2.1.1 Latency

The recognition of the memory wall led to the introduction of caches and hierarchies of caches in many variants and with various levels to hide the latency from a processor view. Caches rely on the concept of temporal (a memory reference is likely to be used more than once) and spatial locality (multiple accessed memory references are within relatively close storage locations). Hence, caching tries to avoid accessing the relatively slow physical memory interface by holding data in processor-local structures. Eventually, data still needs to be transported over the memory interface with potentially many independent processor cores competing for access. This increases the probability of cache misses and leads to additional, latency and bandwidth-wise expensive main memory accesses. And even if there was only one process to access the memory it

might be limited by the interface latency if an application is not able to exploit locality for its memory references.

Although smaller and faster transistors typically help latency, the increasing number of transistors per area and additional memory chips to maximize capacity also result in longer channel distances. These longer traces and higher fan-outs naturally increase the signal latency and limit the switching speed on these lines. Also, the focus of the DRAM semiconductor process has shifted away from maximizing performance to increasing the capacity and reducing the memory cell's leakage current which is critical with today's small feature sizes. As the author in [22] states, the terms bandwidth and capacity are much easier to sell than latency marketing wise, and yet another reason why latency has been missing significant improvements.

2.1.2 Bandwidth

The situation with memory bandwidth is less critical than with latency although Double Data Rate (DDR) as the most commonly used main memory interface also lags behind the requirements of modern processors. For example, an Intel Core i7 CPU with four cores can generate memory references that require a peak bandwidth of 409.6 GB/s [20]. The actual requirement can be even higher as peripheral devices may also request access to main memory via Direct Memory Access (DMA). In contrast, a current DDR4 module provides 25 - 30 GB/s [20].

The main reason for this disparity is that the main memory interface has not seen meaningful changes in more than 30 years. Although each new generation of DRAM modules came with a slightly modified layout (which also required new processor generations), it is one of the last parallel interfaces in modern computing systems. Performance gains were achieved by widening the interface, increasing the pin speeds, and the introduction of DDR signaling and prefetch mechanisms. Adding more memory modules for multi-channel operation remains another viable option but its scaling behavior is limited by the excessive use of processor I/O pins, Printed Circuit Board (PCB) routing issues and a lack of physical space to place the additional components on a board.

Other approaches examined the feasibility of memory latency reduction (which correlates with a bandwidth increase) by either improving the memory access scheduling for multiple cores [23] or asynchronously reorganizing the DRAM banks within a memory chip [24]. And finally, to address the rapidly growing mobile and graphics card

market, derivatives of the DDR interface were developed. These are tailored to the varying requirements of the power oriented mobile market (Low Power Double Data Rate (LPDDR) [25]) and bandwidth-hungry graphics cards (Graphics Double Data Rate (GDDR) [26]). All of these variants, however, are still limited by the memory interface bottleneck which needs to be revisited.

2.1.3 Power

Aside from the memory performance, power consumption of large-scale systems increasingly moves into focus and the memory system plays an important role in this observation. An analysis of a high-end IBM server in [6] showed that memory consumes as much as 40 % of the overall system power and this trend is also observed with current graphics cards [27].

To keep the power requirements of future Exascale systems within a reasonable budget it was decided that such systems should not exceed a total of 20 MWatt [7, 8]. Projections in [7] show that this goal is ambitious and challenging. The authors scale a current large-scale system to Exascale size and predict the power consumption considering technology improvements that enhance efficiency. The outcome of this experiment is that such a system would consume 70 Megawatt. More interestingly, the memory is the largest consumer with over 65 % of the total consumption.

Clearly, memory and in particular its interface will remain one of the most important targets for optimization for current and future systems. To bridge the gap until a new technology with the potential to replace DRAM as main memory hits the market, memory manufacturers recently started proposing alternative interfaces. Additionally, advances in the semiconductor manufacturing process have made layer stacking and heterogeneous stacks a viable option.

2.1.4 Emerging Memory Technologies

The increasing demand for memory performance and capacity, and the I/O and area scalability issues of DRAM DIMMs (Dual In-line Memory Modules) has led to vertically stacked architectures, leveraging recent developments in fabrication process. Multiple layers of DRAM can now be stacked on top of each other, linked via tiny connections called Through Silicon Vias (TSVs) [28]. The ability to pack more memory arrays

per area increases the capacity and results in shorter traces, hence reducing fan-out, latency, and power consumption for signals on the memory interface channels.

Examples for stacked memories are HBM (High Bandwidth Memory) developed by AMD and SK Hynix [29] (high-performance) and the WideIO standard [30] (low-power mobile segment). Both memory types are DRAM based and still rely on a parallel interface. Processor and memory components are typically placed and interconnected on a common silicon interposer which is packaged in 2.5D technology. This brings the two components closer to each other, thus further decreasing trace lengths and routing effort. HBM for instance is already deployed in AMD graphics cards [31] and Altera FPGAs [32].

Although these new technologies only recently entered the market and the cost is relatively high, it is expected that they will continue to gain considerable market shares as both significantly improve the memory performance and power characteristics within their market segments. It must be noted that stacking is also utilized for non-volatile storage class devices such as V-NAND from Samsung [33] as well as 3D-NAND [34] and the recently announced 3D XPoint [35] from Micron and Intel.

The second class of revolutionary packaging options is 3D integration. It benefits from the additional advantage that TSVs enable different processes such as DRAM and CMOS (Complementary Metal Oxide Semiconductor) to coexist within a single stack. 3D integration allows to shift the complexity of a memory controller into a logic layer at the bottom of a memory stack. Popular examples are Intel's Multi Channel DRAM (MCDRAM) and Microns HMC. Intel's latest KNL (Intel Knights Landing) CPUs connects multiple MCDRAM[1] devices via a proprietary interface in a 2.5D package [18]. An MCDRAM is a stack of multiple DRAM layers on top of a logic base that integrates the actual memory controller functions. Similarly, Micron's HMC [37] stacks up to 4 layers of DRAM on top of a logic base that fully integrates up to 16 independent memory controller. The innovative part with HMC is that the traditional parallel interface to the processor is replaced by high-speed serial links. The benefits of such an interface are described next.

[1] According to [36], MCDRAM is based on HMC with a modified logic base and interface.

2.1.5 Serial Interfaces

Utilizing serial high-speed links to connect a memory breaks with the traditional parallel interface, and there are several good reasons to examine this approach: A serial interface

1. shifts the memory controller complexity into the memory stack. It decouples the development of the memory interface from of the actual DRAM array and other memory technologies.
2. likely operates on packets instead of transactions. This enables the existence of potentially many outstanding requests which suits the demands of current multi-core/multi-threaded CPUs with many independent request streams.
3. enables the use of application-specific packets and commands to integrate processing capabilities close to the memory (see Section 2.1.6).
4. reduces I/O pin requirements and routing complexity. The interface itself consists of several high-speed differential lanes and a few sideband signals.

The author in [38] formulated a motivation to adapt serial interfaces and in particular highlighted the benefits of Micron's HMC as a candidate. This motivation is extended by a detailed description and evaluation of the HMC in Chapter 3.

To the best knowledge of the author, at the time of writing the only other device that stacks memory on top of a logic base with a serial interface is the SRAM (Static Random-Access Memory)-based Bandwidth Engine (BE) by Mosys [39]. Although BE provides the lowest memory access latency on the market ($\approx$ 16 ns for a full memory reference) its maximum capacity is currently 1 Gbit which makes it unusable for most applications and impractical as main memory replacement. The HBM specification similarly defines an optional interface die (e.g. with a serial interface). At the time of writing, however, there are no devices with such an interface available.

2.1.6 Processing in Memory

This section so far has only considered changing or improving the existing parallel memory interface. Another approach that has recently become a well discussed topic is to avoid data movement where applicable by shifting the actual processing into the

memory array, or at least as close to it as possible. Among other acronyms that have emerged, the most popular is Processing In Memory (PIM).

PIM describes the tight integration of CMOS logic and memory cells within a single chip. This idea is not novel and several architectures that place combinational circuits right next to the memory were proposed already in the 1990's (e.g. [40, 41]). Their functionality, however, was limited to very basic operations and only the recent advancements in fabrication process have made PIM an interesting topic for researchers. HMC can be categorized as PIM device as it supports atomic functions that can autonomously add values to memory locations. The Active Memory Cube (AMC) [42] takes this capability to a next level. Based on the HMC memory architecture it integrates a full Instruction Set Architecture with caches and pipelines. Although AMC is still a research project and the performance projections are based on simulations, it gives a glimpse into the capabilities of PIM and how it can be used to reduce the memory interface traffic.

2.1.7 Summary Memory

This section highlighted the reasons for the existing and ever increasing gap between the processor and memory performance. It became clear that the main memory interface must change in order to keep up with the increasing number of cores and components that access it. Although recent developments led to performance and capacity improvements they have not significantly changed the way how memory is accessed. A technological breakthrough that is able to replace the DRAM cell is currently not foreseeable, but at the same time seems inevitable to overcome the proposed power budget of next-generation Exascale systems. To speed-up the adoption of future memory technologies, the serial memory interface was introduced. It comes with plenty benefits that have the potential to change the memory landscape. This includes the possibility to rapidly develop and integrate complex processing units within the memory stack without the need to change the interface itself.

2.2 Communication in HPC systems

Today's HPC systems often comprise multiple thousand nodes and projections show that this number will scale up to 260.000 nodes for Exascale machines [4]. Without a significant change in how these systems are designed, future systems will more or

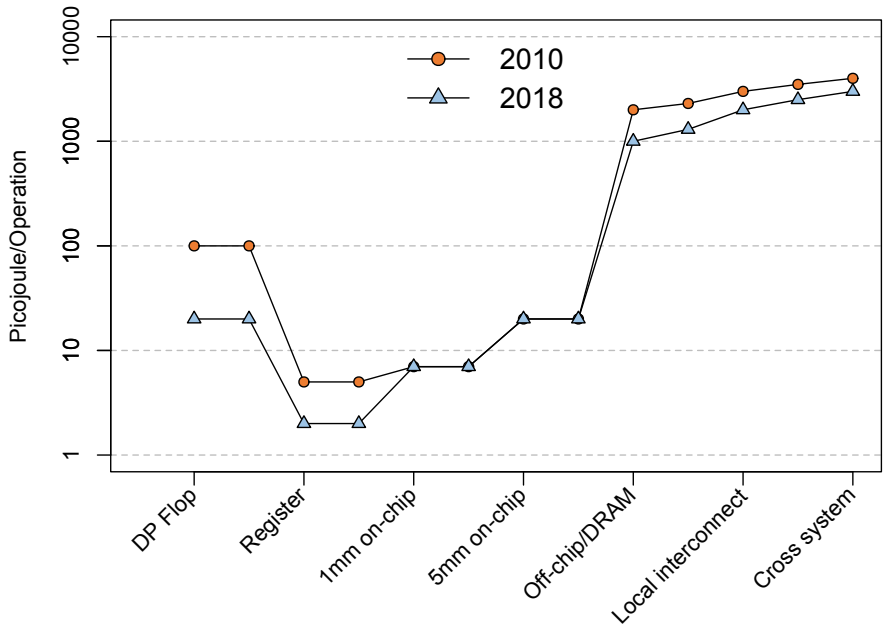


Fig. 2.2 Energy cost for data movement across different layers [7]

less rely on the current prevalent communication scheme: An application uses MPI (Message Passing Interface) to exchange messages between two or more processes. These messages are physically transported via an underlying hardware, the interconnection network (*interconnect*). This section introduces several types of commonly deployed interconnects and summarizes the communication schemes and typical patterns of current large-scale systems.

2.2.1 Interconnection Networks

To leverage the vast amount of processing capabilities of thousands of nodes, the jobs that run on these systems need to be partitioned and parallelized as good as possible. Unless a job (or *the problem*) is perfectly parallelized, and this often does not only depend on the programmer but the on problem itself, inter-process communication will take place. Sometimes, this communication occurs between two processes running on the same node or even the same processor. Most often, however, inter-node communication is inevitable which is expensive in terms of energy and latency. Figure 2.2 illustrates the energy cost to move data through the different possible types of the interconnect hierarchy.

It can be seen that the required energy to transport an information becomes significant when moving off-chip (e.g. to get data from local DRAM), and increases further when using the local (processor-)interconnect or crossing node boundaries. Even worse, whenever data is transported to a remote node this node is likely to be waiting for it. This additional latency results in stall states where no useful computation is performed. Hence, the interconnection network plays a vital role for the overall system performance and energy efficiency. Assuming a perfectly parallelized application the interconnect is often the most important component and a popular optimization target to keep communication overhead at a minimum.

Out of several interconnect technologies, Ethernet and Infiniband [43, 44] have emerged as the most prevalent solutions in HPC. According to the TOP500 list of supercomputers[2] (June 2017 edition), 208 out of 500 systems run Ethernet[3] (41 %) and 177 use Infiniband[4] (35 %). The remaining systems use proprietary interconnects such as Intel Omnipath [46] which gradually gains traction in the list since its introduction in 2015. It must be noted that currently no machine within the 10 most powerful supercomputers of the TOP500 list uses Ethernet or Infiniband. The leading spots are held by non-standard, vendor specific interconnects tailored to these machines and the LINPACK benchmark.

Most of the interconnection networks, including Ethernet and Infiniband, have in common that they come as a PCI Express (PCIe) plug-in card for the corresponding slots in today's commodity hardware. These Network Interface Controllers (NICs) provide host connectivity via PCIe and one link to the network fabric. One exception to this is Intel Omnipath which integrates the NIC with the CPU and therefore removes the often criticized PCIe connection as bottleneck.

A message that targets a remote node is first processed by the NIC and then sent to the network. Switches and hierarchies of such are used to link all nodes together. Unfortunately switches come at a price and limit the scalability as the network becomes non-uniform. Also, physical space must be preserved to place these switches in a rack.

One approach to avoid these drawbacks is the emerging interconnect EXTOLL [47, 48, 49]. Just as with the other interconnects, the EXTOLL NIC plugs into a standard PCIe slot, but it already integrates the switching functionality. Each NIC provides six network links and therefore allows to directly create topologies such as a 3D torus,

[2] The TOP500 list of supercomputers, established in 1993, is a half-yearly updated list that ranks the 500 most powerful supercomputers using the LINPACK benchmark [45].
[3] Ethernet, 1G Ethernet, 10G Ethernet (majority), or 100G Ethernet.
[4] Infiniband QDR, FDR (majority), or EDR.

Table 2.1 Interconnect performance comparison

Interconnect	Latency [us]	Bandwidth [GB/s]
Ethernet 1G [50]	47	0.112
Ethernet 10G [50]	12	0.875
Infiniband QDR [50]	1.6	3.23
Infiniband EDR [51]	0.6	12.5
EXTOLL [52]	0.6-0.8	12.5

maintaining scalability at all times. These are two of the main reasons why EXTOLL has been selected to evaluate the NAM. Section 4.2 will present the technology in detail.

Performance-wise, all of the interconnects mentioned above have significantly improved over the past decade. As Table 2.1 points out, Infiniband EDR and EXTOLL are superior to 1G and 10G Ethernet in bandwidth and latency. The reason why in particular 10G Ethernet is still deployed that often (195 systems in the TOP500) is because of its relatively low cost. State of the art HPC systems require thousands of NICs and hundreds of switches to fully interconnect all nodes.

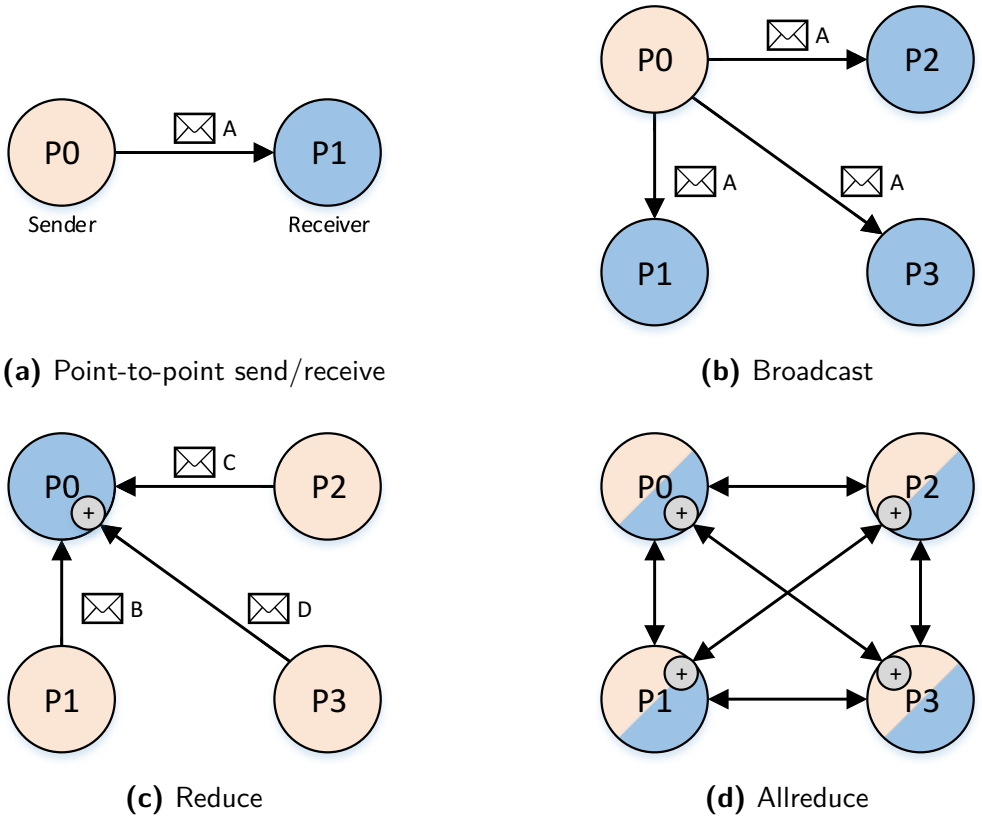
Whichever interconnect is used it still remains a tool for applications to facilitate inter-process communication and the actual utilization of the interconnect depends on the communication characteristics of the application itself. It is in particular important to understand these characteristics in order to optimize the overall system performance as simply improving the interconnect bandwidth and latency might not pay off significantly in all cases.

2.2.2 Message Passing and Communication Characteristics

Message passing is the prevalent inter-process communication scheme in today's large-scale systems and has become the de-facto standard. It abstracts the underlying data movements to a simple concept of messages that are sent and received between two processes.

The most widely used message passing standard is MPI[5] (first introduced in [54]) which itself is not a library, but rather a specification that defines how message passing

[5] The full specification of the current MPI standard version 3.0 can be found in [53].



(a) Point-to-point send/receive

(b) Broadcast

(c) Reduce

(d) Allreduce

Fig. 2.3 Example MPI operations (Legend: Orange - sender. Blue - receiver. '+': Logical operation)

libraries should operate. Out of this standard, several implementations including MVAPICH/MVAPICH2 [55] and the very popular open-source variant Open MPI [56] have evolved.

Figure 2.3 depicts four MPI operation examples. These include, but are not limited to asynchronous point-to-point messaging and collective operations such as Broadcast (distribute data from one process to other processes), Reduce (move data from other processes to one process and perform a logical operation; one process receives the result), and Allreduce (apply logical operation on data from all processes; all processes receive the result). In particular, the actual implementation of Allreduce and other similar collective operations depend on the MPI library that is used. Data exchange for such functions can be realized with all-to-all communication (Figure 2.3d) or various other logical topologies such as a binary tree or a ring.

Apart from the actual scheme that is used for collective operations, an application typically comes with predictable and well-known access patterns. It is essential to

characterize applications by means of their communication behavior to determine useful system optimizations for a specific use case. Naive approaches to simply improve the performance of components such as the processors or the interconnect will not necessarily lead to substantial speed-ups if the actual bottleneck is somewhere else. For example, some applications heavily utilize point-to-point communication while others spend most of their time performing collective operations. This depends on the application itself and how it is implemented.

Recent work in [3] analyzed the MPI characteristics of application traces collected by the U.S. Department of Energy (DOE) [57]. The analyzed dataset comprises 18 different applications with 10 up to 13.000 ranks. The key finding of this work is that these applications spend 36 % on average of their time in MPI routines with a peak of up to 60 %. Interestingly, while the vast bulk of data is transported via point-to-point communication the average application spends most of its MPI time with collective operations. Although it was shown that only small amounts of data are processed with collective operations, synchronization overhead for a large number of processes becomes significant. This insight is in particular important as it highlights that for many applications the focus must be shifted to improving collective operations instead of just focusing on increasing bandwidths.

This issue has already been identified and the latest MPI-3.0 standard foresees non-blocking collective operations which allow to continue program execution while collective operations take place. However, it also brings up the question of architectural changes and encourages the use of dedicated resources to carry out these operations and to reduce synchronization overhead.

2.2.3 Summary Communication

Current large-scale systems and applications rely on a message-based communication schemes which send and receive messages via a physical interconnection network and it is not expected that this general approach will change in the near future.

Although the various types of interconnects as important part for inter-node communication showed substantial progress over the past decade, simply improving these components may not be sufficient. This is in particular the case for applications that spend a large amount of their execution time waiting for completion of collective operations. The resulting processor stalling negatively impacts the application's performance and energy efficiency.

Unarguably the interconnect will remain an important system component and needs to be further optimized. For many applications, however, there is an obvious need to rethink system design and a dedicated resource to mitigate existing application bottlenecks appears reasonable and tempting.

2.3 Fault Tolerance in HPC Systems

Resilience has become a major concern for HPC systems. As systems continue to grow in size, more and more components are added. Unfortunately, each additional component is also subject to faults (e.g. a stuck bit) which are likely to result in errors such as an incorrect value and false program execution. Errors on the other hand may lead to incorrect system states or an application crash known as failure[6]. Previous work in [59] showed that the number of failures per system is almost proportional to its number of processors, which correlates with the amount of memory and other components. Without an increase in component reliability the MTBF of future systems will further decrease as it is expected that Exascale systems will comprise more than 260.000 nodes [4], 6 times more than the current number 1 ranked HPC system Sunway TaihuLight[7]. The equation is easy: with 6 times more components at a given component reliability, a system will fail 6 times more frequently.

In fact, the single component reliability even decreases with technology scaling and design for power efficiency. Smaller transistors typically carry smaller charges and also suffer from manufacturing variances, making them more error prone. The DRAM soft error rate for example has been analyzed in two studies conducted in 2004 [60] and 2009 [61] using respective state of the art memories. A comparison of the results unveils a 25X increase in DRAM soft failure probability in only 5 years. Although ECC (Error Correction Code) technology is able to correct a bulk of such errors their occurrence will further increase.

Along with the obvious challenges caused by technology scaling, semiconductor devices also become less reliable over their lifetime. This is known as aging and it gets worse with smaller features sizes. Interestingly, the authors in [62] found a correlation between the number of component failures and the day of the week and the hour of the day. Hence, components are likely to fail more often under heavy workload.

[6] For more information on the taxonomy see [58].

[7] The Top500 list twice-yearly ranks the performance of the 500 fastest supercomputers in the world. See www.top500.org for more information.

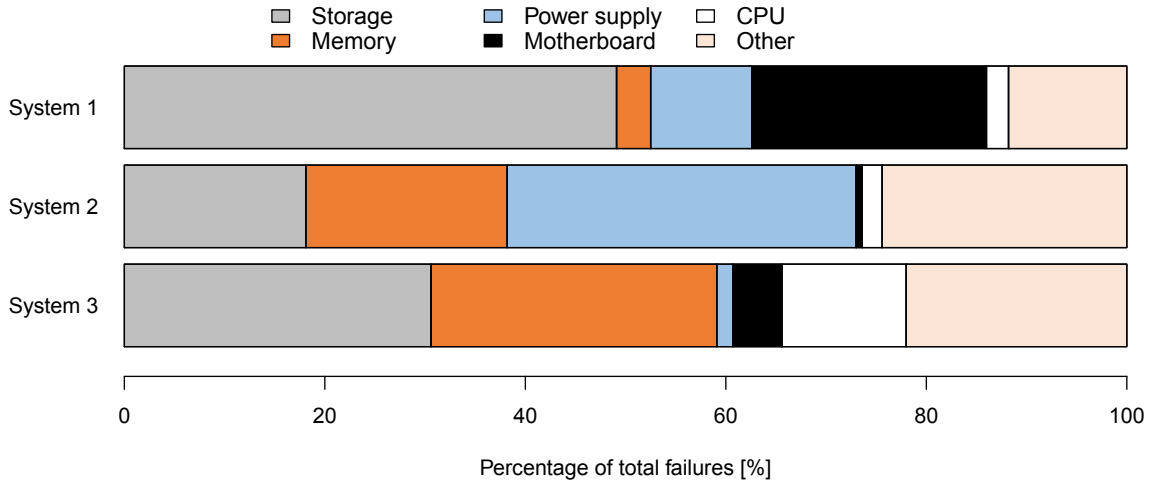


Fig. 2.4 Hardware failure breakdown by component for three different and unrelated systems [65]

Cause	Hardware	Software	Network	Human	Facilities	Unknown
Percentage	60.4 %	22.6 %	1.8 %	0.6 %	1.5 %	13.1 %

Table 2.2 Causes of failures by type collected by LANL from 1996 to 2007 [66]

For today's large-scale systems, the MTBF ranges from a few hours to several days, mainly depending on the system size [63]. Researchers predicted that an Exascale system might fail in the order of every 30 minutes [64].

Fault avoidance techniques such as ECC and redundancy come at the expense of more hardware and increased power consumption. Finally, the recent IC (Integrated Circuit) development is mainly driven by cost-effective segments (e.g. mobile) that do not demand high reliability and can easily tolerate certain errors. The vast majority of available hardware focuses on these markets and high-performance systems built out of commodity hardware especially suffer from a lower MTBF.

2.3.1 Failure Causes

The major cause for system failures is defective hardware, and as Figure 2.4 shows it can be any component in a system. There are, however, several other possible causes for system failures which are summarized in Table 2.2. Software, for example, is ranked

on the second place and is responsible for about 23 % of all failures on the example system. With more complex hardware architectures, hierarchies, and topologies, also software increasingly becomes more complex. [59] observed that there is a relationship between the software that runs on a system and its MTBF. Although software layers are able to detect errors caused by lower layers this process can be very complex. Furthermore, this information may not necessarily be trusted since the state of the software may be corrupted. Sometimes it is even not easy to track down the cause of an error, especially whether or not it was caused by software. For example, when the job finishes but only the final result is incorrect.

Any of the failures mentioned above will likely cause an entire job to fail and fault tolerance techniques were developed to mitigate the effects of system failures. The most commonly used approach is to periodically backup the system state in order to reduce the penalty for restarting jobs after a failure. This is known as Checkpoint/Restart.

2.3.2 Fault Tolerance using Checkpoint/Restart

Checkpointing was introduced to avoid restarting jobs from scratch. With checkpointing, programmers define states (checkpoints) of their application the job can rollback to upon recovery from a failure. Although applications can now restart from a more advanced state, application based checkpointing has a significant characteristic: all processes will roll back to the last well-known state even if only one of many processes has failed. An additional drawback is that the checkpoints have to be stored somewhere. They require extra storage and use I/O (Input/Output) and sometimes network bandwidth to transfer the data. Traditionally, checkpoints were written to the Parallel File System (PFS) which provides only very limited bandwidth since it is most often a shared resource among multiple systems. In an extreme scenario where the time it takes to write a checkpoint is close to or exceeds the MTBF, a job would spend most of the runtime just to checkpoint its data without making progress in the actual task.

Finding the optimum time interval between two consecutive checkpoints is rather complex and subject to intense investigation [67]. It requires a deep knowledge of the system architecture and the application.

Recently exceptional effort has been put into the prediction and prevention of system failures. The results in [68] show that under certain circumstances the failure prediction

recall[8] goes up to 50%. Proactive checkpointing [68] can then be used to back up the system state right before a failure occurs, reducing the amount of work lost. The authors also suggest spare nodes to replace other nodes that will fail soon, migrating repair time.

All these approaches come at a certain overhead and they are currently complimentary to periodic checkpointing which remains the prevalent fault tolerance technique. Checkpointing inevitably leads to longer application runtimes and it is desirable to reduce this overhead.

2.3.2.1 Mitigating Checkpointing Overhead

Several options to mitigate checkpointing overhead and to reduce its negative impact on application runtimes are available:

Reduced checkpoint size

It is the responsibility of the programmer to identify the parts that need to be stored in order to reduce data size but still allow for correct failure recovery. Incremental checkpointing can be used to reduce the size of consecutive checkpoints by only storing data that has changed since the last checkpoint. However, current approaches such as in [69] require significant modification to operating system kernels and may not be easily deployed.

Reduced checkpointing frequency

It is reasonable to decrease the checkpointing frequency to lower its overhead. Since applications will lose more progress upon a failure in this case, checkpointing frequency must be seen as a trade-off between MTBF and the time it takes to store (and restore from) a checkpoint. Interestingly, the more frequent checkpoints are created and written to the storage system, the more frequent specific components such as Solid State Drives (SSDs) with limited durability will fail.

Multilevel checkpointing

Multilevel checkpointing approaches make use of intermediate levels of storage that provide higher bandwidth than the slow PFS such as DRAM and local SSDs. The checkpoint is written to this faster storage and then asynchronously flushed to a higher storage layer via a dedicated thread [70] or an agent [71]. Meanwhile

[8] The prediction recall is the ratio of correctly predicted errors to the number of actual detected failures.

the corresponding process can continue with its task. Typically, the last level of checkpoint storage is still the PFS and not every checkpoint stored on a faster storage will be transferred to the PFS, but instead 1 out of 10 checkpoints for example.

Multilevel checkpointing and reducing the checkpoint size both allow to increase the checkpointing frequency, which may also reduce the rollback penalty for restart. A similar form of multilevel checkpoints is accomplished with burst buffers [72, 73], which are intermediate destinations in front of the PFS but can be mounted as regular file systems. Burst buffers exploit the *bursty* characteristic of checkpoint I/O where high bandwidth is only occasionally requested, which gives enough time to forward it to the PFS as final destination in between two checkpoints. Other approaches such as In-memory checkpointing [74] rely on a memory only checkpointing scheme, avoiding the relatively slow PFS. Although checkpointing to memory undoubtedly delivers the best performance it also requires multiple copies of a single checkpoint and multiple times more memory than required by the application. Moreover, when the memory is non-volatile, a node failure such as a simple power outage will erase the checkpoint.

Multilevel checkpoints provide a good trade-off between traditional PFS-based checkpointing and the in-memory approach. It allows for frequent, fine granular checkpointing and keeps the requirement for additional memory low at a reasonable performance degradation. One example implementation which has evolved as a de-facto standard is provided by the Scalable Checkpoint / Restart (SCR) library [71, 75]. As an alternative to SCR, the Fault Tolerant Interface (FTI) library [70] provides very similar features and is also widely used. As SCR was used in the DEEP-ER project it serves as reference and will be described more in detail.

One criteria that is often unnoticed is the effect of checkpointing on power consumption. Research in [76] showed that there is only little difference between checkpointing protocols and redundancy schemes. Moreover, the power consumption of computing and checkpointing was measured to be close. Depending on the checkpointing interval and duration, creating checkpoints can significantly influence application runtimes and will increase the power footprint.

2.3.3 SCR: Scalable Checkpoint / Restart

The SCR library provides a multilevel checkpointing solution for MPI applications. It is based on two key observations: First, only the most recent checkpoint is required

to successfully restart. Second, a system failure only disables a small portion of the system.
With these two observations SCR was designed to only store the most recent checkpoint to node-local storage, discarding any previous checkpoints. It also implements a redundancy scheme to support some node failures at reasonable network traffic and computation overhead. Storing checkpoints to node-local storage ensures system scalability since the checkpointing bandwidth scales with the number of nodes. However, even with SCR checkpoints must be occasionally written to the PFS. It is still required to recover from larger system or node-local storage failures. It must also be noted that the node-local storage may have limited endurance and frequent checkpointing to e.g. an SSD will limit its average lifetime to approximately 3 years. Additional techniques based on a hybrid DRAM/SSD approach were developed to increase the SSD lifetime [5].

Even though SCR manages checkpointing and restart by itself it is still up to the programmer to identify the parts of the code that need to be saved, and to make use of the respective function calls provided by SCR.

2.3.3.1 Redundancy Schemes

SCR provides three different checkpointing schemes:

Local Checkpoints are only written to the node-local storage. It is the fastest of the three checkpointing schemes but cannot withstand node failures.

Storage required for a checkpoint of B Bytes: B

Partner Checkpoints are written to the node-local storage and additionally to the local storage of a remote partner node (Figure 2.5). This scheme is slower than 'Local' but can withstand node failures, and even multiple node failures as long as a node and its partner do not fail simultaneously.

Storage required for a checkpoint of B Bytes: $2 \cdot B$

XOR With XOR, all available nodes are split into sets with N nodes each. Using a bit-wise XOR reduce operation, a parity information over all checkpoints in a set is calculated. Each node receives only a fraction of the parity which can then

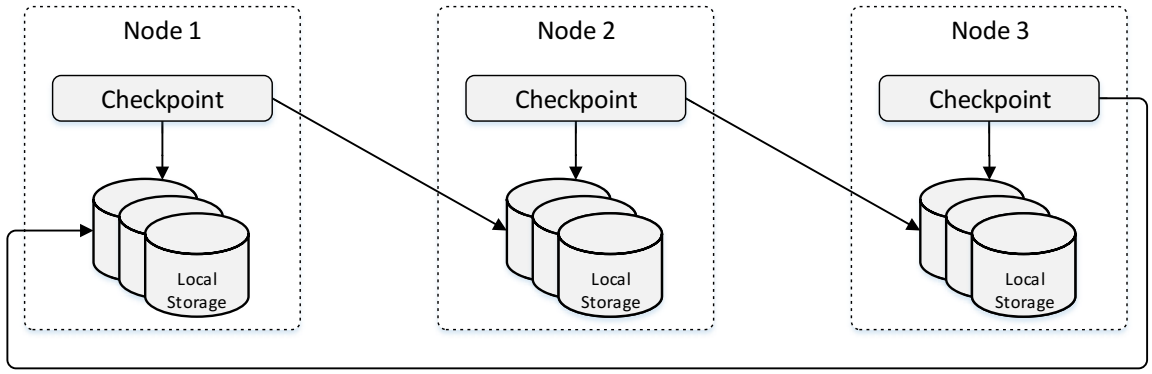


Fig. 2.5 SCR-Partner checkpointing scheme

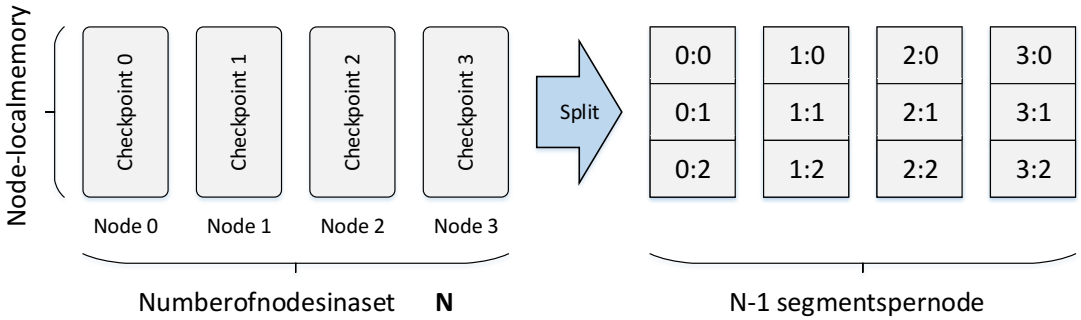
be used to recover from any single node failure within a set. XOR invokes more computation but requires less storage than 'Partner'. It can withstand multiple node failures as long as no more than one node within a set fails simultaneously.

$$\text{Storage required for a checkpoint of B Bytes:} \qquad B + \frac{B}{N-1}$$

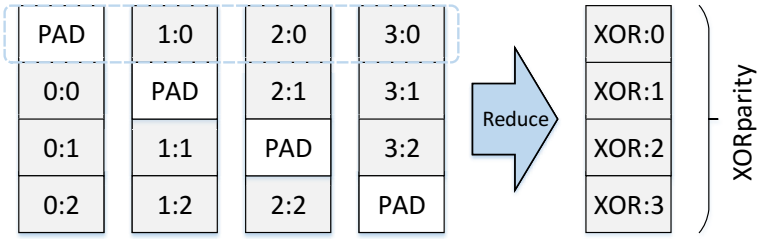
Local checkpointing is not a viable option for most systems as a single storage outage causes the scheme to fail. The Partner approach ensures the highest fault tolerance and is trivial and easy to implement but it requires the most storage space as every checkpoint is stored twice. SCR with XOR is a good trade-off in performance and storage requirements between these two approaches. It will be examined in detail next.

2.3.3.2 XOR Redundancy

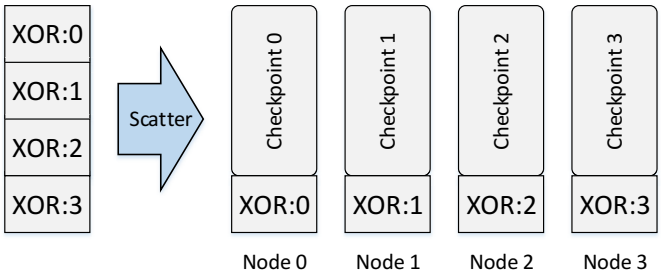
Figure 2.6 shows how SCR generates a XOR parity. As mentioned before, SCR with XOR splits the available number of nodes into sets. In a set of N nodes, the checkpoint file of each node is logically split into N-1 segments (Figure 2.6a). In the next stage, zero-padded segments are inserted so that every checkpoint now consists of N segments. All segments with the same index are then reduced via a bit-wise XOR operation (Figure 2.6b). This process may be implemented as a typical MPI collective operation which has been described in the previous section. Finally, one XOR parity information segment is distributed (*scattered*) to every node (Figure 2.6c). SCR provides several parameters to control the size of sets and the assignments of nodes to these. The configuration in the example provides one XOR set and can only tolerate a single node



(a) Logically split checkpoints of N nodes to N-1 segments



(b) Add alternating zero segments and reduce with bit-wise XOR



(c) Scatter XOR segments to nodes, one segment per node

Fig. 2.6 SCR XOR checkpointing example

failure. It is the responsibility of the user to create a reasonable number of sets to allow withstanding multi-node failures.

SCR is also able to handle multiple processes per node. In this case it will automatically select and create XOR sets so that every set has no more than one process of a particular node. Also, when a process writes more than one file during execution, SCR will combine these to a single checkpoint file. Finally, checkpointing files with arbitrary sizes are managed by determining the size of the largest checkpoint in a set and padding the remaining checkpoints with zeros up to this size.

2.3.4 Summary Fault Tolerance

Every component in a computing system is subject to failures and the Mean Time Between Failure decreases with an increasing number of components in large-scale systems. As systems fail unexpectedly and will continue to do so, work will be lost unless the accuracy of failure prediction models and migration strategies reaches 100%. Until then, fault tolerance using periodic checkpointing is inevitable and remains the prevalent fault tolerance technique.

To mitigate the checkpointing overhead, multilevel checkpointing libraries such as SCR were developed. SCR provides multiple levels of tolerance and different redundancy schemes to account for different checkpointing strategies, and SCR with XOR has been identified as a reasonable trade-off between performance and storage requirements.

SCR with XOR, however, involves inter-node communication and computation of the XOR parity result likewise. This will keep processors busy and increase the memory references to move intermediate results to and from the memory. It is therefore desirable to have an additional device that is able to offload computation, and at the same time reduce communication among nodes. This communication overhead is identical to MPI collective operations which has been identified as a major potential performance bottleneck in the previous section.

2.4 Summary

This chapter highlighted the importance of the memory interface and inter-node communication in today's and future large-scale systems. It became clear that memory has been and will remain one of the most critical bottlenecks with regards to performance and power. For many applications, communication overhead is already a large part of the overall application runtimes and the situation will become worse with growing system sizes.

As future systems will comprise many more components this will also lead to more frequent soft- and hard errors, increasing the importance of fault tolerance using periodic checkpointing to reduce the penalty of such failures. Unfortunately, writing checkpoints takes application time where no actual computation is performed. Since the performance for writing checkpoints also depends on the memory interface, the

interconnection network, and communication performance, it is desirable to improve these key elements.

In conclusion this chapter provided a strong motivation to develop a device that is able to mitigate the negative effects that were described above. Such a device must be able to offload computation from a host processor and simultaneously reduce inter-node communication.

CHAPTER 3

Hybrid Memory Cube

As an alternative to the DDR interface, to overcome its scalability issues (such as I/O pin, area, and load limitations), and to increase channel bandwidth, Micron recently proposed the Hybrid Memory Cube. The first section of this chapter introduces the HMC and analyzes the impact of its novel architecture on performance. Section two presents the implementation of the open-source HMC host controller openHMC. Section three evaluates HMC performance and power efficiency in a real system using the openHMC controller. A final summary concludes this chapter.

The findings of section one and three have been published in [11]. The implementation of the openHMC host controller is detailed in [10].

3.1 Introduction and Architecture Analysis

HMC is leveraging recent 3D fabrication processes to stack multiple layers of DRAM on top of a logic die. Its interface operates on a packet-based protocol utilizing high-speed SerDes (Serializer / Deserializer). As opposed to DDR, the HMC interface is not a JEDEC standard. Instead, Samsung Electronics and Micron Technology formed the *Hybrid Memory Cube Consortium (HMCC)* in October 2011 [37] and released the first HMC specification 1.0 in January 2013 [77]. It was later revised with the HMC specification 1.1 (HMC Gen 2 devices) which is the reference for this work. HMC

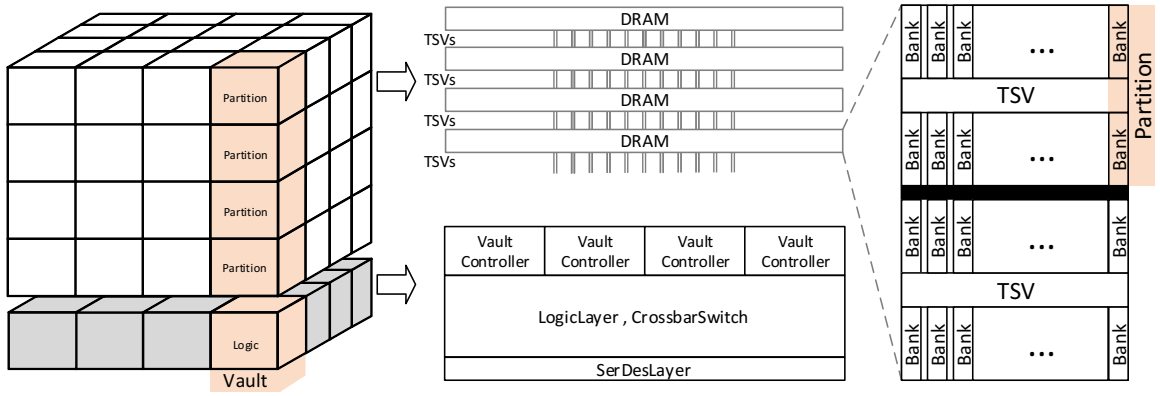


Fig. 3.1 HMC architecture overview

hardware engineering samples were available since 2013 and volume production started in June 2017 with 2 GB devices.

3.1.1 Architecture

Figure 3.1 shows the basic HMC architecture. Multiple layers of DRAM are stacked on top of a CMOS based logic layer using TSVs [28]. The stack is organized in 16 independent vaults where each vault connects the upper DRAM layers with a dedicated memory controller (the vault controller) using 32 TSVs [78]. Every DRAM layer comprises 16 partitions with 2 DRAM banks each. In [78], the HMC Gen1 DRAM stack was introduced as a composition of 68 mm^2 1 Gbit dies manufactured in 50 nm. Initially four layers were stacked for a total capacity of 512 MB. Current HMC Gen2 devices [79] stack four 4 Gbit DRAM dies on top of the logic base which increased the capacity to 2 GB ($4 \text{ layers} \cdot 16 \text{ partitions} \cdot 2 \text{ banks} = 128 \text{ banks}$). The capacity growth from Gen1 to Gen2 is based on denser memory arrays with a bank capacity increase from 4 MB (Gen1) to 16 MB (Gen2).

The HMC logic layer exposes four external links which can connect processors or other HMCs. Hence, multiple HMCs can be 'chained' together with varying routing options to increase the capacity (see Section 3.1.4). A single link comprises 16 bidirectional high-speed serial lanes. Every link is local to four vaults and a crossbar ensures that all links can access all vaults and other links (Figure 3.2). The 4-Link HMC comes in

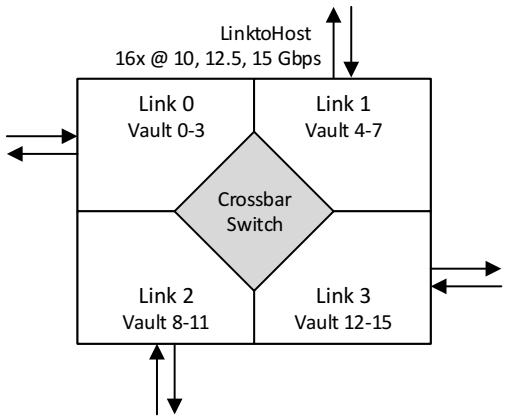


Fig. 3.2 HMC logic layer top view: schematic representation

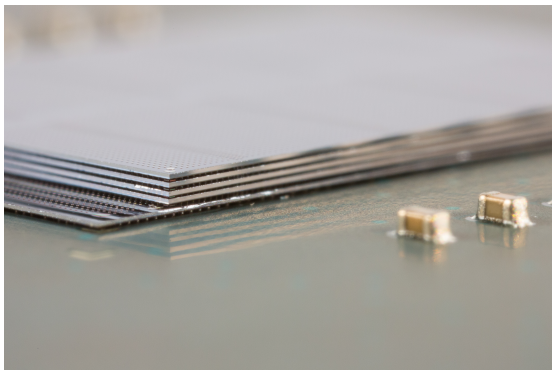


Fig. 3.3 Close-up view of an HMC stack. Image courtesy: Micron

a 31×31 mm package (896 balls)[1]. Figure 3.3 shows an HMC close-up view with four DRAM layers.

3.1.2 DRAM Organization and Performance

HMC implements a DRAM closed-page policy, i.e. the row buffers become inactive after each access. This is opposed to an open-page policy where a row stays active in the sense amplifiers until it times out or another row is accessed. While an open-page policy is in particular beneficial for applications with high locality (i.e. a high page $\frac{hit}{miss}$ ratio) it also increases power consumption since the sense amplifiers stay active after a memory access. Additionally, an open-page policy introduces delay on a page miss as pre-charge of the word-line does not occur immediately after the row has been accessed. As a result, a closed-page policy theoretically performs better for random access patterns.

The DRAM row or page size in HMC has been reduced to 256 Byte from 512 Byte - 2 KB for DDR4 [80] and to up to several kilobytes in DDR3 [81]. A smaller page size reduces the probability for a DRAM over-fetch where only a fraction of the information contained in an opened page is actually used, and therefore also reduces dynamic power consumption. It also makes an open-page policy impracticable and is another reason why the HMC developers preferred a closed-page policy.

[1] Initially, a 2-Link device with equal characteristics was available (19.5x16 mm package). Development and production of this device was canceled in late 2016 for unknown reason.

Performance numbers can be obtained from many sources [78, 79, 82]. Most of them highlight the potential link bandwidth of 240 GB/s (4 Links · 60 GB/s). The effective bandwidth, however, is limited by the vault controllers. With 32 TSVs each and a clock frequency of 1.25 GHz [83] a single vault can deliver 10 GB/s. With 16 vaults the maximum effective bandwidth is 160 GB/s. It is reasonable to provide more link bandwidth than the DRAM stack can deliver due to transaction layer (protocol) overhead on the link. The protocol will be discussed in a later section. Experiments in [83] showed that the maximum usable link bandwidth eventually flattens at 240 GB/s for a given 160 GB/s TSV or DRAM bandwidth.

Finally, HMC can be configured to internally remap memory addresses which can be a useful tool if the most commonly used access patterns are well-known. Per default, sequential requests will be spread over vaults, then banks, and finally DRAM to involve as many vaults as possible. This scheme has a simple, HMC specific background: the more vaults are accessed, the higher the parallelism and the potential bandwidth can get. Other access schemes may result in an imbalance of accessed vaults and address remapping can be used to correct this situation. The impact of various address-mapping modes on bandwidth at fixed access patterns will be evaluated in Section 3.3.3.

3.1.3 Link

A single HMC has four independent links, each comprised of 16 differential pairs (lanes) per direction, i.e. data to and from the HMC can flow at the same time. Individual links can be configured to run at 8 lanes (half-width) instead of 16 (full-width) if required. Available link speeds are 10 Gbps, 12.5 Gbps, and 15 Gbps. That is a maximum bandwidth of 16 lanes · 15 Gbps = 240 Gbps = 30 GB/s per direction or 60 GB/s bidirectional per link and 240 GB/s total. The maximum effective bandwidth is limited to 160 GB/s due to the vault bandwidths. The polarity of individual lanes can be inverted and the lane order can be reversed to simplify signal routing on a PCB. Each link is complemented by two power state signals (RXPS and TXPS). Finally, each HMC devices provides an active-low reset (PRST_N) and a unidirectional, HMC-driven fatal error indicator (FERR_N). Both sides of a link, Host and HMC, share a common reference clock which eliminates the need to transmit a dedicated clock along with the data-lanes.

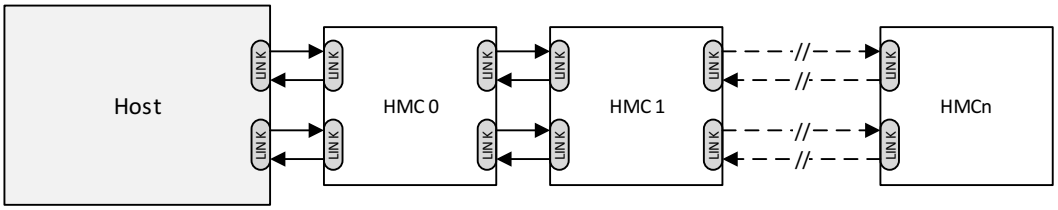


Fig. 3.4 HMC chain example: one host is connected to an HMC chain. Topology suggested in [79]

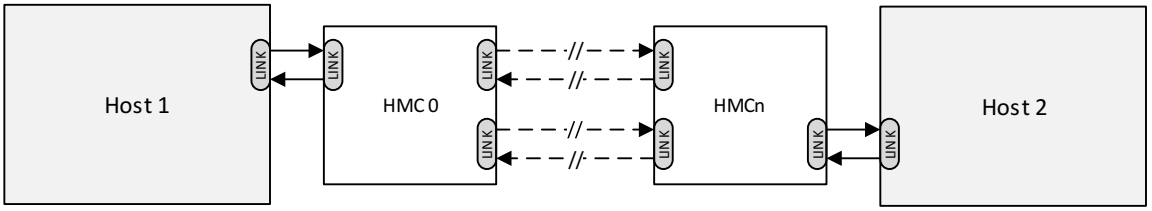


Fig. 3.5 HMC chain example: two hosts are connected to an HMC chain. All hosts can access any memory region. Topology suggested in [79]

3.1.4 Chaining

One notable feature is the ability to directly connect (to *chain*) multiple HMC devices to each other to increase the capacity (Figure 3.4). Also, multiple hosts can be connected to a network of HMCs for a shared memory environment (Figure 3.5). Chaining allows to create novel processor-memory architectures and communication schemes. It does not only increase the overall capacity but also enables processors to communicate through memory. Note that the HMC specification currently limits the total number of HMCs in a single network to 7 devices.

Enhanced approaches foresee dedicated interconnects and switches that connect only the memory modules for a large global or partitioned address space. Such an approach is presented in [84]. The author suggests an interconnection network for NAND-based flash chips. Such a memory subsystem provides a decent increase in capacity in combination with a good overall power footprint and reasonable performance. Similar memory subsystems could be created with HMCs. An intelligent interconnect with routing features would furthermore allow for more than 7 HMCs to coexist in a single memory subsystem. And lastly, if the interconnect provided additional interfaces to e.g. non-volatile memory, heterogeneous memory subsystems become feasible (see Figure 3.6).

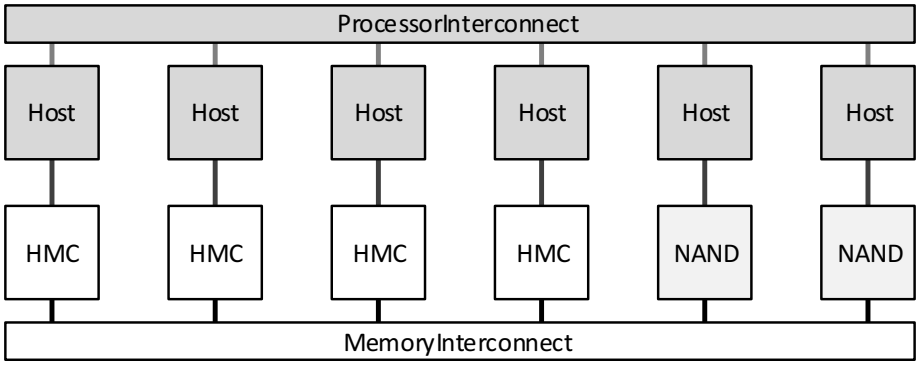


Fig. 3.6 HMC + NAND heterogeneous memory subsystem example. Topology with NAND only suggested in [84]

3.1.5 Protocol

The HMC communicates over a packed-based protocol. It defines a request-response communication with a granularity, or Flow Unit (FLIT) size, of 16 Bytes. The protocol supports reading and writing data packet sizes ranging from 16 to 128 Bytes along with command support for atomic operations and HMC configuration. Packets are framed by a header and a tail (8 Byte each) which results in a 16 Byte overhead per packet. Features such as CRC, a packet length check, and consecutive packet sequence numbers ensure link integrity. Complemented by a retry mechanism the HMC link can withstand bit errors that typically occur on serial high-speed links.

Responses are matched to non-posted requests using a 9 bit TAG field for up to 512 outstanding requests. Since the HMC logic die is able to reorder packets for faster execution (e.g. if a specific vault is accessed more frequently), responses may return out of order. However, HMC internally queues requests to the same vault/bank so that accesses to a specific location accessed from one link are always processed in order. Care must be taken when a memory location is accessed by more than one link since there is no guaranteed order for request execution across links. A small set of atomic operations is provided for computation offloading. These commands either add a single 16 Byte or two 8 Byte operands to a memory location via a read-modify-write operation. The potential benefits of offloading computation to the HMC will be evaluated in Section 3.3.6.

Flow control in both directions is achieved using tokens (credits), where one token represents buffer space for one FLIT. The use of tokens prevents the input buffer of the respective receiver from overflowing. Consequently, tokens are returned after packets

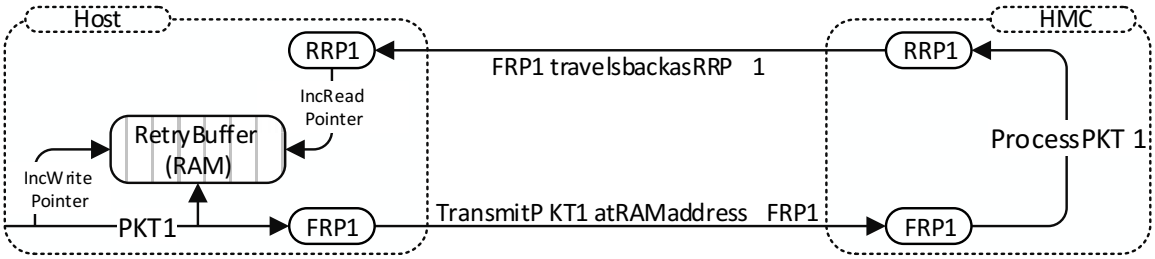


Fig. 3.7 HMC protocol FRP and RRP exchange loop

are processed by the receiver and the corresponding buffer space is freed up. Every packet that is transmitted also carries a pointer, the Forward Retry Pointer (FRP). The FRP represents the position of the packet in the retransmit/retry buffer of the sender. Flow packets are not subject to flow control and do not carry an FRP. As soon as the packet has been processed at the receiver, this pointer will be returned as Return Retry Pointer (RRP). The RRP signals the former requester that the packet was received and the space in the retransmission buffer can be reused. This process is depicted in Figure 3.7. Such flow control features can negatively influence performance and pose significant challenges for a host controller design. This *flow control barrier* will be described next.

3.1.6 The Flow Control Barrier

In order to maintain the best performance, the HMC specification defines two important metrics associated with flow control: the retry pointer loop time and the token return loop time. Both metrics originate from the fact that flow control is mandatory on a serial link that runs a protocol, and critical when it comes to saturate the theoretical link bandwidth. They will be described in the following.
Designers of a host controller should always keep these two metrics in mind. Especially when targeting FPGAs with relatively low operating frequencies, processing pointers and tokens can take up a large amount of the allowable return loop times.

3.1.6.1 Retry Pointer Loop Time

As mentioned earlier every packet that is sent on a link and subject to flow control will also be placed in the retry buffer of the respective requester. In addition, an FRP is also sent along with the packet, uniquely identifying the packet and its location in this buffer. The FRP is then extracted by the remote link partner and returned on

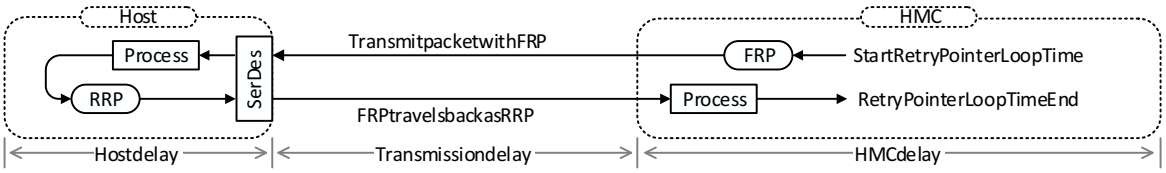


Fig. 3.8 Retry pointer loop time contributors

Table 3.1 Retry pointer loop time summary. The HMC internal clocking frequency is independent of the link width and speed

Lanes	Speed [Gbps]	HMC Retry Buffer Size [FLIT][a]	Retry Buffer Full Period [ns][b]	HMC Delay [ns][c]	Max Host Delay [ns]
8	10	192	307.20	26.5	280.70
	12.5	256	327.68	25.9	301.78
	15	256	273.07	22.3	250.77
16	10	192	153.60	26.5	127.10
	12.5	256	163.84	25.9	137.94
	15	256	136.53	22.3	114.23

[a] See Equation (3.1) and Equation (3.2)
[b] 16 lane values extracted from the HMC specification [79]
[c] Extracted from the HMC specification [79]

the response link as RRP, embedded in any packet that also carries valid flow control fields. This is the case for any packet that is not NULL, IRTRY (used to request a link retry or to clear error status), or erroneous. After the RRP has been extracted the read pointer of the requesters retry buffer can be moved. This invalidates the corresponding packet for potential retry and frees up its location for other packets. While this process is ongoing the requester is able issue many more FLITs and packets which fills up the retry buffer. Performance is throttled when the requester continues to send packets and fills the retry buffer faster than the required space is freed up. As a result, no more requests but NULL FLITs will be sent over the link, decreasing the effective bandwidth. To avoid this situation the HMC specification defines the maximum allowable time for the circulation time of the pointers, the *retry pointer loop time*. Figure 3.8 identifies its contributors: host delay, transmission delay which is negligible, and the delay through the HMC. In the figure, HMC acts as requester but the scheme applies for the host likewise. Now that the contributors are known, Table 3.1 summarizes the maximum values for the host delay portion of the retry pointer loop time. It can be seen that retry buffer full period and the allowable host

delay depend on the HMC link width and speed. Although not mentioned in the HMC specification this leads to the following two observations:

1. **The HMC retry buffer size for a link running at 10 Gbps is smaller than for 12.5 Gbps and 15 Gbps.**
   Equation (3.1) calculates the retry buffer size for a half-width (8 lane) link at 10 Gbps and Equation (3.2) at 12.5 Gbps, respectively. It can be seen that the retry buffer size increases from 192 to 256 FLITs for the faster configuration.

$$\begin{aligned}
&\text{BW: 8 lanes / 10 Gbps} && 8 \text{ lanes} \cdot 10 \text{ Gbps} = 80 \text{ Gbps} \\
&\text{Time to process a bit} && t_{bit} = \frac{1 \text{ bit}}{80 \text{ Gbps}} = 1.25 \cdot 10^{-11} \text{ s} = 12.5 \text{ ps} \\
&\text{Time to process a FLIT} && t_{FLIT} = 128 \cdot t_{bit} = 128 \cdot 12.5 \text{ ps} = 1.6 \text{ ns} \\
&\text{Retry buffer size [FLITs]} && \frac{\text{Full period at 10 Gbps}}{t_{FLIT}} = \frac{307.2 \text{ ns}}{1.6 \text{ ns}} = 192
\end{aligned} \tag{3.1}$$

$$\begin{aligned}
&\text{BW: 8 lanes / 12.5 Gbps} && 8 \text{ lanes} \cdot 12.5 \text{ Gbps} = 100 \text{ Gbps} \\
&\text{Time to process a bit} && t_{bit} = \frac{1 \text{ bit}}{100 \text{ Gbps}} = 1 \cdot 10^{-11} \text{ s} = 10 \text{ ps} \\
&\text{Time to process a FLIT} && t_{FLIT} = 128 \cdot t_{bit} = 128 \cdot 10 \text{ ps} = 1.28 \text{ ns} \\
&\text{Retry buffer size [FLITs]} && \frac{\text{Full period at 12.5 Gbps}}{t_{FLIT}} = \frac{327.68 \text{ ns}}{1.28 \text{ ns}} = 256
\end{aligned} \tag{3.2}$$

   The reason for a smaller retry buffer at 10 Gbps is a decrease in the internal HMC datapath-width to reduce power consumption by shutting down parts of the logic, including the retry buffer[2]. In contrast, running at 12.5 or 15 Gbps increases performance and in particular the 12.5 Gbps option provides the highest allowable retry pointer host delay portion. It can ease the implementation of a corresponding host controller.

2. **The retry buffer full period is twice as high when operating in half-width (8 lane) mode.**
   Table 3.1 highlights the retry buffer full period for all available HMC link configurations. However, only the values for a 16 lane configuration are mentioned in the HMC specification [79]. It is a reasonable expectation that the retry buffer

[2] Further details are available under NDA with Micron.

full period doubles if only half of the bandwidth is provided, assuming the size of the retry buffer is maintained. To prove this, hardware measurements were conducted with 8 and 16 lanes. The host issued read requests without returning RRPs so that the HMC will not free up any used retry buffer space. The results showed that the retry buffer size is independent of the link width. This leads to the conclusion that the time it takes to entirely fill the retry buffer is doubled when a link is operated in half-width. In fact, HMC stopped responding although theoretically there were a few tokens left. It is common practice to set such thresholds lower than the actual limit suggests. It can ease the implementation and save logic as the need for a fine-grain, FLIT or token based granularity is eliminated.

3.1.6.2 Token Return Loop Time

The second important factor to avoid performance throttling is the time it takes to consume, process, and return tokens for transmitted and received packets. Similar to the retry pointer loop time, returning tokens too slow will throttle link packet transmission. This is in particular the case when the input buffer in the host controller or the HMC runs full. Processing tokens is different to pointers which may be returned immediately after passing integrity checks. Packet tokens can only be returned after a packet passed the receiver's input buffer. In addition, current HMC devices only provide a maximum of 219 tokens for the host to transmit packets which can cause the host controller to run out of tokens even faster. Therefore, the token return time constraint is potentially even harder to meet than the retry pointer loop time.

3.1.6.3 Mitigation Techniques

Several techniques can be applied to handle retry pointer and token loop time violations. If tokens are the limitation, the HMC-supported open-response loop mode can be entered. In this mode, HMC will not check for free space in the host's input buffer but instead immediately return responses. Consequently, the hosts input buffer can run full if a user application is not able to receive data at the same speed. Host-sided optimizations for both types of loops include removing link integrity checks to lower the loop delay and a low latency transceiver design. Removing integrity checks, however, is highly discouraged as bit errors may lead to undefined link states. Increasing the host controllers internal operating frequency and moving from 10 Gbps to 12.5 Gbps

may also be considered and both of these options require the least design modification effort.

3.1.7 Summary HMC Architecture

In conclusion, the following advantages and disadvantages of the HMC interface can be obtained from this section. The main performance and power characteristics will be thoroughly evaluated in Section 3.3.

3.1.7.1 Advantages

- **Bandwidth** Due to the high parallelism inside the HMC the total theoretical bandwidth sums up to 160 GB/s per device. The impact of protocol overhead is alleviated by providing 240 GB/s link bandwidth.
- **Average latency** High parallelism and the ability to issue many in-flight transactions decrease the average latency. This is opposed to DDR systems with pins sharing transmit and receive direction and where the number of simultaneous requests is limited by the number of banks connected to this channel.
- **Heterogeneous die stacking** Heterogeneous die stacking using TSVs enables to combine multiple dies that were manufactured in a different technology, such as CMOS and DRAM. The yield increases since single layers can be tested prior assembly.
- **Footprint and I/O requirements** The HMC package significantly reduces footprint requirements by 90% over DDR DIMMs [85]. The serialized links lower the number of I/O pins required to connect a processor from several hundreds to only 64 pins per link (16 differential lanes, two directions) and a few additional sideband signals. Therefore, HMC can help to overcome the memory scaling issues for processors and eases PCB development.
- **Capacity** Stacking multiple DRAM layers increases the memory density while the footprint remains unchanged. It is one solution to delay the impact of the upcoming end of the miniaturization process.
- **Energy efficiency** Shorter memory subsystem traces and reduced wire-loads contribute to the overall power efficiency. HMC also provides a power-down

mode to shut down one or more serial links and parts of the logic base, if desired. Micron claims that HMC uses only 10% energy per bit of current memory systems [85].

- **Atomic operations** HMC is able to carry out simple integer ADD operations which can be utilized to offload such computations from the host processor. These operations, implemented in a logic layer right next to the memory cells, can be categorized as PIM.
- **Interface abstraction** The abstraction of the actual memory interface is another key benefit of the HMC. Although the overhead through (de-)serialization and the transaction layer increase the latency, an abstracted interface significantly eases the implementation of a corresponding host controller. It is furthermore a key element to accelerate future moves to other memory technologies by speeding up their adoption.

3.1.7.2 Disadvantages

- **Single access latency** Serial links and protocol processing introduce additional delay and therefore increase the single access latency. In order to benefit from the HMC characteristics the link should be kept busy with as many in-flight transactions as possible. This might require modifications to existing applications.
- **Capacity** The capacity of current HMC devices is 2 GB and therefore much less than most systems and applications require. It is also approximately 5 years behind the evolution of DDR capacities [86]. Although chaining seems to be a viable way to increase the HMC capacity it comes with the major drawback: the total available bandwidth available to the host will still be limited by the link bandwidth of the 'local' HMC. Also, to the best knowledge of the author, chaining has not been evaluated in detail yet and the real performance remains unclear.

3.1.8 Outlook

In 2014, the HMCC has announced the next-generation HMC Gen3 devices (HMC specification 2.0 [87]). Along with a new very-short reach interface with up to 30 Gbps per link, Gen3 also supports quarter-width (4 lane) link operation. Initial cube

capacities were reported with 4 GB and 8 GB. The protocol is enhanced to support additional atomic and arithmetic operations. In addition, the maximum packet size is increased to 256 Byte to match the HMC DRAM page size and further increase the overall link efficiency.

As of July 2017, Gen2 devices have reached volume production status with 2 GB densities while HMC Gen3 has been taken off the roadmap. Micron states that at least for now there is no demand for HMC links that can provide twice the bandwidth of Gen2 devices.

3.1.9 Lessons learned for an HMC host controller design

The lessons learned in this chapter are very important for the design of an HMC host controller and corresponding applications. It is particularly helpful to note the following key characteristics in order to achieve best performance and usability.

- Serial links in combination with memory abstraction using a communication protocol require flow control and error handling mechanisms on both sides of a link. The drawback here is that even though the maximum bandwidth could be potentially delivered by raw link parameters, processing and the exchange of pointers and tokens are subject to hard restrictions. A well designed host controller must especially provide a short flow control loop to perform well. While this does not seem to affect Application-Specific Integrated Circuit (ASIC) implementations the relatively low frequencies in FPGAs can become a show stopper. Another major contributor to the loop times is the delay through the SerDes (see Section 3.3.5). Especially user-friendly SerDes instances created by the FPGA design tools often use deep buffer structures which results in unnecessary high delays.

- As opposed to a transactional interface (such as DDR) where no overhead is transmitted on the link, a protocol-based communication requires packet framing to exchange flow control items and to distinguish packets. These additional items appear as pure overhead on the link and therefore lower the effective bandwidth. In case of the HMC every packet results in additional 16 Byte not carrying any data. Section 3.3 will highlight that the effective peak link bandwidth is approximately 83 % of the theoretical bandwidth for 128 Byte requests. Smaller requests will decrease the usable bandwidth even further as they increase the

protocol overhead and can cause bank access conflicts. Application developers and memory management units should be aware of that fact in order to optimize link utilization.

3.2 openHMC Host Controller

As for any other memory interface, HMC requires a host controller it can be connected to and the previous section has identified several requirements for such a controller. Besides compliance with the transaction layer of the specification it became clear that a low latency design is crucial for performance reasons. At the time of writing only a few host controller solutions were available (e.g. [88, 89]) and none of them was affordable. One low-cost solution was provided by Altera Corporation (now part of Intel) in 2015, called HMC Controller MegaCore Intellectual Property (IP) [90]. This core can be generated within the Quartus II or Quartus Prime software for use in the Arria 10 FPGA series. A second alternative was made available by Xilinx by the end of 2016. Their IP can be generated in the Vivado Design Suite targeting latest Xilinx Ultrascale and Ultrascale+ devices. Since the target FPGA used in this thesis is a Xilinx Virtex 7 and because the development itself started before either of these cores was available, a custom host controller named openHMC was developed. This section highlights the most important technical details. A full reference is available in [10] and [91].

openHMC is a configurable, vendor-agnostic, and open-source HMC controller IP. The first revision was released in September 2014. Meanwhile the fifth revision is publicly available[3] as a Verilog package including a custom simulation model along with a detailed documentation [91]. It has also been presented and evaluated in [10]. openHMC is licensed under the Lesser General Public License (LGPL) version 3. The LGPL states that the core may be used in proprietary projects without limitations but any changes to the core itself must be made publicly available.

3.2.1 Configurations and Features

openHMC fully complies with the HMC specification 1.1 [79] and provides additional valuable features such as:

[3] http://www.uni-heidelberg.de/openhmc

Table 3.2 Resource utilization for an 8x half-width link at 10 Gbps in a Xilinx Virtex 7 690T FPGA. Percentages reflect the overall usage in the respective device. Xilinx and Altera core statistics provided as reference

Core	IF width	LUTs	Registers	BRAM	DSPs
openHMC standard	256 bit	11710 (2.7%)	12486 (1.4%)	8 (0.4%)	0
	512 bit	25307 (5.8%)	23973 (2.7%)	10 (1.0%)	0
	768 bit	48806 (11.2%)	36129 (4.1%)	23 (1.5%)	0
	1024 bit	81412 (18.8%)	48885 (5.6%)	31 (2.1%)	0
openHMC w/ XILINX define	256 bit	7133 (1.6%)	7580 (0.8%)	8 (0.4%)	10 (0.3%)
	512 bit	16426 (3.7%)	14346 (1.6%)	10 (1.0%)	10 (0.3%)
	768 bit	35652 (8.2%)	21787 (2.5%)	23 (1.5%)	10 (0.3%)
	1024 bit	63531 (14.7%)	29773 (3.4%)	31 (2.1%)	10 (0.3%)
Xilinx IP [92][a]	512 bit	18077	19367	36	0
Altera IP [90][b]	512 bit	24400 (ALMs)	48200	51 (M20K)	–

Legend: BRAM = Block RAM (36 Kb memory unit)
DSP = Digital Signal Processor, IF = Interface, LUT = LookUp Table

[a] Device: Ultrascale XCVU190. Results reflect a full-width configuration
[b] Device: Arria 10. ALM = Adaptive Logic Module, M20K = 20 Kb memory unit

- A configurable, synchronous or asynchronous AXI4 Stream user interface with 256, 512, 768, or 1024 bit datapath
- Half-width (8x) and full-width (16x) HMC link support for all available datapath-widths and link speeds (10, 12.5, 15 Gbps)
- No vendor specific components to target all types of FPGAs and ASICs
- Additional switch to turn selected building blocks into Xilinx specific components to optimally use device resources. Refer to the openHMC documentation [91] for more details

Depending on the selected datapath-width and whether Xilinx specific components shall be used the openHMC resource utilization varies. The amount of device resources required after place and route is shown in Table 3.2. The results were obtained

using the default openHMC parameter set and default synthesis and implementation strategies in Vivado 2016-2. These numbers will more or less slightly vary for other settings or tool versions. It can be seen that doubling the datapath-width also doubles the number of registers but almost quadruples logic complexity and the amount of LUTs required. Using the XILINX parameter results in easy resource savings and more efficient FPGA fabric utilization. There is no change in Block RAM usage because Vivado automatically maps suitable register arrays such as FIFOs (First In - First Outs) and RAM (Random-Access Memory) to Block RAM. Overall, the openHMC controller is a very compact and easy to implement solution. Experience shows that a 512 bit datapath is most often the best trade-off between speed, design complexity, and usability.

3.2.2 Operating Frequencies

The openHMC core provides 24 individual configurations (Table 3.3). The resulting core clock frequency is calculated with Equation (3.3) where NUM_LANES is either 8 or 16, LINK_SPEED is 10, 12.5, or 15 Gbps and DATAPATH_WIDTH the width of user interface in bit.

$$\text{core clock [MHz]} = \frac{NUM\_LANES \cdot LINK\_SPEED}{DATAPATH\_WIDTH \cdot 10^6} \tag{3.3}$$

All frequencies marked in gray were successfully implemented and tested in hardware. The configuration that is used throughout this thesis is highlighted in boldface.

3.2.3 Flow Control and Performance

The requirements for retry pointer and token return loop times were mentioned in Section 3.1.6. The openHMC specification highlights that the controller meets both requirements for most configurations depending on the operating frequency. The results, however, assume that the SerDes are optimized for low latency. In case a host design experiences performance limitations through loop time violations the openHMC controller provides several options to further decrease the delay. These options include HMC open-response loop mode and deactivation/removal of link integrity features.

Table 3.3 openHMC core clock frequencies [MHz] for various configuration. The configuration in bold is the reference for this thesis. Configurations marked in gray were successfully implemented and tested in a Xilinx Virtex 7 690T FPGA

HMC Link Parameters		Datapath-width			
Width	Speed in Gbps	256 bit	512 bit	768 bit	1024 bit
half-width (8x)	10	312.5	156.25	104.17	78.125
	12.5	390.625	195.3125	130.208	97.65625
	15	468.75	234.375	156.25	117.1875
full-width (16x)	10	625	**312.5**	208.33	156.25
	12.5	781.25	390.625	260.147	195.3125
	15	937.5	468.75	312.5	234.375

3.2.4 Comparison with other IPs

Table 3.2 compares the openHMC HMC IP to the Xilinx and Altera ones. It must be noted that the Altera core has fixed user interface widths; 256 bit for a half-width (8x) HMC link and 512 bit for a full-width (16x) link. The comparison highlights that the openHMC controller requires about 50% less registers and only slightly more LookUp Tables (LUTs) without the XILINX parameter set. With this parameter set, openHMC requires only one-third registers and about two-thirds LUTs. In both cases it also consumes about 60% less memory cells. Although the cores were mapped to a different FPGA technology (Altera vs. Xilinx) the comparison of the two is reasonable. The largest difference is the naming since both use 6-input LookUp Tables. In addition to the difference in resource utilization the biggest advantage of the openHMC controller is its flexibility. While the Altera core is limited to only two possible user interface width/HMC link configurations, openHMC supports 24. One remarkable feature of the Altera core is the ability to reorder incoming HMC responses but using this feature will further increase the resources required by 10 to 20 percent.

The resource utilization of the Xilinx IP is comparable to openHMC. It also provides a broader range of user interface widths, from 256 bit up to 2048 bit. Response reordering and a multi-channel user interface are additional valuable features. Both of vendor IPs, however, are based on evaluation-only licenses. The cost of purchasing an enhanced license to integrate these cores into products is not known the author.

Table 3.4 openHMC ASIC implementation results

Process Node	Gates	Area [mm^2]	% SRAM of area	Fmax
65 nm general purpose	41900	0.921	75 (0.69 mm^2)	415 MHz
28 nm high-performance	41600	0.223	62 (0.14 mm^2)	1 GHz

3.2.5 ASIC Implementation

The openHMC controller was implemented with two different process nodes without any additional optimizations. The configuration was set to a 256 bit datapath and all other parameters were left at their standard values. Table 3.4 summarizes the post-synthesis results with the Cadence Genus Synthesis Solution at the slowest process corner (minimum voltage, -40°C). As can be seen, the estimated resource utilization between the two processes remains comparable while the required area is significantly smaller in 28 nm. The maximum operating frequency Fmax is expected to reach approximately 415 MHz in a conservative, 65 nm general purpose process. The relatively slow SRAMs prohibit higher frequencies. For the more advanced 28 nm node, however, it scales up to 1 GHz. According to Table 3.3 it becomes clear that in 28 nm the openHMC controller can be implemented with the fastest available link speed (15 Gbps) at 16 lanes. It furthermore eliminates any concerns regarding flow control performance aspects as processing pointers and tokens takes place much faster.

3.3 HMC Performance Evaluation

Since its introduction in 2011 [37], HMC has been research topic and investigation target in various publications.

In [83] the author theoretically evaluates the HMC using preliminary data that was available at that time. The evaluation is based on simulations and contains many assumptions for a variety of parameters that affect performance and power. Since then, several other simulation models were proposed, e.g. more general ones toward 3D stacked architectures [93] and cycle-accurate simulators [94]. Others presented techniques to improve the HMC architecture by either optimizing the DRAM refresh mechanism ([95, 96]) or reducing thermal dissipation through data compression algorithms as in [97].

[98] initially explores the HMC capabilities with application-near memory access traces. The authors in [99] provide a more general study and highlight the importance of request sizes and access patterns on performance. Another approach in [100] attempts to give a more detailed characterization. In this work, however, the experimental setup turns out to be a performance limitation as it only supports half-width (8x) HMC links and uses the meanwhile discontinued 2-Link HMC device.

The following section extends the findings in [98],[99], and [100] by providing an ultimate general overview about the impact of access patterns on metrics such as bandwidth, latency, and power consumption. Understanding these characteristics is a must for system engineers and application developers who want to optimally use this new technology.

In order to provide solid results, the HMC is thoroughly evaluated in a real system environment. The test setup and host controller can support the full HMC performance in various link configurations. A comprehensible overview for various metrics will determine whether or not HMC can satisfy the expectations.

3.3.1 Metrics

Only a few base metrics are required to qualify a memory device. In general, it is important to clearly understand these to compare individual memory technologies and to select the best candidate for a given scenario. The metrics evaluated in this section are:

Bandwidth The bandwidth is one of the most important memory interface metrics. One must distinct between two related bandwidth measures: the total and the effective bandwidth.

- Total: The total (raw) bandwidth is the maximum number of information a link can transport in a given time period.
- Effective: The effective bandwidth is the maximum number of payload a link is capable to transport. The effective bandwidth is per definition equal to or less than the total. Serialized links that run a protocol require certain overhead to be transmitted along with the actual payload, e.g. 8b/10b encoding and packet framing. Hence their effective bandwidth is less than the total.

Latency Generally, latency is defined as the time it takes to transport an information from one point to another. In case of the HMC, the read latency defines the time it takes for a request to become available on the host controller transmit interface until a corresponding response is seen at the host receiving application.

Power Consumption The power consumption (or simply power) describes how much energy a device uses (or generates) at any point in time. The two most common measures are Joule per second ($\frac{J}{s}$) and Watt (W) with $W = \frac{J}{s}$. Since power increasingly moves into focus it is more important than ever to obey power budgets. Power consumption limits can be identified on a system level and for individual components such as 25 Watts for a PCIe connector (if no additional, external connector is used).

Power Efficiency Power efficiency describes how much energy (not to be confused with power) is required to transmit a given number of information measured in Joule per bit ($\frac{J}{bit}$). When referring to power efficiency only the effective bandwidth is considered.

3.3.2 Test Setup

In order to obtain reliable numbers a test setup as shown in Figure 3.9 was created. It comprises a Xilinx Virtex 7 690T FPGA that connects a 2 GB, 4-Link HMC[4] with a full-width (16x) link at 10 Gbps and 12.5 Gbps. Implementing a 15 Gbps link was not possible as such high lane speeds are not supported by the Xilinx 7 series. The openHMC controller is used as HMC host controller. A low-impedance, high-precision resistor per individual power rail is used to measure the voltage drop via a Linear Technologies DC1613A PMBus module. As the electric current is known this leads to the power consumption. The HMC address-mapping mode is configured to low-interleave and the maximum block size is set to 128 Byte. Address (re-)mapping in the HMC core logic can be a useful tool to optimize performance for given access patterns and will be discussed in Section 3.3.3.

Before the actual results are presented it is crucial to understand the impact of access patterns on bandwidth. This will also help to avoid pitfalls in a host controller and application design.

[4] Logic revision 2, firmware 0.95A, part number MT43A4G40200NFA-S15 ES:A

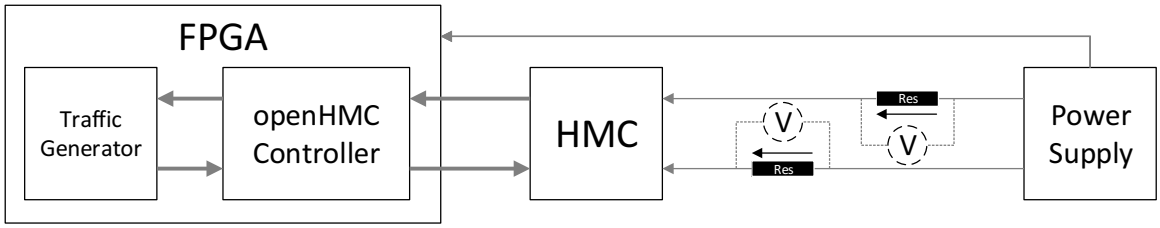
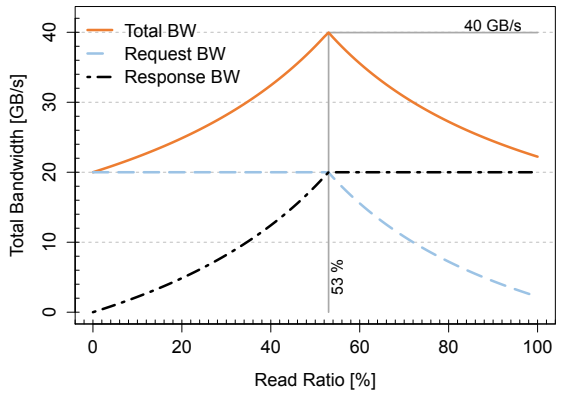


Fig. 3.9 Experimental test setup

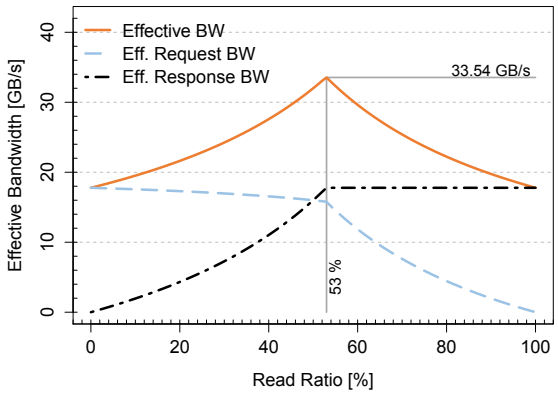
3.3.3 Access Patterns

Traditional DDR interfaces operate on a single transactional data-bus to transmit and receive data to and from the DRAM. Also, a single channel can only serve one command at a time so that execution of subsequent commands requires the previous command to complete first. In contrary, HMC comes with a bidirectional communication scheme where requests and responses are transmitted on separate channels. This circumstance requires well-balanced access patterns, i.e. an optimum read/write ratio in order to efficiently utilize both link directions and to maximize bandwidth. The HMC access granularity is 16 Byte and supported packet sizes range from 16 Byte up to 128 Byte. Every transmitted packet also requires an additional protocol overhead of 16 Byte. Therefore, a maximum-sized write request of 128 Byte comprises 8 FLITs payload and 1 FLIT overhead (=9 FLITs total) to be transmitted on the request channel. A read request appears as 16 Byte overhead on this channel and returns one FLIT overhead and up to 8 FLITs payload in response direction. Due to this fact the optimum HMC read-to-write ratio is not $\frac{1read}{1write}$ as a single maximum-sized read+write results in 10 FLITs on the request channel while only 9 FLITs will be returned.

Figure 3.10a shows the impact of read-to-write ratio on the total request, response, and combined bandwidth for maximum-sized, 128 Byte read and write requests. A read ratio of 53 % maximizes the total bandwidth including packet overhead. Similarly, Figure 3.10b presents the impact on the effective bandwidth. The figures represent the bandwidth for a full-width (16x) link at 10 Gbps. The bandwidth increases linearly with the lane speed and results for 12.5 and 15 Gbps can be obtained by multiplying the bandwidth by 1.25 and 1.5 respectively. It can be seen that the maximum effective bandwidth (i.e. excluding protocol overhead) in a 10 Gbps configuration is 33.5 GB/s (≈83.5 % efficiency). Furthermore the actual optimum ratio depends on the request sizes as shown in Figure 3.11. It can be seen that the optimum ratio shifts toward more read requests as request sizes become smaller since the percentage of overhead



(a) Total bandwidth



(b) Effective bandwidth

Fig. 3.10 Impact of read/write ratio on bandwidth with 128 Byte requests

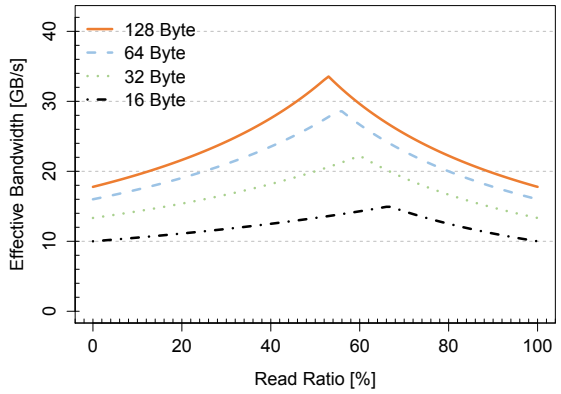


Fig. 3.11 Impact of different request sizes on the optimum read/write ratio

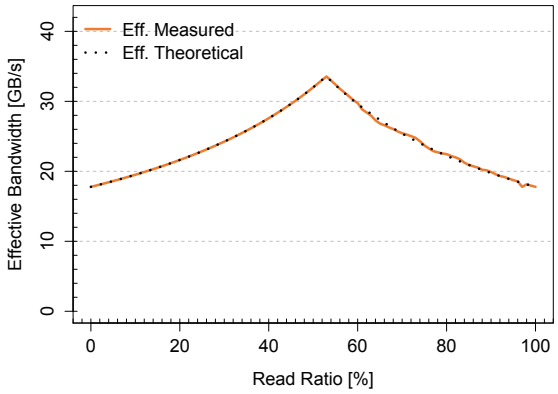


Fig. 3.12 128 Byte request ratio sweep results: theoretical versus measured

per request increases. Table 3.5 summarizes the results for each of the possible request sizes.

It is not only important to maintain the optimum ratio but also the ordering of requests is relevant. In a worst case scenario instead interleaving reads and writes in a stream of 100 × 128 Byte requests with the optimum ratio of 53 %, the user issues 53 reads followed by 47 writes. This is referred to as *bad request practice* and its impact on the overall bandwidth will be discussed in the following.

Table 3.5 Optimum ratio and maximum effective bandwidth per request size

Request Size [Byte]		16	32	48	64	80	96	112	128
Optimum Ratio [% Read]		66	60	57	55	55	54	53	53
Maximum effective BW [GB/s]	10 Gbps	14.93	22.2	26.2	28.6	30.3	31.75	32.6	33.54
	12.5 Gbps	18.66	27.8	32.8	35.7	37.9	39.7	40.8	41.9
	15 Gbps[a]	22.4	33.3	39.3	42.85	45.5	47.6	48.95	50.3

[a] Listed only for reference. Not verified in hardware

3.3.4 Bandwidth

Several access pattern schemes were identified and tested. While some of them perform best with the default address-mapping mode (low-interleave, see Section 3.1.2), others will benefit from a different address-mapping or maximum block size setting. Bandwidth and efficiency numbers in this section were rounded down to account for measurement errors.

The first measurement is shown in Figure 3.12. It compares a sweep of the read ratio for 128 Byte requests between expected and measured effective bandwidth. The results very closely match the theoretical evaluation. Only a slight deviation appears for higher read ratios, most likely due to measurement errors and/or the negative impact of violating the retry pointer or token return loop times. The more reads are issued; the more responses are generated. Consequently, the retry buffer fills up faster and the loop time constraints are tightened. Several ways to alleviate loop time violations were proposed earlier. The results for a 12.5 Gbps link are very similar and not shown. They can be calculated by multiplying the 10 Gbps results by 1.25.

Figure 3.13 shows a plot of various access patterns for 128 Byte requests and their impact on the measured effective bandwidth at 10 Gbps. It can be seen that linear reading and writing deliver the theoretical maximum of 17.7 GB/s with an efficiency of 88.5 % per link direction (see Equation (3.4)). An additional experiment with strided accesses unveils a drop in bandwidth for stride=16, where stride=1 represents linear reading/writing throughout all vaults.

$$\text{Read or write link efficiency:} \qquad \frac{\text{Effective bandwidth}}{\text{Total bandwidth}} = \frac{17.7\ \text{GB/s}}{20\ \text{GB/s}} = 88.5\ \% \quad (3.4)$$

With a stride of 16 and low-interleave address-mapping only 1 vault is continuously accessed. The peak bandwidth for writing a single vault is 9.8 GB/s and 9.35 GB/s for reading, respectively. These results closely reflect the maximum vault bandwidth of 10 GB/s, lowered by packet processing overhead. In general, increasing the stride will only affect the bandwidth when the number of accessed vaults and therefore the provided vault bandwidth is lower than the effective link bandwidth. For a given strided access pattern changing the address-mapping mode can eliminate this limitation. In the previous case, shifting vault and bank address segments to higher address bits will improve stride=16 accesses. All other strided accesses, however, will negatively impact performance due to vault congestion.

The optimum read ratio of 53 % gives the maximum effective bandwidth of 33.5 GB/s for linear accesses (83.75 % efficiency, see Equation (3.5)) and 8.9 GB/s for a single vault. A reasonable expectation would be that the efficiency stayed constant compared to only reading or writing at a time, which was measured with an efficiency of 88.5 %. Mixing reads and writes, however, increases the protocol overhead in request direction which now carries 1 out of 9 FLITs overhead for writes and 1 FLIT pure overhead for every read that is sent.

$$\text{Combined R/W link efficiency: } \frac{\text{Effective bandwidth}}{\text{Total bandwidth}} = \frac{33.5 \text{ GB/s}}{40 \text{ GB/s}} = 83.5 \text{ \%} \quad (3.5)$$

Random accesses do not show an impact when addressing all vaults while the single vault bandwidth drops to 7.58 GB/s due to the increased probability of bank conflicts. Increasing the lane speed to 12.5 Gbps does not improve the single vault performance as shown in Figure 3.14. For all other access patterns, however, the results represent what has been theoretically evaluated earlier.

The term *bad request practice* was introduced to describe bad ordering of requests in a stream for a given read ratio. The negative impact of this bad request practice turns out to be negligible in a stream of 100 requests. Restrictions in the FPGA design prohibited the use of longer sequences which will decrease the achievable bandwidth. Hence, although the HMC is capable to internally reorder independent requests, bad request ordering over a longer period of time should be avoided. It will lead to inefficient utilization of either of the two link directions. If required, host-sided reordering should be performed in order to maintain highest link bandwidth.

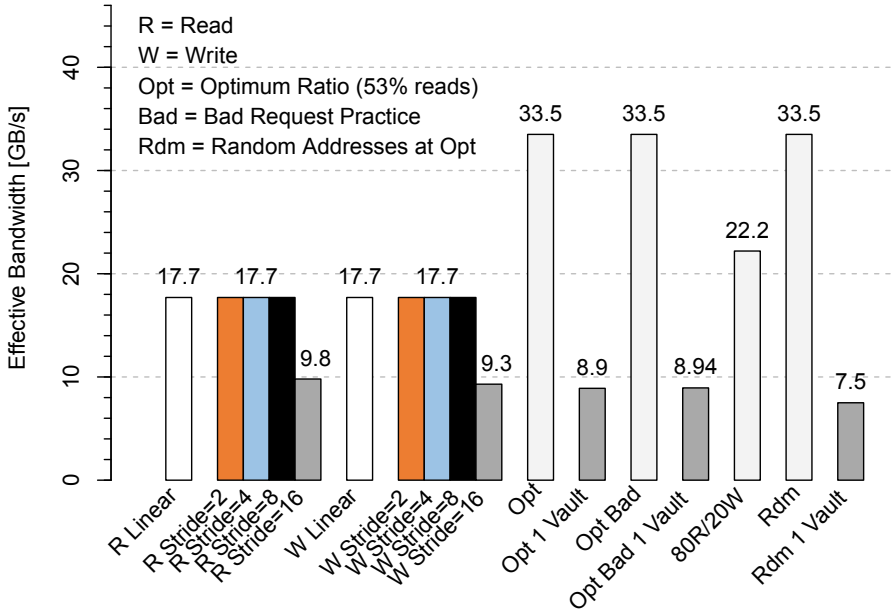


Fig. 3.13 Effective bandwidth for different access patterns with 128 Byte requests at 10 Gbps

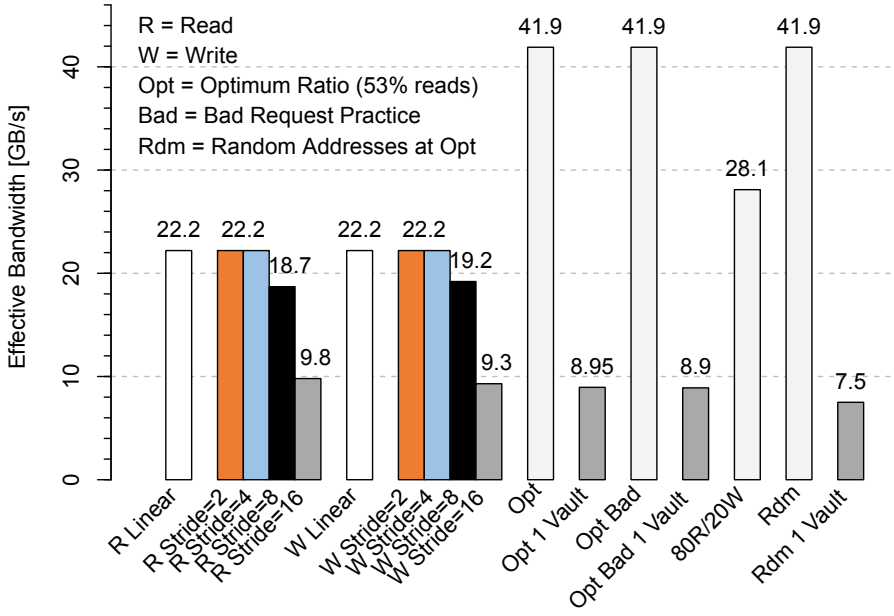


Fig. 3.14 Effective bandwidth for different access patterns with 128 Byte requests at 12.5 Gbps

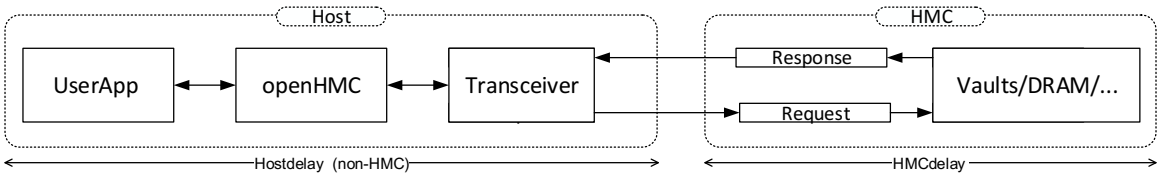


Fig. 3.15 Host to HMC read latency contributors

Table 3.6 Host-sided read latency contributors

Type / Delay	Cycles	at 10 Gbps	at 12.5 Gbps
User application	2	6.4 ns	5.12 ns
openHMC	29	92.8 ns	74.24 ns
SerDes	19	60.8 ns	48.64 ns
Total non-HMC	50	160 ns	128 ns

10 Gbps: 312.5 MHz FPGA clock (t_{cycle} = 3.2 ns)
12.5 Gbps: 390.625 MHz FPGA clock (t_{cycle} = 2.56 ns)

3.3.5 Latency

The latency of individual requests (i.e. the latency of a randomly selected request in a given request stream) for HMC is higher than for a transactional memory interface such as DDR. Several contributors to this latency can be identified as depicted in Figure 3.15. The delays for the user application and the openHMC controller are well known. The SerDes internal loopback mode of the FPGA was run to quantify the delay introduced through serialization and deserialization. It is assumed that the transmission line is not contributing noticeably. The actual HMC read delay can be estimated by subtracting all known delays from the overall latency. Table 3.6 summarizes the results for the individual contributors in the test design at 10 Gbps and 12.5 Gbps. It can be seen that the overall latency can be significantly reduced by increasing the FPGA logic frequency. To provide an application-near scenario the overall read request latency was measured, starting from the point where a packet is created in the user application until the corresponding response is received there. Figure 3.16 and Figure 3.17 plot the latency over a read ratio sweep for 128 Byte requests for linear addressing. Each ratio was applied for 20 seconds and the best, worst, and average latencies were measured for randomly selected individual requests

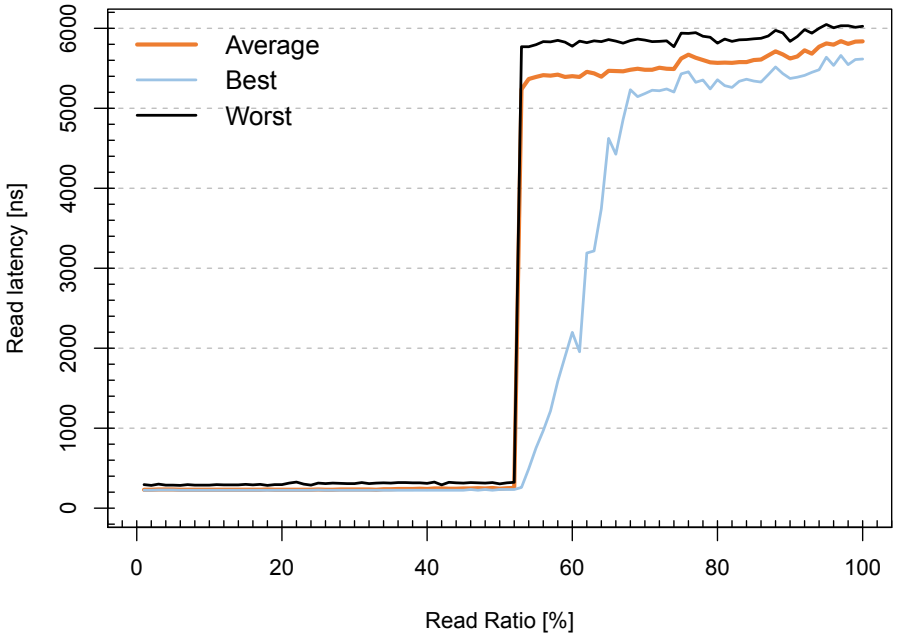


Fig. 3.16 Host to HMC read latency at 10 Gbps ($t_{cycle} = 3.2$ ns)

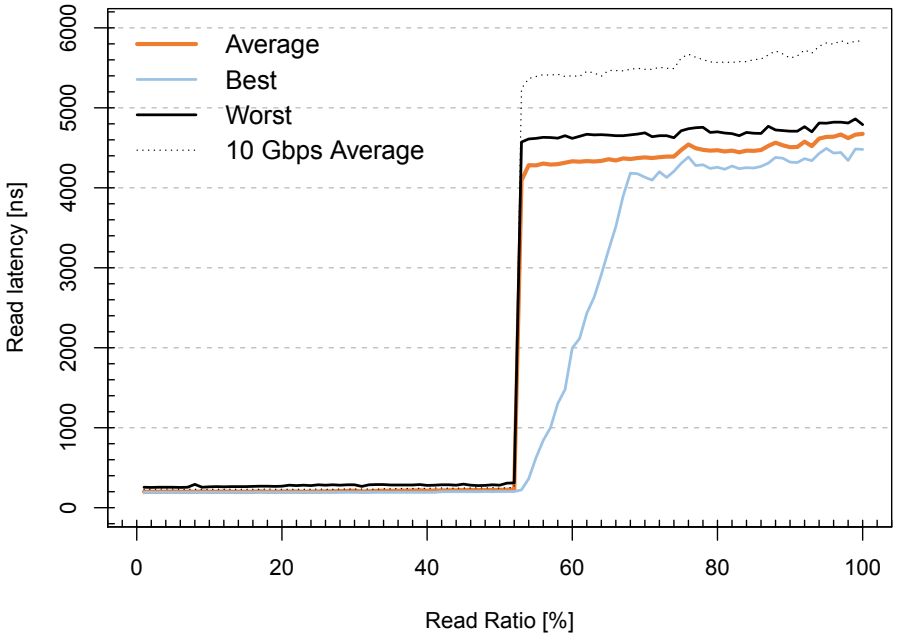


Fig. 3.17 Host to HMC read latency at 12.5 Gbps ($t_{cycle} = 2.56$ ns)

in the access stream. It can be seen that the initial read latency starts out with an average of 224 ns (70 FPGA cycles) for 10 Gbps and 192 ns (75 FPGA cycles) for 12.5 Gbps, respectively. The latency then remains stable until the optimum ratio threshold is reached. At this point, more reads are requested than the HMC and in particular the response link can supply. The read latency continues to increase when more reads are sent and goes up to several microseconds. This is because the HMC input buffer runs full with unanswered requests and the response link is in bandwidth saturation. This prevents the host to continue. The disparities between the best and worst case latencies originate from corner cases where a corresponding read request enters the openHMC controller right at the time that traffic is throttled. The request therefore remains in buffers waiting to be transmitted, while this waiting time accounts for the overall latency.

In summary it becomes clear that the single access latency gets much worse when either of the link directions saturates. A low latency host design including SerDes, host controller, and user application in combination with well-balanced access patterns are the key elements to lower the HMC access latency. Increasing host (FPGA) operating frequencies or HMC lane speeds are additional options. For the given test environment, however, increasing the link speed to 15 Gbps was not an option because it could not be implemented in the target FPGA.

3.3.6 Atomic Operations

The HMC protocol defines packet types for atomic operations that will be executed by the HMC logic layer, eliminating the need for expensive read-modify-write cycles on the host. The two available commands add either an 8 Byte value to a 16 Byte memory operand (16-Byte immediate add) or two 4 Byte values to two 8 Byte memory operands (dual 8-Byte immediate add). Each add operation is referred to as an *update*. Figure 3.18 summarizes the maximum updates per second for accessing a single address, a single vault, and up to all available vaults. This is represented by the corresponding access stride, where stride=16 accesses only 1 vault with HMC standard address-mapping. It can be seen that the maximum number of updates per second increases proportionally with the number of accessed vaults and inverse with the stride size for both types of atomics. Since the actual packet throughput remains the same, dual 8-Byte add immediate commands can update as twice as many values compared to 16-Byte adds. Figure 3.19 points out that increasing the lane speed

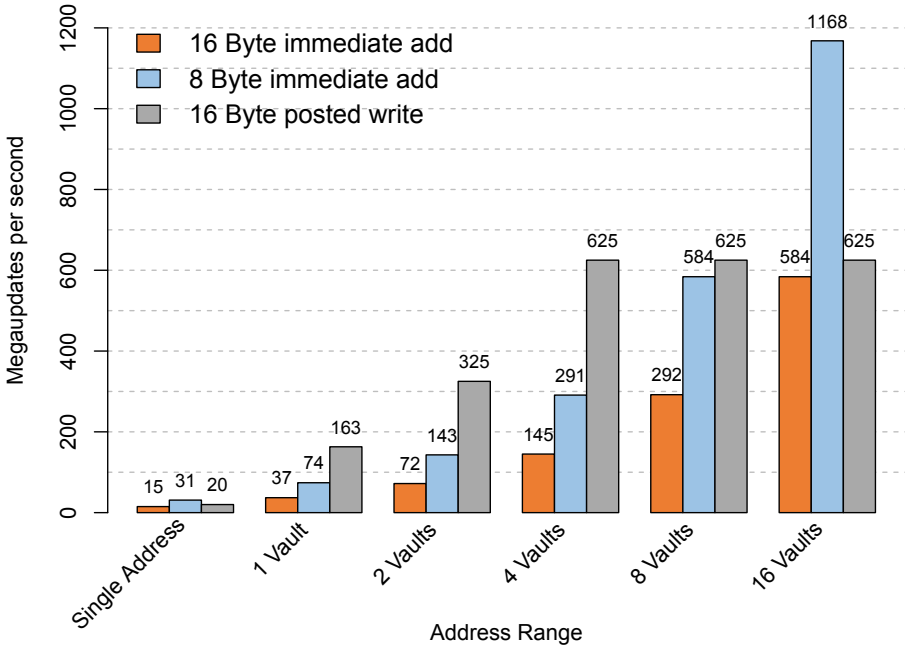


Fig. 3.18 Megaupdates/second versus address range at 10 Gbps

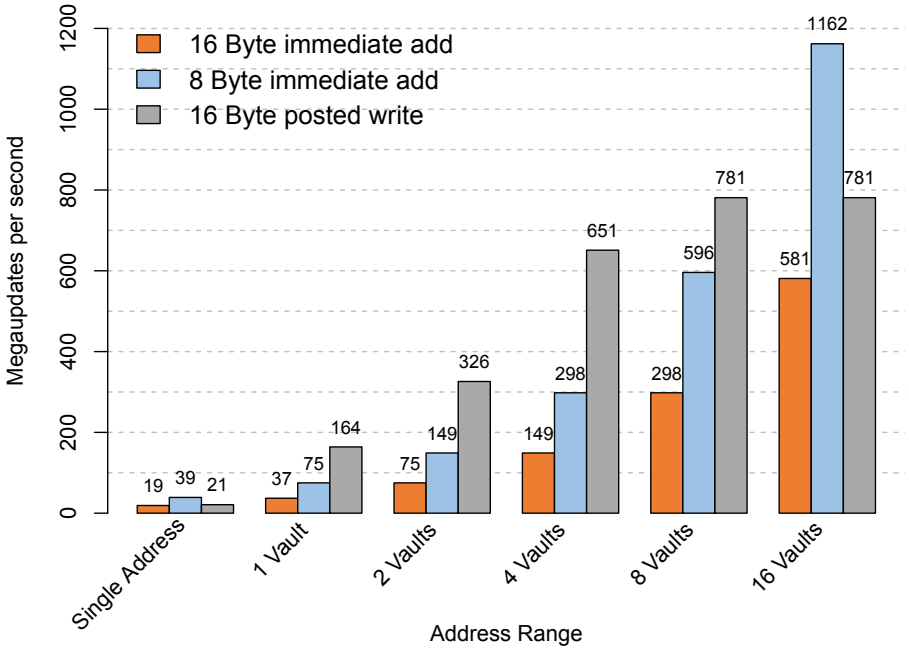


Fig. 3.19 Megaupdates/second versus address range at 12.5 Gbps

does not improve the maximum number of atomic operations since the HMC internal operating frequency is maintained.

The key observation in these plots is that increasing the number of accessed vaults has a positive impact on the total number of updates per second. This is in contrast to regular read/write requests that already saturate with less vaults. The positive effect of accessing more vaults concurrently, however, would also apply for regular reading and writing when more than one link was used.

3.3.7 Power Consumption and Energy Efficiency

The HMC power consumption was measured for the workloads presented in Figure 3.13 and Figure 3.14 using the test setup shown in Figure 3.9. Power was measured via the voltage drop over high-precision resistors. The following results represent experimental measurements at best efforts and are furthermore subject to parasitic effects (e.g. efficiency of the power source and other components) and deviation (e.g. temperature, measurement error).

Figure 3.20 and Figure 3.21 plot the HMC power consumption at 10 Gbps and 12.5 Gbps. The values for HMC power-on/reset and idle states are included as a reference. It can be seen that static and idle power make up a major fraction of the overall consumption. While a link in idle already consumes about 5 watts, actual traffic does not excessively contribute to the overall power footprint. One expected observation is that dynamic power consumption increases as more bandwidth is requested. The main contributors here are the sources for the DRAM and the logic core. Figure 3.22 and Figure 3.23 show the measured power efficiency in [pJ/bit] for the individual workloads at 10 Gbps and 12.5 Gbps. The efficiency is calculated as the power consumption in [Watt] divided by the effective bandwidth. The figures point out an idle power consumption (i.e. after the link has trained) of 5.1 Watt and the best energy efficiency with 23.2 pJ/bit at the optimum read/write ratio for a 10 Gbps link. Similarly, the idle power consumed for 12.5 Gbps is 5.6 Watt and the best efficiency was measured with 21.7 pJ/bit. All efficiencies are relative to the effective delivered bandwidth. There are no values provided for reset, idle, and sleep as there is no data transmitted at that time. Accessing random addresses does not affect power efficiency except when bank conflicts occur which lower the effective bandwidth. Furthermore, there is no difference between properly ordered request streams and the bad request practice

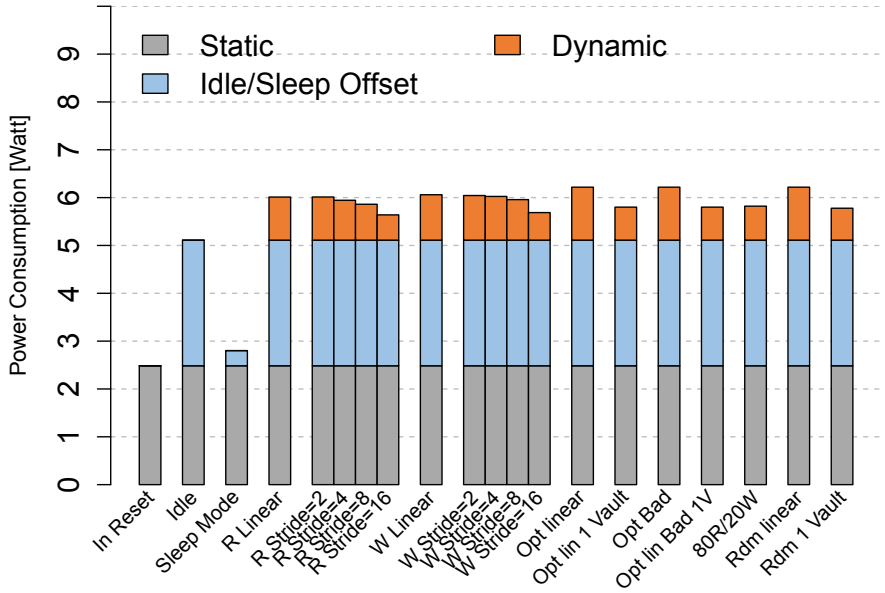


Fig. 3.20 HMC power consumption for various workloads at 10 Gbps (lower=better)

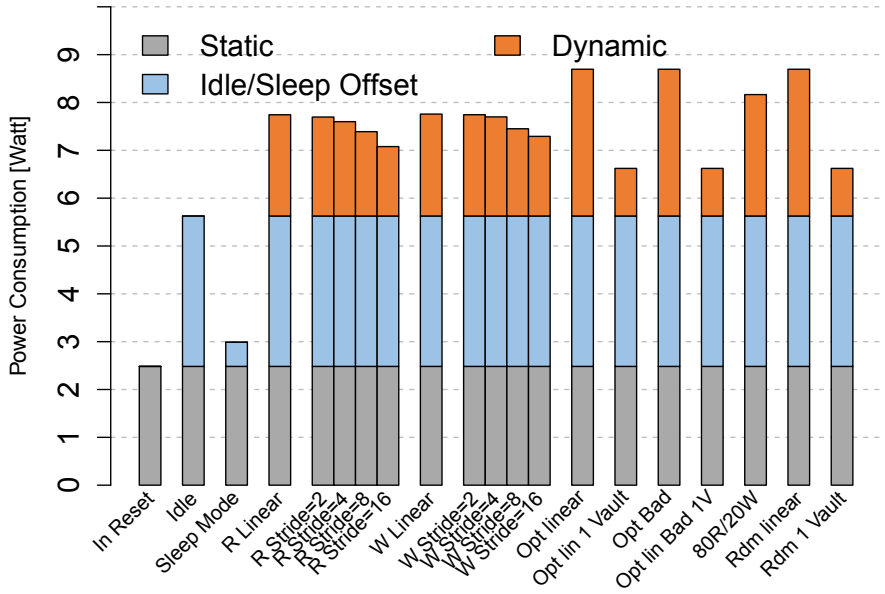


Fig. 3.21 HMC power consumption for various workloads at 12.5 Gbps (lower=better)

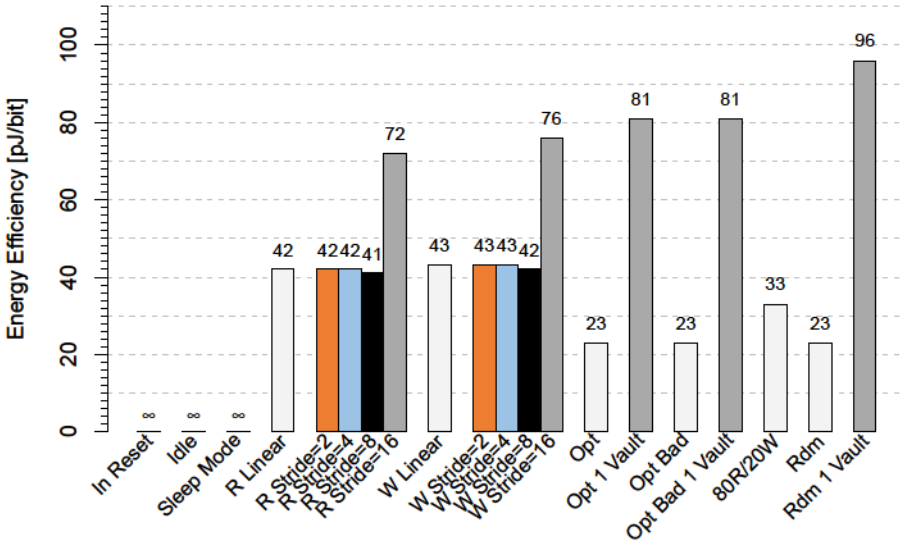


Fig. 3.22 HMC energy efficiency for various workloads at 10 Gbps (lower=better)

(for 100 requests) introduced earlier. As mentioned before it would require very long streams of disordered accesses to see an effect here.

In general, both plots point out that the power efficiency improves (i.e. pJ/bit drops) when the link is kept busy. In contrast, static power consumption dominates for inefficient link utilization and the efficiency is reduced. It is expected that increasing the number of active links and their lane speeds will have a positive effect on energy efficiency as static device power is a major contributor to the overall consumption.

The HMC sleep mode can be entered to reduce power consumption when the link is in idle to save about 45 % at 10 Gbps and 49 % at 12.5 Gbps. However, it must be noted that entering and exiting sleep mode takes time and requires an additional link initialization sequence.

3.3.8 Summary Performance Evaluation

This section provided HMC in-system measurements for bandwidth, latency, computation offloading using atomic operations, and energy efficiency for a single, full-width (16x) HMC link at 10 Gbps and 12.5 Gbps. The key takeaways can be summarized as follows:

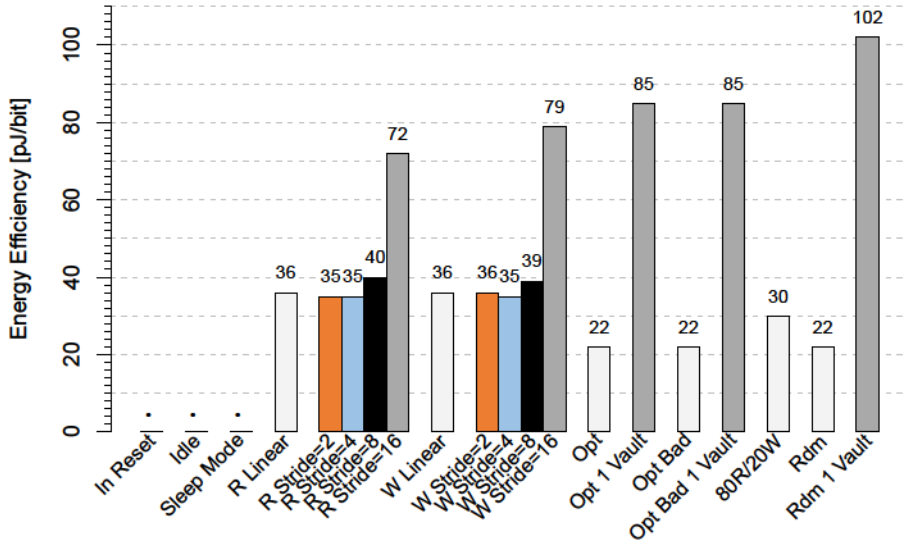


Fig. 3.23 HMC energy efficiency for various workloads at 12.5 Gbps (lower=better)

- A proper understanding of the HMC architecture is mandatory in order to optimize the overall performance. As long as requests access more vaults than actually required to saturate the link bandwidth, the limiting factor is the link itself. Link performance, however, is mainly dependent on the corresponding host controller. A low latency optimized controller is crucial to avoid any flow control drawbacks through the circulation time of packet pointers and tokens.
- Read latency is heavily affected by access patterns and imbalanced link utilization should be avoided whenever possible.
- The host portion of the request/response loop strongly influences the overall latencies. The results, however, reflect an FPGA host implementation. ASICs with higher clock speeds and low latency SerDes implementations would significantly reduce these numbers.
- Offloading computation using atomic operations can be a useful feature to eliminate computation and communication overhead for add operations. More advanced arithmetical and logic functions (such as announced with the HMC specification 2.0) will be required to provide a real benefit for most operations. Complex operations, however, will still remain a processor task and therefore require data movement.

- The results for energy efficiency prove that stacked memories and in particular HMC are capable to reduce the power penalty for accessing memory. Additional experiments with more links and higher lane speeds are required to ultimately identify its full potential.

In conclusion it becomes clear that HMC indeed provides the performance it claims. It furthermore contributes to meet the power and energy requirements for future systems. 3D integration of CMOS logic and processing elements will continue to gain importance and memory makers will hopefully integrate advanced offloading capabilities along with the memory device in the future.

3.4 HMC Summary

This chapter introduced the Hybrid Memory Cube and highlighted its most valuable characteristics. Its abstracted processor interface not only forces application developers to rethink how memory is used, but also requires a corresponding host controller.

openHMC has been presented as a no-cost alternative to other, commercially available host controllers. It was shown that openHMC outperforms at least one comparable host controller in terms of resource efficiency and flexibility, and at the same time maintains the best link performance. Experiments showed that an ASIC implementation of the same design can reach up to 1 GHz with a current process node.

A test setup comprising a 2 GB HMC and an FPGA was created to qualify the HMC performance and power efficiency. It became clear that access patterns have major influence on latency and bandwidth and also affect efficiency. However, if the loads on the link are well-balanced, HMC can provide a powerful, energy efficient, and dense memory alternative for many applications. Unfortunately, HMCs very limited capacity of 2 GB currently limits its use for most applications. Although the next-generation HMC devices were specified they have been taken off the roadmap and it remains to be seen if the capacity of current devices will increase considerably.

HMC and the openHMC host controller are essential building blocks of the Network Attached Memory which will be presented in the next chapter.

CHAPTER 4

Network Attached Memory

This chapter introduces Network Attached Memory (NAM)[1], a novel and standalone component with EXTOLL network interfaces. It provides access to a 2 GB HMC as shared memory resource combined with tightly coupled processing units implemented in an FPGA. As processing takes place in the FPGA and not the HMC memory itself, the NAM can be categorized as Near-Data Computing (NDC) device and not a true PIM architecture. The idea for the NAM originated from the desire to introduce a network device with fast memory and processing capabilities in order to reduce network traffic and speed-up collective operations.

The NAM is first used in the DEEP-ER (Dynamical Exascale Entry Platform - Extended Reach) project where it can be connected to any EXTOLL NIC to provide system-wide, high-performance DRAM access as an additional level in the memory hierarchy. Scalability is preserved as the memory capacity and network bandwidth linearly increase with the number of NAMs in the system. The first particular use case is to improve the performance of the DEEP-ER resiliency features. The NAM therefore implements a Checkpoint/Restart (CR) mechanism to speed-up the creation and reconstruction of parity checkpoints. The decision to use the HMC memory interface is in particular beneficial as it optimally suits the sequential access patterns of reading and writing large checkpoint files.

[1] The NAM concept has been prominently presented in various articles [13, 14] and as peer-reviewed conference poster [15].

The following sections provide background information on DEEP-ER and the EXTOLL network technology. The NAM prototype is presented and the functional units implemented in the FPGA are described in detail. A theoretical performance analysis will support characterization of the measurements conducted in the next chapter. Finally, FPGA implementation results and the required software components to actually use the NAM are presented.

4.1 DEEP-ER Project

DEEP-ER is a European Commission funded project under the Seventh Framework Programme (FP7/2007-2013). It addresses I/O performance and resiliency as two important challenges in building an Exascale-ready architecture. Both problems correlate since I/O performance also affects resiliency throughput. Within its predecessor DEEP, an innovative cluster-booster architecture was developed. While the cluster part is based on commodity Intel Xeon processors to execute complex, low to medium scalable code, the booster is equipped with Intel Xeon Phi accelerators for compute-intensive tasks. DEEP-ER extends this approach with upgraded components linked via the high-performance interconnection network EXTOLL. In addition, to satisfy the increasing demands to I/O performance, DEEP-ER attaches state of the art non-volatile memory and NAMs. Figure 4.1 outlines the system architecture as a high level diagram.

To establish a running hardware platform as early as possible in the project the DEEP-ER team created the Software Development Vehicle (SDV). It consists of 16 high-end Intel Xeon processor nodes and 3 file servers on the cluster side, and 8 Intel Xeon Phi accelerators as booster part. The early availability of the SDV helped software developers to familiarize themselves with the new components, especially the NAM. It was used to run all kind of application benchmarks including those for the NAM. The final DEEP-ER prototype foresees to upgrade the booster part for a total of 72 Xeon Phi accelerators. Although the project officially ended in March 2017, at the time of writing work to establish the final prototype is ongoing.

4.2 Background: EXTOLL

EXTOLL [47, 48, 49] is a high-performance interconnection network developed by the EXTOLL GmbH, a spin-off company of the Ruprecht-Karls University Heidelberg. Its

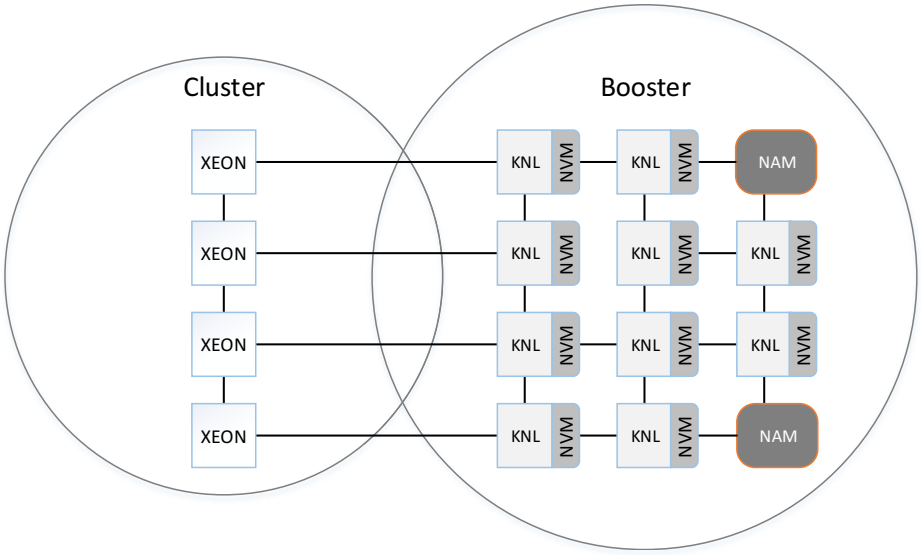


Fig. 4.1 DEEP-ER System Overview: The cluster part is based on Intel Xeon processors. The booster consists of Intel KNL nodes with one NIC and NVMe device each. Two NAMs are attached to available EXTOLL links

switch-less (i.e. the switch is integrated with the NIC) architecture removes the need for external switches and allows to create a variety of network topologies including mesh and 3D torus. Hence the network scales linearly with the system size. The EXTOLL ASIC named Tourmalet (Figure 4.2) comes with a PCIe Gen3 x16 host interface, 6 independent network links (+1 optional), the network switching architecture, and three different functional units used to exchange data between NICs.

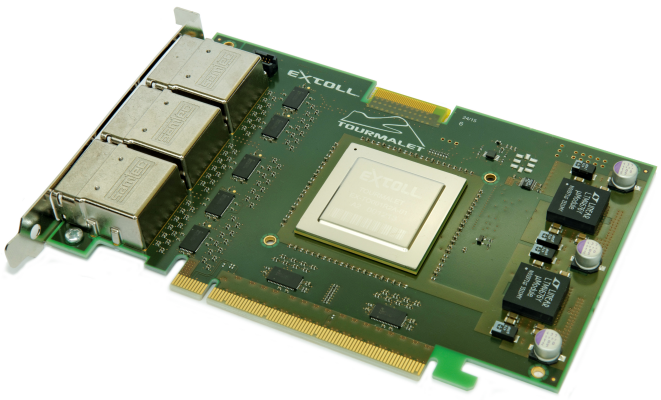


Fig. 4.2 EXTOLL Tourmalet ASIC. Image courtesy: EXTOLL

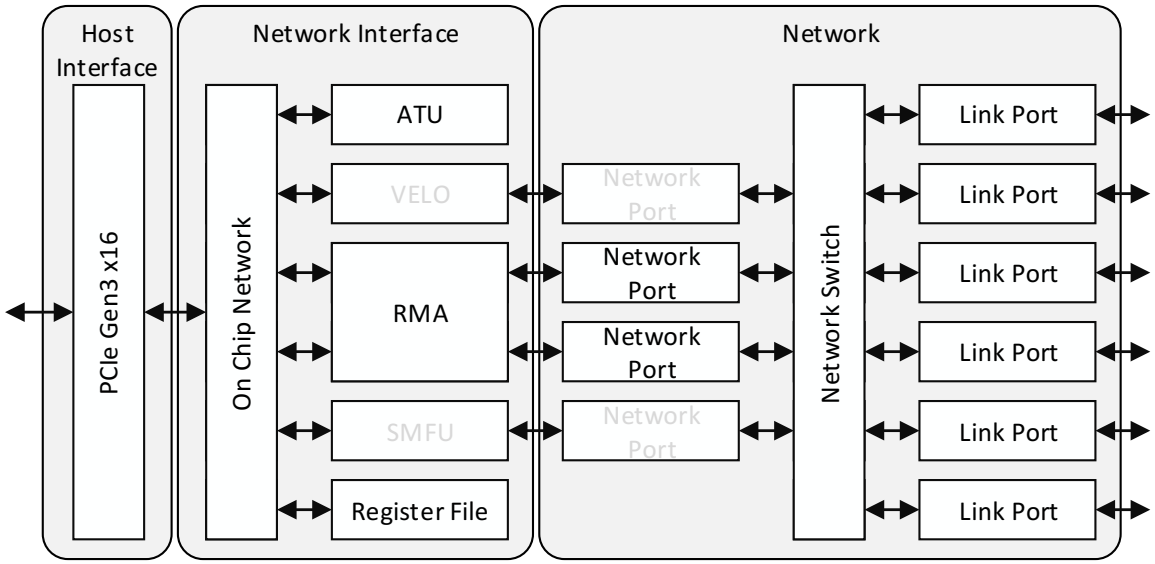


Fig. 4.3 EXTOLL Tourmalet ASIC Block Diagram

4.2.1 Functional Units and Link Performance

Figure 4.3 shows a block diagram of the Tourmalet ASIC. The PCIe Gen3 x16 interface connects a host processor. The three functional units for data transport (RMA, VELO, SMFU) are connected via the network crossbar switch to any of the six network links and via an additional on-chip network to PCIe. Out of these three units, the RMA (Remote Memory Access or Remote Memory Architecture) [101] has been identified as best candidate to communicate with the NAM, which in turn needs to implement a compliant unit. RMA is a throughput oriented unit designed for middle to large message sizes. Data is transferred and received via PUT and GET transactions and data transport is offloaded via a DMA engine. EXTOLL furthermore provides a low-overhead notification mechanism to inform a process whether data has been sent, requested data has arrived, or to inform a remote process that a PUT or GET operation has completed. The set of functional units is complemented by a Register File (RF) that can be accessed from local or remote and an Address Translation Unit (ATU).

In terms of link performance, each of the six EXTOLL network links operates on 12 lanes with a maximum of 8.4 Gbps per lane. Note that EXTOLL links operate in full-duplex mode, i.e. data can be transmitted and received simultaneously, doubling the lane count per link to 24. The following sections consider unidirectional operation and assume that bidirectional traffic results in approximately twice the bandwidth. The total raw bandwidth per link and direction is $12 \cdot 8.4 = 100.8$ Gbps $= 12.6$ GB/s.

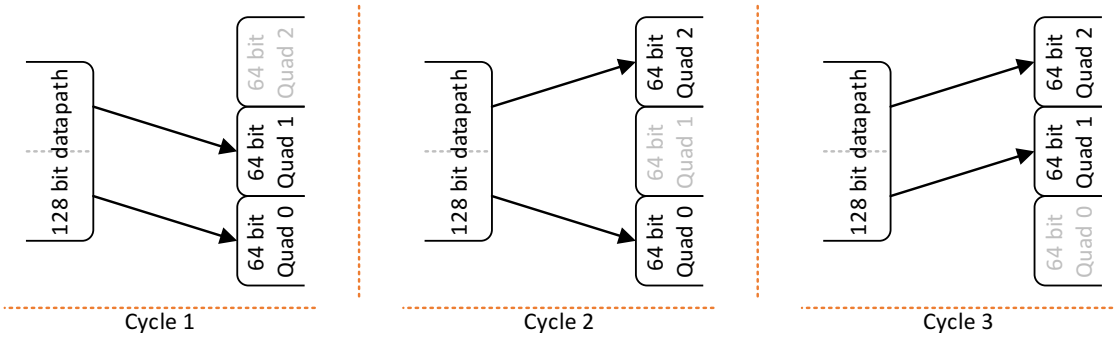


Fig. 4.4 The EXTOLL Link gearbox converts from the 128 bit datapath to the 192 bit link interface

Due to an 8b/10b coding scheme only 80 % of the link bandwidth is useful. The maximum unidirectional link bandwidth is therefore:

$$BW_{LINK} = 100.8 \text{ Gbps} \cdot \frac{8\text{b}}{10\text{b}} = 80.64 \text{ Gbps} = 10.08 \text{ GB/s} \tag{4.1}$$

The links are downward compatible to support smaller links (8 lanes / 4 lanes) and lower link speeds (4.2 Gbps / 2.1 Gbps). When the NAM project started, EXTOLL Link speeds were announced with 2.5/5/10 Gbps. For technical reasons the link speeds had to be decreased, which influenced the reference clock selection on the NAM prototype. It will be discussed in Section 4.3.

All EXTOLL functional units including the RMA operate on a 128 bit datapath at 630 MHz which matches BW_{LINK}:

$$BW_{RMA} = 128 \text{ bit} \cdot 630 \text{ MHz} = 80.64 \text{ Gbps} = 10.08 \text{ GB/s} \tag{4.2}$$

Note that the equation above is only valid for a link between two EXTOLL ASICs. Although at this time the maximum RMA bandwidth with a NAM as link partner is not calculated it is necessary to understand how data is passed from the EXTOLL functional units to the link.

A 12x EXTOLL Link is subdivided into three quads with four lanes each, and every lane takes 16 bit parallel data at a time. Hence, the width of the parallel data input to the link is $12 \cdot 16$ bit $= 192$ bit. A gearbox is used to translate this interface to the 128 bit datapath of the functional units in a 3-stage iterative process as depicted in Figure 4.4. It can be seen that six 64 bit cells are processed within each iteration. It will be shown

that this gearbox has a negative effect on the bandwidth when communicating with the NAM.

4.2.2 From Software to Network Transactions

Every PUT or GET transaction carried out by the RMA is initiated by a user program. The software places a descriptor into one of the descriptor queues of the EXTOLL device. These descriptors contain information such as the destination node and process, the amount of payload to be written or read, and where this payload shall be read from or written to. For PUT operations, the source address is the start location of the payload in the local memory and is either a virtual or physical address. In case of a virtual address the ATU is requested to translate it to a physical one. Without involving the host processor, the EXTOLL NIC fetches the payload via DMA. The data is then packed into network packets and transmitted by the local RMA requester unit. The transaction is directed to the RMA completer unit of the destination node which forwards the data to its local target memory location (Figure 4.5a). GET operations on the other hand will fetch data from a remote memory location and transfer it to the local memory of the requesting node via GET Response transactions. In this case the transaction is requested by the local RMA requester with the remote RMA responder as turnaround unit. As the response returns to the local node it is eventually processed by the RMA completer (Figure 4.5b).

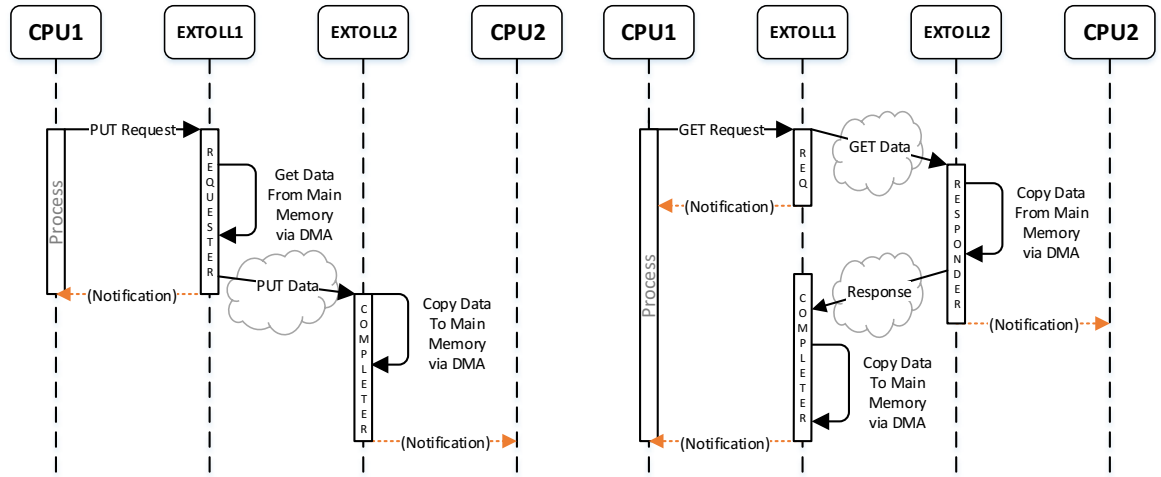
The maximum amount of data movement initiated by a single software descriptor is 8 MB. Hence, to accommodate larger data transfers, multiple transactions must be triggered by placing additional software descriptors.

A third command, PUT IMMEDIATE, is provided for small data transfers (72 bit) without involving the local DMA engine. The payload is already embedded in the software descriptor in this case and it can be useful to e.g. access a remote RF.

It must be noted that the EXTOLL RMA supports additional commands. They are neither relevant for this work nor supported by the NAM.

4.2.3 Notification Mechanism

EXTOLL provides an optional notification mechanism as depicted in Figure 4.5. It can be used to inform processes of the progress or completion of transactions. For PUT transactions, notifications may be generated at the local nodes of the respective



(a) PUTs may generate up to two notifications (local requester, remote completer)

(b) GET operations may generate up to three notifications (local requester, local completer, remote responder)

Fig. 4.5 EXTOLL PUT/GET operations and notification mechanism

RMA units involved in the process, i.e. the local RMA requester or remote RMA completer. GET operations additionally involve a remote responder unit which may generate notifications likewise.

4.2.4 Network Protocol

The EXTOLL network protocol operates on cells as transmission units. Each cell is 64 bit (8 Byte) in size and a network packet consists of multiple cells. The maximum size of one packet is limited by the network Maximum Transmission Unit (MTU), which is fixed to 512 Byte or $512/8 = 64$ cells in this work. Every packet is furthermore preceded by a network descriptor which accounts for the MTU: 16 Byte for PUT, PUT IMMEDIATE, and GET Response commands and 24 Byte for GETs. GET operations, however, do not carry any payload and PUT IMMEDIATE commands only transmit very few data. Both transaction types are not subject to the MTU. PUT requests and GET responses on the other hand may carry $512 - 16 = 496$ Byte payload per packet at most. This packet size limitation introduced by the network MTU also implies that a single software descriptor with a maximum size of 8 MB may trigger multiple network packets. The actual number is determined by splitting the software requested size into 496 Byte network packets. Within each subsequent packet the initial destination target

address as provided by the software descriptor is incremented by the packet size of 496 Byte accordingly.

Beyond the RMA the EXTOLL network protocol frames each packet with two extra cells. These cells (Start Of Packet (SOP) and End Of Packet (EOP)) contain additional network information and ensure packet integrity implemented as a Cyclic Redundancy Check (CRC) check, which will trigger a link retry mechanism if a corrupted packet is received. The overhead through SOP and EOP adds another 16 Byte for a total packet size of $512+16=528$ Byte. According to the information above the RMA efficiency can be calculated with:

$$EFF_{RMA} = \frac{\text{Data Bytes}}{\text{Total Bytes}} = \frac{496}{512+16} = 93.4\ \% \tag{4.3}$$

which gives the maximum effective RMA bandwidth of:

$$\begin{gathered} BW\_EFF_{RMA} = BW_{RMA} \cdot EFF_{RMA} = 80.64\ \text{Gbps} \cdot 93.4\ \% \\ BW\_EFF_{RMA} = 75.75\ \text{Gbps} = 9.47\ \text{GB/s} \end{gathered} \tag{4.4}$$

4.2.5 Link Flow Control

Flow control between two EXTOLL links is handled via credits which reflect the local retry buffer and remote input buffer space of the respective link partner likewise. A 496 Byte RMA packet consumes four credits in total, one per 128 Byte payload. After the packet has passed the remote input buffer these credits will be returned in dedicated flow control cells, freeing up the corresponding space in the local retry buffer.

The buffers were designed to accommodate payload for up to 128 credits, which are shared among ten Virtual Channels (VCs). VCs can be used to create individual and unrelated streams of traffic to prioritize certain types of traffic, often used to handle routing congestion. Some of these VCs are dedicated to specific traffic classes such as broadcasts. Every single VC gets assigned eight credits for exclusive use and the remaining 48 are shared among these on a first-come-first-serve basis. Out of the ten channels, four can be used for regular read/write commands. If packets were distributed evenly on these four channels and no other traffic was flowing, a total of $4 \cdot 8 + 48 = 80$ credits would be available for reading and writing. For a single VC, however, the maximum count is 56 (8 exclusive + 48 shared).

Throttling of the link performance occurs whenever no more or too few credits are available to transmit the next packet. This is the case when credits are consumed faster than they are returned by the remote link partner or simply shared credits were consumed by other VCs. In either event the flow control loop is too slow. A very similar difficulty has been identified with the HMC token return loop time violation in Section 3.1.6.2. A later section in this chapter will show how credits and the flow control loop affect the NAM access performance, how it is currently handled in software, and what needs to be done to improve the situation.

4.2.6 EMP: Network Discovery and Setup

EXTOLL devices can be connected in many different ways to create commonly used mesh and torus or individual non-standard topologies. In any case, all network device routing tables initially must be set up. The EXTOLL Management Program (EMP) supports two types of network setup modes: discovery and topology file based configuration. In discovery mode all EXTOLL links that show an active connection are scanned and a topology is automatically created. Topology file based configuration on the other hand may be used to verify that all devices are properly connected and the desired topology was successfully created. In either of the network setup modes, EMP assigns unique identifiers (Node ID) to every EXTOLL NIC and calculates and sets the routing table entries according to the desired routing scheme. Eventually all nodes are marked as active which unlocks the network for software usage.

4.3 NAM Hardware

The following section presents the NAM hardware prototype and functional modules implemented in the FPGA. In order to estimate performance as early as possible in the design process and to avoid unexpected bottlenecks, design decisions and their potential impact on the achievable performance are evaluated.

4.3.1 Requirements

The first step in developing a new hardware device is to define its requirements. A clear view of the physical interfaces and understanding their impact on the FPGA design is mandatory and allows to develop a prototype early in the design phase. This

lowers the risk of delays due to potential PCB manufacturing and bring-up issues. The following physical and logical requirements for the NAM have been identified based on the DEEP-ER use case, the available components such as HMC, the FPGA, and the EXTOLL interconnect, and the physical size and form factor.

4.3.1.1 Components and Connectors

EXTOLL A Samtec HDI-6 connector as physical interface to connect up to two EXTOLL NICs with 12 lanes per link.

PCIe The PCIe edge card connector is used to power the NAM board and allows to easily integrate it with commodity systems. The connector could also be used to establish host connectivity for management and/or data transport.

HMC-1 One or more HMC links, desirably in a configuration that matches or outperforms the total EXTOLL Link bandwidth (for available HMC link configurations see Section 3.1.3).

HMC-2 A high-speed connector that interfaces one additional HMC link provides the ability to chain HMCs to increase the memory capacity.

RAS Advanced RAS (Reliability, Availability and Serviceability) features require a physical programming and debug interface.

FPGA A suitable FPGA must provide enough resources and I/O capability to implement processing elements and modules for the physical interfaces described above. Especially the total high-speed transceiver count to accommodate all types of serial interfaces (PCIe, EXTOLL, HMC) is essential.

4.3.1.2 FPGA Design Functional Units

EXTOLL Link FPGA implementation The original EXTOLL Link is implemented with a 128 bit datapath in the ASIC Tourmalet at 630 MHz. The source code for this link was provided by EXTOLL. In order to maintain link throughput while keeping the clock frequency within a reasonable region for an FPGA implementation, the link must be extended to support a wider datapath. The implementation must be able to support the maximum EXTOLL Link width and speed.

RMA compatible unit The NAM must be able to communicate with the native EXTOLL RMA unit. A compatible unit implements the required subset of RMA functions and accounts for a wider datapath.

HMC host controller The development of the HMC host controller openHMC was already discussed in Section 3.2.

RMA to HMC (and reverse) protocol converter The most basic requirement, reading and writing to the NAM, demands a module that converts EXTOLL RMA network packets to HMC transactions and vice versa.

RAS One module to provide remote register file configuration and monitoring over EXTOLL. A second module grants RAS access over an external debug connector.

CR The CR unit required to carry out the DEEP-ER resiliency features. It will be discussed in Section 4.5.

4.3.2 Prototype 'Aspin-v2'

Figure 4.6 depicts the NAM hardware prototype Aspin-v2 developed as a standard height PCIe form factor PCB. The Xilinx Virtex 7 FPGA utilizes 16 lanes at 10 Gbps to connect a 2 GB HMC. Additional 16 lanes are connected to the 16x PCIe edge card connector. Although the maximum link width of the Virtex 7 PCIe hard-IP[2] blocks is 8x, the eight additional lanes can be useful if the connector is used proprietary. It is also possible not to use the hard-IP block to set up a 16x PCIe link. In this case, however, PCIe Gen3 (8 Gbps per lane) will not meet timing in the FPGA, limiting the capability of the FPGA PCIe core to Gen2 (5 Gbps) or even Gen1 (2.5 Gbps)[3]. The set of high-speed connections to the FPGA is complemented by two 12x links on the HDI-6 connector used to connect EXTOLL NICs.

The total transceiver count of 56 (16 PCIe + 16 HMC + 24 EXTOLL) narrowed down the number of usable FPGAs from the Virtex 7 device family. Eventually the V7 690T as second smallest device with at least 56 transceivers as a trade-off between logic cells and cost was chosen.

[2] FPGAs typically provide several fixed (*hardened*) logical blocks that implement specific functions such as a PCIe endpoint/root-port complex. Hardened IP is superior to functions implemented with standard registers and LUTs regarding achievable performance.

[3] Besides the actual lane speeds, PCIe Gen3 uses an improved lane encoding which increases the effective bandwidth per lane to ≈98 % compared to 80 % in previous generations.

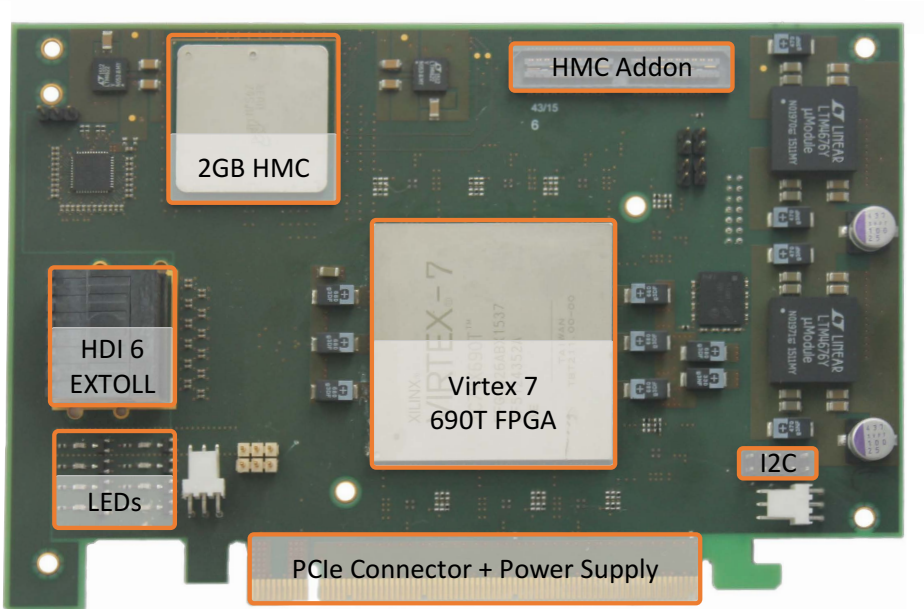


Fig. 4.6 NAM Prototype Board 'Aspin-v2'

A second HMC link is exposed to a dedicated connector which can be used to attach additional HMCs to increase the capacity (*chaining*, see Section 3.1.3). RAS features can be carried out through dedicated I2C (Inter-Integrated Circuit) and JTAG (Joint Test Action Group) connectors. A set of general purpose LEDs is free to use. Power is supplied via the PCIe connector and the required voltages are generated by on-board power regulators. A flash memory chip stores the FPGA configuration so that it does not need to be reprogrammed upon a power cycle.

FPGA and HMC are supplied by a single oscillator and clock distribution network. As stated in Section 4.2, EXTOLL Link lane speeds were reduced for technical reasons. This happened after the NAM prototype was already built. To maintain interoperability with the clocking infrastructure of the Xilinx GTH transceivers, the former 125 MHz shared reference clock was increased to 127.273 MHz. This option provides the least significant change in the clocking infrastructure and leads to a static multiplier F_{MULT}:

$$F_{MULT} = \frac{127.273\ \text{MHz}}{125\ \text{MHz}} = 1.018184 \tag{4.5}$$

The increased reference clock results in overclocking the HMC link as the HMC internally uses a fixed multiplier. The following paragraphs also describe the impact of this design change on other modules.

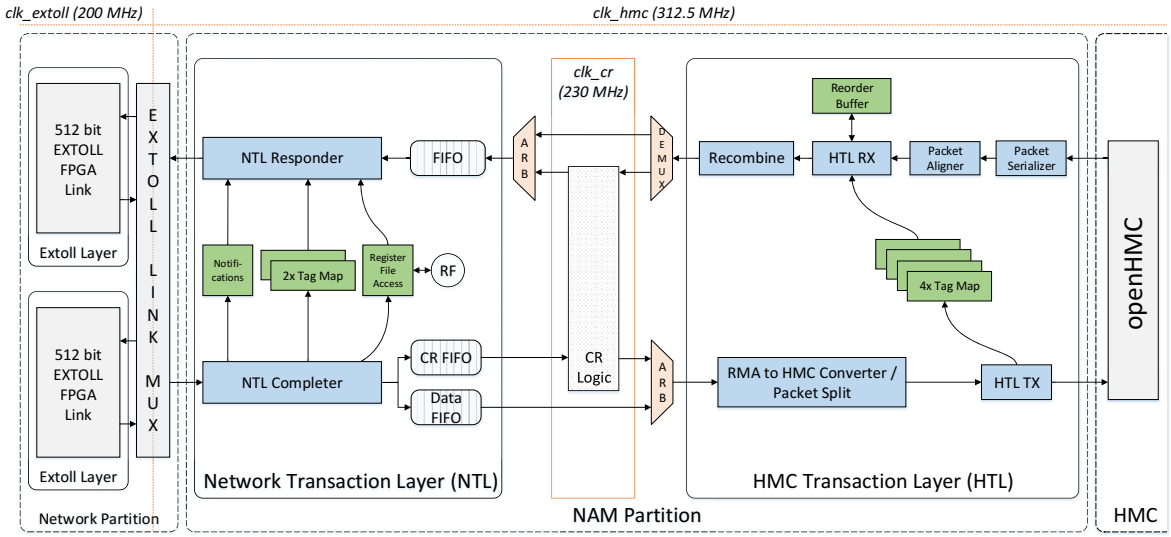


Fig. 4.7 NAM FPGA design block diagram: The design is partitioned into HMC, NAM, and EXTOLL functional layers

4.3.3 FPGA Design Partitions

The NAM FPGA design is depicted in Figure 4.7. It is divided into three main partitions: Network, NAM, and HMC. The network partition integrates two 512 bit EXTOLL FPGA links connected to the NAM logic via a Multiplexer (MUX). Note that this MUX does not implement any routing and will not forward packets from one link to another. Both links are therefore EXTOLL endpoints and only packets that target the NAM as final destination may be received. The NAM partition translates EXTOLL to HMC packets (and vice versa) and provides RF access to remote processes via EXTOLL RRA (Remote Register File Access). It also integrates the CR unit which will be described in Section 4.5. Finally, the HMC partition integrates the openHMC controller and an autonomous HMC configuration module.

Many of the modules also provide a set of registers. These are embedded in a hierarchy of Register Files and allow design control and monitoring at runtime, accessible via RRA or the physical I2C connector.

Figure 4.7 also identifies the three main clock domains. clk_hmc is a 318.1825 MHz clock derived from the HMC link configuration, based on a 312.5 MHz clock multiplied by F_{MULT} (see Section 4.3.3.1), so that the throughput of the 512 bit datapath matches the HMC link bandwidth. The openHMC specification states that a connected user application must operate at the frequency of clk_hmc or faster. To avoid additional clock domain crossings and as it is unlikely that the NAM logic will meet timing

constraints for even faster clocks it is also sourced from clk_hmc. The second main clock domain is clk_extoll which drives the logic of both EXTOLL links. Although there is no restriction on the frequency of this clock it will be shown that it has a major impact on performance. The third clock domain is clk_cr which drives all CR related parts of the design. Another clock domain crossing at this point became necessary as it turned out that the CR logic could not be implemented with a clock as fast as clk_hmc.

This section presents the individual design partitions and theoretically evaluates bandwidth characteristics based on design decisions. Several potential bottlenecks will be identified which will help to interpret the in-system measurements provided in Chapter 5. As a naming convention, packets traveling from the network to the NAM are referred to as requests while a response denotes the opposite direction respectively.

4.3.3.1 HMC Partition / openHMC

The HMC partition integrates the openHMC host controller with a 512 bit user interface and a full-width (16x), 10 Gbps HMC link. In fact, through overclocking, the actual speed per lane is 10 Gbps $\cdot F_{MULT} = 10.18184$ Gbps. Based on Equation (3.3) the resulting operating frequency clk_hmc is calculated with:

$$clk\_hmc = \frac{16 \text{ lanes} \cdot 10.18184 \text{ Gbps}}{512 \text{ bit} \cdot 10^6} = 318.1825 \text{ MHz} \tag{4.6}$$

Using the unidirectional HMC bandwidth of 17.7 GB/s (see Section 3.3.4) and the multiplier F_{MULT} the new HMC read or write bandwidth BW_{HMC} is:

$$BW_{HMC} = 17.7 \text{ GB/s} \cdot 1.018184 = 18.02 \text{ GB/s} \tag{4.7}$$

It is the theoretical peak bandwidth for 128 Byte HMC read or write packets with a sequential access pattern.

4.3.3.2 Network Partition / EXTOLL FPGA Link

The EXTOLL FPGA link has been derived from the native EXTOLL ASIC link implementation which is based on a 128 bit datapath. The nominal EXTOLL ASIC operating frequency is 630 MHz for a throughput of 128 bit $\cdot$ 630 MHz = 80.64 Gbps. To match the throughput at a reasonable frequency in the FPGA the datapath-width

had to be increased to 512 bit. Although a 256 bit datapath would have been a feasible choice as well it would not integrate seamlessly with the remaining NAM logic. The datapath-width here is dictated by the configuration of the openHMC controller and has been set to 512 bit. A unified datapath-width throughout all modules considerably simplifies logic design. The resulting minimum core clock frequency of the EXTOLL FPGA link to support the RMA bandwidth of 80.64 Gbps is calculated with:

$$clk\_extoll_{min} = \frac{\text{throughput}}{\text{datapath-width}} = \frac{80.64 \text{ Gbps}}{512 \text{ bit}} = 157.5 \text{ MHz} \tag{4.8}$$

At a first glance it seems sufficient to set clk_extoll to the minimum required frequency. However, the performance measurements conducted in Chapter 5 will reveal a correlation between clk_extoll and the overall NAM performance, with clk_extoll_{min} performing the worst. This behavior is associated with the EXTOLL network protocol flow control features to support a retransmission scheme when errors were detected on the serial link. Such link integrity features are common practice in serial link protocols.

The EXTOLL Link parameters including the size of the retry buffer were tailored for the ASIC and hence dimensioned to operate on a 128 bit datapath at a frequency of 630 MHz. The relatively low frequency of the 512 bit link implementation implies that packets and credits in the NAM are processed much slower than in the ASIC. The number of credits for reading from and writing to the NAM is fixed to a maximum of 80 using all four available Virtual Channels and 58 on one channel. For this fixed number of credits and if bandwidth throttling comes in, the only mitigation strategy is to increase the frequency of clk_extoll in the NAM. Increasing the frequency, however, significantly complicates placement, routing, and timing closure in the FPGA. Eventually the link logic was successfully implemented at 200 MHz with clean timing which lowered the negative impact of the issues mentioned above.

$$clk\_extoll = 200 \text{ MHz} \tag{4.9}$$

The difficulty with insufficient credits becomes even worse in response direction, with traffic flowing from the NAM to an EXTOLL ASIC. Due to an unintended limitation in the ASIC, the maximum credit count per Virtual Channel the NAM can use to send traffic is 31.

Gearbox, Alignment, and Flow Control

Apart from the operating frequency, the 512 bit link has additional negative side-effects on the maximum achievable bandwidth with an EXTOLL ASIC link partner. Section 4.2 introduced the gearbox that is used to pass data from a 128 bit functional unit to the 192 bit link in the EXTOLL ASIC. Similarly, the 512 bit link implements a gearbox that passes data from the 768 bit link layer (three quads with 256 bit parallel data each, 4 lanes per quad with 64 bit per lane) on the receiving side. The result is again a 3-stage iterative process. For the sake of design simplicity, it is required that packets start at a 512 bit / 64 Byte boundary so that the very first cell (SOP) is seen starting at bit position 0 in a parallel 512 bit cycle. It is the responsibility of the sending side to ensure that packets meet this requirement. Therefore, the EXTOLL ASIC gearbox will issue filler cells up to the next 64 Byte boundary whenever a packet including protocol overhead is not a multiple of 64 Byte.

The use of filler cells for packet alignment limits the effective RMA bandwidth in the EXTOLL ASIC. According to Section 4.2.4 the maximum RMA packet size is 528 Byte of which 496 Byte contain payload. The ASIC gearbox now appends additional filler cells up to the next 64 Byte boundary, which is 576 in this case. This leads to the RMA packet efficiency EFF_{RMA_PKT} of:

$$EFF_{RMA\_PKT} = \frac{\text{Data Bytes}}{\text{Total Bytes}} = \frac{496}{576} = 86.1\% \tag{4.10}$$

Packets also consume credits, four in total for a full-sized RMA-to-NAM write packet and one per read request. Likewise, a full-sized RMA GET Response will utilize four credits on the EXTOLL Link in the NAM. These credits must be returned by the remote link partner so that they eventually can be reused to transmit additional packets. Dedicated credit cells are generated and sent to the former source node.

The threshold for the number of credits at which a credit cell is generated is configurable and has been set to 10 credits on the NAM. This means the NAM will create a credit cell for every 2.5 full-sized RMA write requests, and for every 10 RMA reads it has received. These cells do not affect the request bandwidth for traffic flowing from EXTOLL to the NAM as credit cells travel in opposite direction. Every received credit cell, however, must be acknowledged by the local link[4]. This acknowledge is an eight Byte packet which is again subject to packet alignment boundaries and occupies a full 64 Byte

[4] Acknowledge cells may also carry credits that need to be returned and credit cells may implement acknowledge counter likewise.

of the transmission bandwidth. For writing, acknowledge cells add $\frac{64}{2.5} = 25.6$ Byte overhead per packet which results in an actual total bytecount of $576 + 25.6 = 601.6$ Byte per request. Given these results the actual RMA write efficiency for requests to the NAM can be derived with:

$$EFF_{RMA\_REQ} = \frac{\text{Data Bytes}}{\text{Total Bytes}} = \frac{496}{601.6} = 82.4\ \% \tag{4.11}$$

Using EFF_{RMA_REQ} the maximum effective link RMA request bandwidth $BW\_EFF_{RMA\_REQ}$ is:

$$\begin{gathered} BW\_EFF_{RMA\_REQ} = BW_{RMA} \cdot EFF_{RMA\_REQ} = 80.64\ \text{Gbps} \cdot 82.4\ \% \\ BW\_EFF_{RMA\_REQ} = 66.48\ \text{Gbps} = 8.31\ \text{GB/s} \end{gathered} \tag{4.12}$$

Read responses that return to the local EXTOLL device must also be acknowledged. To reduce the amount of overhead at this point the EXTOLL Link is able to pack several acknowledgments into a single cell. To approximate the performance, it is assumed that credits are also embedded with acknowledge cells traveling back to the NAM.

Read requests to the NAM, on the other hand, will generate a credit cell for every 10 request packets. This adds an average of $\frac{64}{10} = 6.4$ Byte overhead per packet in response direction caused by credit cells, and the same amount of overhead in request direction used for acknowledging these. Given this additional overhead the total bytecount is $576 + 6.4 = 582.4$ for a packet traveling from the NAM to an ASIC. Hence the NAM to ASIC response efficiency EFF_{RMA_RSP} is:

$$EFF_{RMA\_RSP} = \frac{\text{Data Bytes}}{\text{Total Bytes}} = \frac{496}{582.4} = 85.1\ \% \tag{4.13}$$

Using EFF_{RMA_RSP} the actual maximum effective link RMA response bandwidth $BW\_EFF_{RMA\_RSP}$ is:

$$\begin{gathered} BW\_EFF_{RMA\_RSP} = BW_{RMA} \cdot EFF_{RMA\_RSP} = 80.64\ \text{Gbps} \cdot 85.1\ \% \\ BW\_EFF_{RMA\_RSP} = 68.67\ \text{Gbps} = 8.58\ \text{GB/s} \end{gathered} \tag{4.14}$$

Note that communication between two EXTOLL ASICs must be aligned likewise, with reduced packet boundaries at 128 bit / 16 Byte. Therefore, no filler cells are applied for packets that come as a multiple of 16 Bytes such as the largest RMA packet and for all other packets the overhead of alignment is significantly lowered.

There are two ways to alleviate the impact of the gearbox. First, a smaller datapath would reduce the packet boundaries and alignment overhead. The decision for a 512 bit link, however, was made for good reason. It seamlessly integrates with the rest of the design. And second, a link layer design that operates on the datapath-width of the functional units or an integer multiple (e.g. 256 bit link and 128 bit functional unit) of it would significantly reduce the interface complexity.

4.3.3.3 Network Partition / EXTOLL Link MUX

The EXTOLL Link MUX connects both EXTOLL links to the NAM layer and acts as clock domain crossing from clk_extoll to the faster clk_hmc clock domain. The clock domain transition is realized with asynchronous buffers, one per link and direction. The actual switching between links is then performed with the speed of clk_hmc to eliminate the bandwidth of a single EXTOLL Link as bottleneck at this point. The theoretical link MUX bandwidth in both directions, request and response, is linked to the datapath-width, the operating frequency clk_hmc, and the RMA packet efficiency EFF_{RMA_PKT} (not the actual link RMA efficiency as flow control cells were removed already). It is calculated with:

$$\begin{aligned} BW\_EFF_{MUX} &= clk\_hmc \cdot \text{datapath-width} \cdot EFF_{RMA\_PKT} \\ BW\_EFF_{MUX} &= 318.1825\ \text{MHz} \cdot 512\ \text{bit} \cdot 86.1\ \% \\ BW\_EFF_{MUX} &= 140.2\ \text{Gbps} = 17.54\ \text{GB/s} \end{aligned} \tag{4.15}$$

In comparison the combined EXTOLL bandwidth that two links can deliver for requests is:

$$\begin{aligned} BW\_EFF_{RMA\_REQ\_TWO\_LINKS} &= 2 \cdot BW\_EFF_{RMA\_REQ} \\ BW\_EFF_{RMA\_REQ\_TWO\_LINKS} &= 2 \cdot 8.31\ \text{GB/s} = 16.62\ \text{GB/s} \end{aligned} \tag{4.16}$$

And for responses:

$$\begin{aligned} BW\_EFF_{RMA\_RSP\_TWO\_LINKS} &= 2 \cdot BW\_EFF_{RMA\_RSP} \\ BW\_EFF_{RMA\_RSP\_TWO\_LINKS} &= 2 \cdot 8.58\ \text{GB/s} = 17.16\ \text{GB/s} \end{aligned} \tag{4.17}$$

It can be seen that link multiplexing is good enough to keep up with the performance of both EXTOLL links in either direction. However, it will be shown that the aggregate bandwidth of two EXTOLL links outperforms the capabilities of the subsequent NAM logic units.

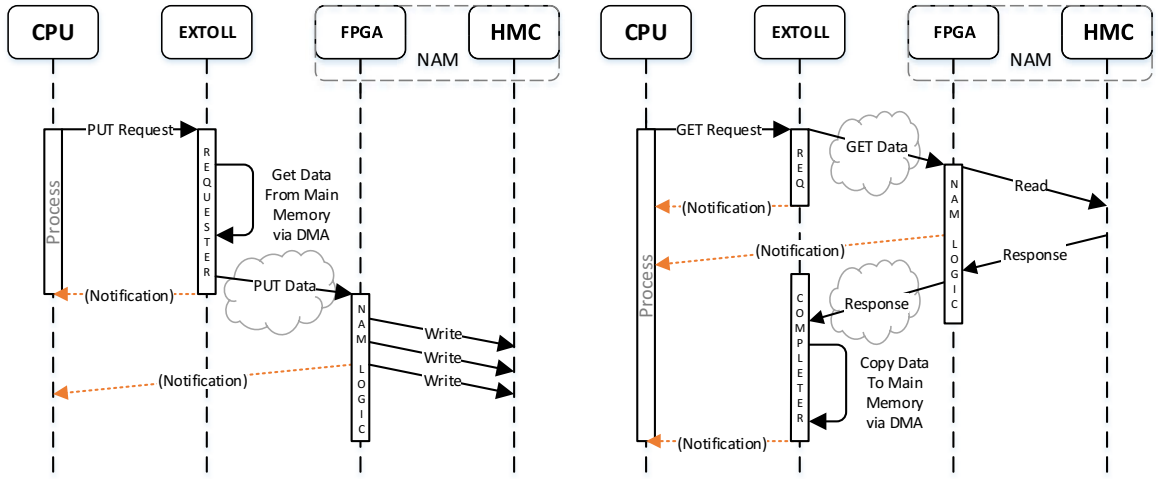
4.3.3.4 NTL - Network Transaction Layer

The Network Transaction Layer (NTL) connects the EXTOLL links via the link MUX to the NAM logic and operates in the clk_hmc clock domain. It decodes and distributes incoming RMA packets targeting HMC as read/write request, the RF for configuration or maintenance, or the NAM CR logic. It is the counterpart of the EXTOLL ASIC RMA unit. For read requests, tag maps are used to retain information that is required to generate corresponding responses. Packets are processed in cut-through mode, i.e. data cycles are immediately forwarded to the next layer. This is opposed to store and forward, where all cycles that belong to a packet are collected first and then forwarded. Cut-through was chosen to enable subsequent layers to receive data faster instead of waiting for an entire packet to become available by the network partition which operates with the relatively slow clk_extoll. Obviously the forwarding mode is only relevant for requests that spread over more than 1 parallel cycle which affects write requests larger than 48 Byte (16 Byte network descriptor + 48 Byte payload in a 512 bit cycle). The 8 Byte SOP cell preceding the network descriptor is initially removed by the NTL. A full-sized RMA packet that carries 496 Byte payload therefore stretches over a total of eight cycles where the first cycle contains 48 Byte payload (+16 Byte protocol overhead) followed by seven cycles with 64 Byte payload each.

Every packet is also subject to a variety of checks. It increases the NAM resistance to false usage by applications or the EMP. Incorrect accesses may include the use of commands other than mentioned in the description of the EXTOLL FPGA link or packets that do not target the NAM. Especially in the initial bring-up phase of the FPGA design and software components it is essential to rather catch exceptions than to risk unexpected behavior. Consequently, the NAM drops any packets out of specification and leaves some debug information in its Register File. The NAM access granularity has been set to 16 Byte to match the granularity of the HMC protocol. This decision reduces design complexity by eliminating various corner cases for packet and address translation between the EXTOLL and HMC protocol.

The NTL strips the packet SOP and otherwise immediately forwards any incoming cycles to the next stage. Its theoretical bandwidth is equal to the capability of the MUX:

$$BW\_EFF_{NTL} = BW\_EFF_{MUX} = 17.54 \text{ GB/s} \tag{4.18}$$



(a) PUT operations may generate up to two notifications. A local requester notification when data has been sent, and another when the data has successfully passed packet checks in the NAM NTL

(b) GET operations may generate up to three notifications. A local requester notification when the GET request has been sent, a second when the request has successfully passed packet checks in the NAM NTL, and a final notification when the requested data has been placed in local memory

Fig. 4.8 NAM/EXTOLL notification mechanism for PUT and GET operations

Notifications

The concept of notifications which can be used to inform processes of certain events has been introduced in Section 4.2.3. The NAM supports this notification mechanism, with the following two modifications as depicted in Figure 4.8: For PUT operations, the completer notification bit set will not generate any notification on the NAM as there is no actual processor present. Instead, a notification directed to the requesting process will be sent as the packet was accepted at the NTL and has passed integrity checks. Similarly, such a notification can be generated when a GET request has been processed in the NTL. Such notifications can be used to ease synchronization between processes that share a common address space on the NAM.

4.3.3.5 HTL - HMC Transaction Layer

The HMC Transaction Layer (HTL) connects the NTL and CR logic to the HMC partition and converts from the RMA protocol to HMC and vice versa. Several properties of the various packet types complicate this protocol conversion. The

following section analyzes these difficulties and presents the implemented translation units. The request direction is examined first.

Requests

The HTL receives packets from either the NTL or the CR functional unit and converts these to HMC packets. Protocol conversion at this point is non-trivial as HMC packets must meet the following requirements:

The maximum packet size is 128 Byte The largest HMC packet that may be transmitted is 128 Byte and an RMA packet can carry up to 496 Byte payload. Hence, a single RMA transfer may trigger several HMC packets.

The memory access granularity is 16 Byte The HMC protocol defines requests with a granularity of 16 Byte and packet sizes ranging from 16 to 128 Byte. Although HMC preserves Byte access using BIT WRITE commands, these are not supported by the HTL to keep the complexity and corner cases of packet conversion at a minimum. This limitation also forces the use of 16 Byte aligned addresses which must be handled in software.

Destination address plus bytecount must not cross a 128 Byte boundary The HMC memory arrays are internally organized in 128 Byte blocks. An issue arises when a request targets an address offset other than zero and the number of Bytes to be read or written would cross a logical 128 Byte boundary. Such an access would cause a wraparound within the block and wrong data would be returned or false memory locations overwritten. This is depicted in Figure 4.9. Hence, block-boundary crossing must be avoided in any case, and in addition to the fact that larger RMA packets must be split regardless it furthermore complicates the protocol conversion.

After all, the requirements mentioned above not only complicate protocol conversion but also negatively affect the achievable bandwidth.

To greatly reduce the complexity of combinational logic it was decided to only utilize a subset of the available HMC packet sizes for write requests. As stated earlier the NTL passes data cycles of an RMA packet independently, and the HTL solely operates with this cycle based approach. Hence, the largest amount of payload to be converted in one conversion step is equal to the datapath-width (64 Byte).

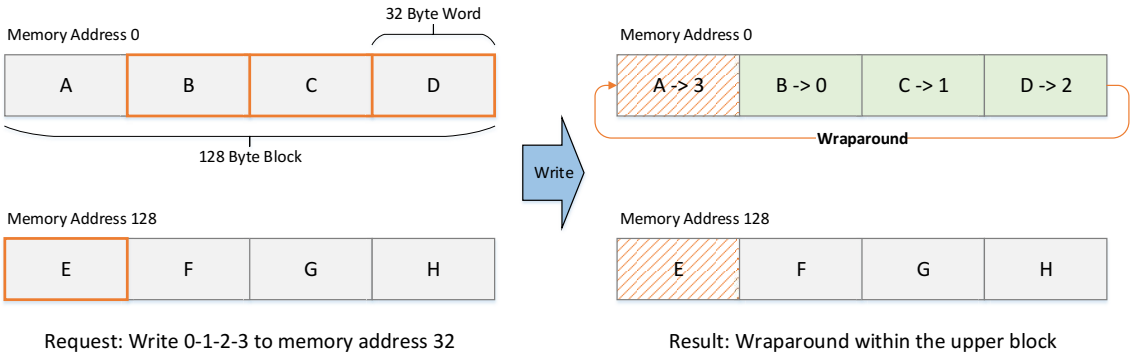


Fig. 4.9 HMC 128 Byte block-boundary crossing example. Left hand side: A request writes the pattern 0-1-2-3 (128 Byte) to memory address 32 intended to overwrite B-C-D-E. Right hand side: Start address plus bytecount cause a wraparound in the upper block. A false memory location was written

The easiest way to perform conversion is to map an RMA request exclusively to 16 Byte HMC packets. This will generate $\frac{496\ \text{Byte}}{16\ \text{Byte}} = 31$ HMC packets out of a full-sized RMA transaction, packed in 16 parallel cycles with two 16 Byte packets per cycle at most (i.e. 2 packets with 16 Byte payload and 16 Byte overhead each). On the one hand this approach eliminates the probability to cross an HMC block-boundary. On the other hand, it is desirable to decrease the number of HMC packets transmitted as many smaller requests targeting a similar memory location limit parallelism and are likely to cause access conflicts in the HMC DRAM. Smaller requests furthermore increase the overhead on the HMC link as every HMC packet includes 16 Byte overhead regardless of its size. 32 Byte HMC packets can be utilized to achieve a reduction in most cases. Still, 16 Byte requests will be issued when approaching a block-boundary or simply no more data is available. It is reasonable to consider 64 Byte packets as the payload of a cycle can be directly mapped to a single packet. However, 64 Byte HMC packets will span over two cycles due to the HMC protocol overhead and can be substituted by a combination of 32B/32B or 48B/16B packets. 48 Byte requests have an additional benefit. They can pack the first cycle of an RMA packet (which has only 48 Byte payload, + 16 Byte RMA header) into a single packet and cycle on the HMC side, whereas a combination of 32B/16B would span over two cycles. In conclusion the three available HMC packet sizes 16B, 32B, and 48B, have been chosen as trade-off between design complexity and resulting number of HMC packets that will be generated.

The HTL first converts an RMA data cycle to HMC packets, one per output cycle, before moving to the next RMA cycle. The obvious side effect of this scheme is that it

increases the protocol overhead and some FLITs remain unused. In order to estimate the implications on performance, Table 4.1 provides two examples of how packets are split in response to the current address and the available HMC packet sizes. The two example packets shown are layered packet A and packet B. The actual layout of the conversion is determined within the first RMA cycle: either the full payload can be packed into a single HMC cycle (packet type A) or it must be split due to block-boundary crossing (packet type B). Hence, packet B type conversion is required whenever the start address of an RMA transaction leaves only 16 or 32 Byte distance to the next 128 Byte boundary as otherwise the full 48 Byte may be processed at once.

Due to the 16 Byte access granularity an RMA packet may target one of eight possible address locations with regard to the 128 Byte block-boundary, i.e. the distance is 16B, 32B, ... up to 128B. Therefore, two out of eight packets will cause a packet B type conversion which requires 16 cycles to complete, while the remaining six packets can be represented by the packet A type and a cycle count of 15.

Using the information above it is possible to calculate the effective bandwidth of the HTL layer for write requests. Out of eight packets, six will take a total of $6 \cdot 15 = 90$ cycles. The remaining two require $2 \cdot 16 = 32$ cycles to complete. This results in an average of $\frac{90+32 \text{ cycles}}{8 \text{ packets}} = 15.25$ cycles per packet. 15.25 cycles can carry $15.25 \cdot 64$ Byte $= 976$ Byte of which 496 Byte are actual payload. The resulting efficiency EFF_{HTL_REQ} is therefore:

$$EFF_{HTL\_REQ} = \frac{\text{Data Bytes}}{\text{Total Bytes}} = \frac{496}{976} = 50.8\ \% \tag{4.19}$$

The effective write request bandwidth $BW\_EFF_{HTL\_REQ}$ is now calculated with:

$$\begin{aligned} BW\_EFF_{HTL\_REQ} &= clk\_hmc \cdot \text{datapath-width} \cdot EFF_{HTL\_REQ} \\ BW\_EFF_{HTL\_REQ} &= 318.1825 \text{ MHz} \cdot 512 \text{ bit} \cdot 50.8\ \% \\ BW\_EFF_{HTL\_REQ} &= 82.85 \text{ Gbps} = 10.35 \text{ GB/s} \end{aligned} \tag{4.20}$$

So far the conversion analysis between the two protocols has only considered write requests. Read requests, however, are treated similarly as they have to obey the HMC packet requirements described above, especially because a read request may also be subject to block-boundary crossing. Luckily, the conversion effort is greatly reduced due to one significant difference: HMC read requests are always 16 Byte in size regardless of the requested payload size. Translation for a maximum-sized RMA request takes

Table 4.1 HTL request packet splitting example. Two 496 Byte RMA packets are converted. Depending on the packet start address the first RMA cycle might be split to avoid a 128 Byte block-boundary crossing. Packets that do not require initial packet splitting (type A) will take 15 cycles. These packets have a target address to block-boundary distance of 48 Byte or more. All other packets (type B) take 16 cycles to complete

RMA packet type A. 496 Byte. Start address 0. No split in the first cycle

RMA Cycle	HMC Cycle	Next Address	Payload [Byte]	Action
1	1	0	48	First cycle with only 48 Byte
2	2	48	48	Send 48 Byte. 16 Byte remain
2	3	96	16	Send remaining 16 Byte
3	4	112	16	Send 16 Byte to avoid boundary crossing
3	5	128	48	Send remaining 48 Byte
4	6	176	48	Send 48 Byte. 16 Byte remain
4	7	224	16	Send remaining 16 Byte
5	8	240	16	Send 16 Byte to avoid boundary crossing
5	9	256	48	Send remaining 48 Byte
6	10	304	48	Send 48 Byte. 16 Byte remain
6	11	352	16	Send remaining 16 Byte
7	12	368	16	Send 16 Byte to avoid boundary crossing
7	13	384	48	Send remaining 48 Byte
8	14	432	48	Send 48 Byte. 16 Byte remain
8	15	480	16	Send remaining 16 Byte

RMA packet type B. 496 Byte. Start address 496. First cycle must be split

1	1	496	16	Send 16 Byte to avoid boundary crossing
1	2	512	32	Send remaining 32 Byte
2	3	544	48	Send 48 Byte. 16 Byte remain
2	4	592	16	Send remaining 16 Byte
3	5	608	32	Send 32 Byte to avoid boundary crossing
3	6	640	32	Send remaining 32 Byte
4	7	672	48	Send 48 Byte. 16 Byte remain
4	8	720	16	Send remaining 16 Byte
5	9	736	32	Send 32 Byte to avoid boundary crossing
5	10	768	32	Send remaining 32 Byte
6	11	800	48	Send 48 Byte. 16 Byte remain
6	12	848	16	Send remaining 16 Byte
7	13	864	32	Send 32 Byte to avoid boundary crossing
7	14	896	32	Send remaining 32 Byte
8	15	928	48	Send 48 Byte. 16 Byte remain
8	16	976	16	Send remaining 16 Byte

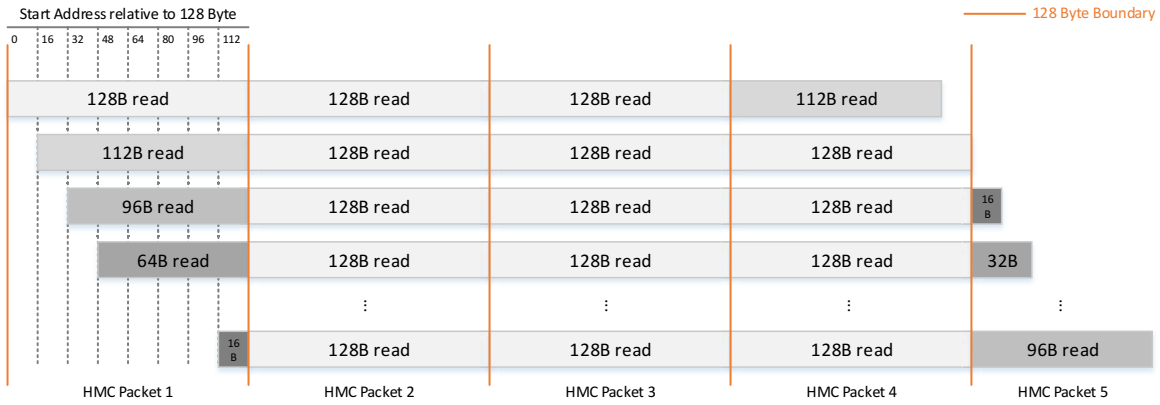


Fig. 4.10 496 Byte RMA read request to HMC packet mapping. Depending on the start address distance relative to the next 128 Byte boundary four or five HMC read requests will be generated

up three cycles at most and results in four to five HMC requests with up to 128 Byte. This process is depicted in Figure 4.10. It shows that the determination of the actual number of HMC requests is based on whether and at what point requests have to be split to avoid reading through block boundaries. In a given request stream that requests eight or more full-sized RMA packets, the translation process for each subsequent RMA packet is deterministic as it iterates through the eight possible variations. All of these have in common that three 128 Byte reads will be issued, complemented by one or two reads to address the remaining 112 Byte of the 496 Byte RMA request. Eventually, eight 496 Byte RMA packets will be translated to 24 128 Byte requests and two additional requests for each of the remaining packet sizes (16 to 112 Byte).

Performance is not considered critical at this point. Although it can take up to three cycles to request 496 Byte of data, neither the HMC nor the response path are able to deliver the requested bandwidth. In fact, even if there was no protocol overhead at all, three response cycles can carry only a maximum of 192 Byte payload (64 Byte per parallel cycle). The response bandwidth will be examined later in this section.

As packets were converted they are sent to the openHMC controller and each read request gets a sequence number assigned. These sequence numbers are stored in four different tag maps as four read requests may be packed into a single cycle. The purpose of tagging is to allow the response path to properly reorder HMC responses and to reassemble these back into larger RMA packets.

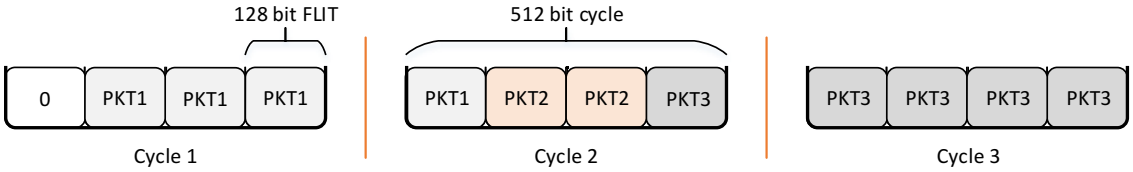


Fig. 4.11 Response packet sampling example: Three cycles were sampled at the openHMC controller output. The second cycle contains a full packet *PKT2* along with the final FLIT of *PKT1* and the first FLIT of *PKT3*

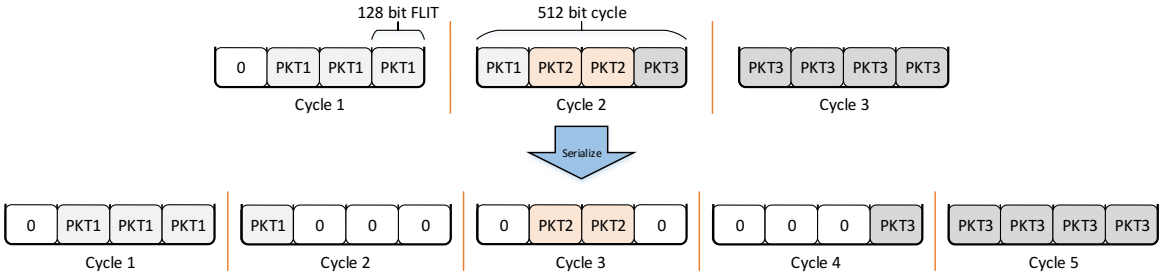


Fig. 4.12 Packet serialization example. The second cycle contains one full and parts of two other packets. Packets are separated using two additional cycles

Responses

The HTL response path receives HMC response packets from the openHMC controller. HMC internally operates in a FLIT granularity and the FPGA will buffer four FLITs to create a 512 bit cycle. A parallel cycle may contain (parts of) several packets at a time as shown in Figure 4.11. To make decoding and processing easier for the following stages, the incoming packets are first separated so that in any given cycle only data from one packet is forwarded. Figure 4.12 depicts how extra cycles are used to serialize the packet stream in Figure 4.11. Unfortunately, extra cycles will also throttle the throughput. To properly estimate the achievable bandwidth at this stage it is necessary to analyze how packets can be aligned within the parallel cycle. Depending on this position and the packet length, the number of cycles required to forward the packet varies. The four possible start positions and their implication on the cycle count are depicted in Figure 4.13. As can be seen a 32 Byte HMC response can have four different layouts and requires one cycle or spreads over two cycles equally in 50% of the time assuming a uniform distribution. Its average required cycle count is therefore 1.5. The splitting scheme shown applies to all other packet sizes likewise and the results are summarized in Table 4.2. It lists the various packet sizes and their minimum, maximum, and average cycle spread count for a single response. In a given request/response stream, eight consecutive, randomly picked requests can be selected

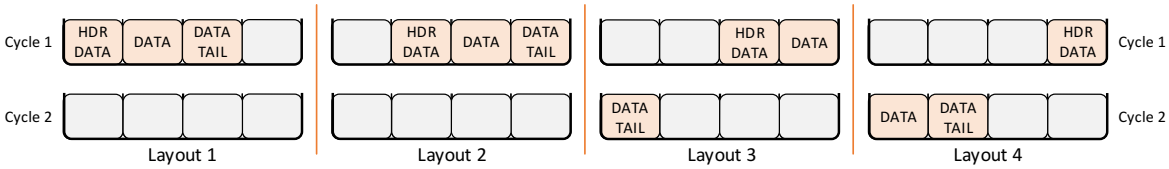


Fig. 4.13 Response packet layouts: A 32 Byte HMC response is embedded in a 48 Byte packet (including 16 Byte protocol overhead). Depending on the FLIT start position within the 512 bit word this packet is received in four different layouts and spreads over one or two cycles

Table 4.2 HMC response packet serialization overview. The data highlights the average cycle count that is required to forward a packet for each size. Translated to the total number of packets and their average cycle count, the efficiency of the HTL response path can be calculated

Packet	Cycle counts		Probability			For eight 496 Byte RMA Packets			
Size	Min	Max	Min	Max	Avg	Count	Cycles	Raw B	Payload B
16B	1	2	3	1	1.25	2	2.5	160	32
32B	1	2	2	2	1.5	2	3	192	64
48B	1	2	1	3	1.75	2	3.5	224	96
64B	2	2	4		2	2	4	256	128
80B	2	3	3	1	2.25	2	4.5	288	160
96B	2	3	2	2	2.5	2	5	320	192
112B	2	3	1	3	2.75	2	5.5	352	224
128B	3	3	4		3	24	72	4608	3072
SUM	-	-	-	-	-	38	100	6400	3968

to calculate the total number of cycles that will be spent. This cycle count, multiplied by 64 Byte, gives the total (raw) number of Bytes that will be forwarded. The actual number of Bytes that carry payload, however, is less than that. Eight 496 Byte RMA read requests will generate 38 HMC packets that will take 100 cycles after serialization, and only 3968 Bytes out of 6400 carry payload.

Since all of the following HTL stages will not delay the response stream any further this information can now be used to calculate the HTL response efficiency with:

$$EFF_{HTL\_RSP} = \frac{\text{Payload Bytes}}{\text{Total Bytes}} = \frac{3968}{6400} = 62\ \% \tag{4.21}$$

Table 4.3 NAM design building blocks bandwidth summary: single EXTOLL Link operation. For reading and writing, achievable bandwidth is limited by the EXTOLL Link

Functional Unit	Request Bandwidth [GB/s]	Response Bandwidth [GB/s]
EXTOLL	8.31	8.58
Link MUX	17.54	17.54
NTL	17.54	17.54
HTL	10.35	12.62
openHMC	18.02	18.02

which results in a maximum effective bandwidth of:

$$\begin{aligned} BW\_EFF_{HTL\_RSP} &= clk\_hmc \cdot \text{datapath-width} \cdot EFF_{HTL\_RSP} \\ BW\_EFF_{HTL\_RSP} &= 318.1825 \text{ MHz} \cdot 512 \text{ bit} \cdot 62 \text{ \%} \\ BW\_EFF_{HTL\_RSP} &= 12.62 \text{ GB/s} \end{aligned} \tag{4.22}$$

The next step in response processing is to reorder HMC packets as they can return out of order. This is done using the sequence numbers that have been placed in the TAG maps by the HTL request modules. Finally, individual packets are recombined to create their corresponding RMA GET responses. These are forwarded to either the NTL if the request was a remote read/write or otherwise to the CR unit.

4.4 Summary Estimated Read/Write Performance

The previous section explained the individual NAM design building blocks and analyzed their estimated performance. Table 4.3 summarizes the results for reading and writing with one EXTOLL Link to easily identify existing bottlenecks. For both, reading and writing, the EXTOLL Link bandwidth is the limiting factor. This is independent of request sizes and access patterns since even in the worst case usage the HMC would be able to deliver more bandwidth[5]. Therefore, the write bandwidth is expected to peak at about 8.31 GB/s and the read bandwidth at 8.58 GB/s, respectively. Similarly, Table 4.4 highlights the expected bottlenecks when writing and reading to and from both EXTOLL links. It assumes that request addresses are somewhat distributed and do not target the same memory location as in this case the HMC bandwidth could have

[5] For more information on HMC access patterns and bandwidths refer to Section 3.3.4.

Table 4.4 NAM design building blocks bandwidth summary: dual EXTOLL Link operation. For reading and writing, achievable bandwidth is limited by the HTL

Functional Unit	Request Bandwidth [GB/s]	Response Bandwidth [GB/s]
2 × EXTOLL	16.62	17.16
Link MUX	17.54	17.54
NTL	17.54	17.54
HTL	10.35	12.62
openHMC	18.02	18.02

a negative impact[6]. Compared to Table 4.3 it becomes clear that the bottlenecks now have shifted into the NAM logic, more specific into the HTL request path for writes, and the HTL response path for reads. Hence the expected maximum bandwidth for writes is 10.35 GB/s and 12.62 GB/s for reads, respectively.

These results will be used as a reference for the real hardware measurements conducted in Chapter 5.

4.5 Checkpoint/Restart

In DEEP-ER the NAM carries out XOR based checkpoint/restart as a potential performance improvement to the existing SCR-Partner checkpointing scheme with SIONlib. Compared to this partner approach where one checkpoint is stored at the task local node and also transferred to another remote node, XOR checkpointing generates a parity via a bit-wise XOR operation from the checkpoints of all participating ranks in a group. The result is as large as the largest individual checkpoint and can then be used to recover from any single rank failure within a group.

XOR checkpointing reduces the overhead in storage capacity required to perform checkpoint/restart as every node holds only a fraction of the parity information, compared to Partner checkpointing where checkpoints are simply duplicated and distributed across nodes. It comes, however, at the expense of calculation overhead to generate the parity. For more information on checkpointing and fault tolerance refer to Section 2.3.

[6] See Footnote 5.

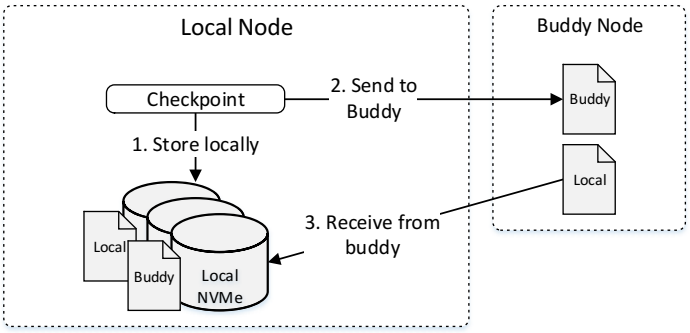


Fig. 4.14 SIONlib-Buddy checkpointing scheme with two nodes

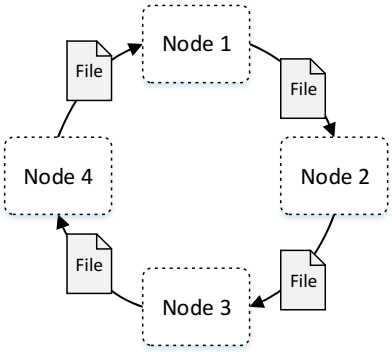


Fig. 4.15 SIONlib ring fashion file exchange with more than two nodes

The NAM carries out the parity computation and stores the result in the HMC. Each NAM in the system is associated with a set of ranks (just like a set in SCR) and within each set a single rank failure may be recovered. Therefore, a system can have as many sets as there are NAMs in the system.

The following section documents the design process of the CR functional unit and describes how the NAM creates the XOR parity, and how it can be used to restart from a failure.

4.5.1 Buddy Checkpointing in DEEP-ER

The DEEP-ER resiliency scheme is based on SCR-Partner checkpointing which has been extended to support the SIONlib [102] parallel I/O library. SIONlib allows to merge I/O streams of multiple processes into one or multiple files, removing file system congestion due to many smaller, unaligned data blocks. This process is applied to checkpoint data on all processes on a node so that only a single file is written per node. The SIONlib-Buddy checkpointing approach writes this file to the local NVMe devices and also creates the same file on a remote *buddy* node. It then initiates a receive routine to fetch the local checkpoint of a remote buddy which is also placed in the local NVMe (Figure 4.14). Note that the buddy node where the local checkpoint is written to is not necessarily the same node a remote checkpoint is received from. SIONlib achieves an additional speed-up over standard SCR-Partner by overlapping the write-out functions to local storage and the buddy node. If more than two nodes

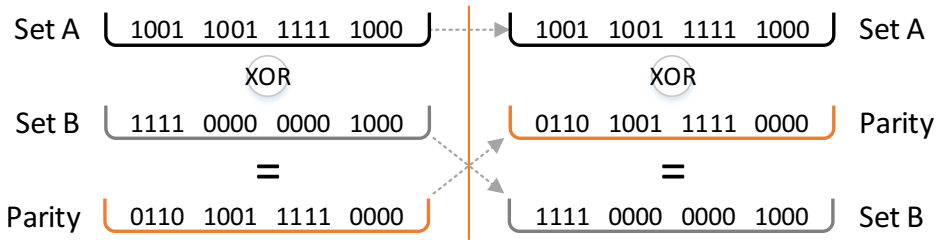


Fig. 4.16 XOR parity generation (left) and reconstruction of a missing checkpoint dataset (right)

participate in the process, buddy nodes are assigned and files are exchanged in a ring fashion (Figure 4.15).

4.5.2 Definitions

The following definitions may be helpful to understand the remainder of this section.

XOR Parity

A XOR (Exclusive OR) operation applied to several sets of data can be used to generate a parity. The size of the parity is as large as the largest dataset. With the help of this parity any single missing set of data can be reconstructed. Figure 4.16 depicts a simple example of this process.

Segmentation

The NAM internally segments checkpoint data into smaller chunks, currently 496 Byte which is the EXTOLL network MTU and reflects a maximum-sized RMA packet. Segment numbers are assigned since the XOR operation is applied on equal segment numbers over all checkpoint data sets.

Rank

A rank may be a remote process or remote node with one or multiple processes depending on the checkpointing granularity. For example, SIONlib merges checkpoints of multiple processes on a node into a single file. In this context a rank equals one node.

4.5.3 Design Space Exploration

In order for the NAM to create a parity out of a group of checkpoints it has to receive all participating datasets. There are three ways to do so:

1. The nodes unconditionally send their checkpoints to the NAM which acts as a passive device. This approach requires the least hardware effort.
2. The nodes send their checkpoints to the NAM which acts as a semi-passive device. All nodes send their checkpoints to the NAM upon request. Synchronization between the nodes and the NAM is required to control data flow.
3. The NAM reads checkpoint data from the nodes. It is up to the NAM hardware to decide when and how much data to fetch. The application will need signal readiness and wait for a notification of completion.

Option 1: Nodes send data In the easiest approach all nodes send their checkpoints whenever ready and without any further inter-process synchronization. If one or more nodes delay the transmission it is not guaranteed that the NAM can hold all relevant segments to generate the next XOR segment because the FPGA buffer capacity might be insufficient. As a result, all currently available segments would have to be XORed and the temporary result would be written to the HMC. When all of the remaining and required segments have arrived, the temporary result would need to be read again before the parity can be generated. There is a high risk that data will be moved between FPGA and HMC multiple times.

Option 2: Nodes send data upon request by the NAM This approach eliminates the risk of flooding the NAM with data from individual nodes. The NAM requests every segment or a set of segments of defined size with notification PUTs directed to the remote process. A disadvantage is the fact that processes stay busy with waiting for these notifications, up to several million times for a 2 GB checkpoint using 496 Byte RMA transactions. In addition, each transaction is eventually sourced by a software descriptor which must be translated and might involve address translation.

Option 3: NAM retrieves checkpoint data autonomously With this approach the NAM has exclusive control over any data movement. It can autonomously request segments in a way that the FPGA internal buffer space is optimally

used. Remote nodes will need to inform that checkpoint data may be retrieved. Completion of the checkpointing process is signaled by a notification to all participating nodes which only have to check for this notification before a next checkpoint may be created. Another advantage is the fact that no software to network descriptor translation is performed, potentially increasing the overall performance.

The dataset granularity must obey the NAM internal access granularity of 16 Byte. It is the responsibility of the software to pad datasets with zeros up to the next 16 Byte boundary. Also the NAM must provide a reasonable amount of buffer space to avoid frequent read/modify/write to the HMC (option 1) or to allow sufficient in-flight transactions to exist (option 2 and 3). Buffer space must be partitioned to allow holding segments of up to 44 nodes at a time to cover all 88 nodes with two NAMs in the DEEP-ER prototype.

4.5.3.1 Summary of Design Decisions

Three approaches to transfer data were described above. Although option 3 requires a higher hardware implementation effort, software complexity and processor-time to transfer data is greatly reduced while providing the most resource efficient solution. To optimally utilize available Block RAM memory of the Virtex 7 FPGA the NAM will request a maximum of 128 in-flight segments with 496 Byte each per node. The required number of nodes that must be handled by the NAM is 44, and the actual number is slightly increased to 48 to allow for a small imbalance in node-to-NAM set assignments.

4.5.4 Vision: NAM-XOR Checkpointing in DEEP-ER

Based on the design decisions for NAM-XOR checkpointing, Figure 4.17 depicts the envisioned checkpoint creation flow with SIONlib. First, SIONlib writes a single file from the checkpoints of all processes on a node to its local NVMe. The file is then re-read into the node-local memory where it is ready to be fetched by the NAM.

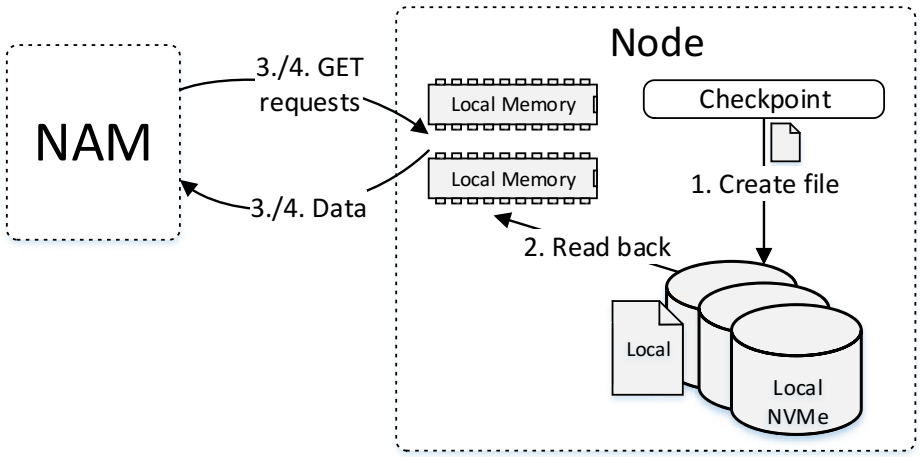


Fig. 4.17 NAM/SIONlib checkpoint creation example with one node

4.5.5 Configuration

Before the NAM CR feature can be used it must be configured by a root process. RRA packets are used to read and write a set of CR registers in the NAM Register File. The CR control unit expects the number of participating ranks (register C0) and the unique EXTOLL NodeID + Virtual Process Identifier (VPID) (C2) of each rank along with the size (C2) and remote memory start address (C1) of the corresponding checkpoint, one rank per access. As shown in Figure 4.18 these steps are repeated until all ranks have been configured. The configuration process and any misconfiguration are monitored in dedicated status registers. As soon as all information has been written the NAM is operational for CR.

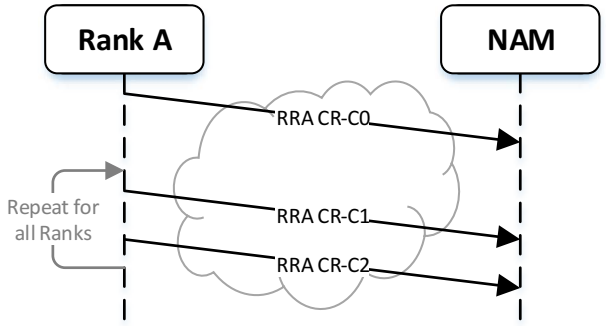


Fig. 4.18 NAM CR configuration process

Configuration and all subsequent processes are carried out by the libNAM library (see Section 4.7.2), which provides interfaces to a higher layer such as SIONlib.

4.5.6 Generating a Checkpoint

Figure 4.19 depicts the checkpointing process on a network transaction level. The checkpoint sizes are 5 segments for Rank A, and 4 segments for Rank B respectively. In this example, the buffer sizes in the FPGA have been set to accommodate 3 segments per rank at most. After configuration has been performed an application may be executed. Whenever a rank is ready to have its checkpoint fetched by the NAM it posts a flag into one of the CR control registers via RRA. This will trigger a burst of RMA read requests up to 128 in-flight segments (only three in this example) to retrieve data from the corresponding rank. As this process is ongoing, additional flags from other ranks may be written which will cause the RMA read request scheme to alternate through all currently flagged ranks. As GET responses return the NAM places these segments into buffers, one per rank. The performance for GET responses from remote nodes to the NAM is expected to be close to what RMA PUTs from an ASIC to the NAM can achieve since both packet types look very similar.

When matching segments from all participating ranks have been received a XOR operation on this set of segments is performed and the resulting parity is written to the HMC. With every processed set, another segment from all nodes may be requested as the buffer space is now freed up. Since checkpoint sizes can vary for each rank, data fetch operations for some ranks may be ongoing while others are finished. The NAM takes the largest available checkpoint as reference and internally pads all other checkpoints with zeros so that the XOR result stays correct. The segment request process is repeated until all checkpoints were fully transported and the last request to each rank will have a notification bit set to signal completion.

For any subsequent checkpoints, step A in the sequence is obsolete when there is no change in the configuration.

4.5.7 Restarting from a Checkpoint

When a rank has failed the root process is responsible to update the entries in the NAM CR control unit accordingly. With completion of this update, the NAM will start requesting segments from all remaining ranks. This process is very similar to the

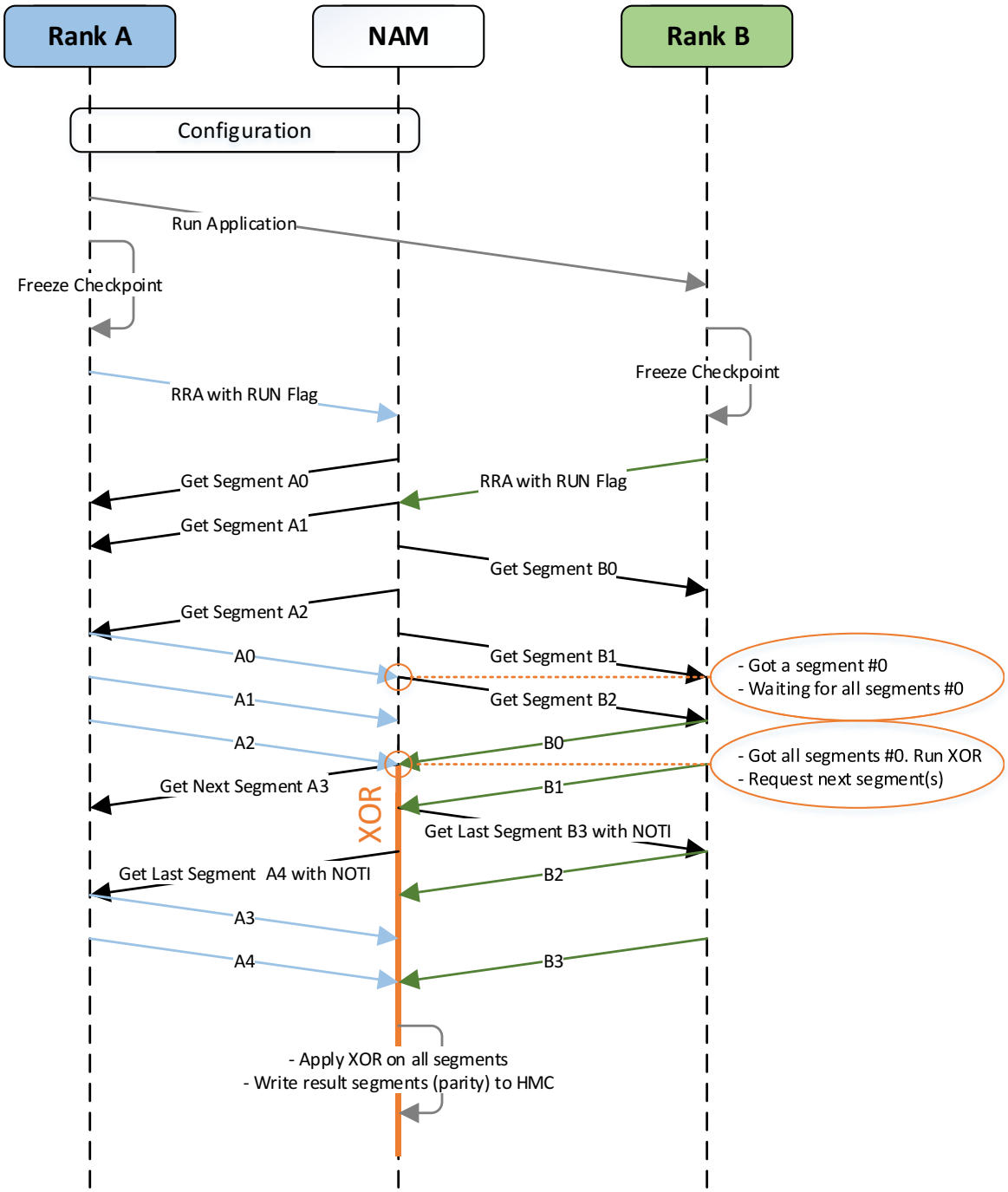


Fig. 4.19 NAM parity checkpoint creation example with two participating ranks A and B. The rank A checkpoint is 5 segments in size and the rank B checkpoint 4, respectively. The resulting parity is as large as the largest checkpoint; 5 segments in this case. Segment GET requests are arbitrated among all currently valid ranks

checkpoint creation process with one main difference: the checkpoint of the failed rank is replaced by the parity information that has been stored in the HMC. A low-level diagram that highlights the individual sequences is depicted in Figure 4.20. The result of the XOR operation on all remaining checkpoints and the parity is again written to the HMC. This information reflects the missing checkpoint. The NAM informs the failed (or newly configured) rank which then fetches the data via regular RMA reads.

A mandatory precondition to restart after a failure is that a parity checkpoint has been written previously. In the unlikely case that a rank fails while a checkpoint creation process is ongoing, the parity information may be invalid and no restart is possible. One possible workaround to avoid this situation is to partition the HMC address space into two equal-sized blocks. The NAM will then alternate between the blocks for each subsequent checkpoint. A major drawback of this scheme is that the available capacity is cut in half.

4.5.8 CR Functional Unit

The CR functional unit is depicted in Figure 4.21. The starting point for any CR process is the control unit. After configuration, it receives the start CR flags which triggers RMA read requests to be issued. Any packets arriving at the NTL completer must be forwarded to the correct unit, depending on whether or not a packet/segment belongs to a CR process. This data is shifted to the input stage which looks up the corresponding buffer index of the remote process. The segment is then shifted into the buffer array where it remains until the matching segments from all participating nodes have arrived. Eventually, these segments are shifted into the XOR stage which creates the parity. A final stage generates the HMC destination address, frames the packet into a suitable format, and forwards it to the HTL layer.

4.5.9 Estimated Performance

The achievable CR performance depends on many factors and without actual measurements it is not possible to make a prediction at this point. Benchmarks will have to show if the dimensioning of the NAM internal buffers is sufficient, how well this approach scales with the number of participating ranks, and if the newly created software components are able to make use of this novel hardware architecture. For the task of collecting checkpoints it is expected that the bandwidth is higher than for

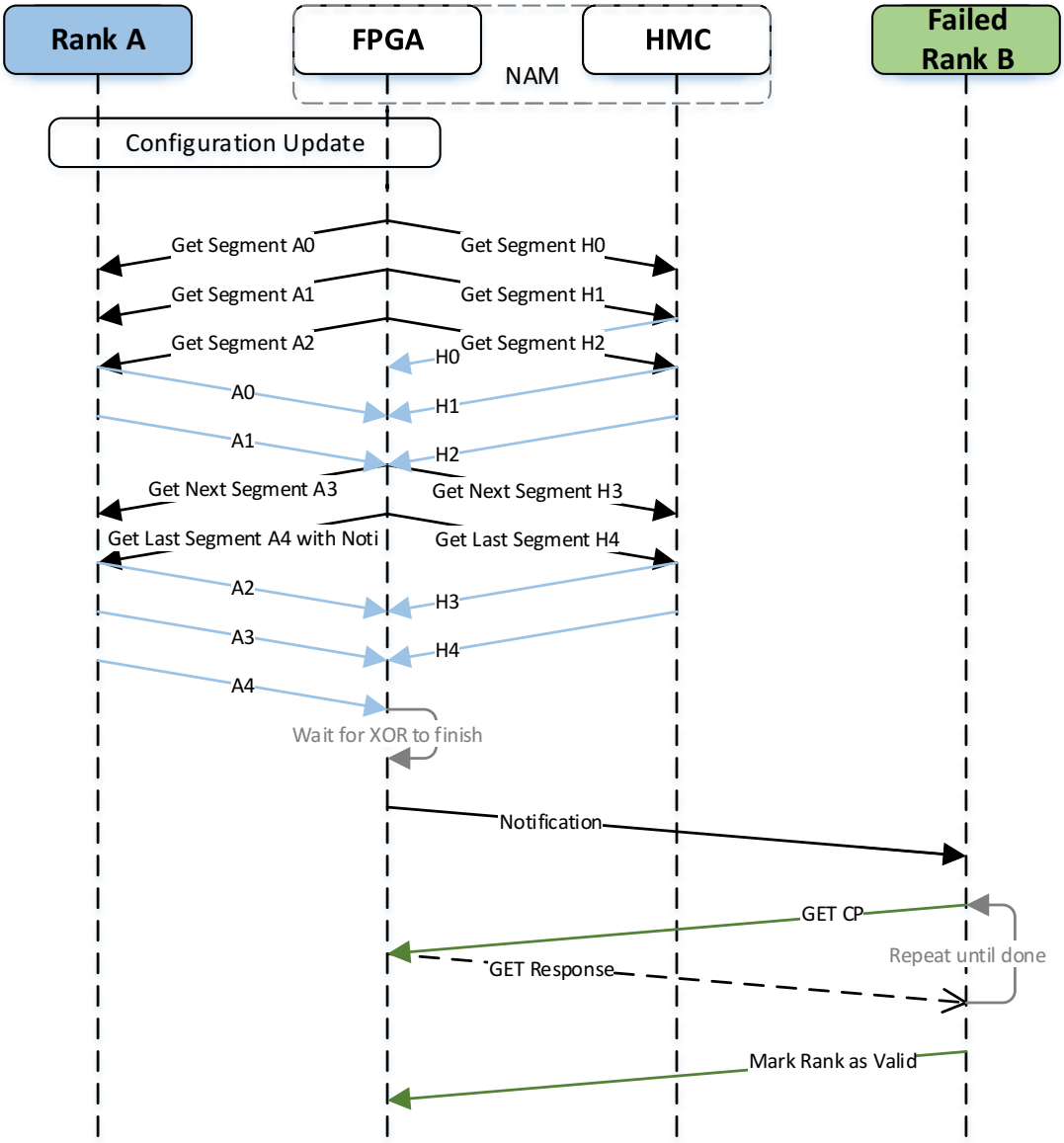


Fig. 4.20 NAM restart process. Rank B failed and its checkpoint is now replaced by the parity which resides in the HMC. Similar to the checkpoint process the NAM now collects the checkpoints from all remaining ranks and again applies a XOR function. This operation results in the missing checkpoint of rank B

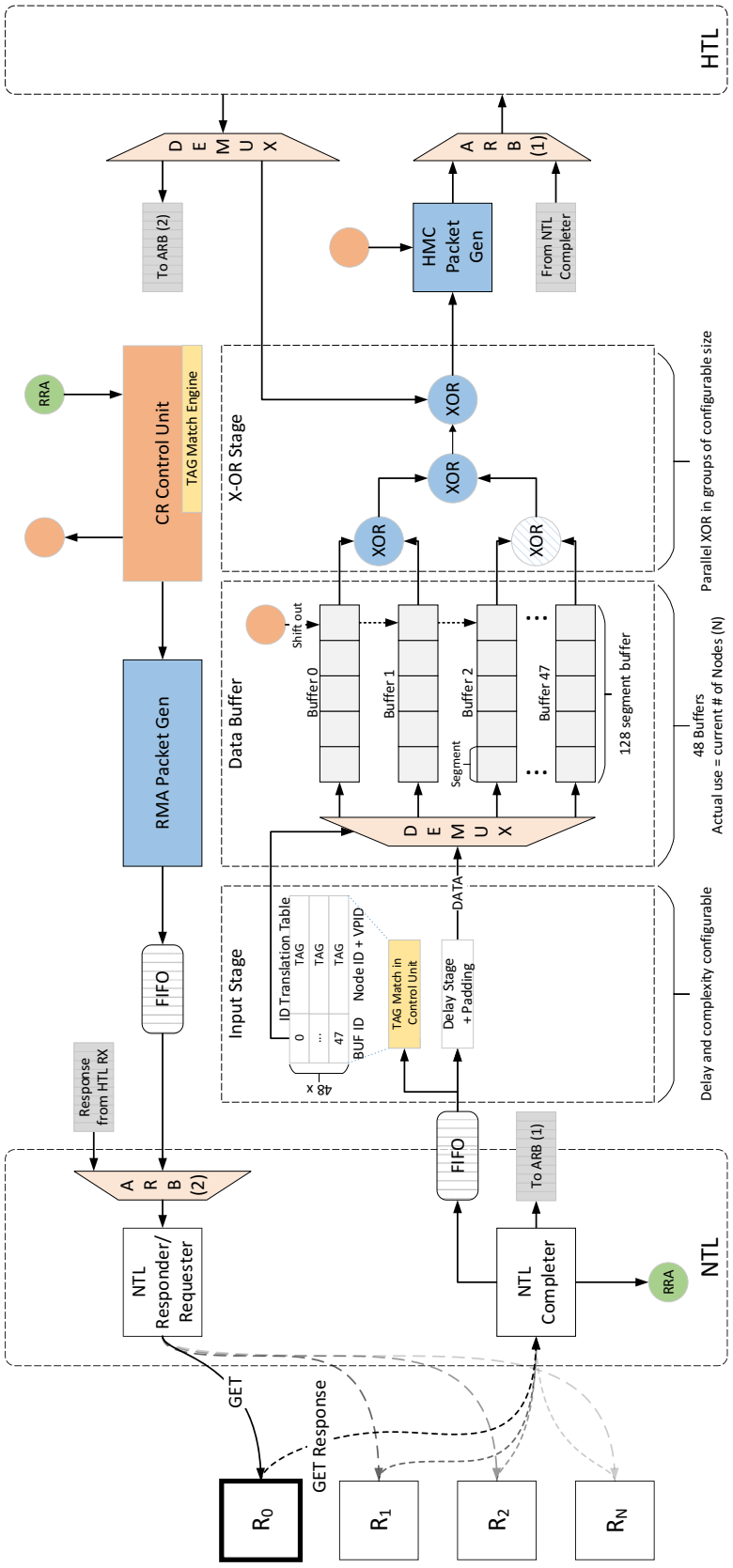


Fig. 4.21 CR functional unit block diagram. An input stage distributes incoming packets onto one of 48 available buffers. As matching segments from all participating nodes have arrived, data is shifted to the XOR stage which generates the parity. The process is controlled by the CR control unit, accessible and configurable via RRA. Two additional modules create request packets directed to the HMC to read or write the parity, or to the network as read requests to get additional checkpoint segments

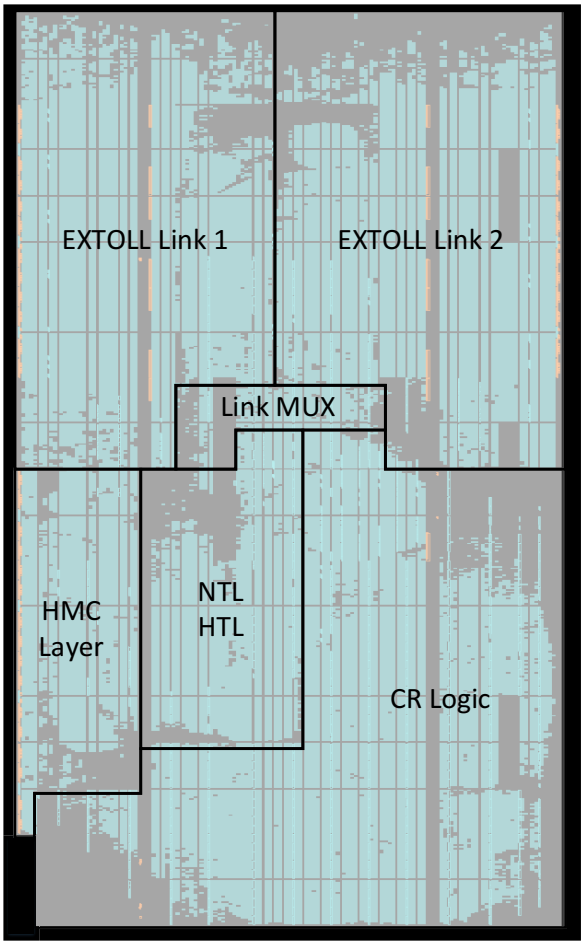


Fig. 4.22 NAM design device view and floor plan

just reading or writing to the NAM. The main reason for this is that the bandwidth limiting NAM HTL layer is mostly avoided except for writing or reading the XOR parity from the HMC. However, this is only true when both NAM links can be accessed and Section 4.7 will discuss how this requirement is influenced by the NAM software stack.

4.6 Implementation Results

The full NAM design was implemented in the Virtex 7 690T FPGA. Figure 4.22 shows that a floor plan was applied to partition the available space in the FPGA. In general, a carefully applied floor plan can reduce routing congestion and place&route runtimes, and will also lead to more reliable results. It can also be seen that the device is reasonably utilized. The CR logic, for example, currently supports buffer space for up

Table 4.5 NAM design resource utilization in a Virtex 7 690T FPGA. Percentages are listed in reference to the total number of available resources of same type

Resource Type	LUTs	Registers	BRAM	DSP
Utilization	273k (63.0%)	199k (23%)	553 (37.6%)	214 (5.9%)
Per Functional Unit				
One EXTOLL Link	66.8k (15.4%)	57.2k (6.6%)	30.50 (2.1%)	47 (1.3%)
EXTOLL MUX	3.8k (0.9%)	2.3k (0.3%)	30 (2%)	0 (0%)
HTL/NTL	24.8k (5.7%)	16.2k (1.9%)	42 (2.9%)	12 (0.3%)
CR Logic	87.2k (20.1%)	43.8k (5.1%)	404 (27.5%)	102 (2.8%)
HMC Layer	21.6k (5%)	19k (2.2%)	15.5 (1.1%)	2 (0.1%)

to 48 ranks at a time. A further increase of the number of ranks would increase Block RAM usage in the specified device region, and significantly increase routing congestion in this area. Routing congestion also comes in heavily when operating frequencies are increased as the implementation tools start to replicate logic in order to reduce trace lengths and fan-out. The modules that suffered most from routing congestion are the EXTOLL links (f_{max} = 200 MHz) and the CR logic (f_{max} = 230 MHz). The final utilization report can be found in Table 4.5.

4.7 NAM Software

Even the best hardware is useless without software that can actually use it. This section describes the software components that were developed or modified to make use of the NAM. There are three main components in this scope: a NAM-aware network setup and management tool to seamlessly integrate this new device with EXTOLL ASICs. An additional user-level Application Programming Interface (API) that provides access to the NAM and implements the CR features. And finally, a service as central instance to handle and manage NAMs and its allocations system-wide.

4.7.1 EMP Extension

The EMP is a software component that is integrated with the EXTOLL software stack. It is used to initially assign NIC identifiers and to setup routing to and between EXTOLL devices in a network. In fact, EMP must be run anytime a system is powered down or even a single node was replaced. In its original form, EMP does not support NAMs as it expects that every connected device also provides routing tables and is able to route through two links via the EXTOLL Crossbar (XBAR). The NAM is an endpoint for any traffic and does not provide a routing table as routing from one link to the other is not supported. Hence, the network must be properly configured to ensure that only packets that actually target a specific NAM will be sent to it. An additional hardware device type was added to the EMP which can now route to and from NAMs but will not attempt to route through it. Currently only fixed and deterministic routing is supported with exactly one path from one node to another.

4.7.2 The libNAM Library

The libNAM library operates on top of the existing EXTOLL RMA API. The function calls provided by libNAM are very similar to libRMA so that existing user applications can be modified without much effort. Listing 4.1 shows a code example to write and read to and from the NAM. In the initial bring-up phase of the NAM hardware-software interaction many of the features that were required to protect the NAM from false usage were implemented in hardware (e.g. a violation of the 16 Byte granularity or unsupported commands). These protection features were gradually shifted into the software, hence reducing hardware and associated implementation complexity.

Reading and writing is realized with send and receive buffers organized in a ring structure. The EXTOLL/NAM notification mechanism is utilized to handle the buffer space, i.e. to free up locations when data has been transmitted (PUT) or received (GET). The number and sizes of the elements a buffer can hold is configurable and at the same time the limit for outstanding transactions.

Currently, data is sent and received on only one of four available EXTOLL Virtual Channels. Measurements conducted in Chapter 5 will have to unveil if and how strong this affects performance. A possible implementation that uses all VCs would require libNAM to use dedicated buffers, one per VC to properly handle GET responses that might return out of order.

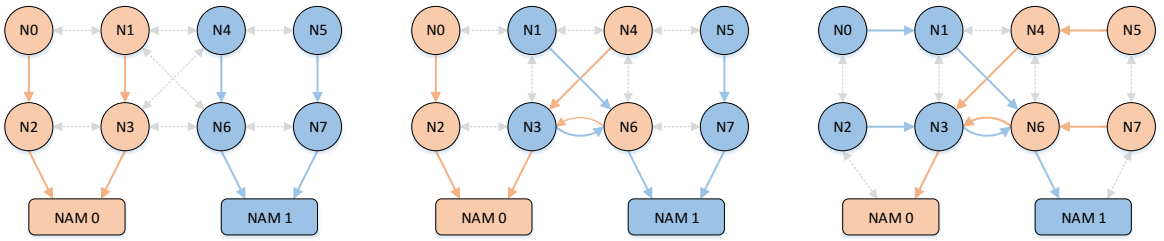
```
int main(int argc, char **argv)
{
   nam_allocation_t *my_alloc;
   char hello[] = "Hello NAM!";
   char transferred[13];
   //Allocate NAM for Read/Write
   my_alloc = nam_malloc(sizeof(hello));
   //PUT and GET data
   nam_put_sync(hello, 0, sizeof(hello), my_alloc);
   nam_get_sync(transferred, 0, sizeof(transferred), my_alloc);
   printf("Transferred from NAM: <%s>\n", transferred);
   //Release Allocation
   nam_free(my_alloc);
   return 0;
}
```

Listing 4.1 libNAM PUT/GET usage example

In subsequent libNAM implementations stages an MPI-based layer was added to allow sharing a NAM allocation between processes. This layer furthermore allows to coordinate checkpoint and restart processes for the NAM CR use case. As there may exist multiple NAMs in a system, libNAM forms sets of participating nodes in a CR process and assigns these sets to one of the NAMs. This assignment process is currently implemented in a pseudo-random fashion that balances the number of nodes among sets.

Unfortunately, assigning nodes to NAMs without additional information about routing comes with obvious drawbacks. Figure 4.23 depicts various possible set assignments for an example network with eight nodes and two NAMs. It can be seen that there exist good mappings with potentially low routing congestion and short distances, but also bad mappings that require more network hops and where only one NAM link will be used. As routes are static the system behavior in response to NAM placement and set configuration is predictable. It is therefore essential to assign sets in consideration of the network topology and routing scheme. This task can either be offloaded to the user, who must provide an appropriate mapping scheme, or to libNAM which could use the information provided by EMP to optimally form sets.

It is also possible that the job scheduler selects a node combination that inevitably leads to a similar condition. Figure 4.24 shows two possible node combinations for a job running on two nodes. Assumed is a shortest-path routing algorithm with fixed

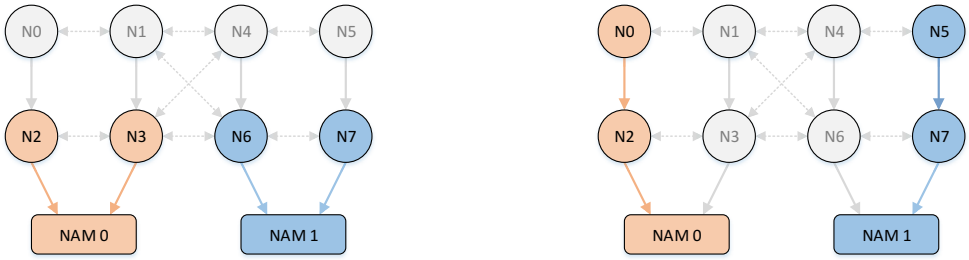


(a) Optimal mapping. The logically nearest nodes are assigned. Distances are small and all NAM links are utilized

(b) Good mapping. Larger distances and higher risk of routing congestion. All NAM links are utilized

(c) Bad mapping. Increased number of hops leads to routing congestion. Only one link per NAM due to static routes

Fig. 4.23 NAM-XOR set mapping examples



(a) Optimal scheduling. Both NAMs will be accessed through both links

(b) Suboptimal scheduling. Both NAMs can only be accessed through one link

Fig. 4.24 Impact of node scheduling on NAM accessibility

routes and the best possible XOR set assignment. The figure points out that the NAM checkpointing performance can be significantly affected by simply scheduling the 'wrong' nodes. The impact of suboptimal mapping on performance will be evaluated in Chapter 5.

For CR, libNAM is also responsible to pad data chunks with zeros up to the next 16 Byte boundary which would otherwise violate the NAM access granularity.

The NAM address space of 2GB per NAM can be allocated as a single or multiple contiguous memory regions. Allocations are granted, managed, and released by a dedicated NAM manager.

4.7.3 NAM Manager

Before a user application can access a NAM it must obtain an allocation. These allocations are managed by the NAM manager. It is implemented as a system service

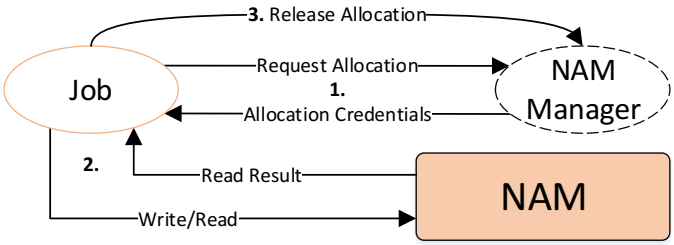


Fig. 4.25 NAM manager interaction: A job requests space on a NAM via the NAM manager. Allocations are either shared or exclusive and may be used to read and write a NAM until released

which returns a handle upon a successful allocation request. This process is depicted in Figure 4.25. Even if a checkpoint or restart process is running, any non-allocated NAM address space can still be allocated and read or written when CR is not using the full memory address space. However, only one CR process may be running at a time.

4.8 NAM Summary

This chapter introduced the NAM hardware prototype and described the implementation and individual functional units of the FPGA design in detail. A theoretical analysis of the estimated performance identified the expected bottlenecks, and there are several recommendations for improving these. In particular, the EXTOLL FPGA link implementation for single link operation and the NAM protocol conversion units for two link operation require optimization. The estimated performance will be validated with in-system measurements using real hardware in the next chapter.

CHAPTER 5

NAM Performance Evaluation

This chapter presents the performance of the NAM prototype using real hardware setups. Various microbenchmarks were executed to characterize bandwidth and latency for reading, writing, and Checkpoint/Restart. One example application was run on the DEEP-ER SDV to cover the full set of functionality under real world conditions.

5.1 Read/Write Microbenchmark Results

A basic PUT/GET microbenchmark was executed to measure the read, write, and simultaneous read/write performance. All of the figures below use a logarithmic base 4 scale on the x-axis. A comprehensible labeling is used for the message sizes from 16 Byte to 1 GB. To eliminate initial software overhead, each message size is requested 5000 times and the time between start and completion is measured. In reference to the theoretical NAM performance evaluation in Section 4.4, the figures also depict the theoretical bandwidth limit where applicable. The presented results highlight the bandwidth and latency for a single NAM link, and the bandwidth for accessing both NAM links at the same time, respectively.

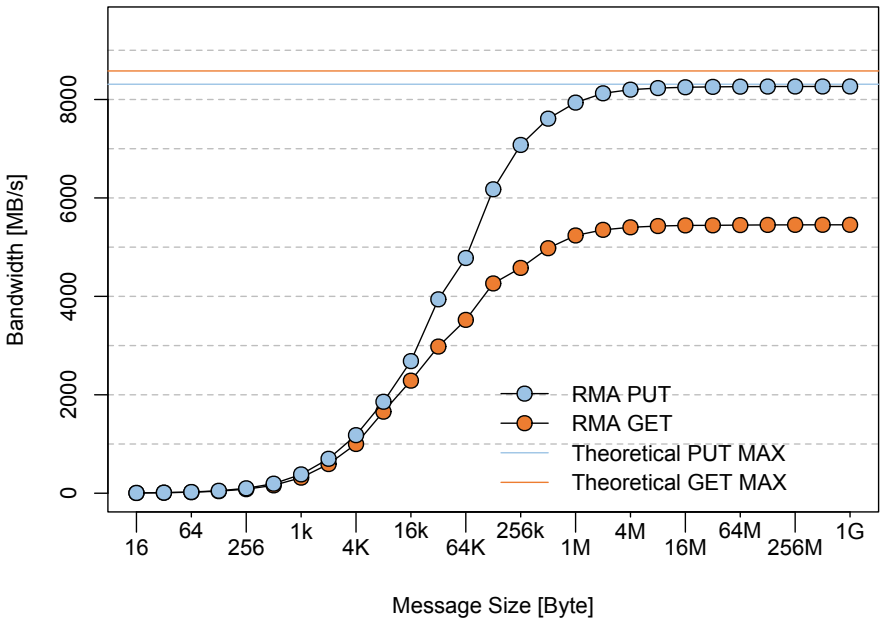


Fig. 5.1 Single link PUT/GET bandwidth

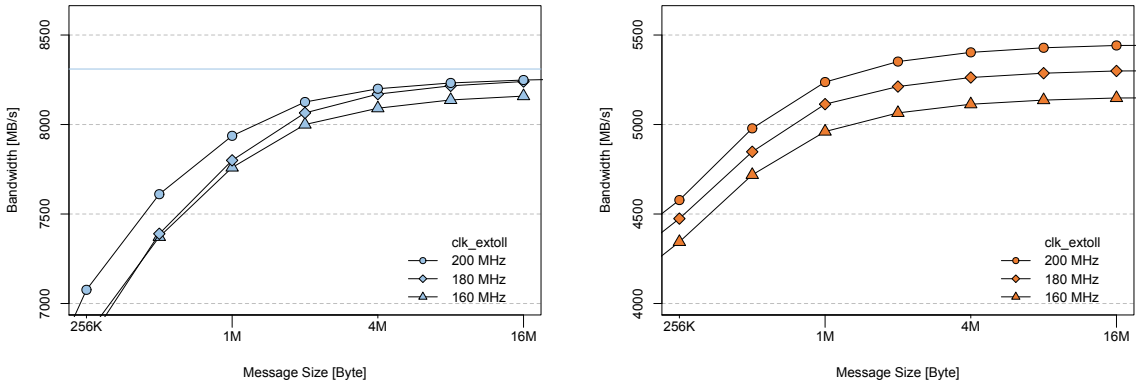
5.1.1 Single Link Performance

In a first measurements series one NAM was connected via one link to an EXTOLL ASIC. The results for bandwidth are presented first.

5.1.1.1 Bandwidth

Figure 5.1 shows the PUT and GET bandwidth for one NAM link. It can be seen that the achievable bandwidth is linked to the message size. Larger messages initiated by a software descriptor lead to less software overhead and network descriptor translation effort. The PUT bandwidth peaks at 8.25 GB/s, close to the theoretical limit of 8.31 GB/s. The performance of GET requests on the other hand is surprisingly low with 5.5 GB/s, about 3 GB/s less than the theoretical limit of 8.58 GB/s. To understand this behavior, it is mandatory to recap some of the lessons learned in the previous chapter.

The operating frequency of the EXTOLL FPGA link in the NAM has been identified as an important factor that affects performance. To quantify its impact on PUT and GET bandwidth, two additional measurements were conducted with three different frequencies. As Figure 5.2 shows the maximum bandwidth correlates with the frequency. The impact on GET requests is slightly higher than on PUTs which is already close



(a) PUT (Y-axis range: 7000-8500 MB/s) **(b)** GET (Y-axis range: 4000-5500 MB/s)

Fig. 5.2 Single link PUT/GET bandwidth in dependency of the NAM EXTOLL Link operating frequency clk_extoll

to the theoretical maximum. The figures point out that decreasing the frequency of the EXTOLL link which was implemented at 200 MHz does not substantially affect bandwidth. This leads to the conclusion that increasing the frequency would not be sufficient to take GET performance to a maximum here.

In a second measurement the NAM was configured to utilize all four EXTOLL Virtual Channels (VCs) for GET responses. Figure 5.3 shows that using more VCs takes the GET bandwidth close to its theoretical limit, still with a slight dependency on the EXTOLL FPGA link operating frequency. The reason why PUT and GET are affected unequally by the credit limitation is that the NAM may only use 31 credits per VC to send packets (GET responses), while an ASIC has up to 58 for requests (PUTs) to the NAM (see Section 4.2.5 and Section 4.3.3.2). Although the usage of multiple VCs is highly recommended, the current implementation of libNAM is not capable to handle more than one VC. It can cause responses to return out of order and must be handled separately.

5.1.1.2 Latency

The NAM PUT and GET latency measurements are depicted in Figure 5.4. The lowest NAM access latency starts with 1.8 µs for PUT and 2.8 µs for GET requests respectively. The values are increasing with larger packet sizes accordingly. The actual numbers and a breakdown of the individual latency contributors can be found in Table 5.1. The values for the ASIC to ASIC communication reference were taken from [49]. It can be seen that accesses to the NAM have a similar, yet slightly lower

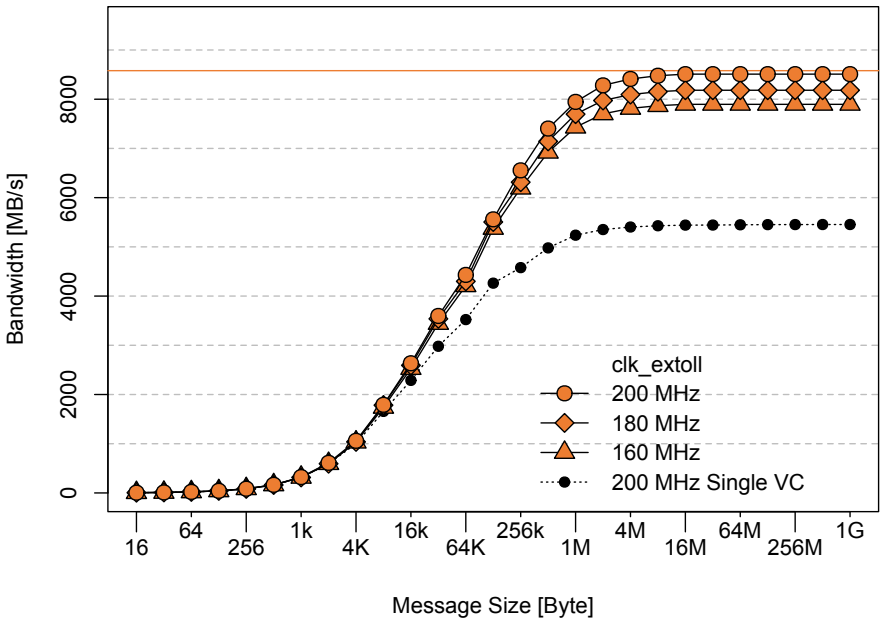


Fig. 5.3 Single link GET bandwidth with four Virtual Channels

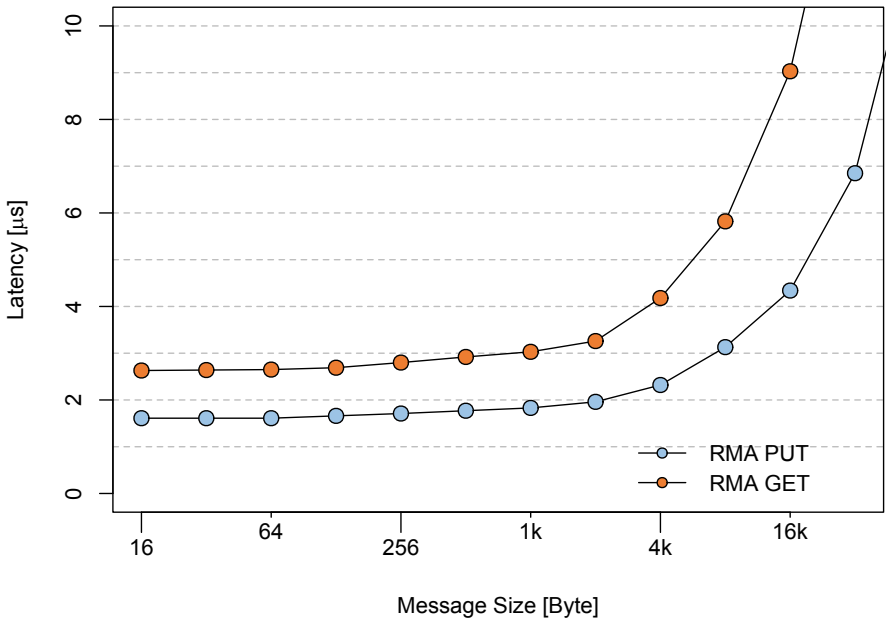


Fig. 5.4 Single link PUT/GET latency

Table 5.1 Overall ASIC-NAM and ASIC-ASIC PUT and GET latencies. Breakdown by sub-operation and functional unit delays

Sub-operation / functional unit	Delay [ns]	# for PUT ASIC-ASIC	# for PUT ASIC-NAM	# for GET ASIC-ASIC	# for GET ASIC-NAM
Software Overhead	300	1	1	1	1
PIO Write	150	1	1	1	1
ATU Translation	70	2	1	2	1
DMA Read	350	1	1	1	0
RMA Unit Delay	50	2	1	3	2
Network Trip ASIC-ASIC	650	1	0	2	0
Network Trip ASIC-NAM	700	0	1	0	2
DMA Write	200	1	0	2	1
NAM Logic Delay	80	0	1	0	2
HMC Read	200	0	0	0	1
HMC Write	80	0	1	0	0
Overall Latency [ns]		1890	1780	2790	2780

latency than packets between two EXTOLL ASICs. The network trip latency for NAM accesses is only slightly higher although the FPGA operating frequency is much lower. The reason for this is that the NAM does not have a network crossbar which saves parts of the delay that exists in the ASIC. For PUT request, an additional DMA write and ATU translation are avoided but latency through the slow FPGA clock domains is added. Although GET requests require two network trips to complete (round-trip, to the NAM and back), their latency is by far less than for two PUTs. This can be explained by the fact that several delay contributors such as the software overhead appear only once. Unfortunately, the advantage of avoiding PCIe for DMA reads on the NAM is nullified by the high delays in the FPGA.

5.1.2 Two Link PUT/GET Bandwidth

To measure the performance of both NAM links simultaneously, two EXTOLL ASICs were connected with one link each to a NAM. The MPI benchmark executes sequential reading and writing and the final result is calculated by aggregating the individual

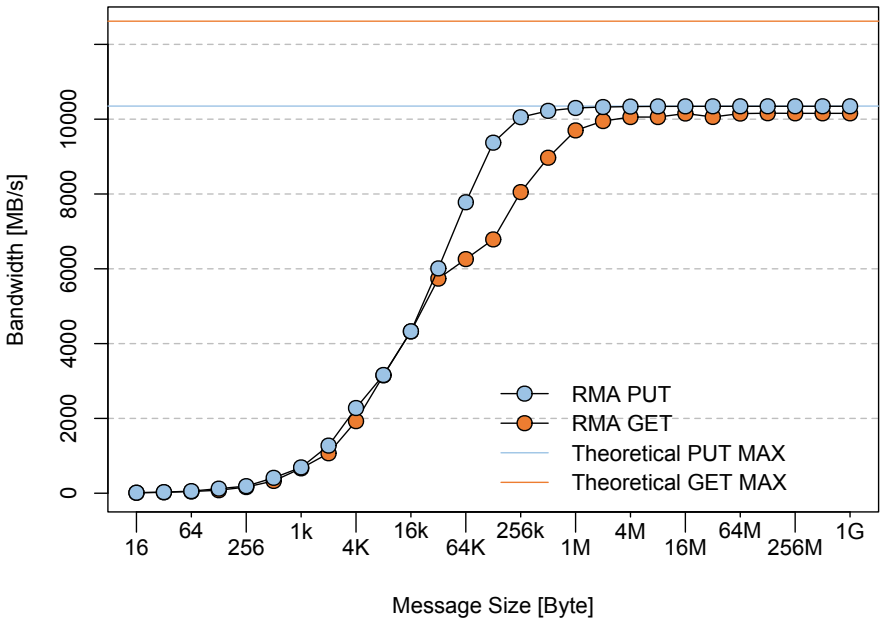


Fig. 5.5 Two link PUT/GET bandwidth

bandwidths. The results are depicted in Figure 5.5. While the PUT bandwidth peaks at the theoretical limit of 10.35 GB/s, the GET Response bandwidth again suffers from the credit limitation of the EXTOLL FPGA links. Eventually the total GET bandwidth on two links settles to 10.15 GB/s, approximately twice the single link bandwidth.

5.1.3 Analysis and Improvements

The estimated values and actual measurement results for PUT and GET operations are summarized in Table 5.2. Based on this comparison and the findings in the previous sections, two key observations and improvement recommendations can be derived.

PUT performance is as expected While the EXTOLL Link itself is the bottleneck in single link operation, the bandwidth limiting component shifts to the NAM internal HTL functional unit when using two EXTOLL Links. Both values measured, however, match the theoretical estimation. Currently the HTL converts a large RMA packet to multiple, smaller sized HMC packets and splitting is subject to various restrictions. A possible solution to increase the HTL performance is to substitute the packet conversion logic by a cache-like unit, for example a

Table 5.2 NAM bandwidth comparison: estimated versus actual measured

Operation	Bandwidth Estimated [GB/s]	Bandwidth Measured [GB/s]
PUT	8.31	8.27
PUT 2 Links	10.35	10.35
GET	8.58	5.54
GET 2 Links	12.62	10.15

set-associative cache with a line size of 496 Byte. The NAM will then be able to select and write-out larger HMC packets more efficiently. Other characteristics such as the cache eviction strategy are implementation specific and can be set to either optimize for area, bandwidth, or power consumption. The latter option would not only reduce the FPGA power footprint but also HMC dynamic power, which is a significant fraction of its overall consumption.

GET performance falls short of expectations For single link operation a bandwidth drop of 3 GB/s over the theoretical limit can be observed. The cause for this lack of performance has been identified with the FPGA EXTOLL Link operating frequency, and even more critical, the number of credits available for packet transmission. Hence, future link ASIC link implementations should increase the credit count to enable best performance with FPGAs and other devices that run on slower clocks. It is the easiest way to further scale the bandwidth as increasing the operating frequencies will cause implementation issues in the FPGA. Alternatively, the NAM could be implemented as an ASIC to eliminate all of the issues mentioned above.

5.2 Checkpoint/Restart

The NAM CR functionality to speed-up the creation of parity checkpoints has been evaluated in the DEEP-ER SDV. Here, two NAMs are connected with both links to a 16-node torus type network. The logical NAM placement within the topology was carefully chosen to balance out the distances from each node to the nearest NAM. A set of microbenchmarks was implemented to independently evaluate the performance of creating and restarting from checkpoints. These measurements are complemented by an application benchmark with one of the DEEP-ER applications.

5.2.1 Microbenchmark Results

The following set of microbenchmarks analyzes the bandwidth of the NAM CR and its scaling behavior in the DEEP-ER SDV from 1 to 16 nodes with 4 processes each. The checkpoint sizes range from 4 KB up to 2 GB per node. The benchmarks directly call libNAM CR functions without involving an additional layer such as SIONlib, and each process is treated as an independent rank. Hence a maximum of 64 checkpoints are created and evenly assigned to both NAMs (maximum 32 checkpoints per NAM).

5.2.1.1 Checkpointing

The first benchmark measures the overall bandwidth for creating XOR parity checkpoints. A root process configures the NAM CR unit and distributes the job to all participating ranks. Each rank then creates a checkpoint and informs the NAM in order to fetch the data and generate the parity. The bandwidth measurement is started as soon as the MPI job starts and stopped when all ranks have received a notification that the parity has been generated. The actual checkpointing bandwidth is calculated using the total amount of data that has been processed divided by the time the process took, which includes MPI start-up times and synchronization. The results of this benchmark are depicted in Figure 5.6. It can be seen that the bandwidth scales with the number of available nodes.

For one participating node, only one NAM is utilized and only one link of this NAM is accessed since there is a static route between the two endpoints. The resulting peak bandwidth is 6.2 GB/ which is less than what has been measured for PUT requests from a node to the NAM. This surprises as the NAM issues GET requests, and GET responses traveling back to the NAM are very similar to PUTs with respect to how they are handled by the EXTOLL network. The reason for this disparity is software synchronization overhead and the generation of the XOR parity which is then also written to the HMC. It is reasonable to include this overhead in the measurements since it is part of the overall CR process.

With two nodes the effective bandwidth is already more than doubled with 14 GB/s as now both NAMs are involved and the software overhead remains at a comparable level. Adding more nodes to the checkpointing process eventually leads to a bandwidth saturation at 24.8 GB/s with 16 nodes. At a first glance this result surprises as it states that the bandwidth per NAM, assuming an equal distribution, is $\frac{24.8\ \text{GB/s}}{2\ \text{NAMs}} = 12.4\ \text{GB/s}$. This is higher than what has been measured for writing data to a NAM via both links.

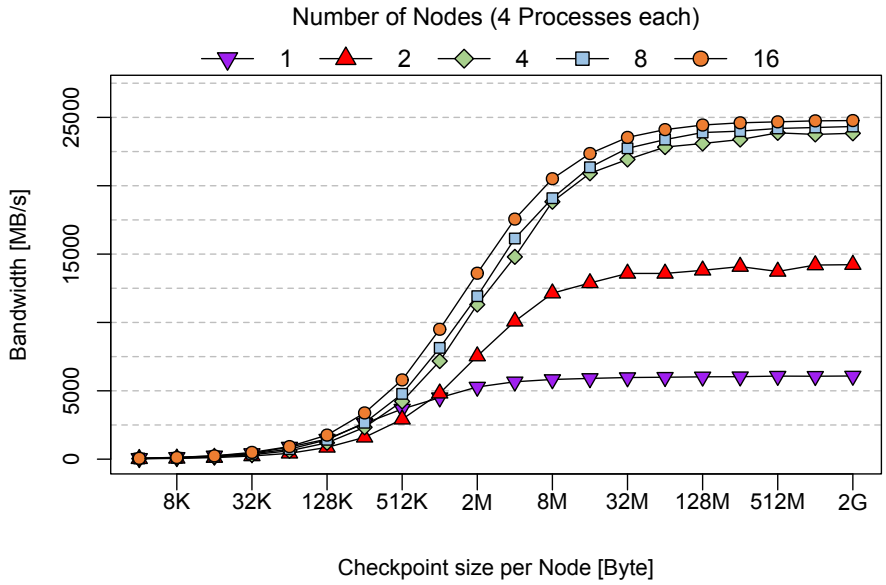


Fig. 5.6 XOR checkpointing bandwidth with 2 NAMs in the DEEP-ER SDV. 4 processes per node with one checkpoint per process

However, the theoretical NAM bandwidth analysis in Section 4.4 pointed out that the bottleneck for a two link operation sits in the HTL protocol conversion logic. In case of Checkpoint/Restart this module is completely avoided except for the task of writing out the XOR parity to the HMC. All other data is directed to the CR layer which operates at a higher throughput (17.54 GB/s) than two EXTOLL links can deliver (16.62 GB/s). Achieving even higher bandwidths for checkpointing remains difficult due to natural overhead of generating and storing the XOR parity, and process synchronization among participating nodes.

5.2.1.2 Restart

Benchmarking a restart requires that a XOR parity has already been generated. Hence, a checkpoint is first created following the scheme presented in the previous section. The bandwidth measurement is started as soon as the root process informs the NAM that a rank failure has occurred and stopped after the failed rank has entirely retrieved its missing checkpoint. Figure 5.7 shows that restart scales similarly to checkpointing for an increasing number of participating nodes. The resulting bandwidths, however, are

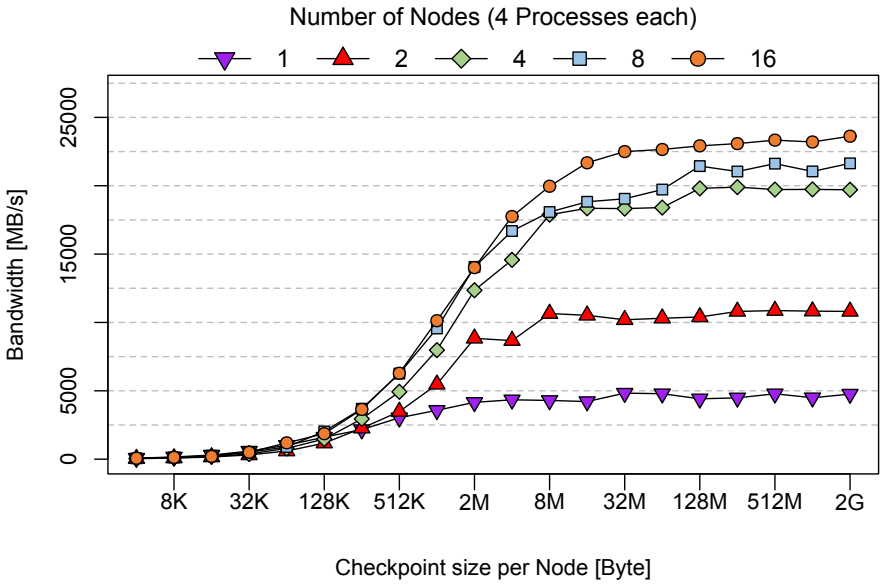


Fig. 5.7 XOR restart bandwidth with 2 NAMs in the DEEP-ER SDV. 4 processes per node with one checkpoint per process

continually lower than for checkpointing. The reason for this behavior is the additional read process to fetch the missing checkpoint after reconstruction has finished.

5.2.1.3 Impact of XOR Set Mapping on CR Performance

One important property that affects CR performance is the assignment of nodes to a XOR set, or more specific the mapping of ranks to one of the two NAMs. The libNAM library currently maps nodes to a set in pseudo-random fashion and the actual topology and routing setup is not considered. As Section 4.7.2 highlighted there exist good and bad mappings for the same node/routing/NAM setup. The measurements so far were executed with manually assigned XOR sets. This is reasonable for a system such as the DEEP-ER SDV. For larger systems and many different applications, however, it is up to libNAM to form these sets. Therefore, it is necessary to measure the performance impact of the mapping scheme.

Figure 5.8 compares the checkpointing bandwidth for two different mappings with 4 nodes. It shows that the potential performance loss for a bad mapping scheme is significant. Therefore, with the current libNAM implementation and without any additional effort it is not guaranteed that always the best mapping is provided. In

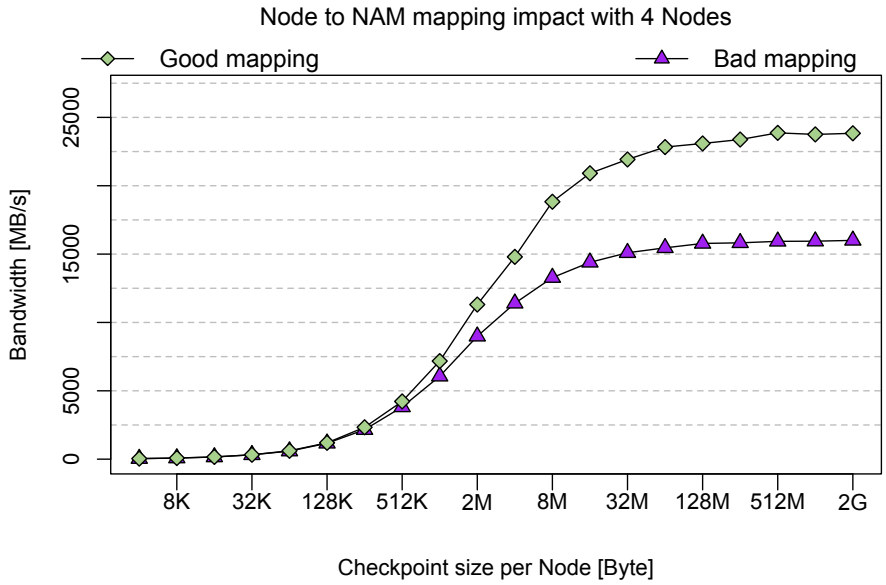


Fig. 5.8 Impact of XOR set to NAM mapping on achievable bandwidth

addition, it can also be due to the job scheduler that a bad mapping is inevitable. In this case the user is responsible to reserve nodes where the routing is guaranteed to target all available NAM links.

5.2.2 Application Performance

One DEEP-ER application was selected to ultimately compare the NAM checkpoint/restart approach with the existing SIONlib-Buddy checkpointing scheme.

iPic3D [103] is a space weather application developed by the Katholieke Universiteit (KU) Leuven. It is meant to deepen the understanding and increase the forecasting accuracy of the impact of sun solar emissions on the earth weather. The application itself operates on two distinct items: computation-intensive particle operations, and communication-dominated inter-particle field calculations. This perfectly suits the cluster-booster architecture of the DEEP-ER system and makes iPic3D a perfect candidate to proof its concept and also to evaluate the NAM as CR target.

In order to run benchmarks, the application code was modified to only execute the checkpointing portion, leaving out actual computation because it is irrelevant for the measurements. iPic3D operates on particle and cell datatypes, where each cell is

approximately 64 KB in size and consists of 1024 particles. Each scenario, NAM-XOR[1] and SIONlib-Buddy[2], was run for various checkpoint sizes on 2 to 16 nodes with four processes per node and a total of 2 XOR sets. Participating nodes were evenly assigned to the two NAMs. Runs were executed 20 times, with 10 checkpoints per iteration, and the total runtime was taken to even out measurement errors.

Two types of scalability were evaluated: weak scaling, which means that the problem size linearly increases with the number of nodes, and strong scaling, where the problem size stays constant but the number of processes and nodes varies.

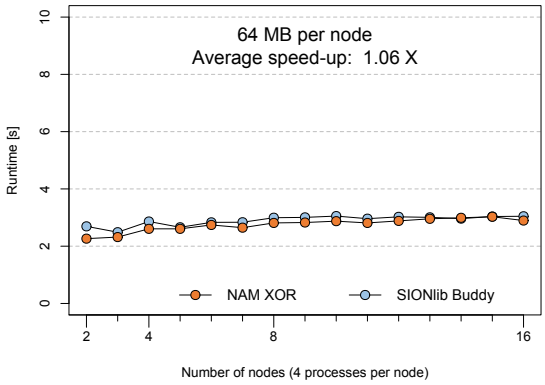
5.2.2.1 Weak Scaling

The first set of benchmarks measures the weak scaling behavior. Checkpoint sizes range from a total of 64 MB (16 MB per process) per node up to 2 GB (512 MB per process), which is the maximum NAM checkpoint size. The results are depicted in Figure 5.9 and the values reflect the average time out of 20 runs. The best case runtimes are slightly better and the worst case runtimes may be significantly higher due to file system and network congestion. Note that the Y-axis range changes.

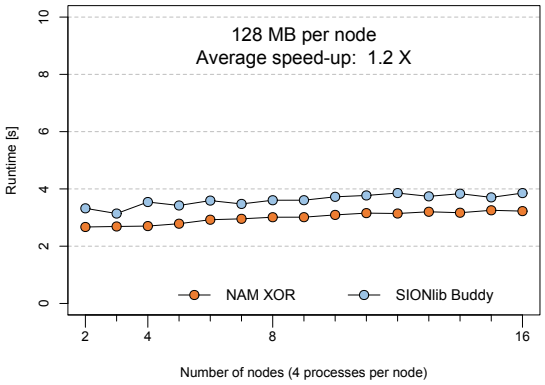
The results clearly show an advantage with the NAM approach and the achievable speed-ups range from 1.06X to approximately 2.1X, i.e. with two NAMs in the system, checkpoints may be created 2.1 times faster than with SIONlib-Buddy. For a given dataset size per node it can be seen that the runtimes on the NAM remain almost constant. This is an indication that the NAM internal CR request mechanism is well-balanced. It also shows that two links per NAM provide sufficient bandwidth for at least 8 nodes. However, at a certain point only two NAMs will not be able to support the link bandwidth of additional nodes and more NAMs should be added to the system.

Noticeably, the speed-up continually increases with larger checkpoint sizes. The reason for this is that with larger checkpoints, SIONlib-Buddy writes more and more data to the local NVMe drives which is much slower than moving the data over EXTOLL to the NAM. It is expected that the achievable speed-up will further increase for larger datasets but such measurements were not feasible due to the NAM memory capacity of 2 GB.

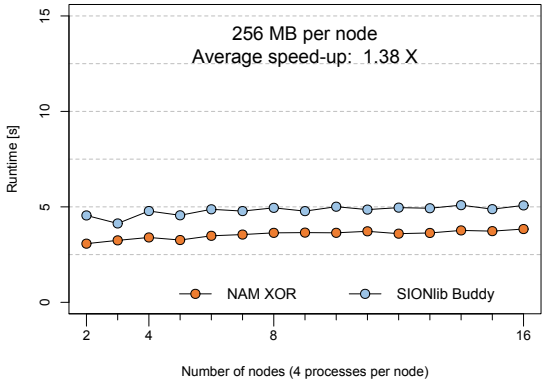
[1] For more information on how the NAM creates checkpoints see Section 4.5.4 and Section 4.5.6
[2] For more information on how SIONlib-Buddy creates checkpoints see Section 4.5.1



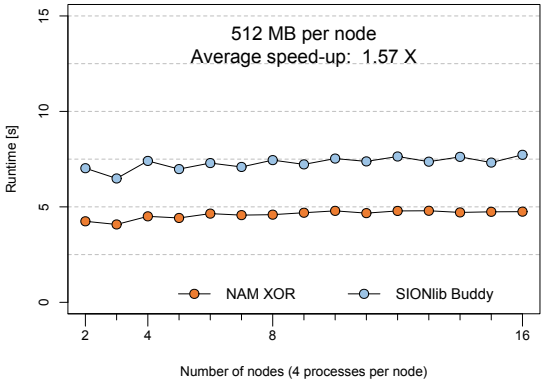
(a) 256 cells per process / 64 MB checkpoint size per node



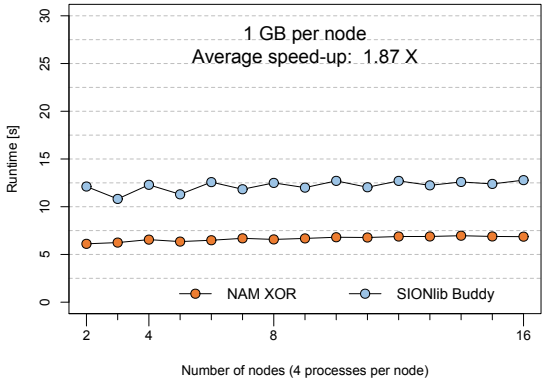
(b) 512 cells per process / 128 MB checkpoint size per node



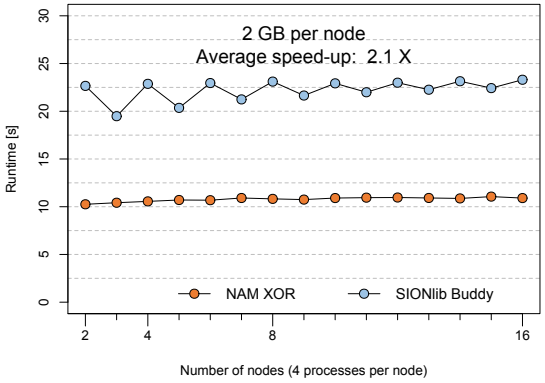
(c) 1024 cells per process / 256 MB checkpoint size per node



(d) 2048 cells per process / 512 MB checkpoint size per node

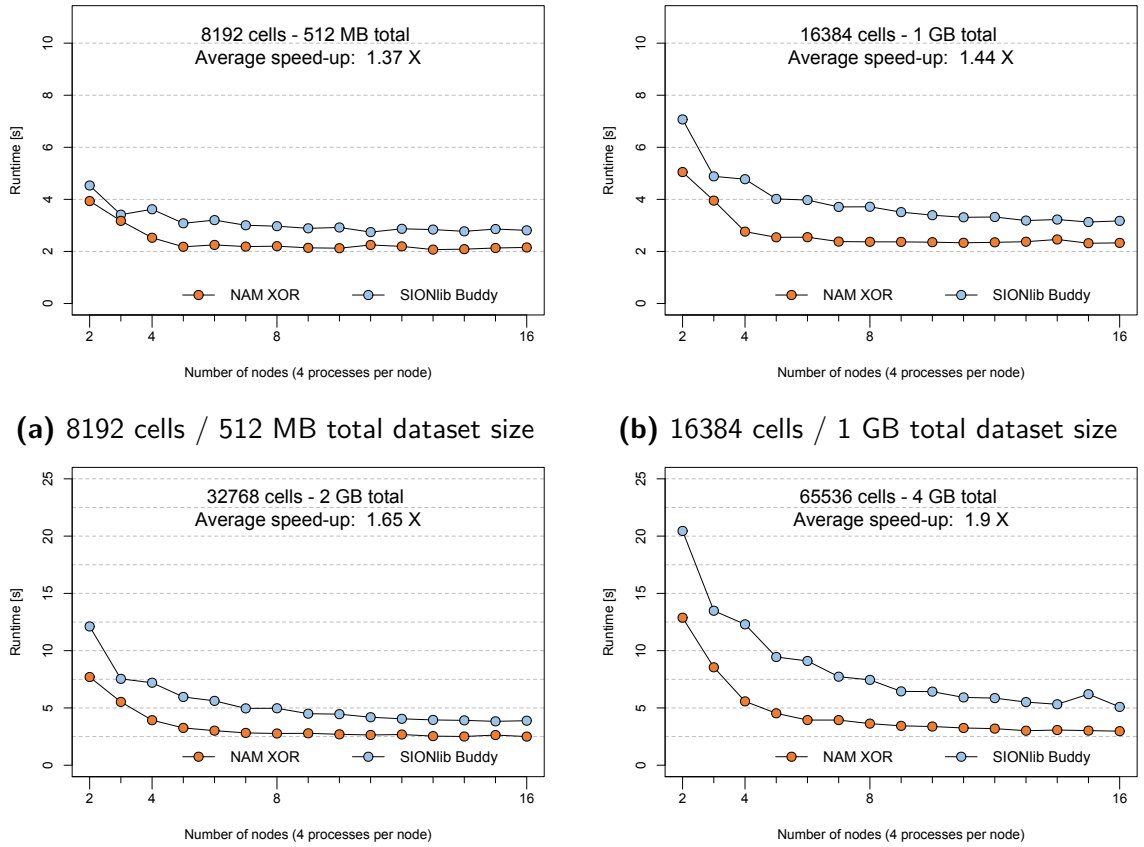


(e) 4096 cells per process / 1 GB checkpoint size per node



(f) 8192 cells per process / 2 GB checkpoint size per node

Fig. 5.9 Xpic3d application performance comparison for weak scaling: NAM-XOR versus SIONlib-Buddy. Note the variable Y-Axes



(a) 8192 cells / 512 MB total dataset size

(b) 16384 cells / 1 GB total dataset size

(c) 32768 cells / 2 GB total dataset size

(d) 65536 cells / 4 GB total dataset size

Fig. 5.10 Xpic3d application performance comparison for strong scaling: NAM-XOR versus SIONlib-Buddy. Note the variable Y-Axes

5.2.2.2 Strong Scaling

The strong scaling behavior was measured on 2 to 16 nodes on the SDV. Figure 5.10 depicts the results for 4 different dataset sizes, where the largest dataset is 4 GB in total, or 2 GB per node. This limitation is again due to the limited NAM capacity.

As for weak performance, NAM-XOR performs better and achieves speed-ups from 1.37X to 1.9X. It can be seen that the runtimes for the NAM significantly decrease from 2 to 4 nodes. The reason for this is that with two nodes, both NAMs with only one link each may be accessed. Already with 4 nodes all NAM links are utilized[3]. From this point the performance remains mainly constant with only a very slight decrease since the inter-node MPI communication and NAM configuration overhead increases.

[3] Node selection is often handled by the job scheduler. A proper selection is critical for the NAM performance. See also Section 4.7.2.

5.3 Performance Summary

This chapter evaluated the NAM prototype and software stack using micro- and application benchmarks in real system setups. It was shown that read and write operations to the NAM perform reasonably well, although several limitations as discussed in the theoretical performance evaluation were discovered. It turns out that the EXTOLL Link FPGA implementation has to be improved to further increase the single link performance, in particular for GET requests. This affects the operating frequency constraints, and more important the credit-based flow control mechanism. Two-link read/write performance, on the other hand, suffers from protocol translation requirements and overhead, and other approaches need to be developed to reduce this penalty.

It was pointed out that the NAM access latency is very close, yet slightly better to what can be achieved between two EXTOLL ASICs. Increasing the operating frequency in all parts of the design would ultimately put the NAM at an advantage. For an ASIC implementation it is expected that both, bandwidth and latency would significantly improve.

An additional set of microbenchmarks was executed to measure the Checkpoint/Restart performance. It became clear that checkpoint and restart with two NAMs in the 16 node SDV test system result in tremendous bandwidth and a good scaling behavior. As NAMs can be attached to any unused EXTOLL Link in the system, the overall memory capacity and bandwidth perfectly scale with the system size and the number of NAMs attached to it. For future use, however, the libNAM library and EXTOLL EMP application should be made aware of the topology to allow the best possible node-to-NAM XOR set mapping.

Finally, application benchmarks with one of the DEEP-ER applications showed that checkpointing using the NAM is superior to SIONlib-Buddy. With a maximum speed-up of 2.1X for weak scaling and 1.9X for strong scaling, NAM-XOR is able to significantly reduce the overhead of fault tolerance features of today's and future large-scale systems.

CHAPTER 6

Conclusion and Outlook

The goal of this work was to develop a hardware prototype that is able to mitigate the negative effects of three common problems in today's and future large-scale systems. These are the memory interface and performance, the rapidly increasing amount of inter-process communication, and fault tolerance. Network Attached Memory was developed and presented as an innovative solution. It is a dedicated device that speeds-up collective operations and provides shared memory access at network bandwidth. As a first use case the NAM serves as target for commonly deployed Checkpoint/Restart mechanisms. The resulting hardware prototype provides high-performance interconnection network interfaces and implements the emerging memory technology Hybrid Memory Cube. The NAM design was fully prototyped in an FPGA and the excellent performance results show that the NAM is able to speed-up the creation process of checkpoints in a 16 node test system by a factor of 2.1X.

The first contribution comprises an overview over memory technology and interface evolution, and typical communication methods and patterns in distributed systems. It became clear that the memory interface must be optimized in order to keep up with the latency and bandwidth requirements of multi-core architectures. Inter-process and in particular inter-node communication already today take up a large amount of the overall application runtimes. Since the general message passing scheme is not expected to change in the near future, it is either desirable to reduce the number of messages that are sent or otherwise to speed-up messaging and commonly used

collective operations. The overview is complemented by an introduction to fault tolerance which has become a major concern in today's and future large-scale systems. Checkpointing introduces additional application overhead and most often stresses the memory interface and the interconnection network due to communication and synchronization. Hence, fault tolerance will also benefit from optimizing either of these two components. Finally, power and energy play an increasingly important role, especially since the total power budget for an Exascale system should not exceed 20 MWatt and today's fastest supercomputers are already close to this limit. The NAM can help to increase the energy efficiency, allowing systems to consume less power or to execute more actual work in a given time period.

The next contribution presented the Hybrid Memory Cube technology and interface as one approach to overcome the bandwidth, scalability, and power efficiency issues of the parallel memory interface. The HMC architecture is thoroughly analyzed and the obtained insights contributed to the development of an HMC host controller, which has become a popular and widely used open-source initiative. This development enabled the evaluation of HMC performance and power characteristics in an FPGA test system. The results clearly emphasize the motivation to adapt serial and abstracted interfaces to enable memory access parallelism and the independent development of memory technology and its interface.

The development of the NAM prototype hardware and FPGA design was described in detail. The PCB was developed as a standard height PCIe form factor card that can be plugged into common PCIe slots. It provides interfaces to directly connect up to 2 EXTOLL high-performance NICs and integrates a 2 GB HMC device. This development process was driven by the vision initially formulated in the introduction and the use case as checkpointing target in the DEEP-ER project. The architectural contributions comprise an FPGA implementation of the native EXTOLL ASIC network link, protocol conversion logic, and an application-specific Checkpoint/Restart functional unit. The NAM design operates in several clock domains with 200 MHz for the EXTOLL links, 220 MHz for Checkpoint/Restart, and 312.5 MHz for the remaining logic at more than 60 % LUT utilization in the Virtex 7 FPGA. A theoretical performance evaluation identified potential bottlenecks and served as reference for the in-system validation. NAM read/write access, allocations, and Checkpoint/Restart functions are orchestrated by libNAM, a user-level API derived from existing EXTOLL libraries.

The final contribution of this work evaluated the presented hardware and software components in a real 16 node system. Although the theoretical performance evaluation

and microbenchmark results have identified existing bottlenecks for reading and writing, checkpointing with the NAM shows an excellent 2.1X speed-up over the SIONlib-Partner approach which is currently deployed in the DEEP-ER system.

In summary, the vision that has been formulated in the introduction of this work was successfully transferred to a hardware prototype. The performance evaluation of the developed hardware and software components undoubtedly prove that the presented approach is able to reduce inter-node communication and to relieve the memory interface in general, while improving state of the art fault tolerance mechanisms of large-scale systems. These outstanding contributions were publicly recognized and an enhanced NAM device will be developed in the follow-up DEEP-EST (Dynamical Exascale Entry Platform - Extreme Scale Technologies) project.

6.1 Improvements

For a future implementation of the NAM, several possible improvements for the following components were identified:

HMC Link

Due to the limited number of the available FPGA transceivers, the HMC evaluation only covered accesses through 1 out of 4 possible links. A different FPGA with more transceivers and a suitable PCB would be required to connect more than one link. Also, the Xilinx Virtex 7 GTH transceiver do not support 15 Gbps line-rates and a newer FPGA generation (e.g. Xilinx Ultrascale) device would be required. For both, multi-link and 15 Gbps operation, it is expected that the achievable bandwidth and energy efficiency will further improve.

NAM Hardware and FPGA Design

The NAM design unveiled only very few weaknesses, and especially the read performance lags behinds the expectations. This is mainly due to the fact the credits in the EXTOLL network protocol are exchanged too slow when communicating with an FPGA that runs at relatively low operating frequencies. The complex FPGA logic prohibits faster clock domains and the number of credits the EXOLL ASIC can provide is fixed. Hence,

the only way to alleviate bandwidth throttling is to make use of all available traffic classes as provided by the EXTOLL network protocol.

For writing and reading in 2-link operation, the FPGA design internal protocol conversion units have been identified as performance limiting factor. Currently, EXTOLL network packets are directly translated to (many smaller) HMC packets. The main issue here are the different access and address granularities, and packet sizes. A cache-like unit would allow writing 'cache-lines' with a maximum size of the network MTU to an internal buffer array, decoupling the HMC-sided access units which may take data and return GET responses in a more efficient way. Although there are many considerations left open, such as associativity and eviction strategies, this approach would make protocol translation much easier.

For Checkpoint/Restart, a clear drawback of the NAM is the volatility of the memory array. Hence a future NAM implementation should combine fast DRAM access (e.g. HMC) and a second level, non-volatile storage such as NAND. It is also desirable to partition the available address space into 2 equally sized regions so that even in the process of checkpoint creation a copy of the stable XOR parity may always be preserved.

NAM Software

In order to utilize the traffic classes mentioned above, a future libNAM implementation should issue GET requests on alternating traffic classes. This results in a slightly more complex software component as responses may return out of order. However, it also unlocks the full network bandwidth. A second libNAM optimization is to use topology and routing information provided by EMP. It could be used to always create the best possible XOR set mapping for checkpointing.

6.2 Outlook

The NAM approach will soon be taken to its next level in the European funded DEEP-EST project. Here, the existing NAM prototype will be extended by multiple Terabyte of non-volatile memory. Additionally, a newer and larger FPGA to accommodate more complex processing capabilities will be introduced. A more general use case toward MPI collective operations will motivate even more application developers to make use of this novel concept. A performance boost is in particular expected for

applications that make use of non-blocking collective operations which are supported by the latest MPI 3 standard. These operations allow host processors to continue program execution while the NAM collects the required information, carries out the collective operation, and then re-distributes the results.

APPENDIX

Acronyms

AMC	Active Memory Cube
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
ATU	Address Translation Unit
BE	Bandwidth Engine
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CR	Checkpoint/Restart
CRC	Cyclic Redundancy Check
DDR	Double Data Rate
DEEP	Dynamical Exascale Entry Platform
DEEP-ER	Dynamical Exascale Entry Platform - Extended Reach
DEEP-EST	Dynamical Exascale Entry Platform - Extreme Scale Technologies
DIMM	Dual In-line Memory Module
DMA	Direct Memory Access
DOE	U.S. Department of Energy
DRAM	Dynamic Random-Access Memory
ECC	Error Correction Code
EMP	EXTOLL Management Program
EOP	End Of Packet
FIFO	First In - First Out

FLIT	Flow Unit
FPGA	Field Programmable Gate Array
FRP	Forward Retry Pointer
FTI	Fault Tolerant Interface
GDDR	Graphics Double Data Rate
HBM	High Bandwidth Memory
HMC	Hybrid Memory Cube
HMCC	Hybrid Memory Cube Consortium
HPC	High Performance Computing
HTL	HMC Transaction Layer
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
I/O	Input/Output
IP	Intellectual Property
JTAG	Joint Test Action Group
KNL	Intel Knights Landing
LED	Light-Emitting Diode
LGPL	Lesser General Public License
LPDDR	Low Power Double Data Rate
LUT	LookUp Table
MCDRAM	Multi Channel DRAM
MPI	Message Passing Interface
MTBF	Mean Time Between Failure
MTU	Maximum Transmission Unit
MUX	Multiplexer
NDC	Near-Data Computing
NIC	Network Interface Controller
NAM	Network Attached Memory
NAND	NAND Flash Memory
NTL	Network Transaction Layer
NVMe	Non-volatile Memory Express
PCB	Printed Circuit Board
PCIe	PCI Express
PIM	Processing In Memory
PFS	Parallel File System
RAM	Random-Access Memory

RAS	Reliability, Availability and Serviceability
RF	Register File
RMA	Remote Memory Access or Remote Memory Architecture
RRA	Remote Register File Access
RRP	Return Retry Pointer
SCR	Scalable Checkpoint / Restart
SDV	Software Development Vehicle
SerDes	Serializer / Deserializer
SOP	Start Of Packet
SRAM	Static Random-Access Memory
SSD	Solid State Drive
TSV	Through Silicon Via
VC	Virtual Channel
VPID	Virtual Process Identifier
XBAR	Crossbar

APPENDIX B

List of figures

APPENDIX C

List of tables

APPENDIX R

References

[1] Erich Strohmaier et al. “The TOP500 List and Progress in High-Performance Computing”. In: *Computer* 48.11 (Nov. 2015), pp. 42–49. ISSN: 0018-9162. DOI: 10.1109/MC.2015.338.

[2] Wm. A. Wulf and Sally A. McKee. “Hitting the Memory Wall: Implications of the Obvious”. In: *SIGARCH Comput. Archit. News* 23.1 (Mar. 1995), pp. 20–24. ISSN: 0163-5964. DOI: 10.1145/216585.216588. URL: http://doi.acm.org/10.1145/216585.216588.

[3] Benjamin Klenk and Holger Fröning. “An Overview of MPI Characteristics of Exascale Proxy Applications”. In: *High Performance Computing: 32nd International Conference, ISC High Performance 2017, Frankfurt, Germany, June 18–22, 2017, Proceedings*. Ed. by Julian M. Kunkel et al. Cham: Springer International Publishing, 2017, pp. 217–236. ISBN: 978-3-319-58667-0. DOI: 10.1007/978-3-319-58667-0_12. URL: https://doi.org/10.1007/978-3-319-58667-0_12.

[4] Daniel Dauwe et al. “A Performance and Energy Comparison of Fault Tolerance Techniques for Exascale Computing Systems”. In: *2016 IEEE International Conference on Computer and Information Technology (CIT)*. Dec. 2016, pp. 436–443. DOI: 10.1109/CIT.2016.44.

[5] Nilmini Abeyratne et al. “Checkpointing Exascale Memory Systems with Existing Memory Technologies”. In: *Proceedings of the Second International Symposium on Memory Systems*. MEMSYS ’16. Alexandria, VA, USA: ACM, 2016, pp. 18–29. ISBN: 978-1-4503-4305-3. DOI: 10.1145/2989081.2989121. URL: http://doi.acm.org/10.1145/2989081.2989121.

[6] Malcolm Ware et al. “Architecting for power management: The IBM®; POWER7™; approach”. In: *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. Jan. 2010, pp. 1–11. DOI: 10.1109/HPCA.2010.5416627.

[7] John Shalf, Sudip Dosanjh, and John Morrison. "Exascale Computing Technology Challenges". In: *High Performance Computing for Computational Science – VECPAR 2010: 9th International conference, Berkeley, CA, USA, June 22-25, 2010, Revised Selected Papers*. Ed. by José M. Laginha M. Palma et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–25. ISBN: 978-3-642-19328-6. DOI: 10.1007/978-3-642-19328-6_1. URL: https://doi.org/10.1007/978-3-642-19328-6_1.

[8] Avinash Sodani. "Race to Exascale: Opportunities and Challenges". In: *Keynote at the Annual IEEE/ACM 44th Annual International Symposium on Microarchitecture*. 2011.

[9] Brian Barrett et al. "On the Path to Exascale". In: *Int. J. Distrib. Syst. Technol.* 1.2 (Apr. 2010), pp. 1–22. ISSN: 1947-3532. DOI: 10.4018/jdst.2010040101. URL: http://dx.doi.org/10.4018/jdst.2010040101.

[10] Juri Schmidt and Ulrich Brüning. "openHMC - a Configurable Open-Source Hybrid Memory Cube Controller". In: *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. Dec. 2015, pp. 1–6. DOI: 10.1109/ReConFig.2015.7393331.

[11] Juri Schmidt, Holger Fröning, and Ulrich Brüning. "Exploring Time and Energy for Complex Accesses to a Hybrid Memory Cube". In: *Proceedings of the Second International Symposium on Memory Systems*. MEMSYS '16. Alexandria, VA, USA: ACM, 2016, pp. 142–150. ISBN: 978-1-4503-4305-3. DOI: 10.1145/2989081.2989099. URL: http://doi.acm.org/10.1145/2989081.2989099.

[12] Computer Architecture Group - University of Heidelberg. *openHMC - an Open-Source Hybrid Memory Cube Controller*. [Accessed 28-July-2017]. URL: www.uni-heidelberg.de/openhmc.

[13] Sabrina Eisenreich and Juri Schmidt. *Interview: Experimenting with DEEP-ER NAM Technology*. [Accessed 27-July-2017]. URL: https://insidehpc.com/2014/10/interview-experimenting-deep-er-memory-technology/.

[14] Primeur Magazine. *European exascale projects DEEP-ER and Mont-Blanc to investigate new Exascale technologies*. [Accessed 12-June-2017]. URL: https://www.youtube.com/watch?v=tr_co6vu-4s.

[15] Juri Schmidt. *Network Attached Memory*. [Accessed 12-June-2017]. URL: http://sc16.supercomputing.org/sc-archive/doctoral_showcase/doc_files/drs106s2-file7.pdf.

[16] Gordon E. Moore. "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff." In: *IEEE Solid-State Circuits Society Newsletter* 11.5 (Sept. 2006), pp. 33–35. ISSN: 1098-4232. DOI: 10.1109/N-SSC.2006.4785860.

[17] Robert H. Dennard et al. "Design of ion-implanted MOSFET's with very small physical dimensions". In: *IEEE Journal of Solid-State Circuits* 9.5 (Oct. 1974), pp. 256–268. ISSN: 0018-9200. DOI: 10.1109/JSSC.1974.1050511.

[18] Avinash Sodani. "Knights landing (KNL): 2nd Generation Intel®Xeon Phi processor". In: *2015 IEEE Hot Chips 27 Symposium (HCS)*. Aug. 2015, pp. 1–24. DOI: 10.1109/HOTCHIPS.2015.7477467.

[19] Richard Sites. “It’s the Memory, Stupid!” In: *Microprocessor Report* 10.10 (1996), pp. 2–3.

[20] John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. 5th. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN: 012383872X, 9780123838728.

[21] Richard Murphy. “On the Effects of Memory Latency and Bandwidth on Supercomputer Application Performance”. In: *2007 IEEE 10th International Symposium on Workload Characterization*. Sept. 2007, pp. 35–43. DOI: 10.1109/IISWC.2007.4362179.

[22] David A. Patterson. “Latency Lags Bandwith”. In: *Commun. ACM* 47.10 (Oct. 2004), pp. 71–75. ISSN: 0001-0782. DOI: 10.1145/1022594.1022596. URL: http://doi.acm.org/10.1145/1022594.1022596.

[23] Saud Wasly and Rodolfo Pellizzoni. “Hiding memory latency using fixed priority scheduling”. In: *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. Apr. 2014, pp. 75–86. DOI: 10.1109/RTAS.2014.6925992.

[24] Young Hoon Son et al. “Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations”. In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. ISCA ’13. Tel-Aviv, Israel: ACM, 2013, pp. 380–391. ISBN: 978-1-4503-2079-5. DOI: 10.1145/2485922.2485955. URL: http://doi.acm.org/10.1145/2485922.2485955.

[25] JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. *JEDEC Standard JESD 209-2B: Low Power Double Data Rate 2 (LPDDR2)*. 2010.

[26] JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. *JEDEC Standard JESD 212: GDDR5 SGRAM*. 2009.

[27] Indrani Paul et al. “Harmonia: Balancing Compute and Memory Power in High-performance GPUs”. In: *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*. ISCA ’15. Portland, Oregon: ACM, 2015, pp. 54–65. ISBN: 978-1-4503-3402-0. DOI: 10.1145/2749469.2750404. URL: http://doi.acm.org/10.1145/2749469.2750404.

[28] Guruprasad Katti et al. “Electrical Modeling and Characterization of Through Silicon via for Three-Dimensional ICs”. In: *IEEE Transactions on Electron Devices* 57.1 (Jan. 2010), pp. 256–262. ISSN: 0018-9383. DOI: 10.1109/TED.2009.2034508.

[29] JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. *JEDEC Standard JESD 235A: High Bandwidth Memory (HBM) DRAM*. 2015.

[30] JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. *JEDEC Standard JESD 229-2: Wide I/O 2 (WideIO2)*. 2014.

[31] Joe Macri. “AMD’s next generation GPU and high bandwidth memory architecture: FURY”. In: *2015 IEEE Hot Chips 27 Symposium (HCS)*. Aug. 2015, pp. 1–26. DOI: 10.1109/HOTCHIPS.2015.7477461.

[32] Manish Deo, Jeffrey Schulz, and Lance Brown. *Intel Stratix 10 MX Devices Solve the Memory Bandwidth Challenge*. Tech. rep. Altera, now part of Intel, 2016.

[33] *Samsung V-NAND technology: Yield more capacity, performance, endurance and power efficiency*. Tech. rep. Samsung Electronics, 2014.

[34] Micron Technology, Inc. *3D-NAND*. [Accessed 28-July-2014]. URL: https://www.micron.com/about/emerging-technologies/3d-nand.

[35] Micron Technology, Inc and Intel Corporation. *3D XPoint Technology*. [Accessed 20-July-2017]. URL: https://www.micron.com/about/our-innovation/3d-xpoint-technology.

[36] Micron Technology, Inc. *Hybrid Memory Cube Webinar July 2017*. [Accessed 20-July-2017]. URL: https://www.micron.com/~/media/documents/products/presentation/hmc_webinar_july_2017.pdf.

[37] Hybrid Memory Cube Consortium. *Micron and Samsung Launch Consortium to Break Down the Memory Wall*. Oct 6, 2011. URL: http://www.hybridmemorycube.org/.

[38] Myles G. Watson. "Applications for Packetized Memory Interfaces". PhD thesis. University of Heidelberg, 2014.

[39] Michael J. Miller. "Bandwidth Engine® Serial Memory Chip Breaks 2 Billion Accesses/sec". In: *2011 IEEE Hot Chips 23 Symposium (HCS)*. Aug. 2011, pp. 1–23. DOI: 10.1109/HOTCHIPS.2011.7477493.

[40] Maya Gokhale, Bill Holmes, and Ken Iobst. "Processing In Memory: The Terasys Massively Parallel PIM Array". In: *Computer* 28.4 (Apr. 1995), pp. 23–31. ISSN: 0018-9162. DOI: 10.1109/2.375174.

[41] Patterson, David and Anderson, Thomas and Cardwell, Neal and Fromm, Richard and Keeton, Kimberly and Kozyrakis, Christoforos and Thomas, Randi and Yelick, Katherine. "A case for intelligent RAM: IRAM". In: *IEEE Micro* 17.2 (Mar. 1997), pp. 34–44. ISSN: 0272-1732. DOI: 10.1109/40.592312.

[42] Ravi Nair et al. "Active Memory Cube: A processing-in-memory architecture for exascale systems". In: *IBM Journal of Research and Development* 59.2/3 (Mar. 2015), 17:1–17:14. ISSN: 0018-8646. DOI: 10.1147/JRD.2015.2409732.

[43] Tsuyoshi Hamada and Naohito Nakasato. "InfiniBand Trade Association, InfiniBand Architecture Specification, Volume 1, Release 1.0". In: *International Conference on Field Programmable Logic and Applications*. Citeseer. 2005.

[44] Gregory F. Pfister. "An Introduction to the Infiniband Architecture". In: *High Performance Mass Storage and Parallel I/O* 42 (2001), pp. 617–632.

[45] Jack J Dongarra et al. *LINPACK users' guide*. SIAM, 1979.

[46] Mark S Birrittella et al. "Intel®; Omni-path Architecture: Enabling Scalable, High Performance Fabrics". In: *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. Aug. 2015, pp. 1–9. DOI: 10.1109/HOTI.2015.22.

[47] Mondrian Nüssle et al. "An FPGA-Based Custom High Performance Interconnection Network". In: *2009 International Conference on Reconfigurable Computing and FPGAs*. Dec. 2009, pp. 113–118. DOI: 10.1109/ReConFig.2009.23.

[48] Holger Fröning et al. "On Achieving High Message Rates". In: *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. May 2013, pp. 498–505. DOI: 10.1109/CCGrid.2013.43.

[49] Mondrian B. Nüßle. "Acceleration of the hardware-software interface of a communication device for parallel systems". PhD thesis. Universität Mannheim, 2009.

[50] HPC Advisory Council. *Interconnect Analysis: 10GigE and InfiniBand in High Performance Computing*. [Accessed 26-June-2017]. URL: http://www.hpcadvisorycouncil.com/pdf/IB_and_10GigE_in_HPC.pdf.

[51] Mellanox Technologies. *EDR Infiniband*. [Accessed 26-June-2017]. URL: https://www.openfabrics.org/images/eventpresos/workshops2015/UGWorkshop/Friday/friday_01.pdf.

[52] EXTOLL GmbH. *EXTOLL Technology Overview*. [Accessed 26-June-2017]. URL: http://extoll.de/images/pdf/Extoll_Technology_Overview_2016.pdf.

[53] University of Tennessee. *MPI: A Message-Passing Interface Standard. Version 3.0*. [Accessed 26-June-2017]. URL: http://mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf.

[54] William Gropp et al. "A high-performance, portable implementation of the MPI message passing interface standard". In: *Parallel Computing* 22.6 (1996), pp. 789–828. ISSN: 0167-8191. DOI: http://dx.doi.org/10.1016/0167-8191(96)00024-5. URL: http://www.sciencedirect.com/science/article/pii/0167819196000245.

[55] NOWLAB: Network Based Computing Lab, Ohio State University. *MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE*. [Accessed 26-July-2017]. URL: http://mvapich.cse.ohio-state.edu/.

[56] Edgar Gabriel et al. "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation". In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 11th European PVM/MPI Users' Group Meeting Budapest, Hungary, September 19 - 22, 2004. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 97–104. ISBN: 978-3-540-30218-6. DOI: 10.1007/978-3-540-30218-6_19. URL: https://doi.org/10.1007/978-3-540-30218-6_19.

[57] U.S. DOE. *Characterization of the DOE Mini-apps*. [Accessed 28-July-2017]. URL: https://portal.nersc.gov/project/CAL/doe-miniapps.htm.

[58] Algirdas Avizienis et al. "Basic Concepts and Taxonomy of Dependable and Secure Computing". In: *IEEE Transactions on Dependable and Secure Computing* 1.1 (Jan. 2004), pp. 11–33. ISSN: 1545-5971. DOI: 10.1109/TDSC.2004.2.

[59] Ifeanyi P Egwutuoha et al. "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems". In: *The Journal of Supercomputing* 65.3 (Sept. 2013), pp. 1302–1326. ISSN: 1573-0484. DOI: 10.1007/s11227-013-0884-0. URL: https://doi.org/10.1007/s11227-013-0884-0.

[60] Tezzaron Semiconductor. *Terrazon Semiconductor. Soft Errors in Electronic Memory — A White Paper*. [Accessed 28-July-2017]. URL: http://www.tezzaron.com/about/papers/soft_errors_1_1_secure.pdf.

[61] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. "DRAM Errors in the Wild: A Large-scale Field Study". In: *Commun. ACM* 54.2 (Feb. 2011), pp. 100–107. ISSN: 0001-0782. DOI: 10.1145/1897816.1897844. URL: http://doi.acm.org/10.1145/1897816.1897844.

[62] Bianca Schroeder and Garth Gibson. "A Large-Scale Study of Failures in High-Performance Computing Systems". In: *IEEE Transactions on Dependable and Secure Computing* 7.4 (Oct. 2010), pp. 337–350. ISSN: 1545-5971. DOI: 10.1109/TDSC.2009.4.

[63] Fabrizio Petrini, Kei Davis, and José Carlos Sancho. "System-level fault-tolerance in large-scale parallel machines with buffered coscheduling". In: *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* Apr. 2004, pp. 209–. DOI: 10.1109/IPDPS.2004.1303239.

[64] Marc Snir et al. "Addressing Failures in Exascale Computing". In: *Int. J. High Perform. Comput. Appl.* 28.2 (May 2014), pp. 129–173. ISSN: 1094-3420. DOI: 10.1177/1094342014522573. URL: http://dx.doi.org/10.1177/1094342014522573.

[65] Bianca Schroeder and Garth A. Gibson. "Understanding Disk Failure Rates: What Does an MTTF of 1,000,000 Hours Mean to You?" In: *Trans. Storage* 3.3 (Oct. 2007). ISSN: 1553-3077. DOI: 10.1145/1288783.1288785. URL: http://doi.acm.org/10.1145/1288783.1288785.

[66] Xiangyu Dong et al. "Leveraging 3D PCRAM Technologies to Reduce Checkpoint Overhead for Future Exascale Systems". In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. SC '09. Portland, Oregon: ACM, 2009, 57:1–57:12. ISBN: 978-1-60558-744-8. DOI: 10.1145/1654059.1654117. URL: http://doi.acm.org/10.1145/1654059.1654117.

[67] Guillaume Aupy et al. "Optimal Checkpointing Period: Time vs. Energy". In: *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation: 4th International Workshop, PMBS 2013, Denver, CO, USA, November 18, 2013. Revised Selected Papers*. Cham: Springer International Publishing, 2014, pp. 203–214. ISBN: 978-3-319-10214-6. DOI: 10.1007/978-3-319-10214-6_10. URL: https://doi.org/10.1007/978-3-319-10214-6_10.

[68] Mohamed Slim Bouguerra et al. "Improving the Computing Efficiency of HPC Systems Using a Combination of Proactive and Preventive Checkpointing". In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. May 2013, pp. 501–512. DOI: 10.1109/IPDPS.2013.74.

[69] Kurt B. Ferreira et al. "Accelerating Incremental Checkpointing for Extreme-scale Computing". In: *Future Gener. Comput. Syst.* 30 (Jan. 2014), pp. 66–77. ISSN: 0167-739X. DOI: 10.1016/j.future.2013.04.017. URL: http://dx.doi.org/10.1016/j.future.2013.04.017.

[70] Leonardo Bautista-Gomez et al. "FTI: High performance Fault Tolerance Interface for hybrid systems". In: *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. Nov. 2011, pp. 1–12. DOI: 10.1145/2063384.2063427.

[71] Adam Moody et al. “Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System”. In: *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. Nov. 2010, pp. 1–11. DOI: 10.1109/SC.2010.18.

[72] Ning Liu et al. “On the role of burst buffers in leadership-class storage systems”. In: *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*. Apr. 2012, pp. 1–11. DOI: 10.1109/MSST.2012.6232369.

[73] Melissa Romanus, Robert B Ross, and Manish Parashar. “Challenges and Considerations for Utilizing Burst Buffers in High-Performance Computing”. In: *CoRR* abs/1509.05492 (2015). URL: http://arxiv.org/abs/1509.05492.

[74] Gengbin Zheng, Xiang Ni, and Laxmikant V Kalé. “A Scalable Double In-memory Checkpoint and Restart Scheme towards Exascale”. In: *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*. June 2012, pp. 1–6. DOI: 10.1109/DSNW.2012.6264677.

[75] Lawrence Livermore National Laboratory. *SCR v1.1.8 User Manual*. [Accessed 12-June-2017]. URL: https://computation.llnl.gov/sites/default/files/public/scr_users_manual.pdf.

[76] Mohammed el Mehdi Diouri et al. “Energy considerations in Checkpointing and Fault Tolerance protocols”. In: *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*. June 2012, pp. 1–6. DOI: 10.1109/DSNW.2012.6264670.

[77] Hybrid Memory Cube Consortium. *Hybrid Memory Cube Specification 1.0*. http://www.hybridmemorycube.org/.

[78] Joe Jeddeloh and Brent Keeth. “Hybrid Memory Cube. New DRAM Architecture Increases Density and Performance”. In: *2012 Symposium on VLSI Technology (VLSIT)*. June 2012, pp. 87–88. DOI: 10.1109/VLSIT.2012.6242474.

[79] Hybrid Memory Cube Consortium. *Hybrid Memory Cube Specification 1.1*. http://www.hybridmemorycube.org/.

[80] JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. *JEDEC Standard JESD 79-4: DDR4 SDRAM*. 2012.

[81] JEDEC SOLID STATE TECHNOLOGY ASSOCIATION. *JEDEC Standard JESD 79-3B: DDR3 SDRAM*. 2008.

[82] J. Thomas Pawlowski. “Hybrid Memory Cube (HMC)”. In: *2011 IEEE Hot Chips 23 Symposium (HCS)*. Aug. 2011, pp. 1–24. DOI: 10.1109/HOTCHIPS.2011.7477494.

[83] Paul Rosenfeld. “Performance Exploration of the Hybrid Memory Cube”. PhD thesis. University of Maryland, 2014.

[84] Bruce Jacob. “The 2 PetaFLOP, 3 Petabyte, 9 TB/s, 90 kW Cabinet: A System Architecture for Exascale and Big Data”. In: *IEEE Computer Architecture Letters* 15.2 (July 2016), pp. 125–128. ISSN: 1556-6056. DOI: 10.1109/LCA.2015.2451652.

[85] Micron Technology, Inc. *Revolutionary Advancements in Memory Performance*. [Accessed 28-July-2017]. Aug. 2011. URL: https://www.youtube.com/watch?v=kaV2nZSkw8A.

[86] Dan McMorrow. *Technical Challenges of Exascale Computing*. Tech. rep. MITRE Corporation, 2013.

[87] Hybrid Memory Cube Consortium. *Hybrid Memory Cube Specification 2.0*. http://www.hybridmemorycube.org/.

[88] Open Silicon, Inc. *Hybrid Memory Cube (HMC) Controller IP*. [Accessed 28-July-2017]. URL: http://www.open-silicon.com/open-silicon-ips/hmc/.

[89] Pico Computing, Inc (now Micron Technology, Inc). *Hybrid Memory Cube (HMC) Controller IP*. [Accessed 28-July-2017]. URL: http://picocomputing.com/hmc-ip/.

[90] Altera Corporation. *Hybrid Memory Cube Controller IP Core User Guide UG-01152*. [Accessed 28-July-2017]. URL: http://design.altera.com/HMCWP.

[91] Computer Architecture Group - University of Heidelberg. *openHMC documentation Rev1.5*. [Accessed 28-July-2017]. URL: http://www.uni-heidelberg.de/openhmc.

[92] Xilinx, Inc. *XHMC v1.0 LogiCORE IP Product Guide*. Nov. 2016.

[93] John D. Leidel and Yong Chen. "HMC-Sim: A Simulation Framework for Hybrid Memory Cube Devices". In: *2014 IEEE International Parallel Distributed Processing Symposium Workshops*. May 2014, pp. 1465–1474. DOI: 10.1109/IPDPSW.2014.164.

[94] Dong-Ik Jeon and Ki-Seok Chung. "CasHMC: A Cycle-Accurate Simulator for Hybrid Memory Cube". In: *IEEE Computer Architecture Letters* 16.1 (Jan. 2017), pp. 10–13. ISSN: 1556-6056. DOI: 10.1109/LCA.2016.2600601.

[95] Yinhe Han et al. "Data-Aware DRAM Refresh to Squeeze the Margin of Retention Time in Hybrid Memory Cube". In: *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 2014, pp. 295–300. DOI: 10.1109/ICCAD.2014.7001366.

[96] Ishan G Thakkar and Sudeep Pasricha. "Massed Refresh: An Energy-Efficient Technique to Reduce Refresh Overhead in Hybrid Memory Cube Architectures". In: *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*. Jan. 2016, pp. 104–109. DOI: 10.1109/VLSID.2016.13.

[97] Mushfique Junayed Khurshid and Mikko Lipasti. "Data Compression for Thermal Mitigation in the Hybrid Memory Cube". In: *2013 IEEE 31st International Conference on Computer Design (ICCD)*. Oct. 2013, pp. 185–192. DOI: 10.1109/ICCD.2013.6657041.

[98] Maya Gokhale, Scott Lloyd, and Chris Macaraeg. "Hybrid Memory Cube Performance Characterization on Data-centric Workloads". In: *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*. IA3 '15. Austin, Texas: ACM, 2015, 7:1–7:8. ISBN: 978-1-4503-4001-4. DOI: 10.1145/2833179.2833184. URL: http://doi.acm.org/10.1145/2833179.2833184.

[99] Khaled Z. Ibrahim et al. “Characterizing the Performance of Hybrid Memory Cube Using ApexMAP Application Probes”. In: *Proceedings of the Second International Symposium on Memory Systems*. MEMSYS ’16. Alexandria, VA, USA: ACM, 2016, pp. 429–436. ISBN: 978-1-4503-4305-3. DOI: 10.1145/2989081.2989090. URL: http://doi.acm.org/10.1145/2989081.2989090.

[100] Ramyad Hadidi et al. “Demystifying the Characteristics of 3D-Stacked Memories: A Case Study for Hybrid Memory Cube”. In: *CoRR* abs/1706.02725 (2017). URL: http://arxiv.org/abs/1706.02725.

[101] Mondrian Nüssle, Martin Scherer, and Ulrich Brüning. “A Resource Optimized Remote-Memory-Access Architecture for Low-latency Communication”. In: *2009 International Conference on Parallel Processing*. Sept. 2009, pp. 220–227. DOI: 10.1109/ICPP.2009.62.

[102] Wolfgang Frings, Felix Wolf, and Ventsislav Petkov. “Scalable Massively Parallel I/O to Task-local Files”. In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. SC ’09. Portland, Oregon: ACM, 2009, 17:1–17:11. ISBN: 978-1-60558-744-8. DOI: 10.1145/1654059.1654077. URL: http://doi.acm.org/10.1145/1654059.1654077.

[103] Stefano Markidis, Giovanni Lapenta, and Rizwan-uddin. “Multi-scale Simulations of Plasma with iPIC3D”. In: *Math. Comput. Simul.* 80.7 (Mar. 2010), pp. 1509–1519. ISSN: 0378-4754. DOI: 10.1016/j.matcom.2009.08.038. URL: http://dx.doi.org/10.1016/j.matcom.2009.08.038.

Acknowledgements

My biggest thanks go out to my parents, Vera and Georg Schmidt, and my sister, Ludmila Schmidt. All my life and unconditionally - you were there for me and always encouraged me to follow my path. I dedicate this dissertation to you.

I would like to express my sincere gratitude to Prof. Ulrich Brüning. He taught me so many things for both, work and life. With his experience and knowledge, he has always been a fantastic supporter, advisor, and my personal role model.

Many thanks to my colleagues at the Computer Architecture Group and the EXTOLL GmbH. Your advice helped me in countless situations and I appreciate the teamwork which has always been excellent. It was a pleasure to work with every single one of you.

I am grateful to the committee members, Prof. Norbert Eicker, Prof. Michael Gertz, and Artur Andrzejak for their support. And of course I would like to say thanks to the Ruprecht-Karls University Heidelberg and especially to the Faculty for Mathematics and Computer Science for the great opportunity to carry out the research that led to this dissertation.

A special thanks goes out to my friends, especially Kevin Rodenhausen, Niklas Sachs, Alexander Schepp, and Christoph Hick. You always supported me in every situation and knew how to properly distract me from work - weekend after weekend ;-)

Thank you Anja Stolzheise for undergoing this journey together with me. I am grateful to have you in my life and looking forward to everything that lies ahead of us. And by the way, as I promised: I made it in time! ;-)

Last but not least, thanks to my roommate and friend Benjamin Klenk, all the friends I made all around the world, and everyone else who believed in me.