Dissertation

submitted to the

Combined Faculty of the Natural Sciences and Mathematics

of the

Ruprechts-Karls University
Heidelberg

for the degree of

Doctor of Natural Sciences

put forward by

Diplom-Informatiker Dirk Frey

Born in
Heidelberg, Germany

March 17, 2021

Design and Implementation of a Network-Attached Accelerator to Improve Data Movement in HPC Environments

Advisor: Prof. Dr.-Ing. Ulrich Brüning

Date of oral examination: .....

Abstract

In this work, the concept of a *network-attached accelerator* was developed. This novel node type connects accelerator and network interface card directly via an artificial PCIe interface. By exploiting the additional functions of the EXTOLL network card, this remote PCIe hierarchy can be integrated into any system with an EXTOLL network interface card via the network without requiring any modifications to the accelerator driver. A Intel® Xeon Phi™was used which can be successfully booted by the remote PCIe hierarchy over the network and is then available as a standalone node. In addition, the Intel® Xeon Phi™itself can actively communicate with other Intel® Xeon Phi™without the need for a CPU. The accelerator and network card form a single unit that can communicate directly with others via the highly specialized EXTOLL network, forming an *cluster of accelerators*. Various experiments demonstrate, that this combination has a higher bandwidth and a lower latency than common systems.

Several prototype systems have been developed to evaluate the concept and reduce the packing density by a factor of four. This is the most densely packed HPC system to date. The enormous cooling demand could only be met by using a novel cooling medium Novec™ 649 . An developed 2-phase cooling system can cool the 12 kW power dissipation of the 32 *network-attached accelerators* with a total of 32 TFLOP of computing power only by the power consumption of a single circulating pump. As a result, the GreenICE system achieves an unprecedented power usage effectiveness of 1.01.

Zusammenfassung

In dieser Arbeit wurde das Konzept eines *Netzwerkangebundenen Beschleunigers* entwickelt. Dieser neuartige Knotentype verbindet Beschleuniger und Netzwerkkarte direkt per künstlicher PCIe Schnittstelle miteinander. Durch geschicktes Ausnutzen der Zusatzfunktionen der EXTOLL Netzwerkkarte kann diese entfernte PCIe Hierarchie über das Netzwerk in ein beliebiges System mit EXTOLL Netzwerkkarte eingebunden werden, ohne das Anpassungen am Beschleunigertreiber notwendig sind. Es wurde ein Intel® Xeon Phi™verwendet der durch die entfernte PCIe Hierarchie über das Netzwerk erfolgreich hochgefahren werden kann und danach als eigenständiger Knoten bereitsteht. Außerdem kann der Intel® Xeon Phi™selbst aktive mit anderen Intel® Xeon Phi™kommunizieren ohne dass eine CPU dafür benötigt wird. Beschleuniger und Netzwerkkarte bilden eine Einheit die über das hochspezialisiertes EXTOLL Netzwerk direkt mit anderen kommunizieren kann und einen *Rechnerverbund aus Beschleunigern* bildet. Verschiedene Experimente weisen diesem Verbund eine höhere Bandbreite und eine geringere Latenz nach als übliche Systeme.

Es wurden mehrere Prototypsysteme entwickelt, um das Konzept zu evaluieren und die Packungsdichte um den Faktor vier zu reduzieren. Dies ist das bis dato dichtgepackteste HPC System überhaupt. Die enorme Herausforderung an die Kühlung und den Gehäuseaufbau konnte nur durch den Einsatz eines neuartigen Kühlmediums Novec™ 649 bewältigt werden. Eine selbst entwickeltes 2-Phasen Kühlsystem kann die 12 kW Verlustleistung der 32 *Netzwerkangebundenen Beschleunigers* mit insgesamt 32 TFLOP Rechenleistung nur durch den Stromverbrauch einer einzigen Umwälzpumpe kühlen. Dadurch erreicht das GreenICE System eine bisher unerreichte Energienutzungseffizienz von 1.01.

Acknowledgments

First of all, I would like to thank my supervisor Prof. Ulrich Brüning, who has always supported me during all this time and has always been an inspiration. Thanks to his broad knowledge he was able to support me in almost all areas. I would also like to thank all members of the Computer Architecture Group for all their support and helpful discussions. But most of all, I would like to thank the person who became first my wife and now the mother of my two children during the course of this dissertation. Thank you Theresa for all the patience you have shown to support me throughout this long journey.

Contents

Chapter 1

Introduction

This chapter describes the research area of this thesis and motivates why investigation in this field is important for the scientific community. In particular, this work aims at the High Performance Computing (HPC) community and hereinafter describes some of the obstacles to master the so called *Exascale Challenge*. This milestone in the field of computer science comprises of several parts and this chapter describes why the focus is set on the power and communication component.

1.1 Motivation

The driving force of innovation in the field of computer science has always been the increasing demand for computational power to solve more complex problems than before. In the early days of computers, the answer was to develop procedures to reduce the structure size of transistors. As a result, more transistors can be placed on the same chip size, the frequency increases, the power consumption decreases and more chips fit on the same wafer, resulting in cheaper hardware.

Until 2010 chip development reached a saturation point, also known as "post-moore era" and reduction in structure sizes abandons *Moore's law* [37] with a doubling of transistors on a die every 18 months. Today, the time it takes to get to the next technology node varies between two or three years. Reasons for this slow down are challenges in the manufacturing process at very small nodes like 22nm, 14nm or below. Another reason is the so-called "power wall". Due to the increasing number of transistors per area, the power density rises above a critical point.

The only chance to still increase the performance and somehow catch up to Moores prediction is parallelization by replicating cores on the same processor chip. Together with new parallel programming paradigms, enabled by the new multi-core architecture, increasing performance is ensured even in the "post Moore-era". Even with the computational power a single machine can provide nowadays, a single machine alone can not solve complex problems like weather forecast, chemical or physical simulations and data analysis in a reasonable time.

Most of the time, scientific simulations contain a high amount of small independent calculations and can therefore execute in parallel. Obviously, multiple machines can work on small chunks of the same problem in parallel to solve the simulation faster. In order to accomplish this, the machines are connected to each other with an interconnection network to share results between the partitions of the simulation. Nodes connected in this way form a cluster and dominate the list of the fastest Supercomputers [61]. This is common practice nowadays.

The challenge is no longer to shrink the technology or increasing the number of replicated compute units per chip. Other factors gain importance too. Due to the permanent growth of the simulation or calculation complexity, the number of cluster nodes required to solve a problem in a reasonable time rises as well and the overall power consumption starts to get the limiting factor.

A study performed by DELL in 2007 [51] states, that power consumption of a data

center can be categorized into three areas. One field is the power delivery from the wall plug, through Uninterruptible Power Supplys (UPSs) to the voltage regulators to provide the operating voltages for the hardware. This takes up to 28 % of the overall power consumption of a data center.

Another area is the power consumption of the hardware components of a compute node like CPUs, GPUs, Network Interface Controllers (NICs) and other external hardware components like network switches. These components have a share of 41 % of the overall power consumption of a data center. To reach the next class of super computers with a performance of 1 ExaFLOP/s (10^{18}FLOP/s) the FLOP per Watt ratio of these hardware components have to go beyond today's clusters. One way to increase this ratio is the use of specialized hardware that provides thousands of low power cores providing a high energy efficiency and large-scale parallelism.

An example for this type of hardware are Graphics Processing Units (GPUs). These accelerators are no longer restricted to pure graphic processing and support a large subset of instructions required for scientific computation like matrix or vector calculation. Each GPU has hundreds of cores that provide a large amount of parallel processing power. Each core has a limited instruction set and a reduces number of specialized units compared to normal Central Processing Unit (CPU) cores. They run at a lower frequency, but the large number of cores outperform CPUs easily. Especially throughput-computing kernels [31] are a lot faster on GPUs then on CPUs. GPUs are available as add-in cards typical as Peripheral Components Interconnect - Express (PCIe) cards. Depending on the chassis used for the compute node commonly one to three GPUs fit into a single machine. Cluster nodes equipped with accelerators can dramatically increase the compute power of clusters and are now common practice. Referring to the TOP500 lists [61] available during the work on this thesis, the number for accelerator equipped clusters has increased from 54 in June 2013 to 110 in June 2018. Conclusion of this trend is that the usage of accelerators in future super computers will increase. How these accelerators are used have a significant impact on the performance of super computers.

Increasing the power efficiency of the computational parts in a cluster is one way to increase the FLOP/Watt ratio. Another one is to save power on the interconnection network. Only recently the research takes power consumption to move data around inside the cluster into account [50]. Data movement includes the movement of data inside chips, like from caches to registers but also data from main memory to the CPU for processing and to I/O devices. Transferring data from local main memory

to a remote memory location across the cluster's interconnection network consumes a large amount of the overall power. The cluster interconnection network got more attention in recent years not only because the communication between cluster nodes is most of the time the limiting factor for the overall performance of the cluster, but also the percentage of the overall power consumption increased over the years. As the number of nodes for an exascale system is growing one or two magnitude higher than todays systems, the interconnection has a large impact on the performance and the power consumption of today's systems.

The last category of power consumption in a data center stated by the study is the power required by the infrastructure to cool all the aforementioned components. As all energy used by the hardware components is dissipated as heat to the environment it is very important to transfer this heat away. This category count fans for forced air flow inside the nodes, fans to provide raised floor cooling for the cluster racks and the air conditioning systems to cool down the air to typically used 18 °C to 21 °C. The overall power consumption of 31 % for the cooling infrastructure of a cluster system is worth to take a closer look on how energy can be saved in this area.

1.2 Challenges to reach Exascale

Nevertheless, all the benefits of accelerators in todays clusters comes with a not negligible cost to get access to the raw performance accelerators provide. As already mentioned, accelerators are add-in cards connected to the host interface which is typically PCIe. The number of accelerators is on the one hand limited by the physical space a compute node provides to house the add-in cards and on the other hand by the number of available PCIe links coming from the CPU or the chipset. This leads to a fixed ratio between hosts and accelerators. While the usage of accelerators local to a compute node is very fast in regards of latency, memory and I/O bandwidth, applications that require more accelerators than available in the node requires explicit communication with other nodes equipped with accelerators. This forces the application programmer to obtain and use detailed knowledge of the cluster infrastructure and network topology to write a scientific application exclusively for this specific cluster to achieve reasonable performance and scalability. The application now must hide additional latencies and bandwidth limitations that arise by the use of the interconnection network to exchange data between the nodes that host the accelerators which may also introduce load imbalance.

To make things worse, accelerators are not able to efficiently source or sink network traffic [46]. The CPU allocates a buffer in the main memory for the data transfer from local memory to the remote node. The data to exchange is copied from the GPU memory to the main memory first. After this copy operation is completed, the CPU triggers the transfer by notifying the NIC which is than responsible to read the data from the previously allocated buffer in main memory and sends the data to the remote node. All communication is driven by the CPU which prevents the CPU from executing workload that requires a high single thread performance. In addition, the copy operation from GPU device memory to main memory and copy from main memory to NIC introduces additional overhead which limits the scalability of the application.

Also the PCIe host interface that is used twice in the above example is a limiting factor. Compared with the GPU on-board memory bandwidth of 208 GB/s [45], the host interface maximum bandwidth of a PCIe 3.0 x16 link is only 16 GB/s. Datta et al. [17] show in their work, that without host interface transfers a stencil application can achieve a speedup of 15× compared to a CPU implementation. If the host interface transfers are taken into account, the GPU performs worse than the CPU. This is clearly a bottleneck and any unnecessary PCIe transactions should be avoided. One technique to avoid host-device copy operations is the use of GPUDirect Remote Direct Memory Access (RDMA) capability [57] of modern NVIDIA GPUs or similar techniques for other types of accelerators. The NIC is given access to the GPUs on-board device memory and the buffer in main memory can be avoided. Nevertheless the CPU is still involved to trigger the communication and is bound to the task of source and sink network traffic. All this introduces significant challenges in the design of future exascale applications that have to utilize the high degree of parallelism.

Therefore it is not surprising that application design is one of the many challenges to reach exascale computing. As a study of the Defense Advanced Research Projects Agency (DARPA) [6] pointed out, not only the programming model is a challenge, the power consumption is too. Cooling based on air is easy to implement but not sufficient for exascale systems. New techniques to remove heat are emerging with water flowing trough a metal plate on top of the compute boards to use water as coolant. Water has better physical properties than air, but requires large metal plates to cover the whole board and uses 2 or 3 times more space than the actual board. Also the use of water in a environment with electrical components has a

high risk for damage when a leakage occurs. Special care is required to avoid this situation. Even if water as coolant is superior to air the use of water cooling as a subsequent change is hard to realize. The usage of this method has to be taken into account in the early design phase.

In order to contribute to overcoming the existing hurdles of reaching exascale, this thesis addresses these topics:

- Dynamically assign accelerators to hosts in a n-to-m ratio
- Increasing the power efficiency with the use of accelerators
- Reducing the power consumed by cooling
- Reducing the power of the interconnection network
- Increase the packaging density
- Source and sink network traffic and avoid CPU driven communication
- Minimize the effect of the host interface on communication
- No additional software layers or driver modifications
- Increase utilization of GPUs and CPUs or other types of accelerators e.g. for AI

1.3 Focus of the thesis

A publication of Rinke et al. [55] proposes a solution to some of the problems described in the section above. Instead of adding accelerators to cluster nodes, a separate cluster is build and consists of accelerators only. The connection between accelerators and compute notes is now dynamically assigned during runtime. An application requests the number of accelerators it needs to perform its calculation from an Accelerator Resource Manager (ARM) which receives requests and grants access to the requested number of accelerators if they are available. This approach increases the utilization of accelerators and enables a finer control of the compute resources of the cluster. Applications that doesn't benefit from accelerators can request compute nodes only and do not block the accelerators from being used by other applications. The programmer no longer need detailed knowledge of the clusters internal hardware structure and can handle any number of accelerators as if they were locally attached to the compute node.

In the publication of *Rinke et. al.* the term of an *accelerator only cluster* is a bit misleading as the accelerators are still connected to a standard compute node. This

system for the proof of concept only host the accelerator, the ARM and the NIC. No job can be scheduled on these nodes. The interconnection network is used for the communication between compute nodes and accelerators. A user library intercepts all calls to the accelerator and tunnels them trough the interconnection network to the accelerator node. The accelerators do not directly access other nodes and require the assistants of a daemon running on the CPU that drives the communication.

This thesis will revive the idea of the described publication and extends it to tackle some of the challenges described in the previous section. The focus will be on the use of the Intel Xeon Phi, since it has unique characteristics compared to other accelerators, which make it particularly useful for the purpose of this work.

Accelerator Node Design The design of an accelerator node described in the publication is a full blown server system. It consists of a CPU that runs a daemon to handle the send and receive requests from the cluster nodes. Local main memory is used to buffer the received or to send data from the NIC or the accelerator respectively. Both devices are on the same PCIe hierarchy and need the CPU to transfer data from on device to the other. Therefore the thesis investigates how an accelerator node that only consists of an accelerator and a NIC can be build with a direct connection between each other.

CPU-less Communication As the accelerator is not able to source or sink network traffic a daemon is running on the accelerator node that drives all network communication. The daemon is responsible to allocate buffers, start and terminate transfers and moves data between the NIC, main memory and the accelerator. All these task are additional overhead and limits the application's scalability. This thesis will investigate how existing and novel network technologies can reduce this overhead of communication between accelerator and NIC.

Direct Network access For every network access the control flow has to switch from GPU to the CPU context. The CPU reads from the device and sends the data over the interconnection network and returns to the GPU control flow. Modern NICs support features like RDMA to access directly memory locations of another node's process. Other programing models like GPU Global Address Space (GGAS) or Partitioned Global Address Space (PGAS) to access remote locations by addresses is another interesting approach for communication between accelerators. All these concept require software assistance that intercept

the normal operation. This thesis will investigate how these concepts can be made available directly to the accelerator without the intervention of a CPU.

Cooling As the improved accelerator node design should be as streamlined as possible new cooling concepts can be used. As the accelerator node only consists of an accelerator and a NIC, they can be packed close to each other. For conventional cooling with air the minimum distance between the boards is defined by the size of the heat sinks. To relax this space requirement, the possibilities of a novel type of cooling liquid is being investigated.

Interconnect power consumption The most common connection networks are based on a star topology with external switches. These connect the nodes with each other. To achieve sufficient performance of the cluster system, it must consist of a large number of nodes, which makes it spatially very widespread. With this spatial expansion, the length of the cables and the power needed to drive the signals increases. In addition, more switches are needed to connect the higher number of nodes. This thesis uses a novel interconnection network which avoids the external switches completely and reduces the cable length, thus saving a large amount of power.

GPU cluster topologie To get a good load balance between the different accelerators the latency and bandwidth has to be known. These values strongly depends on the physical location inside the cluster and the network topology. With the accelerators forming their own cluster, the topology, latency and bandwidth does not change and have less impact to load balance. It also eases the effort to the application programmer. This thesis will build such a cluster of accelerators and evaluates how the overall performance of applications is affected.

All these research topics described above lay their focus on hardware changes like the use of novel network interconnects, custom designed components like Printed Circuit Boards (PCBs) and other hardware components. This is due to the fact that all software related topics beyond basic initialization and configuration are described in more detail in the work of Neuwirth[39]. Nevertheless, all software modules required to initialize and configure the hardware are presented in this thesis and briefly described to provide a better understanding of the underlaying hardware's function.

1.4 Structure of the thesis

The following chapter 2 on page 11 will give some background information on the hardware components used in this thesis. To get familiar with basic aspects of cooling the chapter concludes with a brief summary of different cooling concepts and some important physical metrics. Chapter 3 on page 43 will show the current state of the art of intra and inter node communication and cooling solutions established in the industry. The work briefly described in section 1.3 on page 6 is not the only research done in this field and the state of the art chapter will summarize some of the other important publications that are of interest for this thesis. Chapter 4 on page 65 will analyze the state of the art and the related work on the different interconnection and cooling solutions to derive a design that meets the requirements described in section 1.2 on page 4. After the design is described chapter 5 on page 79 shows details of the different prototype implementations. The chapter 6 on page 107 describes the findings from the different prototype systems. Both the knowledge gained from the accelerator nodes and the experience gained with the new cooling liquid are presented. The chapter 7 on page 117 summarizes the thesis and shows additional research topics that are possible in the future or which parts are open to more intense research.

Chapter 2

Background

This chapter explains fundamental hardware components used throughout this thesis and briefly describes important features and characteristics. Accelerators play an important role in increasing the power-efficiency and compute power of a system. To understand why accelerators have this attributes the first section examines the General Purpose Graphic Processing Unit (GPGPU) and Many Integrated Core (MIC) architectures. All accelerators described here use PCIe as its host interface. As later parts of this thesis describe how PCIe can be used across host boundaries a section in this chapter describes PCIe architecture in detail. The EXTOLL interconnection network provides incomparable features that are essential for the implementation of the proposed Network Attached Accelerator (NAA). These important features are described in this chapter. To build a common ground, basic cooling concepts for electronic devices are covered at the end of this chapter and briefly describes physical metrics for important material properties.

Figure 2.1: NVIDIA Tesla K20 PCIe full-length full height card [45]

2.1 Accelerators

As accelerators are of great importance for this thesis the following sections describe the features and different kind of accelerators that are used throughout the thesis. Even though most of the work is about the Intel Xeon Phi, a small introduction to GPGPUs will be given here. This is to show the differences between the architectures and thereby highlight the advantages of the MIC architecture and the advantages that arise for the intent to build a NAA.

2.1.1 General Purpose Graphic Processing Unit (GPGPU)

The most important kind of accelerators are General Purpose Graphic Processing Units (GPGPUs). In the early days, GPUs were components that receives data from the CPU, stores it in a frame buffer and converts it into a suitable image representation for the display device like a monitor. Most operations on image representations are memory-intensive such as texture mapping to surfaces, rendering polygons and geometrical calculations like rotation and translation into different coordinate systems. This involves many main memory operations from the CPU to calculate the color information, store the result of the calculation back to main

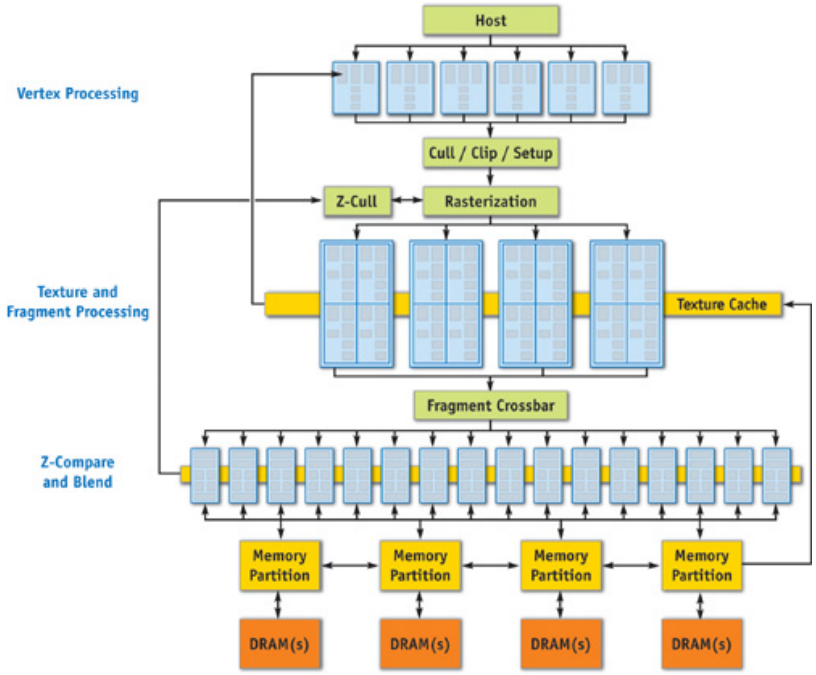

Figure 2.2: Graphics pipeline of NVIDIAs NV40 architecture [1]

memory and then send it to the GPU to display the picture. To accelerate this process and to relieve the CPU from this task the GPUs got specialized hardware functions. GPUs basic data elements are vectors of two to four dimensions and store either 3-dimensional color information (red, green and blue) or 3-dimensional coordinates of vertices. As most of the operation on these vectors do not depend on each other, these operations can be done in parallel with hundred of vectors as input for the calculation. To further increase the performance of image processing fixed pipeline stages with a fixed function are used. These units are called *shaders* and each shader works on the input from the previous stage and outputs its processed data to the next stage. Figure fig. 2.2 shows a simplified graphic pipeline.

The program or kernel for the vertex and fragment shaders are very primitive manipulations on these vector elements such as vector addition or matrix multiplication. Each shader has a lower clock frequency as compared to CPUs, but the same operation is done on many elements in parallel on hundreds of cores per shader unit. As most scientific computation problems are mathematical operations on vectors or matrices this massive parallel computational power is far beyond the capabilities of sequential CPUs. This attracts computer scientists to take advantage of GPUs to solve complex tasks in less time and more power efficient. But these pipelines from

Figure 2.3: Tesla K20 Overview [44]

the early days had limited capabilities of reading input data from memory or writing results back to memory. The vertex shader which performs geometrical calculations on vertices can read its input vertices from memory but only the fragment shader, which calculates color information for pixels, is able to write back the output data to the framebuffer. This increases the effort to map algorithms to these fixed function pipelines together with the restriction that all scientific data have to be translated into textures and vertices to be processed by the GPU.

To circumvent this limitations the GPUs evolved from fixed function pipelines to nowadays GPGPUs with thousands of independent cores that can do either graphical computation on images or general purpose computations on floating point values depending on the program the shaders execute. Figure 2.3 shows one of 15 Streaming Multiprocessor (SMX) with its 192 scalar in-order cores of a NVIDIA K20 accelerator. In addition to the cores which can perform a floating point or a scalar operation, the SMX features 64 double precession units and 32 load/store units. Special functions inside the SMX can execute transcendental instructions such as sin, cosine, reciprocal, and square root.

GPU programing is fundamental different to conventional CPU programing and requires knowledge to the underlying hardware to get a scalable high-performance

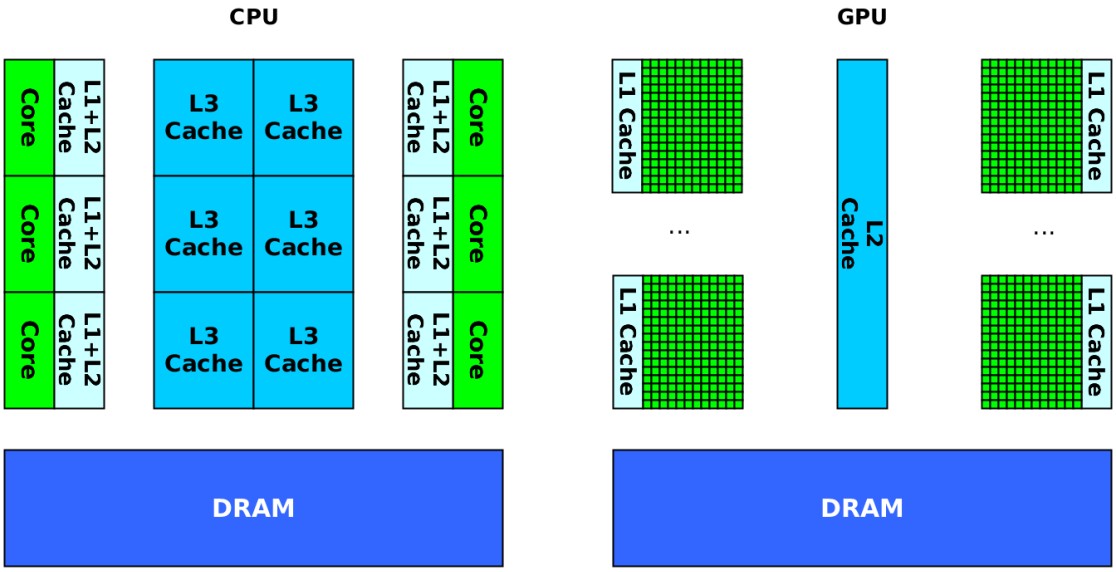

Figure 2.4: Comparison of CPU and GPU architecture [34]

code that uses hardware resources efficiently [8]. Figure 2.4 shows the difference in die area usage. A CPU dedicates most of its die area to on-chip memory for one to three level of caches whereas a GPU trades on-chip memory for a high amount of compute cores. To further increase the number of cores, each core has less control logic and waives complex instruction-level parallelism, branch prediction, speculative and out-of-order execution in favor of smaller core sizes. Each Compute Unified Device Architecture (CUDA) core can execute one integer or one floating-point instruction per clock cycle. To ease the scheduling effort and dispatching of instructions, 32 threads are grouped together and form a *warp* which is the smallest unit that can be scheduled. The warp is then executed on 32 cores of a SMX. Each thread in a warp executes the same instruction in lock-step as Single Instruction Multiple Threads (SIMT). If one thread in a warp can not execute its instruction, because the operands are not present, the warp is suspended and another warp ready for execution is scheduled. The biggest source of latency comes from cache misses and in the worst case access to the global Graphical Double Data Rate 5 (GDDR5) on-board memory. The effort to switch the context from one warp to another is extremely low, which allows for latency hiding of memory accesses by scheduling another warp. This implies to fully utilize the cores in a SMX much more warps must be generated to hide latencies and sustain high throughput.

GPUs struggle with the *memory wall* [67] just like CPUs. It is far more expensive

Figure 2.5: Intel Xeon Phi Co-Processor PCIe full-length full height card [10]

to transfer data to the CPU or GPU than computing on them. Even with the superior memory bandwidth of GPUs fetching a single value from global GPU memory takes in the order of hundreds of clock cycles. As most of the kernels run on the GPGPUs are memory bound, it is important to support a high host interface I/O bandwidth between main memory and GPGPU to occupy as many CUDA cores as possible.

2.1.2 Many Integrated Cores

Intel's answer to the increasing interest in highly parallel accelerators was the Intel Xeon Phi co-processor with its code name Knights Corner (KNC).

The architecture of the GPGPU described in the previous section results in the following properties that must be taken into account in order to get the maximum performance out of the GPGPU. One of these features is that threads are not independent of each other, but are bundled and scheduled in packets, so-called warps. This complicates the programming, because one always has to take care to get a full warp and also to have enough warps to hide latencies caused by memory accesses. This need to hide latency is another property that must be taken into account. Since a GPGPU has more compute cores than on-die memory, it is extremely important

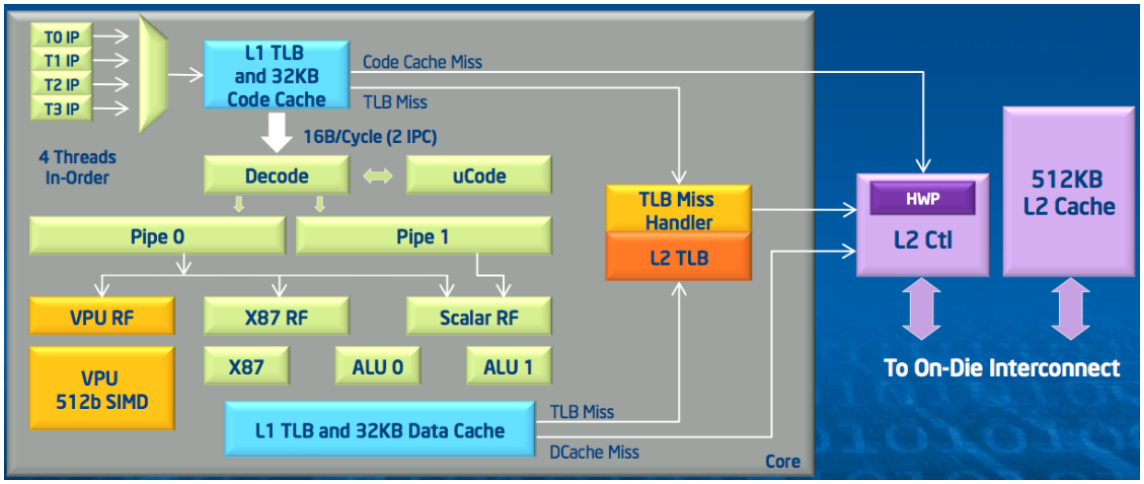

Figure 2.6: Intel Xeon Phi Core Architecture Overview [10]

to consider data locality and to be very careful about which memory holds the data and instructions.

In contrast to the design explained in 2.1.1 Intel does not use light-weight special function cores, instead the KNC uses super scalar cores that are x86 compliant.

This has the immense advantage that existing application code, cross compiled for the KNCs architecture, can run without modifications on KNC add-in cards. Besides the reuse of existing code, the structure of the individual super scalar cores allows the use of significantly more on-die memory compared to a GPU. But this advantage is compensated with a much smaller number of cores. On a KNC of the highest performance level only 61 cores can be accommodated. Also the scheduling of threads is simplified compared to the GPGPU and every thread can be executed on every core. It should be noted that each core can execute 4 threads at once, which makes a total of 244 threads on a KNC with 61 cores.

To further increase the performance, each core has a Vector Processing Unit (VPU) with 32× 512bit vector registers as shown in fig. 2.6 for 16 single-precision floating-point or 32bit integer and 8 double-precision floating-point or 64bit integer operations per vector instruction. Applications that make use of vectorization can use this 512bit vector Instruction Set Architecture (ISA), called 512-bit Advanced Vector Extensions (AVX-512) to further benefit from the KNC's architecture.

Both architectures have their advantages and disadvantages. Neither one is clearly better than the other and it depends significantly on the application whether the GPGPU or the MIC architecture provides better performance. The biggest and most important feature which is a unique feature and of highest interest for the project

NAA is the x86 compliance and the possibility to run operating systems directly on the add-in card. This allows the KNC to be used as an independent node. One can login via Secure Shell (SSH) into the KNC and run the application on the KNC directly or the program on the host system can use the KNC as an offloading target. If the application is split into sequential and parallel portions, each portion can run on the best fitting architecture. But the best part is the availability of a common operating system allows loading of device drivers to use the host systems hardware from the KNCs operating system. This capability is the key to the design of the NAA and is described in detail in the chapter 4.

2.2 Accelerator communication

Now that we have looked at the different architectures of the GPGPU and MIC, we will now turn our attention to the communication between the CPU and the accelerator. It is important to understand the different communication paths on accelerators to fully understand the bottlenecks and performance limitations of accelerators. Therefore, in the further course of the work, a great deal of attention will be paid to circumventing these limitations with new approaches.

2.2.1 Basic Device I/O Communication

The accelerator exposes a set of memory regions for communication between the CPU and the device. During initialization of the I/O devices, physical addresses of the CPUs memory space are assigned to these regions. A driver running on operating system level has access to this physical regions by mapping the physical pages assigned to the device into the operating system virtual kernel space. Communication with the device are now memory read or write operations to kernel virtual addresses which are translated by the CPU's Memory Management Unit (MMU) into physical addresses, the CPU's chip-set translates these addresses into I/O operations of the used I/O interface. The operating system protects the access to the device by prohibiting user applications a virtual-to-physical address mapping to physical addresses of the device. As user application's memory and the operating systems memory are protected from each other, the driver of the device at the operating system provides an interface to interact with the device. User space programs call functions from the kernel space driver which copies data from the user space

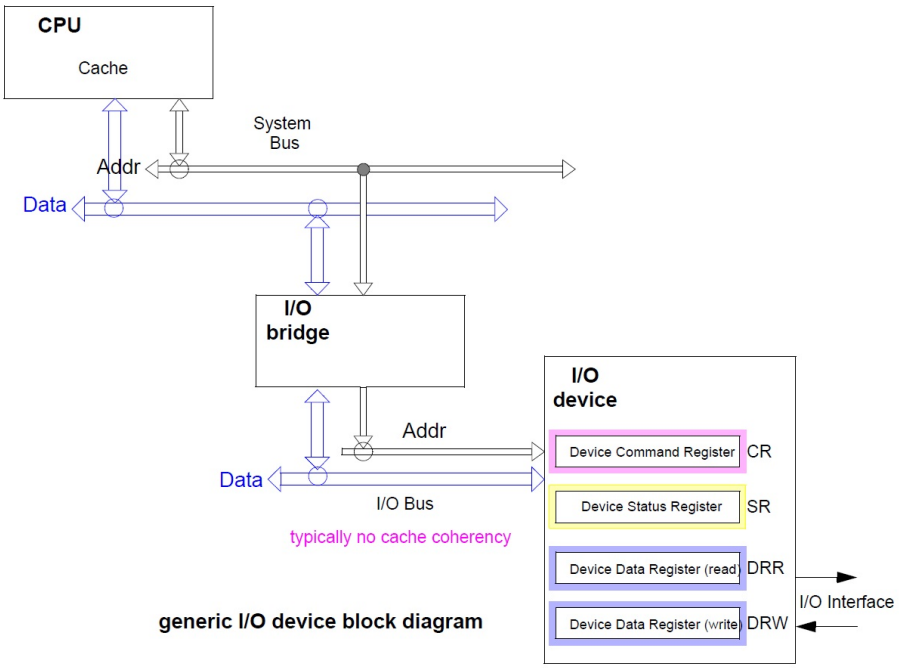

Figure 2.7: Logic Structure of Basic I/O

program into kernel space and then sends the copied data to the device with I/O operations. This is the most robust kind of device communication as each memory space (user space / kernel space) is strictly protected from each other and the direct access to the device is limited, but this kind of communication involves at least two *memcopy* operations which are known to be very slow and restricts the throughput.

2.2.2 Optimized Device I/O Communication

To improve the performance of the device communication compared to the procedure described above, the operating system can relax the virtual-to-physical mapping restriction and allow direct access to the devices memory regions. Memory read or write access to user space virtual addresses are translated into physical addresses and result in I/O operations which directly access the device. This bypasses the kernel and increases the performance of memory and device transfers. All procedures described in the previous section are Programmed Input/Output (PIO) operations that are driven by the CPU. All data written or received by the accelerator is read or written by the processor which causes a high number of unnecessary CPU cycles and occupies the IO subsystem. With direct access to the device, Direct Memory Access (DMA) is possible. Reserved physical memory regions in the user

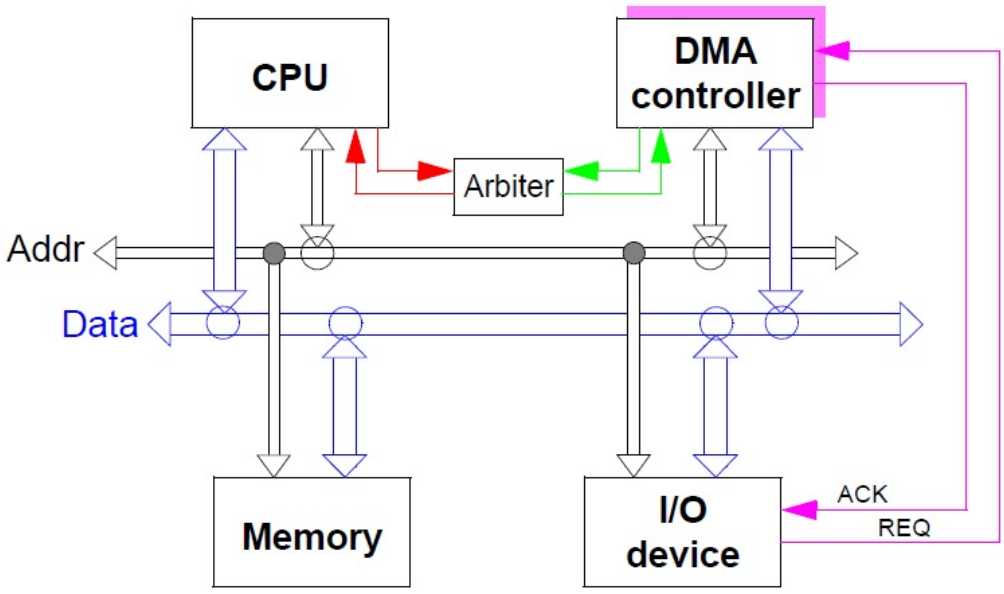

Figure 2.8: DMA I/O

space program are send and receive buffers for DMA transfers. Most accelerators have DMA capabilities of one or several DMA controllers. The CPU sends DMA descriptors with the buffer's physical address and size to the accelerator and the accelerator's DMA controller reads or writes the data without CPU intervention. This releases the CPU from the transfer task and the CPU cycles are available for computational tasks, but the cache coherence is made more difficult by the fact that memory values can be changed that may still be present in one of the CPU caches.

DMA also allows communication between devices without using the CPU. Instead of using reserved physical memory regions in user space, the reserved area can also point to the memory of another device. The memory operations then do not target the main memory, but are directly forwarded within the I/O subsystem to the target device. As already described, this not only saves CPU cycles but also avoids accessing the main memory and saves bandwidth in the main memory interface. This procedure is described in more detail in section 3.1.1 on page 44 and symbolizes the immense importance for the project. But first the used host interface and its peculiarities will be described in more detail.

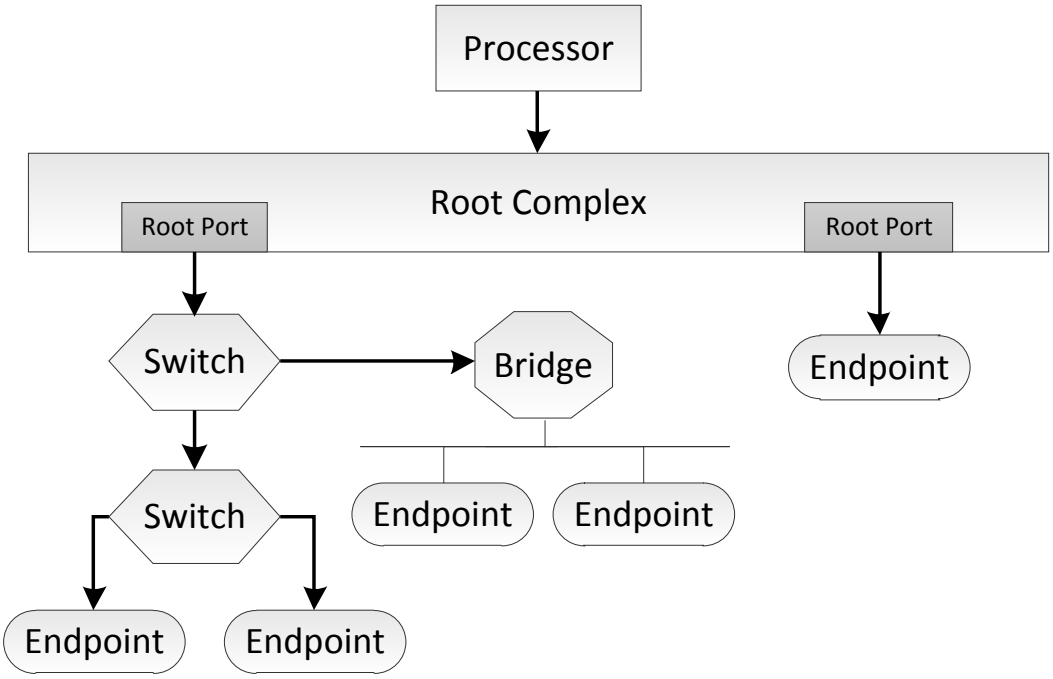

Figure 2.9: PCIe Architecture Overview

2.3 PCI Express Host Interface

Communication with the accelerators is closely linked with transactions on the given host interface. On most modern systems PCIe [9] is used. As we will build our own software driven PCIe hierarchy on the remote accelerator node this sections shows all important features and procedures important for this thesis. This section will start with an architectural overview, shows some details about enumeration and configuration and ends with some information about the packet protocol.

2.3.1 Architecture

The host interface PCIe is the successor of Peripheral Components Interconnect (PCI) and Peripheral Components Interconnect - Extended (PCI-X). Its the most common standard to connect devices between memory and processing cores. Started as a bus interconnect, PCIe is nowadays a Point-to-Point interconnect and is organized as a tree with a single root. A link between two PCIe devices is still called a *bus* but in reality its a point-to-point connection. Each bus consists of up to 16 lanes and each of them are bidirectional differential pairs with a transfer bandwidth of 2.5 Gbit, 5 Gbit and 10 Gbit per direction. Figure 2.9 shows an example for

a simple PCIe hierarchy. At the top is the Root Complex, which serves as a link between the memory management of the processing cores (North Bridge) upstream and the downstream I/O devices. A Root Complex has one or multiple Root Ports. Each of these ports connect either directly to a Endpoint device or a switch. Multiple switches can be arranged hierarchically to support a tree structure to connect more than one device per Root Port. Internally these switches act as PCIe-to-PCIe bridges. Due to its backward compatibility also former versions of PCIe, PCI and PCI-X can be used by bridges which translate between the different protocol versions.

Each switch has a single upstream port (Primary bus) and one or more downstream ports. The links between switches, Endpoints and Root Ports are called buses and are numbered in depth-first-search. The smallest downstream bus number of a switch is called secondary bus and the highest downstream number is called subordinate bus. Switches also specify the memory regions on the downstream site of the switch. Addressing inside this type of interconnect is done by bus IDs or interval routing with memory addresses. To address a device inside the PCIe hierarchy, each device has a unique Device identifier. This identifier consist of an 8 bit bus number, a 5 bit device number and a 3 bit function number. This allows up to 256 buses, with 32 devices per bus with 8 functions per device. The `Device ID` and `Bus ID` are assigned during the enumeration and can change during runtime.

PCIe uses packets to transfer information between the components and are a split transaction model which differentiates between posted or non-posted read or write requests. Posted transactions do not have completions and terminate at the moment they are received by the target of the transaction (completer). Non-posted operations are confirmed with a completion which contains a return code for success or failure. In case of a read request the completion also carries the data from the read operation back to the requester. Unique source tags are used with each non-posted request to keep track of all outstanding transactions and reassociate received completions with the original request.

2.3.2 Interrupts

Interrupts are an important functionality of peripheral devices. Depending on the device and programming model an interrupt is used to tell the driver to execute a predefined operation like moving a received packet from the NIC to the host

memory for further processing. Older host interface standards used hardware routed side-band interface (interrupt lines) to signal the processor of an event that has to be handled. This limits the number of devices to have an exclusive interrupt line assigned to them. Most of the time more devices had to share an interrupt line which is associated with multiple host interface accesses to determine which device caused the interrupt. PCIe uses an in-band mechanism called Message Signaled Interrupt (MSI) which uses the packet protocol to signal an interrupt. During the enumeration process, each device that requests interrupt support is assigned a specific interrupt address and a specific data string. Depending on the PCIe feature used, a device can have 32 interrupts MSI or up to 2048 interrupts Message Signaled Interrupt Extended (MSI-X). If a device signals an interrupt, a write operation is build with the predefined address from the enumeration process and the payload contains the specific string to identify which interrupt number is used. The address targets an Advanced Programable Interrupt Controller (APIC) which can receive interrupts from multiple devices and informs the processor about the interrupt event for further handling.

2.3.3 Configuration Space

Each device has a 4 KB configuration space to store information about the capabilities of the device, the link capabilities and information of the memory regions assigned to the device. To be compatible with legacy *PCI/PCI-X* devices each device implements a 16 DW (256 Byte) configuration header starting on the lower bound of the 4 KB PCIe configuration space. All PCIe extended capabilities are stored in the upper 4 KB in a linked list. Access to this configuration space is done with special configuration packets. An Endpoint implements a *Type0* configuration space header whereas all other device types (*Root Complex*, *Bridge* or *Switch*) implement a *Type1* configuration space header. The most important fields of config space are:

- **DeviceID/VendorID**
  These two 16 bit values are uniquely assigned to registered PCISIG members and serve as an identifier for driver software to adjust the software to the needs of the device.

- **Command**
  After initialization, the device can not respond to memory requests. The device

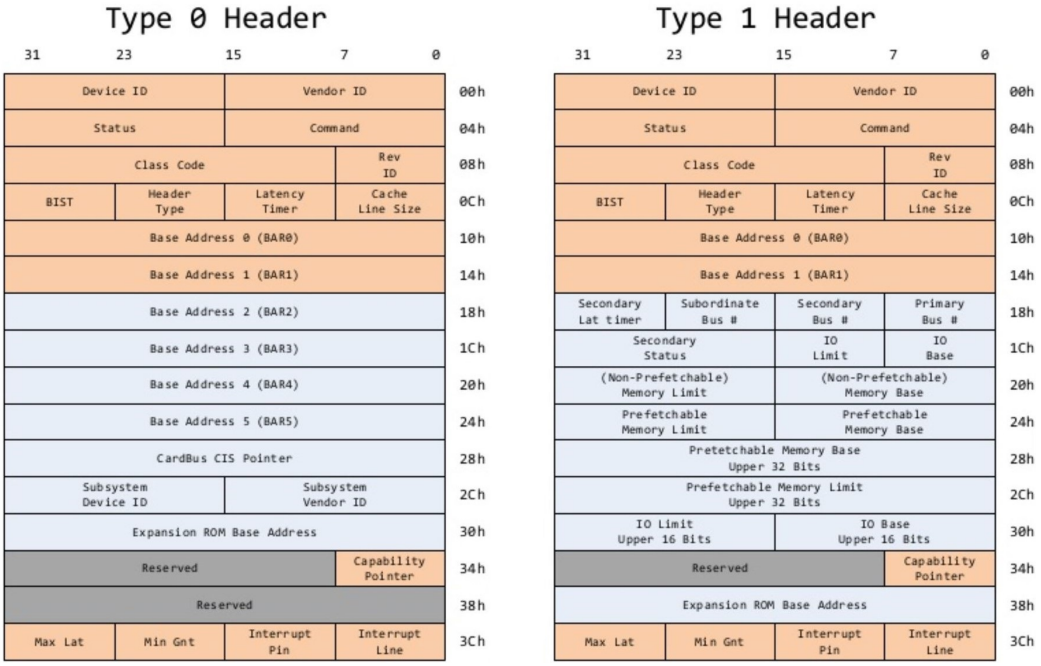

Figure 2.10: PCIe Configuration Space Header Type0 (left) and Type1 (right)[9]

driver can activate the address translation engine by writing to this register and set the busmaster enable, memory implemented and IO implement bits to 1.

- **Base Address Register (BAR)**
  A Device can implement up to 6 × 32 bit memory BARs or up to 3 × 64 bit memory BARs. The lower 4 bit of each BAR contains flags to select between 32 bit or 64 bit BAR and a flag to identify a memory region as prefetchable or not. These BARs are used to request memory space to be mapped into kernel or user space (Memory Mapped Input/Output (MMIO)).

- **Device/Link Capability**
  This register describes the features the device supports. It contains the maximum link speed and the allowed payload size of packets. This size is split into the *payload size* which determines the maximum size of a request or a completion packet. The *read request size* determines the number of bytes a single read operation can request. In both cases, if more bytes are requested or should be transfered, the request has to be split in smaller chunks which satisfy the requirements. The device with the smallest values determine the payload and read request size to the whole path from requester to the final

target. This limitation origins from the buffer management to avoid buffer overruns.

- **MSI**
  This register consist of an address and a data field. The address field holds the address the interrupt packet is send to. To distinguish different interrupts per device the data register contains a string with the least significant bits can be altered to signal different interrupts.

- **Primary,secondary subordinate**
  As the PCIe topology is based on a tree structure with a depth-first search style, each switch or bridge holds a register with the bus numbers of the downstream devices behind the switch or bridge. The primary number contains the upstream bus the switch or bridge is connected to. Secondary number is the smallest bus number towards the depth-first path. Subordinate bus number defines the highest bus number that is located behind the switch.

- **Prefetch / Non-prefetch**
  The memory requested with BARs is distinguished into prefetchable and non prefetchable memory and this register determines the size behind the bridge or switch. Prefetchable memory can be cached, because its content does not change without requests from the processor. Non-Prefetchable memory can change between two consecutive read operations like status registers. Each bridge and switch checks the memory addresses of a request, if it lies within a valid range behind the bridge. If this check fails, the packet is dropped and an error response is sent back to the requester.

2.3.4 Packet format

Transaction Layer Protocol (TLP) are the uppermost protocol layer and can be divided in read and write transactions on physical memory and device internal configuration space. TLP packets are 3 or 4 DW for 32bit and 64bit addressing. The maximum payload length is defined as 4kB but commonly used are lengths between 128Byte and 256Byte.

This section is limited only to the configuration packet format and the completion format, since these packets play an important role in the implementation of the NAA,

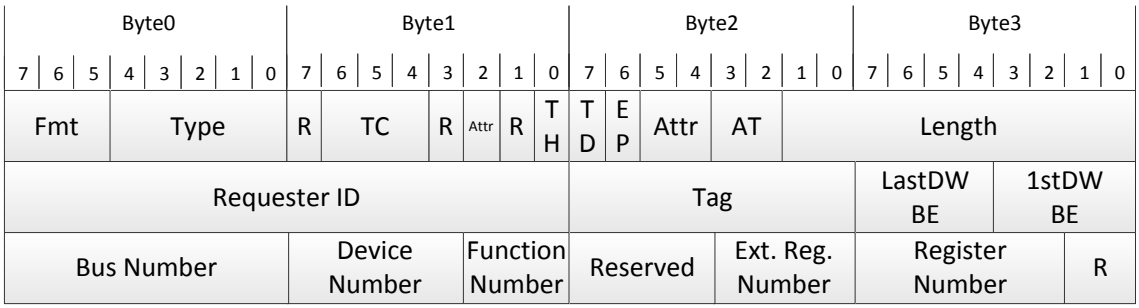

Figure 2.11: Configuration Request Packet

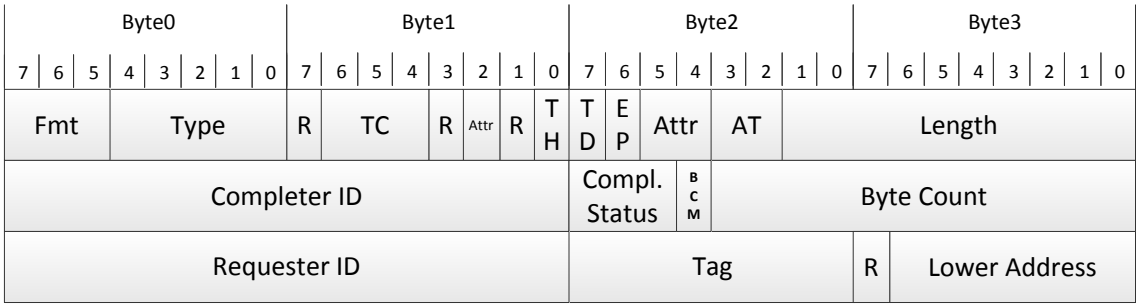

Figure 2.12: Completion Packet

since the configurations software generates these packets to configure both the Root Port and the accelerator.

CfgWr/CfgRd

To access the 4 KB configuration space of PCIe Endpoints, Bridges and Switches, the Root Complex and the Root Port sends configuration packets. To identify the source of the request the Requester ID and a source tag are used to distinguish between different outstanding requests. To select a register of the configuration space header the *Register Number* and the *Extended Register Number* form a double word address.

If a configuration space request arrives at a Bridge or a Switch, the target device is identified by the Bus, Device and Function number which form the *Completer ID*. The Switch or Bridge routes the packet to the port in the direction of the addressed device. After arrival at the targeted device, the configuration request is processed.

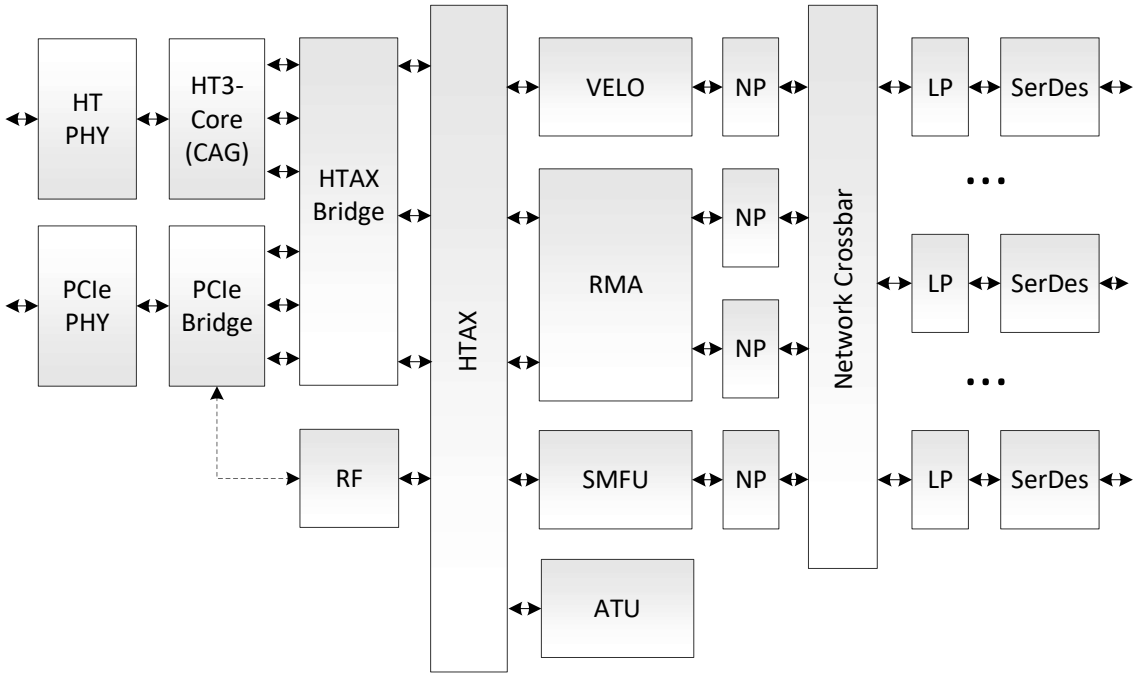

Figure 2.13: EXTOLL Module Overview

Completion Packet Format

Completions are sent for each non-posted request. A request can be answered by one completion or multiple split completions and can have data attached or not.

Completions always use a 3 DW header and the complete packet header with all fields is shown in 2.12. The second DW has a *Byte Count* field to count the remaining bytes of the request. The three bit of *Completion Status* are used to indicate if the completion completed successfully or with a specified error condition.

The Completer ID is the ID of the device generating the completion and is the target of the request. The Requester ID and the source tag forming the Transaction ID, which is used to uniquely identify the source of the request in the PCIe system.

In case of a request with byte access rather than a complete DW access the lower address field specifies the offset from the DW address with the first valid byte in the completion.

2.4 EXTOLL

As the EXTOLL interconnect is used throughout all prototype implementations its key features and most important modules are described in the following sections.

The interconnection network Extended Atomic Low Latency (EXTOLL) [42] is the successor of the Atomic Low Latency (ATOLL) [26] interconnection network and is the spin-off from a long-running research project at the Computer Architecture Group (CAG) at the Institute of Computer Engineering at Heidelberg University. The design has its focus on the needs of the high performance interconnection network market segment and has several features particular designed for HPC environments. Every component of the stack is designed to deliver the lowest latency possible, because "Latency lags bandwidth" [49]. This starts at the application user libraries and goes down to kernel drivers and every single hardware module. Not only the low latency of EXTOLL is important for this thesis, the direct network design is too. Instead of a star topology with external switches, like Ethernet or Infiniband, EXTOLL integrates the switch functionality on-chip and provides up to seven links to directly connect other EXTOLL cards with each other. With a radix of seven, different interconnect topologies are possible to implement. Most of the time a 3D-Torus is the favored topology, because it provides a small diameter compared to other topologies and maps naturally to scientific problems on volumes like weather-forecast or fluid dynamics.

As shown in fig. 2.13 on the previous page the design is divided in three parts, the host interface, the functional units and the network part. The host interface modules are responsible for memory read and write access to either host main memory (DMA) or direct access to other host devices residing on the PCIe interface. The network submodules are responsible for routing of network packets and the translation from the internal HyperTransport On-Chip (HTOC) protocol into the network protocol. The three functional units provide hardware accelerated capabilities for low latency two sided communication like Message Passing Interface (MPI) *send* and *receive*, single sided communication with zero copy for RDMA and shared distributed memory across multiple nodes. Of particular interest is the Shared Memory Functional Unit (SMFU) which can be used to map the memory of a remote PCIe device into the host's local MMIO range. This unit is the key element to build a cluster of accelerators during this thesis. The following sections describe all modules that are important to build a cluster of accelerators in more detail.

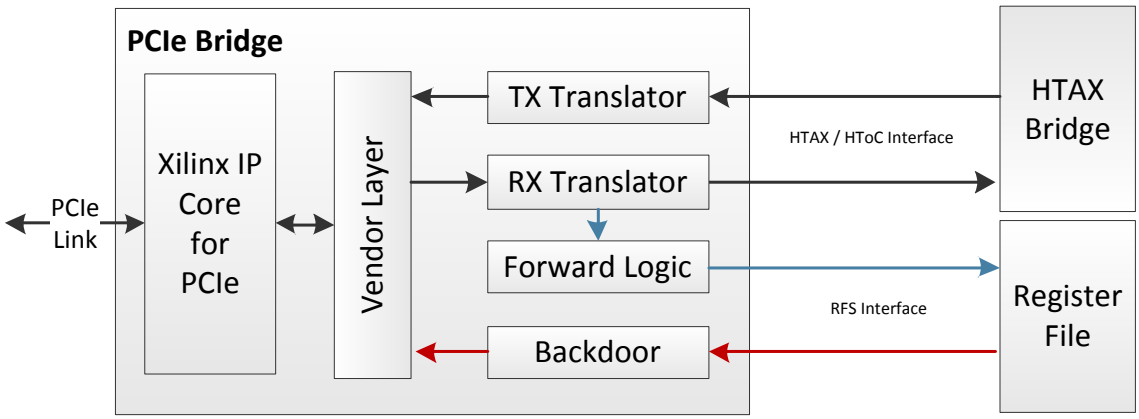

Figure 2.14: PCIe Bridge Overview

2.4.1 PCIe Bridge

The main purpose of the PCIe Bridge is to translate between the host interface protocol PCIe and the internal protocol of EXTOLL, which is a proprietary extension of the HyperTransport (HT) [15] host interface and is called HTOC [11]. The bridge is divided into a RX module and a TX module as shown in fig. 2.14. The RX translator receives PCIe packets from the host interface, translates it into HTOC packets and send the packets to the functional units and the network. The TX translator is responsible to translate received HTOC packets from the network or the functional units into PCIe packets. A PCIe IP Core from Northwest Logic [41] is used to translate the transactional protocol of PCIe into the low level physical layer implementation and vice versa. Important is the possibility to use the IP core not only as PCIe Endpoint but also as Root Port. This function makes it now possible to integrate the accelerator into the PCIe hierarchy. The Vendor Layer is responsible to drive the signals required by the IP Core to transfer and receive packets. The PCIe Bridge also contains the PCIe Backdoor to send arbitrary PCIe packets towards the PCIe host interface. The Forwarding Logic can extract arbitrary PCIe packets from the incoming PCIe stream for debugging purposes or to handle unsupported PCIe packet types. The TX submodule shown in fig. 2.15 on the next page has a software accessible buffer (internal RAM) to store packets that should be send to the outgoing PCIe traffic stream. The RX submodule can filter the incoming PCIe traffic stream and extracts packets into a software readable buffer.

For this thesis, these features are of particular interest. The PCIe bridge handles the extraction of filtered packets from the incoming PCIe traffic and handles the

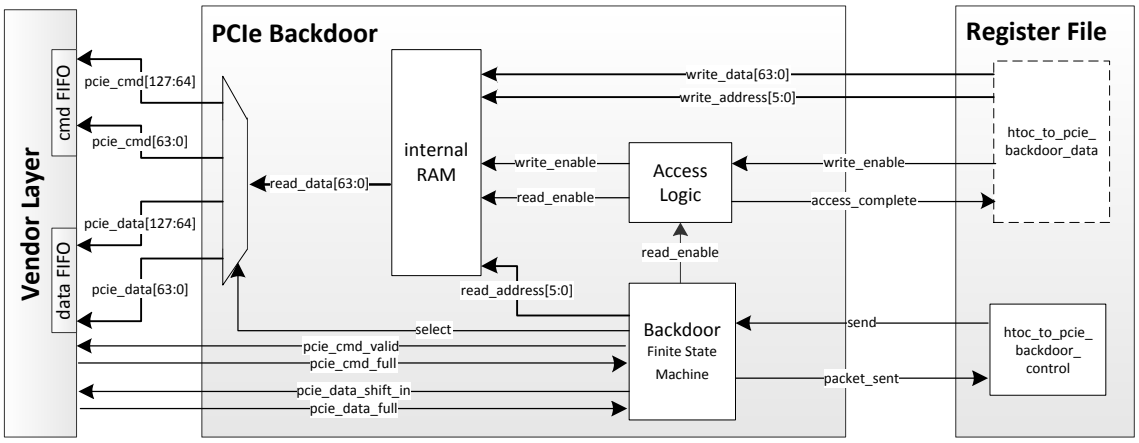

Figure 2.15: PCIe Backdoor Overview

insertion of software generated packets via the PCIe Backdoor. This mechanism can be used to create configuration packets and configure the connected accelerator. The PCIe Bridge TX module is also responsible to detect interrupt packets on the outgoing PCIe traffic. As described in section 2.3.2 on page 22 all necessary information to build an interrupt packet are stored in the PCIe MSI configuration register. A MemWr packet with an address between `0xFD_00000000` and `FD_F8FF_FFFF` is interpreted as an interrupt and triggers a signal to the IP Core to generate an interrupt.

2.4.2 SMFU

The SMFU [25] is designed to span a PGAS between memory regions exported by remote nodes. Without additional software layers load and store operation to specific local memory addresses result in a remote memory access on the target remote node. Simplified, this module can be the central functional unit to map the accelerator's on-board memory into the host's own address space and thus an essential part of the NAA concept. From the point of view of the CPU and the application, only memory addresses are read and written to and the encapsulation of data in network packets, the transfer and routing within the EXTOLL network, the unpacking and the actual memory accesses on the remote node take place entirely in hardware.

However, in order for this mechanism to run without further intervention by the software, the SMFU must first be configured. For this purpose, so-called intervals are defined. An interval is described by a set of three registers, a start address, an end address and a control register. The left side of fig. 2.17 on the next page shows the

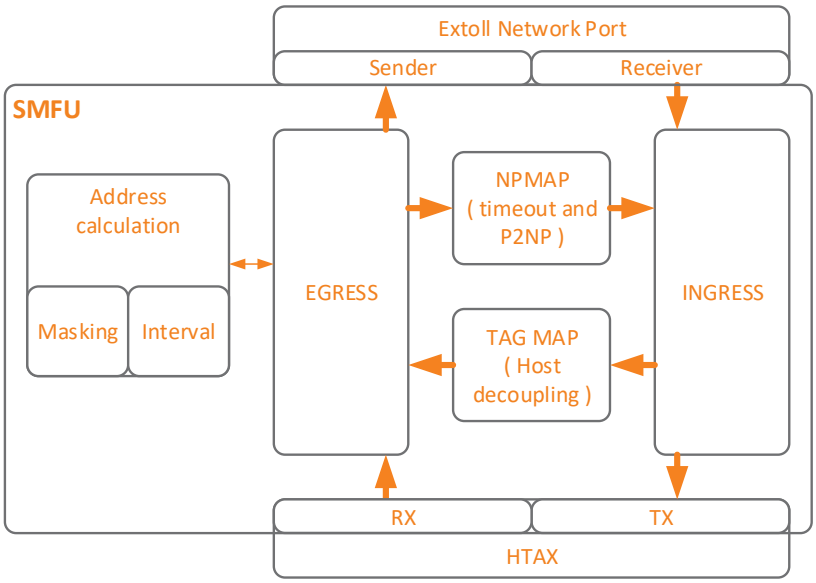

Figure 2.16: SMFU2 Overview with Ingress and Egress submodule [13]

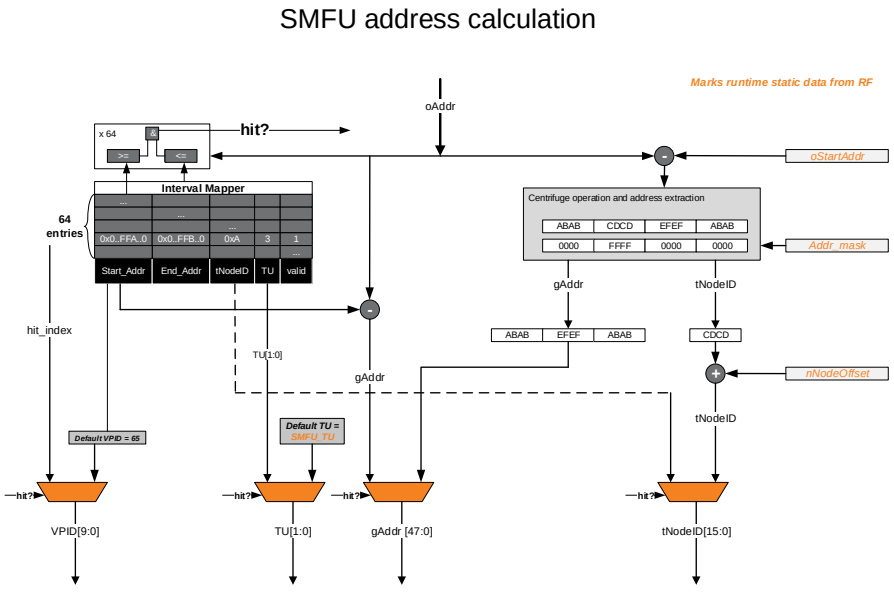

Figure 2.17: SMFU2 Address Calculation in Egress Submodule [13]

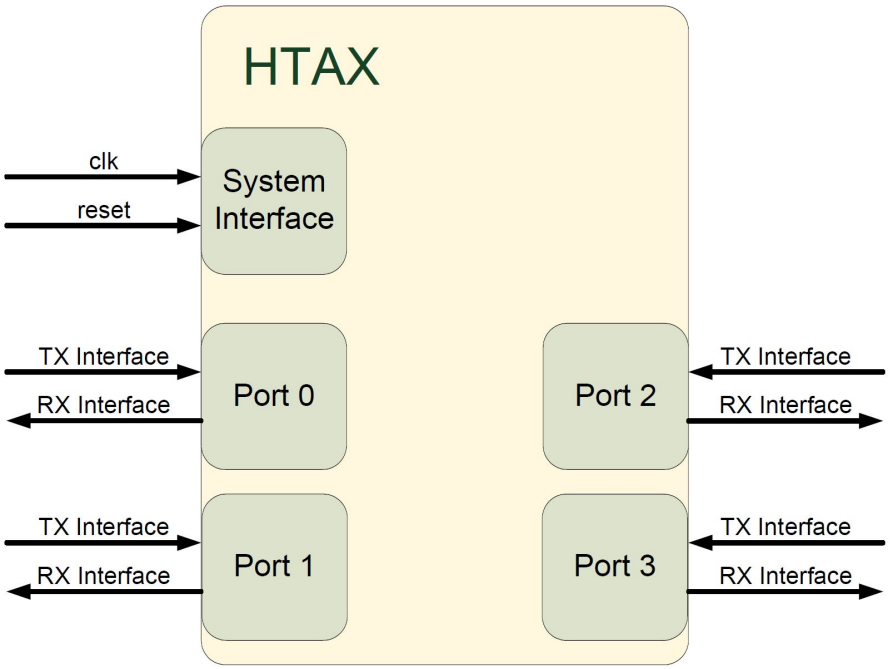

Figure 2.18: HTAX Module Overview [35]

interval scheme and the right side the mask based. Since only the interval scheme is of interest, only this will be discussed in detail. The *Start_Addr* describes the beginning of the interval at local host system's physical address space. Respectively the *End_Addr* describes the last address within the interval of the local host system's physical memory. The target node can be set in the control register and describes to which remote node this local interval of memory belongs to. If all registers are configured for an interval, each incoming packet is compared. If the address within the packet is within an interval, this packet is sent over the network. But first an address conversion is done. In a later chapter the way how these intervals and the address translation are used in connection with the accelerator will be explained in detail in section 4.3 on page 74.

2.4.3 HTAX

HyperTransport Advanced X-Bar (HTAX) [35] is a on-chip switching module that connects modules with each other. In the EXTOLL design, the HTAX is in between the host interface (PCIe or HT) and the functional units. It is based on a Crossbar and each input port can sent to each output port. A base structure can be seen in fig. 2.18. All modules connected to the HTAX have either a receive, a transmit

interface or both. HTAX input to output routing is based on intervals. Each input port has a set of software configurable register to mark a base and a upper limit for the interval mapping. An other register enables/disables the interval and selects the outport. An outport is connected either to a functional unit or the host interface. These intervals define the target functional unit of incoming PCIe packets and also define the host interface channel of outgoing packets from the functional units.

Within the project to be realized the NAA, this module is responsible for giving the accelerator (KNC) the possibility to access the functional units of the EXTOLL NIC. For this the above described intervals are set cleverly and allow the KNC to access certain address ranges directly and to use the highly specialized functional units. How this looks in detail, will be described in a later chapter section 4.5 on page 78.

2.4.4 X-Bar

The Crossbar is the core unit to provide direct connection to other EXTOLL nodes. It has eleven inports and outports. Four ports are connected to the functional units and the remaining seven ports are for the connection to the other nodes. Seven nodes are chosen to provide the 6 links for a 3D torus and one additional link that can be used to connect specialized hardware components like FPGAs without breaking up the torus topology. Each of the outgoing links has a width of 12 lanes and each lane has a link speed of 8.4 Gbit which adds up to 100Gbit per link. The inports support table based routing and each inport has its own exclusive routing table to allow routing decisions per inport to support complex routings. To discover the network or to set up a specific kind of routing the Extoll Management Program (EMP) can be used. It makes use of the remote access to each tourmalets register file for configuration of the routing table entries for each inport at each node of the network.

The ability to change both the routing of a node and its configuration over the network proves to be a great advantage for the NAA, as it eliminates the need for a control entity within the NAA node to make the necessary configurations via a side-band interface. This allows the NAA to be fully configured over the network in-band.

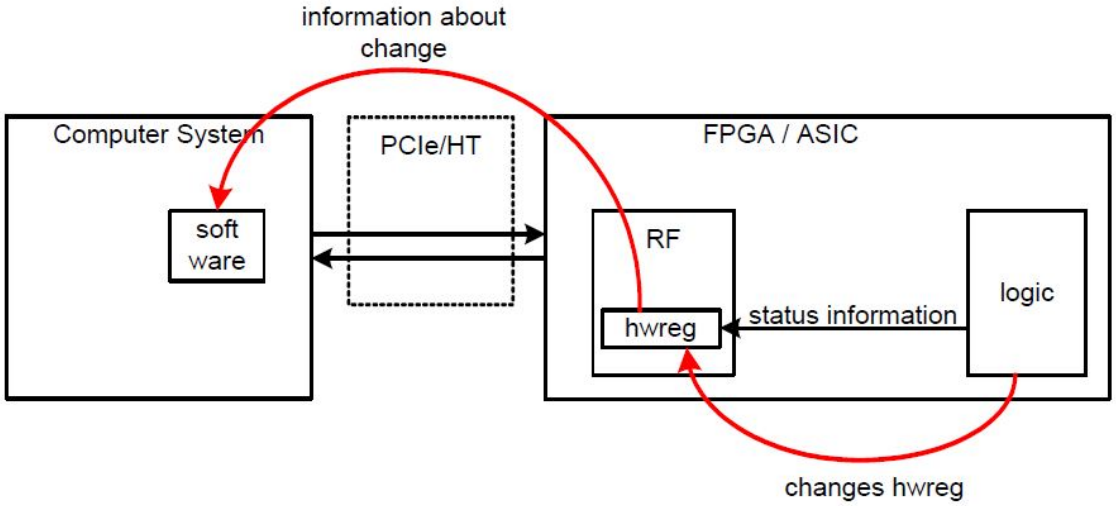

Figure 2.19: SNQ Functional Overview [30]

2.4.5 SNQ

It's important for every I/O device to notify the driver for changes on the device. The common approach is the use of interrupts. The current process running on the CPU is stopped after receiving of an interrupt and an interrupt handler is executed instead. The first task of this handler is to determine the reason for this interrupt by reading registers on the device with I/O operation on the host interface. After values returned to the interrupt handler the relevant actions are executed like reading a received network packet from the device into main memory. If the handler is done, the process that was interrupted is continued again.

Even though the above described approach is very simple it requires multiple I/O operations on the device and increase the time the interrupt handler blocks the CPU. To decrease the interrupt service time, the System Notification Queue (SNQ) [30] was designed. Instead of reading the status bits of the EXTOLL card, all information required to handle the interrupt are written to main memory before the interrupt is generated on the host interface. The main memory read latency is by far smaller than the latency on the host interface.

Since the accelerator of the NAA must also be able to send interrupts, this unit is a central component. A NAA accelerator has no direct connection to a CPU which can react on the interrupt. Therefore the interrupt information must be tunneled over the network. How this mechanism works exactly can be found in section 4.4 on page 76.

2.4.6 RMA2

To provide hardware accelerated single-sided RDMA functionality the Remote Memory Access (RMA) unit [43] supports DMA engines to read from and write to pinned memory regions. In this context, a *put* operation sends data from a local buffer with a pinned address to a remote memory address, whereas a *get* operation reads data from a remote memory location and writes the data into the local memory. The addresses of the buffers are either physical addresses which do not request address translation or virtual addresses that require translation by the Adress Translation Unit (ATU) [30]. During the initialization, also called registration of the pinned buffer, the virtual-to-physical address translation is cached in the ATU module on-chip to allow for a fast virtual-to-physical translation in hardware without a request to the operation system. The CPU is only involved on the sending side by writing a descriptor to the RMA unit with all necessary information, like local and remote address, target node and number of bytes to send. Like a DMA engine, the RMA unit transfers the data from the buffers autonomously without further intervention of the CPU between the remote and the local node or vice versa. Compared to a two-sided communication, whereas the receiving side is waiting for the data, the receiving side of the single-side communication is not aware that the content of the pinned buffer has changed or was accessed. The process that has initiated the *put* operation is informed by *notifications* that e.g. all data was read from local memory and the buffer can be reused. In the same way, the target of a *get* operation can be informed, that the buffer was read and is ready for new values. All submodules involved can generate notifications to inform the local node that the modules part of the *put* or *get* operation has completed.

Beside the RDMA functionality of the RMA another feature is very important for this work and is called Remote Registerfile Access (RRA). The Register File (RF) described in section 2.4.7 on the following page holds status and control registers and software accessible buffers for all modules and functional units of the EXTOLL design. A flag in the RMA descriptor marks a *put* or *get* request as a RRA and the address from the descriptor is interpreted in the Completer and Responder as an address inside the RF. The HTAX ports forward the read or write request from the Completer or Responder to the RF instead towards the host interface. With this mechanism each EXTOLL card can access another EXTOLL card's RF and retrieve status information or reconfigure hardware modules like routing table entries in the

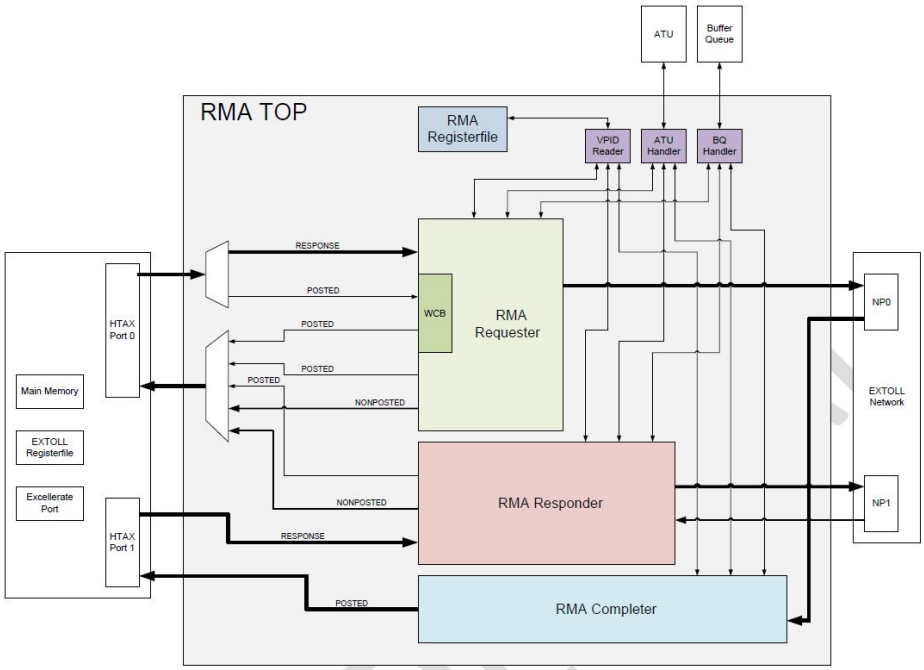

Figure 2.20: RMA2 Submodule Overview [12]

Crossbar or access to software accessible buffers like the PCIe Backdoor. As an accelerator node has no CPU to initialize itself all configuration modifications use the above described RMA unit with the RRA feature for configuration purposes.

2.4.7 Register File

To reconfigure and get access to status information of the modules the RF[30] provides a software accessible interface to retrieve information and write configurations. Each module can have its own small RF. The basic element of a RegisterFile is the register. Each of them is 64bit wide and each register can be further divided into individual fields of any size and position within the 64bit. Software and hardware read and write access can be configured individually for each field. This allows for registers that can only be written by hardware like a packet counter and only be read by software. Registers that can only be written by software and read by hardware enables configuration registers to configure a Finite State Machine (FSM) for example.

All small RFs distributed across the modules are connected with a centralized RF wrapper. Each register in a distributed RF has its unique address and can be accessed like a piece of memory. Figure 2.21 on the next page shows an example of

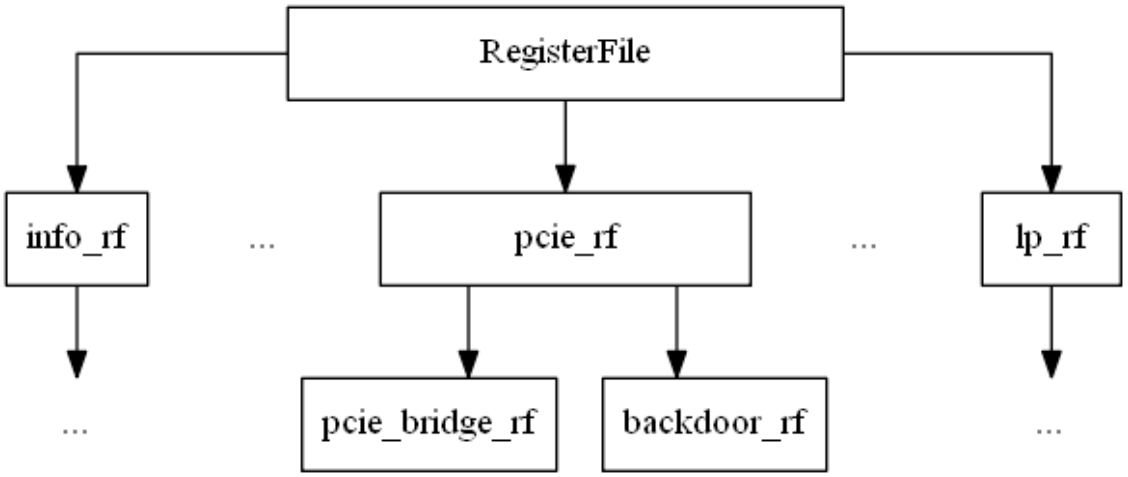

Figure 2.21: Registerfile Hierarchy Overview

the distributed RFs in the PCIe Bridge with the PCIe Backdoor RF and the Vendor Layer RF. The address of a memory packet targeting the RF is used to address a register at a distributed RF and triggers either a write enable or a read enable depending on the memory packet request type. This packet can either come from the host interface or from the network via a RRA transaction from the RMA.

Especially the second source, from the network, is of particular interest for this work. This allows for a remote configuration from anywhere inside the EXTOLL network. Because the accelerator node design has no CPU, all initialization tasks of the remote node have to be performed by the remote host and target the accelerator node.

2.5 Cooling technologies

A large part of this thesis deals with cooling of electronic components. Therefore, this section provides some background information about common topics. First, different cooling media are contrasted. These include air, water, special fluids and also solids such as cold plates.

2.5.1 Air

Air is without question the most common way to cool electronic devices. But unfortunately, air has some properties that make it inappropriate for high density cooling applications. In many applications where power consumption is low, air is often

sufficient to keep the device in a reasonable temperature range. Natural or forced convection is usually sufficient for this purpose. Natural convection is a process that transports heat away from the device. The heated air has a lower density than the surrounding cold air and rises. As a result, cooler air can follow. This effect does not require any external energy to be set in motion, but the cooling effect is limited. The thermal conductivity of air is very low with a value of 0.0262 $W/m \cdot K$ and is more an insulator than a coolant. This means that air above the device accumulates despite convection and increases the temperature of the device. To counteract this, one can create forced convection through fans and accelerate the removal of the warm air. This increases the cooling performance, but this effect is also made difficult by another physical property of air. Due to the very low heat capacity, the temperature of the air rises quickly and can therefore only absorb a small amount of energy before the temperature rises to a level that is harmful to the device. To compensate for this, a very large volume flow is required, which must be generated by powerful fans. Furthermore, the air must first be cooled down by air conditioning systems, which also consume a lot of energy. Nevertheless, air is by far the simplest but also the most energy-hungry way to cool electronics.

2.5.2 Water

Another medium to cool electronic devices is water. Since water is known to be conductive and consequently destroys these devices, it cannot be used directly for cooling. Instead, water is used purely as a transport medium for thermal energy without direct contact with the hot surface. For this purpose, metal plates through which water flows, so-called cold plates, are used. These are mechanically adapted exactly to the devices surface to be cooled. For a better heat transfer, there is a heat-conducting paste between the cold plate and the component to be cooled to ensure good heat transfer. With these cold plates, the much better properties of water can be exploited for cooling. This includes the much higher heat capacity of water. It can absorb four times more energy than air and has a much higher density. However, in order to effectively transfer heat from the metal cold plate into the water, a large number of small tubes or capillaries are required. This creates a great deal of friction within the cold plate and powerful pumps are needed to build up the necessary pressure and flow.

As with air cooling, water must first be brought to the correct temperature. De-

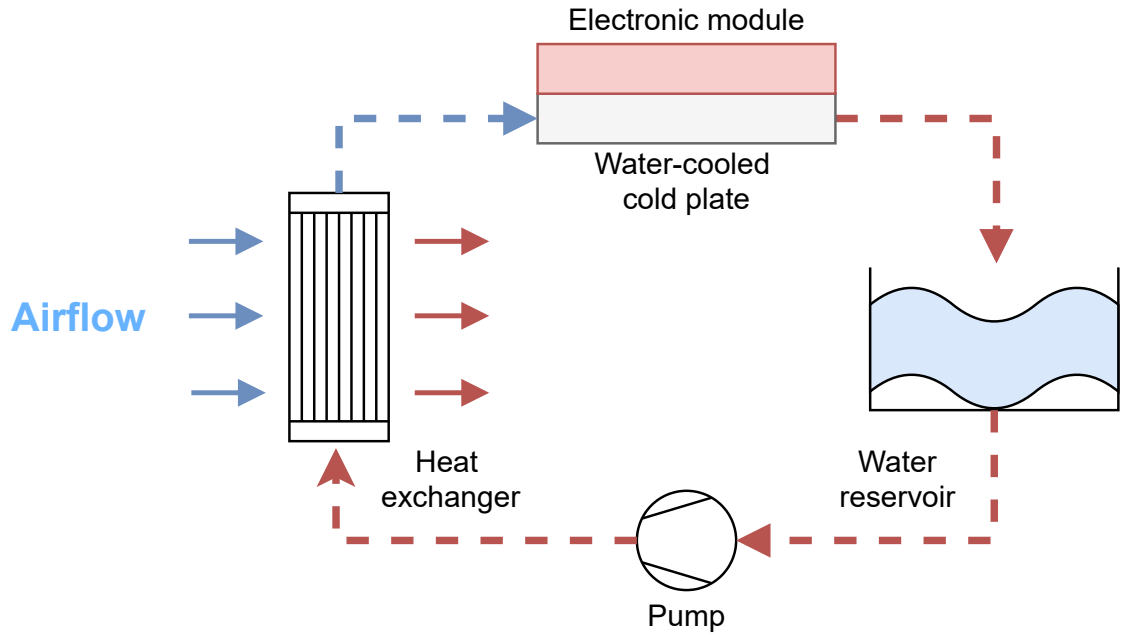

Figure 2.22: Cooling Flow Cold Plate

pending on the components to be cooled, the water can be cooled down to very low temperatures in order to absorb a maximum of energy. Another method is to release the energy of the heated water into the environment through a water-air heat exchanger. This method is called warm water cooling, because in contrast to the previously mentioned method, the average water temperature is higher than that of cooled water. Warm water cooling has the immense advantage that it does not require expensive and energy hungry cooling machines. Warm water cooling can easily be used, for example, to support the heating system and helps to increase the power efficiency.

With all the advantages that water brings with it, it still has a few disadvantages. The coldplates must be matched exactly to the height profile of the equipment to be cooled. If the arrangement of the components changes during the design process, a new coldplate must also be made. Depending on the heat removal requirements, relatively thick coldplates may be needed to provide sufficient water flow. This reduces the packing density. In addition, great care must be taken to ensure that water cannot leak from the cooling system. This can lead to short circuits and the destruction of components. A different class of coolant tries to circumvent some of these circumstances and are discussed in the following section.

Property	Air	Water	Novec^{TM} 649 649	FC-72
Normal Boiling Point (°C)	-190	100	49	56
Thermal conductivity ($W/m \cdot K$)	0.0262	0.597	0.059	0.057
Liquid density (kg/m^3)	-	1000	1610	1680
Specific heat ($J/kg \cdot K$)	1000	4184	1103	1050
Latent heat (kJ/kg)	-	2264	88	88
Critical heat flux (kW/m^2)	-	1000	200	220
Global Warming Potential	-	-	1	9300

Table 2.1: Physical attributes of air, water and Novec^{TM} 649

2.5.3 Engineered Fluids

In addition to air and water, engineered fluids are widely used as coolant for high-power and high-density electronics. These fluids are developed with specific properties that either alter these of water or air, or add new properties like non-conductivity. Most of the engineered fluids are fluorinated hydrocarbons like perfluorcarbons (C_6F_{14}) or fluor ketons ($CF_3CF_2C(O)CF(CF_3)_2$). Not all hydrocarbons are suitable as coolant. The fluids require a dielectric constant near the value of air, non-toxicity to use in areas with humans, non-flameable, chemical compatibility with the electrical components and long term stability.

A special attention is payed to the environmental compatibility. Some fluorinated hydrocarbons are ozone depleting or act as a greenhouse gas. An important property in this context is the Global Warming Potential (GWP) which is a measure how harmful 1 unit of this material is in relation to a reference gas like CO_2. As shown in table 2.1 Novec^{TM} 649 has a GWP of 1, which makes it as harmful as CO_2 whereas FC-72 has a GWP of 9300 and makes 1 unit of FC-72 as harmful as 9300 units of CO_2.

Their most interesting property are non-conductivity and no chemical reaction with other materials which make them perfect coolants for direct cooling. Compared to water were cold plates are used on top of the heat source like a chip's package the whole electronic device is immersed into the fluid and covers all components. This enables dense designs without the need of a cold plate or heat sinks. The coolant can be used to absorb the heat of the hot surface and use convection to transport the heat away from the surface. This can be done either by forced convection with a pump that circulate the liquid along the heat sources or by ordinary convection through displacement of heated liquid by heavier and colder liquid. The

described procedure is called *1-phase-cooling* because the liquid remains in the same liquid phase. Comparing the values in table 2.1 on the preceding page of water and Novec^{TM} 649 , we see a difference for the thermal conductivity and the specific heat of the materials. Water has a 10× higher ability to transport heat away from a heat source as Novec^{TM} 649 and can absorb 4× more heat until the temperature of the material increases. Even though water has superior physical attributes the electrical conductivity prohibits it as a direct cooling liquid.

To compensate this drawback Novec^{TM} 649 can be used as a coolant for *2-phase-cooling* and change its phase from liquid into a vapor phase. With this procedure Novec^{TM} 649 can take advantage of the latent heat during vaporization on hot surfaces. Another important measure is the boiling point. The boiling point of water with 100 $°C$ and the resulting junction temperature on the die is to high to use 2-phase-cooling with water. With a boiling point of 49 $°C$ the junction temperature on the die is in a range of 70 $°C$ to 80 $°C$ depending on the thermal conductivity of the package and thermal interface material and is sufficient for cooling of processors and accelerator chip.

As an example, the Intel Xeon Phi has a package size of $4cm \times 4cm$ or 16 cm^2 and has a thermal design power of $300W$. This results in an heat flux of 180 kW/m^2 and is slightly below the critical heat flux of Novec^{TM} 649 which is around 200 kW/m^2 [24]. If the package is removed, the much smaller die area would exceed the critical heat flux of Novec^{TM} 649 . For all applications that exceed the critical heat flux for a plain surface, boiling enhancement coatings Boiling Enhancement Coating (BEC) can be used. These are materials with a rough surface increasing the overall surface area and increase the critical heat flux.

Compared to 1-phase-cooling where the liquid itself transports the heat and is cooled down at a heat exchanger, in 2-phase cooling two cooling loops exist. The primary loop consists of the liquid that vaporizes at the hot surface and is condensed back at a cold surface like a cooling coil. This transports the heat via the vapor to the cooling coil and the heat is transfered out of the system with water running through the cooling coil. The same conditions apply to the condensation on to cold surface as for vaporization on a hot surface. The latent heat of the vapor is absorbed on the cooling coil and therefor is a very efficient heat transfer due to the phase change back from vapor into liquid.

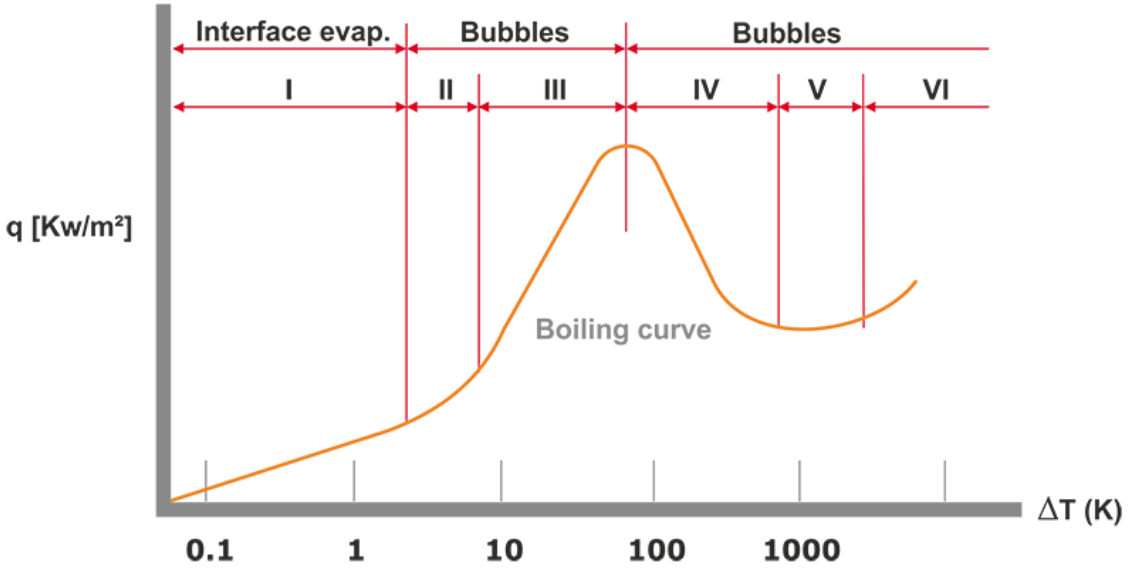

Figure 2.23: Boiling curve of heated liquid[59]

2.5.4 Boiling

Since a more detailed knowledge of the boiling process is necessary for the further course of the work, the individual phases will be examined in more detail in this section. Boiling undergoes different phases as shown in fig. 2.23. If the temperature of the hot surface is minimal higher than the boiling temperature of the liquid, the heat transfer will primary consist of convection through displacement of the heated liquid (I + II). Nucleate boiling starts beyond this point when the temperature of the hot surface increases (III). The heat flux has enough energy to change the phase of the liquid into vapor and the energy is carried in the vapor towards the liquid's surface. As described above, this is the most efficient way to transport heat through the latent heat required for the phase change. If the temperature of the hot surface further increases (between III-IV), there is a point, called critical heat flux, at which the nucleate boiling changes into a film boiling. A layer of vapor now covers the hot surface and dramatically reduces the heat transfer capability between the surface and the liquid (V). This effect increases the surface temperature and may exceed the operating temperatures and can damage the electronic device. An operating point chosen slightly below the critical heat flux is the most efficient way to transfer heat. The critical heat flux is a measured in W/m^2 and depends on the surface area.

Chapter 3

State of the Art

This chapter introduces some real-world implementations of the concepts described in the previous chapter. Different intra and inter node communications are shown with the focus on communication between accelerators. Real-world cooling solutions are shown at the end of this chapter to demonstrate how these solutions are used in the field. As the application of this thesis is in the HPC segment, the Green500[28] list is used to determine candidates as examples for energy-efficient super-computers.

3.1 Intra Node Communication

The TOP500 list is dominated by heterogeneous systems, that use CPUs as well as GPGPUs for their processing power. Therefor each node of such a system consists of a set of CPUs and a set of GPGPUs. As described in the previous chapter, accelerators use an offloading model with tasks or kernels send to the accelerator, complete the task and hand over the path of execution back to the processor. This offloading requires a tremendous amount of communication between the processor or host memory and the accelerator. This communication involves also the data the accelerator computes upon. As the trend is towards each cluster node has not only one but many accelerators the intra node communication between accelerators become more and more important. Different concepts are now common practice and the following sections describe them briefly.

3.1.1 NVIDIA GPUDirect™ peer-to-peer

GPUDirect™P2P was first introduced and developed by NVIDIA, but is now a synonym for similar concepts that take advantage of direct access to the GPUs memory or direct access to other device's memory. Figure 3.1 on the next page shows the basic concept of two GPUs that can communicate with each other over the CPUs chipset. Prior to GPUDirect P2P, accelerator-to-accelerator communication involves *memcopy* operations and cause additional overhead. Each accelerator has first to copy its data into main memory and the other accelerator has to read the data from main memory. As the number of accelerators per cluster note increases, this kind of unnecessary communication can cause contention on the datapath between the accelerators and main memory. To avoid this *memcopy* operation, the accelerator exposes its complete onboard-memory to the host system instead of small memory regions for device communication. As all devices share the same physical host address space, an accelerator can directly access physical addresses of an other accelerator in the same host system.

The DMA controllers of the accelerators can now be programmed with addresses from other accelerators memory space and data transfer between the devices is managed by the devices itself. This reduces the amount of CPU intervention during the device communication and significantly increases the performance. Beside the reduced transfer latency and the increased throughput, GPUDirect P2P makes co-

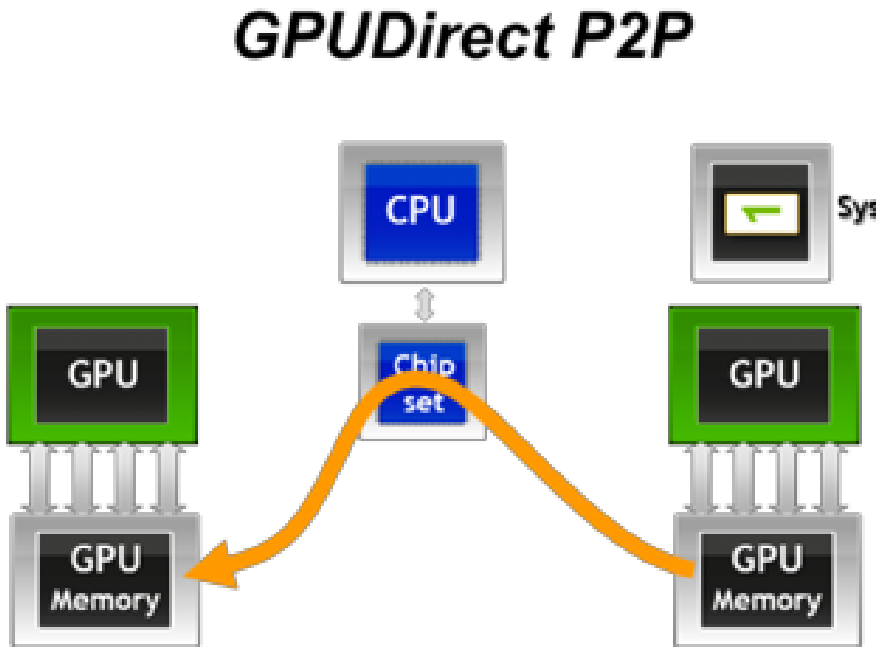

Figure 3.1: Intra-node communication with GPUDirect Peer-to-Peer[20]

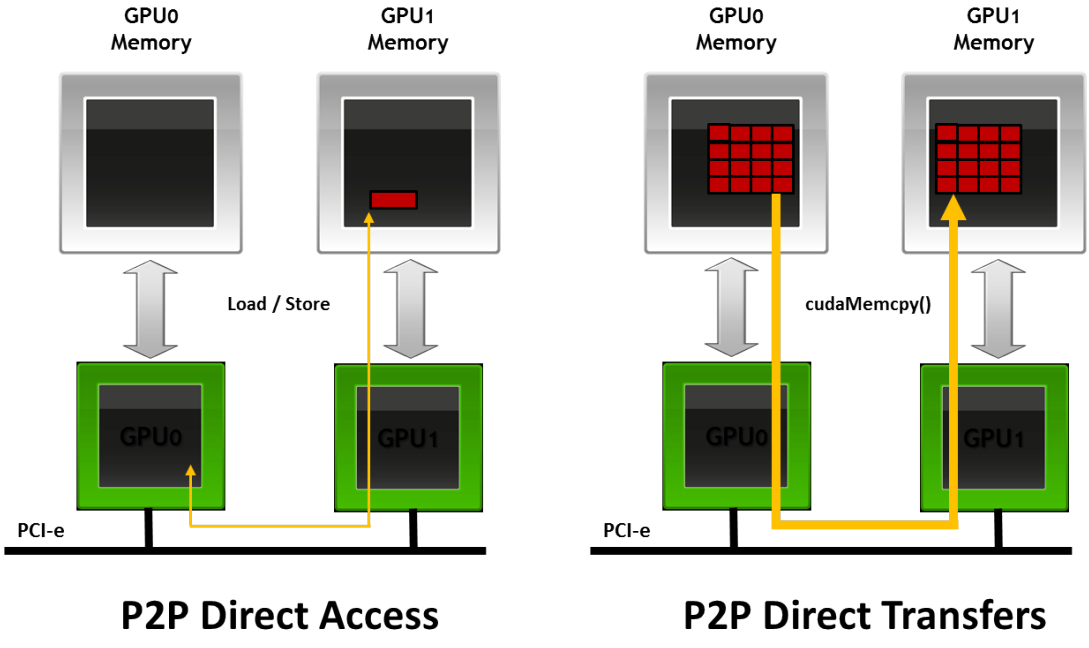

Figure 3.2: NVIDIA GPUDirect Peer-to-Peer (P2P) Communication Between GPUs on the Same PCIe Bus[20]

operation between GPUs easier. Instead of the offloading model a work flow model can be used. An accelerator can work with the data locally accessible to it and make the calculated results available to one or more other accelerators. This can be achieved either by writing the data directly into the device memory of another accelerator or by making the memory address known to the other accelerator, which can then read the data autonomously. Not only other accelerators but also the CPU can access data in this way. Either the accelerator writes the data directly into the main memory or the CPU reads the data directly from the device memory.

In context of NAA, the possibility to communicate without CPU intervention and the direct communication between accelerators with GPUDirect are in sync with the objectives of this thesis, but has the limitation to a single host system. Also the dynamic assignment of CPUs to accelerators can't be satisfied with this approach. Furthermore the number of accelerators per node is limited by the number of available PCIe lanes from the CPU or chipset. To use more accelerators than the CPU or chipset provides, PCIe switches can be used.

3.1.2 Multi-GPU Server

At the time of this writing, 8 GPU systems are commercially available like the Tyan FT77AB7059[62] or Supermicro 4027GR[58] systems. These dual socket hosts do not provide enough PCIe lanes to directly connect all 8 GPUs with the processors. Instead, PCIe switches like the PEX9700[7] can be used to connect more devices than PCIe lanes present on the CPUs or chipsets.

As shown in Figure 3.3 on the facing page these 8 GPUs are connected via PCIe switches to the CPUs. This leads to four different communication paths between the accelerators. One path is between two adjacent GPUs connected to the same PCIe switch (green arrow). To reach a non-adjacent GPU on the same CPU, the local switch, the Root Ports of the CPU and the distant switch have to be traversed (purple arrow). The third path is between GPUs connected to different CPUs and communication has to traverse the InfiniBand NICs on both CPUs and the two PCIe switches on each CPU (blue arrow).

As done in the experiment by Ellis [22] the bandwidth and latency of four different paths is summarized in table 3.1 on the next page.

The communication between adjacent GPUs on the same switch (green arrow) has the best performance as expected. With only a drop of 0.8 GB/s and an increase

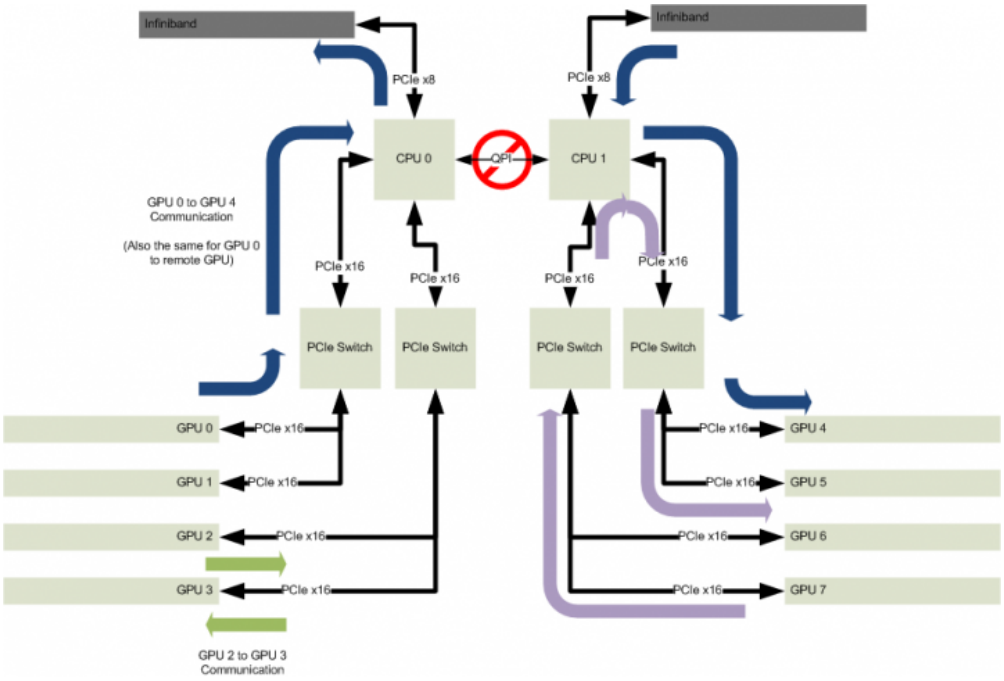

Figure 3.3: 8 GPU System Block Diagram [22]

Path	Bandwidth (GB/s)	Latency (us)
green	10.6	7.7
purple	9.8	7.9
blue (IB)	3.4/3.7*	9.8*
QPI	4.0	32.1

Table 3.1: Performance measurements for different PCIe communication paths

in latency of 0.2 µs the path over two switches and the Root Complex performs second best (purple arrow). The reason for the decreased performance comes from the additional switches in the path between the GPUs. Their processing time and the routing effort between the Root Ports are responsible for the drop in bandwidth and increase in latency. The table also highlights that a communication across an interconnection network performs better than a direct connection between the CPUs with Quick Path Interconnect (QPI)[68]. This is due to the fact that even QPI performs worst in a socket-to-socket PCIe transfer due to buffering limitations in the Sandy Bridge / Ivy Bridge architecture. As the authors of the experiment did not publish results for the InfiniBand path, the values in the table are expected results in combination with a similar experiment taken from Rossetti [56]. The author measured the communication between two GPUs over InfiniBand in an Ivy Bridge system. The peak bandwidth of the used InfiniBand FDR HCA is 6.1 GB/s, but only 3.4 GB/s or 3.7 GB/s with increased GPU clock was achieved. This reduced performance comes again from the buffer issues of the Ivy Bridge architecture. Nevertheless, the expected latency of 9.8 µs is way less than the 32.1 µs latency of the QPI path. The value is expected from the InfiniBand host-to-host latency of 1.9 µs and the latency from the switches and Root Ports from the path between two GPUs on different PCIe switches.

For our intention to build a NAA the connection path over InfiniBand sounds promising as it makes no difference if the GPU "behind" the InfiniBand interconnect is in the same host or somewhere in the network. This advantage comes at the price of a difficult programmability. The application programmer has to pay attention how the application is decomposed and distributed over the GPUs. Depending on the path the latency and bandwidth difference have a tremendous impact on how much computation is required to fully hide the transfer overhead. In addition, remote communication over InfiniBand requires explicit handling of all the network related overhead like buffer management.

3.1.3 PCIe Switch Trees

To support a greater number of accelerators, PCIe switches can be arranged like a tree as shown in fig. 3.4 on the facing page. In this example 8 GPUs are connected to each other with a 3-level binary tree whose root is the Root Complex of the CPU. This connection scheme has the benefit of avoiding the limitations of CPU peer-to-

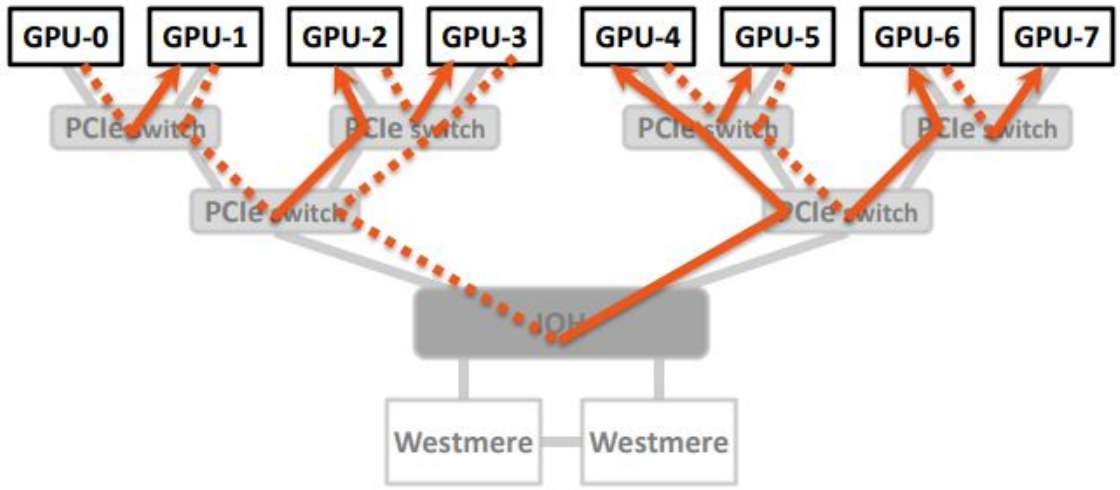

Figure 3.4: 8 GPU System in a tree Block Diagram [36]

peer PCIe traffic as explained in the previous section. A work from Micikevicius [36] deals with the different communication paths between GPUs connected in a tree. Two GPUs connected to the same PCIe switch have the full PCIe bandwidth and the lowest possible latency. All other communication paths to more distant GPUs require the traversal of additional switches which increases the latency. In contrast to the latency the maximum bandwidth between all GPUs is still the same for a single simultaneous transfer between GPUs. If multiple transfers happen at the same time, they all share the same limited bandwidth. As the bandwidth does not increase from the leave switches towards the root, this effect becomes more significant the further the GPU are away from each other. Distance in this regard means how many levels of the tree have to be ascended and descended. To circumvent this limitation the number of conflicting transfers that share the same switch and the same direction has the be reduced. One way to avoid this conflict is to use the proposed communication pattern from Micikevicius [36] which they call the *Left-Right Approach*. As shown in the example picture on 49 the communication phase is split into two parts, the *exchange left* and *exchange right* part. All GPUs exchange their data to the right most neighbor and after that to the left most neighbor. With this split no transfer uses the same direction across a switch twice and the full bandwidth is achieved for all communications.

For the NAA the latency and bandwidth between two adjacent GPUs sounds very interesting and the removal of the PCIe switch in between would be the best possible starting point for the NAA development. Nevertheless, the inflexible communication

in two parts (left and right transfers) is a serious restriction for the applications. The latency between the GPUs depends on their position in the tree and it is hard to determine how much latency to hide. This restriction is also in contrast to the goals of a NAA design.

3.2 Inter Node Communication

The first part of this chapter is about communication inside of a cluster node and the different approaches to connect as many accelerators inside a single host. One of the presented procedures already used an interconnection network to transfer data between GPUs. The following sections describe other approaches to support high bandwidth and low latency communication over a fabric to allow transfers between GPUs in different hosts. Most of them are based on custom hardware designs evaluated in FPGA prototypes to circumvent limitation of Commodity Off-The-Shelf (COTS) solutions. For the sake of completeness also these solutions are presented to argue differing approaches of academic research.

3.2.1 GPUDirect RDMA IB

Most of the scientific simulations are to large to fit into a single accelerator's device memory. Even a multi-gpu node's combined memory is still not enough for the demanding applications of the exascale era. As in the past, the cluster nodes have connected to an interconnection network and work in parallel on smaller chunks of the problem. Adopting this procedure is very easy for homogeneous clusters with CPUs only, but it becomes cumbersome for GPU equipped systems in heterogeneous clusters. Transferring data from one GPU to another GPU in a different node requires several *memcopy* operations as fig. 3.5 on the next page shows. The sending GPU copies data into main memory, the NIC of the used interconnect reads it from main memory, transfers the data to the remote node's main memory, the remote node's GPU receives the data and stores it in its device memory.

To avoid all this *memcopy* operations, GPUDirect from section 3.1.1 on page 44 can not only target other GPUs, instead the target can be a RDMA capable NIC like InfiniBand. Using GPUDirect in conjunction with InfiniBand is called GPUDirect RDMA. InfiniBand is a high-bandwidth low-latency interconnection network and 40% of all TOPP500[61] cluster systems are using this fabric. Applications create

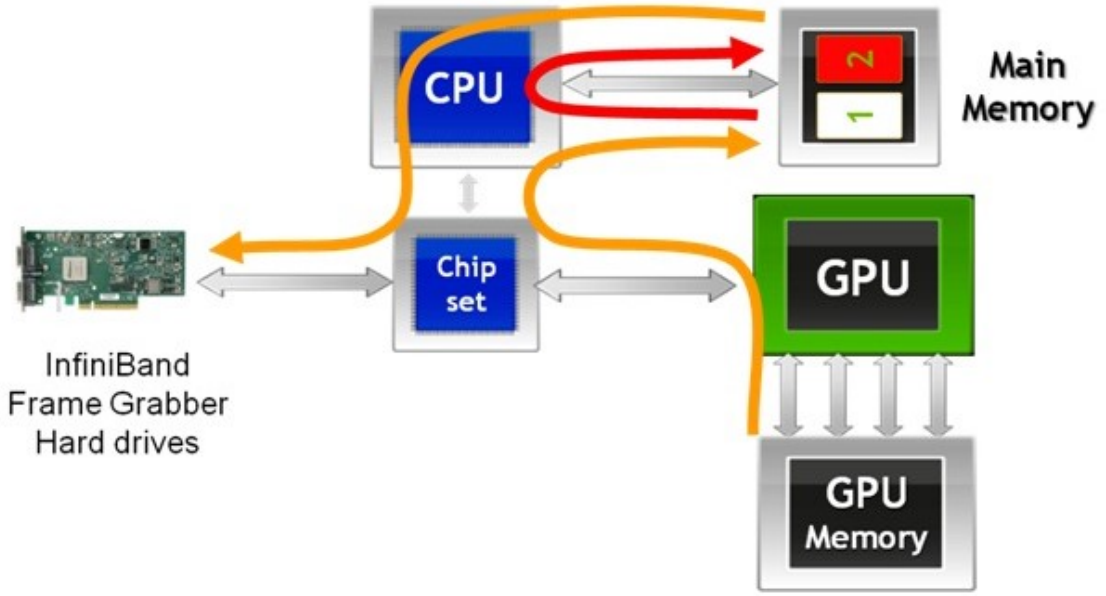

Figure 3.5: Inter-node communication without GPUDirect [20]

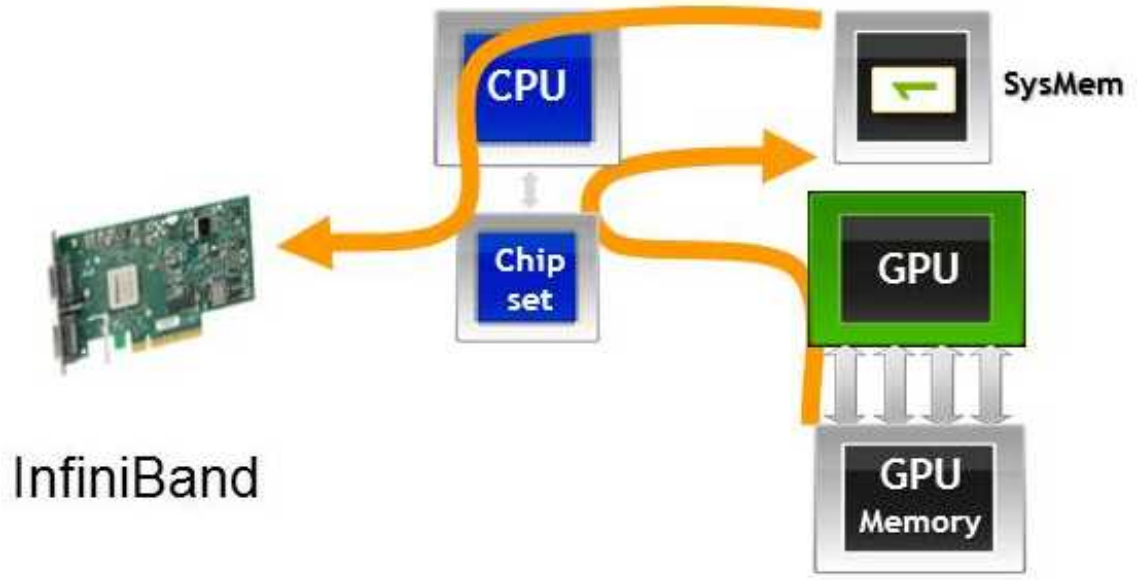

Figure 3.6: Inter-node communication with GPUDirect[20]

work requests and store them in so-called *work queue pairs*. They consist of a submit queue and a corresponding completion queue. CPU sets up the submit queue entries and the NIC creates the completion queue entries. The application polls for completion queue entries determining if a request has been completed. GPU cores can't create these requests efficiently and as a result the CPU still drives all communication. The creation and handling of work request by the CPU is negligible as it avoids four *memcopy* operations. Instead of storing the data in a buffer in main memory, the work request can be set up in such a way that it reads the data directly from GPU device memory. On the receiving side, GPU's device memory stores all the received data and avoids additional unnecessary *memcopy* operations.

The work of Potluri et al. [52] evaluated the use of GPUDirect RDMA and compared it with the most prominent MPI library used in HPC environments MVAPICH2 [60]. This library uses the above described approach of buffers in host memory and *memcopy* operations to move the data to their final destination in GPU Memory. The experiments run by Potluri et al. [52] show a significant reduction in MPI_Send / MPI_Recv latency by 69% for small messages and an uni-directional bandwidth increase by a factor of two. Their modification to the existing MVAPICH2 library makes use of GPUDirect RDMA and avoids most *memcopy* operations. Also these authors stated, that reading from GPU memory requires special care as the used Sandy Bridge CPU has architectural limitations.

In context of this thesis, the ability to communicate with remote nodes is exactly what we need, but the still CPU driven communication and the additional software libraries make this approach less feasible for a NAA.

3.2.2 GPU Virtualization

Most current HPC clusters consist of GPU equipped nodes. Depending on the workload mix, utilizing all GPUs all the time is hard to achieve. GPU virtualization can reduce the number of GPUs in the cluster while improving their utilization in return. This in turn reduces the acquisition costs and the power consumption of idle GPUs in the cluster. Some nodes in the cluster are *GPU Server* and provide their services to non-GPU equipped nodes. Figure 3.7 on the next page shows the general architecture of this GPU virtualization. A cluster node without a GPU calls a standard GPU functions and the local node intercepts with a wrapper library this calls. The library uses the network to transfer the call to GPU functions over to the

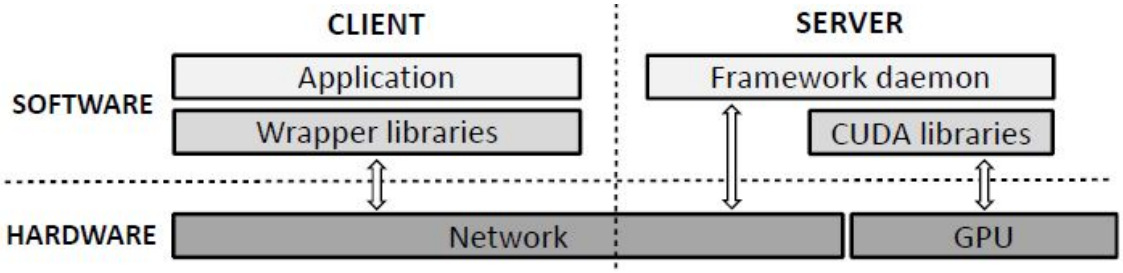

Figure 3.7: General architecture of remote GPU virtualization[53]

GPU Server and the GPU driver residing on the server executes the call and sends the result back to the requesting non-GPU node.

Reaño et al. [53] analyzed the impact of the network on this virtualization and used the rCUDA framework [21] for the case study. The framework uses the above described modular architecture. A communication layer on top of a Gigabit Ethernet or InfiniBand transfer layer communicates with the CUDA runtime of the client and server nodes. Their tests compared a native CUDA test case with a local GPU in the node and a test case where the application uses rCUDA to talk to a remote GPU server. The tests showed that the bandwidth reduction introduced by the interconnection network and the additional software layers is about 9.4% compared to the native bandwidth of a local GPU.

The usage model of multiple cluster nodes share a reduced number of GPU servers breaks up the fix n-to-m ratio between hosts and GPUs and allows for a dynamic resource allocation. The performance degradation of about 10% is not negligible but acceptable as the energy consumption is reduced and the utilization increases. The additional software layers handling all virtualization and communication waste CPU cycles and are part of the 10% bandwidth loss. The share of software on this loss would be interesting to know as it could show if this virtualization could be used for the NAA

3.2.3 Non-Transparent Bridges

Another approach is to use PCIe directly as a network interconnect. This avoids dedicated NICs and protocol conversions from the PCIe protocol to Ethernet or Infiniband. As described in [54] PCIe was build as host interface to interconnect processors and I/O devices. This restricts the use as a network or fabric interconnect by design. One of this limitations is that all devices of a PCIe hierarchy have to

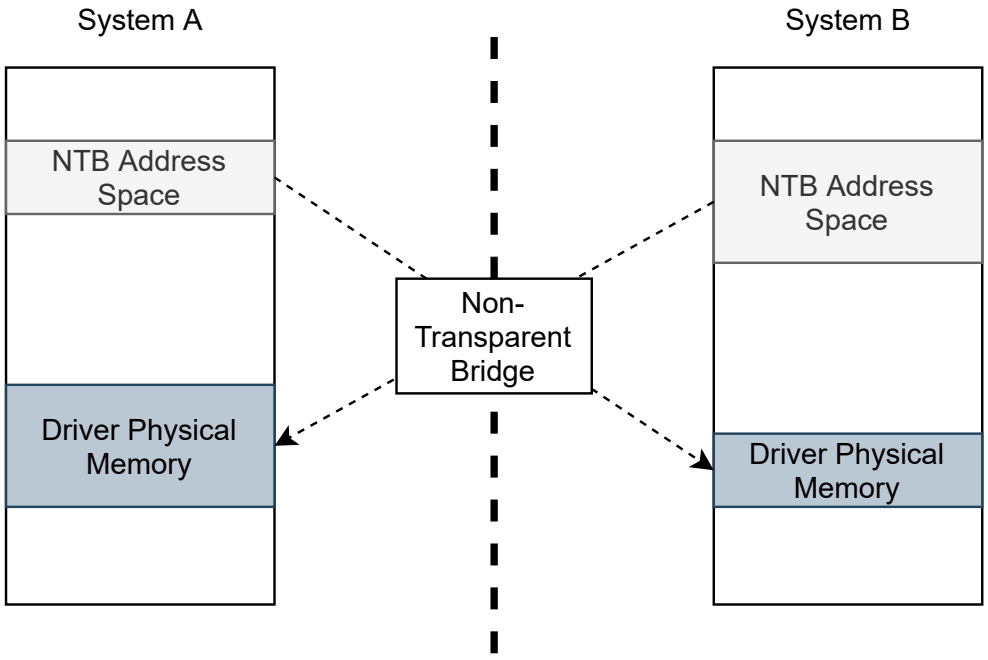

Figure 3.8: NTB address translation example

use the same clock source for their operations. Another host would bring in another clock which is asynchronous by nature and the relation between these clocks is not specified. Due to a second CPU in the hierarchy, each of the two Root Complexes treads the whole hierarchy as its own and makes a clean enumeration of all devices impossible. The next problem is the two separate memory domains of the hosts with no relation to each other. To solve these issues PCIe supports the concept of Non-Transparent Bridges (NTBs). In contrast to a Transparent Bridge, where the CPU can see trough the bridge and has access to all devices on the other side of the bridge a NTB is like an Endpoint and the bridge hides all devices behind it. This bridge type can connect two hierarchies with each other and eliminates the restrictions described above. To translate memory requests from one side to the other, a NTB provides scratch-pad registers, doorbell registers and an address translation service. The scratch-pad registers can be used to transfer information between both sides. Doorbell registers are used to trigger an interrupt on the other side to inform host software about an event. The address translation service uses the BARs of the NTB to define apertures from one side to the other. A mapping table stores all possible translations as shown in fig. 3.8 and incoming requests get translated to the memory domain of the remote side. This enables direct communication between devices on

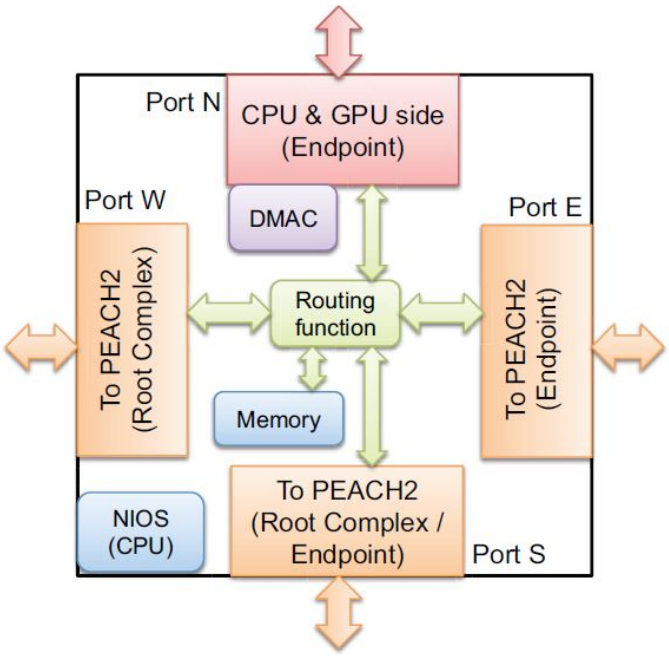

Figure 3.9: PEACH2 Architecture [27]

both hierarchies.

This approach can connect multiple host with each other, but requires a difficult mapping of mappings. The address translation table of a NTB defines a block for every hierarchy/host that is part of the PCIe cluster. A request to an address of host n targets an address inside block n. All these block addresses point to the NTB that separate the remote host from the PCIe fabric. The addresses received in the target NTB are translated to addresses for this domain. More levels of hierarchy requires a more complex address space division and is not straightforward.

NTB avoids protocol conversion to other interconnect formats and the usage of work queues like InfiniBand. Memory addresses can be used to directly address any remote PCIe device if a mapping exists. Nevertheless the lack of network features, bandwidth limitations, small number of nodes per fabric and the difficult routing inside the fabric make this a interesting way to interconnect devices, but is not feasible for the needs of this thesis.

3.2.4 PEACH2

To reduce the impact of protocol conversion from PCIe to InfiniBand the work of Hanawa et al. [27] developed an interconnect that forwards PCIe packets directly to

another node. The PCI Express Adaptive Communication Hub Version 2 (PEACH2) consist of a custom Altera StratixIV Field Programmable Gate Array (FPGA) board and implements routing and DMA functionality. Figure 3.9 on the previous page shows an overview of the FPGA design. To connect with other nodes each board has four PCIe Gen2 x8 links which are labeled North, West, East and South. As all interfaces use the PCIe protocol the ports either implement an Endpoint or a Root Port. According to the PCIe specification a Root Port-Endpoint pair is mandatory to exchange PCIe packets. In the presented implementation all West links are Root Ports and all East links are Endpoint. The North link is exclusively destined for CPU, GPU connectivity and implements an Endpoint. The East/West links are used to build a ring topology. The South link has a particular use and can be either configured as Endpoint or Root Port to connect two ring topologies with each other to extend the fabric.

Routing inside the network is done by address mapping of the PCIe requests. The Most Significant Bits (MSBs) of the address are used to calculate the target's Node ID. To simplify the routing implementation on the FPGA the protocol only transfers PCIe Memory Write Request (MemWr) packets as the routing-by-ID of Completion with Data (CplD) is not supported with the address mapping routing outlined by the authors. A so called driver supported *proxy write* is used to transform Memory Read Request (MemRd) packets into write transactions. Packets that reach their destination require a conversion from the global address space valid in the fabric to the address space of the local host address space.

The benchmark presented in the work was executed on eight Sandy-Bridge nodes with four NVIDIA®-Tesla® K20 and a PEACH2 in each node. A overview of the node setup is shown in fig. 3.10 on the facing page. The results report a latency of 1.9 µs for CPU-CPU DMA communication and a bandwidth of 3.5 GB/s which is 95% of the theoretical peak performance. In the GPU-GPU DMA communication benchmark the latency is 2.3 µs and the bandwidth drops down to only 0.8 GB/s. This is again caused by the buffering issues in the Sandy-Bridge architecture for peer-to-peer PCIe communication.

The work of Hanawa et al. [27] is very promising in *forwarding* PCIe packets between the nodes to avoid protocol conversions from InfiniBand to PCIe and vice versa. In contrast to DMA mechanics used by other approaches, this load/store mechanic is interesting for the NAA design. This avoids explicit programming for the given fabric interconnect used. Instead of explicit programming for MPI or

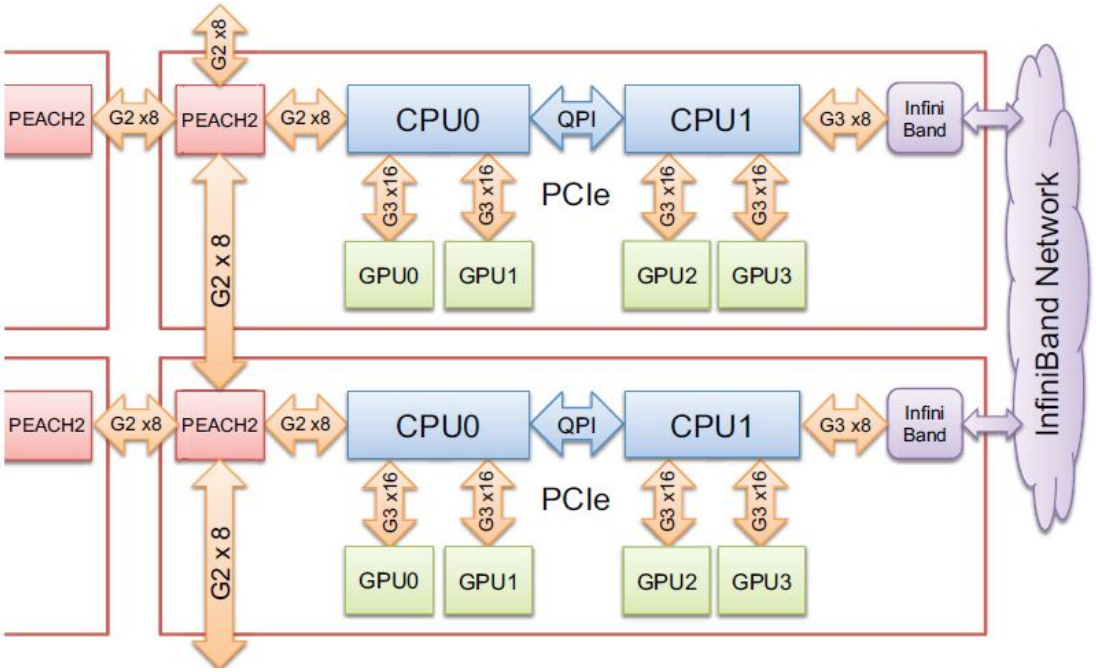

Figure 3.10: PEACH2 Architecture [27]

InfiniBand the software support by a driver is required to translate read requests into write requests (proxy writes) and for address translation from global address space into the local host address space. The ring topology restricts the network size to small node counts. The distance to other nodes increases linear with the node count and increases the latency. In case of non-adjacent communication the bandwidth between two nodes becomes a bottleneck. Like in section 3.1.3 on page 48 transfers share a common connection and each transfer receives a steady decreasing share of the available bandwidth. To circumvent this limitation, again, a restriction of communication direction can preserve the full bandwidth to a single transfer. But in a ring topology this is a severe restriction as only transfers into the same direction are allowed.

3.2.5 APEnet+

Another project from academic research is the APEnet+ card[4]. It's again a custom FPGA board and is part of a larger HPC initiative, the lattice QUantum chromo-dynamics ON Gpu (QUonG) project to boost the performance of Lattice Quantum Chromo-Dynamics (LQCD) applications. Instead of four links on the PEACH2 board, the APEnet+ card has six links to build a 3D torus network. In contrast to

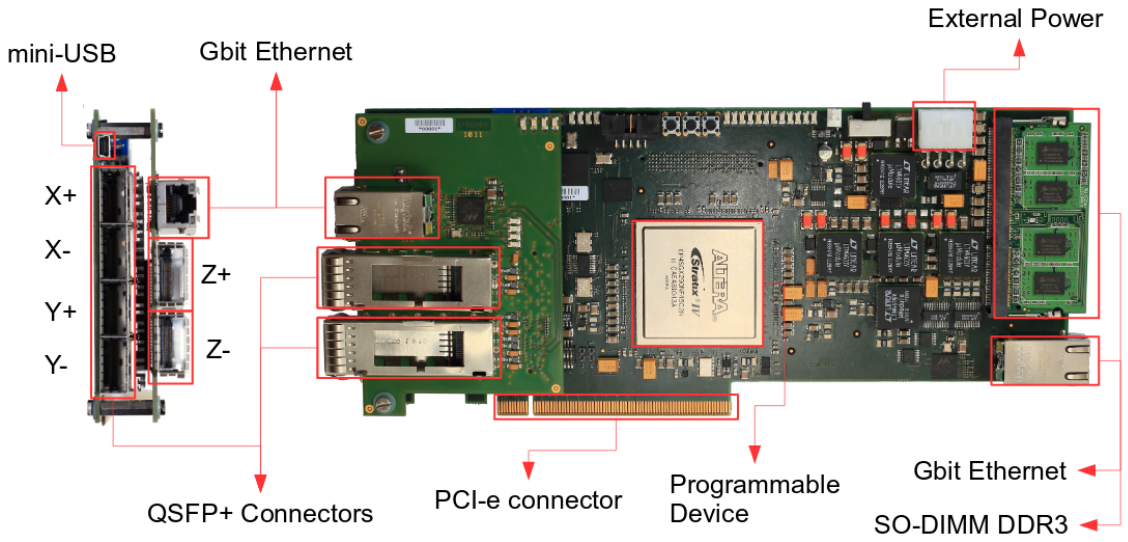

Figure 3.11: APEnet+ card [5]

InfiniBand, which relays on external switches to interconnect nodes, the APEnet+ card has a switch build-in in the FPGA design. Basically, the APEnet+ network is a *network of switches*. RDMA is supported with PUT and GET operations for zero-copy transfer of host-to-host or GPU-to-GPU memory. A micro controller translates virtual-to-physical addresses and manages the registered buffers for the zero-copy.

The board has a PCIe Gen2 x8 interface to the host main memory and CPU. Each link is implemented as a single QSFP+ connector with four lanes and has an unidirectional bandwidth of 34 Gbit/s. To avoid deadlocks in the network, a dimension-order-routing is implemented in the internal switch logic and credit based flow control is used in each link. A photo of the APEnet+ board is shown in fig. 3.11.

The performance of the implementation is tested with a benchmark over two nodes. The receiving node registers a buffer in either host or GPU memory and sends the buffer virtual address to the sender. The sender transfers its host or GPU memory buffer to the target. Four combinations are tested: host-to-host memory, host-to-GPU memory, GPU-to-host memory and GPU-to-GPU memory. Noteworthy results of the benchmarks are that a significant increase in latency and drop in bandwidth can be observed if the transmit buffer on the sending side is located in GPU memory. A latency comparison shows that a reference measurement of GPU-to-GPU transfer over an InfinBand card has a three times higher latency as the corresponding APEnet+ implementation. This is due to the fact that the APEnet+ implementation avoids two *memcopy* operations between GPU-to-main-memory and

main-memory-to-IB. The authors argue, that the poor read performance from GPU memory on the transmit side is the result of unoptimized read operations of the micro controller software. This problem was solved in a subsequent paper [3]. The rate at which read requests are sent to the GPU was increased and the latency at which the GPU responds to the requests was reduced.

The 3D torus direct switch-less network is a very interesting approach which can be built upon. This design reduces the number of required components and in turn saves energy. The experiments in this work illustrate that avoidance of *memcopy* operations is desirable. Another important finding from the work presented is that poor peer-to-peer performance is not a characteristic of the GPU, but depends on the CPU used. This can be avoided with PCIe switches put in front of the CPU Root Complex to optimize the flow between the P2P devices.

3.2.6 GPU Global Address Space (GGAS)

The work of Oden and Froening [47] addresses another problem of accelerators. As a slave device, accelerators can not source or sink network traffic by its own. All other work presented so far, from either industry or academia, depend on a certain degree on the CPU. This can include the data transfer between NIC and accelerator, create and manage buffers in main memory, managing the network protocol like writing a descriptor or command to the NIC or the CPU threads to coordinate the message passing between nodes. RDMA capable NICs like InfiniBand can hide most of this latency during the computation phase of a GPU kernel, but for small messages the context switch between GPU and CPU control flow can cause latency issues. To avoid the CPU involvement completely, the authors propose to keep the COTS aspect of accelerators and enhance the NIC's functionality instead. With assistance of the NIC, message passing can be avoided and each thread of a GPU can communicate with each other GPU by global addresses and therefore with load/store operations.

As shown in fig. 3.12 on the next page the user and system view differ from other concepts so far. Each node is still in charge of its own accelerator and is independent from each other, but a global address space now spans over all accelerators and build the so called *GPU Global Address Space (GGAS)*. Each GPU memory is mapped with a window into this global address space. Each memory location of a GPUs inside this window can now be found with a unique global address. From the

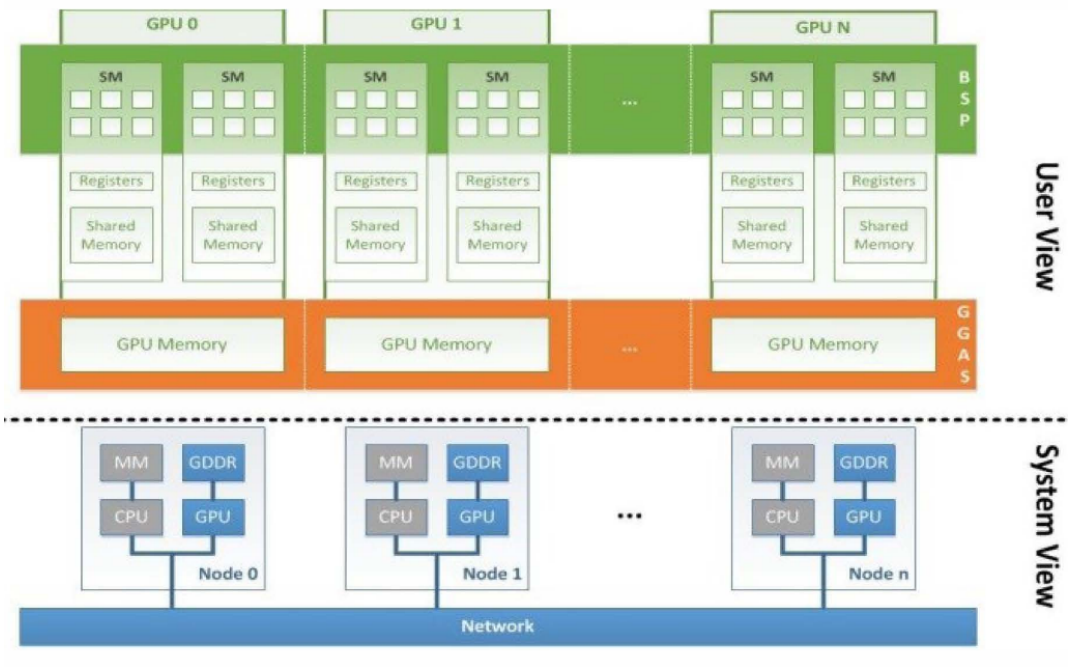

Figure 3.12: System and user view of a GGAS Cluster [47]

programmers perspective, to access a remote GPU memory location, a thread has to call a translation function and receives a pointer to the remote memory location in response. The target's NodeID is used as a parameter for the function and returns a pointer that can be used as an pointer to an array. The thread can now dereference the pointer and read or write operations are translated into network transactions without involvement of the CPU or additional software layers. The array points to a special address which is assigned to a memory region on the NIC. A Shared Memory Engine (SME) developed by the authors receives accesses to these global address pointers, encapsulates the loads or stores in network packets and delivers this request to the remote physical address space assigned to the pointer.

Figure 3.13 on the facing page shows an example of the interaction between the SME, physical host address space, GPU device memory and GPU virtual address space. The incoming network packet's target memory address is altered to target the GPU device space, which is memory mapped into the physical address space of the GPU's host (yellow array). For transfers from local GPU device memory to a remote location, the SME is mapped into the GPU virtual address spaces. Read or writes to this virtual addresses are received by the SME BAR, encapsulates the received packets and sends them over the network.

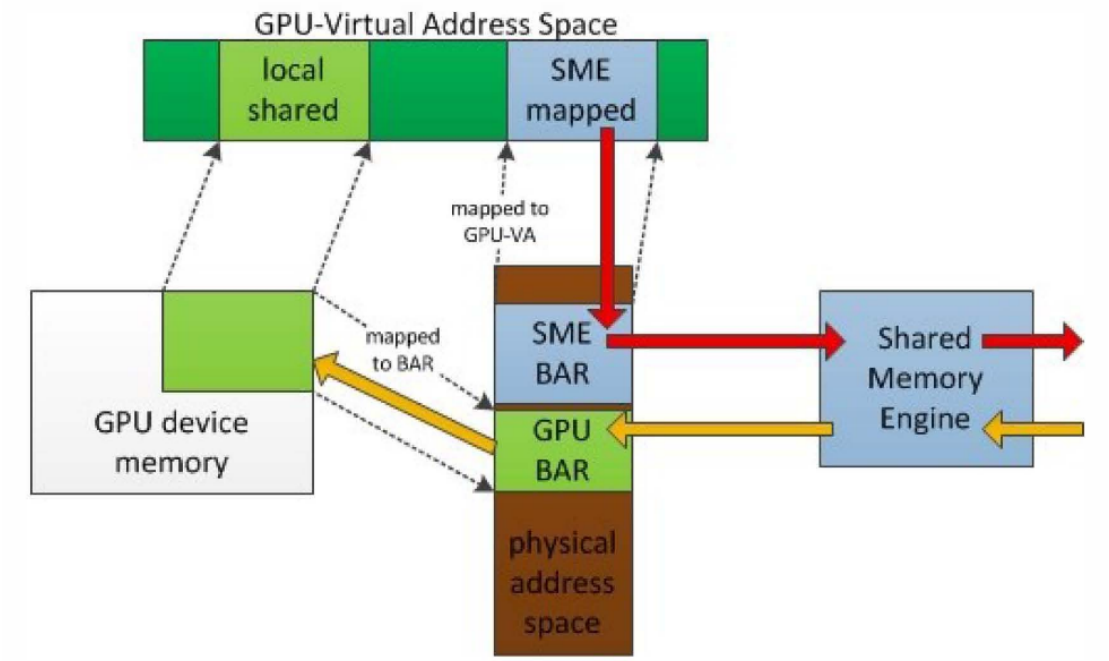

Figure 3.13: GGAS Mappings and Data Flows [47]

The presented work is the first approach that completely avoid CPU intervention in the data transfer and is the first time, that an accelerator can source and sink network traffic by its own. This aspect of the work by Oden and Froening [47] form a basis for the development of a NAA.

3.3 Cooling

The first part of this chapter discussed various approaches to intra and inter node communication. Data movement plays a crucial role in HPC systems, but the power consumption becomes more and more important. One way to improve the power efficiency is reducing the amount of energy wasted by cooling the infrastructure. Changing the coolant can significantly reduce the overall power consumption of a HPC system. Air is still the most used coolant, but locking at the Green500[28] list with the most power efficient computer systems, a trend towards liquid cooled systems can be observed. Comparing the Green500 results from June 2015 to June 2018, at least two of the top 3 super computers on this biannual list using liquid cooling and power-efficient accelerators to reach the best possible energy-efficiency. The following sections briefly describe some of these systems and describe their

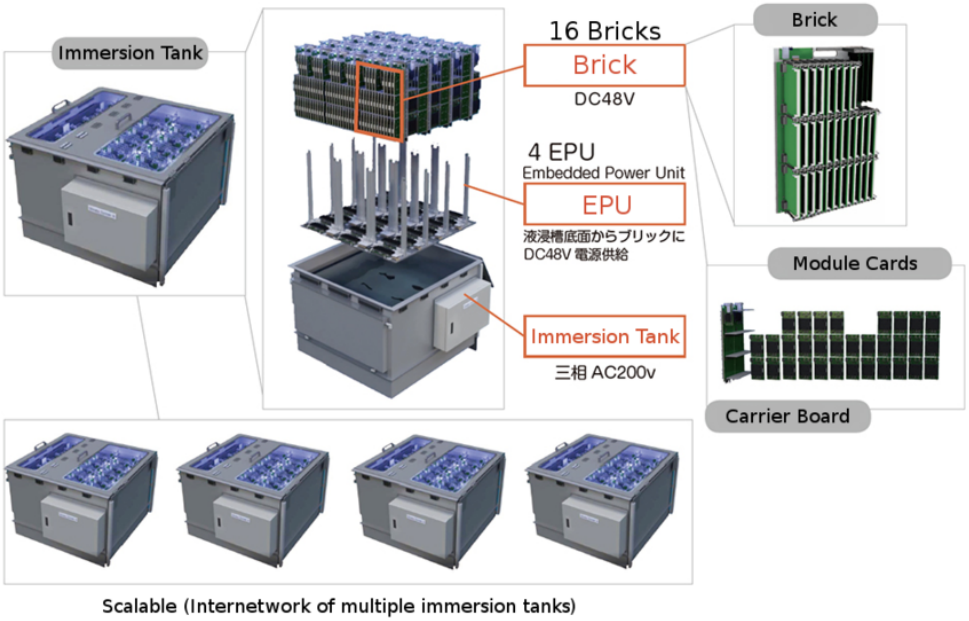

Figure 3.14: Shoubu System B overview [66]

benefits compared to systems cooled with air or water.

3.3.1 Shoubu System B

For several years now, the Shoubu System is the most energy-efficient supercomputers in the world. The system is deployed at the Advanced Center for Computing and Communication, RIKEN (Japan) and is now in its second expansion stage. The first system reached about 6.6 GFlops/W whereas the latest system reaches 17.6 GFlops/W. Both systems use customized PCB designs, custom low-power many-core accelerators and immersion cooling to increase the energy-efficiency. As shown in fig. 3.14 different level of modularity is used. Compute modules are attached to a carrier board and multiple of these boards build a brick. These bricks are connected with custom designed Power supply units (PSUs) and build into immersion tank. Multiple of these tanks are interconnected and form the final Shoubu System.

The immersion cooling uses a 1-phase approach and a forced circulation by pumps. The heated liquid is cooled down on heat exchangers which are settled underneath the tanks.[64] All tanks are sealed to reduce leakage, but the boiling point of the used liquid is high enough with nearly no vaporization.

Figure 3.15: TSUBAME-KFC submerged in oil tank [40]

This system is an example for high energy and space efficiency. A similar system *Gyoukou* [64] with comparable structure has a ratio of $1\,\mathrm{PFlop/m^3}$ which is unrivaled with current systems. As a comparison the Summit super computer has a $0.35\,\mathrm{PFlop/m^3}$ ratio between compute power and required volume. Summit has a LINPACK[19] performance of $143.5\,\mathrm{PFlop/s}$ as of the November 2018 TOP500 list. The system comprises of 256 racks [48] with each rack has a volume of $1.6\,\mathrm{m^3}$. This results in $\frac{143.5\,\mathrm{PFlop/s}}{1.6\,\mathrm{m^3}} = 0.35\,\mathrm{PFlop/m^3}$ and is only a third of the Shoubu System B.

3.3.2 TSUBAME-KFC

This system had its first appearance at the June 2013 Green500 list and was number 1 on this and the following list from November 2013. Even though a performance of $4.5\,\mathrm{GFlops/W}$ is only one quarter of the number one from the November 2018 list, the used coolant makes this system definitely worth a closer look. Compared to the other systems in this section, this system uses mineral oil as coolant. Mineral oil has a high resistance to avoid shorts like the other engineered fluids, but it has a higher viscosity than the engineered fluids. This makes natural convection unsuitable, due to the risk of local overheating, and strong pumps are required for a forced circulation. In contrast to the engineered fluids mineral oil is flammable, but has a very high specific heat. Some mineral oils have such a low flash-point, that the operator of such a system is forced to take special care for fire prevention. This leads to either increased operating costs, due to specially trained employees or

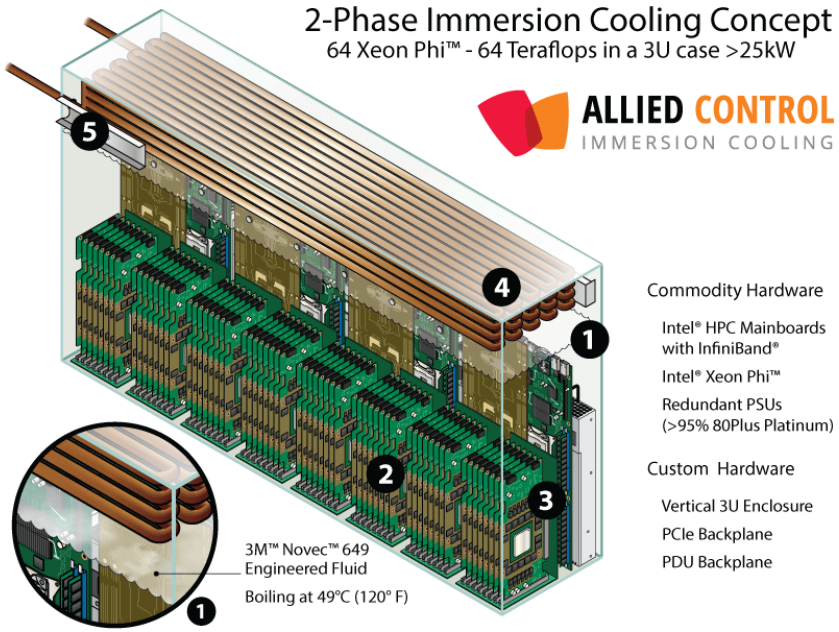

Figure 3.16: ALLIED CONTROL Cooling Concept[16]

constructional fire protection, or national regulations prohibit the commissioning of such a system. Also maintenance is more difficult. All the oil has to be removed before components can be exchanged.

3.3.3 DataTank™

Another example for a 2-phase immersion cooled system is the DataTank™ from ALLIED CONTROL[2]. As shown in fig. 3.16 an example configuration could be a 3U case with commodity and custom hardware. The commodity hardware comprises of standard Intel® mainboards with Intel® Xeon Phi™s. To save precious space, a custom PCIe backplane and PDU backplane are used. All components are immersed in Novec™ 649 in a sealed environment. No pumps are required to force a circulation of fluid like the TSUBAME-KFC system. All heat sinks and moving parts are removed to further optimize the space saving. Cooling coils on top of the containment condense vapor back into its liquid state. This module could be a building block for a larger installation like the DataTank™ with up to 1.4 MW with 6 240 kW flat racks.

Chapter 4

Network Attached Accelerators

The previous chapters laid the groundwork with basic concepts and the current state of the art. Pros and cons of different approaches in communication between accelerators and cooling solutions were briefly described. This chapter starts with a more detailed analysis of the current state of the art and derives a list of requirements the NAA has to fulfill. The rest of this chapter covers the different individual functions to implement and the last sections describe how this design was applied to the Intel® Xeon Phi™ accelerator.

4.1 State of the Art Analysis

All work presented in the previous chapter have their pros and cons. This section will pick up all ideas that are beneficial to fulfill the goals of this thesis as described in section 1.3 on page 6.

4.1.1 Accelerator Node Design

To create an accelerator node, a direct connection between the accelerator and the NIC is necessary. The most common interface to connect these devices with a CPU is PCI. As presented in section 3.1.2 on page 46 and section 3.1.3 on page 48 PCIe Endpoint devices can communicate with each other via a PCIe switch or the Root Complex of the CPU. The proposed accelerator node could consist of such a PCIe switch to allow communication between the NIC and the accelerator. Nevertheless, this still requires a CPU to configure the PCIe hierarchy and map the memory regions between the two devices. The accelerator node would not differ from an ordinary host system. To circumvent this problem, one of the devices has to incorporate the Root Port functionality of the CPU. As the accelerators for high performance computing, nor the Ethernet or InfiniBand NIC support this feature, the EXTOLL NIC has to be used for its build-in Root Port support. With this feature, a node can be build that has only an EXTOLL NIC in Root Port mode and any accelerator that uses PCIe as its host interface.

4.1.2 CPU-less Communication

As we saw in the previous chapter, accelerators and NICs are able to communicate directly with each other as long as they expose their device memory to the host physical address space. Accelerators can exchange their results with other accelerators by store operations to the other accelerator's memory addresses. If the accelerator does not reside on the same host, the NIC is responsible to transfer the data to the remote host. In the best case, the NIC is able to read the data directly from the accelerator's device memory (see section 3.1.1 on page 44) and write it to the remote accelerator's device memory. The NIC transfers the data without CPU assistance, but the whole process of setting up the communication is driven by the CPU. This involves reservation of pinned main memory regions for buffers, creating and writing the descriptors to the NIC and triggering the transfer. The only way

to avoid the CPU in the communication process is the work presented by Oden and Froening [47] with its GGAS approach. Fortunately, as no other NIC provides the Root Port feature, we already use the EXTOLL NIC and can take advantage of the GGAS. With their approach, all network communication is hidden within the NIC and no CPU is required. Application programmers can use memory addresses to reach other accelerators and are no longer concerned about explicit programming the communication.

4.1.3 Direct Network Access

Depending on the accelerator, it may be inefficient [46] or even impossible to start network communication by their own. GPUs for example have an execution model that can not handle all the branching and polling that is necessary to drive a communication efficiently. Most of the work presented so far solve this problem by returning the execution from the GPU context back to the CPU to handle the communication. To ensure, that also these types of accelerators can source and sink network traffic without a CPU present on the accelerator node, the GGAS concept from Oden and Froening [47] can be used. As already stated, the NIC encapsulates load and store operations from the accelerator into network packets and sends them over the network. In this way, the accelerator can initiate network communication by its own without explicit programming or handing over the execution to the CPU context. The Intel® Xeon Phi™is a very important exception, because the KNC is able to handle the complex network communication efficiently due to its core architecture. Furthermore, a standard operating system can be run on the KNC and existing driver infrastructures can still be used. For the above two reasons alone, it is no wonder that the work is mainly concerned with the KNC and tries to exploit this feature.

4.1.4 Cooling

The decreased number of components on the accelerator node reduces the wasted energy by redundant and unnecessary components. This also enables a more compact construction and more accelerators fit in the same space. However, this increases the effort to cool all the components. Air cooling is not sufficient and the large heat sinks waste space and depending on the used material are cost expensive. The *Shoubu System B* and the DataTank™show an unparalleled dense packing and cool-

ing efficiency that is impossible to achieve with air cooling. However, the mineral oil cooled system *Shoubu System B* has a few disadvantages. Maintenance is difficult due to oil residues and the medium must be pumped. The DataTank^{TM}does not have the problems of maintenance, but still requires a circulation pump. In order to not only reuse the existing concepts, but also to extend them, an attempt is made to use 2-phase cooling.

4.1.5 GPU Cluster Interconnect

The accelerators are very sensitive to workload imbalances and the communication latency and bandwidth are a not negligible part of it. To avoid this imbalance, the presented topologies in section 3.1.3 on page 48 or section 3.1.2 on page 46 have a very restricted amount of communication paths. Most of the time, the communication is between adjacent accelerators to avoid over utilizing the available bandwidth, or the transfers are limited to a single direction. In addition, the topology is a tree and the diameter increases significantly with the depth of the tree. To circumvent this, the EXTOLL interconnect offers a sufficient radix to build a 3D Torus, which is known to have a small diameter in the most instances. The network bandwidth exceeds the given host interface bandwidth and allows multiple transfers in the same direction without over utilizing the network link. Compared to other interconnection technologies like InfiniBand or Ethernet, the topology is not as flexible as EXTOLL. Most of the time the topology is based on a fat tree which result in large additional latencies to traverse several intermediate switches. This and all the other features described so far make the EXTOLL interconnect the only choice left to fulfill the requirements to minimize latency and balance bandwidth.

4.1.6 Summery of Requirements

From the problem statement and the analysis of the state of the art above, one can derive a list of requirements and challenges for the design of a NAA:

- PCI Root Port functionality on the remote side
- Enumerate PCIe devices without CPU present on the remote side
- Access to the PCIe ConfigSpace of the accelerator
- Access between different memory domains

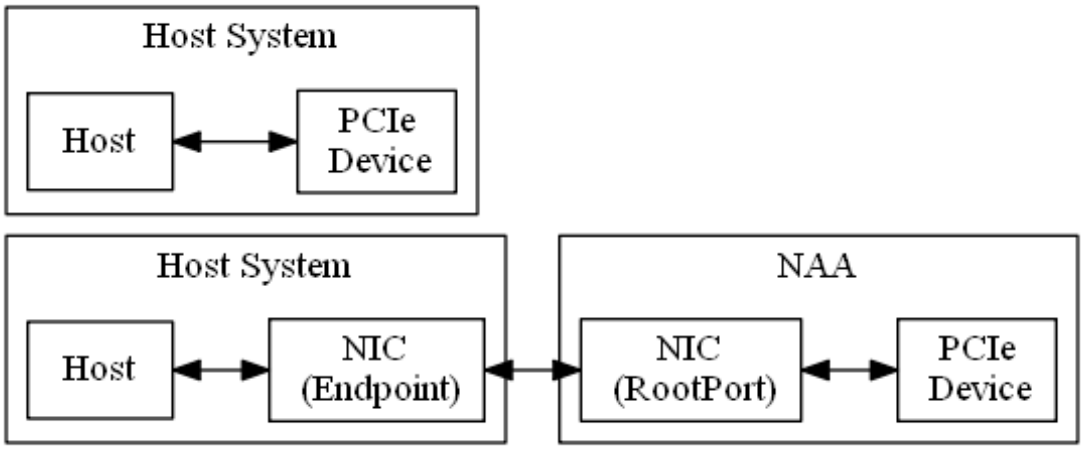

Figure 4.1: PCIe enumeration

- Software transparent mapping of memory
- Direct access to network functions for the accelerator
- Interrupt delivery to the host from the remote side
- flexible configuration of task mapping
- Physical connection between NIC and accelerator
- Power and side-band signals supply for NIC and accelerator

4.2 Remote PCIe bus access

As specified in the design requirements of the NAA, no CPU or BIOS resides on the NAA to handle the host interface initialization. This CPU less remote side has the benefit of a simple structure and reduced number of power consuming components. Figure 4.1 (a) shows a simplified PCIe hierarchy with a directly connected device to the Root Port of a Root Complex. In this situation, the host platform BIOS or CPU enumerates the PCIe hierarchy during power up.

Enumeration of a PCIe device in the NAA use case involves additional tasks, because the NAA has its own independent PCIe hierarchy with its own Root Port. Furthermore, as shown in fig. 4.1 (b), the host platform has no direct access to the remote PCIe devices and during power up of the host system the network connection

between the host system and the NAA is not established. The first step is to discover the EXTOLL network as described in section 2.4.4 on page 33 with the EMP tool to assign the host NIC and the remote NIC on the NAA a EXTOLL Network ID. Even with an established network connection there is no direct way to send PCIe config space packets as described in fig. 2.11 on page 26 from the host system to the remote side. A workaround for this situation is the PCIe Backdoor, which sends arbitrary PCIe packets.

4.2.1 Remote PCIe packet generation

One of the features of the EXTOLL NIC is the ability to be configured from any node of the EXTOLL Network. Every EXTOLL node has read and write access to the NIC's register file, which holds all configuration settings of the NIC as well as status information. As described in section section 2.4.1 on page 29, the PCIe backdoor was initially designed to emulate PCIe packets in software which have no valid counterpart in the internal EXTOLL protocol. PCIe Configuration Read Request (CfgRd) and Configuration Write Request (CfgWr) packets are not supported by the EXTOLL base protocol and have to be emulated by software. The Backdoor has a buffer to store PCIe packets to send arbitrary PCIe packets towards the outgoing PCIe stream. For the purpose of remote PCIe devices enumeration, PCIe config space packets as described in 2.11 are written by the configuration software to this TX buffer. Figure 4.2 shows this process in detail.

The configuration software on the host system wants to store a configuration packet to the remote *Register File* which contains the Backdoor TX buffer. Software generates multiple RMA immediate put descriptor with 64 bit payload and sends it towards the host's EXTOLL NIC. The descriptors are received by the RMA unit, which translates them into RRA write operations with 64bit payload as data. The RRA Requests are sent over the EXTOLL Network to the NAA target node and the receiving RMA Unit writes the configuration packet into the TX buffer of the backdoor. After the complete configspace packet is sent over the network to the target NAA node, a write to the command register at the backdoor starts the transfer.

As all config space packets are non-posted packets and each read or write request is followed by a completion packet. To make the read result or the return value of the write request accessible to configuration software, the backdoor RX module is

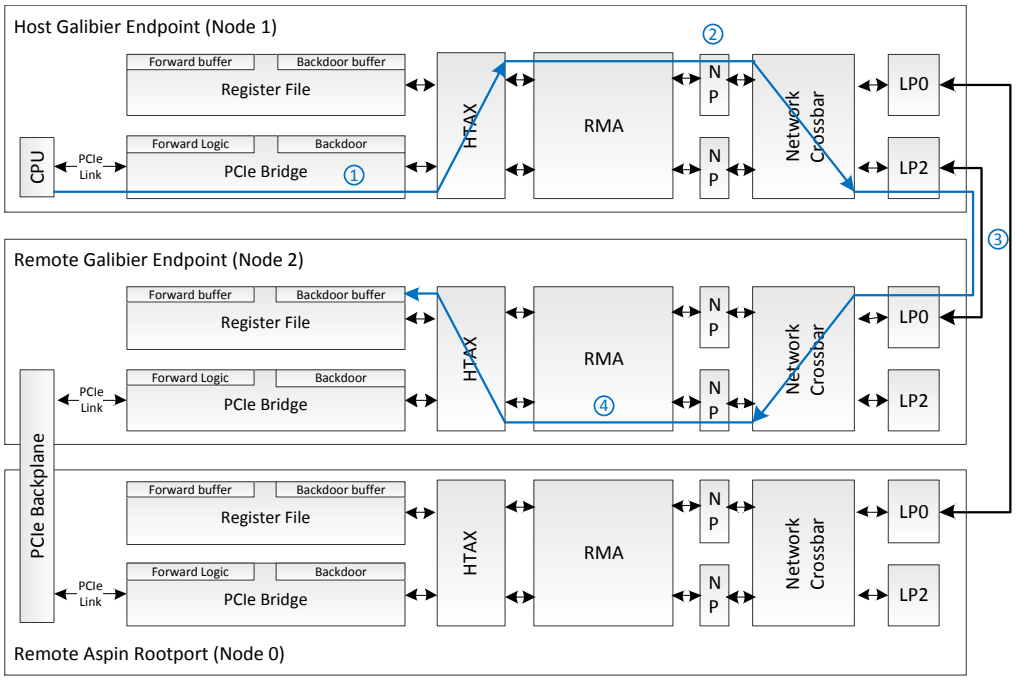

Figure 4.2: Remote RegisterFile write access

configured to extract all completion packets with source tag 0 from the incoming PCIe stream. These extracted packets can be read by software from the host and is shown in figure 4.3.

The forwarding buffer of the backdoor has a status register which indicates if a packet is available in the RX buffer. Furthermore this register indicates how many 64bit read operations are necessary to read the complete extracted packet. First, the configuration software on the host sends a RMA get descriptor to the host's EXTOLL NIC. The received descriptor is translated by the RMA unit into a RRA read operation requesting 64 bit. The RRA Request is send over the EXTOLL network to the NAA target node and the receiving RMA unit reads the value from the Forwarding Buffer status register. The read value is send back to the RMA unit of the host and written to main memory. The configuration software starts now as many read requests as are received in the return value from the RX buffer following the same schema as described before.

With this procedure arbitrary configuration read and write packets can be send over to the remote PCIe bus and also the return value is accessible to configuration software. This allows us to *tunnel* PCIe packets through the network to access the independent PCIe bus of the NAA.

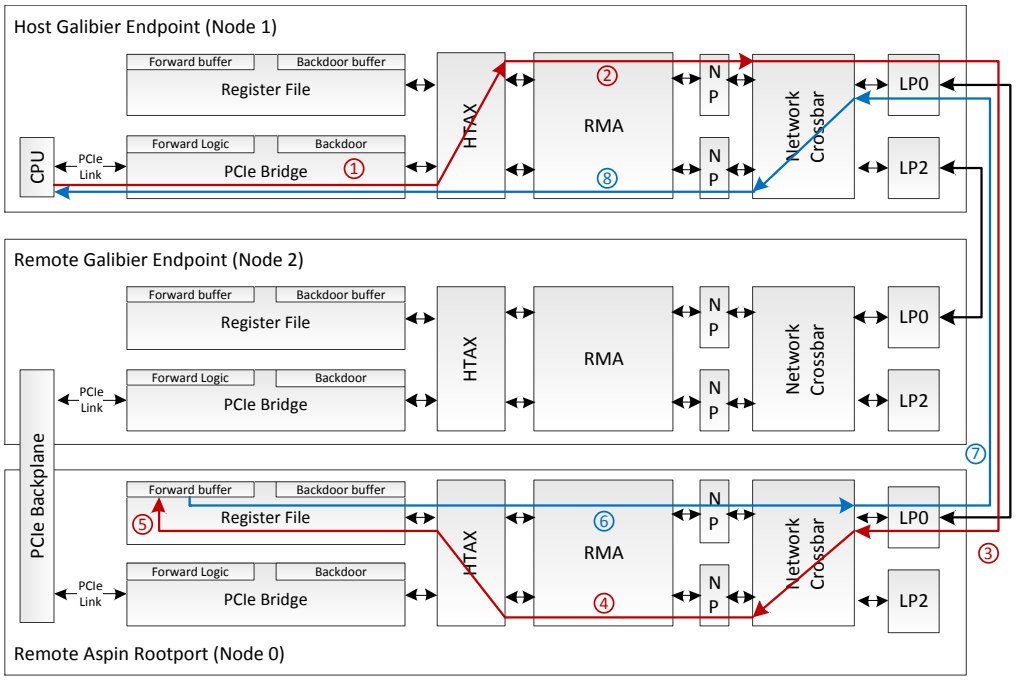

Figure 4.3: Remote RegisterFile read access

4.2.2 Remote PCIe enumeration process

As shown in fig. 4.1 on page 69 the remote PCIe bus consists of a Root Port (EXTOLL NIC) and a PCIe Endpoint device (i.e. Intel® Xeon Phi™). But before the PCIe configspace packets can be sent to the accelerator the Root Port must first be configured, which involves several steps.

First of all the Root Port' bus ID has to be assigned. This is done by writing a CfgWr packet of type 0 to the Root Port, which captures the Completer ID of the request as its new bus id. As the name suggests the Root Port is the root of the PCIe hierarchy and has the bus id of 0. The next step is to configure the routing-by-id capability of the Root Port. Since the accelerator is directly connected to the Root Port, only one bus exists on the downstream port. The Root Port is on bus 0 as its primary bus, which is also the upstream port. Because of the depth-first-search discovery of the PCIe hierarchy, each switch assigns a secondary bus number that is by one larger than its primary bus number.

The subordinate bus number is the highest bus number that is present downstream of the switch. The subordinate and secondary bus are the same, as only one device resides on the bus of the NAA's Root Port, the value of 1 is assigned to both of them. After this step, the routing-by-id is configured and all packets destined for

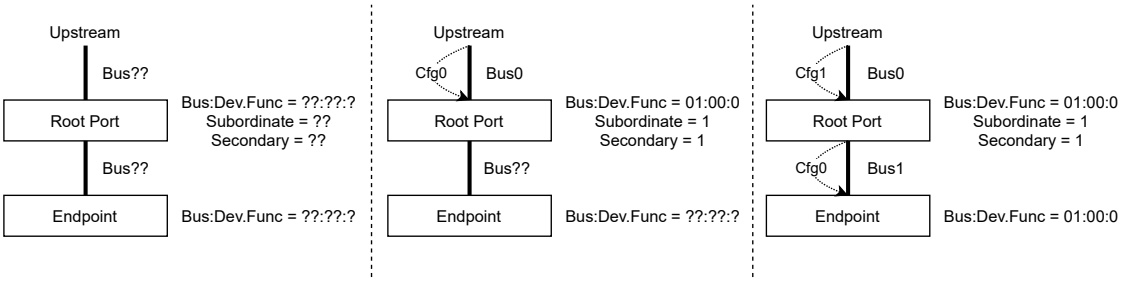

Figure 4.4: Different steps during PCIe enumeration

the busses 0 or 1 (CfgRd/CfgWr requests, Completions) can now be delivered, while all other bus IDs are rejected.

As the accelerator is a memory mapped device it is important to configure also the memory regions that are behind the Root Port. These regions are divided into two types, prefetchable and non-prefetchable memory. The prefetchable memory defines the region that does not change during reads and can be cached or prefetched by the host. The prefetchable memory region is configured in the Root Port, described by a base and a limit register. The non-prefetchable memory region is for changing content like status registers that can change between two consecutive reads and are also defined with a base and a limit register. The non-prefetchable and prefetchable registers are assigned with values based on the requirements of the accelerator. For the used Intel® Xeon Phi™a window of 1 MB non-prefetchable memory must be configured. The register set for control and status information of the Intel® Xeon Phi™is 128 KB, but the smallest window that the PCIe register *non-prefetchable memory* can display is 1 MB. According to the size of the KNC on board memory, which can vary between 8 and 16 GB, a large window must be entered in the *prefetchable memory* register.

To enable the memory forwarding engine of the Root Port the control register is written and busmaster, memory routing and I/O routing are enabled. All memory packets that target addresses inside the memory windows of prefetchable or non prefetchable memory are forwarded to the downstream device while packets with addresses outside of this region are rejected. With these steps, the NAA's Root Port is ready and the accelerator can be enumerated.

Like for the Root Port, the first step is to assign a bus id to the accelerator. This time a CfgWr packet of type 1 is send from the configuration software at the host to the EXTOLL NIC on the NAA side. Instead of addressing the Root Port's configuration space, a type 1 CfgWr packet is forwarded by the Root Port towards

the destination bus id and the CfgWr type changes to 0, if the targeted Bus ID is within the Secondary and Subordinate ID range. The CfgWr is then received by the accelerator and like the Root Port the completer id is captured until a new completer id is received.

4.2.3 Memory size determination

To determine the memory size a device requests for its operation, a set of registers at the PCIe configuration space is used. These BARs of the accelerator device consists of up to two 32bit registers as described in section 2.3.3 on page 23. The device indicates the memory size by the number of hard coded zeros of the BAR. The first task of the configuration software is to count the number of zeros that remain in the BAR after a write of all ones. This encodes the size of the memory window with $memory\ size = 2^{(number\ of\ 0's)}$. The configuration software uses this value and assigns a memory address for the device by writing the non-hard coded registers.

The next task is to map addresses from the host physical address space to the address space of the NAA.

4.3 Transparent Memory Mapping

Up to this point we have built a system consisting of a host node with a specialized NIC and a fully configured remote PCIe hierarchy. In this all devices (root port and KNC as Endpoint) have received a unique bus ID, the physical address routing is configured and the KNC on-board memory has been assigned physical addresses by the BAR registers. Now these two physical address spaces, the host and the accelerator address space, must be connected with each other.

One of the main goals of the NAA is a transparent memory mapping between the host and the NAA. Transparent in this context means, that the host system accelerator driver handles all device interactions as if the device were physically present on the host machine. Communication with PCIe devices is done by mapping one or more memory regions from the device into the host's physical address space as described in section 2.2 on page 18. In the case of accelerators these regions are on-board device memory, control and status register. All memory operations on these addresses are translated into I/O operation on the PCIe host interface. The size of the devices memory mapped I/O is defined by the BAR registers as described

in section 4.2.3 on the facing page.

One way to solve this problem to connect these different address spaces is an additional software layer that intercepts all I/O operations to the device's memory like shown in section 3.2.2 on page 52. Each operation calls a software handler that sends the memory transaction over the network, translates between the address space of the host system and the NAA and manages the transactions on the device's memory. This involves additional overhead and requires modifications of the existing device drivers.

A more elegant and hardware supported approach is to use the Shared Memory Functional Unit (SMFU) of the EXTOLL NIC.

The enumeration already assigned NAA physical address to the BAR registers of the KNC. As described in section 2.4.2 the SMFU can handle distributed shared memory. The KNC's device memory is now shared with the host system. The SMFU defines the mapping of a shared memory region with intervals. Each interval has a lower and upper bound, which defines the size and the location of the shared memory mapping. A control register enables the mapping and assigns a target node to the interval, which is the EXTOLL NodeID of the NAA's NIC. Each PCIe MemRd or MemWr packet that is sent to an address inside the host node's interval is translated into a network packet and send over to the remote NAA. On the receiving side, the network packet is unwrapped and the MemRd or MemWr packet is send towards the accelerator. As host address space and NAA address space are independent of each other the address used on the host side can result in an invalid and not assigned address on the NAA address space.

To avoid this, the address information from the host packet is removed and replaced with the offset within the interval. This intermediate step enables the address conversion within the SMFU to be carried out independently of the actual placement of the SMFU MMIO in the host systems address space. It is only relevant at which offset within the SMFU a read or write operation was initiated, but not its exact physical address which significantly simplifies the implementation in hardware. Instead of sending a full physical address, only the offset and additionally the interval ID is transmitted from the host system to the remote SMFU.

On the remote side for each interval ID exists an register with a base address that is added to the host packet offset. Under normal circumstances this register would contain the address at which the shared memory window in the remote hosts main memory starts. In the NAA case this base address registers are configured to be the

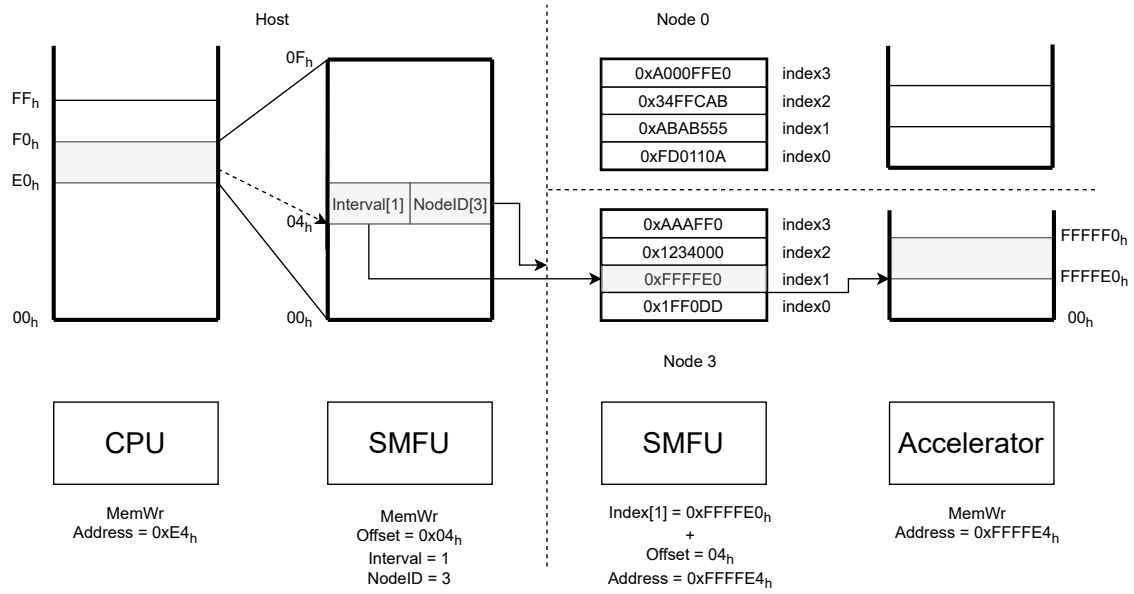

Figure 4.5: Memory Mapping for Host to Accelerator traffic

same address as the entries in the BAR registers of the KNC. This way every read or write access that occurs within the host SMFU intervals is automatically mapped to a physical address on the NAA side and points directly to the KNC's on-board memory without any further action. This combination of host interval offset and NAA base register generates a mapping between the host physical addresses and the NAA physical addresses which is transparent for the host's device driver.

For the opposite direction, from accelerator to the host, the mapping works similar. But instead of defining an interval for specific regions of the host system's physical address space, all memory packets coming from the KNC and target the host memory are not translated and passed unaltered. This is done with a special configuration of the SMFU that lets the NAA addresses unchanged during the transfer from the NAA's NIC to the host. This generally allows any address in the host system to be addressed by the NAA system. However, certain areas are used for special purposes to give the KNC access to the remote NIC functional units, but more about this in section 4.5 on page 78.

4.4 Interrupt remapping

Another capability of PCIe devices that is not directly accessible to the host system's CPU are interrupts. PCIe signals interrupts not with asserting or de-asserting an

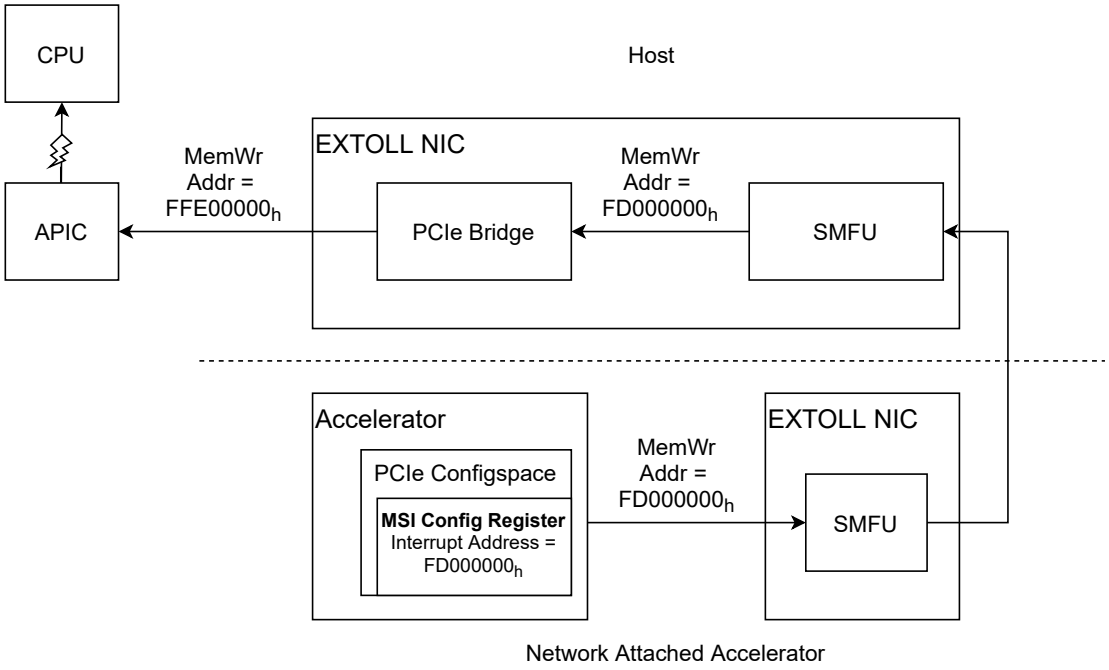

Figure 4.6: Interrupt mapping

interrupt line out-of-band like its predecessors PCI and PCI-X. Instead interrupts are delivered as in-band PCIe packets. A posted memory write packet send to a special address range (FEExxxxx)h is interpreted as a MSI. On the host system, an APIC is present behind this memory range, but not on the NAA side.

The workaround here is to reuse the already registered interrupt of the host's EXTOLL NIC. As all MSIs are memory write transactions, the same memory mapping approach from remote to host side can be used. EXTOLLs internal protocol is based on the HyperTransport host interface and therefore the address range that is recognized as an interrupt inside the EXTOLL core is the FDXXXXXXh address range. The configuration software writes this address into the MSI Config register inside the accelerators configspace. When the KNC triggers an interrupt, this address is used for the MemWr packet towards the host. The SMFU on the NAA side transfers the packet without modification to the host's EXTOLL NIC. The interrupt packet from the accelerator is decoded in the PCIe Bridge IP Core of the host's NIC as an interrupt and a MSI is sent to the host's local APIC with the address taken from the EXTOLL NICs MSI config register. The interrupt handler registered for the NIC is then called and takes care of informing the accelerators interrupt handler that an interrupt was requested.

4.5 Network feature access

Due to the operating system running on the KNC, it is possible to load the NIC device drivers. Unfortunately, as the KNC is a PCIe Endpoint, the operating system has no PCIe subsystem compiled into the kernel. This subsystem is added by a custom kernel module , which implements a PCIe subsystem and provides all necessary kernel functions that are needed to issue PCIe communication from the KNC towards the NIC.

PCIe packets sent from the KNC and arriving at the Root Port of the NIC are forwarded to the HTAX NOC. The HTAX serves as an interval router and distributes incoming packets from the host to one of the functional units like the SMFU. The HTAX on the NAA is configured in a way that it forwards all memory transactions either to the host system via the network or to the functional units of the remote NIC.

In order to be able to address a large area of host memory without shadowing to much of it by the functional units of the NAA, the windows of the functional units are mapped into a very high area of the physical memory of the NAA.

This relatively simple trick makes it possible that an accelerator can use use highly specialized units for network communications without external CPU intervention.

to map to the functional units of the RMA and VELO.

Chapter 5

Prototype implementations

Different prototype implementations served as proof of concept of the NAA design. Most of them were funded by the European Research project DEEP, which was part of the seventh European framework program. DEEP stands for "Dynamical Exascale Entry Platform" and the research focus was the development of new architectures to reach exascale. This chapter describes the different prototypes from the early evaluation board implementation over to the DEEP production system. First, the implementation of the NAA concept on an Altera StratixV FPGA is shown in detail and the additional components required are explained. Then different scalings are presented, a 2 node system, an 8 node system and finally the two production systems from DEEP, a 384 node water cooled and a 32 node liquid cooled system.

5.1 Evaluation Platform

After establishing the theoretical background, the prototype development began in 2013 with several FPGA implementations. During this time, the EXTOLL NIC was available as PCIe add-in cards, called Galibier, and is based on a custom PCB with Xilinx FPGAs. On the NAA node side, an Altera StratixV evaluation board serves as the target for the EXTOLL NIC in Root Port configuration. The accelerator was an Intel® Xeon Phi™ engineering sample with a reduced number of cores compared to the later available production devices. During the prototype development, several challenges arose. One was the porting of the EXTOLL firmware from the well known Xilinx FPGAs to the new Altera FPGA technology. It was not clear, if the existing Xilinx implementation can be mapped to components found in Altera FPGAs or a major redesign of the EXTOLL firmware was necessary. Another challenge was the new Root Port functionality, which wasn't used in a real world application so far. All prior experience with the Root Port and Backdoor submodule was based on functional simulations. The last challenge was the direct connection of the EXTOLL Root Port NIC with the accelerator and provide all the electrical and mechanical connections.

The following sections cover how these challenges were solved and at the end of this section the whole evaluation platform is described.

5.1.1 Backplane

The first challenge was to provide a direct physical connection between the EXTOLL Root Port NIC and the Intel® Xeon Phi™ accelerator. Apart from the PCIe connectivity, also power, reset and clocking has to be provided. To solve these problems and to reduce the number of active and powered components to a minimum a custom hardware design was required. Commercially available products were to expensive and didn't fit the needs. As a solution, the *PCIe Backplane* was designed as part of a student work. The initial design from 2012 is shown in fig. 5.1 on the next page. This PCIe Backplane has two PCIe x16 slots to support PCIe add-in cards. The PCIe lanes are crossed and all TX lanes of one slot are connected to the corresponding RX lanes of the other slot and vice versa. This builds a direct connection without any additional components like switches or bridges.

On the top of the backplane is a ATX 20+4 pin connector to power the board from

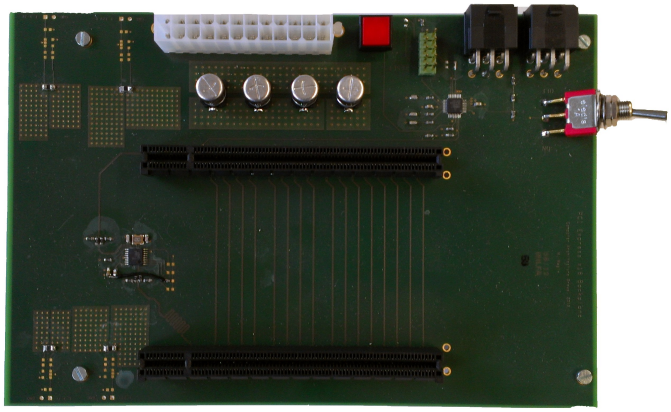

Figure 5.1: PCIe Backplane

a common PC power supply. This power supply feed the 12V and 3.3V rails of the PCIe connectors, which are sufficient to power the EXTOLL Root Port NIC. The KNC requires additional power supply with PCIe power cables as the accelerator draws up to 300 W at full load. A switch on the right side enables the power supply by pulling the sense input to a low state. This powers up the 12V and 3.3V supply rails.

In this first version of the backplane the PCIe reset signal is triggered by the red push button next to the ATX connector. If the button is pushed, the active low reset signal is pulled down to ground and resets the EXTOLL Root Port NIC PCIe host interface and resets the accelerator.

PCIe devices operate with a common 100 MHz reference clock connected to both devices. On a host system, clock generator chips on the mainboard provide this clock source. This reference clock is connected to the CPU or chipset and to the PCIe connector on the mainboard. The PCIe device doesn't require its own clock source, instead, it takes the clock from the PCIe connector.

The EXTOLL Root Port NIC and the KNC require their own reference clock source as no clock is transmit over the network. Therefore the backplane has its own clock generator chip and provides the 100 MHz to both devices after the 3.3V power rails is stable.

As this board was the first version of a PCIe backplane, the power up and reset procedure is not automated and is done manually. To be conform with the PCIe specification, these requirements have to be met:

1. The 12V and 3.3V rails are powered down.

Figure 5.2: StratixV Evaluation Board

2. The active low reset signal is pulled down to ground level.

3. The 12V and 3.3V rails are powered on.

4. The 100 MHz reference clock starts running.

5. The reset remains low for at least 100ms after 12V, 3.3V and the reference clock are stable.

As a consequence of this requirements, to proper reset the devices, the reset button has to remain pressed down and the toggle switch is put into the *ON* position to drive the 12V and 3.3V rails. Then the reset button is released shortly after that and both devices start their operation.

This first generation of the backplane provided sufficient housing for the accelerator and the StratixV evaluation board and solves the first challenge of providing the necessary electrical and mechanical connections. The next step was to port the existing Xilinx EXTOLL design to the new Altera FPGA.

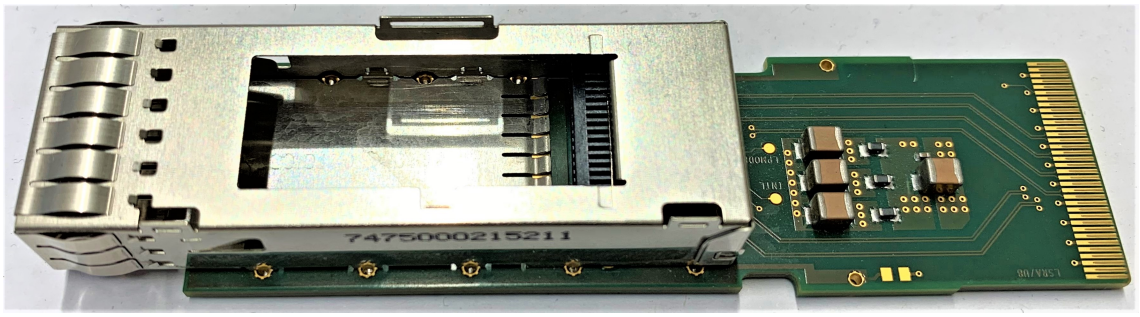

Figure 5.3: QSFP-to-HDI6 Adapter

5.1.2 StratixV Eval Board

The first prove of concepts took place on an Altera StratixV evaluation board, because the funding research project DEEP decided to use Altera FPGAs for their better pricing. The evaluation board is a PCIe half-length full-height add-in card (see fig. 5.2 on the facing page) and provides a widely used Quad Small Form Factor Pluggable (QSFP) connector for Ethernet or other common interconnection protocols. This connector is used to interface with other EXTOLL NICs. As all EXTOLL NICs use Samtec HDI6 connectors an adapter board is required to bridge between the QSFP and HDI6 connector standard. This adapter is shown in fig. 5.3. Due to the limitations of the QSFP connector, the EXTOLL link between the EXTOLL NIC in the host and the EXTOLL Root Port NIC on the remote side only supports 4 lanes. Each lane has a transfer rate of $4\,\mathrm{Gbit\,s^{-1}}$ to be compatible with the Galibier card. The Altera transceivers and the QSFP support higher transfer rates, but the achievable internal logic frequency of the Xilinx and the Altera FPGA are the limiting factor.

All functional units of the EXTOLL core have a datapath width of 128 bit. The highest internal frequency that met timing on both FPGAs is 100 MHz. The matching transfer rate can be calculated as follows:

$$\frac{\mathit{parallel\ datawidth} \times \mathit{internal\ frequency}}{\#\ \mathit{lanes}} = \mathit{serial\ datarate\ per\ lane} \tag{5.1}$$

$$\frac{128\,\mathrm{bit} \times 100\,\mathrm{MHz}}{4} = 3200\,\mathrm{Mbit\,s^{-1}} = 3.2\,\mathrm{Gbit\,s^{-1}} \tag{5.2}$$

The $3.2\,\mathrm{Gbit\,s^{-1}}$ is the data rate without the 8b10b line encoding of the EXTOLL link. After adding the encoding overhead to the serial data rate the raw data rate of the EXTOLL link is $4\,\mathrm{Gbit\,s^{-1}}$.

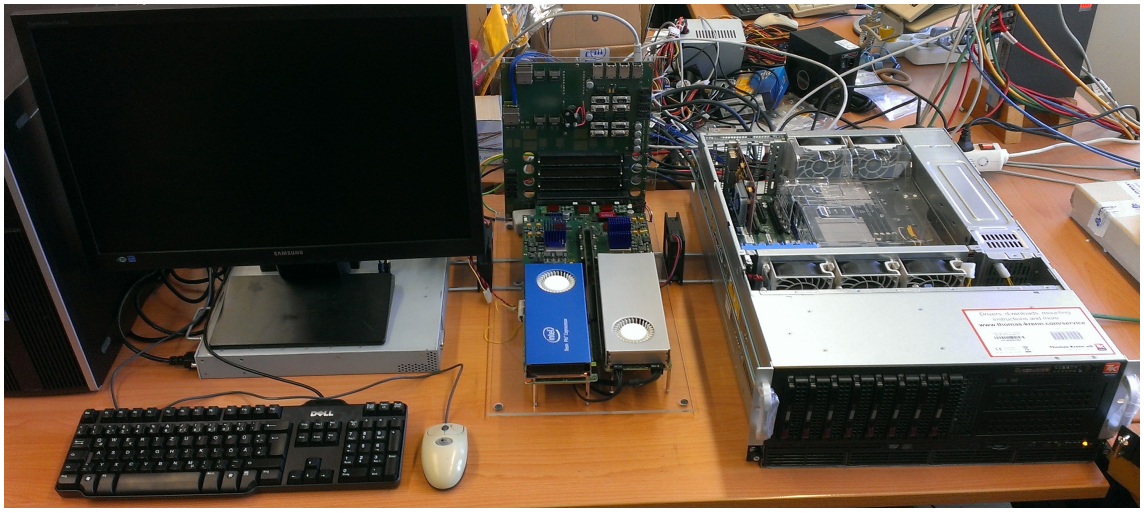

Figure 5.4: Test System with BNC and BIC

The EXTOLL RTL design was synthesized and implemented using Quartus 12.1. The design used for the StratixV evaluation board is a modified version of the module overview shown in fig. 2.13 on page 27. Instead of seven EXTOLL links, only on link is present, which reduces the number of Link Ports and Phy modules to one. Also the Network Crossbar has only five ports instead of 11 in the Tourmalet ASIC design. Other minor design modifications were necessary to fit the design in the relatively small FPGA and to ease the timing closure.

With the EXTOLL Root Port NIC, the Intel® Xeon Phi™ and the backplane, all components are now available to build up a test system to develop the necessary configuration software. The required steps to bring up the test system are the configuration of the EXTOLL Root Port NIC, configure the KNC and set up the memory mapping to boot the KNC.

5.1.3 Test System

To be as close as possible to the final system, the prototype has to consist of three major components. The Cluster Node (CN) are server with high single-thread performance for tasks that can't be multi-threaded. Application parts that can be parallelized run on the Booster Node (BN). The CN have their own independent interconnection network InfiniBand. All BN use the EXTOLL network. To bridge between the different interconnection protocols the Booster Interface (BI) has an InfiniBand NIC and an EXTOLL NIC installed. Software running on the BI receives

requests from the one network and maps it to transactions on the other interconnection network. These nodes serve also as configuration nodes for the NAA.

As the communication between the different interconnects was developed by other project partners, the BI only has an EXTOLL NIC attached to it. The EXTOLL NIC is a Galibier card with a Virtex6 FPGA and connects via a QSFP adapter board to the StratixV evalboard described in section 5.1.2 on page 83. The Galibier card provides a PCIe Gen2 x8 host interface and 4 EXTOLL network links. Due to the limited amount of transceivers available on the FPGA each EXTOLL link consists of 4 lanes instead of 12 in the ASIC implementation. On the remote side, the evaluation board was attached to a backplane as described in section 5.1.1 on page 80 together with an early engineering sample version of the Intel Xeon Phi.

This test system has now all required components to operate very close to the final system and provides a adequate software development vehicle for the configuration software. This test vehicle was used to:

- Validate the remote PCIe configuration from BIC
- Mapping of resources to ALTERA's logic blocks
- Check the memory mapping between BIC's MMIO space and KNCs on-board device memory
- Develop kernel and driver modules to use the remote memory mapping
- Boot the KNC's micro OS from remote CN

5.2 Booster Node Card

The next iteration of the BN was the integration of the Altera FPGA into a custom PCB design. In order to keep the number of connections via backplane PCB or cable as low as possible, there are two network nodes on one PCB. This Booster Node Card (BNC) board will be the basic building element for the large scale implementation and was designed by project partner, Eurotech. The original project planing expected the use of EXTOLL ASICs. Due to unexpected delays during the development of the ASIC design, the project decided to use large Altera FPGAs as replacement for the ASIC. The challenge of this decision was now to fit a full-blown EXTOLL ASIC design into a rather small FPGA. Each network interface contains

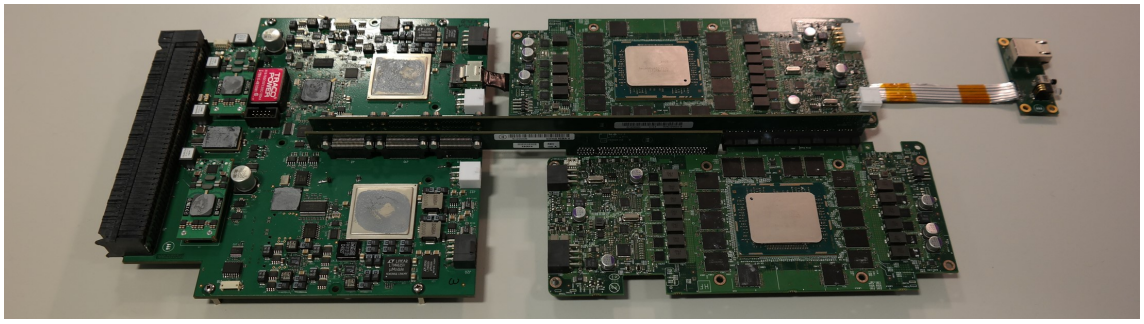

Figure 5.5: DEEP Booster Node Card

12 lanes and for seven interfaces 84 transceivers are needed. In addition there are 16 PCIe lanes to connect the EXTOLL NIC to the host interface. If all transceivers are added together, a complete ASIC replacement requires 100 transceivers. Even the largest FPGAs of that time have neither the required number of transceivers nor the corresponding hard IP blocks to support such a configuration. Therefore, the following restrictions had to be made.

- All EXTOLL Links have 4 instead of 12 lanes.
- The PCIe connection has 8 instead of 16 lanes.
- One Link between the FPGAs on the same BNC is implemented as LVDS instead of serializers.
- The EXTOLL Link bandwidth is reduced to $4\,\mathrm{Gbit\,s^{-1}}$

The basic building block of the large scale implementation consists of a backplane with eight slots to attach eight BNCs. To reduce the cabling effort most of the connections between the nodes is done in the PCB on different layers.

Figure 5.6 on the next page shows the design of the backplane during the evaluation phase. The final design has the bottom 4 slots replicated to an 8 slot backplane. It provides 48V power supply which is converted to 12V on each BNC. Through holes in the backplane are for the liquid cooling quick connects. Each BNC is cooled with a cold plate.

5.3 DEEP Booster

In order to build not only an energy-saving cluster but also a very compact one, the cluster was divided into different modules.

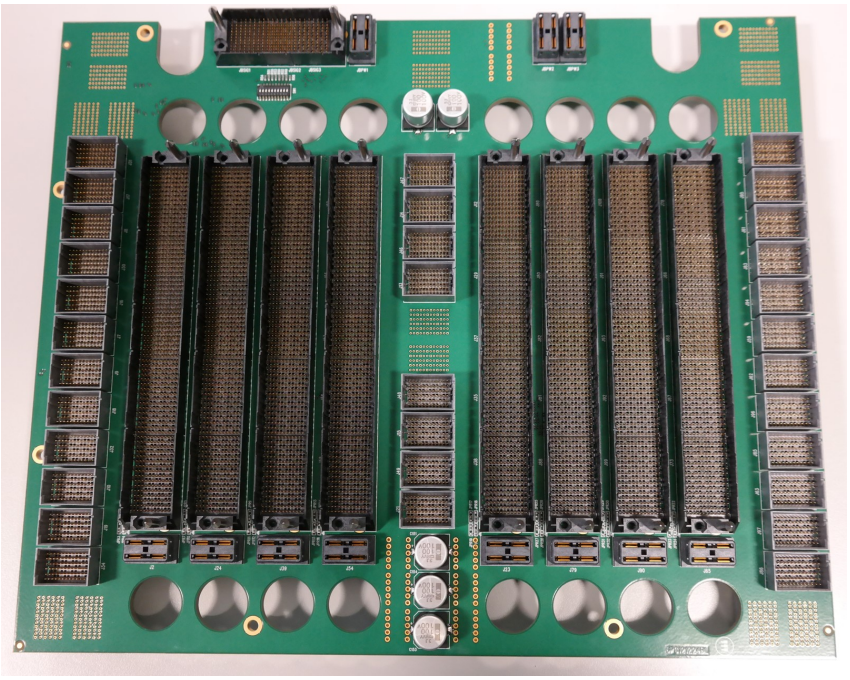

Figure 5.6: Eurotech Backplane

Figure 5.7: DEEP Booster [18]

The smallest of these units represents the BNC and consists of two network nodes on one PCB. In this case, a network node is an EXTOLL NIC and a KNC. The next bigger building block is a chassis and consists of 8 BNC boards and a backplane connecting the boards. The boards are connected to each other by PCB connections as well as by cables. Several chassis in turn form the actual cluster. To allow compute nodes outside this special cluster to interact with the NAA nodes, each chassis contains a BIC which serves as a bridge between the external network (mostly Ethernet or InfiniBand) and the cluster internal network (in this case EXTOLL).

5.4 GreenICE

With the DEEP Booster Eurotech has built a cold plate based cluster. Only three server racks provide space for a total of 384 booster nodes and can be cooled very energy efficiently. However, the thickness of the cold plate alone takes up as much space as the boards underneath. So there is still potential for savings and the possibility to pack the booster cluster more densely. Also, the ASIC implementation of the EXTOLL core was not ready in time to be integrated into the booster design process. In order to evaluate the performance of the Tourmalet ASIC within the project duration, work was done on another booster, the GreenICE system, which will be presented in the following sections.

Initially, the ASIC was planed as a drop in replacement for the BNCs's FPGAs described in section 5.2 on page 85. As all schematic and design files were closed source and the intellectual property of Eurotech, no design aid was given for an ASIC integration. That resulted in a development from scratch and a complete design with standard components.

5.4.1 Requirements

To get a clear picture what components are available or have to be developed, the first phase of the prototype development was to find a set of requirements that the prototype has to fulfill.

Use of standard components As time was one of the most limiting factors, the prototype has to consist of components that already existed or require a minimum of modification. The Intel Xeon Phi add-in cards in a dense form factor were used

because they had a very short lead time and are compatible with the Intel Xeon Phis used in the DEEP Project. This allows the usage of an already known accelerator that reuses all the knowledge of the previous prototypes. The Tourmalet ASIC PCIe Card was chosen for the NIC of the NAA. As the design team of the ASIC had a close relation to the chair of computer architecture, these cards were on stock and if modifications were required the complete design documents and specifications were available. Neither the time nor the financing allowed the design of a comparable backplane like the backplane used in the Eurotech DEEP booster. All network connectivity relies on the existing EXTOLL copper cabling

19 inch rack compatibility Another requirement was that the prototype must fit into a 19 inch rack to reuse existing infrastructure components. This limits the size of the prototype but it can be installed in every data center and allows the product deployment outside of the DEEP project.

Modularity The prototype should contain all necessary components to be operable, which includes the power supply, the internal and external network connectivity, the accelerators and the NICs. Not only the prototype itself should be a module that can be replicated and act as an independent and self containing component, also the inner components of the prototype have to be modular for easy replacement of damaged components. Therefore a certain amount of NAAs build a fixed block. Multiple of these blocks form the whole NAA cluster and external network connections connect to other prototype modules or to CPU nodes.

Dense Design As seen on the BNC boards from Eurotech, the cold plate requires a significant part of the overall volume of a BNC. In order to pack the prototype even more densely, either an even narrower cold plate must be developed that has the same heat transport properties, or it must be abandoned completely. Since it was not possible to develop a cold plate in the short time available and the necessary know-how was also not available, we had to use a new cooling method, the two-phase immersion cooling.

Interconnection All connections between the NAA's NICs has to fit into the prototype to be modular and reduce the number of external connections to a minimum. The network topology chosen has to connect all nodes inside the prototype with min-

imal distances between the nodes. A 3D torus with minimal cable lengths should be used and map on a 2D cable connection structure. The number of external connections and therefore the throughput between the prototype and the external components should be high enough to avoid a bottleneck.

Board Management and safety As a high availability system that should run long lasting simulations, the prototype requires remote management capabilities to safely power the system up and down. The prototype should also detect critical conditions and report them to a logfile and shut the system down to prevent damage. For this purpose, suitable sensors must be found that harmonize with the chosen cooling method.

5.4.2 Mechanical Design

Now that the requirements for the prototype have been defined, the following sections deal with the implementation of the previously mentioned requirements. Different materials could be used for the container. A metal box could be the obvious solution, because of the material properties. Its robust, easy to process, has a good pressure stability and no chemical reaction occurs with the liquid cooling. Nevertheless, the material's conductivity would be a problem and requires special care on the inside to avoid short-circuits. Under normal operation conditions all electrical devices inside the container have their own enclosure to avoid short-circuits between adjacent components. Also the thermal insulation of metal is too low to keep the heat inside the box and would radiate the heat into the server room. This requires additional air condition cooling capacity for the rack and the server room and is in conflict with the requirements to increase the power efficiency. The chosen material was acrylic glass for several reasons. It is transparent and allow visual inspection of the internal components, it is strong enough to handle the pressure that occur during the evaporation of the cooling liquid, can be glued very precisely and glued parts are extremely strong and is easy to process with tools available in our laboratory.

The outer construction consists of a cover with the cooling coils and their corresponding water inlet and outlet and a container that hold the power supplies and the NAA nodes. The exact dimensions can be seen in figure fig. 5.9 on page 92. The width from the left to the right side fits into a 19 inch rack and the length matches the half depth of a common server. With the armatures for the water inlet and outlet of the cover plate the prototype requires 12 height units. On the rear side are

Figure 5.8: Overpressure Valve

several holes for the power connections and the EXTOLL cabling to interface the prototype from the outside EXTOLL network.

The cover has a relief valve for handling the pressure while heating the fluid form cold 20 degrees to operational temperature of 50 degrees. It consists of a metal spring and a metal ball inside a acrylic glass box. The opening on the bottom is put over the hole in the cover plate and is the connection to the pressure chamber. The opening from the side is to adjust the maximum pressure after that the valve opens and releases the pressure. A screw controls the spring force which pushes the ball to the opening. Another hole on the top side of the acrylic box is the pressure outlet.

During the assembly of the cover plate and the container, additional information were available from 3M, the manufacturer of the cooling liquid Novec^{TM} 649 .
They discovered that based on the chemical structure of Novec^{TM} 649 water can diffuse from the acrylic glass into the liquid and causes a number of unintended side effects. Water reacts with Novec^{TM} 649 and create acids which increases the conductivity of the liquid over time and can build conductive deposits on heavily charged pins like ground or supply voltage pins. A workaround for this problem was the installation of glass plates on the inside of the container. The five glass

Figure 5.9: GreenICE Prototype

plates (4 sides + 1 bottom) were glued to the acrylic glass surfaces to insulate the Novec^{TM} 649 from the acrylic glass and avoid the diffusion of water. A drawback is the reduced space available for the inner construction.

5.4.3 Backplane design

To get the highest possible packaging density a new backplane was designed. A single *Dense Backplane* can hold eight Tourmalet NICs and eight Intel Xeon Phi in the dense form factor. Figure fig. 5.10 on the facing page shows the backplane with one NAA attached to the backplane.

Sixteen PCIe connectors are on the topside of the backplane with eight standard x16 PCIe connectors with 164 pins and eight x16 connectors with 280 pins to provide additional 12V power pins for the Xeon Phi DFF. The connectors are mounted as close to each other as possible which result in a distance between two adjacent boards of only 9mm.

Beside the PCIe connectors two 12V connectors are on the bottom side of the backplane. Each connector supplies four NAA nodes with power from two power supplies. Two DCDC converters generate a 3.3V voltage out of the 12V input. One DCDC converter supplies four NAA nodes. To monitor the power consumption of

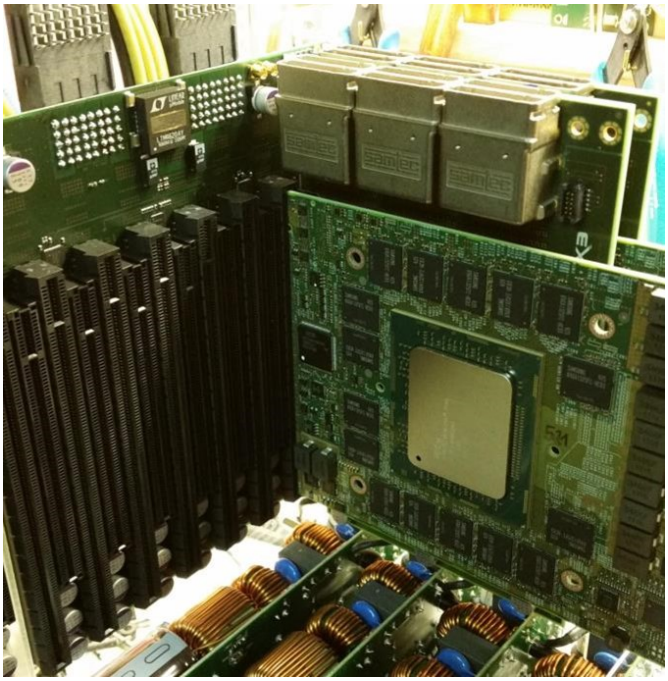

Figure 5.10: Dense Backplane with KNC

the backplane, all LTMs are accessible via I2C to read out the values for voltage and current.

All reset signals are controllable by I2C with two GPIO expanderchips. Each chip controls 8 reset signals with its GPIOs. The command to assert the reset or de-assert the reset is send over the I2C network to the backplane and according to the command executes the desired action.

5.4.4 Pressure resistance

The two phase cooling used for the GreenICE prototype requires the vaporization of the cooling liquid and the condensation of gaseous Novec^{TM} 649 back into its liquid physical state. Even with a global warming potential of 1 (as harmful as CO_2) the price per gallon of Novec^{TM} 649 requires a sealed container to avoid evaporation of Novec^{TM} 649 into the environment to keep the operation cost low. Due to the change of its physical state and the closed system the pressure inside the container fluctuates. To take care of the pressure, openings must be sealed. Special care was taken for the power connection from the outside into the container. Novec^{TM} 649 has a high surface tension and creeps through wires. Fine stranded conductors or conductors with a massive copper wire transport Novec^{TM} 649 between the insula-

tion material around the conductor and the wire from the container to the outside and evaporates. This requires special seals for all connections in and out of the container in particular if the opening is below or at the same level as the cooling coils. As Novec^{TM} 649 evaporate the gaseous Novec^{TM} 649 ascent to the cooling coils all space below the top of the coil is filled with gas that easily leak through all openings like wires. On solution is to put all openings above the cooling coils but this would increase the amount of height units required for the prototype which is not desired. The best solution found to avoid more height units and avoid leakage of Novec^{TM} 649 is the use of massive cooper bars sealed up in a Novec^{TM} 649 resistant adhesive like epoxy resin.

The sum of the power consumed by the prototype is 10KW. Divided by the line voltage of 230V this are 43.5A and a 6mm^2 thick copper bar is required. Instead of using one massive copper bar 5 smaller pins are used for the connection (3 phase, 1 neutral, 1 ground). On the inner side of the container is a 5 pin connector as a male plug-in unit. On the other side is a epoxy resin sealed female connector that is screwed against a sealing rubber.

Two problems arise for the pressure resistance of the covering plate. First, the cover plate has a high surface area and due to the pressure inside the container a huge force pushes against the cover plate. The other problem is the sealing material between the container and the cover plate. The material has to be hard enough for a high contact pressure to resist a partial pressure of 200mbar, soft enough to tightly seal all over the surface and has to be Novec^{TM} 649 resistant.

5.4.5 Interconnect

One of EXTOLL unique features is the direct interconnect with a build-in switch in every EXTOLL NIC. This is used to connect 32 NAAs in a 3D torus with the six EXTOLL links each NIC provides. Each card has three connectors with two links each. Each connector corresponds to a dimension in the 3D torus. One link of the connector is for moving up in the dimension (e.g. X+) and the other link is for moving down in the dimension (e.g. X-).

Different 3D torus variations are possible and a 4×2×4 topology was chosen for its even distribution of nodes over the dimensions to reduce the diameter and its best realization inside the containment. Because of the 2D physical arrangement of the NAAs inside the containment, the 3D network topology has to be mapped. A

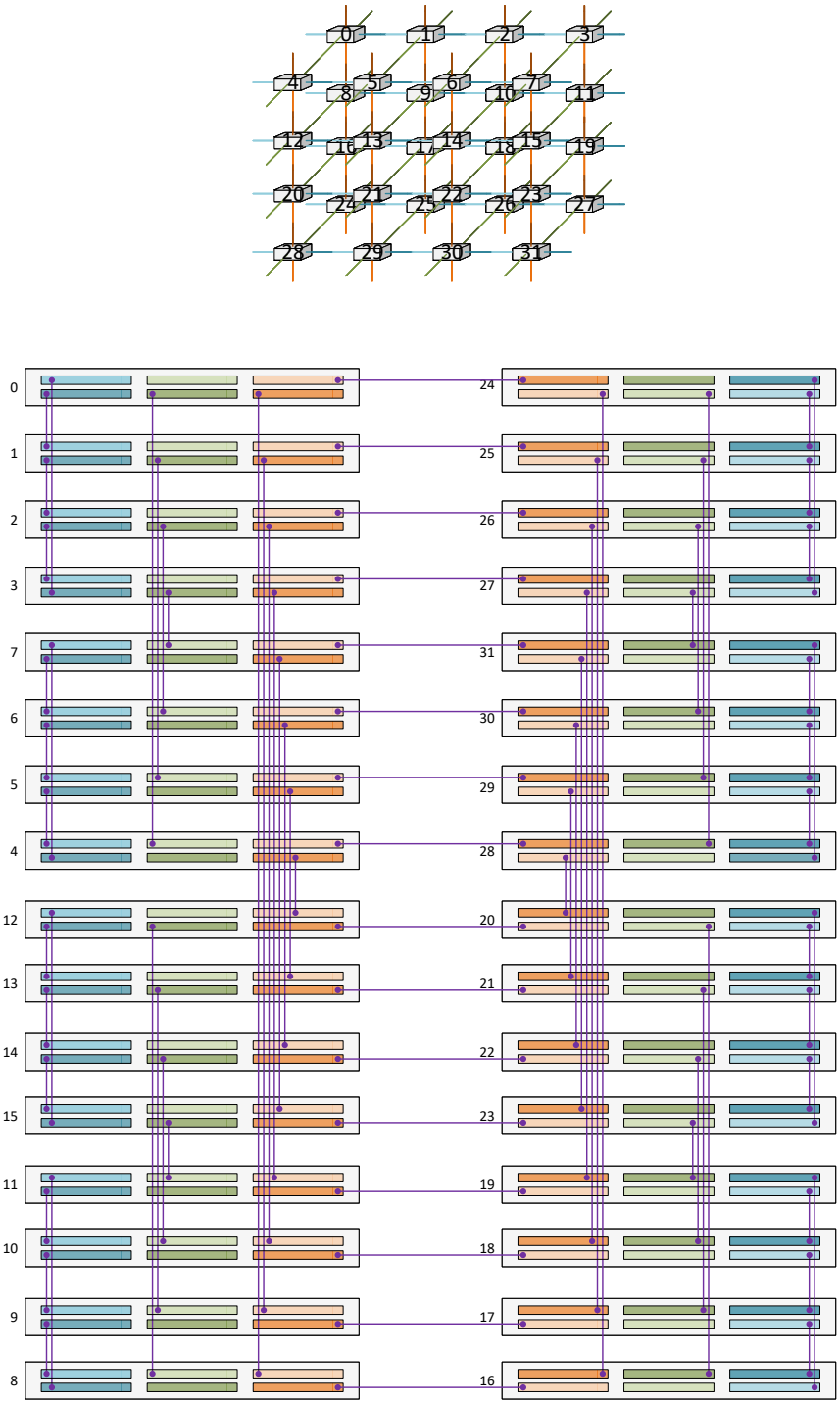

Figure 5.11: 32 Node Topology

constrain for this mapping is the use of the shortest paths possible to reduce the occupied volume of the cables. If the cable volume is to hight, the condensation of vaporized Novec^{TM} 649 will be affected. In the X-Dimension, four consecutive EXTOLL NICs form a X-Dimension ring. Three cards can be connected with very short cables and without crossings, because they are next to each other. A cable from the lowest to the highest EXTOLL NIC in the X-ring and spanning over two cards closes the X-Dimension ring. This cable fits also without crossing over the smaller ones. This allows for a very dense packing in the X-Dimension. The Z-Dimension takes advantage of the reversed order of the opposite column of NAAs. On the 16 nodes on the left side, the Z-Dimension connector has the Z+ direction on the upper link of the connector and the Z- link on the lower side of the connector. The 16 nodes on the right side have a reversed order of the Z-Dimension connector compared to that of the left side, which is Z- on the connector's upper link and Z+ on the connector's lower link. Connecting the Z+ links of the left side with the Z- links on the right side can be realized without crossing cables, because the connectors are next to each other. To close the four node ring in the Z-Dimension on each side the Z+ is connected with the Z- and forms a ring with the connections that connect both halves. The Y-Dimension has a depth of two and a single bidirectional link is required to close the ring in this dimension. The 16 nodes on a side are divided into two blocks of 8 nodes each. Starting in the middle of the block, both EXTOLL cards next to each other ($node_4$ and $node_5$) are connected and form a 2-depth Y-ring. The next ring spans between $node_3$ and $node_6$ and lies without crossing flat over the former ring. This is repeated for the remaining nodes in the block and also repeated on the other blocks.

As already mentioned, the space below the cooling coil is precious. EXTOLL has prefabricated cables with lengths of 30 cm, 50 cm, 75 cm and 100 cm. Their lengths corresponds to the distances between two servers in a rack. If two EXTOLL NICs are hosted in an 1U server, a 30 cm cable will be used to connect both servers that are next to each other. To connect two EXTOLL NICs in 2U servers, a 50 cm can be used and so on. As the nodes inside the containment are closer to each other than in a server rack, these cables are to long and waste to much volume below the cooling coil. Further more, the effort to plug all the cables increases because the cables cross each other.

To circumvent this situation, special cables were designed which are based on a thin flexible 4 layer PCB.

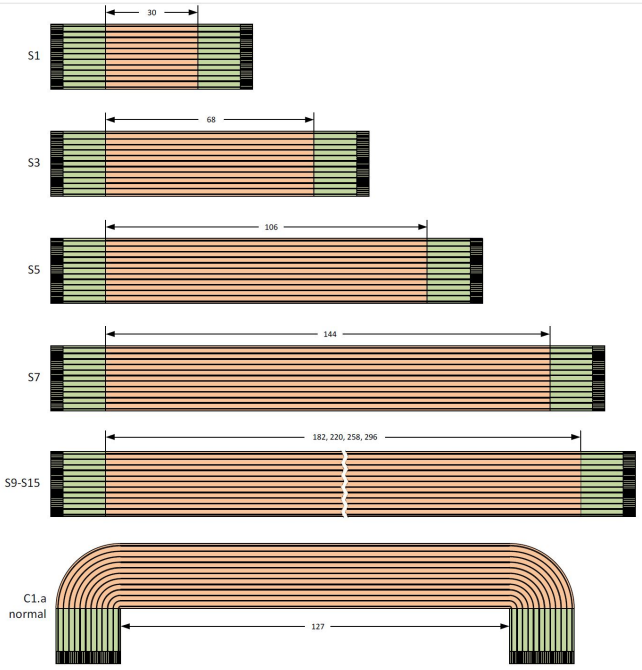

Figure 5.12: Rigid-Flex cables

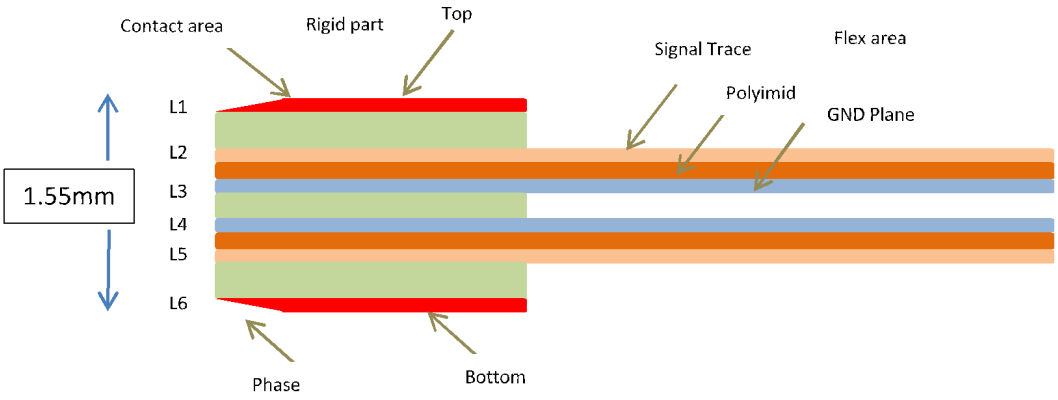

Figure 5.13: Rigid-Flex PCB Stack

Figure 5.14: Power Supplies

5.4.6 Power Supply

To adhere the modularity requirement, the power supply for all components are integrated into the containment. This has the added benefit of a reduced number of through-connections which avoids the already described problems for pressure stability and fluid leak. As the time frame to deliver the prototype to the project made a custom solution not feasible, standard PSUs were used. This choice comes with the risk, that the chosen PSU is not compatible with the Novec^{TM} 649 fluid. As seen in earlier tests with 5V power supplies, this can be a serious risk as these devices collapsed after immersion of Novec^{TM} 649 . Some components expected air and can not compensate the changes.

Eight Compuware[14] high efficiency PSUs were used with 1600 W each. To avoid congestion of vaporized fluid the cases are removed. All PSUs require a firmware patch to work without a fan. The firmware shuts down the PSU in case of a fan failure to avoid damage by overheating. Four KNC and four Tourmalet cards are powered by a single PSU. Each KNC has a Total Design Power (TDP) of 300 W and each Tourmalet card has a maximum power consumption of 25 W. A bundle of four NAA nodes has a total power consumption of $4 \times (300\,\mathrm{W} + 25\,\mathrm{W}) = 1300\,\mathrm{W}$ which leaves some reserve.

Power to the PSU is done with a 32 A three-phase AC current. Phase 1 and 3 are connected to 3 PSUs and phase 2 is connected to 2 PSU in the middle of the 8 PSU. These PSUs are pure 12 V devices. All other voltages are generated on the Dense Backplane (DBP) with DCDC converters. The 3.3V for PCIe are generated by two 12V-to-3.3V step down converters. All I^2C buses are 1.8 Volt and are generated also by step down converters.

5.4.7 Board Management and safety

To provide server class manageability the whole prototype logs important data from sensors and has software accessible switches to control the system operation. This Board Management Controller (BMC) uses a cost efficient Raspberry Pi 2 B+ instead of a full custom micro controller solution. The RaspberryPi provides an I^2C and Serial Peripheral Interface (SPI), several General Purpose Input/Output (GPIO) pins, Ethernet connector for remote access and Universal Serial Bus (USB) and High Definition Multimedia Interface (HDMI) to connect a keyboard and a monitor. The BMC runs a common Linux distribution (raspbian based on debian) and opens a wide range of software languages to manage the prototype. To integrate the program with other tools to observe and manage EXTOLL Tourmalet cards, the BMC uses scala a object oriented language based on JAVA .

5.4.7.1 I^2C network

Figure 5.15 on the next page shows the complete I^2C network used by the prototype. The RaspberryPi is connected to an I^2C expander [32] which drives the main I^2C lines to up to four other I^2C lines.

A custom PCB board was designed as a plug-on module for the 40 pin header of the RaspberryPi. This module as shown in fig. 5.16 on the following page converts the 5V power supply into the 1.8V voltage required by the I^2C interface and provides 4 plugs to connect additional I^2C buses to the first I^2C expander device.

5.4.7.2 Flowmeter

To detect heat removal shortage from the interior of hot vaporized Novec^{TM} 649 the water flowing through the copper pipe is monitored for inlet and outlet temperature and velocity. To measure this values a pipe of defined length is attached to the inlet or outlet plug. On both end of this pipe is a transducer connected as shown

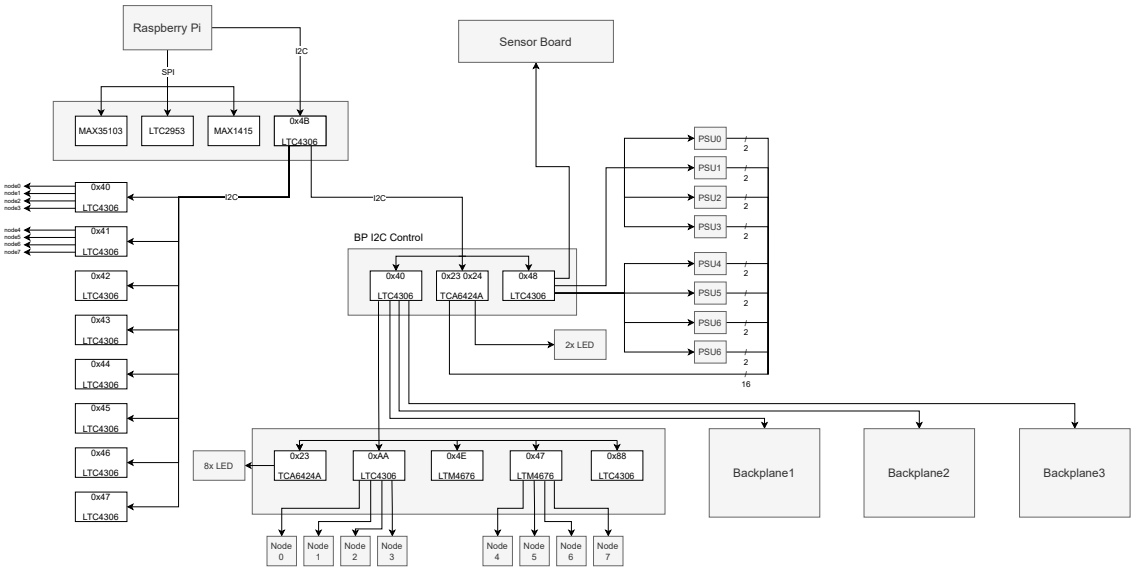

Figure 5.15: I²C Network

Figure 5.16: I²C Management Board

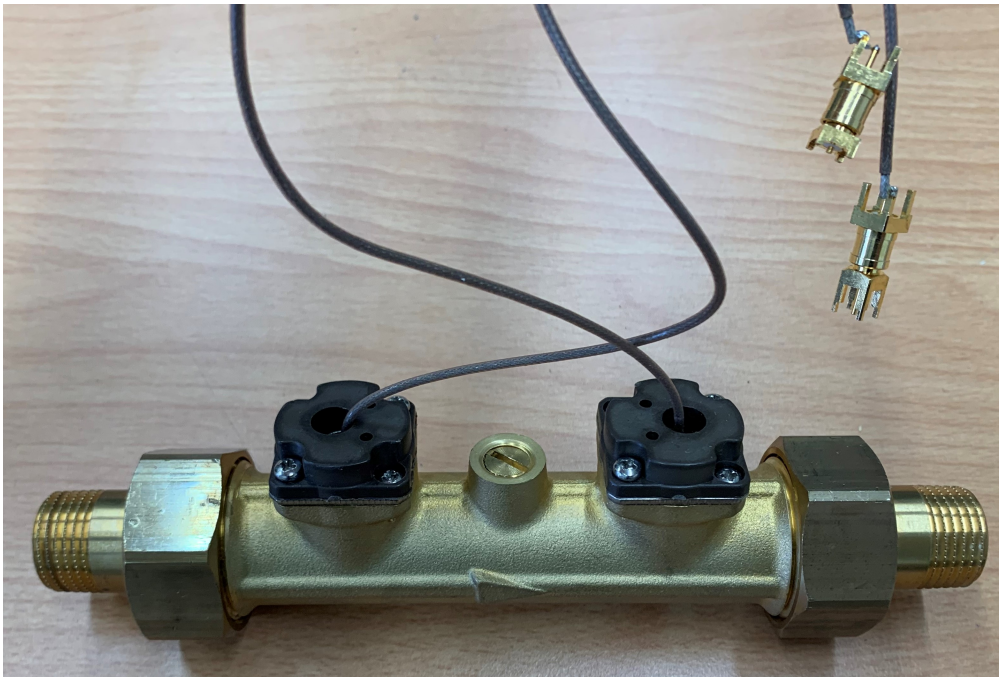

Figure 5.17: Flowmeter

in fig. 5.17. They send pulses of ultrasound waves along the pipe and measure the time difference between send off the wave and receiving of the pulse on the other side of the pipe. Due to the flowing water, the time between send and receive of the pulse in flow direction is shorter than a pulse in the opposite direction. The relation between the signal propagation delay of standing and flowing water can be calculated as the velocity of the water running through the pipe. To measure the water's temperature thermal sensors are placed inside the pipe.

5.4.7.3 Sensor-board

As described earlier, the Novec^{TM} 649 liquid is very volatile and to guarantee a safe operation the liquid's state has to be monitored precisely. The most important properties to monitor are the fluid level and temperature, and the pressure and temperature of the vapor above the fluid level.

For this purpose a sensor-board was developed as part of a student workand is shown in fig. 5.18 on the following page. A micro-controller is connected to a temperature and pressure sensor which measures the vapor's temperature and pressure. To monitor the fluid level a float gauge between light barriers is used and another temperature sensor on the lower end of the board measures the liquid

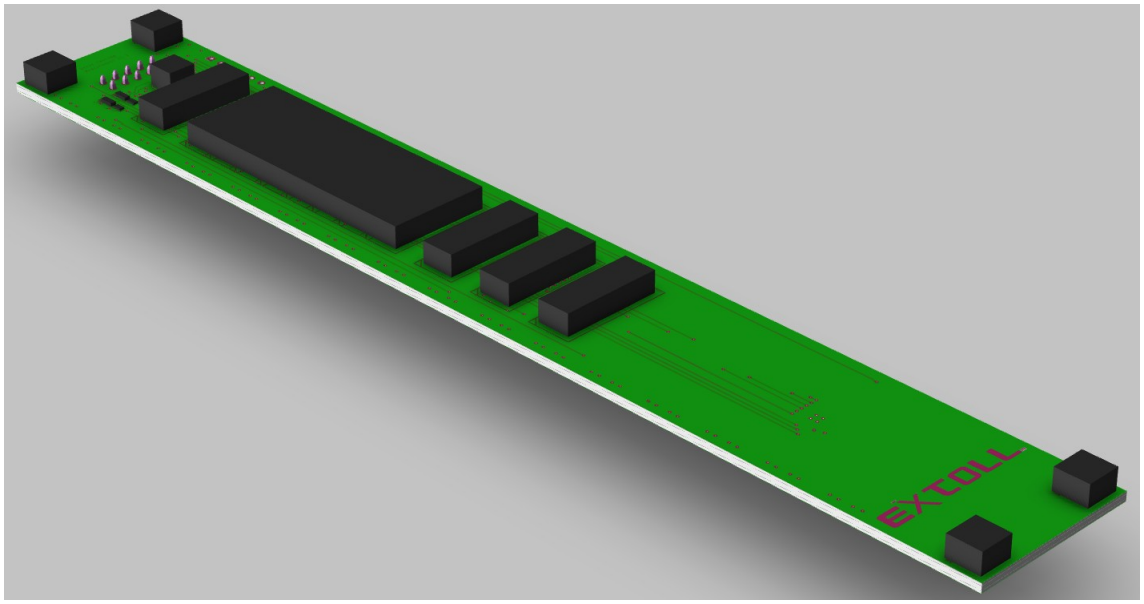

Figure 5.18: Sensor Board

temperature. The covered or un-covered light barriers generate a bit pattern and each pattern is a particular fluid level assigned. The PIC collects all sensor data and has a I^2C interface which is periodically read out by the board management controller.

5.4.7.4 Switch Actuators

All power-supplies can be enabled by a pin. Driving this pin low activates the power supply and the 12V are generated. A stable power output is signaled with power good signal. All these signals are driven by I^2C GPIO devices. Writes to registers activate or de-activate the corresponding PSU and the power state can be read from these registers too. The GPIO device is locate on the I^2C multiplexer board and drive all 8 PSUs.

With the same mechanism the resets are asserted and de-asserted. Each backplane as 16 distinct resets (8x Tourmalet 8x KNC) and a 16xGPIO device [33] is used per backplane to control all resets.

5.4.8 Cooling

As already described in several places, the GreenICE prototype was used for the first time to gain experience with the handling of immersion cooling. Therefore, this

section describes the experiences and observations made during operation.

Two of the most noticeable properties of Novec^{TM} 649 is its high vapor pressure together with its high surface tension. This has a major impact on the design of the GreenICE. If these two properties are not carefully considered, uncontrolled leakage of Novec^{TM} 649 will occur. Especially for through holes from the inside of the container to the outside, the high surface tension has a significant influence. Due to the high tension, Novec^{TM} 649 is able to wet even the finest structures, such as fine wire braids or cable shielding. The liquid is drawn along these and escapes. At the beginning of the prototype development, a considerable amount of Novec^{TM} 649 escaped through an Ethernet patch cable. This cable carries the twisted wires inside an unfilled protective sleeve and Novec^{TM} 649 was able to find its way out of the container.

The high vapor pressure has the consequence that considerable effort must be made to secure the container against high pressures, which arise not only from boiling, but also in the stationary system. The vapor pressure is a measure of the overpressure in a closed vessel at which natural evaporation and condensation are in balance. Below this vapor pressure, Novec^{TM} 649 evaporates. As long as the container is not opened, this balance is established, but with each maintenance a certain amount of liquid escapes until the balance is re-established after closing the container.

Everything described so far is valid for the stationary state. But also during active operation some interesting things happen. During the heat up of the system, Novec^{TM} 649 vaporizes and the partial pressure of 200 mbar can not longer hold back Novec^{TM} 649 from leaving its gaseous state and builds a gas layer over the liquid. As Novec^{TM} 649 is heavier than air two zones are forming up below the cover plate and the liquid surface. Novec^{TM} 649 gas has a higher volume than air, Novec^{TM} 649 compresses the air which leads to a higher pressure inside the container. This force also prevents Novec^{TM} 649 gas to reach the cooling coils and no condensation can occur to close the cooling circuit. To avoid this situation, the relief valve is adjusted in that way that the pressure inside the container is hold at 200 mbar overpressure compared to the surrounding environment. The air inside to container is blown out until the system reaches a point at which the vaporization and condensation at the cooling coils maintain balance and no additional pressure occur.

The dimensioning of the cooling coil also had a surprise in store. As mentioned above, Novec^{TM} 649 has a high surface tension and wets rough structures very well.

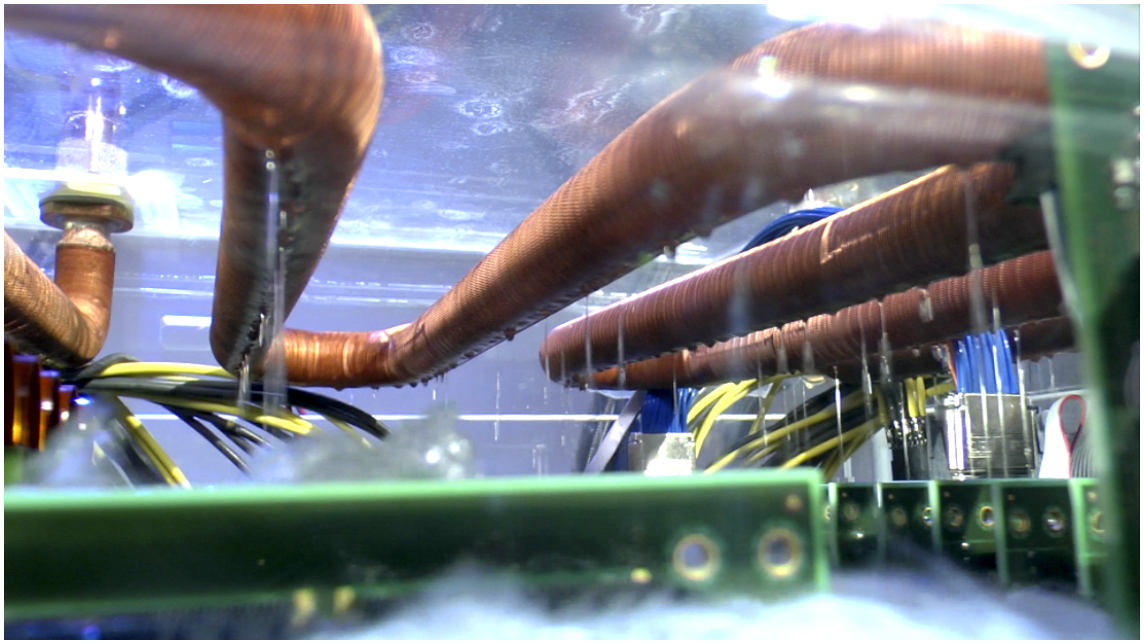

Figure 5.19: Condensation at Wieland cooling coil [65]

As a result, the cooling coil, which has an extra rough surface to increase the contact area to the gas, is constantly wetted with Novec^{TM} 649 . As described in section 2.5.3 on page 40, the thermal conductivity of Novec^{TM} 649 and air differ only slightly. Therefore, the condensed Novec^{TM} 649 that sticks to the cooling coil insulates and reduces the heat transfer considerably. As a result, a second cooling coil had to be installed to remove the 10KW from the system.

Even if the impression arises that you can simply immerse your electronics in Novec^{TM} 649 and the cooling takes care of itself, it is unfortunately not that simple. During the operation of the KNCs without a heat sink and thus a smooth surface, a chip temperature of over 100 degrees could be measured. At this temperature the KNC starts to reduce its performance to avoid overheating. To avoid this, a boiling enhancement coating was applied to all KNCs. These are small copper plates with one rough and one smooth side. Using a thermally stable adhesive, this BEC [24, 23] is bonded to the KNC chips. The surface on which Novec^{TM} 649 can boil is thus extremely increased and can realize a much better heat dissipation. After applying the BEC, the chip temperature was reduced from 100 to 80 degrees and a reduction in performance could no longer be observed.

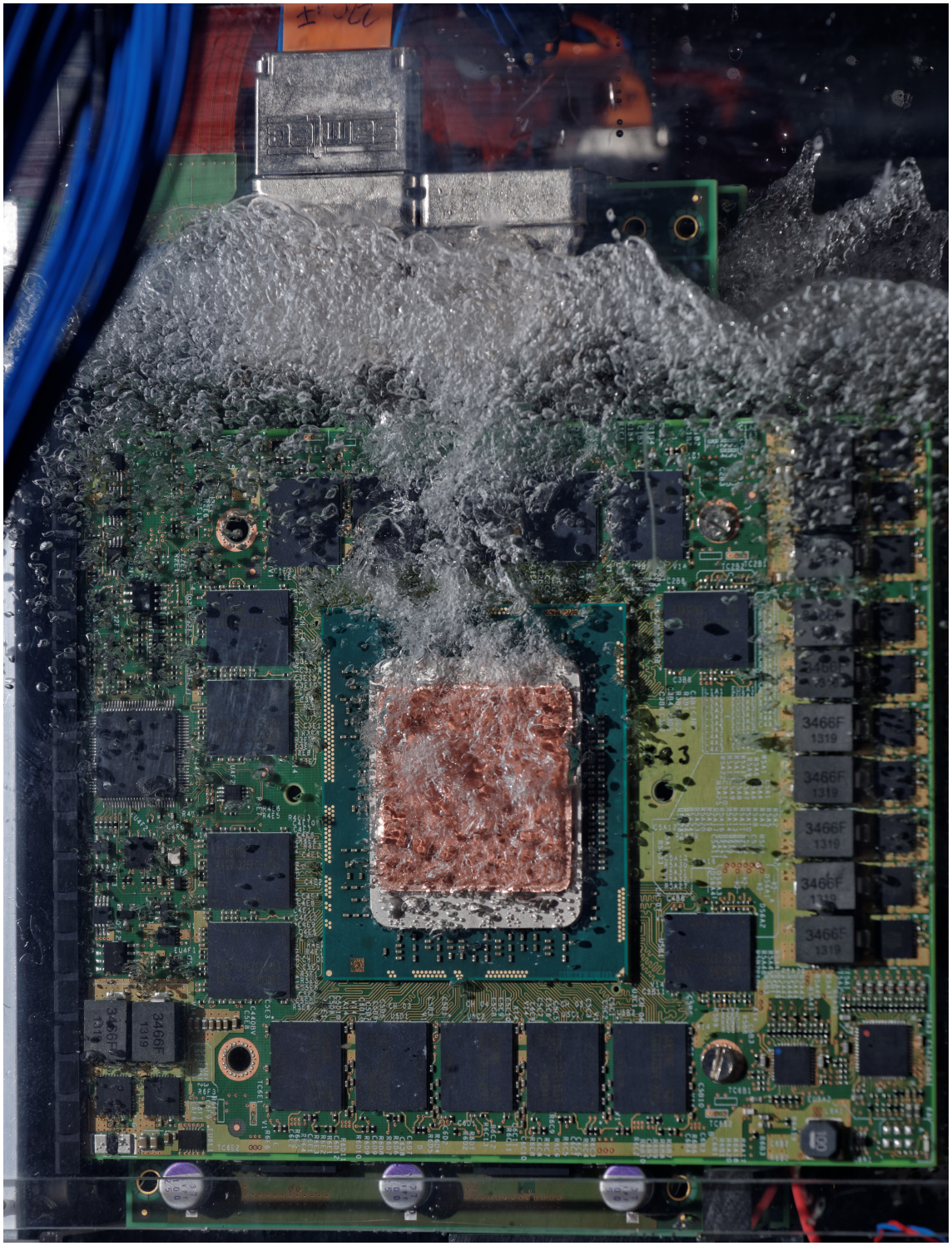

Figure 5.20: Novec^{TM} 649 boiling on KNC package with BEC

Chapter 6

Results

In this chapter the results of the thesis are presented and it summarizes the achievements and scientific contributions during this work. Since the work is divided into two parts, the results and findings are also discussed in two separate sections. First, it is analyzed how much of the Network Attached Accelerator could be implemented and which intermediate steps were necessary. Some of these contain unique approaches that did not exist in this form before. At the end of the chapter the results of the work with liquid cooling are presented and how to create a highly compact system by keeping the computing hardware to a minimum and using novel cooling technologies to achieve the highest density.

6.1 Cluster of Network Attached Accelerators

One aspect of the work was the design and implementation of a network attached accelerator as a follow-up to the idea from the work of Rinke et al. [55]. While in this work complete host systems are needed and additional software layers are required, in the course of this work an independent and not yet existing node type could be implemented. A prototype consisting of two NAAs and a cluster node was described and analyzed in detail in [38]. The results show that the communication time was improved by 32% to 47% and the bandwidth by 30%. But there were several obstacles to be overcome, which are listed below and are in themselves amazing and unprecedented achievements.

6.1.1 Remote PCIe Hierarchy

The main task was to reduce the number of components to a minimum in order to be able to operate an accelerator in a stand-alone, energy and space-saving manner. Since an accelerator does not have any communication capabilities except for its host interface, the missing functions ”device-to-device” and ”device-to-host” communication had to be provided by an additional component. This is done via the EXTOLL NIC whose functions are linked together in a completely new way. Without these features the concept of the NAA would not be possible. As one of the most important functions during this work it was possible to build up an independent PCIe hierarchy, which exists outside the host. This hierarchy serves as a 1-to-1 communication interface for data exchange between the NIC and the accelerator with maximum bandwidth and no other participants influencing the communication. NIC and KNC always have the full PCIe Gen3 x16 bandwidth available. The possibility to use the EXTOLL NIC not only as a normal Endpoint within a CPU / BIOS controlled PCIe hierarchy, but also to build up an independent hierarchy through the Root Port functionality makes this possible.

Besides the purely physical connection of the two components, it is also necessary to configure the remote and independent PCIe hierarchy. Since both the BIOS and the CPU have been removed, this must be done through a software-driven configuration that runs on one of the hosts inside the EXTOLL network. This emulates the configuration that takes place when a normal system is booted. Again, the unique capability of the EXTOLL NIC is used, which allows configuration access

not only from one dedicated node, but from any node within the EXTOLL network. This allows each EXTOLL node to take over the role of the ”configuration host” and provides redundancy and fail over

Furthermore you can use the software to create any PCIe packet and inject it over the network into the remote PCIe hierarchy. This system was originally designed to allow in-system debugging and testing. During this work this system was used and extended to create configuration packets that configure the Root Port of the NIC as well as the Endpoint of the KNC.

Another very important and unique approach is to tunnel the interrupts generated by the accelerator through the network to the host system that is currently using the accelerator. This is done by wrapping the interrupt from the remote PCIe bus into a network packet, sending it over the network and interpreting it as an interrupt on the destination node. Again, two systems of the NIC are used in a completely different way than originally intended.

It is also possible to achieve something unprecedented, namely to enable the accelerator to communicate actively, independently and exclusively driven by the accelerator. This is done by using the x86 architecture of the KNC to load the device drivers of the NIC. With this, all NIC functional units are available to the KNC for low latency two-sided communication and one-side zero-copy bulk transfers. To the best of our knowledge this is unique and has never been achieved before.

6.1.2 Transparent Memory mapping

After all the essential functions of the PCIe hierarchy have been made available to the KNC, another important goal was to ensure that existing drivers and other software do not need to be adapted and continue to function, even if the accelerator is only virtually and not physically connected to the host system. In order to access the on-chip memory of the KNCs and to allow the drivers to communicate with this memory in the usual way, an extensive configuration on the BI side as well as on the BN side is required. Again, one of the functions of the EXTOLL NIC is used, the SMFU. Actually this unit was intended to provide hardware support for shared memory applications, but during this work a transparent connection between host and KNC was created by tricky mapping of memory areas from the BI address space. This transparent connection allows memory accesses on the host to be encapsulated into network packets and sent to the remote NAA with very low latency, without any

additional software layer and without wasting CPU cycles. At the remote side on the NAA the network packets are unpacked without the need of any other component and execute the memory access on the remote side, which would have been intended for the local memory system. The user or the application can bypass all software stacks and thus communicate directly with the KNCs memory only by read or write to certain address ranges. The BI has access to all the KNC device memory that is mapped into the BI system. In the other direction the KNC can access all memory addresses of the BI system.

6.1.3 Dynamic assignment of accelerators

Another important goal was to break up the previous rigid allocation between CPUs and accelerators and make it more dynamic. Until now, only a limited number of accelerators could be installed in a host system. On the one hand, the available space limits the number of accelerators and on the other hand the number of PCIe lanes provided by the CPU. This condition can be relaxed by tree-like hierarchies with the assistance of PCIe switches, but this results in other limitations like a precise orchestration of the communication directions to achieve maximum bandwidth as described in [36].

With the NAA approach, the independent accelerators are not connected to each other by expanding the PCIe hierarchy, but with a specialized and powerful interconnect (EXTOLL). This allows the accelerators to be arranged in any network topology, increasing bandwidth and reducing latencies.

If the cluster nodes are connected to this interconnect or indirectly via a bridge structure between the existing cluster interconnect and the NAA interconnect, a dynamic assignment can be achieved. Several cluster nodes can allocate one or more accelerators and exchange data. Due to a multidimensional topology, e.g. 3D torus, the actual position of the accelerators in the topology has less impact on the available bandwidth and latency. In existing systems, it is necessary to pay very close attention to where the nodes are located in the network. This effort is not required in the NAA approach.

Another positive side effect is the increased utilization of the individual accelerators, since the number of available accelerators is not limited to the number of free host systems. If there are applications that are only executed on the CPU, the accelerators are unused and are not available for other jobs. With the NAA

approach, this dependency is removed because only the number of free accelerators is important and not the number of free hosts. Thus the number of accelerators can be reduced, which reduces the power consumption of the system.

6.2 Liquid cooling

Now that the NAA exists as an independent node capable of operating with a very small number of components, the desire to package these remaining components (KNC and NIC) as densely as possible is inevitable. Since neither suitable housings nor platforms nor cooling capabilities exist to operate the NAA, new solutions had to be developed. This developments resulted in the GreenICE prototype.

6.2.1 GreenICE

To achieve a high packing density and to meet the demands of liquid cooling, a new type of housing had to be developed. This had to meet the requirements for sealing, high thermal insulation against the ambient air, maintainability, low height and width and many other innovations had to be invented. All these requirements are covered by the newly developed GreenICE system. Its design as a container with a lid, in which the cooling coils are installed, allows for good maintenance. Both the lid and the tank are made of acrylic, which has a high thermal resistance, allows a good view of the inside of the tank, and releases little heat into the room. Furthermore the acrylic parts like the bottom plate and the 4 side walls are glued together and get a very high stability to withstand the pressure of 1-2 bar.

In order to be able to operate the GreenICE as a closed system, a special seal was developed and applied between the edge of the tank and the lid. Normal sealing materials are either too soft and do not allow a high contact pressure or react with Novec^{TM} 649 and therefore wear out. The in-house developed gasket is stable but also elastic enough to withstand the high pressure and is stable against Novec^{TM} 649 .

If the system is ”started up” from a cold state, a high pressure is generated, which must be released in a controlled manner. If this pressure would remain in the system, the Novec^{TM} 649 gas would be forced under the cooling surface and circulation would be stopped. To ensure that only as much air has to be released from the container as necessary, a separate pressure relief valve was developed. This

is adjusted in such a way that only the air portion is blown out and closes when Novec^{TM} 649 starts to condense on the cooling surface. This is where Novec^{TM} 649 's useful properties come into play, as it has a much higher density than air. This ensures that there is only air above the cooling surface and Novec^{TM} 649 below the cooling surface. As the pressure rise ends when condensation begins, the system is self-regulating. At this point the system is in a state of balance and no further blow-off or pressure changes occurs.

In order to prevent Novec^{TM} 649 from crawling along cable openings into the room, all cables were made solid. The power supply was provided by a special plug which can be plugged into solid copper pins. These pins were moulded with special sealing material and thus sealed.

To protect the system, several sensors have been placed inside and outside and switch off the system when a critical state is reached. On a sensor board inside the vessel the liquid level is monitored by a light barrier and a pressure sensor is mounted on the same board above the liquid. Temperature sensors inside and outside the liquid can be used as further indicators. Outside the liquid there are sensors that monitor the water flow through the cooling surface, the board management controller (BMC), a RaspberryPi, as well as the external sensors for the BMC power supply and a watchdog. All sensor data is analyzed, measured data is stored and in case of a critical condition the complete system is switched off. If the power supply to the BMC is interrupted or if the BMC hangs and the sensor data is no longer evaluated or there is no flow through the cooling surface, the system is switched off.

The GreenICE system is through all these measures a closed and ready to use system which contains 32 NAA nodes, consumes only 12 U rackspace of a 19 inch rack, delivers up to 32 TFLOP computing power and has a cooling capacity of 10KW. There are also no noise sources such as fans or pumps that need to keep the air or liquid moving. The cooling is highly efficient because it uses the phase transition. It is a continuous process that is self-contained by condensation and evaporation without the need for external energy.

6.2.2 Higher packing density

Over several iterations the backplane, which includes the EXTOLL NIC and the KNC, has been further developed. From a module that built only one NAA to a backplane that can accommodate 8 NAAs. On this Dense backplane are 16 PCIe

x16 connectors, 8 for the KNC and 8 for the NIC. All connectors are mounted side by side without any gaps. The cards have only a distance of 9mm to each other. *If one would put an air-cooled Dense form factor KNC into a slot of the Dense backplane, the heat sink of the KNC alone would cover 3 more PCIe slots. In these 4 occupied slots (one for the KNC, 3 covered by the heat sink) 2 complete NAAs would fit in. This means that a total of 2 complete NAA can be installed in the space consumed by a single air-cooled dense form factor KNC and therefore the same volume with NAAs has 2 times more computing power than an air cooled KNC version alone. This ratio could be further improved by reducing the distance between the PCIe connectors. But here one is limited by the design of the connector. If this distance could be reduced, the cooling capacity would be sufficient, because the effect of the existing thermosiphon is then further enhanced. The rising vaporized Novec^{TM} 649 carries cold liquid Novec^{TM} 649 with it and moves it past the hot surface. If the distance between the boards decreases further, this effect increases, more Novec^{TM} 649 moves past the hot surface and can still provide enough cooling capacity.

Another contribution to the high packing density is provided by the direct interconnect from the EXTOLL NICs. This allows each individual node to be directly connected to 6 further nodes. If the cabling is chosen carefully as described in the section fig. 5.11 on page 95, the number and length of cables can be significantly reduced. Since the existing AWG cables are too long, a special cable was developed to meet the required length. These ”rigid-flex” cables are flexible, thin two-layer printed circuit boards and significantly reduce the volume of cable between the coil and the top of the board. This has reduced the height and made more space available in the rack. Another advantage is the smaller distance between the liquid surface and the cooling coil, as the Novec^{TM} 649 gas reaches the cooling coils faster and condensation occurs faster. Furthermore the volume of the Novec^{TM} 649 gas is reduced and therefore the possible pressure build-up.

With all these individual components it was possible to achieve a 2 times higher packing density compared to an air-cooled version. If you compare the packing density of an NAA node with that of a water-cooled backplane version, you get a factor of 3.4 higher packing density.

6.2.3 Increased power efficiency

In addition to the increased packing density, the energy efficiency of the entire system has also been increased. There are no moving parts like fans or pumps in the entire GreenICE system. There are also fewer components that can wear out over time and consume additional power. If you consider the energy saving alone through the KNC fans that are no longer needed, you save about 20W per NAA [29] or 640W per GreenICE. Furthermore many other components are saved per NAA node like CPU, chipset, memory and various fans. The most important feature is that the whole GreenICE system can be cooled with only 50W pump power.

A further contribution to energy savings was made by the Interconnect. By using the direct interconnect of the EXTOLL network, the detour via an external switch can be avoided, since the switch functionality is integrated in the NIC. This reduces the cable length from several meters per connection to a few centimeters. Due to the much shorter transmission channels, the signal is less attenuated and does not need to be equalized with additional energy. Also the signal transmission can be done without optical transceivers and only in copper, which leads to considerable savings.

However, energy savings are not only achieved at the consuming end, but also in the generation of power at the PSUs. While 8 standard PSUs were installed in the current system, a single specially developed PSU could be installed in a subsequent system instead. This would have a lower height, as many redundant components can be avoided in the 8 standard PSUs. Since the standard PSUs are not designed for use in Novec^{TM} 649 , the self-developed PSU can be designed in such a way that Novec^{TM} 649 's cooling capabilities can be best utilized. In addition, the efficiency would be improved by eliminating AC/DC conversion and instead using a much more efficient DC/DC conversion. With the appropriate components, like Vicor Chips[63] an efficiency of 98% can be achieved with a conversion from 400V to 12V.

6.2.4 Experience with Novec^{TM} 649

In the course of the work, knowledge of Novec^{TM} 649 was gained. Liquid cooling allows a packing density that can never be achieved by other media such as water or air. The fluid also has a very good Global Worming Potential of 1 and therefore harms the environment less than other media such as mineral oil. Due to its non-conductive properties, it can also be used to operate high performance electronics

without causing short circuits. Since Novec^{TM} 649 drips and evaporates without leaving any residues, the components are immediately accessible for maintenance after removal from the liquid and do not have to be cleaned first. The much lower boiling point of Novec^{TM} 649 compared to water or mineral oil makes the fluid a very good medium for 2-phase applications to make use of the latent energy during phase transition. Another advantage of Novec^{TM} 649 is that it does not pose a health hazard. Neither the liquid nor the gases are poisonous and the physical contact with Novec^{TM} 649 is harmless.

However, Novec^{TM} 649 is not an easy material and requires a lot of experience and knowledge. In order to achieve the best possible cooling performance and to be able to exploit the full potential, extensive tests and experiments were necessary. Since Novec^{TM} 649 is a very novel product, no long term experience or best practices could be used. Therefore basic research was necessary in which some problems arose, but for every problem a solution could be found. All this knowledge was a prerequisite for the development and operation of the GreenICE system.

Chapter 7

Conclusion

Communication and energy consumption play an increasingly important role and become limiting factors in the area of high performance computing. The present work contributes to these two points by generating a novel node type, the NAA. This innovative node type, which has not yet been implemented by any other, makes it possible to improve communication and energy consumption in several ways. The NAA successfully eliminates many redundant components and reduces the number of active components to two devices. Thus, this new node consists only of a highly specialized NIC with additional functions and an accelerator, in this case an Intel® Xeon Phi™. Without the special functions beyond those of a normal NIC, the whole concept of NAA is not feasible.

One of the outstanding features is the possibility to build a remote PCIe hierarchy that is fully software configurable over the network. It is possible to create a PCIe bus independent of the host system, which directly connects NIC and accelerator. The enumeration is completely emulated in software and can therefore configure any accelerator. The configuration also includes the creation of an independent physical address space that is unrelated to the host system.

Another unique feature of the NIC is the ability to map remote physical memory into its local address space over the network. This feature allows the host-independent physical memory space on the remote PCIe bus to be transparently accessible to the host applications without any driver changes. As a result, all communication with the accelerator is done with read and write operations on memory addresses and no CPU driven driver stack is required for network communication.

The last function that was successfully implemented was the forwarding of interrupts from the remote PCIe hierarchy to a host system via the network. Again, thanks to the transparent memory mapping, there was no need to modify the existing applications.

During the work it was possible to combine all these functions in such a clever way that the accelerator is available as a fully independent node, after booting the KNC operating system over the network. This way, a CPU is only needed for a short time during initialization. Once the node has been initialized, the CPU is no longer necessary and can perform tasks that it can handle more efficiently. After initialization, the accelerator can perform network communication on its own. It can do this either by transparent memory mapping that encapsulates memory accesses in network packets or by using the specialized communication units of the EXTOLL NIC.

To the best of our knowledge, this is the first time that an accelerator can communicate independently without the assistance of a CPU.

Several NAA nodes can now be connected to form a *cluster of accelerators*. Here, this new node type has further advantages over the previous way of connecting accelerators with each other. The disadvantages of multiple accelerators on one PCIe hierarchy are completely avoided, since highly efficient network communication with higher bandwidth, lower latency and more degrees of freedom in the topology can be used.

Since no host systems are needed anymore, the size of the cluster can be varied arbitrarily. Since there is no fixed assignment of accelerators to a host, both systems can be scaled arbitrarily without affecting each other. This allows an arbitrary number of CPU systems with an arbitrary number of NAAs. Thus the utilization of the CPUs as well as the accelerators can be increased. Furthermore a dynamic allocation of CPU and accelerator is possible which can change during runtime. Up to now this allocation was already fixed by the direct PCIe slots connected to the host CPU.

The NAA was not only designed conceptually and in simulations, but was actually realized in several prototypes of different sizes and its advantages were confirmed by benchmarks. The many experiences from the individual prototypes with one or more NAA nodes finally resulted in the unique GreenICE system, which has unprecedented properties that are only possible through the NAA concept. This system also has a four times higher packing density, making it the most densely packed system in the HPC field. To achieve this, a novel cooling technique based on Novec^{TM} 649 has been successfully applied. To be more efficient than previous systems that rely on single-phase cooling and require energy-consuming recirculation, the GreenICE uses two-phase cooling. This results in an amazing Power Use Efficiency (PUE) of 1.01. An unprecedented performance. This value could only be achieved by the closed and pressure-tight construction and the use of the thermosiphon effect. This allows the 12 kW power inside the GreenICE to be cooled only through a single external circulation pump for the water circuit to remove the heat. To reach this point, a sophisticated pressure management system was developed to handle the interaction between air and Novec^{TM} 649 gas. Also, special seals and cable feedthroughs were developed that are resistant to Novec^{TM} 649 . Furthermore, a sensor network was established to ensure safe operation.

A secondary but very important effect of the four times higher packing density is

the much shorter propagation distances within the cluster of accelerators. This leads to energy savings in the communication, since the signals need less amplification to be received at the target and also the receive logic can be operated more efficiently, since complex signal reconstruction can be avoided. The direct connection of the accelerators to each other can reduce the latency considerably.

All these developments resulted in a cluster of accelerators with 32 nodes and 32 TFLOP and was successfully installed and operated at Forschungszentrum Jülich within the DEEP project.

The concept of the NAA presented in the paper is general enough to work not only with the KNC. However, there are some components that are specific to the KNC and therefore cannot be used for other types of accelerators. GPGPUs would be a worthwhile target for further work on the NAA. While these accelerators do not have the special features of the KNC as the native operating system, their widespread use makes development rewarding. In fact, efforts have been made to test the feasibility of the NAA approach for GPGPUs as well. Again, a remote PCIe hierarchy between the NIC and the GPGPU, in this case a NVIDIA®-Tesla® K20, could be successfully established. The same transparent memory mapping mechanisms could be used for the GPU memory and the GPGPU driver could be loaded without modification. However, after several tests it turned out that CUDA programs could not be executed. The exact cause could not be determined, since the drivers are closed source. Nevertheless, it would be worthwhile to do further research here, since the NAA approach has so many advantages and it is interesting to see how this can affect GPGPUs.

List of Figures

List of Tables

Acronyms

APIC Advanced Programable Interrupt Controller. 23, 77

ARM Accelerator Resource Manager. 6

ATOLL Atomic Low Latency. 28

ATU Adress Translation Unit. 35

AVX-512 512-bit Advanced Vector Extensions. 17

BEC Boiling Enhancement Coating. 41, 104

BI Booster Interface. 84, 85

BMC Board Management Controller. 99

BN Booster Node. 84, 85

BNC Booster Node Card. 85, 86

CAG Computer Architecture Group. 28

CfgRd Configuration Read Request. 70

CfgWr Configuration Write Request. 70, 72

CN Cluster Node. 84

COTS Commodity Off-The-Shelf. 50, 59

CplD Completion with Data. 56

CPU Central Processing Unit. 3

CUDA Compute Unified Device Architecture. 15

DARPA Defense Advanced Research Projects Agency. 5

DBP Dense Backplane. 99

DMA Direct Memory Access. 19, 35

EMP Extoll Management Program. 33

EXTOLL Extended Atomic Low Latency. 28

FPGA Field Programmable Gate Array. 56–58, 80, 85

FSM Finite State Machine. 36

GDDR5 Graphical Double Data Rate 5. 15

GGAS GPU Global Address Space. viii, 7, 59, 67

GPGPU General Purpose Graphic Processing Unit. vii, 11, 12

GPIO General Purpose Input/Output. 99, 102

GPU Graphics Processing Unit. 3

HDMI High Definition Multimedia Interface. 99

HPC High Performance Computing. 1, 28, 52, 61

HT HyperTransport. 29

HTAX HyperTransport Advanced X-Bar. 32, 35

HTOC HyperTransport On-Chip. 28, 29

I²C Inter-Integrated Circuit. ix, 99, 100, 102, 125

ISA Instruction Set Architecture. 17

KNC Knights Corner. 16

LQCD Lattice Quantum Chromo-Dynamics. 57

MemRd Memory Read Request. 56

MemWr Memory Write Request. 56

MIC Many Integrated Core. 11

MMIO Memory Mapped Input/Output. 24, 28

MMU Memory Management Unit. 18

MPI Message Passing Interface. 28

MSB Most Significant Bit. 56

MSI Message Signaled Interrupt. 23, 77

MSI-X Message Signaled Interrupt Extended. 23

NAA Network Attached Accelerator. 11, 18, 46, 48–50, 52, 53, 56, 61, 65, 68, 69, 85, 98, 118, 119

NIC Network Interface Controller. 3

NTB Non-Transparent Bridge. 54, 55

PCB Printed Circuit Board. 8, 85, 86, 88, 96, 99

PCI Peripheral Components Interconnect. 21, 66

PCI-X Peripheral Components Interconnect - Extended. 21

PCIe Peripheral Components Interconnect - Express. 3, 21

PEACH2 PCI Express Adaptive Communication Hub Version 2. 56, 57

PGAS Partitioned Global Address Space. 7, 30

PIO Programmed Input/Output. 19

PSU Power supply unit. 62, 98, 99, 102

PUE Power Use Efficiency. 119

QPI Quick Path Interconnect. 48

QSFP Quad Small Form Factor Pluggable. 83

QUonG lattice QUantum chromo-dynamics ON Gpu. 57

RDMA Remote Direct Memory Access. 5, 28, 35, 58

RF Register File. 35–37

RMA Remote Memory Access. 35–37

RRA Remote Registerfile Access. 35–37, 70

SIMT Single Instruction Multiple Threads. 15

SME Shared Memory Engine. 60

SMFU Shared Memory Functional Unit. 28, 30

SMX Streaming Multiprocessor. 14

SNQ System Notification Queue. 34

SPI Serial Peripheral Interface. 99

SSH Secure Shell. 18

TDP Total Design Power. 98

TLP Transaction Layer Protocol. 25

UPS Uninterruptible Power Supply. 3

USB Universal Serial Bus. 99

VPU Vector Processing Unit. 17

Bibliography

[1] *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, 2005. ISBN 0321335597. URL `https://developer.nvidia.com/gpugems/GPUGems2/gpugems2_frontmatter.html`.

[2] Allied control to reveal 1.4 megawatt datatank container data center with pue 1.01 and 240 kw racks for 3m ™ novec ™ engineered fluids. *ALLIED CONTROL*, Nov 2014. URL `http://www.allied-control.com/publications/Allied_Control_Revealing_DataTank_Press_Release.pdf`.

[3] R. Ammendola, M. Bernaschi, A. Biagioni, M. Bisson, M. Fatica, O. Frezza, F. Lo Cicero, A. Lonardo, E. Mastrostefano, P. S. Paolucci, D. Rossetti, F. Simula, L. Tosoratto, and P. Vicini. Gpu peer-to-peer techniques applied to a cluster interconnect. In *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pages 806–815, May 2013. doi: 10.1109/IPDPSW.2013.128.

[4] Roberto Ammendola, Andrea Biagioni, Ottorino Frezza, Francesca Lo Cicero, Alessandro Lonardo, Pier Paolucci, Roberto Petronzio, Davide Rossetti, Andrea Salamon, Gaetano Salina, Francesco Simula, Nazario Tantalo, Laura Tosoratto, and Piero Vicini. Apenet+: a 3d toroidal network enabling petaflops scale lattice qcd simulations on commodity clusters. 12 2010.

[5] Roberto Ammendola, Andrea Biagioni, Ottorino Frezza, F Lo Cicero, Alessandro Lonardo, Pier Stanislao Paolucci, Davide Rossetti, Francesco Simula, Laura Tosoratto, and Piero Vicini. Apenet+: a 3d torus network optimized for gpu-

based hpc systems. In *Journal of Physics: Conference Series*, volume 396, page 042059. IOP Publishing, 2012.

[6] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, 15, 2008.

[7] Broadcom. *PEX9700 Series Switch Chips*. Broadcom, May 2018.

[8] André R Brodtkorb, Trond R Hagen, and Martin L Sætra. Graphics processing unit (gpu) programming strategies and trends in gpu computing. *Journal of Parallel and Distributed Computing*, 73(1):4–13, 2013.

[9] Ravi Budruk, Don Anderson, and Tom Shanley. *PCI express system architecture*. Addison-Wesley Professional, 2004.

[10] George Chrysos. Intel® xeon phi™ x100 family coprocessor - the architecture. November 2012. URL `https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner`.

[11] Computer Architecture Group. *HyperTransport On-Chip (HTOC) Protocol Specification*. Heidelberg University, . Version: 1.6.

[12] Computer Architecture Group. *RMA2 Specification*. Heidelberg University, . Revision: 2.0.4.

[13] Computer Architecture Group. *SMFU2 / Excelerate Technical Documentation*. Heidelberg University, . Revision: 1.2.

[14] Compuware Technology Inc. *Power Supply Specification Model: CPR-1621-1M21*, December 2011. Rev: 1.0.

[15] HyperTransport Technology Consortium et al. Hypertransport i/o link specification. *Revision*, 1:111–118, 2008.

[16] ALLIED CONTROL. Allied control immersion cooling, 2019. URL `http://www.allied-control.com/xeon-phi-immersion-cooling-concept/`.

[17] Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 4:1–4:12, Piscataway, NJ, USA, 2008. IEEE Press. ISBN 978-1-4244-2835-9. URL `http://dl.acm.org/citation.cfm?id=1413370.1413375`.

[18] DEEP. Deep prototypes, 2015. URL `https://www.deep-projects.eu/hardware/prototypes.html`.

[19] Jack J Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: past, present and future. *Concurrency and Computation: practice and experience*, 15(9):803–820, 2003.

[20] Dr Donald Kinghorn. P2p peer-to-peer on nvidia rtx 2080ti vs gtx 1080ti gpus, 2020. URL `https://www.pugetsystems.com/labs/hpc/P2P-peer-to-peer-on-NVIDIA-RTX-2080Ti-vs-GTX-1080Ti-GPUs-1331/`.

[21] J. Duato, A. J. Peña, F. Silla, R. Mayo, and E. S. Quintana-Ortí. rcuda: Reducing the number of gpu-based accelerators in high performance clusters. In *2010 International Conference on High Performance Computing Simulation*, pages 224–231, June 2010. doi: 10.1109/HPCS.2010.5547126.

[22] Scott Ellis. Exploring the pcie bus routes. URL `https://intrepid.warped.com/~scotte/OldBlogEntries/web/index3429.html?replytocom=1848`.

[23] E. C. Forrest, L. Hu, T. J. McKrell, J. Buongiorno, and Y. Ostrovsky. Pressure effects on the pool boiling of the fluorinated ketone c2f5c(o)cf(cf3)2. In *2010 12th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, pages 1–9, June 2010. doi: 10.1109/ITHERM.2010.5501414.

[24] Eric C Forrest, Lin-Wen Hu, Jacopo Buongiorno, and Thomas J McKrell. Pool boiling heat transfer performance of a dielectric fluid with low global warming potential. *Heat Transfer Engineering*, 34(15):1262–1277, 2013.

[25] Holger Fröning and Heiner Litz. Efficient hardware support for the partitioned global address space. In *Parallel & Distributed Processing, Workshops and Phd*

Forum (IPDPSW), 2010 IEEE International Symposium on, pages 1–6. IEEE, 2010.

[26] Holger Fröning, Mondrian Nüssle, David Slogsnat, Patrick R Haspel, and Ulrich Brüning. Performance evaluation of the atoll interconnect. In *Parallel and Distributed Computing and Networks*, pages 129–134. Citeseer, 2005.

[27] Toshihiro Hanawa, Yuetsu Kodama, Taisuke Boku, and Mitsuhisa Sato. Interconnection network for tightly coupled accelerators architecture. In *High-Performance Interconnects (HOTI), 2013 IEEE 21st Annual Symposium on*, pages 79–82. IEEE, 2013.

[28] https://www.top500.org/green500/. Green500. `https://www.top500.org/green500/`, January 2019. URL `https://www.top500.org/`. Accessed: 2019-01-28.

[29] *Intel® Xeon Phi™ Coprocessor x100 Product Family Datasheet.* Intel, April 2015. URL `https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-phi-coprocessor-datasheet.pdf`. Version: April 2015.

[30] Christian Leber. *Efficient hardware for low latency applications.* PhD thesis, Universität Mannheim, 2012.

[31] Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu. *SIGARCH Comput. Archit. News*, 38(3):451–460, June 2010. ISSN 0163-5964. doi: 10.1145/1816038.1816021. URL `http://doi.acm.org/10.1145/1816038.1816021`.

[32] Linear Technology. *LTC4306 4-Channel, 2-Wire Bus Multiplexer with Capacitance Buffering*, .

[33] Linear Technology. *TCA6424 A Low-Voltage 24-Bit I 2 C and SMBus I/O Expander With Interrupt Output, Reset, and Configuration Registers*, .

[34] Martin Lingnau. Traversal algorithms for ray tracing – an architectural evaluation. Master's thesis, Heidelberg University, December 2017.

[35] Heiner Litz. *HyperTransport Advanced X-Bar (HTAX) Specification*. Computer Architecture Group - Heidelberg University. Version 0.14.

[36] Paulius Micikevicius. Multi-gpu programming. *GPU Computing Webinars, NVIDIA*, 2011.

[37] Gordon E Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

[38] S. Neuwirth, D. Frey, M. Nuessle, and U. Bruening. Scalable communication architecture for network-attached accelerators. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 627–638, 2015. doi: 10.1109/HPCA.2015.7056068.

[39] Sarah Neuwirth. *Accelerating Network Communication and I/O in Scientific High Performance Computing Environments*. PhD thesis, University of Heidelberg, Germany, 2019. URL `http://www.ub.uni-heidelberg.de/archiv/25757`.

[40] Tokyo Tech News. TSUBAME-KFC/DL supercomputer ranked No.2 in the world in Nov. 2015 edition of the energy efficiency Green500 List. `https://www.titech.ac.jp/english/news/2015/032904.html`, December 2015. URL `https://www.top500.org/`. Accessed: 2019-01-30.

[41] Northwest Logic. Northwest logic, 2020. URL `https://www.rambus.com/interface-ip/controllers/pci-express-controllers/`.

[42] M. Nüssle, B. Geib, H. Fröning, and U. Brüning. An fpga-based custom high performance interconnection network. In *2009 International Conference on Reconfigurable Computing and FPGAs*, pages 113–118, Dec 2009. doi: 10.1109/ReConFig.2009.23.

[43] M. Nussle, M. Scherer, and U. Bruning. A resource optimized remote-memory-access architecture for low-latency communication. In *2009 International Conference on Parallel Processing*, pages 220–227, Sept 2009. doi: 10.1109/ICPP.2009.62.

[44] *NVIDIA's Next Generation CUDA Compute Architecture: Kepler TM GK110*. NVIDIA, 2012.

[45] *NVIDIA® TESLA® GPU ACCELERATORS*. Nvidia, October 2013.

[46] L. Oden, H. Fröning, and F. Pfreundt. Infiniband-verbs on gpu: A case study of controlling an infiniband network device from the gpu. In *2014 IEEE International Parallel Distributed Processing Symposium Workshops*, pages 976–983, May 2014. doi: 10.1109/IPDPSW.2014.111.

[47] Lena Oden and Holger Froening. Ggas: Global gpu address spaces for efficient communication in heterogeneous clusters. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pages 1–8. IEEE, 2013.

[48] Tom Papatheodore. Summit system overview. URL `https://www.youtube.com/watch?v=_-z8ErBlBSo`. Accessed: 2019-01-28.

[49] David A Patterson. Latency lags bandwidth. In *ICCD*, pages 3–6, 2005.

[50] A. J. Pena and S. R. Alam. Evaluation of inter- and intra-node data transfer efficiencies between gpu devices and their impact on scalable applications. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 144–151, May 2013. doi: 10.1109/CCGrid.2013.15.

[51] John Pflueger and Sharon Hanson. Data center effciency in the scalable enterprise. *DELL Power Solutions*, 2007. URL `http://www.dell.com/downloads/global/power/ps1q07-20070210-CoverStory.pdf`.

[52] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D. K. Panda. Efficient inter-node mpi communication using gpudirect rdma for infiniband clusters with nvidia gpus. In *2013 42nd International Conference on Parallel Processing*, pages 80–89, Oct 2013. doi: 10.1109/ICPP.2013.17.

[53] C. Reaño, R. Mayo, E. S. Quintana-Ortí, F. Silla, J. Duato, and A. J. Peña. Influence of infiniband fdr on the performance of remote gpu virtualization. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–8, Sept 2013. doi: 10.1109/CLUSTER.2013.6702662.

[54] Jack Regula. Using non-transparent bridging in pci express systems. *PLX Technology, Inc*, 1, 2004.

[55] Sebastian Rinke, Daniel Becker, Thomas Lippert, Suraj Prabhakaran, Lidia Westphal, and Felix Wolf. A dynamic accelerator-cluster architecture. In *2012*

41st International Conference on Parallel Processing Workshops, pages 357–366. IEEE, 2012.

[56] Davide Rossetti. Benchmarking gpudirect rdma on modern server platforms, 2014.

[57] Gilad Shainer, Ali Ayoub, Pak Lui, Tong Liu, Michael Kagan, Christian R. Trott, Greg Scantlen, and Paul S. Crozier. The development of mellanox/nvidia gpudirect over infiniband—a new model for gpu to gpu communications. *Computer Science - Research and Development*, 26(3):267–273, Jun 2011. ISSN 1865-2042. doi: 10.1007/s00450-011-0157-1. URL https://doi.org/10.1007/s00450-011-0157-1.

[58] SuperMicro. *SUPERSERVER 4027GR-TR/TRT USER'S MANUAL*, October 2014. Revision 1.0a.

[59] SWEP. Refrigerant-handbook, 2020. URL https://www.swep.net/refrigerant-handbook/6.-evaporators/asas9/.

[60] The Ohio State University. Mvapich: Mpi over infiniband, omni-path, ethernet/iwarp, and roce, 2019. URL http://mvapich.cse.ohio-state.edu/.

[61] TOP500.org. Top500. https://www.top500.org/, September 2018. URL https://www.top500.org/. Accessed: 2018-09-05.

[62] Tyan. *DUAL-SOCKET 8GPGPU PLATFORM*. MITAC INTERNATIONAL CORP.

[63] Vicor. Dc-dc isolated regulated converters, 2018. URL http://www.vicorpower.com/dc-dc/isolated-regulated.

[64] Vicor Corporation. Gyoukou supercomputer leveraging 48v factorized power. URL https://www.youtube.com/watch?v=_-z8ErBlBSo. Accessed: 2019-01-28.

[65] Wieland-Werke AG — Thermal Solutions. *GEWA-C CONDENSER TUBES*. Version: 2018-09-10.

[66] WikiChip.org. WikiChip. https://en.wikichip.org/wiki/zettascaler, January 2019. URL https://www.top500.org/. Accessed: 2019-01-28.

[67] Wm A Wulf and Sally A McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.

[68] Dimitrios Ziakas, Allen Baum, Robert A. Maddox, and Robert J. Safranek. Intel®quickpath interconnect architectural features supporting scalable system architectures. In *Proceedings of the 2010 18th IEEE Symposium on High Performance Interconnects*, HOTI '10, pages 1–6, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4208-9. doi: 10.1109/HOTI.2010.24. URL `https://doi.org/10.1109/HOTI.2010.24`.