CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 0000; **00**:1–18
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

The DEEP Project
An alternative approach to heterogeneous cluster-computing in the many-core era

Norbert Eicker[1,2*] Thomas Lippert[1,2], Thomas Moschny[3], and Estela Suarez[1]
for the DEEP project

[1] *Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, D-52425 Jülich, Germany*
[2] *Bergische Universität Wuppertal, Fachbereich C, Gaußstr. 20, D-42119 Wuppertal, Germany*
[3] *ParTec Cluster Competence Center GmbH, D-81679 München, Germany*

SUMMARY

Homogeneous cluster architectures, which used to dominate high-performance computing (HPC), are challenged today by heterogeneous approaches utilizing accelerator or co-processor devices. The DEEP (Dynamical Exascale Entry Platform) project is implementing a novel architecture for High-Performance Computing, in which a standard HPC Cluster is directly connected to a so-called "Booster": a cluster of many-core processors. By these means heterogeneity is organized differently as in today's standard approach, where accelerators are added to each node of the Cluster. In order to adapt application codes to this Cluster-Booster architecture as seamless as possible, DEEP has developed a complete programming environment. It integrates the offloading functionality given by the MPI standard with an abstraction layer based on the task-based OmpSs programming paradigm. This paper presents the DEEP project with an emphasis on the DEEP programming environment. Copyright © 0000 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Cluster computing, Heterogeneity, Exascale

1. INTRODUCTION

From an architecture point of view cluster computers are dominating High-Performance Computing (HPC) today[†]. They leverage the increasing performance of commodity off-the-shelf components used in general computing in the field of HPC. Additionally, their modular setup allows optimizing the system configuration for a specific application portfolio. For instance, network topology, processor generation, and memory capacity can be chosen to fulfill the specific needs of a given user community. Even more a next generation processor might replace its predecessor without the need for simultaneously exchanging the interconnect technology, etc.

While multi-Petaflop (10^{15} floating-point operation per second (FLOPs)) systems are in production now, the next target in HPC is to achieve Exascale (10^{18} FLOPs) by the end of the decade. In face of the new challenges, the question arises whether cluster computers as we know them will be competitive in the future. The importance of heterogeneous clusters is increasing, as more and

*Correspondence to: E-mail: n.eicker@fz-juelich.de

Contract/grant sponsor: EC's Seventh Framework Programme (FP7/2007-2013); contract/grant number: 287530

[†]On the Top 500 list from November 2014, 86% of the systems are Clusters, against 14% systems with an Massively Parallel Processing architecture. [1]

more systems are populated with graphic cards or many-core co-processors to accelerate parts of the computation. Attaching one, two or more accelerator devices to each cluster node has become state of the art today. But making this approach efficient and scalable for HPC is very challenging due to the latency penalties and bandwidth limitations of the device-to-device communication originating in the node's PCIe bus shared by the accelerator and the host processor.

Two of the authors have proposed a novel heterogeneous cluster architecture [2], which aims at extracting the accelerating many-core processors out of the cluster nodes, creating an autonomous cluster of accelerators. This so-called "Booster" is attached to a conventional cluster of multi-core processors. A first realization of this Cluster-Booster architecture in hardware is currently installed and put into operation by the EU project DEEP [3]. The DEEP project has also implemented a corresponding programming environment, which strives for minimizing the effort needed to port existing MPI applications to the DEEP System.

This paper is organized as follows: Firstly, we motivate the Cluster-Booster architecture in view of the Exascale challenges and the scalability requirements evolving from them. In section 3 we give an overview on the DEEP project. The hardware realization of the Cluster-Booster architecture in the context of DEEP is presented in the next section. The main part of this work is dedicated to the description of the DEEP programming environment in section 5. It combines a heterogeneous global MPI [4] with the task-based OmpSs programming paradigm [5]. Finally, we conclude and give a brief outlook on the remaining work in DEEP.

2. MOTIVATION

The HPC community is currently preparing to move from today's multi-Petaflop compute systems[‡], to Exascale systems (10^{18} floating-point operations per second) by the end of the decade.

2.1. Exascale Challenges

Kogge et. al. showed that Exascale systems using updated versions of today's technology and concepts would run into severe troubles [6]:

- **Power consumption:** Projecting 2008's architectures onto end of the decade's technology predicts power requirements of several 100 MW for an Exascale system.
- **Resiliency:** The ever increasing number of components in HPC systems will continue to grow in the future. Projections allude to millions of devices in the Exascale era. Since mean time to failure (MTTF) of single components is not expected to be increased significantly, Exascale systems would become barely usable with system's MTTF in the range of hours or even minutes.
- **Memory hierarchies and I/O:** The rising gap between the compute performance and bandwidth of memory and storage will require additional layers in memory hierarchy like higher-level CPU caches or flash memory.
- **Concurrency:** With the levels of parallelism growing into the millions, application developers are challenged even harder to achieve performance. Additionally, as discussed in the next section, new requirements on the scalability of Exascale applications are introduced.

Altogether, this leads to the conclusion that radically new concepts need to be explored to attain the ambitious goal of Exascale in the proposed time frame. Obviously, also the concept of cluster computing has to be revisited. In particular, the core idea of clusters – the utilization of commodity CPUs – has to be reviewed on competitiveness against more proprietary developments for HPC.

[‡]Examples are Tianhe-2 at the National Super Computer Center in Guangzhou, Titan at the Oak Ridge National Lab, IBM Sequoia at LLNL, the K computer at RIKEN, or JSC's JUQUEEN.

DOI: 10.1002/cpe

The yardstick in HPC today was set down, on the one hand, by IBM's efforts leading to BlueGene/Q and, on the other hand, is challenged by the success of accelerators in the context of cluster computing. An analysis of results achieved on JSC's BlueGene/Q system JUQUEEN creates reasonable doubts whether commodity CPUs will be sufficient in the future. The design of commodity CPUs is focused on fulfilling their general purpose objectives, what occasionally comes at the price of limiting the efficiency of their floating-point abilities, a key capability in HPC. For typical workloads in scientific simulations, both BlueGene technology and GPGPU accelerators show superior energy efficiency and price vs. performance ratios due to their ability to very efficiently execute vectorized floating-point operations. In contrast to that, general purpose CPUs have their advantages in the field of single-thread performance.

Having commodity processors ruled out by this analysis, one has to strive for a new working horse in HPC. Good candidates are accelerators, which provide an order of magnitude higher floating-point performance compared to today's commodity CPUs, at the cost of limited flexibility. Even though the different incarnations of accelerators are versatile[§], they share some features leading to a superior energy efficiency compared to commodity CPUs. This includes the lack of complex hardware mechanisms for out-of-order operations and speculative execution. Instead, simultaneous multi-threading (SMT) is used to keep the execution units busy while waiting for data fetched from memory. Furthermore, wide vector registers are used for floating-point operations in order to increase the instruction-level parallelism (ILP). These technologies are leading to more lightweight cores compared to the ones in today's general purpose multi-core CPUs.

In fact, 15% of the top 500 systems in the current TOP500 list [1] are already equipped with accelerator cards – either GPGPUs or many-core co-processors – and the ratio raises to 50% when looking at the top 10 systems. However, the prevailing architecture of accelerated clusters[¶] suffers from severe limitations concerning programmability and balance. The latter is basically affected by the fact that while the cluster-nodes' compute performance is significantly enhanced by accelerators included in the nodes, both network and memory bandwidth basically stay constant. At the same time programmability suffers from several challenges, especially in the case of GPGPUs: In order to offload compute-kernels that show the required density of floating-point operations to the accelerator, their code has to be separated from the communication instructions required to realize the high level data-exchange in today's distributed memory architectures. In addition to that, most often accelerators require to port these kernels to different programming languages, since the dominating ones in HPC – C and Fortran – are only poorly supported on such devices – if at all.

A good way out of this dilemma would be creating a cluster of accelerators. In this concept, the node is consisting of an accelerator only – accompanied by some memory and a high-speed interconnect – skipping the commodity CPU. In particular, programs running on the accelerator cores shall be capable of initiating communication operations without the support of some commodity processor. A good example for this concept is the QPACE system [7].

However, this concept has limitations, too. First of all, very few accelerators are capable of acting autonomously and are flexible enough to efficiently drive a high-speed interconnect. Furthermore, the gain introduced by the direct connection between the compute element and the interconnect fabric might be wasted by the fact that accelerators suffer when running general purpose codes.

Thus, a radically new concept is required for cluster systems that shall be competitive at Exascale. Since it has to benefit from both processor worlds – general purpose multi-core CPU and many-core accelerators – we strive for a heterogeneous concept.

[§]Early examples are dedicated devices from ClearSpeed, ranging over processors originally developed for gaming like the Cell Broadband Engine and the aforementioned GPGPUs, towards newer developments like Intel's Xeon Phi.

[¶]We will distinguish between *accelerated clusters*, i.e. classical clusters comprising of nodes equipped with accelerators, and *clusters of accelerators*, i.e. clusters connected to each other in an own entity such as in the Booster concept introduced by DEEP.

DOI: 10.1002/cpe

2.2. Scalability Considerations

Having identified massive parallelism as a prerequisite for Exascale, a brief review of scalability serves as a good starting point for further discussion. Amdahl showed in his seminal paper [8] that the scalability of parallel programs is inherently limited by the sequential fraction of an application.

In practice, the effects of Amdahl's law are attenuated. Typical HPC use cases are better described by Gustafson's law [9]. While Amdahl assumed that using larger machines will leave the problem size untouched, in reality most often the problem is scaled according to the capabilities of the machine. This means: a computer twice as capable is not used for solving the original problem in half the time, but to tackle a problem twice as big in the same time or, alternatively, to increase the detail of a simulation requiring double the amount of operations.

Of course, Gustafson postulates the possibility of implementing the parallel portion of a program in a scalable way. Unfortunately, there are several caveats leading to the observation that the scalability of a parallel application might be significantly restricted in reality.

A different viewpoint to the question on how to reach Exascale might be taken by looking at actual applications and their inherent scalability. Analyzing JSC's application portfolio today one finds basically two classes of applications.

1. Highly scalable codes using very regular communication patterns. These are the codes that are able to exploit JSC's BlueGene systems.

2. Less scalable but significantly more complex programs. Most often their codes require complicated communication patterns. Such requirements constrain these applications to clusters today.

A more detailed view on the second class of applications reveals that several of them present highly scalable code parts, too. In principle these portions of the their code should also be able to exploit BlueGene-type of machines. However, in analogy to the serial work in Amdahl's law, their overall scalability is limited by the least scalable kernel.

To make things worse, the relative amount of applications of the second category in HPC application portfolios is expected to increase, since many codes are likely to shift from the first to the second class on the way from Petascale to Exascale. Two reasons for this are: *(i)* HPC simulation codes scaling well on Petascale machines might not do so on future Exascale systems due to the increasing degrees of parallelism; and *(ii)* codes get progressively more complex with the addition of new aspects of a given scientific question, generally limiting their scalability. In addition, problems completely out of range today due to their complexity might become feasible with the availability of Exascale systems. Their high degree of complexity makes them likely to belong to the second category as well.

Taking all this into account, enabling codes of the second category running on highly scalable systems is crucial for science to benefit from the computing power available at Exascale.

2.3. Cluster-Booster Architecture

The concept of the proposed architecture is sketched in figure 1. It foresees a Cluster element comprising of nodes (CN) connected by a highly flexible switched network. It is accompanied by a so-called Booster part built out of Booster nodes (BN). Each BN hosts an accelerator-type processor capable of autonomously booting and running its own operating system. The BNs are interconnected by a highly scalable torus network[‖] sketched in the figure as a mesh connecting the BNs. Booster Interfaces (BI) connect the Booster and its network to the Cluster fabric.

As discussed in more detail in [10], the Cluster-Booster architecture allows mapping applications onto the hardware according to their different scalability levels and provides several advantages:

[‖] In principle the Cluster-Booster approach doesn't put any constraints on the Booster network topology. Since the Booster is claimed to be scalable, its fabric must have a scalable topology like torus, butterfly, etc.

DOI: 10.1002/cpe

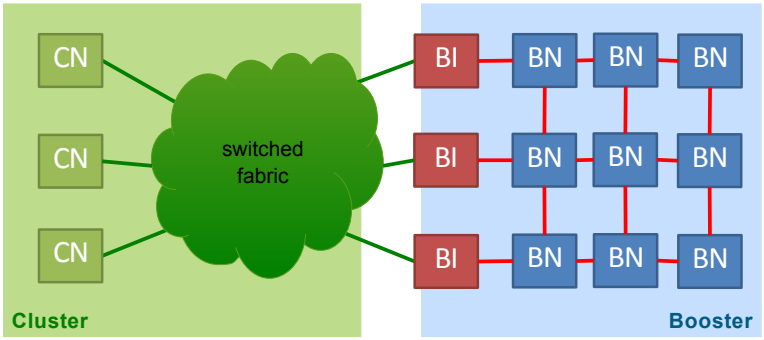

Figure 1. Sketch of the Cluster-Booster architecture.

- The Cluster element allows running complex and less scalable code parts. The limitations observed when running parallel kernels with complicated communication patterns or very irregular codes on today's highly scalable MPP machines are avoided.
- The Booster element is able to run highly scalable code parts in the most energy efficient way. The limitations of today's accelerated clusters are avoided by enabling the compute elements – i.e. the accelerators in the BNs – to do direct communication to remote BNs.
- Accelerators (i.e. BNs) might be assigned to CNs in a dynamical way. The ratio of the amount of work to be executed by the commodity CPUs and the accelerators is not expected to be fixed among different applications, or even between different kernels of a single application. The proposed architecture supports this fact by detaching the accelerators.
- The dynamical assignment of CNs and BNs improves resiliency: faulty BNs do not affect CNs and vice versa.
- Optimize the overall system usage: while in today's accelerated clusters applications using only the host processors block the use of the accelerators for other applications, in a Cluster-Booster system applications do only block the amount of BNs and CNs that they actively use, leaving all others free for the rest of applications.

Furthermore, the proposed architecture helps users to employ the high degree of parallelism of future machines. Today, the type of kernels to be offloaded onto accelerators is very limited due to their missing ability to efficiently exchange data with other accelerators. In contrast, the Cluster-Booster architecture allows more complex kernels to be offloaded to the Booster. These highly scalable code parts might include communication, as long as the corresponding communication patterns are regular enough not to swamp the Booster interconnect. In this context the Booster might be seen as a highly scalable system on its own. Thinking of the highly scalable codes that are able to exploit BlueGene or QPACE today, it should be possible to run many of them on the Booster alone.

On the other extreme BNs might be assigned to single CNs and used in the same fashion as in today's accelerated clusters. Both use cases – and anything in between – are possible without modification of the hardware concept. They can be implemented by just choosing a specific configuration on the level of system and application software.

3. DEEP PROJECT

DEEP (Dynamical Exascale Entry Platform) [3] is a three and a half year project partially funded by the European Commission (EC) through the Seventh Framework Programme. It joins sixteen partners from eight different European countries including public research centers, universities and

DOI: 10.1002/cpe

industry. With a total budget of more than 18 M€ – 8.03 M€ coming from the EC – DEEP aims for a proof of concept of the Cluster-Booster architecture.

DEEP does not only build the first prototype of this architecture, the so-called *DEEP System* as discussed in section 4, but has also implemented a complete software stack including programming environment, libraries and performance analysis tools as presented in section 5. Furthermore, the usability and performance of the final prototype will be evaluated using six scientific applications, representative for HPC at Exascale and coming from fields of great relevance for European science, industry, and society. The application fields and their supporting organizations in DEEP are:

- Brain simulation / Ecole Polytechnique Federale de Lausanne (EPFL)
- Space-weather / Katholieke Universiteit Leuven
- Climate simulation / The Cyprus Research and Educational Foundation (CyI)
- Computational fluid engineering / Centre Europeen de Recherche et de formation Avancee en calcul scientifique (CERFACS)
- High-T_c superconductivity / Consorzio Interuniversitario CINECA
- Seismic imaging / CGG Veritas

At the time of writing the DEEP project is in project month 38.

4. DEEP SYSTEM HARDWARE

In order to challenge the idea of the Cluster-Booster architecture, DEEP implements the first hardware realization of the concept. This section gives an overview of the main hardware systems used in the DEEP project: the Cluster, the Booster Interface and the Booster. In addition to that, we provide more details on the EXTOLL network. Finally, the so-called ASIC Evaluator is presented, an alternative realization of the Booster-part of the DEEP system based on an application-specific integrated circuit (ASIC) implementation of the EXTOLL network.

4.1. Cluster

The DEEP Cluster is a commercially available cluster utilizing Eurotech's AURORA line [11] of direct water-cooled blade-servers. It comprises 128 standard servers equipped with two Intel Xeon E5 2680 processors, each. They are interconnected by a Mellanox InfiniBand ConnectX fabric using QDR-generation technology.

To achieve a high energy efficiency the Cluster supports, on the one hand, power conversion in a centralized fashion by distributing only 48 V DC to each node and, on the other hand, direct water cooling enabling the system to be run with warm water. The CNs are mounted on aluminum coldplates connected to the chassis' water circuit through quick disconnects. The AURORA technology allows for an inlet temperature of the cooling liquid as high as 40°C. Contact with outside air through dry coolers is enough to recover this temperature for $> 99\%$ of the year. By this means the amount of energy spent for cooling is minimized. The same cooling technology is applied in the Booster.

4.2. Booster Interface

As sketched in figure 1, Booster Interface (BI) nodes bridge between the InfiniBand fabric of the Cluster and the EXTOLL interconnect of the Booster. Their physical realization are Booster Interface Cards (BICs). Each BI is equipped with an EXTOLL NIC and an InfiniBand HCA. Both are attached to an PLX PCIe switch allowing for high-bandwidth, low-latency PCIe communication within the BIC. This feature will be used for an efficient communication channel between CN and BN as discussed in section 5.1.5.

DOI: 10.1002/cpe

Figure 2. Booster during installation.

The physical realization of a BI combines a custom designed Booster Interface Card (BIC) with an Eurotech JUNO board, a full-blown Intel E3-12xx v3 server system in PCIe form factor [12]. The BIC is equipped with an EXTOLL NIC implemented in an Altera Stratix-V FPGA. The JUNO board's low-power Intel Xeon E3 CPU of the Haswell generation is used for mapping the memory of the 16 KNCs orchestrated by each BI into the PCIe address-space. This mapping is a key requirement to enable remote booting of the KNCs and efficient communication between the Booster and the Cluster part of the system.

4.3. Booster

An extensive description of the overall design and components of the DEEP Booster has been given in [13]. In this section we summarize the most important hardware aspects of the Booster and provide further implementation details of the actual realization.

The DEEP Booster is, as the Cluster, based on Eurotech's AURORA technology. The Booster Nodes (BNs) have a very lean design with one Intel Xeon Phi many-core processor and one EXTOLL NIC only, i.e. without an additional host processor attached**.

Physically, two BNs are integrated into a so-called Booster Node Card (BNC) forming the basic building block of the DEEP Booster. The first Intel Xeon Phi generation commercially available – code-named Knights Corner (KNC) – comes in different flavors. For the DEEP Booster, the model 7120X [14] has been chosen. It provides the largest memory capacity (16 GB) available in standard PCIe form factor. The latter is required for its integration in the Booster Node Card. The EXTOLL NIC in the BNCs has been implemented on FPGA basis. An Altera Stratix V [15] device has been chosen for this purpose.

A Booster chassis integrates 16 BNCs together with 2 BIs, providing them with power and direct water-cooling. In order to support AURORA's direct water cooling, both BIs and BNCs are mounted on coldplates connected to the chassis' water circuit. Each chassis houses a passive backplane supplying the internal EXTOLL links required for interconnection between the local BNCs, cable connectors for inter-chassis connectivity, Ethernet and serial lines for management and finally 48 V DC power. Within one rack copper cables are sufficient for inter-chassis connections.

As of today, a small part of the Booster, containing 32 BNs and 2 BIs, has been deployed. Figure 2 gives an impression of the hardware during installation. The full machine will consist of 384 Booster

**The KNC generation of Intel Xeon Phi processors – in contrast to future generations of this architecture – requires a host-processor for starting. Therefore, the BNs utilize the BI's processor for booting. Advanced features of the EXTOLL network allowing for remote access of KNC's memory and transparent forwarding of PCIe packets enable us to do so.

DOI: 10.1002/cpe

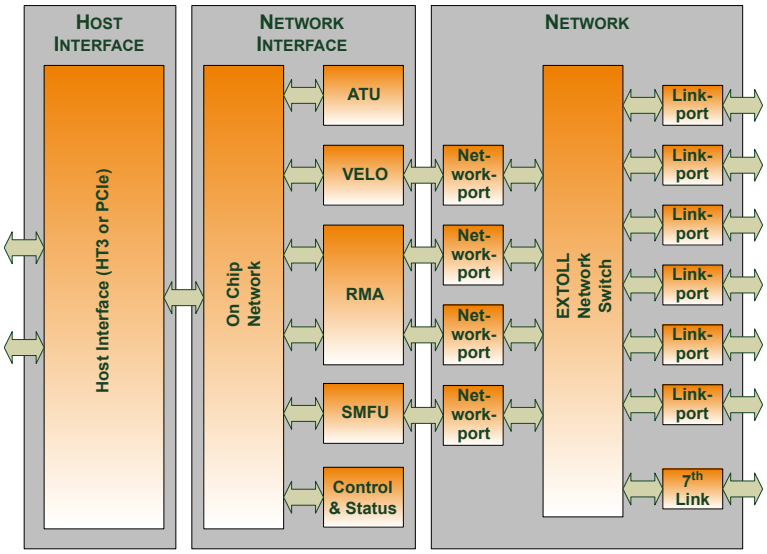

Figure 3. EXTOLL interconnect: Block diagram of the functional units.

Nodes and 24 BIs. Its maximum compute power is estimated to be about 464 TFlop/s for double precision operations[††] at a total power consumption of less than 150 kW. The total bi-sectional bandwidth of the Booster Interface – i.e. the bandwidth between the Cluster and Booster part of the machine – will be 768 Gbit/s.

4.4. EXTOLL

The interconnect implementation of the DEEP Booster uses EXTOLL technology developed at the University of Heidelberg. Each instance of a network node provides a total of seven links that might be connected to other instances. Six links are used to form the 3-D torus of the DEEP Booster fabric. Some of the 7th links will be used to connect the Booster torus with the BIs, and through them to the Cluster side of the DEEP System.

Within the lifetime of the DEEP project – but independently and outside of it – an ASIC implementation of EXTOLL has been developed. Initially, it was planned to build the Booster using this implementation but, unfortunately, delays in the ASIC availability have prevented this. Instead, the Booster is being built with an FPGA implementation of the EXTOLL protocol.

Compared to the final ASIC implementation, the FPGA provides full functionality with just less performance. The performance drawbacks include higher latency and limited bandwidth. The latter is due to a limited number of fast SERDES-links[‡‡] of FPGAs compared to an ASIC's capabilities in combination with the harder constraints on the number of logic-modules in FPGA confining the width of internal data-paths. This reduces the FPGA's link bandwidth to 4×4 Gbit/s compared to a target bandwidth of the ASIC of 12×10 Gbit/s. Latency is mainly affected by the fact that the FPGA implementation runs at significantly lower clock-speed (150 MHz vs. 750 MHz).

Independently of its flavor – FPGA or ASIC –, the EXTOLL protocol integrates a host interface, the network interface controller functionality and a router. The overall hardware architecture block diagram is shown in figure 3. In the Booster Nodes, the EXTOLL NIC is directly connected to the Intel Xeon Phi coprocessor via a PCIe link. EXTOLL is capable of acting as the PCIe root port in this case.

[††]The peak performance for double precision operations of a Each Xeon Phi 7120 is 1208 GFlop/s, with a Thermal Design Power (TDP) of 300 W.

[‡‡]Serializer-Deserializer modules multiplexing the wide data paths within the device into few high-speed signals for outside use

DOI: 10.1002/cpe

Figure 4. ASIC evaluator prototype

The NIC portion of EXTOLL offers three different network communication engines, all to be utilized by the DEEP software stack. The *very efficient low-overhead* (VELO) unit is responsible for low-latency, high-message rate, two-sided message-based communication [16]. The *remote memory access* (RMA) unit in cooperation with the ATU (address translation unit) implements one-sided communication primitives like *put* and *get* [17]. The *shared memory functional unit* (SMFU) implements a PGAS environment, actually a non cache-coherent view onto a global address space [18]. Within DEEP, the first two units are used to implement a most efficient MPI layer for the Booster. The SMFU is foreseen to realize the data movement of the bridging functionality between DEEP Cluster and Booster. Both communication protocols will be discussed in more detail in section 5. Beyond that, the SMFU is crucial for the ability of the BI's processor to remotely boot and control the KNCs.

4.5. ASIC Evaluator

A first version of the EXTOLL ASIC, with limited performance, already exists today. The final ASIC achieving full performance (12×10 Gbit/s per link), is expected to be available in Q2 of 2015. To evaluate the use of this high-speed network in a Booster-like system, a small prototype – the *ASIC Evaluator* (AE) – is being built in DEEP, too.

The AE will be different from the Booster described in section 4.3 not only in its network implementation, but also in its overall concepts of physical integration and cooling.

Similar to the DEEP Booster, the Booster Nodes of the ASIC Evaluator (AE-BNs) consist of a KNC and an EXTOLL NIC, too. However, here the NIC is implemented as an EXTOLL Tourmalet PCIe card [19] employing the EXTOLL ASIC. This standard PCIe form factor card is simply attached via standard PCIe connectors to an Intel Xeon Phi 7120D – a denser form factor implementation of the KNC. The EXTOLL NICs are interconnected to each other via copper cables that realize the 3-D torus topology of the AE. Groups of 8 KNC and 8 Tourmalet cards share the same backplane, which is mainly responsible for conducting the PCIe signals between KNC and NIC and for providing the necessary electrical power to both, KNC and Tourmalet. An AE-Chassis includes 4 of such dense backplanes, such that each chassis contains a total of 32 AE-nodes. The AE will consist of 64 KNCs, i.e. two fully populated AE-chassis.

Each AE-chassis is in fact a hermetically closed container, filled with the Novec [20] cooling liquid produced by 3M. All KNC and Tourmalet cards together with the backplanes and cables are immersed in this fluid that dissipates the heat produced by this components during operation. This is achieved by a phase-change of the cooling-agent from liquid to gas. The Novec liquid boils at 49°C and the convection generated through the component's heat and the phase-change assure that the warmth is transported to the upper part of the chassis basin. A water cooling serpentine is located here to cool down the Novec cooling agent back below its boiling-point temperature, i.e. to care for

DOI: 10.1002/cpe

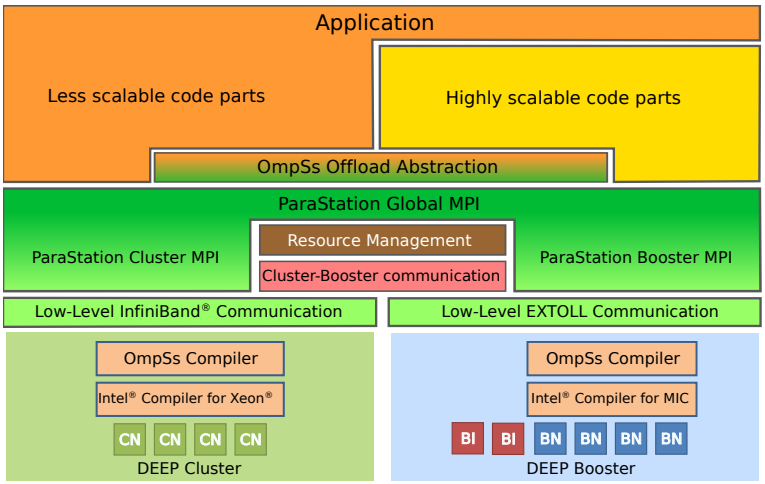

Figure 5. Sketch of the DEEP software architecture.

its condensation. Figure 4 shows an early prototype of the AE cooling concept with a dense form factor KNC in the front inside the boiling Novec cooling-agent.

The Booster Interface of the ASIC Evaluator (AE-BI) is realized by standard air-cooled Xeon servers, each equipped with a Tourmalet card and a Mellanox FDR InfiniBand adapter. The latter will be connected to the QDR InfiniBand fabric of the DEEP Cluster. The role of this Booster Interface here is fully equivalent to the one in the Booster: remote boot of the KNCs and driving the communication between the DEEP Cluster and the AE. Four AE-BIs are enough to support the full 64-KNC ASIC Evaluator, providing 1920 Gbit/sec bi-sectional bandwidth. Since the maximum compute power of the AE is just about 77 TFLOPs due to the lower number of KNCs included, the AE is expected to be significantly better balanced.

5. PROGRAMMING THE DEEP SYSTEM

Programming a heterogeneous system like DEEP is a challenging task for developers of HPC applications. In order to minimize the effort of porting existing applications to the Cluster-Booster architecture special emphasis was taken within the DEEP project to develop a programming model that supports the programmers as much as possible.

Figure 5 sketches the overall software architecture of the DEEP System. The heterogeneous layout of the machine is reflected by the fact that dedicated development and runtime environments supporting the distinct hardware feature of both parts are provided. While on the Cluster side a MPI library specifically optimized for the InfiniBand fabric is provided, on the Booster side the corresponding Booster-MPI supports EXTOLL. The latter will be used by the highly scalable code parts for intra-Booster communication. The choice of MPI supports the fact that the guiding applications of the DEEP Project are all based on the MPI programming paradigm.

The Booster Interface introduces constraints especially on the communication latency but also on bandwidth. For this reason, the DEEP programming model foresees to trench applications at a boundary involving only less frequent communication with limited data volume. By offloading highly scalable code parts including intra-Booster communication, collective operations shall for the most part be restricted to either the Cluster part or the Booster part of the applications.

5.1. *MPI Offloading*

The actual mechanism used for offloading of the highly scalable code parts to the Booster shall be as close to existing standards as possible. For this reason we have chosen to employ the dynamic process model of MPI-2, namely `MPI_Comm_spawn`, as the basis of DEEP's offloading mechanism.

DOI: 10.1002/cpe

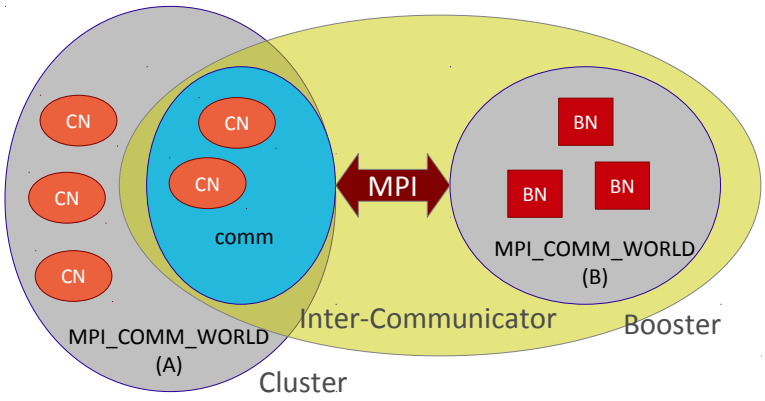

Figure 6. `MPI_Comm_spawn` schematics.

This choice provides the mechanism to start a group of new processes within the system that are capable to use MPI semantics to exchange data among themselves. Beyond that, the MPI standard also foresees to provide an efficient mechanism for exchanging data with MPI semantics between both groups of processes, i.e. between Cluster and Booster nodes in our case. Both parts of the applications – the part residing on the Cluster containing the `main()` function, and the offloaded part on the Booster – have their own `MPI_COMM_WORLD`s allowing to use full MPI functionality on either side. To send data back and forth between both groups of processes inter-communicators have to be used, see figure 6. It is worth to mention that *inverse offloading*, i.e. starting applications on the Booster and offloading part of them to the Cluster, is also supported by the actual implementation of the MPI offloading mechanism.

5.1.1. MPI_Comm_spawn details `MPI_Comm_spawn` is a collective operation performed by a subset of the processes of an application started on the Cluster. At least the name of the binary as well as the number of new processes to be started have to be specified in the call. A new inter-communicator is returned, providing a connection handle to the children. Each child has to call `MPI_Init`, as usual, and can use its `MPI_COMM_WORLD` communicator to exchange data with the sister processes started within the same call to `MPI_Comm_spawn`. In order to grab a handle on the inter-communicator shared with the parent processes, `MPI_Get_parent` has to be called.

The behavior of `MPI_Comm_spawn` can be further influenced using the `info` argument, passing a dictionary of string-based *key-value* pairs. The meaning of the keys is largely implementation-dependent, with some being predefined by the standard. An important key supported in the DEEP implementation is `arch`, specifying the node-architecture to be used in order to spawn new processes. The two obvious choices are `cluster` and `booster`.

We will also support the `soft` key foreseen by the MPI standard. By this means an application can instruct MPI to determine the number of processes to be created depending on the amount of resources currently available. This dynamic resource management promises to optimize the overall resource usage within the DEEP system by providing applications the ability to occupy unused resources. Of course, a close integration with the batch system is required in order to prevent applications from "stealing" resources from other applications expected to start soon. A detailed discussion of the resulting implications on the batch system can be found in [21].

5.1.2. Inter-communicators An inter-communicator, as defined by the MPI standard, contains two groups of processes and naturally allows point-to-point communication between a member of one group and a member of the other group.

Starting with MPI-2, collective operations have been extended and defined for inter-communicators. In the taxonomy of collective operations there are three classes: The first contains all *all-to-one* and *one-to-all* operations: For these, the data transfer is always uni-directional between the two groups of an inter-communicator. A gather operation e.g. will collect data from all members

DOI: 10.1002/cpe

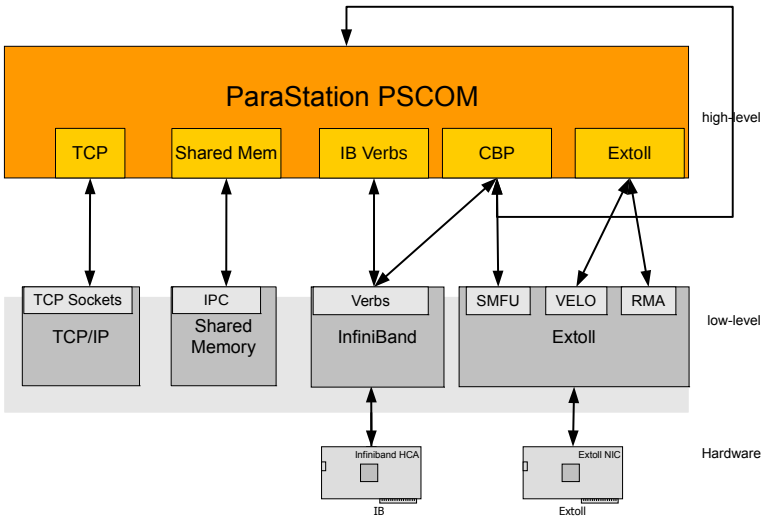

Figure 7. Architecture of the ParaStation communication library `pscom`.

of the sending group and pass it to one process of the receiving group. The second class contains *all-to-all* operations, which for inter-communicators are split into two phases. In each phase, data is collected in one group and the result is sent to all members of the other group. Finally, a third class contains the collective operations which are not allowed on inter-communicators, like `MPI_Scan`.

5.1.3. Global MPI implementation As sketched in figure 5, DEEP's programming model is based on MPI for intra-Booster as well as for intra-Cluster communications. Furthermore, the offloading mechanism uses the dynamic process model defined in MPI-2.

The two MPIs together with the offloading mechanism and Cluster-Booster communication actually form a global MPI: an MPI implementation that is usable on all node types and allows for communication within CNs, within BNs, and between the Cluster and the Booster parts of the system.

DEEP's global MPI implementation uses ParTec's ParaStation MPI [22], which in turn is based on MPIch, and relies on *pscom*, its own communication library. *pscom* supports various network interconnects for inter-node communication and shared memory for intra-node data transfers as sketched in figure 7. The library is extensible by plug-ins to support several interconnects or protocols at the same time. Communication is connection-oriented, the actual communication path to use is chosen automatically at runtime while the connection is established. This happens independently for each connection with an adjustable priority system to find the best available path for each combination of processes.

Supporting InfiniBand and thus intra-Cluster communication out of the box, new plugins for EXTOLL and its VELO and RMA protocols (driving the intra-Booster communication) on the one hand, and for the so-called *Cluster-Booster protocol* on the other hand (both described below), have been developed. This way, the global MPI implementation covers all possible communication paths within the DEEP architecture.

5.1.4. Intra-Booster communication A key requirement to run highly scalable code parts on the Booster is a most efficient implementation of the MPI standard for the Booster hardware, i.e. the KNC multi-core processor and the EXTOLL interconnect. Major effort was invested in order to achieve this goal. The main obstacles we unveiled had their seeds in performance restrictions in the KNC processor when using it in a naive way. To give an example, it turned out that in general the `memcpy` operation is rather expensive on this platform. Therefore, it becomes crucial to strive for a zero-copy implementation of the communication operations in the Booster MPI.

DOI: 10.1002/cpe

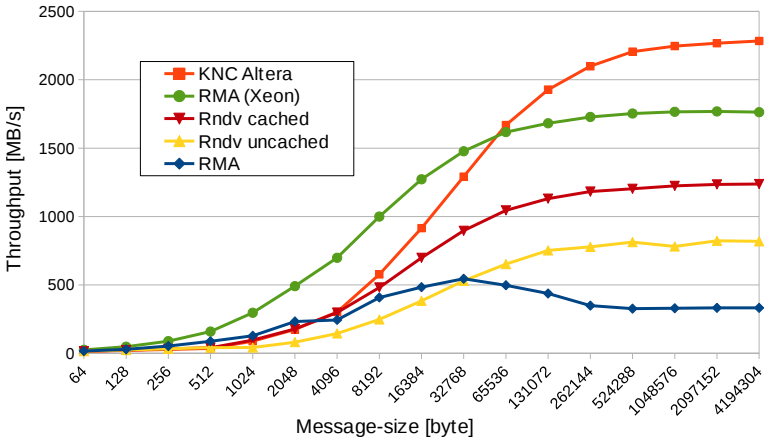

Figure 8. Bandwidth comparison of the Booster MPI against other platforms using similar implementations of the EXTOLL interconnect.

After early experiments with the combination of KNC and EXTOLL had unveiled the mentioned problems, it became clear that a zero-copy implementation of the communication layer will become crucial for the scalability of the Booster part of the DEEP system. Figure 8 show this findings. Comparing the lines for RMA and RMA (Xeon) it becomes obvious that communication bandwidth suffers significantly from the poor `memcpy` performance on KNC while the bandwidth observed on a standard Xeon platform is basically identical to the low-level hardware capabilities of the FPGA implementation of the EXTOLL protocol.

Fortunately, EXTOLL provides a low-latency message-based protocol for small data: VELO. Thus, a combination of VELO and RMA for remote DMA communication is used to implement a rendezvous based zero-copy protocol for EXTOLL in *pscom*. While small messages, i.e. messages smaller than a rendezvous size of 1024 bytes, are sent directly via VELO, the rendezvous for large messages works as follows:
Sender:

1. Register the data buffer of the message for RMA usage. For the EXTOLL stack this means to pin down all memory pages containing message data, preventing these pages from being swapped out to disk or being moved to a different location in physical memory. After that, map the physical addresses of all affected pages to a consecutive range of EXTOLLs network logical addresses (NLA).
2. Transmit the metadata of the message together with the NLA of the message data to the receiving side.

Receiver:

3. Identify the destination buffer of a receive request matching the message metadata.
4. Register the destination buffer for RMA usage (do pinning and assign an NLA).
5. Initiate an RMA `get` operation from the remote source buffer to the local destination buffer. The RMA Get will address the source and destination regions with the NLAs from step 1) and step 4). A completion notification when the RMA is done is requested at the same time.
6. After completion of the RMA operation:
   - Notify the application about the new message.
   - Unregister the receive buffer.
   - Send an acknowledge control message via VELO back to the sender.

DOI: 10.1002/cpe

Sender:

7. When receiving the acknowledge message:
    - Unregister the send buffer.
    - Notify the application about the completed send. It is now safe for the application to reuse the send buffer.

The resulting bandwidth numbers are marked as "Rndv uncached" in figure 8. Obviously the result is only partially satisfactory. In fact it turns out that registration and de-registration of data buffers are expensive operations on KNC, too. Therefore a caching layer for buffer registration was introduced resulting in the bandwidth numbers annotated as "Rndv cached" in the figure. Still a significant bandwidth gap compared to the numbers on the Xeon platforms is observed. Further analysis unveiled that it originates to a PCIe switch that was required in the early evaluation platform that was used to achieve these numbers. Actually, moving to the final hardware employing a slightly more capable FPGA to implement the EXTOLL network provides the "KNC Altera" tagged results. These show for large messages an overall bandwidth near to the physical limits of the utilized EXTOLL implementation.

The remaining differences to the Xeon platform, i.e. the shift of raising bandwidth towards larger message sizes, stems from larger latency observed on KNC. While for the RMA protocol on the Xeon platform the half round-trip time is 4.5 μs, the rendezvous protocol on KNC achieves just 5.8 μs even though we expect reduced latency for the rendezvous. In fact, we reach a half round-trip time of 2.9 μs for the rendezvous protocol on Xeon. As a result KNC is latency-wise worse by a factor of two compared to Xeon. This most probably originates from the inferior single thread performance of KNC due to its in-order architecture in combination with the slower processor clock compared to Xeon. Therefore, the execution of the critical path of the communication protocol will take longer on the KNC platform. Nevertheless, as soon as messages are large enough in order to relieve the protocol from the dominance of management overhead, KNC is doing well on intra-Booster communication.

5.1.5. Cluster-Booster protocol Since both parts of the DEEP System – Cluster and Booster – will have their own interconnect adapted to the requirements of the specific application code parts, a network bridge between both parts is implemented in the Booster Interface, enabling all CNs to make use of whole partitions of BNs in order to offload highly scalable code parts to the Booster.

For an efficient use of the offloading mechanism a high-throughput, low-overhead network protocol is crucial. The implementation of this protocol features the SMFU functionality of the EXTOLL network realizing significant benefits compared to a standard store-and-forward implementation. A detailed discussion of the design and implementation of this protocol will be provided in [23].

As sketched in figure 7, the Cluster-Booster protocol is integrated into *pscom* as a plugin, too. In fact, this plugin is special, since it does not only provide a new communication layer to *pscom* but at the same time makes use by the library itself for connection instantiation.

5.1.6. Resource Management integration In order to be usable for the DEEP architecture, a resource management system has to be able to handle its heterogeneity, i.e. needs to know about Cluster and Booster nodes (CNs and BNs). While the resource allocation mechanism for the CNs is not different from that used on other clusters, modifications are necessary for the additional *static* and *dynamic* allocation of BNs for a parallel job. In general, BNs are allocated exclusively for a job. In the DEEP system, we use a modified version of Torque [24] together with the Maui job scheduler [25].

In the *static* case, BNs are allocated before the job is started and remain reserved until the job is terminated. Torque has been extended to accept requests for BNs together with CNs. Such a request can be placed by the user e.g. via a modified version of the `qsub` command:

```
qsub -l nodes=cn:cluster+bn:booster <<<"mpiexec -np n <exec>"
```


DOI: 10.1002/cpe

where cn and bn denote the number of Cluster and Booster nodes, respectively. For applications doing *direct offloading* $n \leq cn$ will hold as they are initially started solely on CNs, and can issue `MPI_Comm_spawn` calls to run their highly scalable code parts on the Booster, as described above. Torque will pass lists of all allocated resources to ParaStation's process management system.

If inverse offloading is required for a distinct application, this is also supported by the DEEP resource management. In fact, only the resource request in the `qsub` command has to be adapted such that Booster resources appear in front of the Cluster resources. This implies that the Job will start on the Booster part of the system and can later on spawn additional processes on the Cluster part of the DEEP system.

DEEP also foresees *dynamic* resource allocation. This feature is triggered by an application calling `MPI_Comm_spawn` requesting more Booster nodes than previously allocated statically. Per default the call would fail, as described earlier, but it will be possible to tell the DEEP runtime that it should try to dynamically allocate more resources by the `soft` key as discussed above. The MPI will forward such requests to Torque. However, depending on the current load of the machine it may block the application until enough Booster nodes are available or an optionally specified timeout expires and the call finally fails.

5.2. OmpSs offload abstraction

Extending an existing application to make use of the DEEP offloading is cumbersome and error-prone. Therefore, DEEP has extended the OmpSs [5] data flow programming model developed by the Barcelona Supercomputing Center to ease application porting to heterogeneous machines like the DEEP system.

5.2.1. OmpSs task model Based on OpenMP and belonging to the StarSs [26, 27] family, OmpSs exploits task-level parallelism and supports asynchronicity, heterogeneity and data movement.

OmpSs was originally developed to support programming on standard SMP machines populated with GPUs, what was becoming an increasing tendency in HPC environments. It allowed including tasks written in CUDA or OpenCL to execute them on the graphics cards. Nowadays, OmpSs supports as well other kinds of hybrid architectures such as SMP machines populated with Intel Xeon Phi. For DEEP, OmpSs has been extended to feature not only both sides of the DEEP System, but also to support offloading large complex tasks from Cluster to Booster, or vice-versa.

To use OmpSs [28] an application must be *taskified*. This is done by annotating the code with OpenMP-like pragmas that indicate data dependencies between the different tasks of the program. OmpSs pragmas are written in front of a function or piece of code that shall be designated as a task and typically look as follows:

```
#pragma omp task input(a[i-1]) inout (a[i]) output(b[i])
        function(&a[i-1], &a[i], &b[i]);
```

where the clauses `input`, `output` and `inout` specify which data is needed and which is produced by a task. Additionally, a `target` clause allows the user to specify one or a series of hardware devices where a given task should be executed, and if data needs to be copied from/to those devices. Various versions of the tasks can exist to target different architectures.

The OmpSs annotations are interpreted by the OmpSs source-to-source compiler – so-called *Mercurium* –, which supports Fortran, C, and C++ languages. For each call to the annotated functions the compiler generates a call to the OmpSs runtime system – so-called *Nanos++* – to create a new task. The result is compiled by a native compiler and linked with *Nanos++*.

Each time a new task is created its input and output dependencies are matched against those of the already existing tasks. Taking these dependencies into account, the runtime decides on the order of the tasks and whether concurrent execution is allowed, creating a task dependency graph at run-time. All this information is used to schedule the tasks on the available devices.

DOI: 10.1002/cpe

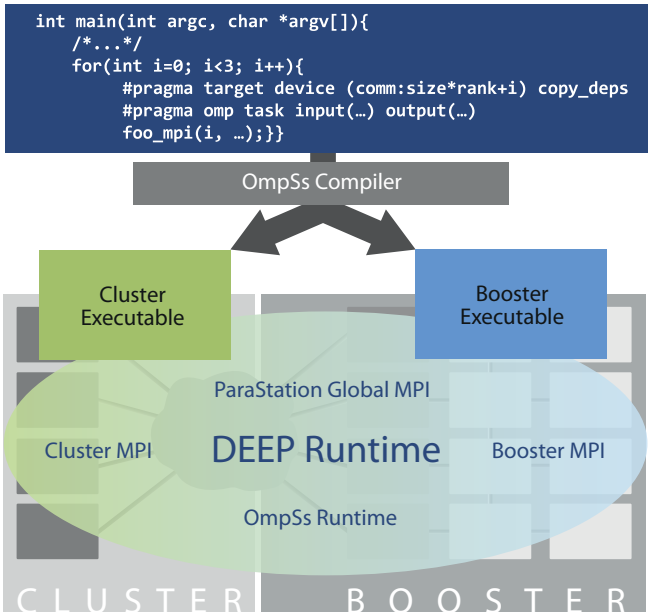

Figure 9. DEEP application workflow: From code to execution.

5.2.2. The DEEP offload In DEEP, the OmpSs programming model runs not only at the node level, but also as an abstraction of the global MPI as sketched in figure 5. Directly using the `MPI_Comm_spawn` primitive means for the programmer to coordinate and manage two or more sets of parallel MPI processes, explicitly sending the required data from one side to the other of the DEEP System. This would make the port of large and complex applications the Cluster-Booster architecture very cumbersome. For this reason, OmpSs has been extended to implement the so-called *DEEP Offload* [29]. It combines the power and flexibility of `MPI_Comm_spawn` with the OmpSs data flow model. The result is an offload mechanism that is similar to the Intel Offload but also allows offloading MPI kernels, i.e. within the offloaded code part it supports direct communication between MPI processes running on different Xeon Phis.

The overall workflow conducting from application code to execution on the DEEP System is sketched in figure 9. Using OmpSs pragmas the application developer labels the highly scalable parts of its code, which shall run on the Booster. The compiler and the runtime of OmpSs cooperate to transparently manage all data transfers between the MPI processes running on the Cluster and the Booster, making use of functions like `MPI_Comm_spawn` and `MPI_Comm_send`.

It is important to mention that the highly-scalable code parts running on the Booster typically contain internal MPI operations. These operations are also orchestrated by the OmpSs runtime. Support is guaranteed for any offloading pattern that can be expressed with the low-level MPI spawning mechanism.

The DEEP Offload has been implemented and is already validated for large C, C++ and Fortran applications. Tests on various large production machines have been performed to demonstrate the scalability of the concept. It will be installed and tested on the DEEP System, allowing for both static and dynamic allocation of its resources.

6. CONCLUSION AND OUTLOOK

DEEP implements a first incarnation of the heterogeneous Cluster-Booster architecture. It aims for pursuing the successful concept of cluster computing into the many-core era, carrying the potential

DOI: 10.1002/cpe

to reach Exascale. We expect this target to be barely reachable with standard HPC clusters as they are in use today.

At the time of writing, the DEEP project was in project month 38 and had already achieved several important milestones: the Cluster part and a 32 node subsection of the Booster part of the DEEP System are up and running at the Jülich Supercomputing Centre; most of the components of the ASIC Evaluator have been already constructed and just wait for the availability of the final EXTOLL ASIC to get the system up and running; the low-level Cluster-Booster Protocol has been implemented and is being tested on the DEEP System; ParaStation MPI supports EXTOLL and has been ported to Intel Xeon Phi; OmpSs has been extended to create the DEEP Offload; the performance analysis tools Scalasca and Extrae/Paraver are running on Xeon Phi, and the latter already supports the DEEP Offload; the application codes have been analyzed and re-structured to implement the code division between Cluster and Booster, several of them already including the OmpSs pragmas, etc.

The remaining time of the project will be used to produce and install the rest of the components needed to complete the 384 node Booster, to finalize the ASIC Evaluator and install both at JSC. The software infrastructure as described in this paper is almost final and currently under installation on the existing small-size DEEP System. Minor bugs and errors are being corrected to arrive at a fully stable software stack. As soon as this step is ready, the DEEP's guiding applications will run on the machine and serve as a yardstick for assessing the Cluster-Booster architecture.

ACKNOWLEDGEMENTS

The authors would like to thank the people and partners involved in the DEEP consortium for their ongoing engagement and strong commitment towards the project, which led to several of the results described in this paper. Special gratitude goes to M. Nüssle from University of Heidelberg for helpful discussion on EXTOLL and V. Beltran of Barcelona Supercomputing Centre for valuable input on OmpSs.

The research leading to these results has been conducted in the frame of the DEEP (Dynamical Exascale Entry Platform) project, which receives funding from the European Commission's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement n° 287530.

REFERENCES

1. TOP500 project. TOP500 list. URL http://www.top500.org.
2. Eicker N, Lippert T. An accelerated Cluster-Architecture for the Exascale. *PARS '11, PARS-Mitteilungen*, vol. 28, Gesellschaft für Informatik e.V., Parallel-Algorithmen und Rechnerstrukturen, 2011; 110 – 119.
3. DEEP project. The DEEP project 2011. URL http://www.deep-project.eu.
4. MPI Forum. MPI: A Message-Passing Interface Standard. Version 2.2 September 4th 2009. Available at: http://www.mpi-forum.org (Dec. 2009).
5. Duran A, Ayguadé E, Badia RM, Labarta J, Martinell L, Martorell X, Planas J. OmpSs: A proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters* 2011; **21**(02):173–193, doi:10.1142/S0129626411000151. URL http://www.worldscientific.com/doi/abs/10.1142/S0129626411000151.
6. Peter Kogge et al. Exascale Computing Study: Technology Challenges in Achieving Exascale Systems 2008. URL http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf.
7. Baier Hea. QPACE: power-efficient parallel architecture based on IBM PowerXCell 8i. *Computer Science - Research and Development* 2010; **25**(3-4):149–154, doi:10.1007/s00450-010-0122-4. URL http://dx.doi.org/10.1007/s00450-010-0122-4.
8. Amdahl GM. Validity of the single processor approach to achieving large scale computing capabilities. *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), ACM: New York, NY, USA, 1967; 483–485, doi:10.1145/1465482.1465560. URL http://doi.acm.org/10.1145/1465482.1465560.
9. Gustafson JL. Reevaluating Amdahl's law. *Commun. ACM* May 1988; **31**(5):532–533, doi:10.1145/42411.42415. URL http://doi.acm.org/10.1145/42411.42415.
10. Mallon DA, Eicker N, Innocenti ME, Lapenta G, Lippert T, Suarez E. On the scalability of the Clusters-Booster concept: a critical assessment of the DEEP architecture. *Proceedings of the Future HPC Systems: the Challenges of Power-Constrained Performance*, FutureHPC '12, ACM: New York, NY, USA, 2012; 3:1–3:10, doi:10.1145/2322156.2322159. URL http://doi.acm.org/10.1145/2322156.2322159.
11. Eurotech SpA. Aurora HPC systems. URL http://www.eurotech.com/en/hpc/hpc+solutions/aurora+hpc+systems.
12. Eurotech SpA. HiVE JUNO. URL http://www.eurotech.com/en/hpc/hpc+solutions/aurora+hive+series/Aurora+HiVe.
13. Eicker N, Lippert T, Moschny T, Suarez E. The DEEP project. Pursuing cluster-computing in the many-core era. *In Proceedings of the 42nd International Conference on Parallel Processing Workshops (ICPPW) 2013. Workshop on*

DOI: 10.1002/cpe

Heterogeneous and Unconventional Cluster Architectures and Applications (HUCAA), Lyon, France, 2013; 885–892, doi:10.1109/ICPP.2013.105.
14. Intel Corporation. Intel Xeon Phi 7120 2012. URL http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html.
15. Altera. Stratix V. URL www.altera.com/devices/fpga/stratix-fpgas/stratix-v.
16. Litz H, Fröning H, Nüssle M, Brüning U. VELO: A Novel Communication Engine for Ultra-Low Latency Message Transfers. *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, 2008; 238–245, doi:10.1109/ICPP.2008.85.
17. Nussle M, Scherer M, Bruning U. A resource optimized remote-memory-access architecture for low-latency communication. *Proceedings of the 2009 International Conference on Parallel Processing*, ICPP '09, IEEE Computer Society: Washington, DC, USA, 2009; 220–227, doi:10.1109/ICPP.2009.62. URL http://dx.doi.org/10.1109/ICPP.2009.62.
18. Fröning H, Litz H. Efficient Hardware Support for the Partitioned Global Address Space. 10^{th} *Workshop on Communication Architecture for Clusters (CAC2010), colocated with* 24^{th} *International Parallel and Distributed Processing Symposium (IPDPS2010)*, Atlanta, Georgia, USA, 2012.
19. EXTOLL GmbH. Tourmalet. URL http://www.extoll.de/index.php/productsoverview/tourmalet.
20. 3M. Novec. URL http://multimedia.3m.com/mws/media/569865O/3mtm-novectm-649-engineered-fluid.pdf?&fn=Novec649_6003926.pdf.
21. Prabhakaran S, Iqbal M, Rinke S, Windisch C, Wolf F. A batch system with fair scheduling for evolving applications. *Proc. of the* 43^{rd} *International Conference on Parallel Processing (ICPP)*, Minneapolis, USA, 2014; 351–360, doi:10.1109/ICPP.2014.44.
22. ParTec GmbH. ParaStationV5 2012. URL http://www.par-tec.com/products/parastationv5.html.
23. Galonska A, Eicker N, Hauke J, Nüssle M. Bridging the DEEP gap – implementation of an efficient forwarding protocol in preparation.
24. Adaptive Computing. TORQUE Resource Manager. URL http://www.adaptivecomputing.com/products/open-source/torque/.
25. Adaptive Computing. Maui. URL http://www.adaptivecomputing.com/products/open-source/maui/.
26. Ayguadé E, Badia R, Igual F, Labarta J, Mayo R, Quintana-Ortí E. An Extension of the StarSs Programming Model for Platforms with Multiple GPUs. *Euro-Par 2009 Parallel Processing*, *Lecture Notes in Computer Science*, vol. 5704, Sips H, Epema D, Lin HX (eds.). Springer Berlin Heidelberg, 2009; 851–862, doi:10.1007/978-3-642-03869-3_79. URL http://dx.doi.org/10.1007/978-3-642-03869-3_79.
27. Perez J, Badia R, Labarta J. A dependency-aware task-based programming environment for multi-core architectures. *2008 IEEE International Conference on Cluster Computing*, 2008; 142–151, doi:10.1109/CLUSTR.2008.4663765.
28. Pogramming Models @ BSC. The OmpSs Programming Model 2013. URL https://pm.bsc.es/ompss.
29. Labarta J, Beltran V. Deliverable 5.3 of the DEEP project: OmpSs runtime for the DEEP System feb 2014. URL http://www.deep-project.eu/SharedDocs/Downloads/DEEP-PROJECT/EN/Deliverables/deliverable-D5.3.pdf.