SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

FP7-ICT-2011-7

DEEP

Dynamical Exascale Entry Platform

Grant Agreement Number: 287530

D4.5
Cluster Booster protocol ported to final hardware

Approved

Version: 2.0
Author(s): A. Galonska (JUELICH), N. Eicker (JUELICH), M. Nüssle (UniHD), J. Hauke (ParTec)
Date: 24.02.2014

Project and Deliverable Information Sheet

DEEP Project	**Project Ref. №:** 287530	
	Project Title: Dynamical Exascale Entry Platform	
	Project Web Site: http://www.deep-project.eu	
	Deliverable ID: D4.5	
	Deliverable Nature: Report	
	Deliverable Level: PU *	**Contractual Date of Delivery:** 30.11.2013
		Actual Date of Delivery: 30.11.2013
	EC Project Officer: Luis Carlos Busquets Pérez	

* - The dissemination level are indicated as follows: **PU** – Public, **PP** – Restricted to other participants (including the Commission Services), **RE** – Restricted to a group specified by the consortium (including the Commission Services). **CO** – Confidential, only for members of the consortium (including the Commission Services).

Document Control Sheet

Document	**Title:** Cluster Booster protocol ported to final hardware	
	ID: D4.5	
	Version: 2.0	**Status:** Approved
	Available at: http://www.deep-project.eu	
	Software Tool: Microsoft Word	
	File(s):DEEP_D4.5_Cluster_Booster_protocol_ported_to_final_hardware_v2.0-ECapproved.docx	
Authorship	**Written by:**	A. Galonska (JUELICH), N. Eicker (JUELICH), M. Nüssle (UniHD), J. Hauke (ParTec)
	Contributors:	
	Reviewed by:	E.Suarez (JUELICH), F.Affinito (CINECA)
	Approved by:	BoP/PMT

Document Status Sheet

Version	Date	Status	Comments
1.0	30/November/2013	Final	Ready for EC submission
2.0	24/February/2014	Approved	Approved by EC

Document Keywords

Keywords:	DEEP, HPC, Exascale, Cluster Booster protocol, Global MPI

Table of Contents

List of Figures

Executive Summary

The interconnects of the two parts of the DEEP System – InfiniBand (IB) in the Cluster and EXTOLL in the Booster – adapt to the requirements of the specific application code-parts. A network bridge between both parts, the Booster Interface (BI), enables all Cluster Nodes (CN) to make use of whole partitions of Booster Nodes (BN) when highly scalable code-parts are offloaded to the Booster.

A key aspect in the design of the proposed network architecture is a high throughput network protocol bridging the Cluster and Booster fabrics. Its purpose is enabling communication between Cluster and Booster Nodes and allowing offloading application's highly scalable code-parts with low overhead from Cluster to Booster and exchanging their respective data and results in both directions. In order to physically connect both fabrics the Booster Interface Card (BIC) as the realisation of the BI in hardware is equipped with two PCIe ×16 slots attached to a PLX PCIe switch. These slots house an InfiniBand host channel adapter (HCA) and an EXTOLL network interface card (NIC), respectively. The PLX switch allows for PCIe peer-to-peer communication between both devices. To avoid unnecessary utilisation of the BIC's processor, data transfers are based on a combination of Remote Direct Memory Access (RDMA) techniques provided by both networks.

Communication between Cluster and Booster will be divided into two kinds of channels: control channels and data channels. Control channels are used to set-up a data channels based on RDMA between a CN and a BN via the BI. A dedicated Booster Interface Daemon (BID), together with the appropriate client software running on the CNs and BNs orchestrates the Cluster-Booster protocol.

This document describes the adaptation of the full Cluster-Booster protocol (CBP) – as it is foreseen for the DEEP System and described in D4.2 – to the Super-BIC evaluator. The hardware and functionality of this prototype is similar to the final hardware, which is not yet available at this phase of the project.

This report addresses developers of high-level communication functionality in the context of the DEEP project. Members of task 4.4 implementing the Booster resource-management as well as the programmers of the OmpSs off-load abstraction layer shall take this document as a basis for their own work in order to understand the underlying functionality and restrictions.

1 Introduction

The DEEP project develops a next generation heterogeneous HPC cluster integrating multi- and many-core processors in an innovative way. While today's default way to implement heterogeneity is to equip Cluster Nodes (CN) with accelerators to off-load suitable application kernels, the DEEP approach foresees a separate cluster of accelerators named Booster. By this means a group of accelerators each residing in a Booster Node (BN) can be utilised collectively by a group of Cluster Nodes. Cluster and Booster have different interconnects. Therefore, a bridge is required – the Booster Interface (BI) – in order to connect both fabrics and to allow for communication between both networks. The actual data transfer will be realised by the low-level Cluster Booster protocol (CBP).

The implementation and optimisation of the Cluster-Booster protocol is a central component to obtain a well performing DEEP System. The aim of this development is to gain high transfer rates with low overhead for the communication between the Cluster and the Booster. In the Cluster part of the system a switched InfiniBand fabric [1] with fat-tree topology allows to use complex communication patterns as required by the less-scalable software parts of the applications. The Booster interconnect is based on EXTOLL technology [2] implementing a highly scalable 3D-torus topology accommodating the requirements of the highly-scalable code-part. Both networks are joined in the Booster Interface, where an InfiniBand HCA and an EXTOLL NIC are connected via PCIe ×16 links. In the final system each BI will serve 8 Booster Node Card (BNC) as a bridge to the Cluster fabric. Each BNC will house 2 Intel Xeon Phi many-core processors of the Knights Corner (KNC) generation. The BI in DEEP is physically realised as a so called Booster Interface Card (BIC), built from the minimal hardware required to establish a connection between both networks via PCIe. Due to the fat-tree topology of the InfiniBand Cluster fabric the BICs are accessible from all CNs at comparable bandwidth and latency.

For porting and testing purposes the BIC evaluator (as described in D4.2) was extended with two additional servers. This updated so-called Super-BIC evaluator consists now of five standard x86 servers representing one CN, one BI and three BNs. The two additional BNs in the Super-BIC evaluator are similar to the original BN but equipped with an extra Intel Xeon Phi (named MIC along this document) many-core processor each. From a functional point of view this setup is almost identical to the final hardware. Therefore the software stack tested on the Super-BIC evaluator is expected to be moved easily to the final DEEP System.

Starting with a description of the development environment this document explains the adaptation of the CBP to match the requirements of the final hardware. Several steps in order to port the protocol are explained. Tests were conducted to evaluate the performance of the protocol on the Intel MIC platform. The implementation of a Global MPI spanning Cluster and Booster is described here with respect to the integration of the CBP into ParaStation MPI. Since memory is a scarce resource on the BNs, finally, a strategy to reduce the memory footprint of the protocol is discussed.

2 Development Environment – The Super-BIC evaluator

The development vehicle used for this task is an enhancement of the BIC evaluator as described in D4.2. It now consists of five standard servers equipped with Intel Xeon CPUs of the Westmere (CN, BI & BN1) and Sandy Bridge (BN2 & BN3) generation. This Super-BIC evaluator was employed to carry out all modifications, enhancements and tests of the CBP reported in this document. Its hardware was chosen to satisfy the need for a porting environment. At the same time it minimizes the delays introduces by the lack of existing

Booster Interface Cards or Booster Node Cards equipped with Intel MIC processors. Both are currently still under test by task 3.2.

The five machines of the Super-BIC evaluator used for porting the CBP are representing a CN, a BI and three BNs, respectively. The machine configuration is sketched in Figure 1 .

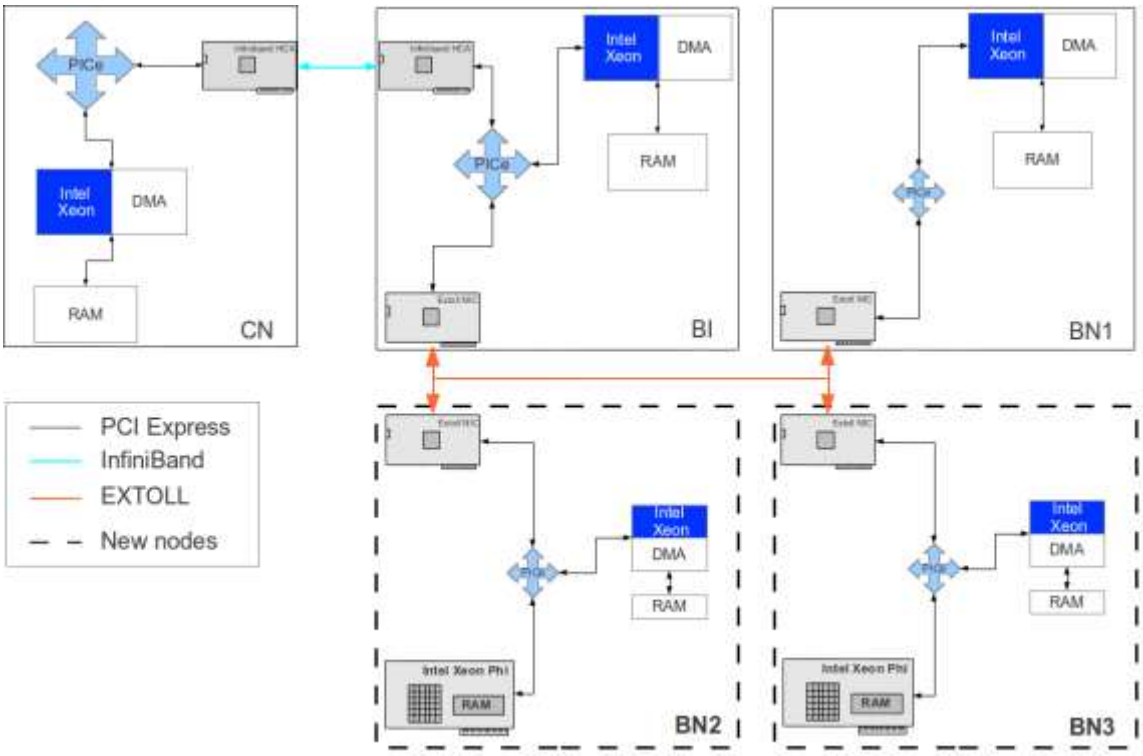

Figure 1: Architecture of the Super-BIC evaluator system

The server representing the CN is solely equipped with a Mellanox ConnectX3 InfiniBand HCA, connected to the server representing the BI. The BI server houses an identical Mellanox ConnectX3 InfiniBand HCA and an FPGA-based EXTOLL NIC code-named „Galibier“. Both are PCIe ×8 cards connected via a highly configurable PLX 8624 24-lane PCIe gen2 switch. It allows configuring up to 6 ports with variable line widths. This switch is intended to bridge both networks physically, controlled by a special software running as a daemon on the BI. Finally the FPGA-based EXTOLL NIC is connected to its identical counterparts on the other servers (BN1, BN2 & BN3). While BN1 was lacking MIC support, the newly installed BNs BN2 and BN3 are both equipped with an Intel Xeon Phi Card. Due to the similar configuration of the setup of CN, BI and BN2 and BN3 compared to the final hardware of the DEEP System, this evaluator allows extensive functional studies of the CBP software.

It is important to note that the current peak bandwidth between two EXTOLL NICs is limited to < 1 GB/s. This is due to the fact that the NICs used in the Super-BIC evaluator are implemented by means of an FPGA, in contrast to the ASIC version planned for the DEEP System. This will be the main bottleneck for the bandwidth results observed in our tests since both, PCIe gen2 at about 3.2 GB/s, and the IB HCA at more than 4 GB/s show significantly higher throughput.

3 Global MPI

The software architecture of DEEP foresees a Global MPI spanned over the whole system (Figure 2). On the Cluster, low-level communication is done via InfiniBand while on the Booster the EXTOLL communication engines VELO and RMA come into play.

In between Cluster and Booster the Cluster-Booster communication is done by means of the CBP. All low-level communications are managed by the ParaStation pscom library described in the next subsection. This library serves as the communication engine of ParaStation MPI, spanning the Global MPI over both parts, and thereby over the whole DEEP System. The Global MPI can be used standalone by the user application or in conjunction with an OmpSs offload abstraction. The OmpSs abstraction itself also utilises the Global MPI for these purposes.

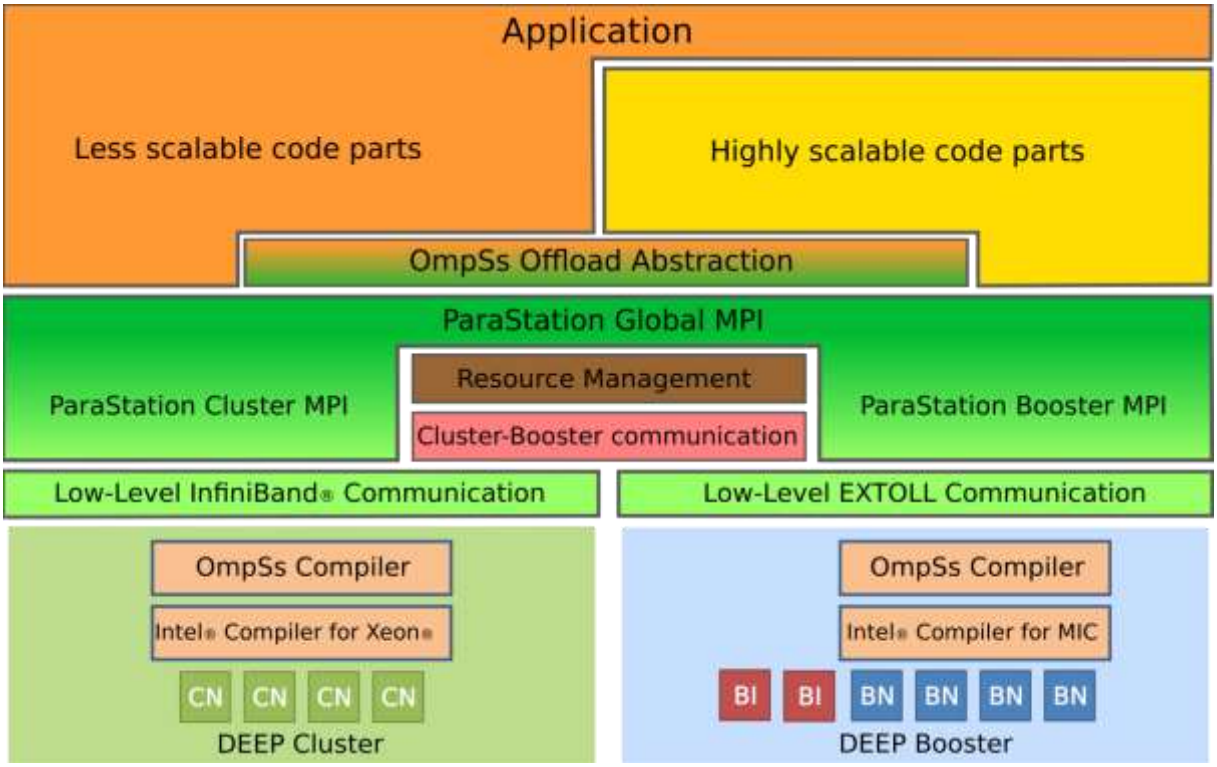

Figure 2: DEEP software architecture

3.1 ParaStation pscom

ParaStation's pscom library [3] implements an API for inter-process communication employed by ParTec's implementation of the MPI standard [4]: ParaStation MPI. As sketched in Figure 3, it supports various network interconnects for inter-node communication and shared memory for intra-node data transfers. 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 mechanism will act independently for each connection, with an adjustable priority system to find the best available path for each connection.

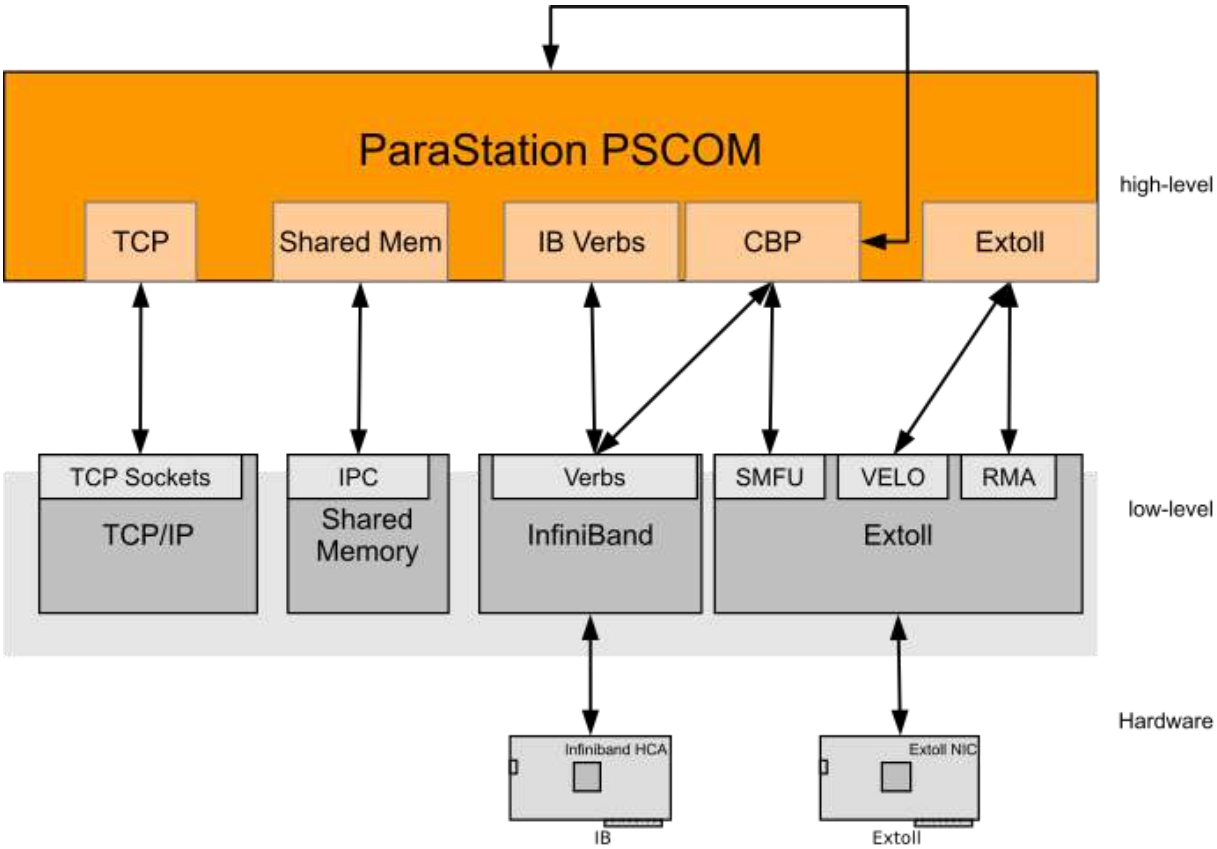

Figure 3: Schematic view on ParaStation pscom with the CBP as an additional plug-in

Initially designed as a network abstraction layer for the ParaStation MPI implementation, the pscom library might also be used stand-alone. It provides synchronous and asynchronous point-to-point message delivery with send/receive semantic, single-sided RDMA with put/get, and collective operations on groups of connections.
In the context of this work pscom is used in a two ways: On the one hand it is employed by the CBP as a transport layer for protocol metadata (connection of CN and BN to the BID, remote calls, etc.) within the control channel. For this, between CN and BI the InfiniBand verbs layer is used, between BI and BN EXTOLL's VELO or RMA protocols are utilised. On the other hand the CBP will also be part of the pscom library serving within an additional plug-in. This plug-in provides the functionality of direct routes between a CN and a BN using the CBP. From pscom's point of view, this is an additional network protocol. In this context the CBP plug-in has to avoid ending up in a dead-lock when simultaneously using pscom and providing it with a new network protocol. For DEEP, this plug-in provides an important functionality to pscom in order to span a Global MPI over Cluster and Booster of the DEEP System.

3.2 Pscom's CBP plug-in

The CBP plug-in of pscom (Figure 3) aims to implement transparent data transfers between Cluster and Booster suitable to be used as a low-level transport protocol underneath MPI. It provides the following functionalities to the pscom library:

- Mapping of the EXTOLL Shared Memory (SMFU) and initialisation of the InfiniBand verbs layer for the actual connection.
- Connection to another process
- Accept connections from another process

- Look-up for new data
- Send data to another process
- Close a CBP connection

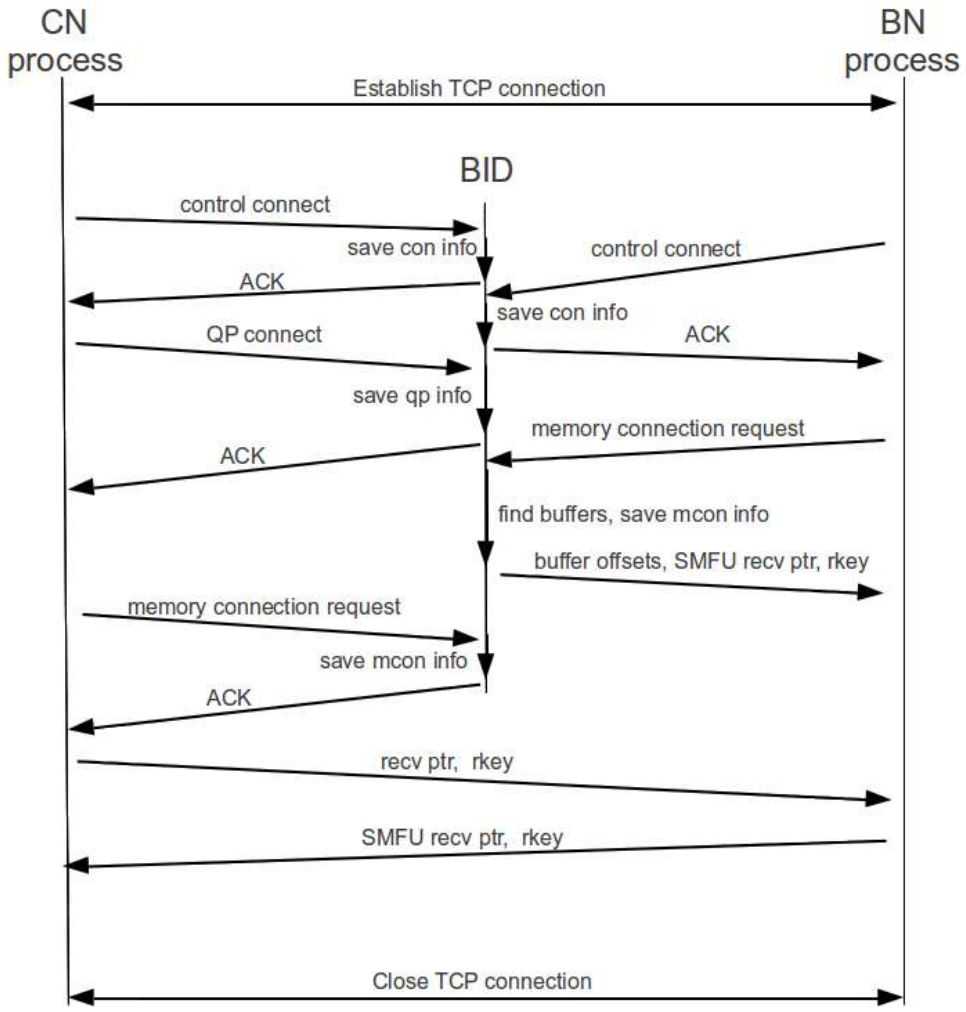

Figure 4: Connection start-up diagramm

The connection start-up had to be slightly modified in order to match the pscom requirements. Connection start-ups have to be two-sided. Our protocol also needs an additional connection to the BI. First of all a TCP connection between the two peers is established. Then every peer creates a control and a data connection to the BI. On the InfiniBand side a QueuePair (QP) has to be created, on the EXTOLL side buffers within the SMFU region have to be determined. These buffers are used by the hybrid data connection and can be accessed via IB on the BI. A more detailed description of this mechanism was provided in D4.2.
All information on these data connections is then exchanged by the peers in order to establish the hybrid data connection.

4 Extensions of the protocol

4.1 Introduction

As the protocol itself was already described in D4.2, only extensions will be explained in detail within this document.

These extensions are necessary to fulfill the requirements of the final DEEP hardware. The final systems will have multiple BNs and CNs and also several BIs (Figure 5). Each set of BNs can be accesses through a certain BI. On each of the compute nodes (i.e. CNs and BNs) multiple MPI processes might be running communicating amongst each other. Therefore the protocol has to deal with different data paths between the processes.

Furthermore the Booster architecture will not be based on Intel Xeon processors but on Intel Xeon Phi many-core processors, which comprise an in-order architecture with a slightly different application binary interface (ABI) and offering a lot more of compute cores. Therefore the code has to be adapted and compilation as well as the resulting binary will be different on this architecture.

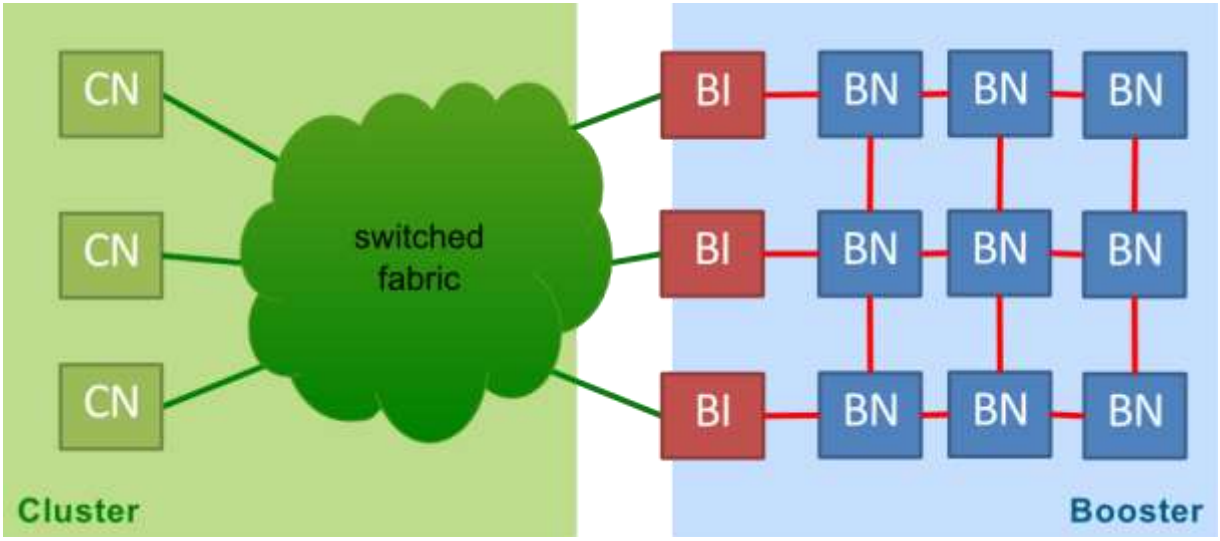

Figure 5: DEEP hardware architecture

4.2 Routing

At first glance a static routing at start-up time is foreseen for the protocol. Each BN will be accessed by the CNs through a certain BI. This information is stored in a routing table which is read initially by the protocol on both kinds of nodes. This makes every BN aware of the BI assigned to it and enables the CN to determine the data and control path to use in order to access a certain BN.

The routing table one an entry for every BN determining the associated BI in the following manner:

< BN IP > : < BI IP >

This information is used during connection start-up in order to determine the relevant BI to be used by both peers.

4.3 Multiple-Daemons

Since DEEP is a designated multi-job and multi-user system, it was decided to start on a per job basis one BI daemon (BID) on each BI involved in a certain job. Of course, a job might utilise multiple BIs for communication between Cluster and Booster and therefore will then start multiple BIDs. Furthermore BIs serve multiple BNs and thus might be shared between different jobs. In the latter case a single BI might run more than one BID. This approach has the advantage that different jobs do not need to share a single daemon on a shared BI, what would blow up the time spent for connection management. In addition, error recovery of the

daemon is much simpler because it will then be restricted to a single job and not multiple ones, reducing the impact of a restart.

To achieve the goal of enabling multiple daemons to listen for incoming connections on a certain BI, each daemon will listen to its own port. The information on this specific port of each daemon has to be propagated to the associated processes in order to enable them to make use of the BID's services. This will be done within the ParaStation process management developed for DEEP in Task 4.2.

4.4 Reducing memory footprint

The amount of available memory per core in a KNC (about 256 MB) is quite limited. For this reason, the amount of memory bound to the communication buffers of each connection is a major burden for the memory footprints of the Booster processes. The buffers are organised in queues, one serving as a send queue the other acting as a receive queue. In the original design each connection had its own queues. To improve this, a send queue shared among all connections of one KNC process was implemented within the protocol – the Shared Send Queue (SSQ). With this approach, the memory footprint within the KNC memory exported by the SMFU is reduced roughly by a factor of two because only receive buffers are allocated on a per connection basis. This effectively leads to significantly reduced memory footprint on a huge number of connections.

To evaluate the impact of this enhanced implementation on the bandwidth, several tests were conducted on the Super-BIC evaluator (with CN, BI and BN1). For this the ParaStation Global MPI as described in section 3 was utilised. In order to be able to compare results to the ones reported in D4.2, only the Xeon machines (i.e. CN, BI and BN1) were used. In our implementation the Shared Send Queue, as described above, only affects connections of processes running on the BN side. Using a shared send queue on the communication endpoints residing on the Cluster side of the system is beyond the scope of this work. To take this fact into account, the following scenario was created: one single MPI process running on the BN side talking to 1 to 8 MPI processes running on the CN.

This scenario leads to an increasing number of connections on the BN side, while every single CN process has to manage only one connection. Our MPI-Ping-Pong benchmark is started on the BN side using the non-blocking MPI_ISend and MPI_IRecv calls. In the send step the BN process initiates send transfers via MPI_ISend to every participating CN process and calls MPI_WaitAll to wait for the completion of the request. Every CN process calls the corresponding MPI_Recv to accept the data. In the receive step the same is done via MPI_IRecv followed by MPI_WaitAll on the BN and MPI_Send on the CN side. In our benchmark the PingPong-time is being measured on the BN side considering all MPI_ISends + MPI_WaitAll as a Ping and all MPI_IRecvs + MPI_WaitAll as a Pong. The bandwidth results presented here are based on the mean-value of both leading to the half round-trip time.

Due to the dispatch architecture of pscom and the CBP, all pseudo-simultaneous transfers of the MPI_ISend and MPI_IRecv calls will lead to teethed transfers of the atomic payloads consisting of just one buffer of the CBP.

The benchmarks were conducted using a varying number of CN processes as well as different sizes of the send queues. The original Private Send Queue (PSQ) and new Shared Send Queue (SSQ) approaches were compared. Only the most interesting results were picked out of a large number of tests driven (56) to identify the limitations of the SSQ.

Figure 6 depicts the benchmark results testing the PSQ with 256 kB buffers per connection. The data transfer rates shown are cumulative over all connections of the BN process. The blue line shows that a single connection between the BN and one CN process is not enough to

saturate the links on the Super-BIC evaluator (~878 MB/s). The best performance is achieved when using connections to two or more CN processes from a single BN process saturating the EXTOLL link at ~917 MB/s.

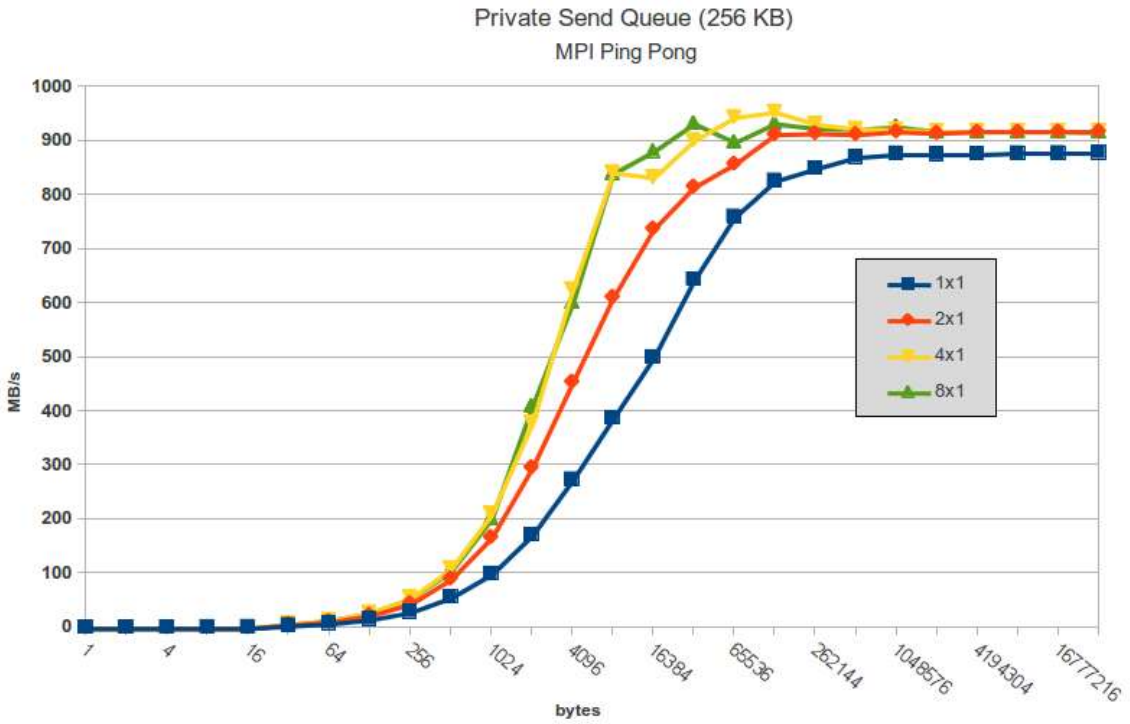

Figure 6: (1 .. 8) × 256 kB Private Send Queue communication with a single BN process and 1 to 8 CN processes. The abscissa shows message size, the ordinate shows the cumulative data transfer rate of all connections at a certain message size.

Performance impact of using the SSQ is only expected to manifest when using a higher number of connections. In Figure 7 the outcome of tests employing a single BN process talking simultaneously to 8 CN processes is depicted. The Bordeaux red line shows that using a small SSQ of just 128 kB in conjunction with many connections leads to a performance degradation in peak bandwidth at around 884 MB/s. A SSQ of 256 kB or larger already restores the accumulated bandwidth observed for the PSQ with its total memory footprint of 2048 kB.

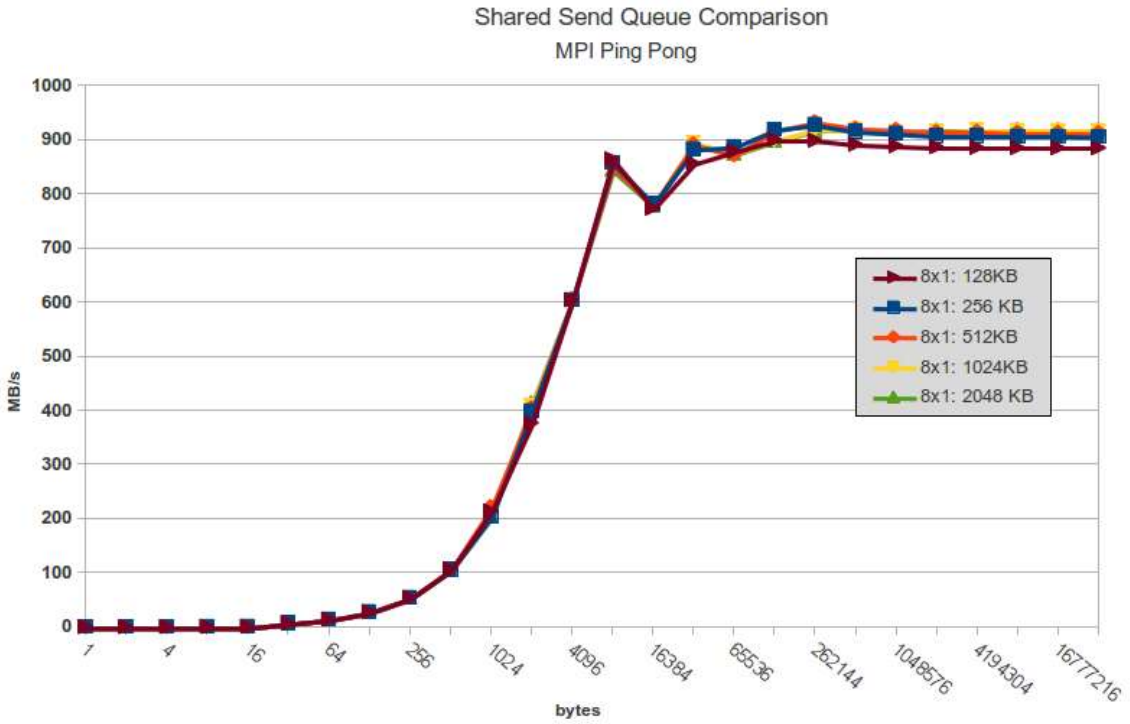

Figure 7: (128 .. 2048) kB Shared Send Queue communication involving 1 BN process and 8 CN processes. The abscissa shows the message size, the ordinate gives the cumulative data transfer rate at a certain message size.

To allow for a direct comparison, Figure 8 presents the usage of a small 128 kB SSQ compared to a PSQ approach using the regular size of 256 kB. Both experiments used 8 connections leading to a total buffer size of 2048 kB in the case of the PSQ. This result emphasizes the impact of an insufficient number of send buffers available to saturate the link. The SSQ (blue line) in this scenario reaches only ~96 % of peak bandwidth compared to the PSQ (red line).

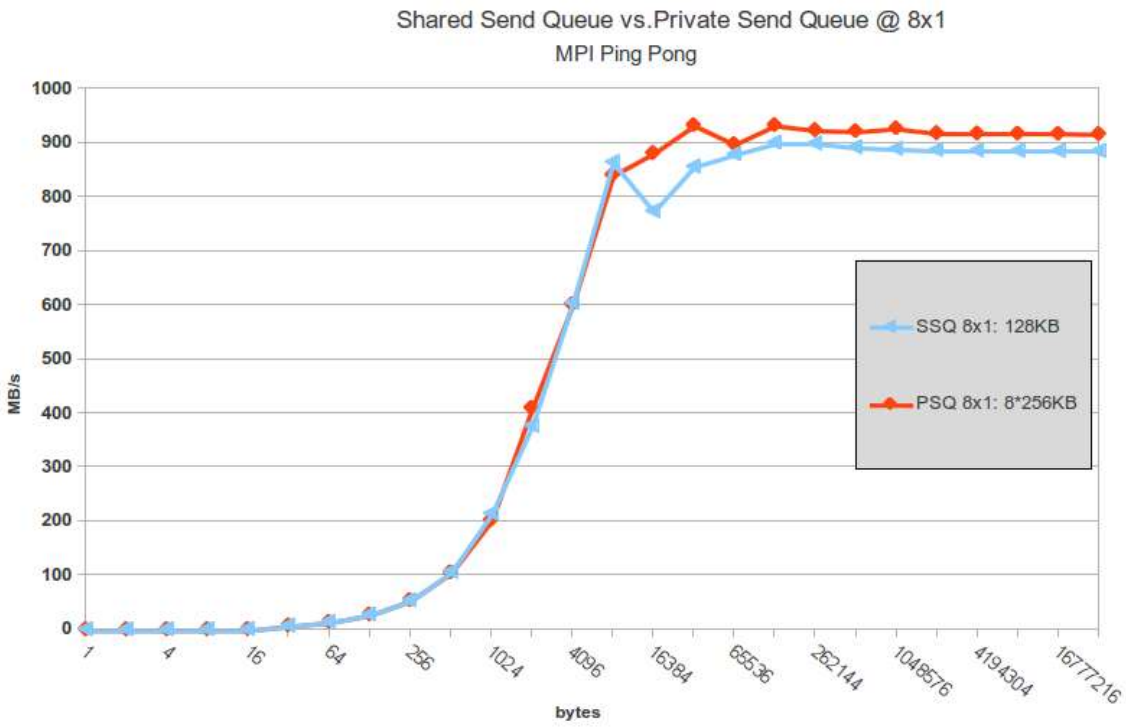

Figure 8: 128 kB SSQ vs (8 × 256) kB PSQ communication using 1 BN process and 8 CN processes. The X-axis shows the message size, the Y-axis shows the cumulative data transfer rate at a certain message size.

It can be foreseen that the impact will be even bigger on a larger number of connections. Such scenarios could not be tested in our small testing environment on the Super-BIC evaluator. Since the overall bandwidth in all our experiments is limited by the relatively slow FPGA-implementation of the EXTOLL interconnect, all results are preliminary. We plan to reinvestigate these benchmarks as soon as the ASIC-implementation is available. Nevertheless, to save the scarce memory resources on KNC this approach will be taken into account when using the CBP on larger systems as well as on the final hardware.

The implementation of a shared send queue is rather trivial. However, much more effort has to be spent in the design and implementation of a shared receive queue. This is due to the need of acknowledging the sending side on the next usable receive-buffer to every associated connection. This will induce additional latency to the protocol and thereby degrade performance. Nevertheless, even though this approach is not foreseen in the current implementation, it will be investigated in the future.

5 Porting to KNC MIC

In order to actually port the CBP to MIC several steps have to be taken. First of all, since KNC will only directly connect to EXTOLL NICs, all code parts related to InfiniBand are excluded by pre-processor macros. This disables the use of InfiniBand libraries and header files during the compilation and link steps when compiling for MIC. Otherwise it would have been necessary to port the OFED stack to MIC, too. Beyond that, no further changes were necessary to port the CBP to the MIC platform.

In a first step the existing code-base was re-compiled with the two compilers available on the MIC platform, *gcc* and *icc*. During evaluation it turned out that binaries created by *icc* achieved about double the bandwidth compared to *gcc*-compiled binaries. Unless explicitely

stated all results presented in the next section are created by means of the Intel *icc* compiler version 13.1.3.

In a last step the SMFU driver for the EXTOLL card had to be adapted. This was necessary as we now need read/write access to KNC's memory, instead of accessing the memory of the host. Furthermore the memory mapping with respect to SMFU on the KNC is slightly different compared to the host. While on the host only single pages had to be mapped to the BI's PCIe address-space, on KNC the whole memory can be mapped as once. This simplifies the setup of the communication buffers significantly.

6 Performance Evaluation

For several reasons we have to expect a performance impact to CBP's bandwidth when moving one communication endpoint to the MIC. First of all, the single-thread performance of the KNC is expected to be significantly lower than on the Intel Xeon. The main reasons for that are a lower clock-rate of KNC and the fact that KNC is an in-order architecture. Secondly, the actual behaviour of KNC's DMA engine while interacting with the EXTOLL NIC has to be investigated. Furthermore, the PCIe switch required in BN2 and BN3 in order to attach the additional MIC-card will add some latency. To get an idea on the actual performance impact, several tests were conducted on the Super-BIC evaluator using the adapted protocol.

For the final prototype similar performance numbers are expected. This is due to the fact that the Proto-Booster will have a very similar hardware setup than the Super-BIC evaluator, omitting only the host processor and the PCIe switch in the BNs. Nevertheless, for the final hardware of the DEEP System a significant increase of communication bandwidth is expected due to the use of the EXTOLL ASIC.

All presented benchmarks were carried out using pscom_pp. This low-level ping-pong benchmark uses the pscom routines directly without going through the MPI layer sitting on top of pscom in the DEEP software architecture. Every test was repeated 1000 times. The numbers presented are the average of the timing measurements. Message sizes are ranging from 1 byte to 4 MB.

6.1 RDMA MIC-to-MIC

In order to get a reference for the amount of bandwidth that might be achieved by the CBP, data transfers employing EXTOLL RMA directly between two MICs were investigated. The tests were carried out between the two MICs residing in BN2 and BN3. Both are using the EXTOLL interface of their host machines via PCIe, without putting any load on the respective host CPUs. As shown in detail in D4.1 and D4.2, we expect to get similar performance from RMA and SMFU, since both are remote memory access techniques working in an analogous manner.

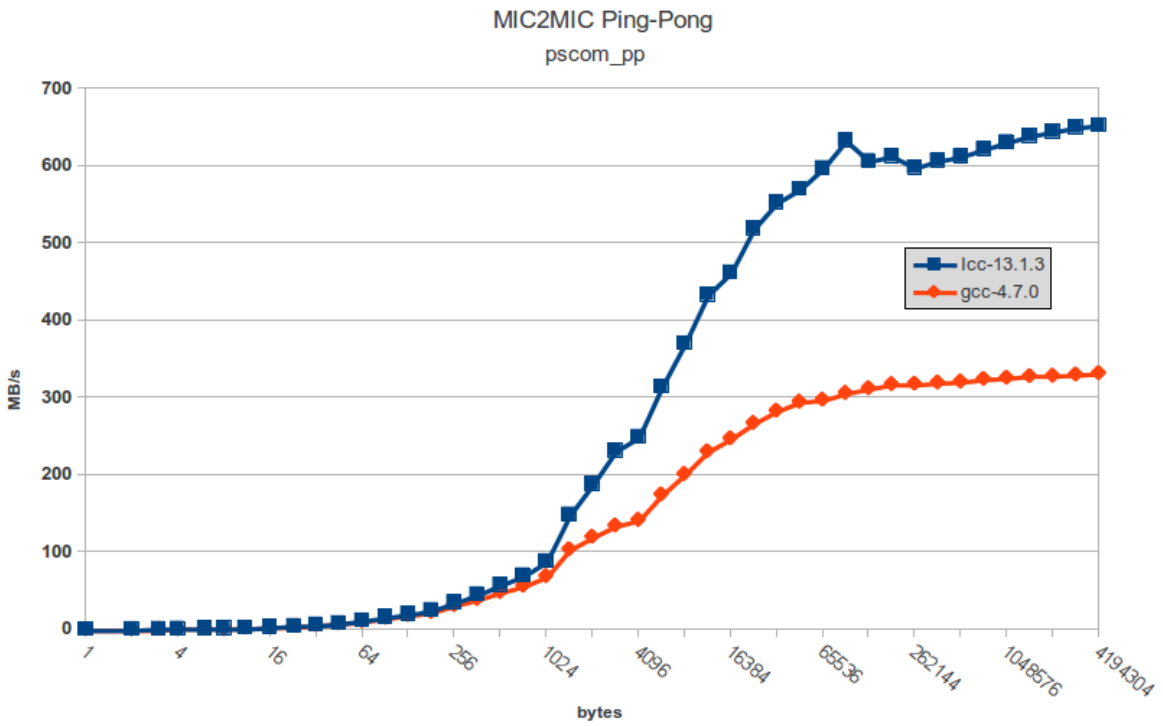

Figure 9: Performance diagram of pscom_pp communicating between two MICs. The X-axis shows the message size, the Y-axis shows the cumulative data transfer rate at a certain message size.

Figure 9 presents the outcome of this performance evaluation comparing a version of pscom_pp built with the Intel C Compiler (*icc* 13.1.3) and a version built with the GNU C compiler (*gcc* 4.7.0).

In the case of the *gcc* binary (red curve) a peak bandwidth of 330 MB/s is achieved at a message size of 4MB. This is only a quarter of the bandwidth seen when communicating directly between two host CPUs via EXTOLL RMA. Also the latency of 5.4 µs is quite high and might have a negative impact on application performance.

In order to exclude bad optimisation of *gcc* on the MIC architecture as the root-cause of the low bandwidth, we decided to use the *icc* compiler to double-check if this leads to a better performance. It turned out that this is actually the case. With an *icc*-compiled binary we reach nearly twice the peak performance (blue curve) than with a *gcc*-compiled binary. In absolute numbers we reach a bandwidth of about 650 MB/s. This at least shows that *icc* is capable to produce much more efficient code for the KNC than the *gcc*. The latency, however, remains the same, which is one of the reasons the performance is worse than in the host-to-host scenario. Nevertheless, this limitation on absolute bandwidth requires further investigation on its root-cause.

6.2 Cluster-Booster Communication

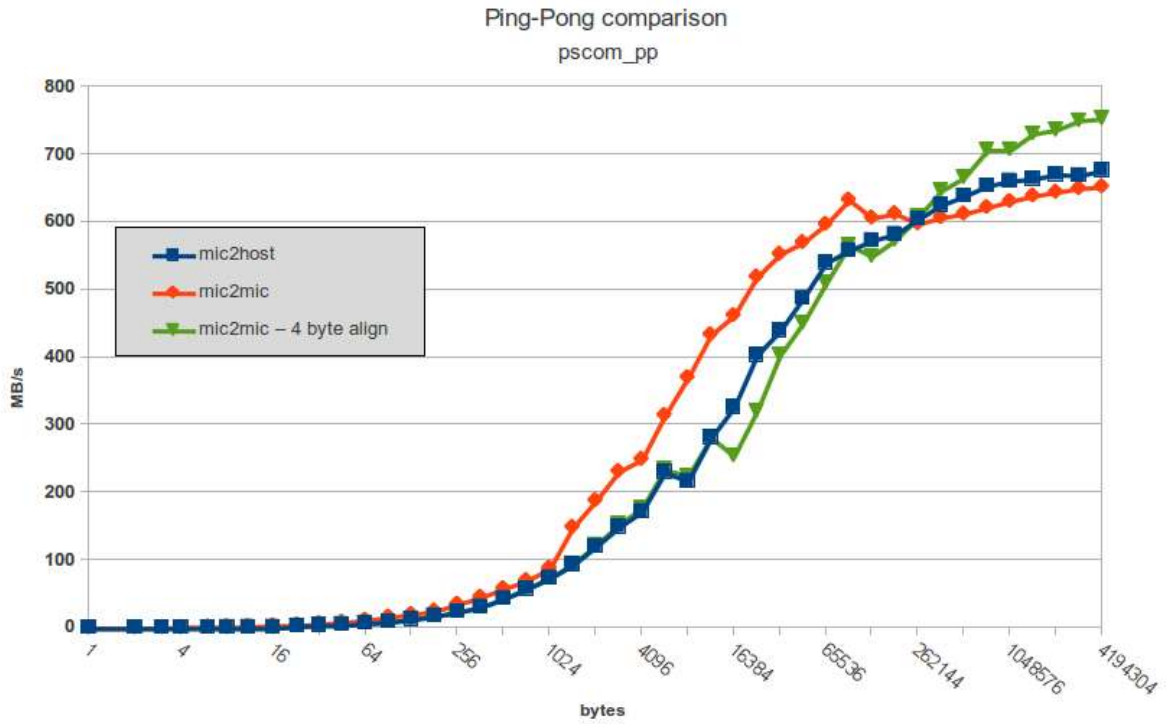

Figure 10: Performance comparison of MIC2MIC communication (EXTOLL RMA) and MIC2Host communication (CBP) via pscom_pp. The X-axis shows the message size, the Y-axis shows the data transfer rate at a certain message size.

Finally, Figure 10 shows the results of a benchmark utilising the Cluster-Booster protocol on the Super-BIC evaluator. Again the pscom_pp benchmark (mic2host) was used but now between CN and the MIC in BN2, with the BID running on the BI. This scenario is very similar to the Cluster-Booster communication to be implemented on the Proto-Booster. Again, for the final hardware (the DEEP System) we expect a significantly improved bandwidth due to the use of the EXTOLL ASIC. For comparison the benchmark described in section 6.1, where two MICs are communicating directly via EXTOLL RMA (mic2mic) is presented, too. Both benchmarks show a similar performance. From 1 KB to 256 KB the mic2mic benchmark delivers a better bandwidth than the mic2host benchmark. This might be explained by the latency impacts due to the much more complex connection setup of the CBP compared to RMA. Beyond this threshold the CBP delivers a higher bandwidth of ~676 MB/s compared to 652 MB/s for mic2mic. This better peak performance might be explained by the fact that one communication end-point is not a MIC but a Xeon host CPU, which is capable to process the protocol faster. Additionally, the MIC cannot make full use of its multi-threading technology, because of pscom's serial dispatch mode.

In order to investigate, if data alignment has an effect on the bandwidth, a further benchmark was performed (mic2mic – 4 byte align). Here a data alignment inside the send buffers of 4 bytes was enforced. It turns out that this leads to an improved bandwidth. The peak performance is now around 750 MB/s. A possible explanation for this behaviour is the fact that MIC is optimised to access data in atomic memory blocks of 4 bytes. When data is not aligned to 4 byte address, a 4 byte data element might cause two accesses to memory, even if only one would be necessary. As the CBP uses *memcpy* to copy the desired data to or from the buffers, the 4 byte alignment also speeds up the protocol.

7 Summary & Outlook

The Cluster-Booster protocol was successfully ported to an environment as similar to the final system as possible for the time being. We managed to achieve a reasonable performance when communicating between a Xeon host and a MIC many-core processor connected by InfiniBand and EXTOLL via a BI node. By introducing a Shared Send Queue a reduction of the memory footprint by a factor of two was attained. Furthermore, the integration of the CBP as a plug-in into pscom is a big step towards a Global MPI between Cluster and Booster as required by the DEEP programming model.

The current peak bandwidth between host and MIC of nearly 750 MB/s for the CBP is less than the bandwidth achieved for host-to-host communication at 920 MB/s. Further investigations are necessary in order to increase the CBP performance when KNC is involved. For that it will be important to fine-tune the alignment with respect to the MPI header.

It will be interesting to see the CBP running on the final hardware. In order to ease optimisation here, several tuneable parameters where introduced into the code.

Adaptions of the protocol as well as of the daemon will be made to fit the CBP perfectly into the ParaStation MPI and the overall Booster middleware developed in task 4.2. In addition, the buffer region approach described in D4.2 will be extended to take into account the ability of the upcoming EXTOLL ASIC to support full KNC memory access via SMFU. This will be done as soon as the EXTOLL ASIC will be available.

References and Applicable Documents

[1] O. Pentakalos, “oreillynet.com,” 2002. [Online]. Available: http://www.oreillynet.com/pub/a/network/2002/02/04/windows.html.

[2] N. Nüssle, B. Geib, H. Fröning and U. Brüning, “An FPGA-based custom high performance interconnection network.,” In Proceedings of the 2009 International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, 2009.

[3] “http://www.par-tec.com/products/parastationv5.html,” ParTec GmbH, 2012. [Online].

[4] M. Forum, “MPI: A Message-Passing Interface Standard. Version,” 4 September 2009. [Online]. Available: http://www.mpi-forum.org.

[5] H. Fröning and H. Litz, “Efficient Hardware Support for the Partitioned Global Address Space,” 10th Workshop on Communication Architecture for Clusters (CAC2010), co-located with 24th International Parallel and Distributed Processing Symposium (IPDPS 2010), Atlanta, Georgia, 2012.

List of Acronyms and Abbreviations

A

B

BI: Booster Interface: System which connects the Booster to the Cluster InfiniBand network within the BIC evaluator

BIC: Booster Interface Card: Interface card to connect the Booster to the Cluster InfiniBand network

BIC evaluator: A platform consisting of three x86-based nodes equipped with (i) an EXTOLL NIC, (ii) an InfiniBand HCA, (iii) both, EXTOLL NIC and InfiniBand HCA, developed and used only in the DEEP project

Super-BIC evaluator: A platform based on the BIC evaluator extended by two servers both housing an EXTOLL NICs and an Intel Xeon Phi MIC processor card.

BID: Booster Interface Daemon

BN: Booster Node (functional entity)

BNC: Booster Node Card: A physical instantiation of the BN

C

CN: Cluster Node (functional entity)

CBP: Cluster-Booster protocol

D

DEEP: Dynamical Exascale Entry Platform: EU-FP7 Exascale Project led by Forschungszentrum Juelich

DEEP Architecture: Functional architecture of DEEP (e.g. concept of an integrated Cluster Booster Architecture)

DEEP Booster: Booster part of the DEEP System

DEEP Supercomputer: A future Exascale supercomputer based on the DEEP Architecture

DEEP System: The production machine based on the DEEP Architecture developed and installed by the DEEP project

E

EC: European Commission

EU: European Union

Exaflop: 10^{18} floating point operations per second

Exascale: Computer systems or applications, which are able to run with a performance above 10^{18} floating point operations per second

EXTOLL: High speed interconnect technology for cluster computers developed by University of Heidelberg

EXTOLL evaluator: Platform for evaluation of EXTOLL technology, developed and used in the DEEP project

F

FLOP: Floating point Operation

FPGA: Field-Programmable Gate Array: Integrated circuit to be configured by the customer or designer after manufacturing

G

Global MPI: MPI allowing communication between the Booster and Cluster part of the DEEP System. Based on the ParaStation process-management and the Cluster-Booster protocol acting as a plug-in for the pscom library. Provides the MPI_Comm_spawn() call used by application processes running on the CNs to start additional processes on the BNs.

H

HCA: Host Channel Adapter
HPC: High Performance Computing
HW: Hardware

I

IB: InfiniBand

J

JSC: Juelich Supercomputing Centre
JUELICH: Forschungszentrum Jülich GmbH, Jülich, Germany

K

KB: Kilo Bytes

L

M

MB: Mega Byte or Mother Board (depending on the context)
MIC: Intel Many Integrated Core architecture
MIC-OS: Operating System of the MIC architecture
MPI: Message Passing Interface: API specification typically used in parallel programs that allows processes to communicate with one another by sending and receiving messages
MPICH: Freely available, portable implementation of MPI

N

NIC: Network Interface Card: Hardware component that connects a computer to a computer network

O

P

ParaStationMPI: Software for cluster management and control developed by ParTec

ParTec: ParTec Cluster Competence Center GmbH, Munich, Germany
PCI: Peripheral Component Interconnect: Computer bus for attaching hardware devices in a computer
PCIe: PCI Express: Standard for peripheral interconnect, developed to replace the old standards PCI, improving their performance
PFlop/s: Petaflop, 10^{15} floating point operations per second
PM: Person Month or Project Manager of the DEEP project (depending on the context)
PMT: Project Management Team of the DEEP project
PSQ: Private Send Queue: A message queue private to each connection.

Q

R

RDMA: Remote Direct Memory Access
RMA: Remote Memory Access: An EXTOLL RDMA protocol

S

SMFU: Shared Memory Functional Unit
SSQ: Shared Send Queue: A message queue shared among all connections of a certain process

T

U

UniHD: University of Heidelberg, Germany

V

VELO: Virtualised Engine for Low Overhead: An EXTOLL communications channel

W

WP: Work Package

X

x86: Family of instruction set architectures based on the Intel 8086 CPU

Y

Z

ZITI Heidelberg: Institut für Technische Informatik Uni Heidelberg, Germany