Connecting the Cloud: Transparent and Flexible Communication for a Cluster of Intel SCCs

Pablo Reble*, Carsten Clauss*, Michael Riepen†, Stefan Lankes* and Thomas Bemmerl*
*Chair for Operating Systems, RWTH Aachen University, Aachen, Germany
Email:{reble,clauss,lankes,bemmerl}@lfbs.rwth-aachen.de
†Intel Labs Braunschweig, Braunschweig, Germany
Email: michael.riepen@intel.com

***Abstract*—The focus of this paper is the analysis of Inter-Processor Communication for future many-core processors. We present a prototype that connects two Intel Single-chip Cloud Computer boards via PCI express in a transparent way, which creates a high flexibility concerning the target of a remote memory access. This enables message passing based applications using RCCE as well as MPI to run on a Cluster-of-Clusters consisting of 96 SCC cores. As a result, it is possible to analyze programming models for the SCC Many-Core Processors regarding scalability and hierarchy-aware communication in more detail. Along with this, we present first performance results and develop alternative communication and synchronization schemes for optimization.**

I. Introduction

The Single-chip Cloud Computer (SCC) experimental processor [1] is a *concept vehicle* created by Intel Labs as a platform for many-core software research, which consists of 48 cores arranged in a 6×4 on-die mesh of tiles with two cores per tile. As a research prototype, the SCC platform has two different types of non-coherent shared memory: *off-die* memory by four DDR3 memory controllers as well as *on-die* memory, called Message Passing Buffers (MPBs). In this paper we introduce a third type of memory, that is located on-die at a remote SCC.

Currently, two possibilities exist for the *Inter-Board Communication* (IBC) of multiple SCC systems. Both are network based, and use the Ethernet protocol, either via physical ports (eMAC) or software tunnel (PCIe). Since the presence of the eMAC ports, the tunneled version via PCIe has become more or less obsolete for network based communication but the connection is still used for setup and debugging purpose. Nevertheless, explicit communication is provided both ways, including overhead of the network stack.

RCCE as the communication library for the SCC processor uses the Message Passing Buffers for communication and synchronization. The approach to transparently map the remote MPBs arises a third possibility for *Inter-Board Communication*. This method allows a reuse of the common RCCE API for multiple SCCs without any modifications. In this paper we present first performance results for the use of RCCE in *big flags* mode with multiple SCCs and resulting optimizations. Furthermore, we evaluate the performance of our prototype with iRCCE and MPI application benchmarks.

The next section covers related work for a connection of Many-core Processors and draft an alternative approach for IBC implemented in the System Interface FPGA. The rest of this paper is structured bottom up concerning the communication layers. In Section III, we detail the basic functionality of our prototype and first performance results for RCCE. Section V includes the use of the MPI implementation SCC-MPICH for multiple SCCs. In Section VI, we discuss benchmark results that point out the potential of our prototype. Section VII summarizes this paper and gives an outlook for future work.

II. Related Work

A broad variety of options exist for the connection of computing systems, each consisting of one or more processors. In the field of High Performance Computing (HPC), mass market products like Ethernet and Infiniband as well as specialized products like Bull Coherent Switch (BCS), Dolphin PCI Express or NumaScale Interconnect are used for the communication between computing nodes. This creates clusters, huge SMP systems, or hybrid variants out of coherent and non-coherent interconnects for commodity hardware. Software applications typically use those interconnects in an implicit or explicit way of communication, depending on the parallel programming paradigm.

At the *Chair for Operating Systems* the performance of a bare-metal application using the *MetalSVM* eMAC driver and the lwIP stack has been analyzed [2]. For the eMAC driver of sccLinux *2.6.38* a performance issue has been identified by comparing its throughput to the bare-metal variant. Meanwhile, a new version of sccLinux is available based on the *3.1.4* Linux kernel, which we use in this paper for a comparison with the transparent IBC.

Intel Labs in Braunschweig developed a concept for a cluster of eight SCCs, directly connected through the System Interface FPGAs. This basic idea is based on a re-use of the SCC silicon router hardware by the FPGA to create a Cluster-of-Clusters (CoC) with 2×4 nodes SCCs, which allows running 384 cores in parallel. The System Interface (SIF) FPGA – a Xilinx Virtex 5 device – contains fast general purpose IO ports (*Rocket I/O*) that can be used to communicate between two neighboring SCC systems. By re-mapping the mesh packets that leave the SCC and routing them through the SIF FPGA similar to the method that is presented in Section III, it would be possible to allow message passing

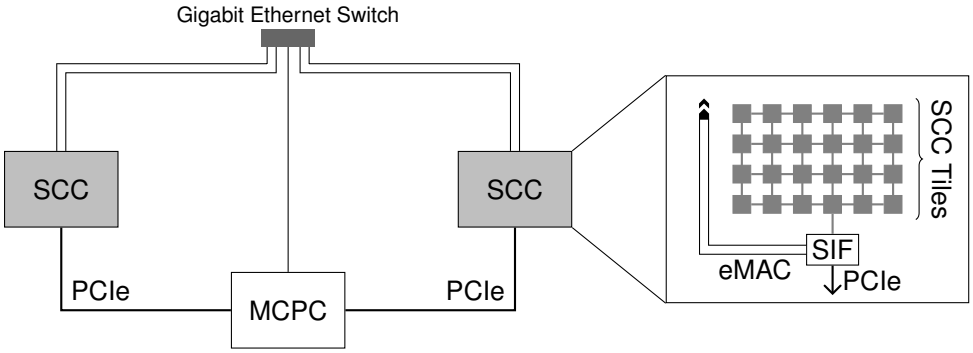


Fig. 1: *Inter-Board Communication* (IBC) Prototype

across the SCC nodes. As the mesh interfaces in each SCC tile are highly optimized, there are no redundant bits in the packets that can be used to encode the *Chip ID* in addition to the *Tile ID* which is already contained in the packets.

Therefore, it is possible to modify the so called *Transaction ID* without an impact on the behavior of the Mesh Interface. This technique allows the clustering of up to eight SCC systems. Because of a high utilization of the FPGA resources it would be necessary to limit the number of parallel memory transactions to a maximum of eight. Thus, if more than eight cores would need to access off-chip message passing buffers, the requests would be serialized, with a maximum of eight parallel requests per SCC in flight. Due to these restrictions, Intel Labs decided not to develop a prototype of a SCC Cluster-of-Clusters that is implemented in the FPGA and uses *Rocket I/O*. Indeed, the prototype that is presented in the next section has similar restrictions but has the advantage of a faster prototyping to build a Cluster of SCCs.

III. Prototype

The SCC platform has been developed by Intel Labs as a concept vehicle for many-core software research. The standard software framework for the SCC is called *sccKit*, which contains a Linux version that provides drivers for network support as the only option for *Inter-Board Communication* (IBC). Provided options are a tunneled network device via PCI express and a physical Ethernet support through the System Interface FPGA.

The communication model of the prototype presented in this section is different. As proposed in the previous section, we target a transparent access at low level between two SCC nodes without changing the System Interface FPGA. Figure 1 shows the setup of the prototype, which is presented in this section. Therefore, we first detail in this section the needed changes to the SCC infrastructure for a realization of the prototype. And second in the next section, we detail needed changes to the SCC communication library RCCE and and its extension iRCCE to optimize the performance.

A. Alternative Hardware

The idea is to connect two SCCs via the existing PCIe communication path between the SCC cores and the System Interface (SIF) and thereby allow a transparent communication on low level without any hardware changes to the SCC systems.

Although, we had to choose alternative Management Console PC (MCPC) hardware for the realization of a prototype because the hardware recommended by Intel Labs does not support the use of a second PCIe expansion card. The use of alternative hardware for the MCPC has been an active discussion in the MARC community. Many users reported bad experiences by using different hardware than the recommended configuration. Our configuration is based on an Intel Server Board S5000PSL, which works for the operation with two SCCs.

In addition to that, an extension of the Linux Kernel driver `crbif` and the *sccKit* was needed to support the use of multiple SCC devices with one MCPC. Additionally, our `crbif` extension has to provide routing functionality in software to enable a transparent *Inter-Board Communication*. This functionality is detailed in the next paragraph.

B. MCPC Router Functionality

Because of the 32 bit architecture of the SCC cores and a larger amount of addressable memory than 4 GByte of the SCC platform, another indirection between core addresses and physical system addresses is needed. One Lookup Table (LUT) per core holds 256 entries, what results in a page frame size of 16 MByte that is addressable per LUT entry.

Here, for the realization of a transparent interconnect a single LUT entry can be used to map all MPBs of a remote SCC. This is possible, because of the `crbif` driver, which is extended to serve the generated requests by identifying incoming requests for a remote SCC and forward them to the specific destination.

Since the hardware routing of the on-die 2D mesh network is fixed, it is not possible to encode the destination on a remote SCC for IBC within the address field of a physical address. Packages that are intended for a remote SCC have to be addressed to the System Interface first (cf. Figure 1). These packages are tagged with a unique key in the address field, similar to the method of the tunneled Ethernet driver. Besides this, an IBC package has to contain information about

the remote destination and sub-destination, so that the MCPC can generate a correct request package.

As mentioned in the last section, the reserved bits of a system address are useless for this purpose because of the highly optimized mesh of the SCC, which does not transfer these bits. A workaround, which we applied to the prototype that is presented in this section is to encode the remote destination within the remaining address offset. This obviously shrinks the maximum amount of memory which can be addressed by one LUT entry that targets a remote SCC. For the completion of an outstanding read request, the MCPC has to identify the returning package and forward it to the specific core. If the outstanding request is a write request and the automatic acknowledges of the SIF are disabled, the MCPC has to generate a specific package.

C. Optimization

Due to the reused Pentium® architecture, each SCC core can only have one outstanding memory request. The SIF handles the communication between SCC chip and MCPC and has the option to generate automatic write acknowledges.

For the prototype of a transparent *Inter-Board Communication* for the SCC, enabling automatic write acknowledges significantly decreases the latency for a remote write access. Moreover, write accesses to a remote SCC can be interleaved, which makes the remote write operation preferable for the communication of multiple SCCs. Therefore, we analyze in the next paragraph the possibility to change the communication protocol of RCCE to obtain an optimal performance for the inter-processor communication.

Besides the significantly lower latency of a write access, this option to generate automatic write acknowledges turned out to be problematic. Especially situations with a high load can lead to lost packages and even to a complete system crash. As a result, the higher performance mode of our prototype limits a stable operation to situations with a moderate load.

Similar issues that arise from a high load for the on-die mesh network resulting from a central communication point have been discovered in [3]. Here, more than 24 cores can generate a high contention through busy waiting to the mesh, which leads to a starvation of cores. This effect can be be reacted, by introducing a back-off. Such a workaround in software would break the transparency of the *Inter-Board Communication*. To build a transparent setup with an increased stability and optimal performance it would be necessary to change the behavior of the SIF and serialize a certain amount of outstanding requests, as proposed in Section II.

Further optimization includes the behavior of the `crbif` driver. Communication with the FPGA is handled by DMA transfers. If no communication is pending the driver saves CPU cycles on the MCPC. Turning-off this mode, decreases the latency of a remote communication by polling for incoming data instead.

IV. RCCE Communication Library

RCCE [4] is limited to blocking communication using the MPBs to transfer messages. The communication protocol is based on a *local-put/remote-get* (LPRG) communication scheme (cf. 2a). A communication with `RCCE_send()` and `RCCE_recv()` works as follows. First, the sender puts the message to its local MPB. Second, the sender toggles a flag at the receiver side for the indication of this event and finally waits at a synchronization point for indication that the receiver has copied the message to its private memory.

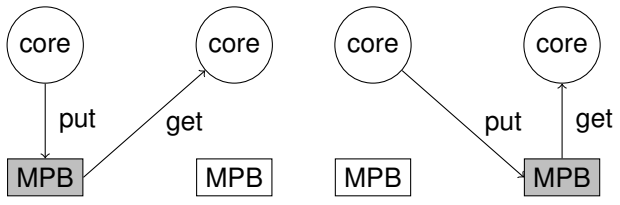
For *On-Die Communication* (ODC), which RCCE has been developed for, this scheme has several advantages. Here, it is guaranteed that each core writes to its local MPB only, which simplifies the flag based synchronization method.

The transparent *Inter-Board Communication*, as proposed in the previous section, generates the possibility to provide an access to the MPB of a remote SCC. As a result, RCCE can be used for *On-Die* and *Inter-Board Communication* on the SCC platform, by extending the communication ranks in a linear way (`rck00` ... `rck95`). However, a throughput of maximum 1 MByte/s can be achieved for the pingpong application using the default communication scheme (cf. Figure 3). The reason is that the communication with LPRG is based on remote read requests to transfer data.

For our prototype, the *Inter-Board Communication* path is routed through the MCPC, which increases the latency for a remote access significantly (cf. Table I). In fact, a read access takes approximately two timer longer than a write access. Obviously, the option where the FPGA generates automatic write acknowledges, even magnifies this effect by accelerating only write accesses. For the performance evaluation of our transparent IBC prototype we decided to implement an alternative communication scheme for RCCE called *remote-put/local-get* (RPLG), where the sender puts the message to the Message Passing Buffer of the receiver.

A. Communication Schemes

Figure 2 illustrates the two alternative communication schemes for RCCE. In Figure 2a and 2b, its basic communication scheme *local-put/remote-get* (LPRG) is shown besides the alternative communication scheme *remote-put/local-get* (RPLG). For the ODC, the LPRG communication scheme performs slightly better than the RPLG because of the simplified synchronization.



(a) *local-put/remote-get* (LPRG) (b) *remote-put/local-get* (RPLG)

Fig. 2: Communication Schemes

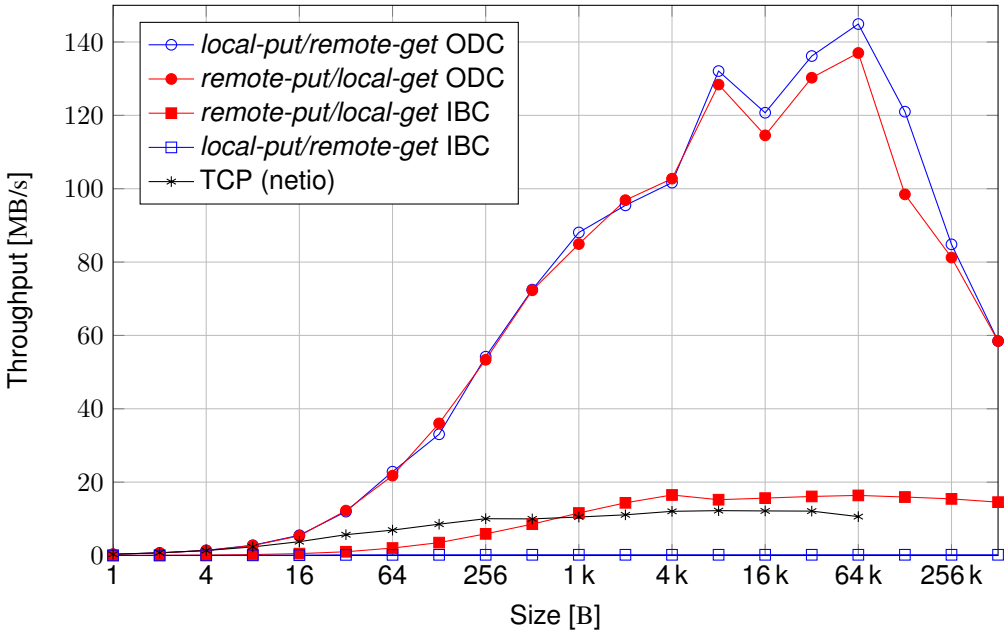


Fig. 3: Throughput for iRCCE Ping-Pong and TCP netio

The protocol of the RPLG communication scheme has the disadvantage that multiple cores can concurrently write to the same MPB, which requires additional synchronization (cf. Section V). Here, the throughput for IBC raises up to 16 MByte/s for a larger message size, because the write accesses to a remote memory location such as the MPBs can be pipelined. Our measurements have shown that the bisection throughput for this configuration is 47 MByte/s. For all these measurements, the automatic write acknowledges were enabled. Compared to the *On-Die Communication*, an exchange of the communication scheme creates a maximum throughput for the *Inter-Board Communication* that ranges from 5 to 10 percent. This is in the same order of magnitude than the communication via TCP/IP using the eMAC ports.

B. iRCCE Communication Extension

The iRCCE library [5] has been developed at the RWTH Aachen University as an extension to the common RCCE library in order to support a light weight non-blocking communication for the SCC. Additionally, iRCCE offers new inquiring functions for the *Inter-Board Communication*. Quite similar to the common functions `RCCE_ue()` and `RCCE_num_ues()`, these new functions `iRCCE_dev()` and `iRCCE_num_devs()` return the ID and the count of SCC boards involved. Furthermore, by means of the new function `RCCE_num_ues_dev()`, the number of active cores per board can be inquired. All together, these five functions provide enough information so that each core can determine its position within the hierarchical setup of a coupled iRCCE session. That way, future iRCCE applications may take the hierarchical nature of such a cluster of SCCs into account.

V. SCC-MPICH

SCC-MPICH is based on MP-MPICH, a multi-platform message-passing library that in turn is derived from the original MPICH and that particularly offers some features for building Cluster-of-Clusters [6]. This part of MP-MPICH, called MetaMPICH, provides hierarchy-awareness and allows for using different communication facilities (in terms of multiple communication *devices*) at the same time.

The SCC-related part of SCC-MPICH is a new communication device that makes use of the on-die MPB for the core-to-core communication [7]. This device, which in turn is based iRCCE, is highly optimized for providing low latencies and therefore offers three different communication protocols (*Short*, *Eager*, *Rendezvous*), which are chosen according to the message length. In addition to these, a fourth communication protocol (*SHM-Eager*) can make use of the off-die shared-memory for passing messages between the cores. However, since this detour via the DDR3 memory badly impacts latency and throughput, this protocol is usually only used for comparison purpose.

As a hierarchy-aware Message Passing Interface (MPI) library, SCC-MPICH allows the user to spawn one single MPI session across multiple SCC boards. In doing so, SCC-MPICH uses TCP/IP for the inter-board communication while the on-die communication is still handled via iRCCE and thus via the MPBs [8]. As the IBC obviously constitutes a bottleneck in terms of lower throughput and higher latencies, SCC-MPICH offers some features to make such hierarchical topologies queryable and thus visible even at application level. Therefore, at least likewise hierarchical algorithms, as for example coupled codes, can be partitioned to circumvent this bottleneck.

On-Die Communication (ODC) vs. *Inter-Board Communication* (IBC):	*Normal* flags		*Tagged* flags	
	ODC	IBC	ODC	IBC
iRCCE in remote-put/local-get mode:	2.4 μs	32.2 μs	1.3 μs	26.4 μs
iRCCE with additional *probe flags*:	2.9 μs	72.4 μs	2.1 μs	26.5 μs
SCC-MPICH with common *Short Protocol*:	3.4 μs	—	3.1 μs	—
SCC-MPICH in *Coupled Mode*:	7.3 μs	46.1 μs	3.5 μs	29.8 μs

TABLE I: Latency Comparison regarding *Tagged Flags*

Although, the TCP/IP-based coupling of multiple SCC boards works quite well, a transparent inter-chip communication as presented in the prior sections promises much lower latencies due to the omission of a higher-level network stack. Because SCC-MPICH is based on iRCCE, the application of a transparent coupling in terms of an iRCCE session spawned transparently across two or more SCC boards seems, at least at a first glance, quite simple.

However, it turned out that especially the switchover from *local-put/remote-get* to the *remote-put/local-get* pattern within iRCCE, as described in the previous section, constitutes a major pitfall for its integration into SCC-MPICH. This is because SCC-MPICH, in its capacity as an MPI library, needs (in contrast to RCCE and iRCCE) a so-called *Progress Engine*[1] that iteratively checks for incoming messages and that is able to dispatch and even to reorder messages according to *MPI rank*, *MPI tag* and *MPI communicator* used.

This can be achieved on top of iRCCE by means of polling on all remote MPBs in a round-robin fashion for new messages addressed to a particular receiver. However, when switching to *remote-put/local-get* semantics, a receiver can just poll on its own local MPB while potential senders have to access this MPB in a concurrent manner.

In order to fix this issue, additional *probe flags* (`nue-1` in count) can be introduced for each core to reconstruct the needed communication pattern, where `nue` is the total number of started processes. However, the implementation of this method in a straightforward manner introduces an additional synchronization overhead that leads to an increased latency, as illustrated in Figure 4. Therefore, we have spent a lot of optimization effort to keep this additional overhead as low as possible.

The way we followed to realize this was to use so-called *Tagged Flags* that can carry additional payload information alongside with their synchronization signals. This can be achieved when using the so-called *Big Flags* mode of iRCCE, where each synchronization flags is embodied by a whole cache line of 32 Byte. In the common use of this mode, only one integer word of 4 Byte is used for the actual synchronization whereas the remainder is unused. However, in *Tagged Flags* mode these remaining 28 Bytes can be used for small payloads in a *piggyback* fashion.

[1]A *Progress Engine* is a dedicated part of an communication layer that handles signaling messages, e.g. for flow control, and takes care of progress for still pending payload messages in an iterative but transparent manner.

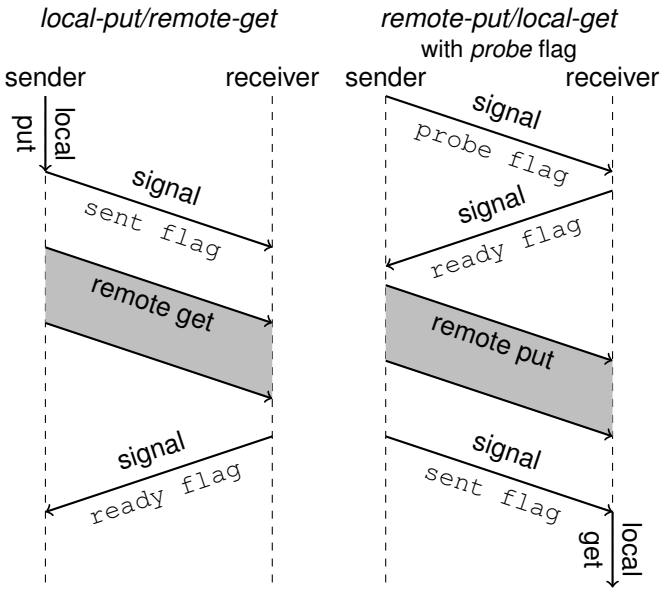


Fig. 4: Timely behavior of Communication Protocols

This is because on hardware level, all data exchange via the mesh is conducted in cache line granularity.

Table I holds the obtained latencies for MPI and iRCCE with *On-Die Communication* (ODC) and *Inter-Board Communication* (IBC) for the common *PingPong* application. Theses measurements demonstrate that the resulting overhead of the introduction of additional probe flags for small messages can almost be compensated by using the described *Tagged Flags* feature. Obviously, this feature reduces also the latencies between cores communicating on the same die. Therefore, we plan to integrate the option for Tagged Flags also into the next major release of iRCCE in terms of a shortcut mechanism that can improve the latencies of the common `iRCCE_send()` and `iRCCE_recv()` functions.

	ODC	IBC
iRCCE (with *remote-put/local-get*):	1.3 μs	26.4 μs
SCC-MPICH on iRCCE:	3.1 μs	29.8 μs
SCC-MPICH with TCP over MPB:	140 μs	—
SCC-MPICH with TCP over eMAC:	108 μs	108 μs
Raw TCP (Sockets) over MPB:	134 μs	—
Raw TCP (Sockets) over eMAC:	94 μs	94 μs

TABLE II: Latency Comparison between iRCCE and TCP

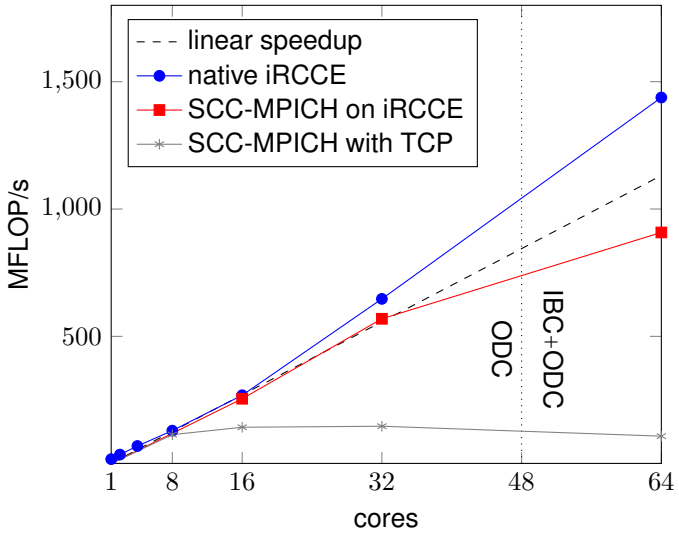


Fig. 5: LU (CLASS B) from the NAS Parallel Benchmarks

Since SCC-MPICH provides TCP/IP-based communication, we applied the *PingPong* benchmark for such a configuration. The advantages of omitting the TCP/IP software stack becomes blatantly obvious, by comparing the measured latencies from Table II for TCP/IP and iRCCE-based communication.

VI. Benchmarks

For a more challenging problem for the SCC, Mattson et al. ported the LU and BT benchmarks from the well known NAS [9] parallel benchmarks to RCCE [10]. To benchmark our prototype, we run the LU benchmark (Class B) with iRCCE and SCC-MPICH using the transparent *Inter-Board Communication*. Here, we obtained measurements for up to 64 cores[2] by connecting two SCC boards. That means that up to 32 cores, the benchmark runs only on a single SCC, whereas the scenario with 64 cores represents the coupled case.

As one can see in Figure 5, the benchmark scales quite well as long as there is *On-Die Communication* only.[3] Surprisingly, the benchmark with iRCCE on 64 cores, the computing performance scales *super-linearly* for the coupled case.

The reason is that the improved `memcpy` for the SCC of iRCCE [11] uses software prefetching of data, which leads to a better cache behavior for the benchmark. The Class B problem of the LU kernel is small enough to fit into the agglomerated caches of the 64 cores. This shows very plainly that spawning one parallel session across multiple SCCs may not only benefit from the higher degree of parallelism but also from an increased aggregated cache size.

However, when looking at the MPI performance for 64 cores, a major drawback of using additional probe flags (at least in iRCCE's *big flags* mode) becomes obvious:

[2]The LU benchmark requires a power of two as the number of started processes. Hence, all 96 cores cannot be used in this scenario.

[3]Frequency settings (Core/Mesh/Memory): 533 MHz/800 MHz/800 MHz

Since the allocation of these flags decreases the amount of MPB space remaining for the actual payload transfer, the achievable communication throughput decreases, too. In fact, when starting an SCC-MPICH session running on 64 cores, 6 kByte MPB space per core gets just allocated for synchronization purpose (and of course for short messages that can be sent via Tagged Flags), whereas merely 2 kByte are effectively left for transferring larger messages. Finally, the bad impact of this effect can be observed in Figure 5 for the MPI case where the computing performance significantly suffers for the 64 cores scenario.

VII. Conclusion and Outlook

In this paper we presented a concept for a low-level *Inter-Board Communication*. We developed and analyzed a prototype for the SCC platform which connects 96 cores in a transparent way. As a result, the RCCE communication library can be used for the communication between multiple SCC boards with a hierarchical communication structure. We figured out that with *remote-put/local-get*, the change of the communication scheme leads to best performance for the *Inter-Board Communication*.

For future research, it has to be evaluated how SCC-MPICH can be optimized if both connections (PCIe and eMAC) are used in parallel. Parallel applications can use hierarchy-aware communication patterns to get best performance, especially with respect to collective communications. Therefore, we draft extensions to the common iRCCE API to expose locality to the programmer. We see potential that other research projects from the MARC community may profit from the results that has been presented in this paper. For this purpose, we plan to integrate the *remote-put/local-get* communication scheme and tagged flag support to the next iRCCE release. Additionally, we plan to expand our prototype by mapping remote synchronization and interrupt registers, as well as off-die shared memory regions with the techniques that are presented in this paper. This will enable the analysis of approaches like Remote Direct Memory Access (RDMA) [12], Software based Coherence [13] or Inter-Kernel Communication and Synchronization [14] with an increase of the scalability and overcome the limitation using only *big flags* mode of RCCE (cf. Section VI) for *Inter-Board Communication* as in the scope of this paper.

The general concept of a transparent communication with an MCPC that acts as a software router is not limited to a Cluster of SCCs with two boards, and a resulting core count of 96 as in our setting. Obviously, the limitation is generated by the maximum number of PCIe devices, that can be connected to a single MCPC. Moreover, if one of the MARC research projects can benefit from a setup with up to eight SCC boards connected by *Rocket I/O*, the project owner should not hesitate to contact the authors in order to discuss the options for a realization of the concept.

ACKNOWLEDGMENT

This research was funded by Intel Corporation. The authors thank Michael Konow, Jan-Michael Brummer and Niels Ole Salscheider for their help and guidance.

REFERENCES

[1] *SCC External Architecture Specification (EAS)*, Intel Corporation, November 2010, Revision 1.1. [Online]. Available: http://communities.intel.com/docs/DOC-5852

[2] S. Lankes, P. Reble, C. Clauss, and O. Sinnen, "The Path to MetalSVM: Shared Virtual Memory for the SCC," in *Proceedings of the 4th Many-core Applications Research Community (MARC) Symposium*, Potsdam, Germany, December 2011. [Online]. Available: http://communities.intel.com/docs/DOC-19214

[3] P. Reble, S. Lankes, F. Zeitz, and T. Bemmerl, "Evaluation of Hardware Synchronization Support of the SCC Many-Core Processor," in *Proceedings of the 4th USENIX Workshop on Hot Topics in Parallelism (HotPar 12)*, Berkeley, CA, USA, June 2012. [Online]. Available: https://www.usenix.org/system/files/conference/hotpar12/hotpar12-final9.pdf

[4] T. Mattson and R. van der Wijngaart, *RCCE: a Small Library for Many-Core Communication*, Intel Corporation, January 2011, Software 2.0-release. [Online]. Available: http://communities.intel.com/docs/DOC-5628

[5] C. Clauss, S. Lankes, T. Bemmerl, J. Galowicz, and S. Pickartz, *iRCCE: A Non-blocking Communication Extension to the RCCE Communication Library for the Intel Single-Chip Cloud Computer*, Chair for Operating Systems, RWTH Aachen University, July 2011, Users' Guide and API Manual, V1.2. [Online]. Available: http://communities.intel.com/docs/DOC-6003

[6] B. Bierbaum, C. Clauss, R. Finocchiaro, S. Schuch, M. Pöppe, and J. Worringen, *MP-MPICH – User Documentation and Technical Notes*, Chair for Operating Systems, RWTH-Aachen, University. [Online]. Available: http://www.lfbs.rwth-aachen.de/users/global/mp-mpich/mp-mpich_manual.pdf

[7] C. Clauss, S. Lankes, and T. Bemmerl, "Performance Tuning of SCC-MPICH by means of the Proposed MPI-3.0 Tool Interface," in *Proceedings of the 18th European MPI Users Group Meeting (EuroMPI) 2011*, Santorini, Greece, September 2011. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-24449-0_37

[8] C. Clauss, S. Pickartz, S. Lankes, and T. Bemmerl, "Hierarchy-Aware Message-Passing in the Upcoming Many-Core Era," *Grid Computing – Technology and Applications, Widespread Coverage and New Horizons*, pp. 151–178, 2012. [Online]. Available: http://www.intechopen.com/articles/show/title/hierarchy-aware-message-passing-in-the-upcoming-many-core-era

[9] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, R. Fatoohi, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The NAS parallel benchmarks," *Intl. Journal of Supercomputer Applications*, vol. 5, no. 3, pp. 66–73, 1991.

[10] T. Mattson, R. van der Wijngaart, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, and S. Dighe, "The 48-core SCC Processor: The Programmer's View," in *Proceedings of the 2010 ACM/IEEE Conference on Supercomputing (SC10)*, New Orleans, LA, USA, November 2010.

[11] C. Clauss, S. Lankes, P. Reble, and T. Bemmerl, "Evaluation and improvements of programming models for the intel scc many-core processor," in *Proceedings of the 2011 International Conference on High Performance Computing and Simulation (HPCS 2011)*, Istanbul, Turkey, July 2011, pp. 525 –532. [Online]. Available: http://dx.doi.org/10.1109/HPCSim.2011.5999870

[12] S. Christgau and B. Schnor, "One-Sided Communication in RCKMPI for the Single-Chip Cloud Computer," in *Proceedings of the 6th Many-Core Applications Research Community (MARC) Symposium*, July 2012, pp. 19–23. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00719017

[13] R. Rotta, T. Prescher, J. Traue, and J. Nolte, "Data Sharing Mechanisms for Parallel Graph Algorithms on the Intel SCC," in *Proceedings of the 6th Many-Core Applications Research Community (MARC) Symposium*, July 2012, pp. 13–18. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00718993

[14] P. Reble, S. Lankes, C. Clauss, and T. Bemmerl, "A Fast Inter-Kernel Communication and Synchronization layer for MetalSVM," in *Proceedings of the 3rd MARC Symposium, KIT Scientific Publishing*, Ettlingen, Germany, July 2011. [Online]. Available: http://communities.intel.com/docs/DOC-6871