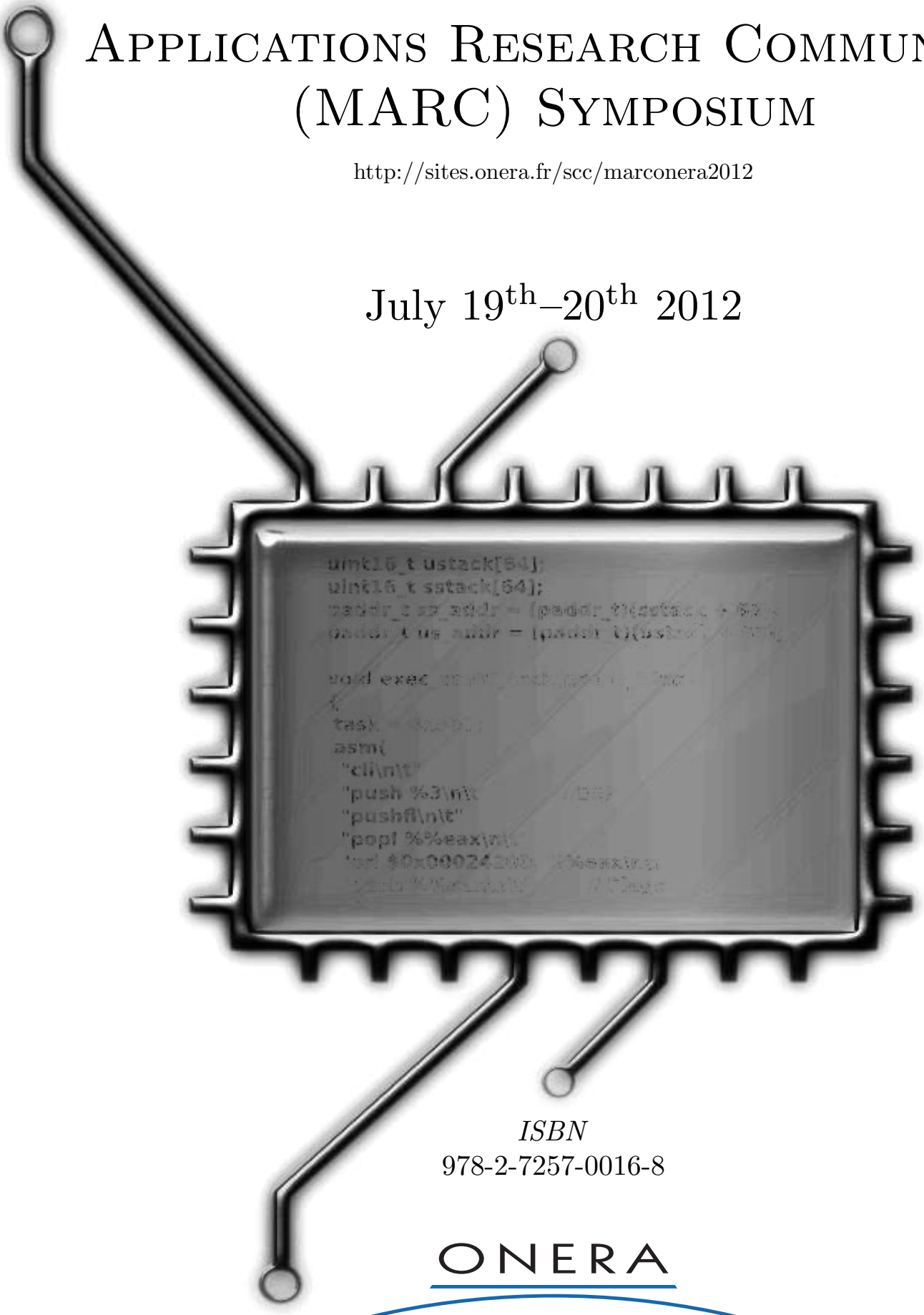
One-Sided Communication in RCKMPI for the Single-Chip Cloud Computer

Steffen Christgau, Bettina Schnor

Proceedings of the 6th Many-core Applications Research Community (MARC) Symposium

http://sites.onera.fr/scc/marconera2012

July 19th–20th 2012



ISBN
978-2-7257-0016-8

ONERA
THE FRENCH AEROSPACE LAB

 ISBN: 978-2-7257-0016-8

One-Sided Communication in RCKMPI for the Single-Chip Cloud Computer

Steffen Christgau and Bettina Schnor
Institute of Computer Science, University of Potsdam
August-Bebel-Strasse 89, 14482 Potsdam, Germany
Email: {christgau, schnor}@cs.uni-potsdam.de

***Abstract*—One-Sided Communication functions have been defined in the Message Passing Interface standard since version 2. Modern implementations of the interface, such as MPICH2, support One-Sided Communication. This paper presents insights to the MPICH2 architecture and the implementation of One-Sided Communication in its low-level CH3 communication modules. Further, issues for using MPI's One-Sided Communication features on the Single-Chip Cloud Computer are presented and resolved. The paper also presents a comparative scalability study of an example application both on the SCC and on an InfiniBand cluster.**

I. INTRODUCTION

The Message Passing Interface (MPI) [1] is a commonly used programming API to implement parallel applications. Version 2 of the MPI Standard [2] specifies synchronous, asynchronous, one-sided, and two-sided communication, where one-sided operations are defined as being asynchronous by default.

MPI two-sided communication calls are due to the classic send-recv scheme where each send call must have a matching receive call at the destination. In contrast, One-Sided Communication (OSC) defines a process interaction mechanism where only *one* process specifies the communication parameters. The other process (called *target*) only has to provide the memory area (called *window*) and is not required to call any communication routine.

MPI-2-OSC separates communication from synchronization and defines so called synchronization *epochs*. An arbitrary number of communication calls can be synchronized within a single epoch. This bundling reduces the need to make synchronization calls for each transfer. MPI-2 provides several API calls to open and close such an epoch. Figure 1 shows some pseudo code of the typical use of one sided and two sided communication.

The remainder of the paper is organized as follows: First, an overview of the architecture of MPICH2 and the implementation of MPI's OSC method in the low-level CH3 devices is presented. Based on this discussion, issues in the MPI implementation for the SCC are revealed and fixed. The corrected implementation is then used in Section V to compare the scaling of an example CFD application both on the SCC using one- and two-sided communication as well as on an InfiniBand cluster. The conclusion and an overview of related work constitute the end of the paper.

```
Node A                          Node B
```

One-sided example:

```
MPI_Win_Create                  MPI_Win_Create

MPI_Win_Post                    MPI_Win_Start
 <computation>                   <computation>
                                 MPI_Put
                                 <computation>

MPI_Win_Wait                    MPI_Win_Complete
```

Two-sided example:

```
                                <computation>
 MPI_Irecv                      MPI_Isend
 <computation>                  <computation>
 MPI_Wait                       MPI_Wait
```

Fig. 1. Pseudo code examples of unidirectional asynchronous data exchange via one-sided and two-sided communication.

II. MPICH2 ARCHITECTURE

MPICH2 is a portable MPI implementation [3]. Its modular architecture is shown in Figure 2. The methods defined by the MPI standard are implemented on top of third generation of the abstract device interface (ADI3) [4]. This layer wraps around the message transportation device of the target platform and thereby provides hardware-independence. A downside of this interface is the huge amount of functions that have to be implemented for new platforms. To facilitate the adaption on new hardware platforms the CH3 device was introduced. This device implements the ADI3 and presents a much simpler interface to so-called channels. The channels implement the CH3 interface and are responsible to receive and send messages from resp. to the hardware.

Within an CH3 channel implementation several requirements and conventions from the higher layers have to be considered. First, all MPI processes are considered to be connected by virtual connections which do not have to manifest in physical connections. Device specific information for a connection between two processes, a socket e.g., can be attached to the according virtual connection. If a message is going the be sent to a remote process, the upper layers pass a request to the CH3 channel which activates the according virtual connection. If it is not possible to satisfy the request

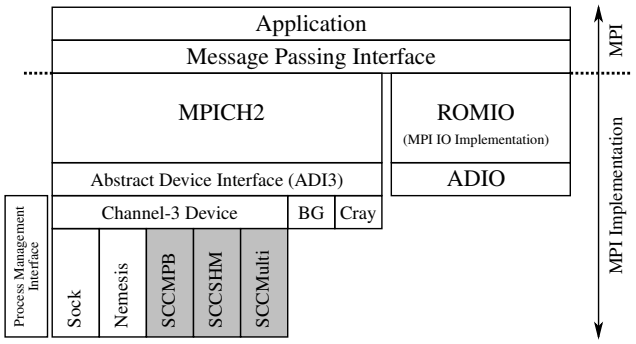


Fig. 2. MPICH2's layered architecture

immediately, that is the message was not transferred at all or only a partial transfer was possible, the channel is responsible to try a retransmission of the unsent part at a later time. This can be done by adding the request to a queue associated to the virtual connection.

To send outstanding messages, the CH3 device invokes a so-called progress engine which must be implemented by the channel. The implementation checks whether requests have been enqueued and if so, it tries to send these messages or even fragments. With the invocation of the progress engine, the channel must also check for new messages that have been received by the hardware.

If the channel has detected a new message, it asks upper MPICH2 layers to create a new receive request based on the received data. As soon as more data for this request is received by the channel, it must check whether the request has been completed by the incoming data. To do so, call-back functions are invoked. Besides a default completion call-back function, a request may provide its specific routine that has to be called by the channel. This notification scheme also applies to the requests created when sending messages. Due to the callback mechanism, the decision whether the request has been completed successfully is not up to the channel itself, but to the higher MPICH2 layers. The channel implementation must therefore be aware of changes in the send resp. receive requests.

III. OSC WITH CH3 DEVICES

MPICH2 does not require a CH3 channel to implement any specific function to support one-sided communication. Instead, the only requirements for a fully-functional channel is to implement some book-keeping functions, four methods to start sending a new message resp. processing a send request and one function that represents the progress engine. As the latter one is responsible for receiving new messages, there are no explicit receive methods.

With these minimal requirements, a CH3 channel is able to process send and receive requests issued by point-to-point operations like MPI_SEND and MPI_RECV as well as collective routines such as MPI_BCAST or MPI_GATHER. Moreover, the functions to be implemented by a channel also enable the usage of one-sided communication. This is made possible by the CH3 device which provides an implementation for the OSC-related functions of MPI resp. the ADI3 layer. This implementation is based on point-to-point operations. Thus, a channel device is not required to implement OSC-functions, since the CH3 device breaks the according calls down to an appropriate call sequence of point-to-point primitives.

The implementation of the OSC functions in the CH3 device basically relies on a queue. The operations issued by communication calls like MPI_PUT, MPI_GET and MPI_ACCUMULATE and initial locking methods like MPI_WIN_LOCK are stored in the queue. By calling a final synchronization method, like MPI_WIN_UNLOCK, the enqueued operations are sent to the target process by the same mechanism used for point-to-point communication.

The synchronization calls use this mechanism as well. For locking a window with MPI_WIN_LOCK, for example, a request to lock the window is sent to the target process. After that, the progress engine is invoked until a message is received which grants access to the lock.

To address hardware that offers support for OSC, e.g. by remote direct memory access (RDMA), the CH3 device allows channels to overwrite the default OSC implementation. This flexibility is achieved through function pointers within the internal structure that represents a remote memory access (RMA) window. During the window creation, any of these pointers can be overwritten by the CH3 channel implementation. Thereby, MPICH2 can be instructed to use the channel's implementation of OSC synchronization or communication calls.

IV. OSC SUPPORT ON THE SCC

MPICH2 had been ported to the Single-Chip Cloud Computer (SCC) by implementing three different CH3 channels. The port is named RCKMPI and is based on MPICH2 1.2.1p [5]. All of the SCC-specific CH3 channels use shared memory which is either backed by the message passing buffer (MPB), the external memory or both. For the latter case, the memory used for the messages is switched at a constant threshold. Although the memory used for message transport differs, the implementation is basically similar: The memory is partitioned into sections such as each process resp. core offers dedicated write areas for every other MPI process. Regarding bandwidth, the channel that uses the MPB delivers best performance.

As all channels implement the functions required by the CH3 device, all communication methods are expected to work. On the one hand, this is actually true for point-to-point operations. On the other hand, one-sided communication is not usable with the original RCKMPI implementation. For instance, the SCC-specific CH3 channels only cover some basic cases of the MPICH2 test suite. That way, RCKMPI applications are likely to crash if this communication paradigm is used.

There are several reasons for this issue. First, the RCKMPI source code always calls the default message completion handler defined by MPICH2. Thus, RCKMPI does not account the case where a special completion call-back function has been associated with the send request. As those call-backs are only defined when OSC-related messages should

be transferred, applications based on RCKMPI will show unpredictable behaviour when using OSC.

Even if the appropriate call-back is invoked on message completion, the channel implementation is not aware of request modifications caused by upper MPICH2 layers. These modifications are mainly applied when OSC messages are transferred. As a consequence, even when invoking the completion method, OSC applications are again likely to crash.

Therefore, the following implementation issues were fixed for the MPB CH3 channel implementation of RCKMPI: (a) Invoke the message completion call-back function when send and receive requests are assumed to be completed. (b) Check that the call-back handler confirms the completion of the request. (c) If the call-back signals an incomplete request, further process the unfinished request. (d) On message reception, copy its payload back to the according user buffers. The fixed source code is available to the MARC community [6].

V. EXPERIMENTAL RESULTS

Based on this work, a CFD-like cellular automaton [7] was used to compare the applications scalability on the SCC with an InfiniBand cluster. The cellular automaton is a 9-point-stencil computation. We tested two different versions of the application: one uses OSC and the other two-sided routines from the MPI standard. The programs use the communication patterns shown in Figure 1. The communication is performed after each time step: every MPI process has to exchange 8 KB of data with its two neighbor processes. The according messages are sent before the process starts to calculate the inner values of its field fraction to gain optimal computation-communication-overlap.

In the experiment, the application was executed with an increasing number of nodes while the size of the computed two-dimensional field was fixed. Since there are only 28 nodes in the InfiniBand cluster, the scaling was only analyzed up to a number of 28 nodes for both systems. Each node of both the SCC and the InfiniBand cluster executed a single MPI process. The runtime of the sequential version is divided by the runtime of parallel program running on n processors to obtain the speedup.

A. Experimental Environment

The InfiniBand cluster consists of 28 machines equipped with two Intel Xeon 5520 CPUs each providing four cores (Hyperthreading disabled). On each node, 48 GB of DDR3 main memory are installed. The InfiniBand connection is based on 20 Gb/s Mellanox MHGH19-XTC ConnectXZ cards and a Mellanox MTS3600R-1UNC switch. Further, OpenMPI 1.4.3 was used as an MPI-2 implementation that supports InfiniBand. During the experiments, OpenMPI's parameters were choosen such that OSC functions use InfiniBand's RDMA features. It is therefore not expected that the OSC application on the SCC scales better than on the InfiniBand cluster as there is no RDMA support in RCKMPI.

On the SCC, a modified Linux 2.6.38.3 kernel was used in conjunction with Busybox. For compiling the application and RCKMPI, GCC version 4.5.2 has been chosen. On the InfiniBand cluster, Scientific Linux with Kernel 2.6.18 and GCC 4.3.2 is installed.

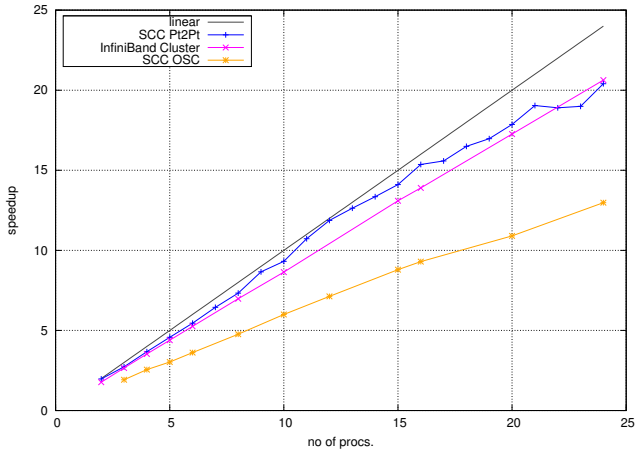


Fig. 3. Speedup of the CFD application compared to its sequential version on the SCC and the InfiniBand Cluster

B. Results on SCC

The obtained results are presented in Figure 3. The OSC version on the SCC shows linear scaling, but it is outperformed by the two-sided communication version on the SCC. The speedup values are about one third smaller than the two other versions and only one half of the optimal linear speedup.

The scaling of both the two-sided and the OSC version of the application can be attributed to the low latency communication network and the RCKMPI CH3 channel implementation. A downside of this implementation is the inherited handling of OSC by the CH3 device implementation. That is queueing all communication actions until a final synchronization call is issued (see Section III).

By processing the queue entries messages are sent for locking the window, transferring the data and finally unlocking the window. Thus, at least three MPI message are sent to the target process in a synchronous manner. Compared to the single non-blocking communication call (MPI_Isend) that is used in the point-to-point version of the application two additional messages are required. On the SCC, the two additional messages are sent with an handshake protocol between the communication processes, whereas on the InfiniBand cluster RDMA does not require any participation of the target process.

C. Comparison with InfiniBand Cluster

The OSC version running on the InfiniBand cluster reached approximately linear speedup (see Figure 3). This shows the benefit of an RDMA capable network which allows the implementation of communication libraries which support computation-communication-overlap.

VI. RELATED WORK

The authors of [8] report up to 30 % runtime improvement for an atmospheric simulation, when using MPI-2 one-sided

communication in combination with OpenMP instead of point-to-point communication. It is not clear if the improvement is due to the use of OpenMP or one-sided communication or the combination.

Different implementation options for MPI2-OSC were analysed in the literature regarding the different capabilities of the underlying networks [9]–[14]. While one would expect inefficiencies of OSC over transports without support for remote direct memory access (RDMA), interconnects like InfiniBand are expected to offer similar or better performance for OSC than for two-sided communication. However, asynchronous two-sided communication still offers the best performance if only a few number of communications are synchronised with one epoch [9], [10], [15].

In [14] the authors observe that one-sided communication performs much worse than two-sided communication for short and medium-sized messages. The reason is spotted in the overhead of synchronization functions.

The same observation is also made in [15], [16]. A comparison of two- and one-sided MPI2 communication over Gigabit Ethernet [15] shows that the design of the MPI2-OSC API is the key performance problem. The design of MPI2-OSC was compared with another OSC API called NEON in [16] on top of InfiniBand networks. While InfiniBand is an ideal network for computation-communication-overlap, the MPI2-OSC implementation again suffered from the additional overhead and was outperformed by NEON. The authors show that applications which use MPI-2-OSC suffer from the overhead of the additional synchronisation message that has to be sent in order to complete a remote memory access.

Currently, the MPI community discusses the MPI-3 standard [17]. The One-Sided communication interface proposed by the MPI-3 RMA Working group retains all of the calls from MPI-2, but adds new additional calls for window creation, synchronisation, and communication. Some of the proposed new features are implemented and investigated in [18]. Especially, the new Request-Based operations seem to be promising to achieve optimal computation-communication-overlap.

VII. CONCLUSION AND FUTURE WORK

In this paper, we gave insights on the implementation of One-Sided Communication in MPICH2 which is the base for the MPI implementation of the SCC – RCKMPI. Based on this insights, we identified missing functionalities within that implementation. These issues were fixed and the revised source code was made available to MARC members.

Further, the scalability of a CFD-like application based on the OSC implementation of RCKMPI was analyzed and compared to both a two-sided version on the SCC and a one-sided version on an InfiniBand cluster. It was revealed that all variants scale in a linear fashion, while the two-sided version on the SCC outperforms the two others. Moreover, the OSC variant shows poor scaling behavior on the SCC. On the other hand, the good scaling of the OSC version on the InfiniBand cluster is obviously due to the opportunity to use the RDMA features. This effect has already been observed when comparing the MPICH2 implementation with the lightweight OSC API called NEON both on Gigabit Ethernet [15] and InfiniBand installations [16].

Future work goes into two directions: First, the scaling of the OSC features of RCKMPI can be improved. As the SCC offers hardware support for defining and accessing shared memory areas, a MPI window for OSC might be defined with help of the hardware features: A window may reside in a shared memory which is backed either by the MPB or the external DDR3 main memory. Further, accesses of a MPI process resp. core could be supported by dedicated cores. For example one core per tile could be responsible for performing RMA operations while the other handles computational tasks of an application [19]. That way, both RMA and overlap of communication and computation would be made possible.

Second, it would be interesting to get experiences with applications using the Global Arrays toolkit on the SCC. This framework offers a "virtual shared memory programming interface." Moreover, the implementation is based on message passing libraries, such as MPICH2 and requires OSC features. Especially, application from the field of quantum chemistry are heavily using frameworks like Global Arrays [20].

REFERENCES

[1] Message Passing Forum, "MPI: A Message-Passing Interface Standard," Tech. Rep., 1994.

[2] ——, "MPI-2: Extensions to the Message Passing Interface," Tech. Rep., Jul. 1997.

[3] W. Gropp, "MPICH2: A new start for MPI implementations," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, ser. Lecture Notes in Computer Science, D. Kranzlmueller, J. Volkert, P. Kacsuk, and J. Dongarra, Eds. Springer Berlin / Heidelberg, 2002, vol. 2474, pp. 37–42.

[4] W. Gropp and E. Lusk, "MPICH abstract device interface version 3.3," Mathematics and Computer Science Division Argonne National Laboratory, Reference Manual Version 3.3, Dec. 2001, draft.

[5] I. Comprés Ureña, M. Riepen, and M. Konow, "RCKMPI – lightweight MPI implementation for Intel's Single-chip Cloud Computer (SCC)," in *18th European MPI Users' Group Meeting, EuroPVM/MPI 2011*, ser. Lecture Notes in Computer Science, Y. Cotronis, A. Danalis, D. Nikolopoulos, and J. Dongarra, Eds. Springer Berlin / Heidelberg, 2011, vol. 6960, pp. 208–217.

[6] S. Christgau, "MARC Bugzilla Bugreport 386: Missing support for one-sided communication," http://marcbug.scc-dc.com/bugzilla3/show_bug.cgi?id=386, Feb 2012.

[7] "The Cellular Automaton Application," http://www.cs.uni-potsdam.de/bs/cellularautomat/, 2006.

[8] A. A. Mirin and W. B. Sawyer, "A scalable implementation of a finite-volume dynamical core in the community atmosphere model," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 3, pp. 203–212, 2005.

[9] B. Barrett, G. M. Shipman, and A. Lumsdaine, "Analysis of Implementation Options for MPI-2 One-Sided," in *Proceedings of the 14th European PVM/MPI Users' Group Meeting*, ser. Lecture Notes in Computer Science, vol. 4757. Springer-Verlag, September/October 2007, pp. 242–250.

[10] W. Gropp and R. Thakur, "An evaluation of implementation options for mpi one-sided communication," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface 12th European PVM/MPI User's Group Meeting*, D. M. et al., Ed. Sorrento, Italy: Springer-Verlag, September 2005, pp. 415–424.

[11] W. Huang, G. Santhanaraman, H.-W. Jin, and D. K. Panda, "Design alternatives and performance trade-offs for implementing mpi-2 over infiniband," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface 12th European PVM/MPI User's Group Meeting*, D. M. et al., Ed. Sorrento, Italy: Springer-Verlag, September 2005, pp. 191–199.

[12] T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, "High Performance RDMA Protocols in HPC," in *Proceedings of the 13th European PVM/MPI Users' Group Meeting*, ser. Lecture

Notes in Computer Science, vol. 4192. Berlin, Heidelberg: Springer-Verlag, September 2006, pp. 76–85.

[13] J. Liu, W. Jiang, H. Jin, D. Panda, W. Gropp, and R. Thakur, "High performance mpi-2 one-sided communication over infiniband," in *International Symposium on Cluster Computing and the Grid (CCGrid 04)*, April 2004.

[14] Rajeev Thakur and William Gropp and Brian Toonen, "Optimizing the Synchronization Operations in MPI One-Sided Communication," *High Performance Computing Applications*, vol. 19, no. 2, pp. 119–128, 2005.

[15] L. Schneidenbach and B. Schnor, "Design issues in the implementation of MPI2 one sided communication in Ethernet based networks," in *PDCN'07: Proceedings of the 25th conference on Parallel and Distributed Computing and Networks (PDCN)*. ACTA Press, Feb. 2007, pp. 277–284.

[16] L. Schneidenbach, D. Böhme, and B. Schnor, "Performance Issues of Synchronisation in the MPI-2 One-Sided Communiction API," in *15th European PVM/MPI Users' Group Meeting*. Dublin, Ireland: Spinger, Lecture Notes in Computer Science 5205, 2008, pp. 177 – 184.

[17] MPI 3.0 Standardization, "MPI: A Message-Passing Interface Standard, Draft," http://meetings.mpi-forum.org/MPI_3.0_main_page.php, Nov. 2010.

[18] S. Potluri, S. Sur, D. Bureddy, and D. K. Panda, "Design and implementation of key proposed mpi-3 one-sided communication semantics on infiniband," in *Proceedings of the 18th European MPI Users' Group conference on Recent advances in the message passing interface*, ser. EuroMPI'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 321–324. [Online]. Available: http://dl.acm.org/citation.cfm?id=2042476.2042514

[19] M. W. van Tol, R. Bakker, M. Verstraaten, C. Grelck, and C. R. Jesshope, "Efficient memory copy operations on the 48-core intel scc processor," in *MARC Symposium*, D. Göhringer, M. Hübner, and J. Becker, Eds. KIT Scientific Publishing, Karlsruhe, 2011, pp. 13–18.

[20] J. Dinan, P. Balaji, J. R. Hammond, S. Krishnamoorthy, and V. Tipparaju, "Supporting the Global Arrays PGAS Model Using MPI One-Sided Communication," in *IPDPS 2012*, Shanghai, China, May 2012, in press.