Implementation and Analysis of Nonblocking Collective Operations on SCI Networks

Christian Kaiser
Dolphin Interconnect Solutions
Siebengebirgsblick 26,
53343 Wachtberg, Germany
kaiser@dolphinics.com

Torsten Hoefler
Open Systems Laboratory,
Indiana University
150 S Woodlawn Ave,
Bloomigton, IN 47405, USA
htor@cs.indiana.edu

Boris Bierbaum, Thomas Bemmerl
Chair for Operating Systems,
RWTH Aachen University
Kopernikusstr. 16,
52056 Aachen, Germany
{*bierbaum,bemmerl*}*@lfbs.rwth-aachen.de*

Abstract

Nonblocking collective communication operations are currently being considered for inclusion into the MPI standard and are an area of active research. The benefits of such operations are documented by several recent publications, but so far, research concentrates on InfiniBand clusters. This paper describes an implementation of nonblocking collectives for clusters with the Scalable Coherent Interface (SCI) interconnect. We use synthetic and application kernel benchmarks to show that with nonblocking functions for collective communication performance enhancements can be achieved on SCI systems. Our results indicate that for the implementation of these nonblocking collectives data transfer methods other than those usually used for the blocking version should be considered to realize such improvements.

1. Introduction

The Scalable Coherent Interface (SCI) [1] is an interconnect technology for clusters which is currently distributed as “Dolphin Express D” by Dolphin Interconnect Solutions. A part of a cluster node’s physical memory can be mapped into the global SCI address space and can be accessed from a remote node via CPU load and store operations to mapped memory. While in this so called “PIO mode” a data transfer stalls the sender’s CPU, CPU offload can be achieved by using the DMA engine on the Dolphin Express D cards.

Dolphin Express D networks are composed of ringlets (essentially a 1d torus) that can be combined to build tori of higher dimensions (2d and 3d are supported). Especially for collective communication with large messages, it is important to avoid contention of the SCI links. Therefore, we have implemented algorithms that are tailored for torus networks and take the network topology into account (Section 3).

1.1. Nonblocking Collective Communication

Nonblocking collective operations [2] offer an interface that enables the application programmer to start and complete a collective communication operation independently. A nonblocking interface to all Message Passing Interface (MPI [3]) collectives is discussed in the MPI Forum for inclusion in the MPI 3 standard. Implementations typically divide the operation in an initialization call (e.g., MPI_Ibcast()), which only starts the communication but does not depend on other processes, and a blocking (e.g., MPI_Wait()) or nonblocking (e.g., MPI_Test()) test for completion of the operation. This scheme enables overlapping communication with computation if some computation can be performed while the communication is running.

Such an optimization has been discussed for parallel applications [4]. A particular example with a three-dimensional Poisson equation showed a performance improvement of 34% by applying nonblocking collective operations [5]. The runtime of a strong-scaling medical image reconstruction algorithm [6] could be improved up to 8%.

However, a nonblocking interface does not mean that the operations are actually performed asynchronously in the background. Asynchronous execution of collective operations is a complex task that requires careful hardware-dependent optimization. Such an optimization for InfiniBand networks is described in [7].

Our work extends this line of research towards another interesting HPC network which allows to choose between Programmed I/O (PIO) and Direct Memory Access (DMA) for the host-to-host communication: The Scalable Coherent Interface (SCI). We wanted to investigate whether this very different network, as compared to InfiniBand, also offers comparable performance improvements via the usage of nonblocking collectives and how such operations should best be implemented on SCI networks.

1.2. Related Work

Several research groups begin the implementation of optimized nonblocking collective operations. The baseline is set with LibNBC [2] which only requires MPI to run. However, its algorithms are generic, without special hardware optimizations. LibNBC fully relies on the MPI library's asynchronous progression. An optimized version of the library was implemented for InfiniBand [7] and shows significant better performance for the communication time as well as asynchronous progression. IBM's Component Collective Messaging Interface (CCMI [8]) also implements optimized nonblocking collective operations for different architectures.

Several research groups have investigated blocking collective communication operations on SCI clusters. [9] describes the combination of DMA and PIO in pipelining algorithms for Broadcast, Reduce, and Allreduce. We used the ideas presented in [10] to tailor our algorithms for torus networks. [11], [12], [13] describe an SCI optimized MPI implementation, including algorithms and benchmark results for collective operations. To the best of our knowledge, the implementation of nonblocking collective operations on SCI networks has not been analyzed before.

2. Implementation

On our test cluster equipped with current-generation SCI hardware (see Section 4), a CPU store of 4 Byte to a remote memory location stalls the CPU for 200 ns if the I/O pipeline is fully saturated. On idle, it just stalls the processor for the duration of that single instruction. It takes between 1.3 and 1.4 μs for this data to arrive on the remote node and we have measured a maximum throughput of 325 MiB/s with PIO transfers.

Using the DMA engine adds a constant overhead to each data transfer for setting up the DMA queue and requires an additional local data transfer at the receiver's side. In our measurements, DMA transfers achieved a maximum throughput of 195 MiB/s and an 8 Byte transfer (the minimum possible size) took 34.5 μs. Although using the DMA engine seems to be a natural choice when implementing nonblocking collectives to achieve overlap between calculation and communication, the performance difference between PIO and DMA motivates the investigation of all implementation alternatives.

In fact, NMPI, an MPICH2-based MPI implementation for SCI clusters, uses polling (PIO) to detect incoming messages and does not utilize the DMA engine, even for nonblocking point-to-point operations. This eliminates the possibility of overlap completely [14]. Polling may be avoided by exploiting remote interrupts, which are triggered by a small-latency remote write that notifies the blocked receiver. The "SuperSockets" kernel module from Dolphin, which we used for benchmarking together with Open MPI's TCP Byte Transport Layer (BTL), does not use DMA, but PIO and remote interrupts to provide low-latency, high-throughput STREAM sockets over SCI [15].

2.1. The SCI Collectives Library

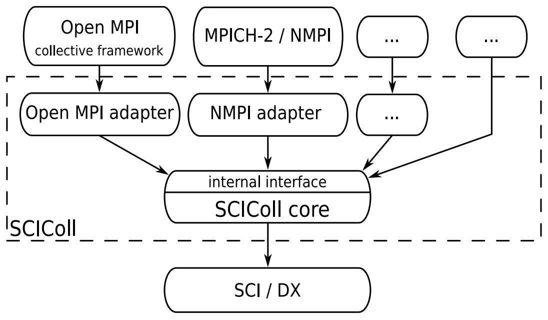


Figure 1. Architecture of the SCI Collectives Library

As a tool to evaluate our implementations, we used the SCI Collectives library (SCIColl) [16]. Fig. 1 shows its position between higher-level software, coupled via adapter modules, and the lower-level SISCI (*Software Infrastructure for SCI*) [17] interface to the SCI interconnect. Among other functionality, SISCI provides operations to import and export memory regions for PIO and to setup DMA transfers.

So far, we had already implemented several blocking collective operations and adapter modules for Open MPI and NMPI to give these MPI implementation access to these optimized functions. The adapter modules also provide a registration interface for point-to-point operations, such that the SCI Collectives library can use these functions from the MPI libraries. We added nonblocking collectives to the SCIColl core plus a new adapter module for the LibNBC interface to be able to use the benchmarks described in Section 4.

2.2. Implementation Considerations

We evaluated several alternatives in order to find the optimal implementation for nonblocking collective operations over SCI networks. This also allows us to compare our implementation decisions to those that have been made while implementing nonblocking collective operations on InfiniBand networks, a technology that is significantly different to SCI.

As the main purpose of nonblocking communication operations is the overlapping of computation and communication, using a communication co-processor to offload the CPU seems a natural choice when implementing nonblocking collectives. This choice is certainly being made on InfiniBand networks, but given the shortcomings of the DMA engines on our SCI cards, as described in Section 2, the question was whether PIO transfers were preferable in this case even if this eliminates the possibility of overlap. Apart from overlap, parallel applications may benefit from a reduction of blocking wait times that occur because of process skew and network jitter. Minimizing these wait times with nonblocking collectives is independent of the transfer method.

Another open question was the use of parallel threads to achieve asynchronous progress, without a DMA engine. There are several things to say against a multi-threaded implementation of nonblocking collective operations [2], but we wanted to try this out nonetheless.

As a result of these considerations we implemented the following types of nonblocking collective operations in SCIColl:

- single-threaded with DMA transfers that needs manual progress
- single-threaded with PIO transfers that needs manual progress
- additional communication thread with DMA transfers
- additional communication thread with PIO transfers

3. Collective Operations

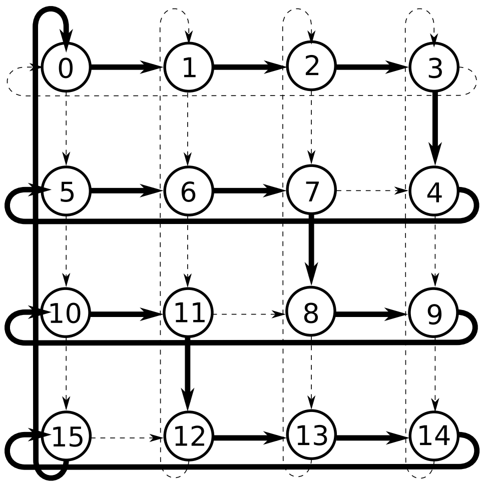
So far, we have nonblocking block and vector variants of Gather and Alltoall available in the first version of the SCIColl Library. The rationale for choosing these communication patterns first was the availability of application kernel benchmarks that use these in a nonblocking manner (Section 4.2). We will explain our implementation choices in the following.

3.1. Gather

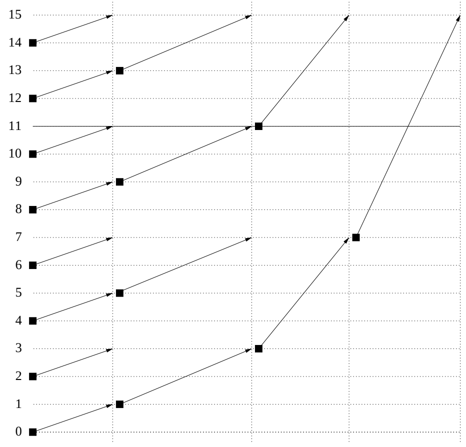
We have implemented four gather algorithms: *Binary Tree*, *Binomial Tree*, *Flat Tree* (essentially all processes sending to the root process at the same time, without any coordination), and *Sequential Transmission*, which is like *Flat Tree*, but the messages are sequentialized by the root processes sending a signal to the other processes to trigger the data send operation.

Our gather algorithms communicate in a way that is aware of the torus topology of the SCI network. The basic idea is the association of the processes to virtual ranks that reflect the processes' order in a Hamiltonian Path [10] through the 2d torus as shown in Fig. 2 (a). Because the SCI links are unidirectional, sending, for example, from node 0 to node 2 and from node 1 to node 3 concurrently may cause contention of the link from node 1 to node 2 in the case of large messages. Our Binomial Tree gather algorithm as shown in 2 (b) communicates in a way that avoids sending two messages over the same link during any step. We ignore other traffic, such as acknowledgment packets here, but this is feasible for large messages as the data packets are much longer than packets of any other type in this case.

For the vector variant we implemented a *Flat Tree* and *Sequential Transmission* only. The *Sequential Transmission* has been implemented as a DMA and a PIO variant. All other variants are PIO-only implementations. As nonblocking variants, we support all blocking variants in an additional communication thread and the *Sequential Transmission* with an additional progress functionality in a single thread.



(a) Hamiltonian Path



(b) Binomial Gather

Figure 2. Avoiding Link Contention in a 4x4 SCI Torus

3.2. Alltoall

For the Alltoall communication scheme, we considered four approaches: The algorithm developed by *Bruck* et al. [18], a *Pairwise Exchange* in $N-1$ steps if N is an even number of processes [19], a *Ring* algorithm with recursive doubling in which a Hamiltonian Path is used to established the order of the processes in the ring [11], [19], and a *Flat*

Tree function that just posts all send and receive operations and waits for completion.

All four variants are available as blocking functions and as nonblocking functions using an additional communication thread. Out of these, the *Pairwise Exchange* algorithm has been implemented with DMA as a single-threaded, nonblocking function. The same holds true for the vector variant, except that we do not have functions using the *Bruck* or *Ring* algorithms available here.

4. Benchmark Results

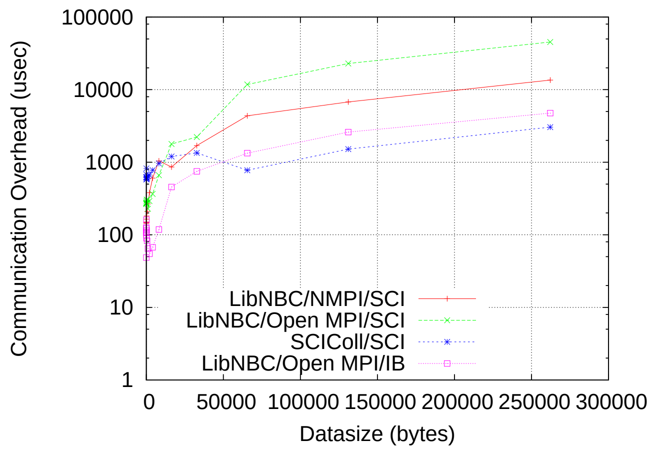
Benchmark results were obtained at the Chair for Operating System's PD Cluster, consisting of 16 nodes in a 4x4 2d SCI torus, each equipped with a single Pentium D (dual core) processor running at 2.8 GHz, 2 GiB RAM and a D352 SCI card from Dolphin. The nodes are also attached to an InfiniBand Switch (x4, DDR) via Mellanox MHGS18-XT HCAs. We used Open MPI 1.2.8, NMPI 1.3.1, and NBCBench 1.0.

4.1. Synthetic Benchmark Results

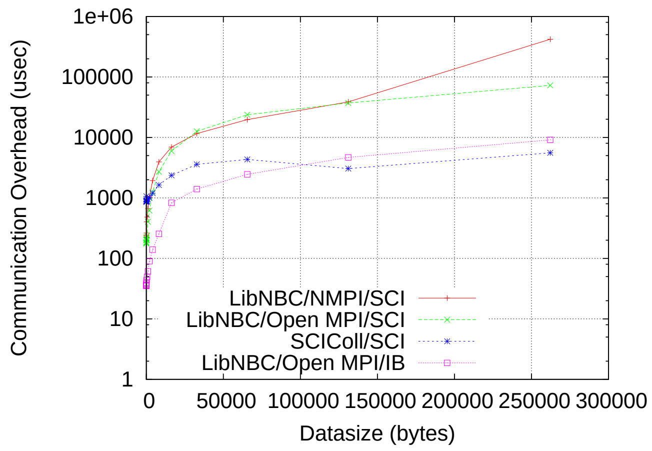
First, we discuss some microbenchmarks in order to assess the minimal communication overhead of the different implementations. In Figure 3, we compare the overheads of:

- **LibNBC/NMPI/SCI** LibNBC over NMPI (SISCI),
- **LibNBC/Open MPI/SCI** Open MPI's TCP BTL over SuperSockets,
- **SCIColl/SCI** the SCIColl library over SCI (SISCI),
- **LibNBC/Open MPI/IB** Open MPI's openib BTL over InfiniBand

We used NBCBench to gather these overhead results. NBCBench starts a nonblocking operation, and tries to overlap the communication with computation. Figure 3 shows the remaining overheads (nonoverlappable communication parts) for Gather and Alltoall. We see that NMPI and the SuperSockets implementation allow very little or no overlap. The SCIColl library shows significant performance improvements by optimizing the overlap of communication and computation. To achieve this, it uses the *Sequential Transmission* Algorithm for Gather and the *Pairwise Exchange* Algorithm for Alltoall with DMA transfers. Both were executed in a single thread and progressed every 2048 bytes. In comparison to the faster InfiniBand, we see that the SCIColl library enables higher overlap than LibNBC with Open MPI. However, we also see in the experiments experiments, that the InfiniBand-optimized LibNBC performs better than any SCI configuration. This is mostly due to the higher bandwidth and highly optimized DMA implementation of the InfiniBand network. In the following, we analyze the different SCI configurations with several application kernel benchmarks in the next section.



(a) Gather



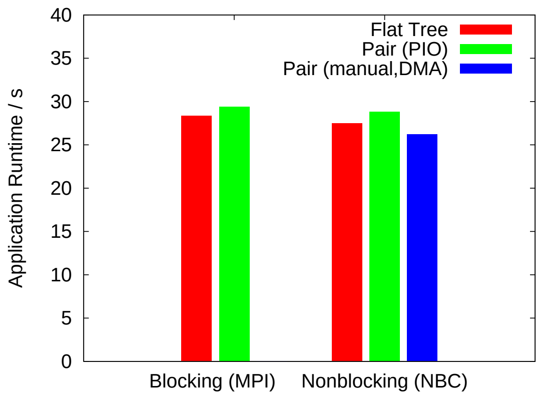
(b) Alltoall

Figure 3. Communication Overhead measured with NBCBench

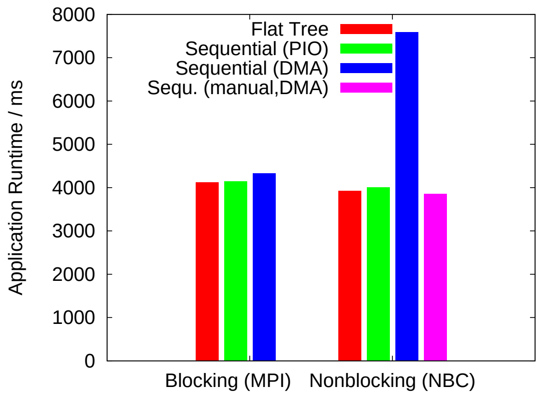
We would expect a constant overhead for SCIColl with growing message-sizes as the DMA setup costs are constant for all message sizes. Since we test for progress every 2048 bytes we add an overhead that grows with the message size.

4.2. Application Benchmark Results

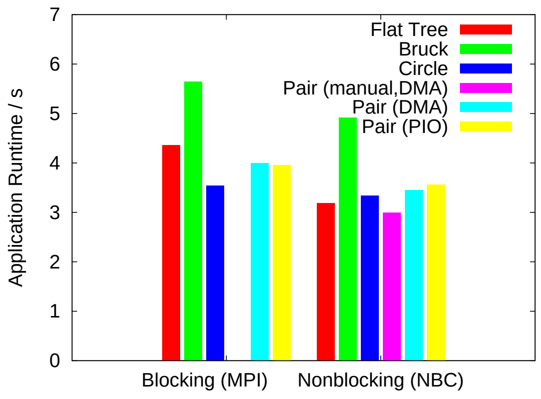
We used three application kernel benchmarks to show the feasibility of using nonblocking collective operations on SCI networks: a Conjugate Gradient Solver (CG) [5], a parallel Compression benchmark (PC) [2], and a three-dimensional Fast Fourier Transform kernel (FFT) [20]. In the CG benchmark, each process overlaps computation with the exchange of halo zones with its six neighboring processes in a three-dimensional grid. This Communication is performed with a blocking or nonblocking Alltoallv operation. PC compresses data in parallel and gathers the results with a blocking or nonblocking Gatherv operation (pipelined). The FFT kernel performs a parallel matrix transpose with a blocking or nonblocking Alltoall operation. A three-dimensional FFT is split into three one-dimensional FFTs and the data redistribution for the z transformation is overlapped with the transformation in the other two dimensions which do not require communication among processes.



(a) CG



(b) PC



(c) FFT

Figure 4. Comparison of different Algorithms with Application Kernel Benchmarks

These benchmarks compare blocking MPI collectives with nonblocking LibNBC collectives. Fig. 4 shows the results for 32 processes, comparing the different algorithms we implemented. Unless marked with “manual” (for “manual progression”), the nonblocking algorithms are just like their blocking counterparts, but executed in an additional communication thread. The following findings can be drawn from Fig. 4:

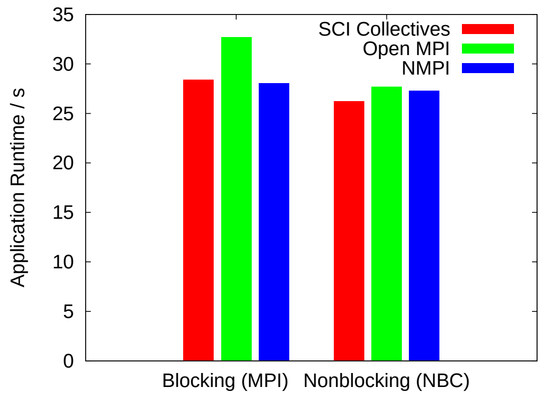
- Moving the collective communication to an additional thread almost always increased the application performance, although with 32 application processes and PIO transfers, this led to having more active threads than processor cores.
- As the algorithms stayed the same, moving them into a separate communication thread should preserve the relative performance among them. But there are a few exceptions to this rule.
- For all three collective operations measured, the single-threaded implementation, i.e., one that requires manual progress, using the DMA engine turned out to be the best choice for the SCIColl library in the nonblocking case, whereas for the blocking versions, PIO based implementations performed best. That means that using the DMA engine should be considered seriously when implementing nonblocking collectives on SCI networks, although it is usually not used for blocking collectives.

Please note that we had some issues with the SCI card’s DMA engine that occasionally caused the FFT and CG runs to last extremely long, the results shown here are best case results.

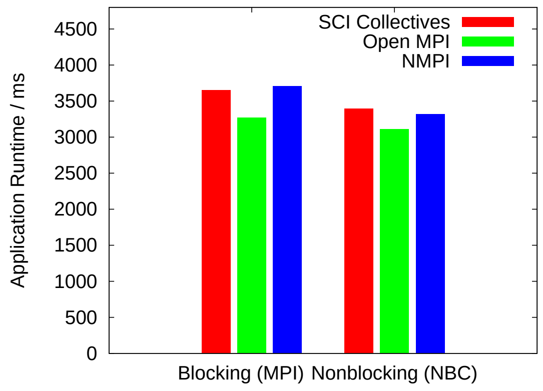
Fig. 5 compares the SCIColl library with Open MPI (communicating via TCP over SuperSockets) and NMPI. For our implementation, only the results for the best-performing algorithms are shown. Neither Open MPI nor NMPI communicates with DMA here. For all three benchmarks and all three communication libraries, the benefits of using nonblocking collectives are clearly visible. This is especially true for the CG and FFT (using Alltoall(v)) results for Open MPI, we attribute this to the combination of PIO and remote interrupts inside of SuperSockets. The results for the SCIColl library in Fig. 5 (b) differ from those in Fig. 4 (b) because they are from runs with different system software versions. We still need to investigate the reason for the comparably good results of Open MPI in the PC benchmark (using Gather).

5. Conclusion and Outlook

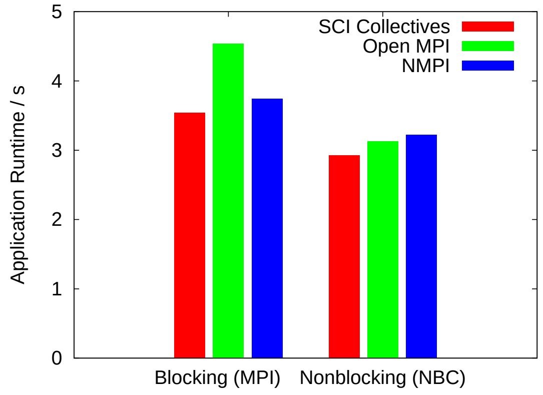
We show that parallel applications may benefit from the use of nonblocking collective communication operations on SCI clusters. We evaluated different implementation alternatives for such operations. Although usually, including the implementation of blocking collective operations, Programmed



(a) CG



(b) PC



(c) FFT

Figure 5. Application Kernel Benchmark Results over SCI

I/O is used for data transfer over current-generation SCI hardware, because it provides significantly lower latency and higher throughput, compared to DMA transfers, according to our benchmark results, DMA is preferable for nonblocking collectives because it acts as a communication co-processor.

Our implementation may provide a blueprint for future nonblocking collective communication functions in NMPI in case the MPI standard includes such functions in the future. From a research perspective, we plan to support the new DX interconnect from Dolphin [21] with our SCI Collectives library. The DX technology provides significant improvements in terms of latency, throughput and DMA support, compared to SCI.

References

[1] IEEE, “Scalable Coherent Interface (SCI), Std 1596-1992.”

[2] T. Hoefler, A. Lumsdaine, and W. Rehm, “Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI,” in *In proceedings of the 2007 International Conference on High Performance Computing, Networking, Storage and Analysis, SC07*. IEEE Computer Society/ACM, Nov. 2007.

[3] MPI Forum, “MPI: A Message-Passing Interface Standard. Version 2.1,” June 23rd 2008, `www.mpi-forum.org`.

[4] T. Hoefler, P. Gottschling, and A. Lumsdaine, “Leveraging Non-blocking Collective Communication in High-performance Applications,” in *SPAA’08, Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures*. Association for Computing Machinery (ACM), Jun. 2008, pp. 113–115.

[5] T. Hoefler, P. Gottschling, A. Lumsdaine, and W. Rehm, “Optimizing a Conjugate Gradient Solver with Non-Blocking Collective Operations,” *Elsevier Journal of Parallel Computing (PARCO)*, vol. 33, no. 9, pp. 624–633, Sep. 2007.

[6] T. Hoefler, M. Schellmann, S. Gorlatch, and A. Lumsdaine, “Communication Optimization for Medical Image Reconstruction Algorithms,” in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 15th European PVM/MPI Users’ Group Meeting*, vol. LNCS 5205. Springer, Sep. 2008, pp. 75–83.

[7] T. Hoefler and A. Lumsdaine, “Optimizing non-blocking Collective Operations for InfiniBand,” in *Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Apr. 2008.

[8] S. Kumar, G. Dosza, J. Berg, B. Cernohous, D. Miller, J. Ratterman, B. Smith, and P. Heidelberger, “Architecture of Component Collective Messaging Interface,” in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 15th European PVM/MPI Users’ Group Meeting*, vol. LNCS 5205. Springer, 9 2008.

[9] J. Worringen, “Pipelining and Overlapping for MPI Collective Operations,” in *Proceedings of the Workshop on High-Speed Local Networks (HSLN), in conjunction with 28th Annual IEEE International Conference on Local Computer Networks (LCN 2003)*, Bonn/Königswinter, Germany, October 2003, pp. 548–557.

[10] S. Oral and A. George, “Multicast Performance Analysis for High-Speed Torus Networks,” in *LCN ’02: Proceedings of the 27th Annual IEEE Conference on Local Computer Networks*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 619–628.

[11] L. P. Huse, “Collective Communication on Dedicated Clusters of Workstations,” in *Proceedings of the 6th European PVM/MPI Users Group Meeting 1999 (EuroPVM/MPI)*, Barcelona, Spain, September 1999.

[12] ——, “MPI optimization for SMP based clusters interconnected with SCI,” in *Proceedings of the 7th European PVM/MPI Users Group Meeting 2000 (EuroPVM/MPI)*, Lake Balaton, Hungary, September 2000.

[13] L. P. Huse, K. Omang, H. Bugge, H. Ry, A. T. Haugsdal, and E. Rustad, “ScaMPI - Design and Implementation,” *SCI: Scalable Coherent Interface. Architecture and Software for High-Performance Clusters, Vol. 1734 of LNCS*, 1999.

[14] T. Hoefler, A. Lichei, and W. Rehm, “Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks,” in *Proceedings of the 21st IEEE International Parallel & Distributed Processing Symposium*. IEEE Computer Society, Mar. 2007.

[15] F. Seifert and H. Kohmann, “SCI SOCKET - A Fast Socket Implementation over SCI,” [Available on WWW at http://www.dolphinics.com].

[16] B. Bierbaum, G. Wassen, S. Lankes, and T. Bemmerl, “Evaluation of Optimized Barrier Algorithms for SCI Networks with Different MPI Implementations,” in *Proceedings of the 3rd Workshop on Communication in Cluster- and Grid-Systems (KiCC, Kommunikation in Clusterrechnern und Clusterverbundsystemen)*, RWTH Aachen University, Germany, December 2007. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:hbz:82-opus-21137

[17] *Low-level SCI software functional specification, Version 2.1.1*, Dolphin Interconnect Solutions, March 1999.

[18] J. Bruck, C.-T. Ho, S. Kipnis, and D. Weathersby, “Efficient algorithms for all-to-all communications in multi-port message-passing systems,” in *SPAA ’94: Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM, 1994, pp. 298–309.

[19] R. Thakur, R. Rabenseifner, and W. Gropp, “Optimization of Collective Communication Operations in MPICH,” *International Journal of High Performance Computer Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[20] T. Hoefler, P. Kambadur, R. L. Graham, G. Shipman, and A. Lumsdaine, “A Case for Standard Non-Blocking Collective Operations,” in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, EuroPVM/MPI 2007*, vol. 4757. Springer, Oct. 2007, pp. 125–134.

[21] V. Krishnan, “Evaluation of an Integrated PCI Express IO Expansion and Clustering Fabric,” in *Proceedings of the 16th IEEE Symposium on High Performance Interconnects*, Stanford University, CA, August 2008.