

Concepts for Integrating SISCO into Open-MPI

Torsten Mehlan, Torsten Hoefler, Frank Mietke, Wolfgang Rehm
{*tome, htor, miei, rehm*}@cs.tu-chemnitz.de

Chemnitz University of Technology

Abstract. Since version 2 of the Message Passing Interface (MPI) the MPI job has the ability to start new processes dynamically at runtime. The MPI implementation has to care for the creation of new communication channels in those situations. This work describes the implications of dynamic process creation on an communication module for the Scalable Coherent Interface (SCI). The module is developed for the PTL framework of Open MPI.

1 Introduction

The Open MPI project aims at unifying the various efforts of writing performant MPI libraries. Many researchers focus on a particular aspect of the semantics of the Message Passing Interface (MPI) standard. Thus many different MPI implementations have emerged. To unify those efforts the Open MPI project provides a flexible and extensible framework allowing to add new components to the existing subsystems. Until now there is no SCI support for Open MPI. This work examines an important aspect for the integration of SCI into Open MPI.

As many other high-speed interconnects SCI requires locking of memory areas to ensure that no side effects occur due to virtual memory management. To prepare the communication memory properly the size of the memory area has to be known in advance. During the operation of MPI there might occur cases where the special memory areas have to be scaled. Thus the MPI library has to reallocate this memory to provide sufficient space for control messages and user data. Since the MPI implementation can not rely on the availability of multiple SCI memory areas all communication has to use a single memory segment at each MPI process.

This work describes a mechanism to synchronize the access to SCI memory segments without affecting the performance of conventional point-to-point messages. We explain the condition under which SCI memory has to be reallocated and how to achieve synchronization in these cases. The remaining paper is organized as follows. Section 2 gives a summary of the properties of SCI and the semantics of the interface. Section 3 introduces the Open MPI framework and the PTL module interface. The mechanism of synchronizing access to SCI memory is discussed 4. Finally section 5 gives a conclusion.

2 SCI

The SCI network provides unique capabilities to cluster computers and networks of workstations. The interface is based on pure shared memory that can be created between distinct computers. Access to remote memory results in a network transaction which is about 2 μ s latency for 1 Byte messages. Due to the very short latency of messages the SCI technology is also attractive for applications using message passing.

2.1 SISCI

The "Software Infrastructure for SCI" (SISCI) interface [2] provides functions to manage shared memory and other resources of the SCI subsystem. The main resource types are shared memory segments, DMA (Direct Memory Access) resources and remote interrupts. Usually the shared memory segment is allocated by the SISCI library and has to be published to other computers by an explicit function call. Subsequently the shared memory may be mapped into the local address space of a process. Remote computers have to connect to shared memory segments to initiate data transfers.

To signal changes in the status of a shared memory segment the SISCI library creates asynchronous events. The application uses a callback mechanism to notice the occurrence of such an event. When the remote computer sets the memory segment unavailable or removes the segment any connected application receives the event to take appropriate actions.

2.2 Properties

To use SCI in a portable way functions of the SISCI interface provide several services to manage the appropriate resources as described in 2.1. The application can query some attributes of the underlying system such as suggested message sizes for DMA transfers. To enable data transmission a memory segment has to be created and specially prepared. Other applications on different computers connect to these segments and map the memory to virtual addresses. However, there are restrictions to the set of addresses where the mapping can take place. Also the association of memory allocated by means of "malloc" and SCI memory may fail. Thus any MPI library has to be prepared to hold a distinct SCI memory area. The availability of remote interrupt triggering may not be implemented for some SCI systems. The support of asynchronous event notification lacks some functionality on several systems. For instance one can not rely on the availability of notification of segment status changes through callbacks. Also the application can not rely on the availability of more than one SCI memory segment. Since creation and destruction of those memory appears as a time consuming operation, there is no way to create the appropriate memory areas on demand for each data transmission. In this case a single memory segment serves as target memory for copying user data. The zero-copy technique described by [6], [5] can not be used due to the lack of a sufficiently high number of supported memory areas. When an SCI memory segment is removed there must be no access to this segment, otherwise the behavior of the system is undefined.

All those restrictions must be taken into account when developing an SCI module for the popular MPI library Open MPI. Because it is unacceptable to invoke conventional TCP connections in the performance sensitive path every communication has to use SCI. In [4] the authors give a detailed discussion of the challenges to work around specific shortcomings of SISCI. This is done in the context of the Virtual Interface Architecture (VIA). First efforts to implement MPI on top of SCI networks are described in [7]. The work in [9] gives a discussion about the concepts of the integration of SCI into an MPICH library.

3 Open MPI

The Open MPI project aims at providing a high quality open source implementation of the MPI-2 specification. Open MPI leverages a modular architecture to distinguish several frameworks. Each framework covers a specific aspect. In [1] the authors give a detailed description of the concepts of Open MPI. The actual data transmission is split into a general point-to-point framework [8] and a framework for MPI collective communication. Other frameworks provide functions to support this data transmission. For instance there is a framework to manage memory registrations required by some networks to pin down memory pages.

3.1 PTL-Framework

The abbreviation PTL stands for point-to-point transmission layer. This framework defines a set of functions managing the connections and data transmissions between the processes of an MPI job. The functions of this framework provide means of establishing connections to other MPI processes, taking user data from buffers and performing actual data transmission. The transfer of data may be queued for later processing if resources are currently exhausted. The PTL module is responsible for synchronizing the access to the communication network in an appropriate way. Thus the occurrence of events affecting the basic communication resources has to be announced to the PTL modules of all involved MPI processes.

3.2 Dynamic Process Spawning

The MPI-2 specification features dynamic process management. Thus an MPI application has the ability to spawn multiple new MPI processes after the initial startup. The basic communication modules have to support this feature by means of establishing new connections on demand. When the application requests the creation of new processes, the PTL module has to create new connections and initialize the communication resources associated with the new processes. Using the SISCO interface this may result in changing the status of the shared memory segments and raises the need for explicit synchronization.

4 Strategies of Memory Reallocation

Assuming only one SCI memory segment is available for each node all the information has to be placed inside this memory area. Management data, such as flow control information as well as the data from an MPI envelope and the user data has to be written to the single memory segment of a given node. To avoid synchronization overhead each remote process owns a private area within the SCI segment of all other processes. The address and size of the appropriate shared memory region is known to both peers of each point-to-point connection. The private area inside the memory segment contains a header with management data. This data includes pointers determining the current status of a two ring buffers. The local buffer serves as container for user data written by the remote process to the local process. The local buffer also resides inside the local SCI segment. Even though the remote buffer is located at the remote process the local process maintains a copy of the current read and write pointers of the ring buffer.

The assignment of the appropriate SCI memory usually takes place during the initial setup. As new processes are created there is a need for additional space inside the SCI memory segment. Each new process needs a private area inside the SCI memory. Thus the SCI memory has to be reallocated. Each MPI process has to delete the existing memory segment, and has to create a new one of different size. Obviously this procedure may cause confusion if not all participating processes are aware of the situation.

The recreation of this SCI memory segment does not necessarily occur at all processes within the MPI job. This behavior comes due to the definition of `MPI_COMM_SPAWN`. This function spawns new MPI processes during runtime and may cause the actions mentioned above. Since not all members of the MPI job need to call this function there may be some processes participating in the collective routine, while other processes do not even notice the event. Thus it may happen that one process sends some data to a peer that currently aims at removing the SCI memory segment. To avoid this situation there has to be a point of synchronization.

To reduce the overhead as much as possible each process maintains a flag for all remote processes inside the local SCI memory segment. Before moving any data to a peer the sender reads the associated flag from local SCI memory. Only if the flag indicates no

special event the data transfer is safe. The flag is maintained on a per-process base to avoid the need for additional synchronization between multiple processes accessing the same flag. The flag has to be located close to the local CPU because the local process reads this memory location frequently. Reading remote SCI segments does not leverage the local CPU cache and behaves less performant.

The proposed flag is not sufficient to prevent from access to a disappearing SCI segment. During the time period between flag reading and writing of data the receiver may signal an event that would not be noticed. A handshake synchronization avoids this situation but leads to unacceptable performance. Thus the SCI communication module defines a specific time between setting the flag and actually removing the SCI segment. Thus there is a lower bound to the time the sender can assume data transmission to be safe. To calculate the value of this time period the SCI module determines the duration of data transfers. The size of this data transfer is determined by the current maximum size of the SCI segment divided by the number of processes. Processes reserve this size for peers to write data to their portion of local SCI memory. Since a single data transfer never involves more data this size serves as valid reference to calculate the lower time bound. Among the raw data transmission time the operating system introduces delays through scheduling decisions. Since there is no guarantee on the delays caused by conventional schedulers the SCI module can only use a raw estimation. This estimation can be done by examining the size of the time slice granted to each process at maximum. Moreover the SCI module multiplies this time by the number of currently running processes on a single machine to gain an estimation of the upper bound of the scheduling delay. Even if there are situations causing a larger delay this case never happens under normal operation of the system. Equation 1 summarizes the calculation.

$$t_{delay} = t_{maxdata} + t_{rrsched} * n \quad (1)$$

The value of $t_{rrsched}$ is specific to the operating system. For instance the function `sched_rr_get_interval(pid_t pid, struct timespec *tp)` returns the desired value. To get the value of $t_{maxdata}$ the maximum size of a single transfer can be divided by the bandwidth of the network.

4.1 Performance Impact

The following measurement uses the Intel Microbenchmarks (IMB) [3]. Two MPI processes perform a PingPong test using `MPI_SEND` and `MPI_RECV`. As visible from figure 1 there is no impact to the message latency in the case of no calls to `MPI_COMM_SPAWN`. The difference between both results is far below the measurement error. The impact on the execution time of `MPI_COMM_SPAWN` is expected to be significant. An estimation of the time needed by this call is given in figure 2. This figure plots the upper bound of transmission time for a value of $\frac{1}{100}s$ for $t_{rrsched}$ and $457\mu s$ for $t_{maxdata}$. The waiting time depends on the number of processes executing at one CPU. Since there are very rare cases of oversubscribing a single CPU with many processes the large wait times are expected to be not relevant to real world situations.

5 Conclusion

The definition of `MPI_COMM_SPAWN` in the MPI-2 specification implies the possibility of dynamically changing the set of processes belonging to a single MPI job. Thus during the execution of the application new connections have to be established on demand. The Open MPI implementation provides a flexible framework for implementing point-to-point transfers on top of specific network types. Point-to-point modules have to be prepared for this event. Using SISCO to implement the point-to-point framework requires for special

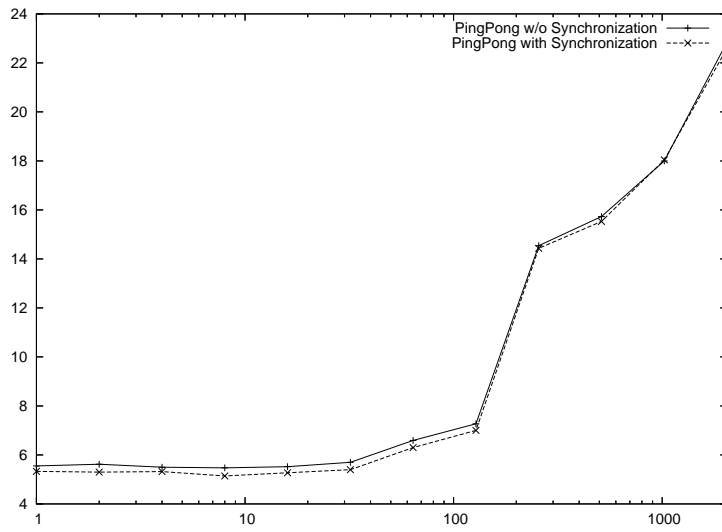


Fig. 1. Message latency for MPI_SEND with and w/o synchronization

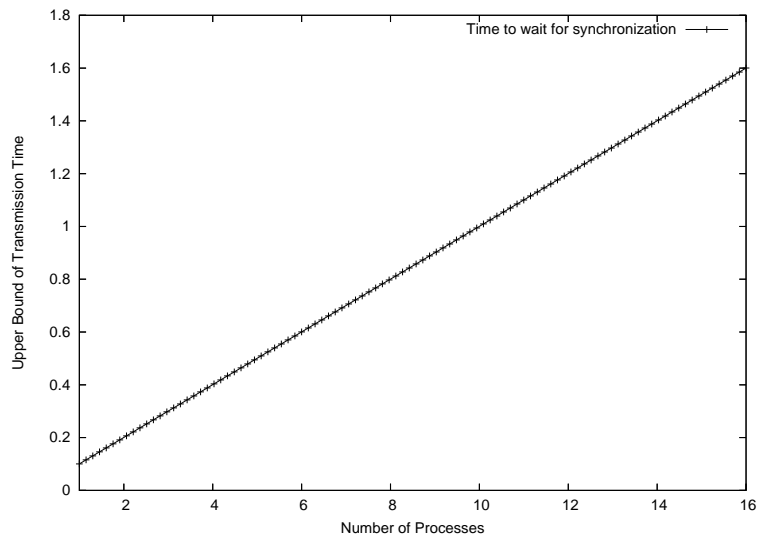


Fig. 2. Time to wait until all processes are expected to note the event

attention in the case of dynamically changing connections. Since some implementations of SISI restrict the number of available resources a special synchronization protocol is required. In this paper we proposed a way to synchronize the data transmission without affecting the performance in terms of message latency. We used an estimation of an upper bound of transmission time to ensure that no access is performed during changes in shared memory segments.

References

1. Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.
2. F. Giacomini, T. Amundsen, A. Bogaerts, R. Hauser, B. D. Johnsen, H. Kohmann, R. Nordstrøm, and P. Werner. *Low-level SCI software functional specification*, March 1999.
3. Intel GmbH Munich, Dornacher Strasse 1, D-85622 Feldkirchen, Germany. *Intel® Cluster Toolkit 2.0.1*.
4. Torsten Mehlan and Wolfgang Rehm. Via2sisci – a new library that provides the via semantics for sci connected clusters. In *Proceedings of 7th Workshop 'Parallel Systems and Algorithms' (PASA'04) held in conjunction with ARCS'04*, Augsburg, Germany, March 2004.
5. R. Rex. Analysis and evaluation of memory locking operations for high-speed network interconnects. Technical report, Chemnitz University of Technology, October 2005.
6. H. Tezuka, F. O'Carroll, A. Hori, and Y. Ishikawa. Pin-down cache: A virtual memory management technique for zero-copy communication. In *Proceedings of 12th Int. Parallel Processing Symposium*, March 1998.
7. J. Werner, L. Grabowsky, and T. Radke. Sci-mpi – an optimized implementation of a mpi subset for sci connected smp-systems. Technical report, Chemnitz University of Technology, 1997.
8. T.S. Woodall, R.L. Graham, R.H. Castain, D.J. Daniel, M.W. Sukalski, G.E. Fagg, E. Gabriel, G. Bosilca, T. Angskun, J.J. Dongarra, J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, and A. Lumsdaine. TEG: A high-performance, scalable, multi-network point-to-point communications methodology. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 303–310, Budapest, Hungary, September 2004.
9. Joachim Worringer and Thomas Bemmerl. Mpich for sci-connected clusters. In *Proceedings of SCI Europe '99, held in conjunction with EuroPar '99*, pages 3–11, Toulouse, France, September 1999.