Impact of On-Demand Connection Management in MPI over VIA*

Jiesheng Wu* Jiuxing Liu * Pete Wyckoff† Dhabaleswar Panda*

*Computer and Information Science
The Ohio State University
Columbus, OH 43210
{wuj, liuj, panda}@cis.ohio-state.edu

†Ohio Supercomputer Center
1224 Kinnear Road
Columbus, OH 43212
pw@osc.edu

Abstract

Designing scalable and efficient Message Passing Interface (MPI) implementations for emerging cluster interconnects such as VIA-based networks and InfiniBand are important for building next generation clusters. In this paper, we address the scalability issue in the implementation of MPI over VIA by on-demand connection management mechanism. The on-demand connection management is designed to limit the use of resources to what applications absolutely require.

We address the design issues of incorporating the on-demand connection mechanism into an implementation of MPI over VIA. A complete implementation was done for MVICH over both cLAN VIA and Berkeley VIA. Performance evaluation on a set of microbenchmarks and NAS parallel benchmarks demonstrates that the on-demand mechanism can increase the scalability of MPI implementations by limiting the use of resources as needed by applications. It also shows that the on-demand mechanism delivers comparable or better performance as the static mechanism in which a fully-connected process model usually exists in the MPI implementations. These results demonstrate that the on-demand connection mechanism is a feasible solution to increase the scalability of MPI implementations over VIA- and InfiniBand-based networks.

1. Introduction

In recent years, clusters of workstations have become both a popular and powerful environment on which to do parallel processing due to the tremendous improvement in network hardware and protocols, and the ever decreasing cost of commodity components. On these cluster systems, it is crucial to have an efficient communication system that can make effective use of the capability of the underlying network hardware and deliver the actual performance of the hardware to applications. A significant body of research in user-level communication protocols has been developed over the years.

The Virtual Interface Architecture (VIA) [9, 7] has been developed to standardize these protocol efforts. VIA defines an abstraction of network protocols and hardware capabilities that provides applications with direct access to the network, eliminates intermediate data copies using remote direct memory access (RDMA), and thus increases host processor application productivity.

More recently, the InfiniBand [13] architecture has been proposed to provide the next generation high-performance communication architecture for both inter-processor communication and I/O. InfiniBand incorporates many of the features of the VI architecture and draws on research efforts from high performance networked I/O. In the next several years, VIA/InfiniBand will be the standard architecture for high performance computing systems, servers, and clusters.

On these cluster systems, MPI [15] has become the *de facto* standard for developing portable parallel applications. Several MPI implementations built on VIA are currently available: MPI/Pro [8] based on GigaNet cLAN [10]; MVICH [14], a port of the generic MPICH implementation on M-VIA [1], GigaNet VIA and Servernet VIA [18]; and LAM/MPI [4] on M-VIA. Several complex design issues have been addressed in the literature describing these efforts and in overview papers [17]. However, the aspect of connection management has not yet been explored in detail.

MPI 1.2 does not specify a connection model. It assumes that all processes are fully connected after initialization. VIA is connection-oriented. So, for any pair of processes that will communicate, a VI endpoint must be created on each node and a connection between these two VI endpoints must be established beforehand. These connections between processes must be handled explicitly by the MPI library itself when implemented on the VI Architecture. In MVICH, for instance, each process creates $N - 1$ VI endpoints and then establishes $N - 1$ connections to other pro-

*This research is supported in part by Department of Energy's Grant # DE-FC02-01ER25506 and an NSF Grant #EIA-9986052.

cesses statically during MPI_Init(), where N is the number of processes in the MPI application. Thus, after initialization, there is a fully-connected network among all participating processes from the point of view of the VI layer. The total number of VI endpoints is $N \times (N-1)$, with the number of connections being half of that. This connection management mechanism is simple to implement; however, there are both scalability and performance problems in this *static connection mechanism*:

1. In large systems, the time to establish and to destroy a fully-connected process network is considerable and significantly affects the time to start and to terminate a parallel application. This is because connection setup is typically a costly operation with operating system involvement.

2. Beyond the basic connections required for message passing, connections are likely to be needed for other operations, such as for I/O operations on a parallel file system and for debugging. The number of connections supported in a specific VIA system serves as a hard limit to scaling. Thus, a fully-connected parallel application can easily exceed this limitation with only a moderate task size.

3. In an MPI implementation based on VI, each VI is associated with certain internal buffers and pre-preposted descriptors [7]. The amount of these resources for each process would ideally be a function of an application's communication pattern when scaling to large systems. However, in the static connection mechanism, it is a function of the number of processes in the application, regardless of what the application really needs. In reality, many large-scale scientific parallel applications do not require a fully-connected process model. Table 1 lists the average number of communication destinations per process in several applications [19]. Consequently, a large amount of resources is never used in these applications for the static connection mechanism approach. The estimation of 1024 processes is based on communication patterns described in [19] and implementation of collective communication using binomial tree algorithms [6].

4. The number of VIs and connections may have an impact on performance of the underlying VIA communication system, even if the endpoints are never used after setup, as discussed in [3] for Berkeley VIA [5].

An alternative approach that reduces the number of VIs and connections required by a large application is to establish connections when they are needed. We call this approach the *on-demand connection mechanism*. In this approach, the creation of two VI endpoints and establishment of a connection between them are on a per-use basis for any pair of processes, and undertaken only when it is known that they need a connection to pass messages.

Table 1. Average number of distinct destinations per process

Application	# of processes	Average # of destinations
sPPM	64	5.5
	1024	≤ 6
SMG2000	64	41.88
	1024	≤ 1023
Sphot	64	0.98
	1024	≤ 1
Sweep3D	64	3.5
	1024	≤ 4
Samrai4	64	4.94
	1024	≤ 10
CG	64	6.36
	1024	≤ 11

In this paper, we focus on designing and implementing an on-demand connection mechanism in MPI over VIA. Detailed performance evaluations are also presented. The main contributions of this paper are as follows:

- We have addressed the issues in incorporating the on-demand connection mechanism into an MPI implementation on VIA-based communication systems.

- We have implemented this on-demand connection mechanism in MVICH over cLAN VIA and over Berkeley VIA on Myrinet.

- We have shown that by using the on-demand connection mechanism, the resource consumption of parallel applications, such as the NAS parallel benchmarks, is dramatically reduced.

- We have conducted a series of performance evaluations. Performance results of both microbenchmarks and the NAS parallel benchmarks show that there is very little performance degradation after incorporating the on-demand connection mechanism in MVICH over cLAN VIA. The performance even increases for certain applications. The performance over Berkeley VIA on Myrinet is better than the static mechanism.

The rest of the paper is organized as follows. Background and related work are presented in section 2. Section 3 describes the design issues of incorporating the on-demand connection mechanism into a generic MPI implementation, while section 4 presents our implementation of the on-demand connection mechanism in MVICH. The performance results are presented in section 5, followed by conclusions and future work in section 6.

2. Background and related work

The Virtual Interface Architecture (VIA) [7] specifies the interface between high performance network hardware and computer systems. This architecture defines mechanisms to eliminate much of the protocol processing overhead and intermediate data copies by providing user applications a protected and directly accessible network interface called the Virtual Interface (VI).

A VI is a bi-directional communication endpoint. To use a VI, a point-to-point channel must be established between two VIs, in a process known as connection setup. Each channel supports the traditional two-sided send/receive model, as well as a one-sided remote memory access model. VI connection establishment is typically a costly operation with operating system involvement. Initially in version 0.95 of the VIA specification, only a *client-server connection model* [7] was defined. The server side waits for incoming connection requests and then either accepts them or rejects them based on the attributes associated with the remote VI. Since VIA specification 1.0, a *peer-to-peer model* has been included as well. In this model, either of the two processes to be connected can initiate the connection by calling a VIA peer-to-peer connection setup function. The connection is established after both of them have called the connection setup function.

There is some literature which analyzes the VI architecture and its ability to support parallel and distributed communication in general, and MPI implementations in particular. The inherent costs of using VI primitives to implement Active Messages and Split-C were analyzed in [2], while [8] described efficiency features of the MPI/Pro architecture gained by exploiting certain characteristics of the VI architecture. [12] compared the performance of Myrinet and GigaNet and their respective impacts on an MPI implementation. Their results indicated that the implementation of MPI is crucial for system performance. In [20], Wong *et al.* present a study of the scalability of the NAS Parallel benchmarks from the aspect of resident working set and communication performance. None of these works studied the connection scalability issues of the implementation of MPI over VIA nor the impact of connection management on application performance.

Brightwell *et al.* [17] analyze the scalability limitations of VIA in supporting the CPlant runtime system as well as any high performance implementation of MPI. The authors of this paper claim using qualitative analysis that the on-demand connection mechanism is not a good approach to increase the scalability of the MPI implementation; however, their analysis does not consider the impact of the number of connections on the underlying VIA communication system nor the impact of the allocation of large physical memory buffers on system performance. We argue that with efficient design and implementation, the on-demand connection mechanism can achieve comparable performance. Also, it does not require an extra thread to make progress, and it keeps the same properties of determinism, predictability, and fairness as the static mechanism.

3. Design

In the on-demand connection mechanism, the creation of VI endpoints and the establishment of a connection between two VI endpoints are performed strictly on a per-use basis for any pair of processes, and undertaken when they pass messages for the first time. Although conceptually simple, it is not trivial to incorporate it into current MPI implementations. In this section, we present the design issues in the on-demand connection mechanism.

3.1. Threading vs. Polling

The on-demand connection mechanism requires that an MPI process should be able to handle communication requests and connection requests simultaneously. The process cannot just block for communication, or just block for connection. It must be ready to handle both. Since VIA itself does not provide such a capability, this problem must be addressed explicitly in the MPI implementation.

Two alternatives can be used to solve the problem. The first one is to use a separate thread to handle all connection requests, while the main thread is dedicated to computation and established communication. In this way, communication progress can be ensured. However, a separate thread, which will incur a large overhead on context switch between the main thread and itself, may degrade application performance. Some MPI implementations, like MPI/Pro, are multi-threaded. For these implementations, it may be possible to use the existing second thread to handle connection requests. However, many MPI implementations, such as MPICH, are based on a single thread. These MPI implementations may not even be thread safe. Incorporating the on-demand connection mechanism into them by adding a separate thread will be very difficult.

Another method is using polling to handle both communication and connection requests. In a polling-based approach, the process checks periodically to see if there are pending communication or connection requests. This matches quite well with single threaded MPI implementations such as MPICH and it has very little overhead. In this paper, we have chosen such an approach and implemented it for MVICH on top of both cLAN VIA and Berkeley VIA.

3.2. Client/Server vs. Peer-to-Peer connection model

VIA provides two connection models: client/server and peer-to-peer. In theory both models can be used to implement an on-demand connection mechanism. However, we have found that the peer-to-peer connection model is generally better for the on-demand connection mechanism.

In the client/server connection model, one process acts as the server and the other acts as the client. The two processes have different actions during the connection setup procedure. Consequently, we have to be careful choosing the client and the server; otherwise deadlock situations may occur. The asymmetry of the client/server model also makes the implementation awkward.

In the peer-to-peer model, both processes involved in the connection setup behave similarly. And the order in which the two processes perform these actions does not matter. This matches quite well with the MPI communication model because either the message sender or the receiver can initiate communication setup. The symmetry of the model also makes the implementation task easier.

3.3. Progress rule

The progress rules of MPI [15] are both a promise to users and a set of constraints on implementors, though different interpretations seem to be possible. For example, MPICH [11] and MPI/Pro [8] follow a loose interpretation and a strict interpretation, respectively. In incorporating the on-demand connection mechanism into MPI for the VI Architecture, it is important to maintain this progress rule.

The on-demand connection mechanism can be incorporated into the above two typical implementations of MPI without violation of their respective progress semantics. The requirement of a "server" thread waiting to establish connections [17] depends on which interpretation of the MPI progress rule is provided. In MPICH, no thread is needed to maintain the loose interpretation of progress. In the case that there is a progress thread as implemented in MPI/Pro, this progress thread can take care of connection requests as well as communication requests.

3.4. Pre-posted send requests

In the on-demand connection mechanism, one important scenario must be carefully examined for both correctness and message delivery order. This scenario reflects the situation where an MPI application issues multiple nonblocking communication requests before the corresponding connection is established. For the VI Architecture, any requests posted into the Send Queue of a VI which is not yet connected are discarded, which would result in MPI message loss. To prevent this, pre-posted send requests must be stored if the corresponding connection is not yet established. To ensure the MPI message order rule, these pending requests should be processed in a first-in-first-out order when connections are available.

Note that all receive requests and send requests issued after the related connection is established can be handled directly.

3.5. Message reception with MPI_ANY_SOURCE

MPI allows a special "wildcard" parameter to be specified as the source host in a receive. Since the receive may potentially match a message issued from *any* sender, the receiver must be prepared to receive a message from any host and thus must establish connections with all other processes.

This communication pattern exhibits a mismatch with the on-demand connection mechanism. The only solution is to issue connection requests to all other processes in the specified communicator upon encountering a receive from MPI_ANY_SOURCE. The receiver will then have an established connection with the process with which it will eventually communicate. In this solution, each process has the same probability to establish a connection and communicate with the receiver as long as it wants.

Note that there is no problem with fairness in this scenario since any nondeterminism is inherent in the application itself. If the receiver chooses to use MPI_ANY_SOURCE, and there happen to be multiple senders which could issue messages that might all match the receive, MPI offers no ordering guarantees, nor any concept of fairness in this case. As messages arrive, including data and connection requests, they will be processed in order and matched against the receive queue.

4. Implementation

We implemented the on-demand connection mechanism for MVICH on top of both GigaNet cLAN VIA and Berkeley VIA on Myrinet. MVICH is a freely available port of MPICH on several VIA implementations. All modification for incorporating the on-demand connection mechanism occurs in the ADI layer [11].

Unlike the original MVICH implementation with static connection management, there are no VI creation and VI connection setup in the MPI low-level initialization routine, MPID_Init(). Instead, a VI is created and a peer-to-peer connection request is issued during the processing of the first communication request. Before the connection for a VI is established, all send requests on that VI are stored in a FIFO queue.

MVICH adheres to the weak form of the MPI progress rule in that message progress is guaranteed only when user processes call the MPI library. This is achieved by running a common device check routine, MPID_DeviceCheck(), as part of most of the MPI library calls. Essentially, MPID_DeviceCheck() is the function in MVICH which handles all message progress. Modification is thus made to MPID_DeviceCheck() to maintain both connection progress and message progress.

Our implementation of the on-demand connection mechanism keeps the same communication semantics and progress guarantees as the original MVICH implementation using the static connection mechanism except for a small difference in the *non-local* semantics of the standard send mode. In the static mechanism, the standard mode send is

non-local and the successful completion of a send operation may depend on the occurrence of a matching receive, for example, when flow control credits are used up for short messages or a long message switches to a rendezvous protocol. In the on-demand connection mechanism, even if flow control credits are available for short messages, the completion of short messages still depends on whether the connection can be established. That is, the completion of preposted send requests for short messages at the source process depends on whether the receiver has planned to communicate with the sender. This is not a problem in any correct MPI program because the receiver can always find a chance to issue a connection request to the sender in any communication requests destined for the sender. Note that this also complies with the original interpretation of the MPI progress rule.

5. Performance results

To evaluate the performance of the on-demand connection mechanism as compared to the static connection mechanism, we conducted a series of measurements using a set of microbenchmarks and evaluated application performance with the NAS parallel benchmarks.

5.1. Experimental setup

Our experimental testbed is a cluster system consisting of 8 Dell Power Edge 6400 nodes connected by GigaNet cLAN and Myrinet. We use cLAN 1000 Host Adapters and cLAN5300 Cluster switches. LANai 7.0 Adapters are used for Myrinet. Each node has four 700MHz Pentium III Xeon processors, built around the ServerWorks ServerSet III HE chipset, which has a 64-bit 66 MHz PCI. Thus, there are actually 32 CPUs in total. These nodes are equipped with 1GB of SDRAM and 1MB L2-level cache. The linux kernel is 2.2.17.

5.2. Scalability

One of our main objectives is to increase scalability of MPI implementations on the Virtual Interface Architecture. As mentioned earlier, implementing the on-demand connection mechanism in MVICH enables the implementation of MPI over VIA to limit the use of resources to what applications absolutely require. Table 5.2 lists the average number of VI endpoints created in each process in the static mechanism and in the on-demand mechanism for tests in this paper (due to space limit, some of them are not presented and can be found in [21]). As shown in this table, the utilization of resources associated with VIs is very low with the static mechanism. With increasing the size of applications, it can be expected that the utilization becomes much lower. This is true for most parallel applications in which the number of communicating processes does not grow in proportion with the size of applications, instead it remains constant or grows as the surface-to-volume ratio of the computational grid. The on-demand mechanism eliminates all unused VI endpoints, connections, and their related resources in most cases, although potentially unused endpoints will be created if MPI_ANY_SOURCE is used by the application.

Table 2. Average number of VIs and resource utilization on each process

App	Size	Ave. # of VIs		Resource Utilization	
		static	on-demand	static	on-demand
Ring	16	15	2	0.13	1.0
	32	31	2	0.06	1.0
Barrier	16	15	4	0.27	1.0
	32	31	5	0.16	1.0
Bcast	16	15	4	0.27	1.0
	32	31	5	0.16	1.0
Allreduce	16	15	4	0.27	1.0
	32	31	5	0.16	1.0
Allgather	16	15	5	0.33	1.0
	32	31	6	0.19	1.0
Alltoall	16	15	15	1.0	1.0
	32	31	31	1.0	1.0
CG	16	15	4.75	0.32	1.0
	32	31	5.78	0.19	1.0
MG	16	15	15	1.0	1.0
	32	31	31	1.0	1.0
IS	16	15	15	1.0	1.0
	32	31	31	1.0	1.0
SP	16	15	8	0.53	1.0
	36	35	9.83	0.28	1.0
BT	16	15	8	0.53	1.0
	36	35	9.83	0.28	1.0

Furthermore, decreasing the number of VIs and connections in the communication system can relieve the issue of performance scalability in the underlying communication system, such as Berkeley VIA, which exhibits a performance penalty to maintain a large number of VIs and connections.

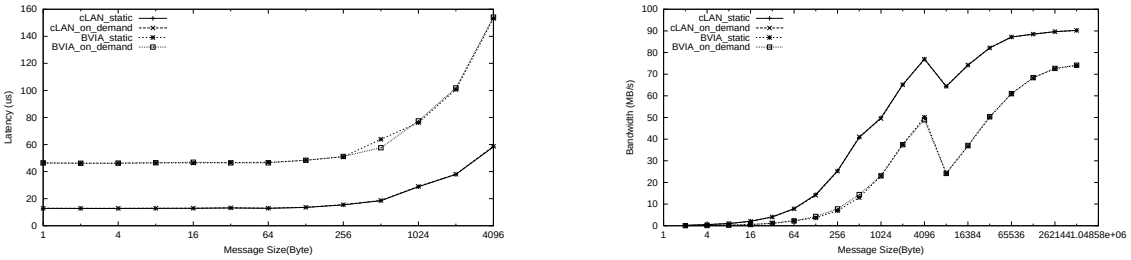
5.3. Latency and bandwidth

Figure 1(a) shows the latency results of MVICH on cLAN and Berkeley VIA for 1000 tests using each of the two mechanisms under study. They perform almost identically. Note that the on-demand mechanism suffers one obvious handicap: the cost of connection establishment is included in the timing loop of the benchmark because the connections are not pre-established as they are in the static connection mechanism, but this expense is amortized across the 1000 loops of the test.

Similar results are shown in Figure 1(b) for bandwidth. Since the default threshold of transition from the eager protocol to the rendezvous protocol is 5000 bytes, a jump can be seen in the bandwidth test.

5.4. Collective communications

MPI_Allreduce is one of the most frequently used operations [19] in large scientific applications. The test program



(a) Latency

(b) Bandwidth

Figure 1. Latency and Bandwidth of MVICH on cLAN VIA and Berkeley VIA.

we used is the *LLCbench* [16] benchmark suite. The operation in MPI_Allreduce is *MPI_SUM*. This program repeats these collective operations multiple times and then each process reports its own average latency. Only the latency on the master process (process 0 as default) is reported in Figure 2.

On cLAN VIA, the on-demand mechanism can achieve the same performance as the static mechanism with negligible degradation.

On Berkeley VIA, the reduction of VIs in the on-demand mechanism is beneficial. It can be seen that the on-demand mechanism achieves better performance than the static mechanism. For example, the latency of MPI_Allreduce with a message size of 1024 bytes on 8 nodes is 440 μs using the on-demand mechanism, while 515 μs using the static mechanism. The number of VIs is 3 in the former case, and 7 in the latter case.

Similar results are achieved for other collective operations and can be referred in [21]. Overall, the on-demand mechanism delivers the same performance as the static mechanism on cLAN VIA and better performance on Berkeley VIA.

5.5. NAS parallel benchmarks

The NPB suite consists of a set of 8 programs: EP, FT, MG, CG, IS, LU, SP and BT. In this paper, we show the results of MG, CG, IS, SP and BT only. For other three programs, there is little communication in EP and there is some compiling and/or running problems with LU and FT. Because we are interested in only the impact of the on-demand mechanism as compared to the existing static mechanism, we show normalized CPU times. Complete data can be found in [21].

Figure 3 shows the performance of 5 NPB programs with different program sizes and numbers of processes. Our testbed has 32 processors, the largest number of processes tested in CG, MG, IS is 32, while only 16 in SP and BT since they require a square number of processes. For MVICH over Berkeley VIA, we could not run more than one process on each node. Thus, the largest number of processes supported there is eight. The x-axis lists the tested combinations of the class and the number of processes for each program, and the y-axis represents the normalized CPU time. For example, in Figure 3(c), C.32 indicates class C data size on 32 processors.

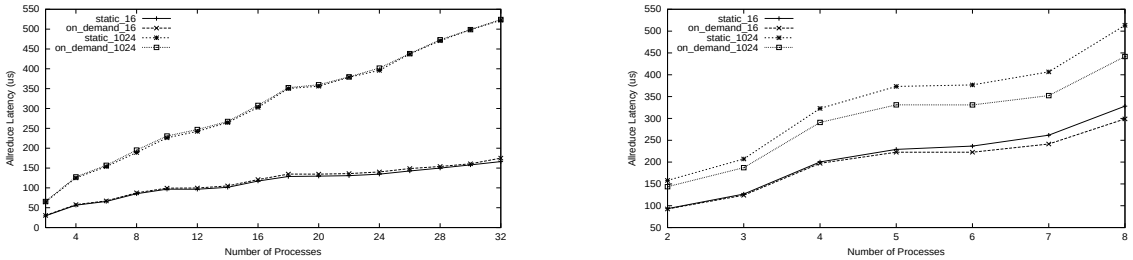
It should be noted that in these benchmarks, the time for establishing connections is not included when run using the static connection mechanism. In the on-demand mechanism, part of connections are established during MPI initialization and others are established during the timed execution of applications. That is, part of the connection time is included in the CPU time reported by the NAS benchmarks with the on-demand mechanism. This connection overhead is amortized by all communication operations on that connection. Thus, when there is much communication, the performance improvement by the reduction of VIs and connections and other related resources can still show an improvement over the static connection scheme.

In Figure 3, it can be observed that the on-demand mechanism delivers performance comparable to the static mechanism, performing better in several of them. The average improvement or degradation in performance is less than 2% in all cases.

Figure 4 shows the performance of IS, CG, SP and BT of MVICH over Berkeley VIA. The on-demand mechanism performs better than the static mechanism. This is attributed to the performance improvement with the reduction of VIs and connections. Note that even though there are same number of VIs with static and on-demand mechanisms in the IS benchmark, the on-demand mechanism still performs better. This is because the number of VIs gradually increases as needed with the on-demand mechanism.

5.6. Initialization time

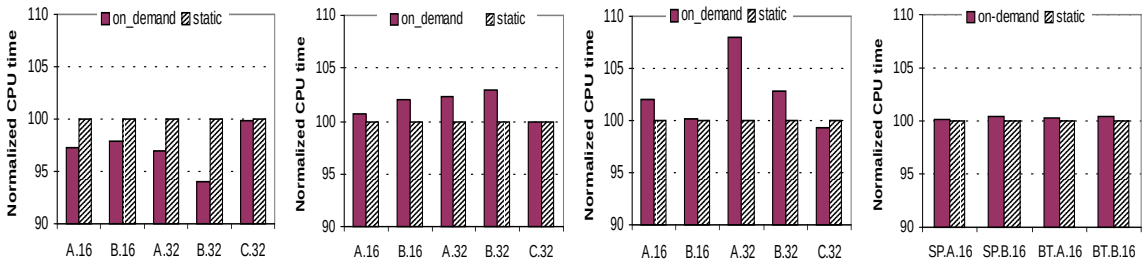
As mentioned earlier, the static mechanism can be implemented under either connection model: client-sever and peer-to-peer models. cLAN VIA supports both models, while Berkeley VIA only supports the peer-to-peer model. We measured the time for finishing the MPI_Init() function in MVICH over cLAN VIA and Berkeley VIA, respectively. The initialization time shown here is an average of the initialization time from all processes.



(a) MVICH on cLAN VIA

(b) MVICH on Berkeley VIA

Figure 2. Allreduce Latency in MVICH on cLAN VIA and Berkeley VIA.



(a) MG (b) IS (c) CG (d) SP and BT

Figure 3. Performance results of NPB programs in MVICH over cLAN.

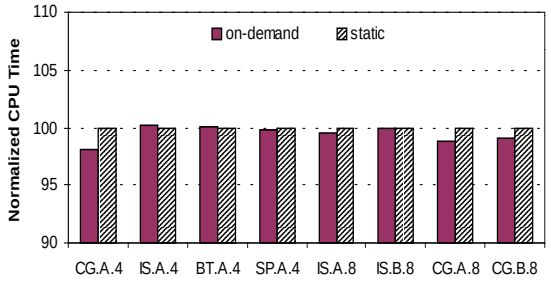


Figure 4. Performance results of NPB programs in MVICH over Berkeley VIA on Myrinet.

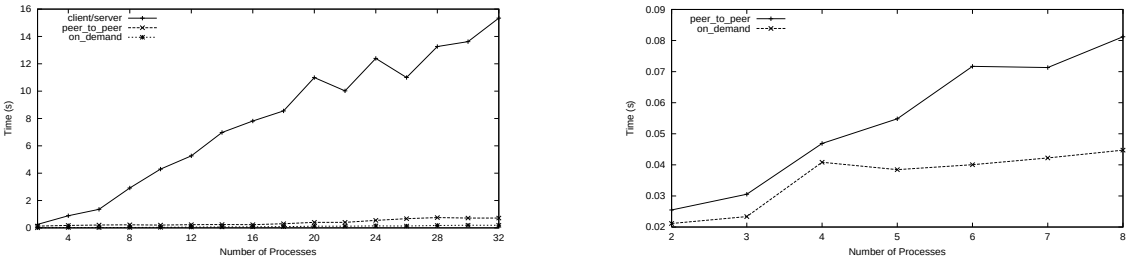
Figure 5(a) shows the initialization time on cLAN VIA. The current implementation of the client-server model in MVICH is serialized, in which connections are established strictly in order regardless of the actual arrival order of connection requests from peer processes, while the peer-to-peer scheme avoids this serialization by checking each potential connection request. Note that the peer-to-peer model is also used in our implementation of the on-demand connection mechanism. However, a fully-connected network is not created; thus, the initialization time is lower than in the peer-to-peer implementation of the static mechanism. In the on-demand mechanism, it also uses peer-to-peer model. However, it does not create a fully-connected network. Thus, the initialization time is lower than the peer-to-peer implementation of the static mechanism.

Figure 5(b) shows the initialization time in MVICH over Berkeley VIA. Similarly, the initialization time is lower than the peer-to-peer implementation of the static mechanism. This initialization time has an impact on the wall clock time for the execution of applications.

6. Conclusions and future work

The scalability of an implementation of MPI over VIA is one of the crucial issues in cluster systems connected by VIA-based networks. Since InfiniBand has many characteristics in common with VIA and with VIA-based I/O specifications such as Next Generation I/O (NGIO), this issue will continue to exist along with future InfiniBand hardware. In this paper, we addressed the design issues of incorporating the on-demand connection mechanism into an implementation of MPI over VIA. A complete implementation was done for MVICH over cLAN VIA and over Berkeley VIA on Myrinet. Performance evaluation on a set of microbenchmarks and NAS parallel benchmarks demonstrates that the on-demand mechanism can limit the use of resources to what applications absolutely require. Thus, the MPI implementation ensures that resource usage scales only as demanded by the application itself, not the underlying system.

Furthermore, performance evaluation also shows that the on-demand mechanism delivers comparable performance for a set of microbenchmarks and the NAS application



(a) MVICH on cLAN VIA

(b) MVICH on Berkeley VIA

Figure 5. Initialization time in MVICH on cLAN VIA and Berkeley VIA.

benchmarks to the static mechanism over cLAN VIA, and better performance compared to the static mechanism over Berkeley VIA on Myrinet.

We also addressed how to design and implement the on-demand mechanism to maintain MPI semantics, determinism, predictability, and fairness. We believe that the on-demand mechanism is a feasible solution to address one important current scalability limitation in the implementation of MPI on VIA-based networks.

Truly large-scale application performance evaluation is a natural extension of this work. Combination of on-demand connection establishment and dynamic flow-control on each VI connection is another planned work.

Acknowledgments

We are grateful to the anonymous reviewers for their helpful comments. Thanks to Darius Buntinas and fellow students in the NOWlab for the many profitable discussions they have undertaken with us.

References

[1] M-VIA: A High Performance Modular VIA for Linux. http://www.nersc.gov/research/FTG/via/.

[2] A. Begel, P. Buonadonna, D. Gay and D. Culler. An Analysis of VI Architecture Primitives in Support of Parallel and Distributed Communication. *to appear in Concurrency and Computation: Practice and Experience*, 2002.

[3] M. Banikazemi, J. Liu, S. Kutlug, A. Ramakrishna, P. Sadayappan, H. Sah, and D. K. Panda. VIBe: A Micro-benchmark Suite for Evaluating Virtual Interface Architecture (VIA) Implementations. In *IPDPS*, April 2001.

[4] M. Bertozzi, M. Panella, and M. Reggiani. Design of a VIA based communication protocol for LAM/MPI suite. In *9th Euromicro Workshop on Parallel and Distributed Processing*, Sept. 2001.

[5] P. Buonadonnaa, A. Geweke, and D. Culler. An Implementation and Analysis of the Virtual Interface Architecture. In *Proceedings of the Supercomputing (SC)* , pages 7–13, Nov. 1998.

[6] R. can de Geijn, D. Payne, L. Shuler, and J. Watts. A Streetguide to Collective Communication and its Application. http://www.cs.utexas.edu/users/rvdg/pubs/streetguide.ps, Jan 1996.

[7] Compaq, Intel, and Microsoft. VI Architecture Specification V1.0, December 1997.

[8] R. Dimitrov and A. Skjellum. An Efficient MPI Implementation for Virtual Interface (VI) Architecture-Enabled Cluster Computing. http://www.mpi-softtech.com/publications/default.asp, 1998.

[9] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. Merritt, E. Gronke, and C. Dodd. The Virtual Interface Architecture. *IEEE Micro*, pages 66–76, March/April 1998.

[10] Emulex Corp. cLAN: High Performance Host Bus Adapter, September 2000.

[11] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI, Message Passing Interface Standard. Technical report, Argonne National Laboratory and Mississippi State University.

[12] J. Hsieh, T. Leng, V. Mashayekhi, and R. Rooholamini. Architectural and performance evaluation of giganet and myrinet interconnects on clusters of small-scale SMP servers. In *Supercomputing*, 2000.

[13] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.0, October 24 2000.

[14] L. L. N. Laboratory. MVICH: MPI for Virtual Interface Architecture, August 2001.

[15] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.

[16] P. J. Mucci. LLCbench. http://icl.cs.utk.edu/projects/llcbench/, July 2000.

[17] R. Brightwell and A. Maccabe. Scalability limitations of via-based technologies in supporting mpi. In the Proceedings of the Fourth MPI Developer's and User's Conference, March 2000.

[18] ServerNet. http://www.servernet.com.

[19] J. S. Vetter and F. Mueller. Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures. In *IPDPS*, April 2002.

[20] F. C. Wong, R. P. Martin, R. H. Arpaci-Dusseau, and D. E. Culler. Architectural requirements and scalability of the nas parallel benchmarks. In *In the Proceedings of Supercomputing'99*, 1999.

[21] J. Wu, J. Liu, P. Wyckoff, and D. K. Panda. Impact of On-Demand Connection Management in MPI over VIA. Technical Report, OSU-CISRC-05/02-TR11, May 2002.