

Hot-Spot Avoidance With Multi-Pathing Over InfiniBand: An MPI Perspective *

A. Vishnu M. Koop A. Moody[†] A. R. Mamidala S. Narravula D. K. Panda

Computer Science and Engineering [†] Lawrence Livermore National Lab
The Ohio State University 7000 East Avenue
{vishnu, koop, mamidala, narravul, Livermore, CA 94550
panda}@cse.ohio-state.edu {moody20}@llnl.gov

Abstract

Large scale InfiniBand clusters are becoming increasingly popular, as reflected by the TOP 500 Supercomputer rankings. At the same time, fat tree has become a popular interconnection topology for these clusters, since it allows multiple paths to be available in between a pair of nodes. However, even with fat tree, hot-spots may occur in the network depending upon the route configuration between end nodes and communication pattern(s) in the application. To make matters worse, the deterministic routing nature of InfiniBand limits the application from effective use of multiple paths transparently and avoid the hot-spots in the network. Simulation based studies for switches and adapters to implement congestion control have been proposed in the literature. However, these studies have focussed on providing congestion control for the communication path, and not on utilizing multiple paths in the network for hot-spot avoidance. In this paper, we design a new MPI functionality, which provides hot-spot avoidance for different communications, without apriori knowledge of the pattern. We leverage LMC (LID Mask Count) mechanism of InfiniBand to create multiple paths in the network and present the design issues (scheduling policies, selecting number of paths, and scalability aspects) of our design. We implement our design and evaluate it with Pallas collective communication and MPI applications. On an InfiniBand cluster with 48 processes, MPI All-to-all Personalized shows an improvement of 27%. Our evaluation with NAS Parallel Benchmarks on 64 processes shows significant improvement in execution time with this functionality.

Keywords: MPI, Clusters, Hot-spot, Congestion Control, InfiniBand

1 Introduction

In the past decade, introduction of high speed interconnects like InfiniBand, Myrinet and Quadrics has escalated

the trends in *cluster computing*, with MPI being the *de-facto* programming model. InfiniBand in particular is being widely accepted as the next generation interconnect, due to its open standard and high performance. Large scale InfiniBand clusters are becoming increasingly popular, as reflected by the TOP 500 [1] Supercomputer rankings. At the same time, *fat tree* [4] has become a popular interconnection topology for these clusters, since it allows multiple paths to be available in between a pair of nodes. However, even with fat tree, hot-spots may occur in the network depending upon the route configuration between end nodes and communication patterns in the application. To make matters worse, the deterministic routing nature of InfiniBand limits the application from effective use of multiple paths transparently and avoid the hot-spots in the network. Simulation based studies for switches and adapters to implement congestion control have been proposed in the literature [14, 5, 9]. However, these studies have focussed on providing congestion control for the communication path, and not on utilizing multiple paths in the network for hot-spot avoidance. This leads to the following challenges:

1. What are the mechanisms available for utilizing multiple paths in InfiniBand?
2. What are the design issues at the MPI level in utilizing these mechanisms efficiently?
3. How much benefit can be achieved compared to the current state of the art MPI implementation?

In this paper, we address these challenges. We design an MPI functionality which provides hot-spot avoidance for different communication patterns, without apriori knowledge of the pattern. We leverage LMC (LID Mask Count) mechanism of InfiniBand to create multiple paths in the network, and study its efficiency in creation of contention free routes. We also present the design issues (scheduling policies, selecting number of paths, and scalability aspects) associated with our new MPI functionality.

We implement our design and evaluate it with micro-benchmarks, Pallas collective communication and with MPI applications. On an InfiniBand cluster with 64 processes, we observe an average improvement of 23% for displaced ring communication pattern amongst processes. For collective operations like MPI All-to-all Personalized and MPI

*This research is supported in part by DOE grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342 and #CNS-0509452; grants from Intel, Mellanox, Cisco systems, Linux Network and Sun Microsystems; and equipment donations from Intel, Mellanox, AMD, Apple, Appro, Dell, Microway, PathScale, IBM, SilverStorm and Sun Microsystems.

Reduce Scatter, we observe an improvement of 27% and 19% respectively. Our evaluation with NAS Parallel Benchmarks [2] shows an improvement of 6-9% in execution time for the FT Benchmark, with class B and class C size using 32-64 processes for evaluation. For other NAS Parallel Benchmarks, we do not see a degradation in performance compared to the original design.

The rest of the paper is organized as follows. In section 2, we present the background of our work. In section 3, we present the motivation of our work. In section 4, we present the design issues of our functionality at the MPI layer. In section 5, we present the performance evaluation on large scale InfiniBand clusters. We present the related work in section 6. We conclude and present our future directions in section 7.

2 Background

In this section, we provide background information for our work. We provide a brief introduction of InfiniBand followed by Message Passing Interface (MPI). Lastly, we provide a brief introduction to fat tree interconnection networks.

2.1 Overview of InfiniBand

The InfiniBand Architecture (IBA) [6] defines a switched network fabric for interconnecting processing nodes and I/O nodes. An InfiniBand network consists of switches, adapters (called Host Channel Adapters or HCAs) and links for communication. For communication, InfiniBand supports different classes of transport services (Reliable Connection, Unreliable Connection, Reliable Datagram and Unreliable Datagram). In this paper, we focus on the reliable connection model. In this model, each process-pair creates a unique entity for communication, called *queue pair*. Each queue pair consists of two queues; *send queue* and *receive queue*. The requests to send the data to the peer are placed on the send queue, by using a mechanism called *descriptor*. A descriptor describes the information necessary for a particular operation. For RDMA (Remote Direct Memory Access) operation, it specifies the local buffer, address of the peer buffer and access rights for manipulation of remote buffer. InfiniBand also provides a mechanism, where different queue pairs can share their receive queues, called *Shared Receive Queue* mechanism. The completions of descriptors are posted on a queue called *completion queue*. This mechanism allows a sender to know the status of the data transfer operation. Different mechanisms for notification are also supported (polling and asynchronous).

From the network management perspective, InfiniBand defines an entity called *subnet manager*, which is responsible for discovery, configuration and maintenance of a network. Each InfiniBand port in a network is identified by one or more local identifiers (LIDs), which are assigned by the subnet manager. Since InfiniBand uses destination based routing, each switch in the network has a routing table corresponding to the LIDs of the destination. Thus, decisions to route messages adaptively cannot be taken by the intermediate switches. Instead, InfiniBand provides a mechanism, in which each port can be assigned multiple LIDs,

to exploit multiple paths in the network. Leveraging this mechanism for avoiding hot-spots is the focus of this paper.

2.2 Overview of MPI Protocols

MPI defines two types of communication protocols; *eager* and *rendezvous*. These protocols are handled by a component in the MPI implementation called *progress engine*. In the eager protocol, the message is pushed to the receiver side regardless of its state. In the rendezvous protocol, a handshake takes place between the sender and the receiver via control messages before the data is sent to the receiver side. Usually, Eager protocol is used for small messages and Rendezvous protocol is used for large messages.

For small messages, a copy based approach benefits over the cost of the handshake. For the large messages, it is beneficial to perform exchange of buffer addresses. This is a requirement for RDMA (Remote Direct Memory Access) mechanism, which allows remote data to be read/written with kernel bypass. The application buffer(s) need to be pinned so that the operating system does not swap them during communication. In this paper, we use this mechanism for large messages. Using multiple paths, we divide the application buffer into *stripes* for efficient use of multiple paths.

2.3 Fat Tree Topology

Fat Tree is a general purpose interconnection topology, which is used for effective utilization of hardware resource devoted to communication. In a fat tree based interconnection network, leaf nodes represent processors, internal nodes represent switches, and edges correspond to bidirectional links between parents and children. In a traditional

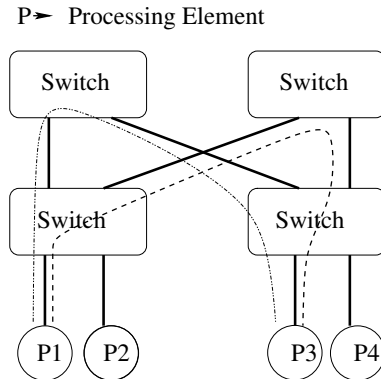


Figure 1. A Fat Tree with Four Switches

binary tree, the bandwidth at different levels of the network is not constant. Due to this configuration, congestion may occur near the root of the tree. Figure 1 shows an example of fat tree with four processing elements connected with four switches. The physical links are represented by the solid lines. Some of the possible paths between processing elements P1 and P3 are shown by dotted lines of different dot formats. Please note that the other possible paths (which are not min-hop) are not shown for clarity.

3 Motivation

In this section, we present the motivation of our work. We take a cluster with a fat-tree switch and execute an MPI program using this switch to understand the contention and occurrence of hot-spots in the network. Figure 2 represents the switch topology used for our evaluation. Each switch block consists of 24 ports. The leaf switches (referred to as leaf blocks from here onwards) have 12 ports available to be used by the end nodes, the other 12 ports are connected to the spine switches (referred to as spine blocks from here onwards). In the figure, blocks 1 - 12 are leaf blocks; blocks 13 - 24 are spine blocks. The complete switch has 144 ports available for end nodes. Each block is a crossbar in itself. Please note that in the figure, we have not shown all blocks and their internal connections for clarity.

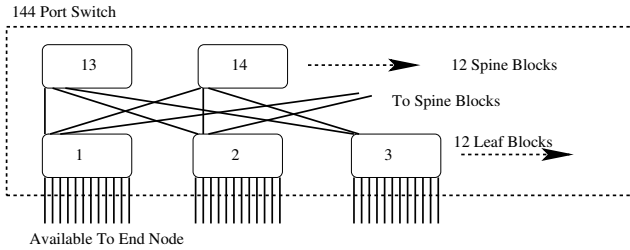


Figure 2. 144-port InfiniBand Switch Block Diagram

To demonstrate the contention, we take a simple MPI program, which performs ring communication with neighbor rank increasing at every step. The communication pattern is further illustrated in the Figure 3 (only step1 and step2 are shown for clarity). Executing the program with n processes takes $n-1$ steps. Let $rank_i$ denote the rank of the i th process in the program. and $step_j$ denote the j th step during execution. At $step_j$, an MPI process with $rank_i$ communicates with MPI process $rank_{i+j}$. This communication pattern is referred to as *DRC (Displaced Ring Communication)* for the rest of the paper.

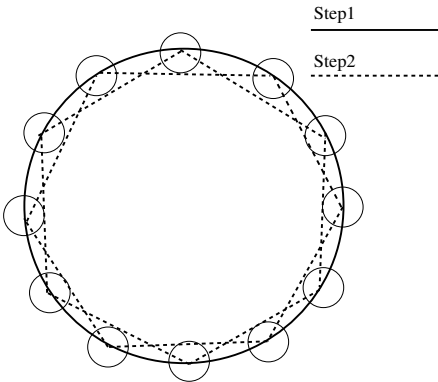


Figure 3. Communication Steps in Displaced Ring Communication

We take an instance of this program with 24 processes and schedule MPI processes with $rank_0 - rank_{11}$ on nodes connected to block 1 and $rank_{12} - rank_{23}$ on block 2. We use MVAPICH¹, a popular MPI over InfiniBand as our MPI implementation for the evaluation of DRC. Since each block is a crossbar in itself, no contention is observed for intra-block communication. However, as the step iteration increases, the inter-block communication increases and a significant link contention is observed. The link contention observed during the step 12 (each process doing inter-block communication) is shown in Figure 4, with thicker solid lines representing more contentions. The quantitative evaluation is shown in Figure 5. From Figure 4, we can see that some links are over-used to a degree from four to zero. As the degree of link usage increases, the bandwidth is split amongst the communication instances using the link(s), making them *hot spots*. In our example, paths using block 13 split bandwidth for four different communication instances making the set of links using this block hot-spots. In Figure 5, we show the results of our evaluation. On the x-axis and y-axis, we show the process ranks. The bandwidth achieved during communication of $rank_x$ and $rank_y$ is shown with a square generated by drawing lines for the process ranks (shown as an example with the intersection of dotted lines in Figure 5). The darker the squares, the worse is the bandwidth achieved in comparison to the best bandwidth (The best bandwidth is achieved by the processes doing intra-block communication). We notice that as inter-block communication increases, the corresponding squares become darker.

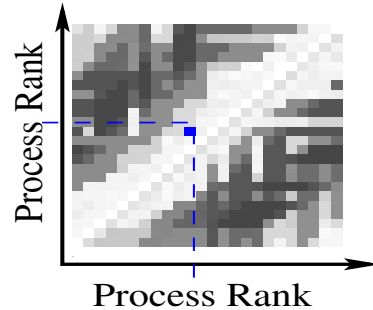


Figure 5. Displaced Ring Communication, 24 Processes

As indicated in Figure 5, even though, there are sufficient links for an independent path of communication between blocks 1 and 2 (using spine blocks), DRC is not able to utilize them in a contention free manner. The usage of these links is highly dependent upon the path configuration done by the subnet manager. This route configuration done by the subnet manager may benefit one communication pattern and show contention for other patterns, leaving un-utilized

¹MVAPICH/MVAPICH2 [8] are currently used by more than 450 organizations worldwide. It has enabled several large InfiniBand clusters to obtain top 500 ranking. A version is also available in an integrated manner with OpenFabrics stack

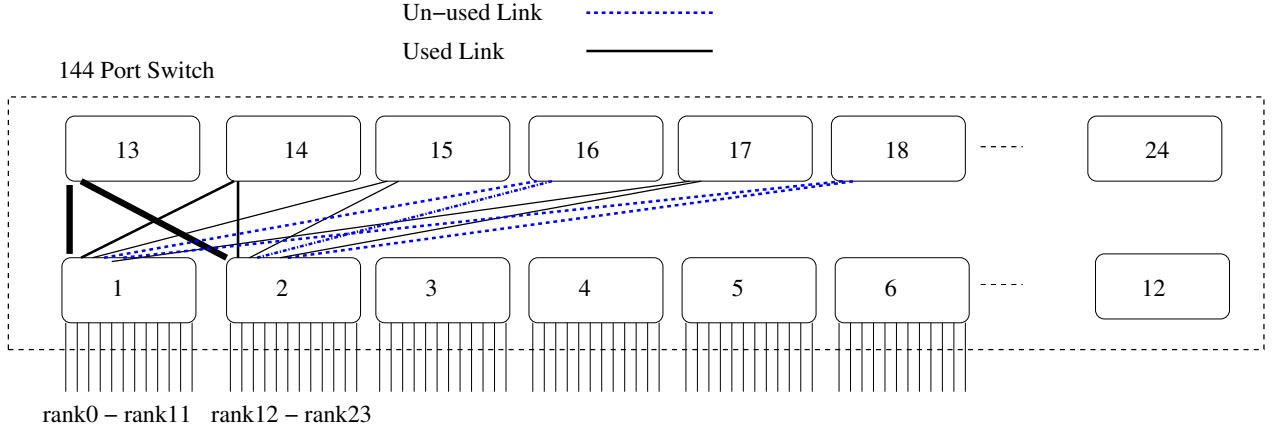


Figure 4. Link Usage with Displaced Ring Communication

links in the network. At the same time, the deterministic routing nature of InfiniBand does not allow us to use these links efficiently. In this Figure, the unused paths are shown with a dotted line. We notice that paths using switch blocks 16 and 18 are left un-utilized. (The other un-utilized links are not shown in the figure).

Under such a scenario, the utilization of the links is the responsibility of the MPI implementation. Hence, designing an efficient MPI library, with effective use of multiple paths is critical to hot-spot avoidance. In the next section, we explore the LMC mechanism provided by InfiniBand for creation of multiple paths between a pair of nodes. We leverage this mechanism to design an MPI implementation by taking advantage of these paths in an efficient manner and provide hot-spot avoidance for different communication patterns, without a-priori knowledge of the pattern.

4 Hot-Spot Avoidance MPI over InfiniBand

In [7], we presented an initial framework using multi-rail networks and presented different scheduling policies for their efficient utilization. We leverage this framework for designing hot-spot avoidance functionality. However, the existing framework suffers from following limitations:

- A key functionality missing in the existing framework is to leverage the LMC mechanism for hot-spot avoidance. Using the subnet manager to configure disjoint paths and their efficient usage is a major functionality, which is added to the existing framework.
- The existing framework assumes the presence of one path to be utilized per end port. In the motivation section, we observed the presence of multiple un-utilized paths per end port. Utilizing very few of these paths may not significantly help to avoid hot-spots. However, utilization of all the existing paths has practical implications due to startup costs, and accuracy in the estimation of path bandwidth. We study this issue in detail in our design and evaluation.
- Increasing the number of paths leads to increased memory utilization. In the existing framework, we did not address the memory scalability issue, due to the

presence of only one path/end port. We address this problem using InfiniBand's shared receive queue mechanism.

Our existing frame-work is based on MVAPICH, introduced in the motivation section. We call our enhanced design, HSAM (Hot-Spot Avoidance MVAPICH). Next, we present the overall design of HSAM.

4.1 Overall Design

Figure 6 represents the overall design of HSAM. In the figure, we can see that besides MPI Protocol Layer and InfiniBand Layer, our design consists of three major components: *Communication Scheduler*, *Scheduling Policies*, and *Completion Filter*. The Communication Scheduler is the central part of our design. It accepts protocol messages from the MPI Protocol Layer, and stripes them across multiple paths. In order to decide how to stripe, the Communication Scheduler uses information provided by the Scheduling Policies component. Scheduling Policies can be static that are determined at initialization time. They can also be dynamic and adjust themselves based on input from other components of the system. Since a single message may be striped and sent as multiple messages through the InfiniBand Layer, we use the Completion Filter to process completion notifications and to inform the MPI Protocol Layer upon completions of all stripes.

4.2 Leveraging Multiple Paths Using LMC

In the motivation section, we have noted that some of the paths became hot-spots, while other paths are left un-utilized in the network. One important mechanism to efficiently use the available paths is by changing the routing table of each switch block. Using this mechanism, contention free paths can be created for a particular communication pattern. The subnet manager allows a user to input its own routing table for different switches, which can be used for communication. However, this mechanism suffers from the fact that the optimization can be done only for a single communication pattern. At the same time, in the presence of other jobs in the network, exact scheduling of each MPI

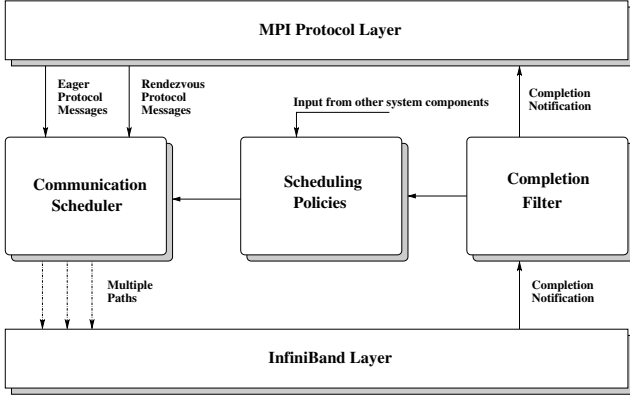


Figure 6. Overall Architecture [7]

task can complicate the generation of user-assisted routing tables.

To overcome the above limitations, we leverage the LID (Local Identifier) Mask Count (LMC) mechanism of InfiniBand, which allows multiple paths to be created between a pair of nodes. Using an LMC value of x , we can create 2^x paths, with 7 (128 paths) being the maximum value allowed for LMC. We use OpenSM, a popular subnet manager for InfiniBand to configure these paths. Using the *trace-route* mechanism of InfiniBand, we calculate the exact path (set of switch blocks and ports) taken by each pair of source and destination LID for different values of LMC. We notice that the subnet manager is able to configure paths utilizing different spine blocks in the switch. Hence, the LMC mechanism provides us as many contention free paths as possible. However, efficient utilization of these paths is dependent upon the scheduling policy for data transfer. In the next section, we discuss the scheduling policies used for evaluation.

4.3 Scheduling Policies

Different scheduling policies can be used by the Communication Scheduler to decide the paths to use for message transmission. We categorize different policies into two classes: *static schemes* and *dynamic schemes*.

- In static schemes, the policy and its parameters are determined at initialization time and do not change during the execution of MPI applications.
- In dynamic schemes, we can switch between different policies or change parameters during the program execution.

Even Striping: For static schemes, the weight distribution of each path is fixed and does not change depending upon the feedback of different components in the MPI Layer. In our design, we use this mechanism to design a striping policy for large messages, which stripes messages evenly across all paths used for communication. We refer to this policy as *even striping*.

Adaptive Striping: In this policy, we leverage the completion notification mechanism of InfiniBand to calculate path bandwidth. The completion notification is generated

upon the data delivery to the remote destination’s adapter. This provides a relatively accurate estimation of the time spent in the network. To begin with, we stripe the messages evenly on all paths. The weights of each paths is adjusted in accordance with the completion times of the stripes. Updated paths can then be used for followup iterations. A detailed discussion of weight adjustment and efficiency of this policy is discussed in our previous work [7]. We call this policy as *adaptive striping* policy.

This policy helps us in avoiding hot-spots as much as possible. To begin with, the adaptive striping policy behaves like even striping. However, on using a path with hot-spot, the stripe delivery time increases considerably. Since the paths are adjusted accordingly, this policy is able to avoid the hot-spots. At the same time, if the hot-spots disappear, this policy adjusts the weights accordingly. Hence, this policy benefits in the presence of hot-spots, however does not lead to performance degradation in their absence.

4.4 Selecting Number of Paths

In our previous work with multi-rail InfiniBand networks, we used one path of communication/end port. However, in the motivation section, we observed that there are multiple paths available for communication between every pair of processes, even though there is only one physical port available at the end node. Using the maximum value of LMC allowed by InfiniBand specification, we can create 128 virtual paths. However, some of the paths may physically overlap with each other. For a two-stage Fat Tree, as shown in the motivation section, we observe that there are only twelve disjoint physical paths. Hence, using a maximum of twelve paths would suffice our need for hot-spot avoidance. As per the InfiniBand specification connection model, a queue pair/path is needed to use them simultaneously. Once a path is specified for a queue pair, it cannot be changed during the communication (An exception is Automatic Path Migration for InfiniBand, which is beyond the scope of this paper). However, there are some practical considerations in leveraging all the paths simultaneously:

- Sending a message stripe through each path requires posting a corresponding descriptor. Hence, this may lead to significant startup overhead with increasing number of paths.
- For each message stripe, a completion is generated on the sender side. With increasing number of paths, more completions need to be handled, which can potentially delay the progress of the application.
- The accuracy of path bandwidth is significantly dependent upon the discovery of the completions, as mentioned in the scheduling policies sections. With increasing number of paths, the accuracy may vary significantly.
- The memory usage increases with increasing number of paths. We handle this issue in the scalability section later.

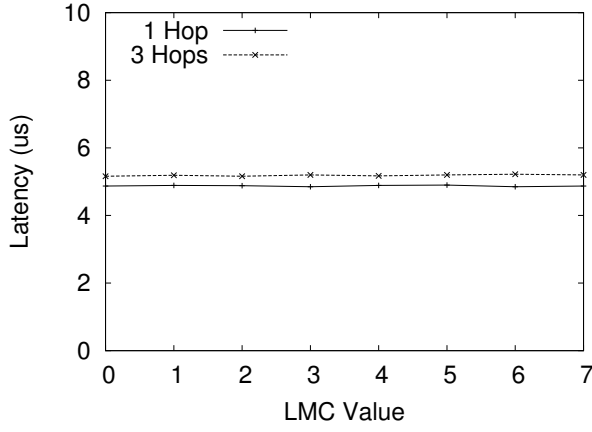


Figure 7. MPI Latency (UP mode)

Hence, a judicious selection of number of paths is imperative to performance and memory utilization of the MPI library. In the next section, we discuss the memory utilization aspect of our design.

4.5 Scalability Aspects of HSAM

In the previous section, we discussed that increasing number of paths leads to more memory utilization. In essence, the memory utilization per path can be represented as follows:

$$mem_{qp} = mem_{qp-context} + ne_s * mem_{sqe} + ne_r * mem_{rqe} \quad (1)$$

where mem_{qp} is the connection memory usage per path, ne_s and ne_r are number of send and receive work queue elements, mem_{sqe} and mem_{rqe} are the sizes of each send queue and receive queue elements, respectively. $mem_{qp-context}$ is the size of each QP context, corresponding to each path in our design.

To make our design more scalable, we use the shared receive queue mechanism of InfiniBand [10] to handle the scalability aspect for receive queue. In this paper, receive queues corresponding to different processes were shared. We allow different paths for the same set of processes to be attached to the shared receive queue. As a result the memory usage in our design can be represented as:

$$mem_{qp} = mem_{qp-context} + ne_s * mem_{sqe} \quad (2)$$

Although our current design focusses only on reducing the memory usage for receive queue, additional methods such as setting up connections only as needed (on-demand connection management) have also been shown to significantly reduce memory usage and can be used in conjunction with our design. In future, we plan to address these issues.

5 Performance Evaluation

In this section, we evaluate the performance of HSAM (Hot-spot avoidance MVAPICH) and compare its performance with the current version of MVAPICH (referred to

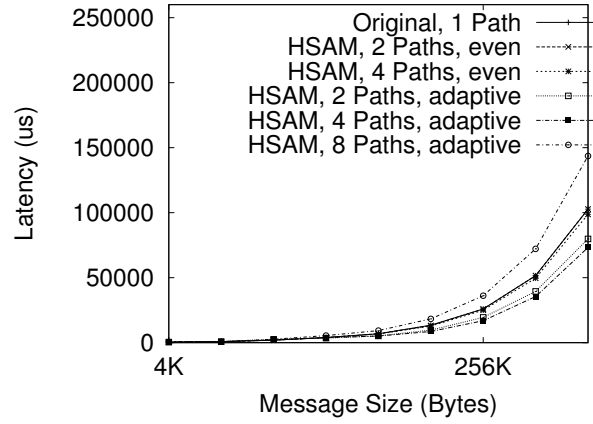


Figure 8. MPI Alltoall Personalized (48x1)

as Original for the rest of the section). We use one process per node (hence 48 process run is referred to as 48x1). Our evaluation consists of two parts. In the first part, we show the performance benefit we can achieve compared to the original MPI implementation using collective communication. In the second part, we provide an evaluation of our design with MPI applications. We use NAS Parallel Benchmarks [2] and for our evaluation.

5.1 Experimental Testbed

Our testbed cluster consists of 64 nodes; 32 nodes with Intel EM64T architecture and 32 nodes with AMD Opteron architecture. Each node with Intel EM64T architecture is a dual socket, single core with 3.6 GHz, 2 MB L2 cache and 2 GB DDR2 533 MHz main memory. Each node with AMD Opteron architecture is a dual-socket, single core with 2.8 GHz, 1 MB L2 cache and 4 GB DDR2 533 MHz main memory. On each of these systems, the I/O bus is x8 PCI-Express with Mellanox MT25208 dual-port DDR Mellanox HCAs attached to 144-port DDR Flextronics switch. The firmware version is 5.1.400. We have used Open Fabrics Enterprise distribution (OFED) version 1.1 for evaluation on each of the nodes and OpenSM as the subnet manager, distributed with this version.

5.2 Performance Benefits of HSAM with Collective Communication

Figure 7 shows the ping-pong latency achieved using two processes by a 4-byte message with increasing value of LMC. The motivation is to understand the impact of increased routing table size (present on each switch block), with increasing value of LMC, since the number of entries in the table grow exponentially. We notice that increasing LMC does not impact the latency. The figure also represents the performance, when both processes are located on the same block (1-hop) and different blocks (3-hops). We notice that 3-hops increases the latency by 0.25us, an increase of around 0.12us for every switch block.

In Figure 8, we show the performance of MPI Alltoall

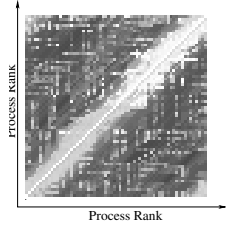


Figure 9. Displaced Ring Communication, 64x1, HSAM, 4 Paths, adaptive

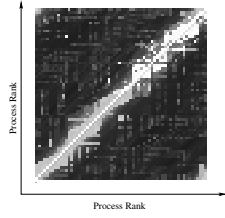


Figure 10. Displaced Ring Communication, 64x1, Original, 1 Path

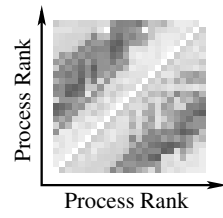


Figure 11. Displaced Ring Communication, 24x1, HSAM, 4 Paths, adaptive

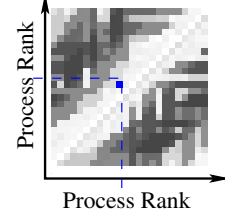


Figure 12. Displaced Ring Communication, 24x1, Original, 1 Path

Personalized communication for 48 processes. We compare a combination of HSAM parameters; number of paths, striping policy, LMC use with original implementation. In our design, the completion filter waits for the completion of all stripes, before notifying the application with the completion. The time is dominated with the slowest stripe, even though other stripes may have finished earlier, and as a result the benefit from using hot-spot free path is nullified. Hence, using even striping does not improve the performance compared to the original implementation. For rest of the evaluation, we only focus on adaptive striping policy with HSAM. Using adaptive striping with HSAM improves the performance significantly (both 2 paths and 4 paths).

Figures 9 and 10 show the results for displaced ring communication, explained in the section 3. We compare HSAM’s adaptive policy with the original implementation. We notice that using HSAM and the adaptive policy, the spots compared to the original policy achieve much better bandwidth. The average bandwidth improves by 23% compared to the original implementation. The points near the diagonal are also much wider implying the benefits of HSAM and adaptive striping. In section 3, we presented the results for 24 processes case. Figure 11 shows the results for 24 processes with HSAM. We can clearly notice that the corresponding dark spots in the original case are much lighter with HSAM. We are able to improve the average bandwidth by 21%.

5.3 Performance Benefits at Application Level

In this section, we present the results for MPI applications with HSAM. We use NAS Parallel Benchmarks [2]. For NAS Parallel Benchmarks, we focus on the FT benchmark, which uses MPI All-to-all personalized communication. We use class B and class C problem size for evaluation. Although not included in the paper, we have not seen performance degradation for rest of the NAS Parallel Benchmarks using HSAM.

Figure 13 shows the results for FT benchmark, Class B problem size, for 16, 32 and 64 processes. We compare the performance of HSAM’s adaptive striping with the original policy. Using 16 processes, we do not see any im-

provement, since the contention in the network is negligible. However, with 32 processes, we see an improvement of 8% with HSAM and 6% with 64 processes, compared to the original design. Figure 14 shows the results for Class C problem size with FT benchmark. With 32 processes, we see an improvement of 9%. The increased improvement is attributed to the increased size of data transfer during MPI All-to-all phase. With 64 processes, an improvement of 8% is observed, compared to the original design.

6 Related Work

Many researchers have focused on providing MPI support for multi-rail networks [7, 3, 11]. In our previous work, we have designed MPI-2 one sided communication using multi-rail InfiniBand networks [13]. Handling network heterogeneity and network faults with asynchronous recovery of previously failed paths has also been presented [12]. However, the above works have focused on design and evaluation with multi-rail networks on the end nodes (multiple ports, multiple adapters), rather than the network.

Congestion control has been studied with simulations by multiple researchers [14, 5, 9]. It has also been proposed as a part of the IBA specification [6]. In [9], the researchers proposed a notification mechanism to the subnet manager for reducing the static rate of the affected connection to perform congestion control. The above works have proposed explicit congestion notification mechanisms (forward congestion notification and backward congestion notification) to allow the switches and channel adapters to become aware of congestion in the network. However, these works are simulation based and have focussed on congestion control for the existing path of communication, rather than utilizing the presence of multiple paths in the network. Hence, designing software based solutions to perform hot-spot avoidance is imperative for upcoming large scale InfiniBand networks. In this paper, we have presented a new MPI functionality for hot-spot avoidance and shown benefits for different communication patterns for MPI applications.

7 Conclusions and Future Work

In this paper, we have designed an MPI functionality which provides hot-spot avoidance for different communi-

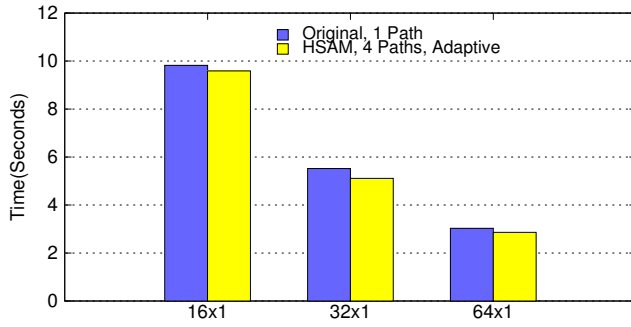


Figure 13. NAS Parallel Benchmarks, FT, Class B

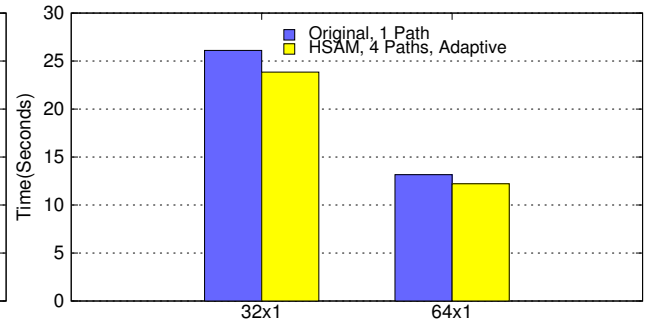


Figure 14. NAS Parallel Benchmarks, FT, Class C

cation patterns, without apriori knowledge of the pattern. We have leveraged LMC (LID Mask Count) mechanism of InfiniBand to create multiple paths in the network, and studied its efficiency in creation of contention free routes. We have also presented the design issues (scheduling policies, selecting number of paths, and scalability aspects) associated with our MPI functionality. We have implemented our design and evaluated it with collective communication and MPI applications. On an InfiniBand cluster with 64 processes, we have observed an average improvement of 23% for displaced ring communication pattern. For collective communication like MPI All-to-all Personalized, we have observed an improvement of 27%. Our evaluation with NAS Parallel Benchmarks has shown an improvement of 6-9% in execution time for FT Benchmark, with class B and class C problem size.

As a part of future work, we plan to evaluate our design with very large scale InfiniBand clusters. We plan to study the interaction of different jobs for communication and partitioning them logically for minimal interaction of the paths in the network. As the InfiniBand products become mature, we plan to study the proposed explicit congestion notification mechanisms for congestion control on large scale InfiniBand clusters.

References

- [1] TOP 500 Supercomputer Sites. <http://www.top500.org>.
- [2] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. Number 3, pages 63–73, Fall 1991.
- [3] S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie, and L. Gurvits. Using Multirail Networks in High-Performance Clusters. In *IEEE Cluster 2001*, Newport Beach, CA, October 2001.
- [4] R. I. Greenberg and C. E. Leiserson. Randomized Routing on Fat-Trees. In *Proceedings of the 26th Annual Symposium on the Foundations of Computer Science*, pages 241–249, 1985.
- [5] M. Gusat, D. Craddock, W. Denzel, T. Engbersen, N. Ni, G. Pfister, W. Rooney, and J. Duato. Congestion Control in InfiniBand Networks. In *Hot Interconnects*, pages 158–159, 2005.
- [6] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.2, October 2004.
- [7] J. Liu, A. Vishnu, and D. K. Panda. Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation. In *SuperComputing Conference*, 2004.
- [8] Network-Based Computing Laboratory. MVA-PICH/MVAPICH2: MPI-1/MPI-2 for InfiniBand on VAPI/Gen2 Layer. <http://nowlab.cse.ohio-state.edu/projects/mpi-iba/index.html>.
- [9] J. R. Santos, Y. Turner, and G. J. Janakiraman. End-to-End Congestion Control for InfiniBand. In *INFOCOM*, 2003.
- [10] S. Sur, L. Chai, H.-W. Jin, and D. K. Panda. Shared Receive Queue based Scalable MPI Design for InfiniBand Clusters. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.
- [11] A. Vishnu, B. Benton, and D. K. Panda. High Performance MPI on IBM 12x InfiniBand Architecture. In *Proceedings of International Workshop on High-Level Parallel Programming Models and Supportive Environments, held in conjunction with IPDPS '07*, March 2007, March 2007.
- [12] A. Vishnu, P. Gupta, A. Mamidala, and D. K. Panda. A Software Based Approach for Providing Network Fault Tolerance in Clusters Using the uDAPL Interface: MPI Level Design and Performance Evaluation. In *Proceedings of SuperComputing*, November 2006.
- [13] A. Vishnu, G. Santhanaraman, W. Huang, H.-W. Jin, and D. K. Panda. Supporting MPI-2 One Sided Communication on Multi-Rail InfiniBand Clusters: Design Challenges and Performance Benefits. In *International Conference on High Performance Computing, HiPC*, 2005.
- [14] S. Yan, G. Min, and I. Awan. An Enhanced Congestion Control Mechanism in InfiniBand Networks for High Performance Computing Systems. In *AINA '06: Proceedings of the 20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*, pages 845–850, Washington, DC, USA, 2006. IEEE Computer Society.