Improving Application Performance and Predictability using Multiple Virtual Lanes in Modern Multi-Core InfiniBand Clusters*

Hari Subramoni, Ping Lai, Sayantan Sur, and Dhabaleswar K. (DK) Panda

Department of Computer Science and Engineering, The Ohio State University
{subramon, laipi, surs, panda}@cse.ohio-state.edu

***Abstract*—Network congestion is an important factor affecting the performance of large scale jobs in supercomputing clusters, especially with the wide deployment of multi-core processors. The blocking nature of current day collectives makes such congestion a critical factor in their performance. On the other hand, modern interconnects like InfiniBand provide us with many novel features such as Virtual Lanes aimed at delivering better performance to end applications. Theoretical research in the field of network congestion indicate Head of Line (HoL) blocking as a common causes for congestion and the use of multiple virtual lanes as one of the ways to alleviate it. In this context, we make use of the multiple virtual lanes provided by the InfiniBand standard as a means to alleviate network congestion and thereby improve the performance of various high performance computing applications on modern multi-core clusters. We integrate our scheme into the MVAPICH2 MPI library. To the best of our knowledge, this is the first such implementation that takes advantage of the use of multiple virtual lanes at the MPI level. We perform various experiments at native InfiniBand, microbenchmark as well as at the application levels. The results of our experimental evaluation show that the use of multiple virtual lanes can improve the predictability of message arrival by up to 10 times in the presence of network congestion. Our microbenchmark level evaluation with multiple communication streams show that the use of multiple virtual lanes can improve the bandwidth / latency / message rate of medium sized messages by up to 13%. Through the use of multiple virtual lanes, we are also able to improve the performance of the Alltoall collective operation for medium message sizes by up to 20%. Performance improvement of up to 12% is also observed for Alltoall collective operation through segregation of traffic into multiple virtual lanes when multiple jobs compete for the same network resource. We also see that our scheme can improve the performance of collective operations used inside the CPMD application by 11% and the overall performance of the CPMD application itself by up to 6%.**

***Index Terms*—Virtual Lanes, InfiniBand, MPI, High Performance Computing, QoS in InfiniBand**

I. Introduction

The rapid increase in the size and scale of modern clusters has resulted in a corresponding increase in the amount of traffic generated on the interconnection network. Although the network devices are being scaled up to meet the challenges of the clusters of tomorrow, they lag behind the rate of development of the various multi-core processors. This can lead to congestion at the Host Channel Adapter (HCA) as well as in the network fabric. With current rate of growth of multi- and many-core processors, this congestion is expected to worsen in the future. The predictability of inter packet arrival time is inversely proportional to the amount of congestion in the network. Unpredictable or varying inter packet arrival time at different destinations can have adverse impact on the performance of supercomputing applications.

On the other hand, modern interconnects like InfiniBand provide us with many novel features such as Virtual Lanes (VL) and Service Levels (SL) aimed at delivering better performance to the end applications. Theoretical research in the field of network congestion indicate Head of Line (HoL) blocking [1] as one of the common causes for congestion and the use of multiple virtual lanes as one of the ways to reduce this [2]. Unfortunately end users are not able to take advantage of these features due to the lack of support from the various implementations of popular programming models, such as MPI, that are in use today.

MPI based applications such as CPMD [3], which rely heavily on collective communication, will be badly affected by un-predictable arrival of packets due to the large number of processes taking part in the collective communication and synchronization [4]. With such operations, a delay in the reception of just one packet by any process can hold up the progress of all other processes. This is especially true for any tree-based collective algorithm where the amount of skew increases as packets propagate down the tree during execution.

In this paper we propose a strategy to reduce the amount of contention in the network and improve the performance of end applications through the use of multiple virtual lanes. We aim to improve the predictability of the arrival pattern of packets, thereby reducing the amount of delay and achieving better performance.

We implement our schemes in the MVAPICH2 MPI library [5] and perform various tests at the native InfiniBand, microbenchmark as well as application levels. To the best of our knowledge, this is the first such implementation that takes advantage of the use of multiple virtual lanes at the MPI level. The results of our experimental evaluation show that the use of multiple virtual lanes can improve the predictability of message arrival by up to 10 times in presence of network congestion. Our microbenchmark level evaluation with multiple communication streams show that the use of multiple virtual lanes can improve the bandwidth / latency / message rate of medium sized messages by up to 13%. Through the use of multiple virtual lanes, we are also able to improve the performance of the Alltoall collective operation for medium message sizes by up to 20%. Performance improvement of

*This research is supported in part by U.S. Department of Energy grants #DE-FC02-06ER25749 and , #DE-FC02-06ER25755; National Science Foundation grants #CNS-0403342, #CCF-0702675, #CCF-0833169, #CCF-0916302, #CCF-0621484 and #OCI-0926691; grant from Wright Center for Innovation #WCI04-010-OSU-0; grants from Intel, Mellanox, Cisco, QLogic, and Sun Microsystems; Equipment donations from Intel, Mellanox, AMD, Obsidian, Advanced Clustering, Appro, QLogic, and Sun Microsystems.

up to 12% is also observed through segregation of traffic into multiple virtual lanes when multiple jobs compete for the same network resource. We also see that our scheme can improve the performance of collective operations used inside the CPMD application by 11% and the overall performance of the CPMD application itself by up to 6%.

The rest of the paper is organized as follows. Section II presents the motivation for our work. In Section III we give a brief overview of InfiniBand, MPI, and other technologies used in this paper. Section IV explains the design methodology we followed. Our experimental results and analysis are described in Section V. Section VI gives an overview of the related work. Finally we summarize our conclusions and possible future work in Section VII.

II. Motivation

Modern HPC systems like TACC Ranger [6] has many thousands of jobs executing on it at any given point in time. A large number of these are multi-node jobs, requiring the use of the networking infrastructure. In such a scenario, it is common for multiple processes to share the same physical link for communication. Figure 1 gives a graphical representation of the routes taken by a subset of jobs on TACC Ranger on a given day. The lines represent the SDR (8 Gbps) physical link between the various components of the system. Green dots are Network Elements (NEMS), black dots are line card switches and red dots are fabric card switches. The various ovals represent the compute nodes with their logical identifiers specified inside the oval. The line thickness and dot size is proportional to the number of routes passing through that link or switch. We also use the color of the lines to represent the number of links going through it. Table I provides the mapping of line colors to the number of routes passing through it. As we can see, many links have multiple routes going through them (as indicated by the thickness of the line). Such sharing of physical links can lead to the degradation in performance of the individual applications using the links.

TABLE I
Mapping of Line Color to Number of Links

Color	Number of Links
Black	1
Blue	2
Red	3 - 4
Orange	5 - 8
Green	> 8

In this context, we use the native InfiniBand (verbs) level ping pong benchmark [7] to quantify the performance degradation that may occur if multiple processes use the same HCA / link for communication. We measure the time elapsed between two consecutive network level send / receive operations (i.e., inter packet arrival time) at the sender / receiver side respectively and plot the frequency distribution curve of the elapsed time in each case. In an ideal scenario, the pattern for such a frequency distribution would be a sharp Gaussian curve [8] with peak occurring around the latency value reported by the verbs level test for that message size. There would also be

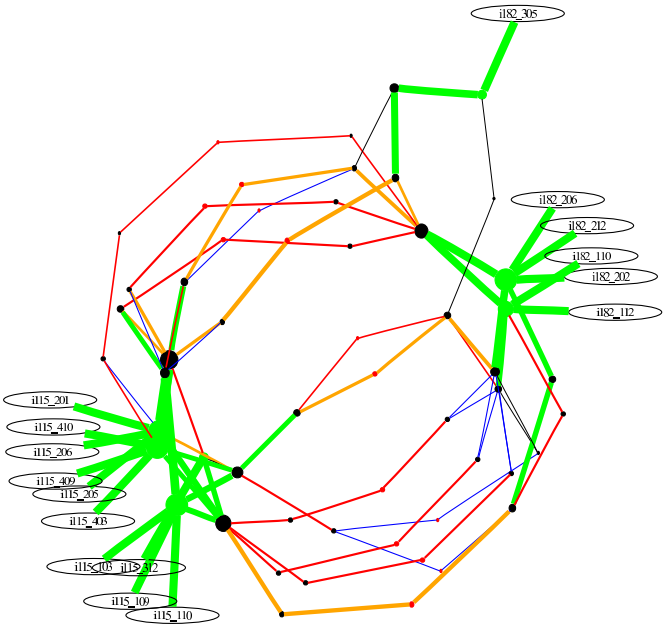

Fig. 1. Routing Diagram of Various Flows on TACC Ranger (Courtesy TACC)

very few values (referred to as *outliers*) that lie far away from the peak. As the number of processes using the system increases, we will see increasingly higher number of *outliers* in the frequency distribution curve. This lack of predictability in the arrival times of packets will show up as increased execution time at the application level. We do not expect the same lack of predictability to affect the results of various benchmarking suites like IMB [9] and OMB [10] due to the large number of repetitions performed.

The verbs level tests were done on a pair of Intel Nehalem E5530 nodes equipped with Mellanox QDR HCA's and PCI-Ex Gen2 interfaces. The two machines were connected through a Mellanox QDR InfiniBand switch. Intel Nehalems have eight cores so one to eight pairs of communication can be performed. As the patterns for the elapsed time between two consecutive send operations (inter packet transmission time) and receive operations (inter packet arrival time) were similar, we only show the frequency distribution graphs for the inter packet arrival time in this paper. Figures 2 (a) and (b) show the pattern of inter packet arrival for 99.999% of 50,000 packets of size 1 MB for one and eight pairs of communicating processes, respectively. As we can clearly see the arrival pattern is much sharper with just one communicating thread than with eight. Such widely dispersed arrival patterns will be more common as the core counts of the systems increase. Although this test was performed for packets of size 1 MB, the patterns observed for other message sizes were similar.

III. Background

In this section we give the necessary background on InfiniBand and the latest QoS support for improving performance.

A. Overview of InfiniBand Architectural Support

InfiniBand Architecture (IBA) [11] defines a switched network fabric for interconnecting processing nodes and I/O

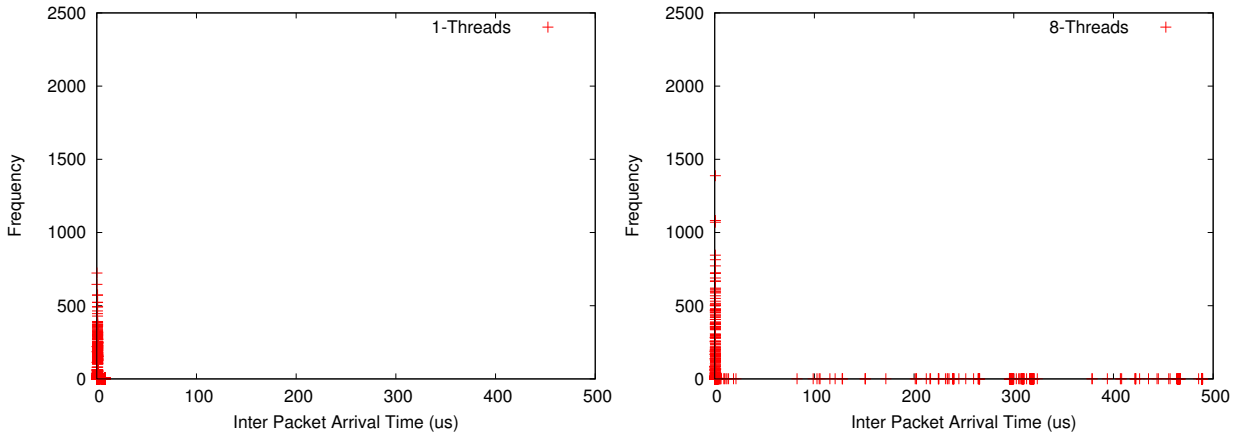

Fig. 2. Distribution of Inter Transmission Time of 99.999% of Packets for (a) One Pair of Processes and, (b) Eight Pairs of Processes

nodes using a queue-based model. A Queue Pair (QP) consists of a send queue and a receive queue. The send queue contains instructions for transmitting data and the receive queue contains the instructions describing where the receive buffer is. At the low level, InfiniBand supports different transport services including Reliable Connection (**RC**) and Unreliable Datagram (**UD**).

IBA supports QoS at two levels, using Service Level (SL) at switch level and Traffic Class (TClass) at router level. SL is a field in the Local Route Header (LRH) of a packet. The QoS mechanism in IBA consists of three architectural components: Virtual Lane (VL), Virtual Lane arbitration and link level flow control. Although the IBA specifies that each port must have a minimum of two and a maximum of 16 virtual lanes, most current day HCA implementations (including the one used in this paper) only offer upto a maximum of 8 virtual lanes. Depending on the type of implementation, each virtual lane can have different transmitting and receiving buffers.

Every network element in the subnet maintains a Service Level to Virtual Lane (SL2VL) mapping table that specifies to which virtual lane packets belonging to a service level should be sent out. It also has a virtual lane arbitration (VLArb) table which defines two strict priority levels - low priority and high priority. The table containing the high priority entries is processed before the one containing the low priority entries. In each priority level, it defines a weighted round-robin scheme of arbitration between the virtual lanes. It indicates the number of data units that can be sent out through a virtual lane before yielding to the next one. These two tables together ensure that each packet will be forwarded according to its agreed upon service level (priority) across the entire network. Note that it is also possible to put a cap on the number of data units that can be sent by all the high priority entries combined before yielding to the low priority entries. This ensures that the applications using the low priority queues are not starved. Finally, link-level flow control protects the QoS on each virtual lane by preventing it from being affected by the traffic on other virtual lanes. At router level, TClass performs functions similar to those of the SL and helps support QoS in a similar way as described above.

All QoS architectural parameters are configured by the InfiniBand subnet manager. We use the popular open source implementation of the InfiniBand subnet manager - OpenSM [12].

ConnectX: ConnectX [13] is the fourth generation InfiniBand Host Channel Adapter (HCA) from Mellanox Technologies. It has two ports, with 8 virtual lanes for each. It provides fine-grained end-to-end QoS and congestion control with the latest drivers. Each port can be independently configured to be used either as InfiniBand or 10 Gigabit Ethernet. Recently, it also added the scalable connection solution XRC [14].

B. MPI

Message Passing Interface (MPI) [15] is one of the most popular programming models for parallel computing. MVAPICH2 [5] is a high-performance MPI implementation over InfiniBand clusters, used by more than 1,000 organizations world-wide. Most MPI applications make use of the RC queue pair for their communication needs. Small message (including small data message and control message) and large message passing have different performance requirements in terms of latency. Although many MPI designs [5], [16] employ different protocols for them, they do not differentiate their priorities. As the hardware-based QoS support is maturing in IBA at multiple layers, including verbs level, it is possible that we can utilize this feature to provide QoS provisioning to different messages in MPI.

C. OFED Perftest

The Open Fabrics Enterprise Distribution (OFED) [17] software stack provides us with a set of native InfiniBand level performance benchmarks used to measure the basic network performance. We modify these benchmarks to keep track of send and receive times and plot the corresponding frequency distribution graphs. We also modify the benchmarks so that it is now able to run multiple concurrent threads. We make this modification so as to measure the ability of the network devices to handle multiple flows.

IV. Design and Methodology

Advances in InfiniBand technology has led to the introduction of many novel features with which we can perform traffic

shaping as well as segregation. Such capability allows us to ensure that multiple flows do not conflict with each other, allowing us to improve their performance and predictability. One such mechanism provided by the InfiniBand standard is the ability to direct desired flows into separate channels called Virtual Lanes (VL). The Subnet Manager (SM) is used to configure all the network devices with the necessary instructions to properly direct traffic. Consequently, once the packets have been assigned to different virtual lanes at the sender side, the InfiniBand fabric ensures that this mapping is maintained throughout the lifetime of the packet in the network.

As we saw in Section II, the predictability arrival of packets degrades when we increase the load on the system. The most obvious reason for this could be contention for network resources at the HCA as well as at the link levels. As long as the total traffic flowing through the physical link does not exceed the capacity of the physical medium, we should not face any problems with transmitting the packet over the physical link. But the criteria for resource contention at the HCA are different. The two resources the HCA has to offer are network buffers and CPU time. Due to the nature of the virtual lane arbitration scheme, the amount of time spent by the processing unit in the HCA at each virtual lane will be proportional to the number of virtual lanes and hence can be considered as constant for our purposes. That leaves the network buffers as the major cause for resource contention at the HCA.

As mentioned in Section III, the network buffer pool of an InfiniBand HCA can be broadly classified as those allocated to a particular virtual lane and those that are common to all the virtual lanes (common buffer pool). Depending on the hardware vendor, there are various ways that the buffers can be split up among the virtual lanes. Figure 3 shows a common implementation followed by InfiniBand HCA vendors.

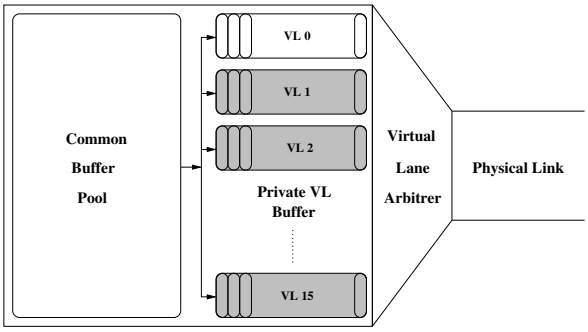

Fig. 3. Buffer Organization in ConnectX IB HCA

No open source MPI implementation, regardless of the underlying communication protocol (RC or UD), leverages virtual lanes as of now. Such usage can result in Head of Line (HoL) blocking [1]. Prior theoretical research in the field of HoL blocking avoidance [2] indicates that using multiple virtual lanes with individual buffers is a good to way to achieve this. In this context we design a scheme, as depicted in Figure 4, to efficiently distribute the traffic across all available virtual lanes, there by reducing contention. Multiple service levels are defined in the MPI library. These service levels can be used to convey any performance characteristics we want. Depending on whether we need distribution or segregation of traffic, we can use one of the following schemes.

- Traffic distribution and,
- Traffic segregation

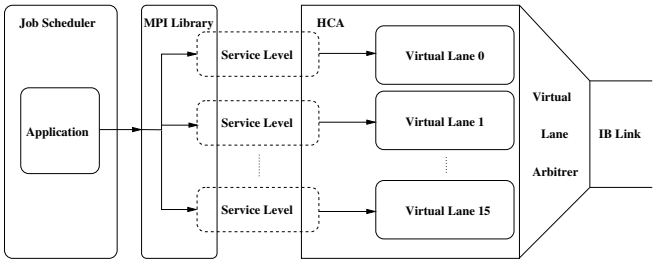

Fig. 4. Traffic Distribution using Multiple VL's

In traffic distribution based scheme, we distribute the traffic of the entire job evenly across all the available virtual lanes. This kind of distribution would be helpful when we have uniform distribution of load across all the processes, i.e. all processes perform same amount of communication - there is no imbalance. But if there are any imbalances in communication or load distribution among the various processes taking part in the application it could be that one of the processes hogs all the resources of the system. In such a scenario, it would be desirable for us to segregate the traffic instead of distributing it evenly across all the virtual lanes. The segregation based scheme can also be used when we need to isolate applications as much as possible from being adversely impacted by other applications running on the same system. As most of the applications tend to distribute the load evenly across all the available processes, we focus mainly on the *Traffic Distribution* scheme in the remainder of the paper. Figure 5 shows the design of the *Traffic Distribution* scheme. As our requirement here is to purely perform some form of load distribution across virtual lanes, we define all the service levels to have similar performance metrics. RC queue pairs are created between processes to transmit data. Packet transmission through multiple virtual lanes takes place in a round robin manner.

One point to note here is that, by increasing the number of virtual lanes, we force the Virtual Lane Arbiter to poll more number of virtual lanes than it normally otherwise would. This could cause a decrease in the amount of time the Virtual Lane Arbiter spends on each virtual lane. Though this could have a possible performance impact, we believe that the benefits gained in terms of performance and predictability will outweigh it. The code has been implemented in such a way that all schemes described in this paper can be controlled at job launch time by the user through the use of environment variables.

V. Performance Evaluation and Results

We detail the results of our experimental evaluation in this section. All tests were run on a quiet system without

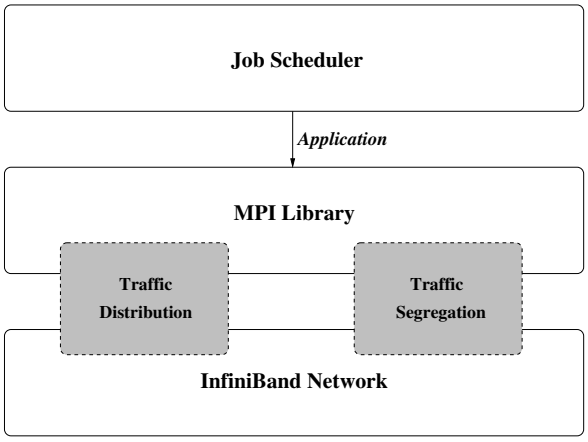

Fig. 5. Overall Design Framework for using Virtual Lanes in MPI

any background traffic. Hence, we expect the improvement in performance seen on production systems with cross traffic to be much higher.

A. Experimental Setup

Figure 6 shows the basic experimental setup. Each node of our testbed has eight Intel Xeon cores running at 2.40 Ghz with 8192 KB L2 cache. The cores are organized as two sockets with four cores per socket. Each node also has 12 GB of memory and Gen2 PCI-Express bus. They are equipped with MT26428 QDR ConnectX HCAs with PCI-Ex interfaces. We use a 36-port Mellanox QDR switch to connect all the nodes. Each node is connected to the switch using one QDR link. The HCA as well as the switches use the latest firmware. The operating system used is Red Hat Enterprise Linux Server release 5.3 (Tikanga). OFED version 1.4.2 is used on all machines, and the OpenSM version is 3.1.6. We use the modified version of OFED perftest [7] for all the verbs level tests. We use the mpiBench collective benchmark suite [18] and CPMD [3] application for higher level tests. All microbenchmark level tests were run for 50,000 packets each.

Fig. 6. Experimental Setup

B. Native InfiniBand Level Performance Results

Here, we analyze the difference in performance observed while running our modified version of the OFED *perftest* mentioned in Section III-C. Eight parallel streams of native IB level latency test are performed for varying message sizes between two host nodes in our experimental testbed. In all our tests, we use eight virtual lanes for multiple virtual lane case. The frequency distribution graphs show the distribution of inter packet arrival time for 99.999% of 50,000 packets.

Figures 7 (a) and (b) show the impact of using multiple virtual lanes on the native IB latency of 2 KB and 4 KB messages respectively. Although using multiple virtual lanes seems to increase the average latency of most the packets by a small amount, it also has the effect of making the pattern more predictable in terms of inter packet arrival times as seen in Figures 8 and 9. The use of multiple virtual lanes also eliminates the number of packets which arrive with extremely high delay, which is the main cause of lack of predictability and performance at the microbenchmark and application levels. The results at the microbenchmark, collective and application levels are detailed in the coming sections.

Figures 8 (a) and (b) shows the frequency distribution of inter packet arrival time for packets of size 2 KB with single and multiple virtual lanes respectively. As we can see from Figure 8 (b), the inter packet arrival times form a very sharp Gaussian curve with very few *outliers* when we use multiple virtual lanes. Such a sharp distribution will result in less delay when running large scale applications. On the other hand, we see many *outliers* on running the same benchmark with all processes using just one virtual lane as depicted in Figure 8 (a). The performance varies on a wide range which is not desirable. Furthermore, we observe that the largest inter packet arrival time in the case of one virtual lane is almost 10 times worse than what is seen when we use multiple virtual lanes. This result hints at the possible level of process skew that can occur if the virtual lanes are used in an over subscribed manner.

Figures 9 (a) and (b) show the behavior for 4 KB message size. As we can see, the behavior for both the 4 KB and 2 KB messages are similar. We also observe that, by using multiple virtual lanes we improve the predictability of the results almost by a factor of 10.

C. Microbenchmark Level Performance Results

We ran various microbenchmark level tests from the OMB [10] suite. As we were focusing mainly on the effect that multiple streams of traffic have on each other, we used the multi-pair bandwidth, multi-pair message rate and the multi-pair latency tests. These tests were run with eight pairs of processes across two of the nodes in our experimental setup. As expected, little variation was seen with very small message sizes as enough data was not generated to cause congestion even for one virtual lane. Very large messages on the other hand are capable of using up all the available link bandwidth so that the effects of traffic distribution are effectively nullified.

Figures 10, 11 and, 12 shows the performance of medium message sizes in multi-pair bandwidth, multi-pair message rate and multi-pair latency tests respectively. We can see that using multiple virtual lanes has a significant impact on the performance. We observe up to a 15% improvement in

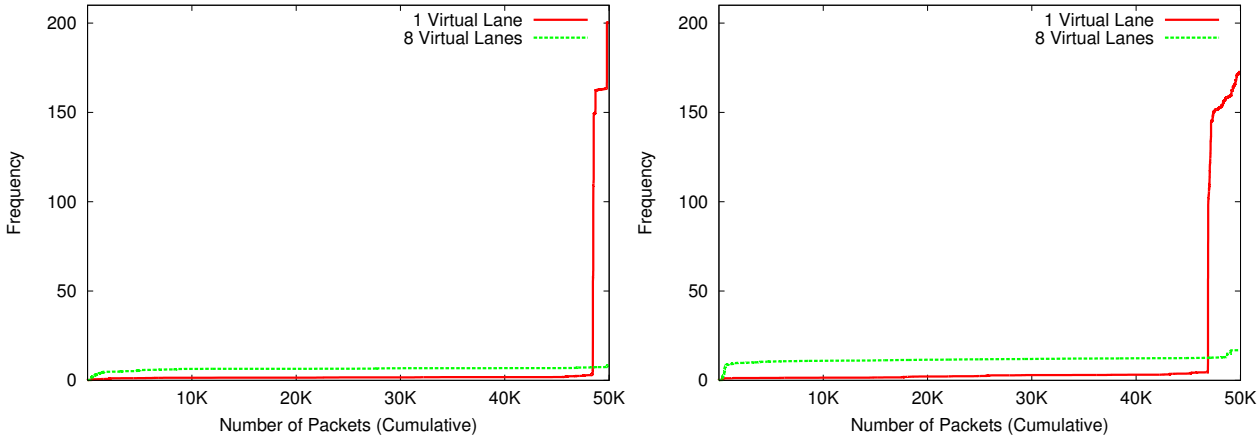

Fig. 7. Impact of using Multiple Virtual Lanes on Native IB Latency for (a) 2 KB Message Size and, (b) 4 KB Message Size

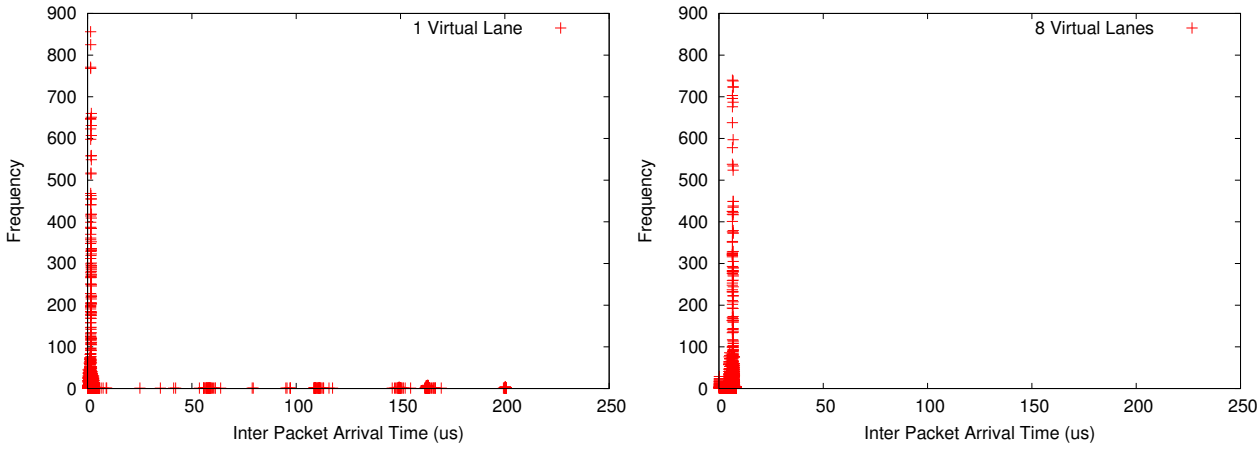

Fig. 8. Impact of Traffic Distribution on Native IB Performance for 2 KB Message Size with (a) One VL and, (b) Eight VL's

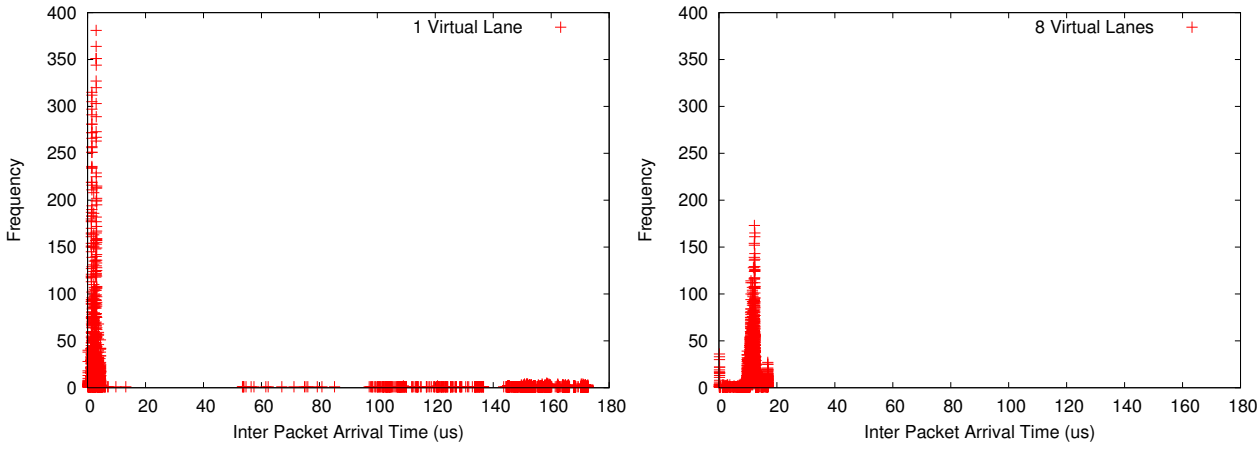

Fig. 9. Impact of Traffic Distribution on Native IB Performance for 4 KB Message Size with (a) One VL and, (b) Eight VL's

performance of 2 KB message size in the multi-pair bandwidth and from 20% - 30% improvement in performance of 64 KB - 512 KB message size in the multi-pair latency tests.

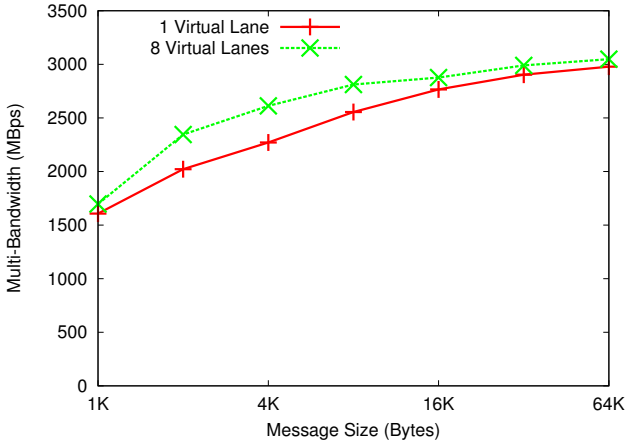

Fig. 10. Impact of Traffic Distribution on Multi-Pair Bandwidth of Medium Size Messages

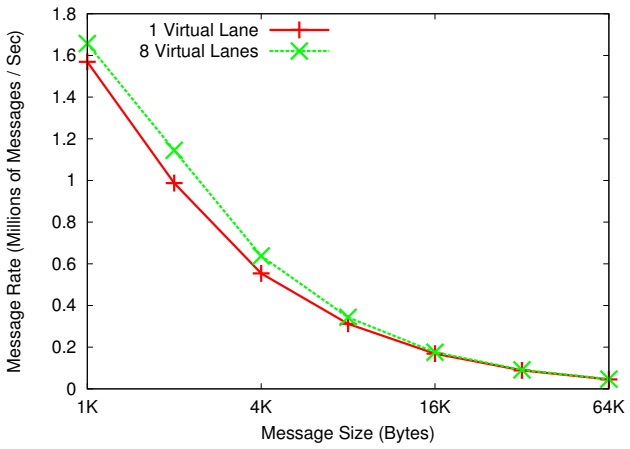

Fig. 11. Impact of Traffic Distribution on Multi-Pair Message Rate of Medium Size Messages

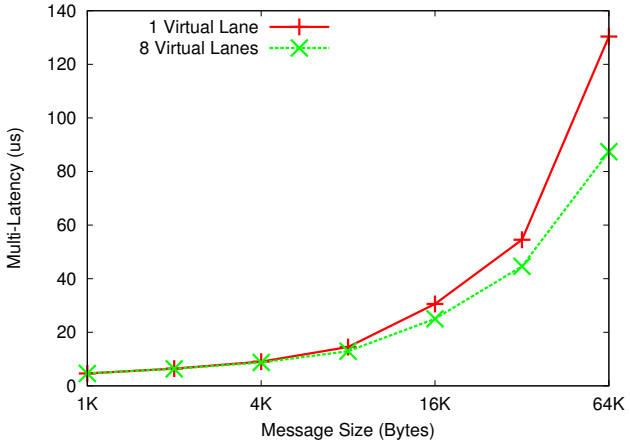

Fig. 12. Impact of Traffic Distribution on Multi-Pair Latency of Medium Size Messages

D. Collective Level Performance Results

We examine the performance of various collective algorithms in this section. A stripped down version of the *mpiBench* collective benchmark suite [18] is used for our tests. Figure 13 shows the performance of a 64 process Alltoall collective operation for medium message sizes. We see that a performance benefit of up to 20% can be obtained through the use of multiple virtual lanes as opposed to a single virtual lane. Figures 14 (a) and (b) shows distribution of execution times of the Alltoall operation for 4 KB message size. As we can see, the average execution time of the collective operation increases when we have only one virtual lane as opposed to eight virtual lanes. This is a clear indicator that we should use multiple virtual lanes in order to enhance the performance of collective operations.

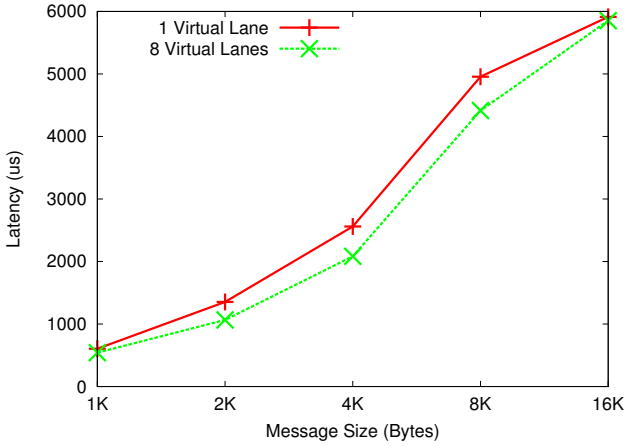

Fig. 13. Impact of Traffic Distribution on Performance of 64 Process Alltoall Collective for Medium Message Sizes

Figures 15 (a) and (b) show the distribution of execution times of the Gather collective algorithm for a message size of 512 bytes with one virtual lane and multiple virtual lanes respectively. Figures 16 (a) and (b) show similar performance numbers for the Allgather collective algorithm. As we can see, irrespective of the underlying communication algorithm in use, using multiple virtual lanes improves the predictability of collective algorithms by reducing the amount of delay introduced. This enhanced predictability will show up as improvement in run times at application levels.

E. Impact of Traffic Segregation on Performance of Collectives

Figure 17 shows the impact of using multiple virtual lanes for segregating traffic into different flows. We run two eight process (4x2) Alltoall collective operations between two of the nodes in our test setup. Due to limitations of our experimental setup, both sets of operations are run on the same pair of nodes. As the nodes have eight cores each, there is no over subscription of cores that occur here. The *Base* line shows the performance that would have been obtained if only one instance of the eight process Alltoall operation was executing on the nodes (only half of the total available cores are utilized on either system). *With Segregation* and *Without Segregation* shows the average performance of two eight Alltoall collective operations executing on the same nodes with and without the use of multiple virtual lanes to segregate the traffic respectively (in effect, a 16 process Alltoall). We can see that the use of

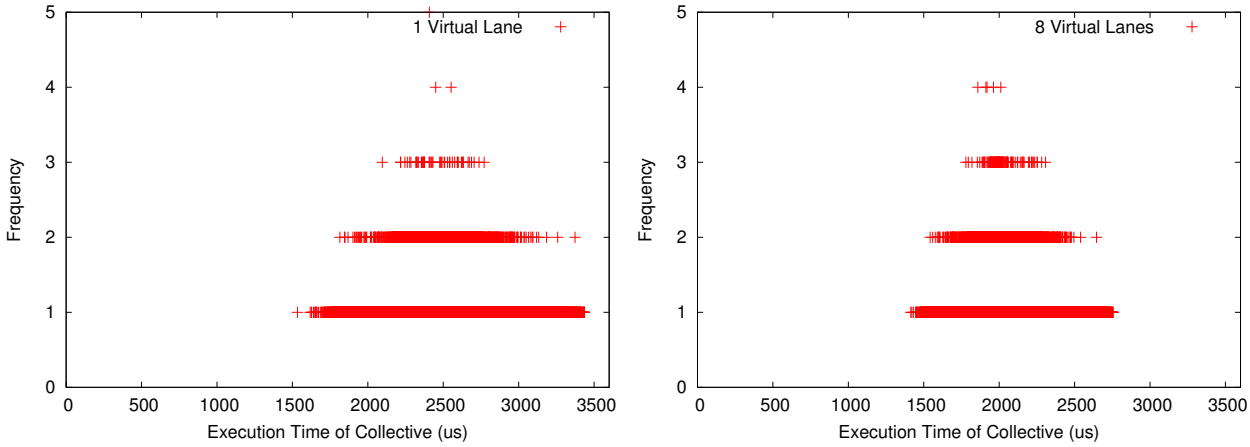

Fig. 14. Impact of Traffic Distribution on Performance of 64 Process Alltoall for 4KB Message Size with (a) One VL and, (b) Eight VL's

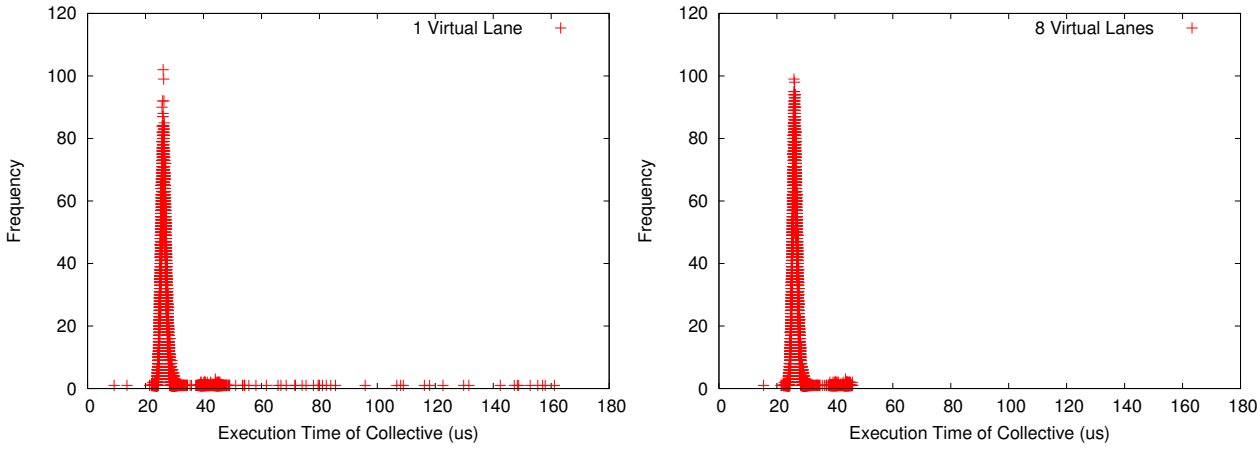

Fig. 15. Impact of Traffic Distribution on Performance of 64 Process Gather for 512 Byte Message Size with (a) One VL and, (b) Eight VL's

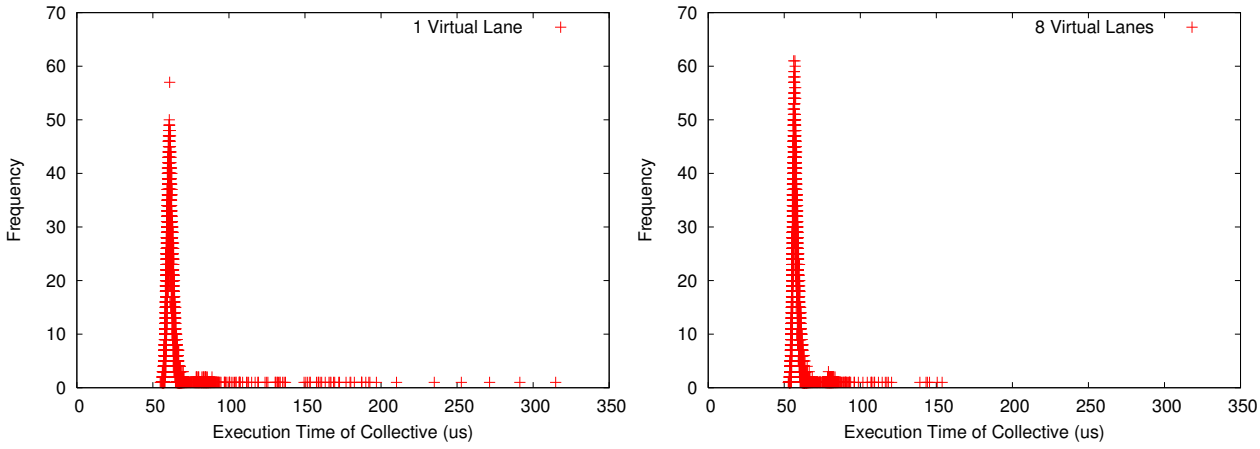

Fig. 16. Impact of Traffic Distribution on Performance of 64 Process Allgather for 512 Byte Message Size with (a) One VL and, (b) Eight VL's

multiple virtual lanes helps us improve the performance of the Alltoall operation for medium message sizes by up to 12%.

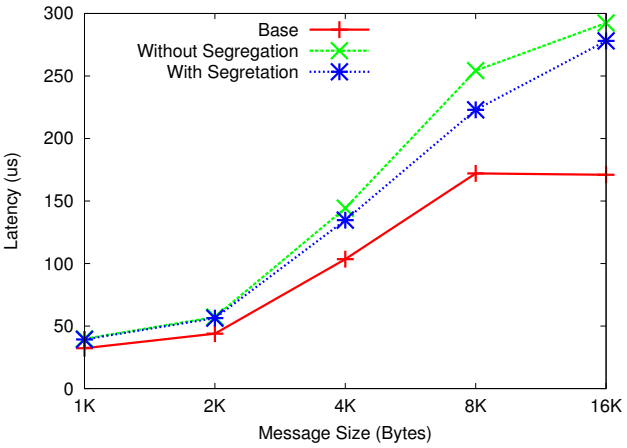

Fig. 17. Impact of Traffic Segregation on Performance of Alltoall Collective for Medium Message Sizes

F. Application Level Performance Results

We show the performance results obtained by running a 64 process CPMD job on a sample work-set in this section. As in previous sections, we repeat the experiments with one and multiple virtual lanes. Figure 18 shows the performance comparison of average of multiple runs with one and multiple virtual lanes in terms of normalized time. We normalize the time against the time taken to execute the application with one virtual lane. Apart from the total execution time, we also profile the main communication time by *Alltoall* collective. For 64 process runs of the CPMD application, the time taken by Alltoall accounts for about 20% of the total execution time. We observe from our experiments that we get an overall gain in performance about 6% and a 11% gain in the performance of the time taken for *Alltoall* collective (which forms a significant part of communication of any CPMD application).

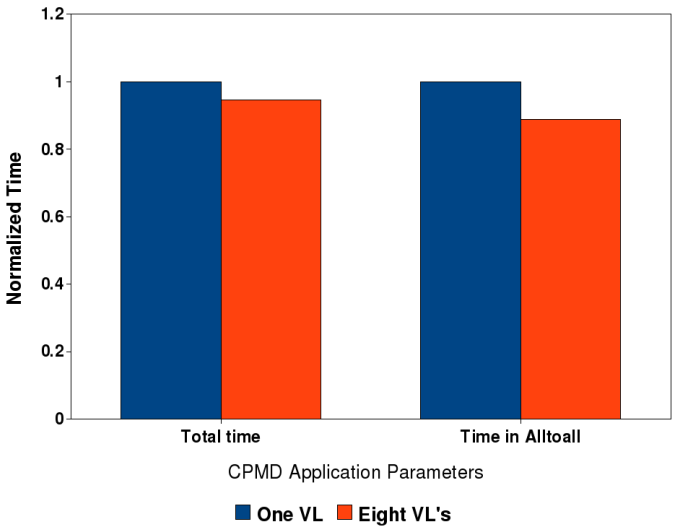

Fig. 18. Impact of Traffic Distribution on Performance of 64 Process CPMD Application

VI. Related Work

Some researchers have investigated how to use the IB QoS support to improve performance. In [19], authors defined the QoS problem and provided a generalized approach using IBA mechanisms. The papers [20], [21] studied how to configure InfiniBand VL arbitration table to optimize required performance, and how to manage the time sensitive traffic through IBA QoS support, respectively. In [22], the same group also proposed an effective strategy for configuring the IBA networks to provide QoS for each kind of traffic, and improved the arbitration algorithm in [22] to maximize the number of requests to be allocated in the arbitration table that the output ports have. They further made a lot of efforts [23], [24], [25] on improving the cluster QoS based on IBA supported mechanism and trying to establish a unified system architecture. Although all of these research stems from InfiniBand QoS specification, they mainly focus on the theoretical exploitation based on simulation. On the contrary, in [26] authors made investigation on real IB cluster and experimented its QoS provisioning capability by assigning different service levels to different traffic. The paper also presents the possible sources of bottleneck that constrains the application to achieve desired QoS performance. Other works in the area of congestion and flow control include [27], [28], [29]. Multiple ways exist that allow us to decouple the process from the skews introduced by the communication operations. The use of non-blocking collectives [30], [31], [32], [33], [34] is one of the proposed approaches. This is also slated to go into the MPI-3 standard [35]. Offloading communication operations to the HCA is another emerging trend intended to decouple this [36] [37]. Both these approaches are complimentary to our work as our scheme can be used to enhance the performance of either one of these approaches. More details about other aspects of our work are included in the OSU technical report [38].

VII. Conclusions and Future Work

In this paper we explore the use of virtual lanes provided by the InfiniBand standard to improve the predictability and performance of high performance computing applications. We explore how the use of multiple virtual lanes helps to reduce the number of *outliers* in InfiniBand communication. The experiments run at the verbs level clearly shows the impact multiple concurrent streams can have in the arrival patterns of InfiniBand packets. We integrate our scheme into the MVAPICH2 MPI library and perform various experiments with collective benchmarks as well as end applications. The results of our experimental evaluation show that the use of multiple virtual lanes can improve the predictability of message arrival by up to 10 times in presence of network congestion. Our microbenchmark level evaluation with multiple communication streams show that the use of multiple virtual lanes can improve the bandwidth / latency / message rate of medium sized messages by up to 13%. Through the use of multiple virtual lanes, we are also able to improve the performance of the Alltoall collective operation for medium message sizes by up to 20%. Performance improvement of up to 12% is also observed through segregation of traffic into

multiple virtual lanes when multiple jobs compete for the same network resource. We also see that our scheme can improve the performance of collective operations used inside the CPMD application by 11% and the overall performance of the CPMD application itself by up to 6%.

As part of future work, we plan to delve deeper into the reasons for the small degradation in latency observed during the use of multiple virtual lanes. As well as explore the ideas of *Inter Job* and *Intra Job* bandwidth provisioning. We also plan to conduct large scale experiments on supercomputing clusters like TACC Ranger to study the impact the proposed schemes have on production systems.

VIII. Acknowledgment

We thank Dr. Bill Barth and Dr. Karl Schulz from TACC for their help in preparing the routing diagrams for the TACC Ranger system. We like to thank Todd Wilde and Jim Mott from Mellanox for the help they extended. We also thank Dr. Fabrizio Petrini for the use of the code used to generate the frequency distribution graphs.

References

[1] M. Karol, M. Hluchyj and S. Morgan, "Input Versus Output Queuing on a Space-Division Packet Switch," in *IEEE Transactions on Communications*, Dec 1987.

[2] M. E. Gomez, J. Flich, A. Robles, P. Lopez and J. Duato, "VOQ-SW: A Methodology to Reduce HOL Blocking in InfiniBand Networks," *Parallel and Distributed Processing Symposium, International (IPDPS)*, 2003.

[3] "CPMD Consortium," http://www.cpmd.org/.

[4] T. Hoefler, T. Schneider, and A. Lumsdaine, "The Effect of Network Noise on Large-Scale Collective Communications," *Parallel Processing Letters (PPL)*, Dec. 2009, accepted for publication (in print).

[5] MVAPICH2: High Performance MPI over InfiniBand and iWARP, http://mvapich.cse.ohio-state.edu/.

[6] Texas Advanced Computing Center, "Ranger," http://www.tacc.utexas.edu/resources/hpcsystems/.

[7] "CPMD Consortium," http://www.openfabrics.org/downloads/perftest/.

[8] "Gaussian Distribution," http://www.cs.princeton.edu/introcs/11gaussian/.

[9] "Intel MPI Benchmark," http://software.intel.com/articles/intel-mpi-benchmarks.

[10] OSU Micro-benchmarks, http://mvapich.cse.ohio-state.edu/benchmarks/.

[11] Infiniband Trade Association, http://www.infinibandta.org.

[12] OFED, "Open Fabrics Alliance," http://www.openfabrics.org/downloads/management/README.

[13] Mellanox Technologies, "ConnectX Host Channel Adapters," http://www.mellanox.com/pdf/products/hca/ConnectX_IB_Card.pdf.

[14] M. Koop, J. Sridhar and D.K. Panda, "Scalable MPI Design over InfiniBand using eXtended Reliable Connection," in *Cluster'08*, 2008.

[15] MPI Forum, "MPI: A Message Passing Interface," in *Proceedings of Supercomputing*, 1993.

[16] MPICH2: High Performance portable MPI implementation, http://www.mcs.anl.gov/research/projects/mpich2.

[17] Open Fabrics Enterprise Distribution, http://www.openfabrics.org/.

[18] D. Grove and P. Coddington, "Precise mpi performance measurement using mpibench," in *In Proceedings of HPC Asia*, 2001.

[19] J. Pelissier, "Providing Quality of Service over InfiniBand Architecture Fabrics," in *In Proceedings of the 8th Symposium on Hot Interconnects*, 2000, pp. 127–132.

[20] F. J. Alfaro, "A Strategy to Compute the InfiniBand Arbitration Tables," in *IPDPS '02: Proceedings of the 16th International Symposium on Parallel and Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2002, p. 6.2.

[21] F. J. Alfaro, J. L. Sánchez, and J. Duato, "A Strategy to Manage Time Sensitive Traffic in InfiniBand," in *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*. Washington, DC, USA: IEEE Computer Society, 2002, p. 47.

[22] F. Alfaro, J. Sanchez, and J. Duato, "A New Proposal to Fill in the InfiniBand Arbitration Tables." Los Alamitos, CA, USA: IEEE Computer Society, 2003.

[23] F. Alfaro, J. Sanchez, M. Menduina, and J. Duato, "A Formal Model to Manage the InfiniBand Arbitration Tables Providing QoS," vol. 56, no. 8, 2007.

[24] A. Martinez, G. Apostolopoulos, F. Alfaro, J. Sanchez, and J. Duato, "Efficient Deadline-Based QoS Algorithms for High-Performance Networks," vol. 57, no. 7, July 2008, pp. 928–939.

[25] Martinez R and Alfaro F.J and Sanchez J.L , "A Framework to Provide Quality of Service over Advanced Switching," 2008.

[26] R. R. Grant, M. J. Rashti and A. Afsahi, "An Analysis of QoS Provisioning for Sockets Direct Protocol vs IPoIB over Modern InfiniBand Networks," in *P2S2 Workshop, in conjunction with ICPP*, 2008.

[27] Frank Olaf Sem-Jacobsen, Sven-Arne Reinemo, Tor Skeie, Olav Lysne, "Achieving Flow Level QoS in Cut-Through Networks Through Admission Control and DiffServ," in *HPCC*, 2006.

[28] 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*, 2005.

[29] C. Minkenberg and M. Gusat, "Speculative Flow Control for High-Radix Datacenter Interconnect Routers," March 2007, pp. 1–10.

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

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

[32] C. Kaiser, T. Hoefler, B. Bierbaum, and T. Bemmerl, "Implementation and Analysis of Nonblocking Collective Operations on SCI Networks," in *Proceedings of the 23rd IEEE International Parallel & Distributed Processing Symposium, CAC'09 Workshop*, May 2009.

[33] T. Hoefler, F. Lorenzen, and A. Lumsdaine, "Sparse Non-Blocking Collectives in Quantum Mechanical Calculations," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 15th European PVM/MPI Users' Group Meeting*, vol. LNCS 5205. Springer, Sep. 2008, pp. 55–63.

[34] T. Hoefler and A. Lumsdaine, "Optimizing non-blocking Collective Operations for InfiniBand," in *Proceedings of the 22nd IEEE International Parallel & Distributed Processing Symposium, CAC'08 Workshop*, Apr. 2008.

[35] "Message Passing Interface Forum," http://www.mpi-forum.org/.

[36] R. Graham, S. W. Poole, P. Shamis, G. Bloch, N. Boch, H. Chapman, M. Kagan, A. Shahar, I. Rabinovitz, and G. Shainer, "Overlapping Computation and Communication: Barrier Algorithms and Connectx-2 CORE-Direct Capabilities," *The 10th Workshop on Communication Architecture for Clusters*, 2010.

[37] R. Graham, S. W. Poole, P. Shamis, G. Bloch, N. Boch, H. Chapman, M. Kagan , A. Shahar, I. Rabinovitz and G. Shainer, "ConnectX2 InfiniBand Management Queues: New support for Network Offloaded Collective Operations," *The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2010.

[38] H. Subramoni, P. Lai, and D. K. Panda, "Exploiting InfiniBand Architectural Features to Design QoS-Aware MPI," The Ohio State University, Tech. Rep. OSU-CISRC-12/08-TR02, 2008.