Reducing Network Contention with Mixed Workloads on Modern Multicore Clusters

Matthew J. Koop, Miao Luo and Dhabaleswar K. Panda
Network-Based Computing Laboratory
The Ohio State University
Columbus, OH USA
Email: { koop, luom, panda }@cse.ohio-state.edu

***Abstract*—Multi-core systems are now extremely common in modern clusters. In the past commodity systems may have had up to two or four CPUs per compute node. In modern clusters, these systems still have the same number of CPUs, however, these CPUs have moved from single-core to quad-core and further advances are imminent. To obtain the best performance, compute nodes in a cluster are connected with high-performance interconnects. On nearly all clusters, the number of network interfaces is the same on current multi-core systems as in the past when there were fewer cores per node. Although these networks have increased bandwidth with the shift to multi-core, there still exists severe network contention for some application patterns.**

In this work we propose mixed workload (non-exclusive) scheduling of jobs to increase network efficiency and reduce contention. As a case-study we use Message Passing Interface (MPI) programs on the InfiniBand interconnect. We show through detailed profiling of the network that accesses of the network and CPU of some applications are complementary to each other and lead to increased network efficiency and overall application performance improvement. We show improvements of 20% and more for some of the NAS Parallel Benchmarks on quad-socket, quad-core AMD systems.

I. Introduction

Over the last few years, multi-core systems have come to dominate computing as well as high-performance computing platforms. In the past there may have been only two or four single-core processors per node, now platforms can have 16 or more cores per node. However, to efficiently utilize the full benefit of multicore systems, many problems that may have been minor in single-core systems need to be overcome. The network is one of these areas that can quickly become saturated in a multi-core system.

Nearly all of the resources in multi-core system, such as memory, network, and I/O devices, are shared by processes on a single node. An issue of increasing importance is how to schedule jobs on high-performance clusters to best utilize these shared resources more efficiently and to avoid constraints. "Symbiotic space-sharing" [1] has been suggested as a technique to alleviate pressure on shared resources for Massive Parallel Processing (MPP) systems by executing parallel applications in a mixed pattern. Areas that have been explored were largely to reduce contention for memory and disk resources.

This previous work, however, has not deeply explored the network. Some of the issues, such as the memory bandwidth are being addressed by NUMA systems developed by AMD and Intel. The question of how to effectively use the network across jobs, however, is still an open issue.

In this paper, we seek to address this issue of network contention. We propose splitting jobs up by network-level access patterns to maximize performance. Our case studies are all carried out using Message Passing Interface (MPI) [2], which is one of the most popular programming models for cluster computing. We also focus on the InfiniBand interconnect since it is used on over 30% of the clusters in the Top 500 supercomputing list. The concept should be applicable to other networks and programming models as well.

We first use microbenchmarks to show the effects of network contention. We then use the NAS Parallel Benchmarks as examples to determine the effect of network patterns on performance. We show up to a 20% improvement in application performance numbers over a traditional exclusive scheduling by using mixed workloads with complementary applications that share the same set of nodes and network interfaces.

As part of our evaluation we develop a network profiling tool to determine the detailed usage of the network. We can directly view different time slices and view how different applications interact. This differs from other existing tools that simply look at MPI-level message patterns. We instead can directly track when InfiniBand messages are sent. We include this analysis when discussing application communication patterns.

The rest of the paper is organized as follow: We start with related work in Section II. We give the necessary background on InfiniBand in Section III. In Section IV, we introduce the concept of mixed workloads and the motivation. We describe our design for capturing network-level traffic patterns from different applications in Section V. In Section VI we use microbenchmarks and analyze the performance of pairing different NAS Parallel Benchmarks together to look at speedups possible by reducing network contention. We conclude and point out future work in Section VII.

II. Related Work

Many other researchers have previously looked at job scheduling based on the alleviation of shared resource contention, however, these studies have been focused on cache, memory bus and synthesized resource contention.

Cache contention is a critical topic in job scheduling for multicore systems along with additional cache sharing between cores on the same node. Kim et al. [3] studied the fairness in cache sharing between threads in a CMP architecture. Jiang et al. [4] proposed an algorithm that finds the optimal co-schedules in polynomial time for job co-scheduling systems to reduce cache contention.

Liedtke et al. [5] first raised the memory bus contention problem in SMP system, where multiple processors on n-process system share a single memory bus and propose memory-bus scheduling. Kondo et al. [6] proposed a technique to mitigate memory bus and memory bank contention by controlling execution speed of each thread running on each core through Dynamic Voltage and Frequency Scaling (DVFS).

Weinberg and Snavely proposed "symbiotic space-sharing" [1], which is a technique to alleviate pressure on shared resources on MPP system by executing parallel applications in combinations. They first investigated constraints on memory and I/O resources for modern parallel systems and then proposed a prototype symbiotic scheduler to implement symbiotic space-sharing. This work is the most similar to ours, however, we are investigating network-level accesses and contention.

Cong et al. [7] have addressed job scheduling problem on Reconfigurable high-performance computing (RHPC) systems. They have proposed algorithm to assign jobs to processors with consideration of coprocessors at global optimization steps and coprocessor selection in the local optimization step.

Sondag et al. [8] propose an approach for automatic thread-to-core assignment for heterogeneous multicore processors, which are presented to address the matching problem of resources needs of a thread and resource availability at the assigned core. They first use a preliminary static analysis-based approach for determining similarity among program sections, then they use a thread-to-core assignment algorithm to make scheduling decision based on statically generated information and execution information from a small fraction of the program.

The cache and memory bus contention are also being addressed by vendors, while network contention problem still remains as an open topic. To the best of our knowledge, our work is the first attempt to address the severe network contention that exists in modern multicore systems, profile network-level communication information and propose solutions to avoid such contention.

III. InfiniBand

InfiniBand [9] was designed as a high-speed, general-purpose I/O interconnect, and in recent years it has become a popular interconnect for high-performance computing to connect commodity machines in large clusters.

The communication model in InfiniBand is based off of Queue Pairs (QPs). A QP consists of two queues, a Send Queue (SQ) and a Receive Queue (RQ) for initiating send and receive operations, respectively. Each QP is associated with a Completion Queue (CQ), allowing an application to poll or use an event-based interface to receive notification of operation completion. All completions, both send and receive operations are placed into the CQ.

There are two sets of communication semantics in InfiniBand: channel and memory semantics. Channel semantics include send and receive operations that are similar to those found in traditional interfaces, such as sockets, where both sender and receiver must be aware of communication. Memory semantics include one-sided operations where one host can access or modify memory on a remote node without a posted receive; such operations are referred to as Remote Direct Memory Access (RDMA).

To receive a message on a QP using channel semantics, a receive buffer must be posted to that QP. Buffers are consumed in a FIFO ordering. Using a Shared Receive Queue (SRQ) allows receive buffers to be shared across QPs for scalability.

IV. Reducing Network Contention

Traditional scheduling of jobs assumes that each node is exclusive to a single job. In this way it is known that there will not be any other processes on the node other than those in the same job. There are benefits

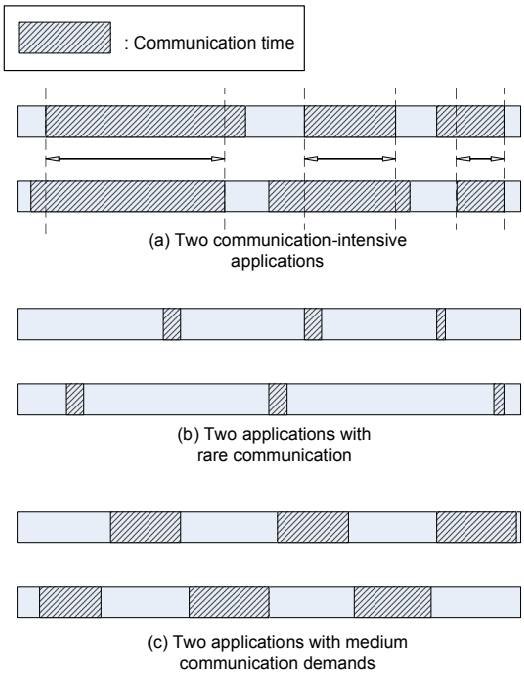


Figure 1. Application Sharing Combinations

to this mode including allowing shared memory communication for more processes. However, since many applications are very synchronous in communication, all of the processes within a job will typically require simultaneous access to the network and thus create a significant bottleneck.

A. Shared vs. Exclusive

To alleviate the contention for the shared network resource, one method is to reduce the number of processes on a single node. Using this method may reduce contention, but cores will remain idle and is not an efficient use of resources. To keep all the cores utilized but achieve less contention in shared resource, one possible method is to break the synchronization among processes on the single node. The simplest implementation is to instead of having a single application occupying m nodes with n cores on each node, we can schedule k applications sharing $k*m$ nodes. Then on each node, every application can use n/k cores.

We term the traditional scheduling as *Exclusive-Full mode*, and the new scheduling method of sharing the compute nodes as *Shared mode*.

When sharing nodes for two concurrent jobs, the total number of cores required by the application is the same as in Exclusive-Full mode, however, double the number of nodes is required. Another job can run simultaneously on the other cores on the nodes. Given the synchronicity of many applications, it means that half of the cores will use the network simultaneously instead of all of them. Note that this concept could be further extended to have only 1/4 of the cores for a job with four concurrent jobs.

B. Sharing Combinations

As a result of this new mapping of processes, overlapping link usage can be reduced. There are several possible combinations that can occur when splitting the cores of the node:

- *Dual Communication-Intensive*: Figure 1(a) shows this kind of combination. This is the worst case, when both of the two applications use the same network link to communicate with other nodes frequently, especially when the message size is large. In this situation, the results of shared mode may be similar as that of exclusive mode.
- *Dual Communication-Infrequent*: Figure 1(b) shows this mode where communication is rare or only small amounts of data are transferred. Even in exclusive mode, this kind of application seldom has network contention, so only trivial improvement can be gained in shared mode.
- *Communication Compatible*: These combinations generally include one application that communicates infrequently and one that communicates more often. This class also includes applications that may make frequent use of the network, but generally less than 50% of the time. This ideal situation is shown in Figure 1(c). The communication patterns of the two applications are complimentary with each other. Every application is able to have the same performance as if it has exclusive access to the network.

In the ideal *Communication Compatible* mode, if the link contention is the main performance bottleneck, the performance of shared mode should be the same as the performance of *Exclusive-Half* mode. By Exclusive-Half mode, we refer to a job working on half of the cores on each node while the other half of the cores are idle.

In this paper we will evaluate the performance of different application combinations. We will also trace and examine the communication patterns of different applications. Using this data it should be possible to create the best application sharing arrangements by combining most complimentary applications together based on the communication patterns exhibited.

V. Network Profiling Design

In this section we discuss existing profiling tools for MPI and InfiniBand. Then we discuss our proposed profiling method and the various techniques that can be employed in the absence of perfect timestamps.

A. Existing Profiling Techniques

Current MPI profiling tools such as mpiP [10], Sun Studio Analyzer [11], Vampir [12] and others can be exceedingly beneficial to look at MPI message patterns. These tools rely on the PMPI interface of MPI that allows tools to interposition the tool between the application and the MPI library. This PMPI interface allows current MPI tools to view when the application sends a message or makes a collective call. They can also track the completion of a MPI call as well.

While these are extremely useful for debugging, this interface lacks the detailed knowledge of the network that is needed. For example, many MPIs will copy smaller messages into another buffer and then send the message. According to the MPI semantics, the send can be marked complete once the send buffer is free again. This means that the PMPI interface is disconnected from the actual network operations and and cannot capture the network traffic behavior accurately.

B. Proposed Profiling Design

In this subsection we explore network profiling options for tracking message patterns across processes on the same node.

1) Tracking Messages: The only entities that know when messages are sent in InfiniBand are the HCA and the MPI library. Since InfiniBand is an OS-bypass network there are no kernel calls in the send/receive path. Thus, there is no opportunity to track such usage in the kernel as it is possible with TCP/IP.

The HCA would be an ideal place to track these send requests, however, there are not any options given by the hardware manufacturers to get this information.

Thus, we propose to use profiling from within the MPI library of each process. The MPI library knows when each message is sent. We can record a CPU timestamp for each message that the MPI library sends. Thus, messages can be tracked for every process on a node that is using a given HCA.

2) Message Send Completion: As discussed in Section III, InfiniBand has an asynchronous interface for sending and receiving messages. When a message send is complete, an entry is placed into the CQ. Unless the library uses an interrupt to detect completion, which incurs high overhead, the only method is to poll the CQ for the completion. Thus, since MPIs over InfiniBand generally poll only when in the MPI library, the time when the send completion is noticed may be significantly after when the message was sent.

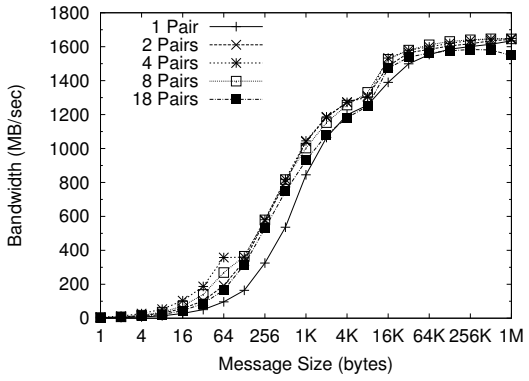


Figure 2. Aggregate bandwidth with increasing numbers of concurrent pairs

3) Extrapolating Data: Since the message completion data may not be current, we devise a method for determining message completion after collecting all data from each of the processes on a node.

We base our model of the network based on a few experiments. We take throughput numbers of increasing numbers of pairs of processes. For example, we collect aggregate bandwidth from concurrent bandwidth tests between multiple pairs of processes. Figure 2 shows an example of this data. This allows us to determine approximate completion times for messages.

For example, if Process 0 of Job A started sending out a message whose size is L through network port of Node 0 at time *t_start*, assume the bandwidth is w, then Process 0 will occupy this network port of Node 0 until $t\_start + L/w$. But if Process 1 on Node 0 started another sending action before $t\_start + L/w$, Process 1 has to "wait" until Process 0 finished transferring its message, so the bandwidth is approximated based on the above throughput experiments. We can roughly approximate the bandwidth using a modified leaky bucket method.

Developing a model for the reception of messages is somewhat more difficult. We can use the completion time for a base and then use a reverse approximation in much the same way as the send operations.

From this data we can develop a timeline of the communication on a single node. This gives insight into the performance of how applications perform when they share a network interface. We believe such profiling would also be beneficial to other application writers to understand how to better utilize the network within even

a single job.

VI. Experimental Evaluation

In this section we evaluate the concept of mixed workload scheduling. We first describe the experimental platform and methodology and then look into the performance of different application kernel combinations using our profiling tool.

A. Experimental Setup

Our experimental platform is a 128-core InfiniBand Linux cluster. Each of the 8 compute nodes has 4 sockets each with a Quad-Core AMD Opteron 8350 2GHz Processor with 512KB L2 cache and 2 MB L3 cache per core. Each node has a Mellanox MT25418 dual-port ConnectX HCA. InfiniBand software support is provided through the OpenFabrics/Gen2 stack [13], OFED 1.3 release.

We implement our profiling design into the MVAPICH2 MPI library. MVAPICH2 [14] is an MPI-2 implementation over InfiniBand, which is optimized for InfiniBand. MVAPICH and MVAPICH2 are currently being used by more than 940 organizations worldwide.

B. Methodology

In this evaluation we use three different execution modes:

- **Exclusive-Full**: This is the traditional job scheduling where each node only contains processes from a single job.
- **Exclusive-Half**: This combination is the same as Exclusive-Full, but the processes are spread out over twice as many nodes and the half the cores of each node are idle. This shows the maxiumum speedup possible using a mixed workload.
- **Shared**: This is the proposed mixed workload scheduling where there are two jobs sharing each node.

Figure 3 is a simple example that shows the difference between Shared mode and Exclusive-Full mode. Job A and Job B both have 64 processes. In the exclusive mode, Job A exclusively occupies node 1 to node 4, Job B exclusively occupies node 5 to node 8; In Shared mode, Job A and Job B share each node by taking up half of the CPUs separately.

Figure 3(b) shows a basic configuration of Shared mode. However, more complicated configurations, such as unbalanced cores distribution based on the communication pattern of different combinations, or k (>2) jobs combination are also possible. However, in the rest of this paper, we will restrict our study to the case of only two jobs per node to explore the existence of network contention and its effects. The investigation of more complicated combinations will be left as future work. Also, since AMD Opteron 8350 in our experiment platform has a 2MB L3 cache shared by CPUs on the same socket, we map processes of the two different jobs on different sockets, in order to avoid possible effects of the shared L3 cache.

Table I
AVERAGE LOOP COMMUNICATION EXECUTION TIME FOR THE MICROBENCHMARK

	Exclusive-Half	Shared	Exclusive-Full
Sleeping time = 0s	0.14s	0.27s	0.27s
Sleeping time = 1s	0.14s	0.18s	0.27s

C. Microbenchmark Evaluation

To explore the existence and strength of link contention in multicore systems, we use a microbenchmark to test the bandwidth and communication time. The microbenchmark performs `MPI_Alltoall` functions and sleep operations (to simulate computation) in a loop. In every loop, every process sleeps for *n* second(s) and then performs one `MPI_Alltoall` and one `MPI_Barrier`. Since this microbenchmark focuses only on exchanging messages, no file I/O or computation exists. We run the microbenchmark in three settings as mentioned in Section VI-B: 1) Exclusive-Full mode: the microbenchmark runs with 64 processes on 8 nodes, on each node, all the 16 cores are used by this microbenchmark. 2) Exclusive-Half mode: we have the microbenchmark running with 64 processes on 16 nodes, on each node, only half of the cores are used and another half of the cores are idle; 3) Shared mode: we have two of the same microbenchmarks (each has 64 processes) running at the same time on 16 nodes, on each node, cores are equally divided for two jobs. The result shown in Table I is the average time length for an interation in each of the three modes.

From the communication time result, we observe that when no sleep (or computation) exists, communication times of Exclusive-Full mode and Shared mode are nearly identical, and both are twice of that of Exclusive-Half mode. This is because when the sleeping time is set to 0, the `MPI_Alltoall` function will be running constantly, leading to continous utilization of network. As a result, this combination of the two microbenchmarks falls into the first category of combinations as we present in Section IV: *dual communication-intensive*. Thus the shared mode can not provide any benefit since the utilization of network are in the same pattern as in exclusive mode. At the same time, the number of

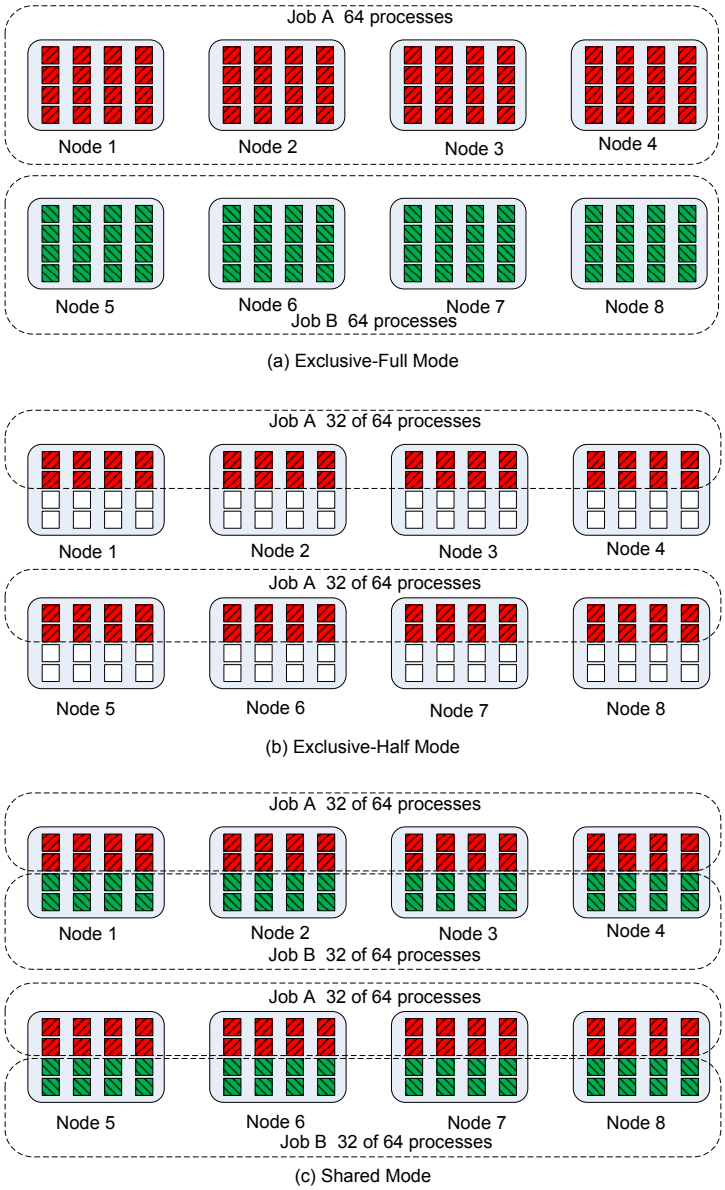
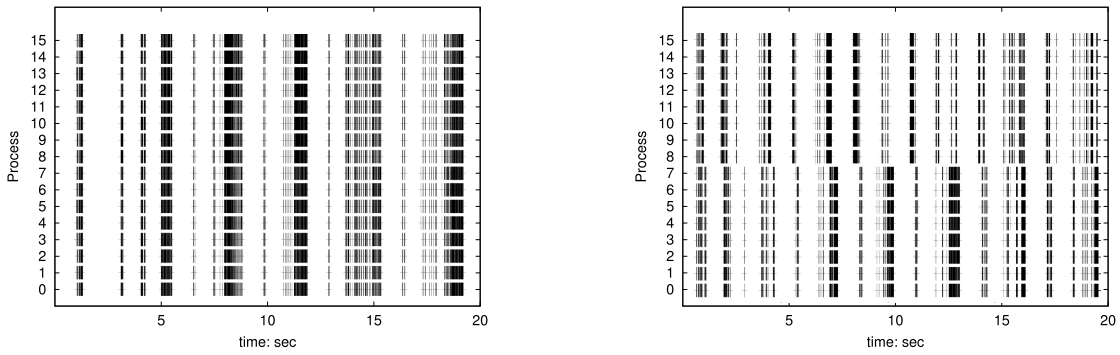


Figure 3. Exclusive-Full vs. Exclusive-Half vs. Shared modes

processes who are fighting for the network port at the same time in Exclusive-Half mode is only half of that in exclusive and shared mode, which results in the half communication time.

However, when the sleep time (or computation) is increased to 1 second, the average loop time for Shared mode is reduced to 0.18 seconds, which is a 33% percent improvement. This improvement can be explained by a contrast between Figure 4(a) and Figure 4(b), which are the profiling results for these microbenchmarks. The network utilization information was tracked and plotted for a single node. In this figure the x-axis is time and y-axis represents each process. From bottom up, each row shows how frequently a single process on this node utilizes the network port to send out data, from process 0 to process 15.

From Figure 4(a), we can observe that synchronization exists between all the 16 processes on the same node. But in shared mode (Figure 4(b)), the synchronization has been limited only between 8 processes on



(a) Exclusive-Full Mode

(b) Shared Mode

Figure 4. Node-Level Network Usage

the same node. The communication pattern of the two microbenchmarks now complement with each other. As the sleeping time increases, the combination of two microbenchmarks moves from *dual communication-intensive* to *communication compatible*.

D. Application Evaluation

In the microbenchmark evaluation section, the results show that when the application is very network-intensive, the shared mode doesn't provide any benefit. However, real applications seldom perform communication all the time. Most parallel applications fall into a repeated loop of communication and computation, which is decided by the parallel algorithm of the application.

To explore the real effects of this shared model on various workloads, we evaluate the NAS Parallel Benchmarks on a quad-core AMD multicore clusters. Since we are using NUMA AMD systems and processes within the same job are pinned to the same CPU sockets, the memory bandwidth between processes is isolated between jobs. Thus, the performance differences seen are primarily due to the network traffic.

The NAS Parallel Benchmarks (NPB) [15] are a small set of application kernels designed to help evaluate the performance of parallel supercomputers. The benchmarks, which are derived from computational fluid dynamics applications, consist of five kernels and three pseudo-applications. We run in all of the three modes described earlier.

1) Maximum Performance Improvement: Table II is a comparison of the results for Exclusive-Full and Exclusive-Half modes. By comparing these results we can determine the maximum possible performance that the Shared mode configurations are capable of acheiving since in the Exclusive-Half mode half of the cores on the node are idle. We see that for CG, FT and IS there are large performance improvements that are possible if we can reduce the contention with the runtimes reduced to 75%, 89.4% and 74.8% of the normal Exclusive-Full configuration.

This difference in performance is dependant on the communication pattern of these benchmarks. EP, which is a compute-bound program with little communication, has no link contention even in the Exclusive-Full mode and no improvement is possible even if half of the cores are idle. For other benchmarks that have more significant communication demands, such as FT, which performs an All-to-all, significant performance improvement is possible.

2) Mixed Workload Performance: In this set of experiments we evaluate each of the NAS benchmarks running with each of the other benchmarks. In this case one benchmark is run and then another benchmark from the suite is run in the background multiple times so no cores are ever idle. Table III shows the percentage of Shared mode execution time divided by Exclusive-Full mode execution time, in every possible combination. There is a significant improvement for many application combinations with very little degradation of performance in the worst case. This shows that using mixed workloads can improve overall performance significantly in many cases.

3) Understanding Mixed Workload Performance: While the performance of the mixed workload (Shared) configuration is very compelling, in this section we further look into how this is achieved. We use the profiling tool that we described in Section V to look at the detailed network usage.

Table II
BENCHMARK RUNTIME FOR EXCLUSIVE-FULL AND EXCLUSIVE-HALF

	Benchmark							
	BT	CG	EP	FT	IS	LU	MG	SP
Exclusive-Full	198.22	34.82	28.53	44.19	3.10	180.46	15.14	196.29
Exclusive-Half	196.71	26.17	29.07	38.89	2.29	179.86	14.78	188.84
Half / Full	99%	75.2%	99%	88.0%	73.9%	99.6%	97.6%	96.2%

Table III
PERCENTAGE OF SHARING MODE RUNTIME COMPARED WITH EXCLUSIVE-FULL MODE

		Measured Application							
		BT	CG	EP	FT	IS	LU	MG	SP
Background Application	BT	99.1%	**77.3%**	100.2%	**91.9%**	**81.6%**	99.7%	98.4%	**96.6%**
	CG	100.5%	101.8%	100.5%	**96.0%**	**90.2%**	100.9%	100.8%	102.0%
	EP	98.8%	**75.2%**	99.6%	**93.8%**	**80.1%**	100.1%	**97.9%**	**97.2%**
	FT	99.4%	**84.3%**	99.9%	**89.6%**	**87.6%**	100.5%	99.5%	98.9%
	IS	100.2%	**79.0%**	99.1%	**91.0%**	**84.4%**	99.6%	98.8%	**96.2%**
	LU	99.2%	**76.2%**	100.0%	**88.0%**	**80.7%**	100.4%	98.9%	**97.0%**
	MG	99.0%	**77.3%**	100.4%	**89.4%**	**73.9%**	99.6%	98.1%	100.5%
	SP	99.6%	**79.2%**	100.4%	**93.2%**	**86.7%**	100.3%	**97.6%**	99.5%

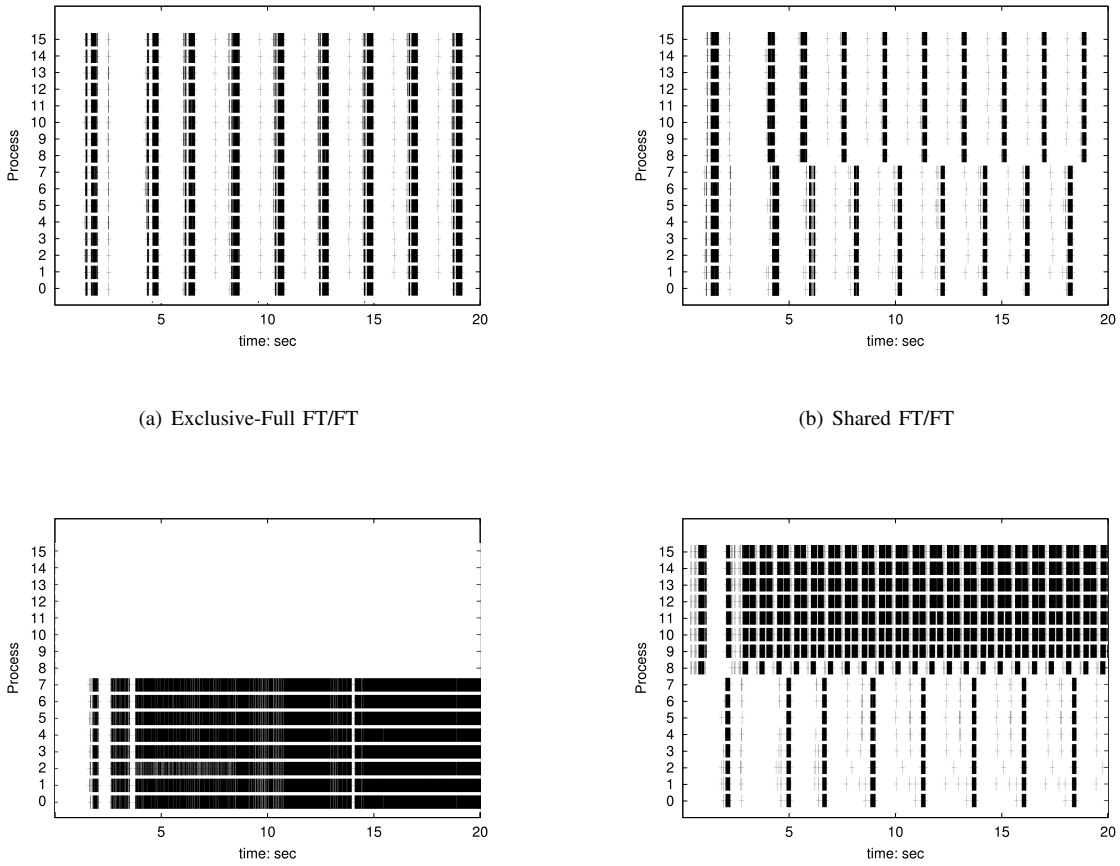
We first examine the performance of the FT benchmark, which has significant performance benefit in all combinations. Fig. 5(a) shows the 64-process FT benchmark in the Exclusive-Full mode running. As in the microbenchmark section, the network utilization information was tracked and plotted for a single node. In this figure the x-axis is time and y-axis represents each process. From bottom up, each row shows how frequently a single process on this node utilizes the network port to send out data, from process 0 to process 15. The figure clearly shows synchronization among processes belonging to the same job result network contention: all processes need the network port during the same short period of time, resulting in severe overlaps and delays in transmission of message. It is also important to note that the network is idle for a significant amount of time – it is effectively being wasted since there is a bottleneck for other parts of the application.

Fig. 5(b) shows the same FT benchmark, but in Shared mode. Here we have two different 64-process FT benchmarks split onto twice the number of nodes, and occupy 8 cores per node. The axes are the same as in the previous figure and denote the network usage by different processes. From bottom up, processes 0 to 7 show the network port usage of the first FT benchmark while processes 8 to 15 show the network port usage of the second FT benchmark. This figure is quite different from exclusive mode. Since the synchronization only exists between processes of the same job, processes are divided into two groups. After the first group of conflicts, the communication periods of the two groups of processes tends to stagger with each other. Then the delay caused by waiting for a free network port is reduced to half during the communication period. Though one of the job is slower than another, the final execution time of both of them are shorter than that in exclusive mode. From Table III we can observe that using shared mode we achieve a runtime of 89.6% of the Exclusive mode. Referring to the ideal speedup from Table II we see that the ideal is only 89.4%, so we have achieved over 99% of the possible improvement with this combination.

The big improvement is due to the basic communication pattern of the FT benchmark. The concentration of large message transmission in short period of time makes it perfect for running with another benchmark in the Shared mode with it.

Not all the benchmarks have this same pattern. EP is an extreme example, which seldom utilizes the network due to the rare communication between processes. So the mixed workload Shared mode can not bring any benefits to EP. However, the infrequent communication of EP means that it can be an ideal partner for another benchmark. Figure 5(c) shows the network port utilization when EP and CG are run in Shared mode (both EP and CG are 64-process jobs). The CG benchmark is at the extreme from EP: the CG kernel is communicating nearly all the time. As a result, when CG shares nodes with EP, it is as if CG is running on the nodes with half of the cores idle. This can be shown by the result in Table II and Table III: CG has the best performance when combined with EP, which results in a 26.6s execution time, nearly the same time as in Exclusive-Half mode.

To explore why some applications do not benefit from



(a) Exclusive-Full FT/FT

(b) Shared FT/FT

(c) Shared CG/EP

(d) Shared FT/LU

Figure 5. Node-Level Network Usage

the mixed-workload Shared mode, we study the communication pattern when combining FT and LU in Shared mode. From the network traffic diagram in Figure 5(d), LU should be able to improve due to the computation period in FT. However, the result in Table III shows that the performance of LU is the same as in Exclusive-Full regardless of which benchmark it is paired. This can be explained by examining the proportion of large messages, which will often overlap in the network port utilization, in LU and FT separately. After profiling the message size for these benchmarks, we find out that FT has large message proportion of 24% while LU only has 0.6%. As a result, LU can be recognized as one of the applications in the second category showed in Section IV, which has rare overlapping communication even in the exclusive mode due to the small message sizes. Thus, there is little room left for improvement in Shared mode to improve the performance of LU and other similar applications.

VII. Conclusions and Future Work

In this paper, we have demonstrated the potential of mixed workloads running on shared sets of nodes. We show that significant improvement can be gained when applications are paired such that the network contention is lessened. As part of our work we designed a profiling interface within the MPI library to track when messages are sent and the network is busy. With this data we can evaluate which applications are complementary in terms of network usage.

We first focused on microbenchmarks and saw how shared network usage of complementary applications could increase the available bandwidth for a process

when it needed it. We evaluated the NAS Parallel Benchmarks on a 128-core InfiniBand cluster and showed that through these mixed workloads the execution times can be reduced by up to 20%. We run with Exclusive-Full, Exclusive-Half and Shared modes. This allowed us to measure the effects of combining different applications to eliminate the contest for contention of links connected nodes between processes of same application. The compared result shows that the shared mode can achieve the same performance as half-exclusive mode in the ideal situation.

To further investigate the deeper reason of the different improvement percentage in different combinations, we used the results from our network profiling tool to look at the communication pattern of different NAS Parallel Benchmarks. The result proves that shared mode can help to break the synchronization between processes on the same node and then reduce the network contention, for certain combinations of benchmarks.

In the future, we plan to utilize the time-profiling information to implement job scheduler to automatically match the best suitable group of jobs. The job scheduler will first trace and record the time-profiling information of different applications. Then it will match all the existing applications in pairs to reduce the overlapping of usage of network to minimum. We also plan to explore more into the sharing mode such as unbalanced CPU distribution and multiple job striping.

ACKNOWLEDGMENTS

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, and #CCF-0833169; grant from Wright Center for Innovation #WCI04-010-OSU-0; grants from Mellanox, Intel, Cisco, QLogic and Sun Microsystems; Equipment donations from Intel, Mellanox, AMD, Advanced Clustering, Appro, QLogic, and Sun Microsystems.

REFERENCES

[1] J. Weinberg and A. Snavely, "User-Guided Symbiotic Space-Sharing of Real Workloads," in *Proceedings of the 20th annual international conference on Supercomputing (ICS '06)*, June 2006.

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

[3] S. Kim, D. Chandra, and Y. Solihin, "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture," in *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques (PACT '04)*, September 2004.

[4] Y. Jiang, X. Shen, C. Jie, and R. Tripathi, "Analysis and Approximation of Optimal Co-Scheduling on Chip Multiprocessors," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT '08)*, October 2008.

[5] J. Liedtke, M. Volp, and K. Elphinstone, "Preliminary Thoughts On Memory-Bus Scheduling," in *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system (EW 9)*, September 2000.

[6] M. Kondo, H. Sasaki, and H. Nakamura, "Improving Fairness, Throughput and Energy-Efficiency on a Chip Multiprocessor through DVFS," in *SIGARCH Computer Architecture News , Volume 35 Issue 1*, March 2007.

[7] J. Cong, K. Gururaj, and G. Han, "Synthesis of Reconfigurable High-Performance Multicore Systems," in *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays (FPGA '09)*, February 2009.

[8] T. Sondag, V. Krishnamurthy, and H. Rajan, "Predictive Thread-to-Core Assignment on a Heterogeneous Multicore Processor," in *Proceedings of the 4th workshop on Programming languages and operating systems (PLOS '07)*, October 2007.

[9] InfiniBand Trade Association, "InfiniBand Architecture Specification," http://www.infinibandta.com.

[10] J. Vetter and C. Chambreau, "mpiP: Lightweight, Scalable MPI Profiling," http://mpip.sourceforge.net/.

[11] Sun Microsystems, "Sun Studio Performance Analyzer," http://developers.sun.com/sunstudio/overview/topics /analyzerindex.html.

[12] W. E. . Nagel, M. W. A. Arnold, H.-C. Hoppe, and K. Solchenbach, "VAMPIR: Visualization and Analysis of MPI Resources," http://www.vampir.eu/.

[13] OpenFabrics Alliance, "OpenFabrics," http://www.openfabrics.org/.

[14] Network-Based Computing Laboratory, "MVAPICH: MPI for InfiniBand," http://nowlab.cse.ohio-state.edu/projects/mpi-iba.

[15] NASA Advanced Supercomputing, "The NAS Parallel Benchmarks," http://www.nas.nasa.gov/Resources/Software/npb.html.