Designing Power-Aware Collective Communication Algorithms for InfiniBand Clusters

Krishna Kandalla, Emilio P. Mancini, Sayantan Sur and Dhabaleswar K. Panda
Department of Computer Science and Engineering, The Ohio State University
{kandalla, mancini, surs, panda}@cse.ohio-state.edu

***Abstract*—Modern supercomputing systems have witnessed a phenomenal growth in the recent history owing to the advent of multi-core architectures and high speed networks. However, the operational and maintenance costs of these systems have also grown rapidly. Several concepts such as *Dynamic Voltage and Frequency Scaling* (DVFS), *CPU Throttling* have been proposed to conserve the power consumed by the compute nodes during idle periods. However, it is necessary to design software stacks in a power-aware manner to minimize the amount of power drawn by the system during the execution of applications. It is also critical to minimize the performance overheads associated with power-aware algorithms, as the benefits of saving power could be lost if the application runs for a longer time. Modern multi-core architectures such as the Intel "Nehalem" allow for DVFS and CPU throttling operations to be performed with little overheads. In this paper, we explore how these features can be leveraged to design algorithms to deliver fine-grained power savings during the communication phases of parallel applications. We also propose a theoretical model to analyze the power consumption characteristics of communication operations. We use micro-benchmarks and application benchmarks such as NAS and CPMD to measure the performance of our proposed algorithms and to demonstrate the potential for saving power with 32 and 64 processes. We observe about 8% improvement in the overall energy consumed by these applications.**

I. Introduction

The advent of multi-core architectures and high performance networks have fueled a sharp growth in the scale of supercomputing and these systems are allowing applications to scale out to tens of thousands of tasks. Supercomputing systems are typically comprised of hundreds of compute nodes based on multi-core architectures such as the Intel "Nehalem" [1] to satisfy the computational demands of the current generation High End Computing (HEC) applications. The number of compute cores within each node is constantly on the rise with the current generation systems offering as many as 4 cores per CPU socket and 8 to 16 cores per node. High performance networks such as InfiniBand [2] and 10GigE [3] have also evolved rapidly to satisfy the communication requirements of such systems. InfiniBand Quad Data Rate - QDR [4] offers a data rate of 40Gbit/s and is increasingly being used in large scale supercomputing systems.

The amount of energy consumed by these systems has rapidly grown in the recent history, mainly due to the growth in the scale of these systems leading to very high operational and maintenance costs. These costs are bound to increase further with the next generation exascale systems and this poses a serious challenge to system designers. The next generation supercomputers need to be designed in a power efficient manner to minimize their operational costs without sacrificing on the overall throughput and the performance delivered to the applications. It is also necessary to design software and middleware stacks in a power-aware manner to leverage the benefits offered by these systems.

Message Passing Interface (MPI) [5] is the de-facto programming model used for designing parallel applications. The MPI Standard allows application designers to use point-to-point and collective message exchange operations to address the communication requirements of parallel applications. Researchers have proposed energy aware optimizations to MPI in [6]–[10]. These approaches involve identifying communication phases of parallel applications that are not CPU intensive and using *Dynamic Voltage and Frequency Scaling* (DVFS) concepts during such phases to conserve power. Researchers have also proposed novel ideas such as link-shutdown [11], [12] to conserve the power drawn by the interconnection fabric when the network is not being stressed. Researchers have also explored the possibility of saving CPU power on clusters based on networks such that offer RDMA capabilities in [13], [14]. However, these studies consider communication operations to be a "black-box" and try to conserve power during these regions without analyzing the nature of the algorithms that are used to implement them. In this paper, we take on the following broad challenge: *Can we design collective message passing algorithms in a power-aware manner to offer fine-grained power savings with little performance overheads*. We have addressed the following major problems in this paper:

- Several optimized collective algorithms have been proposed and their performance and scalability characteristics have been deeply analyzed. But, are these algorithms power efficient?
- The performance overheads associated with power-aware algorithms are critical. The benefits of saving instantaneous dynamic power during the operation will be lost if the overheads are too high. Is it possible to leverage the benefits offered by modern multi-core architectures, such as the Intel "Nehalem", to minimize these overheads and deliver fine-grained power savings to applications?
- Theoretical models have been proposed to analyze the power consumption characteristics of generic work-loads, from the computation perspective. Can

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 and #OCI-0926691; grant from Wright Center for Innovation #WCI04-010-OSU-0; grants from Intel, Mellanox, Cisco, QLogic, and Sun Microsytems.

we derive models to analyze the power consumption characteristics of collective communication algorithms?

The rest of the paper is organized in the following manner. In Section II, we describe all the necessary background information that is relevant to this paper. In Section III, we discuss some of the ideas that have already been proposed by researchers in this area. In Section IV, we briefly describe the nature of the state-of-the-art algorithms for collective operations on modern multi-core architectures and demonstrate the potential for conserving power during these collective operations. In Section V, we propose our power-aware algorithms for collective operations. In Section VI, we propose a set of models to analyze the power consumption characteristics of collective algorithms. In Section VII, we describe our experimental methodology. We also analyze the performance and power consumption characteristics of our proposed algorithms and compare them with some of the well known algorithms. In Section VIII, we conclude our discussion and also provide insights into our future work.

II. Background

In this section, we provide the necessary background information that is relevant to our research.

A. Intel "Nehalem"

Compute nodes based on multi-core architectures featuring as many as 8-16 compute cores have become ubiquitous in the field of high performance computing. Current generation architectures have a range of operational frequencies (P-States) and throttling levels (T-States). The operating system can choose specific CPU frequencies and throttling states to conserve power and to lower the temperature when the CPU is not loaded. The Intel "Nehalem" architecture offers 4 cores per CPU socket and allows frequency scaling and CPU throttling operations to be performed within 10-15 usecs, which is significantly better when compared to some of the earlier Intel multi-core architectures. The "Nehalem" architecture offers eight throttling levels T0 - T7, with the CPU being 100% active in the T0 state and only 12% active in the T7 state.

B. Message Passing Interface

The Message Passing Interface (MPI) [5] is one of the popular programming models used for designing parallel applications. MPI defines primitives for point-to-point and collective message exchange operations that can be conveniently used by application designers to address the communication requirements of parallel applications. In our work, we use the MVAPICH2 [15] software stack to design power-aware collective algorithms and to demonstrate the performance and power characteristics of our proposed designs. MVAPICH2 is being used by more than 1,000 organizations world-wide, including several large scale clusters.

C. InfiniBand

InfiniBand has emerged as the popular I/O interconnect standard and almost 36% of the Top500 Supercomputing systems [16] rely on InfiniBand to address the communication requirements of the current generation applications. InfiniBand networks allow several network protocols to be offloaded to the Host Channel Adapters (HCA), that reside on each compute node. Owing to this reason, the CPU does not have to be completely involved in the communication operations. MPI implementations that are designed for InfiniBand networks offer two modes of message progression - "polling" and "blocking" modes. In the "blocking" mode, an MPI process waits for a new incoming message for a short period before yielding the CPU. When a new message arrives, the InfiniBand HCA generates an interrupt and the process can resume its execution once it is scheduled. In the "polling" mode, the MPI process constantly spins while it waits for a new arriving message. MPI implementations that use the "polling" mode deliver better performance, but require the CPU to be busy during communication. The "blocking" mode delivers better CPU availability to applications with lower performance for network communication operations due to the overheads associated with the interrupts and OS scheduling policies. Also, MPI implementations cannot offer a high performance intra-node communication mechanism with the "blocking" mode, as they fall-back to the network loop-back based communication instead of using the shared-memory channels. The performance and power trade-offs between these modes were demonstrated in [17]. Like all high-performance MPI stacks, MVAPICH2 uses the "polling" mode as the default mode.

D. Collective Message Exchange Algorithms in MVAPICH2

In MVAPICH2, several multi-core aware algorithms have been proposed for collective operations defined in the MPI Standard. As shown in Figure 1, these algorithms rely on detecting the node-level topology of the systems to create sub-communicators. Processes that are within the same compute node are grouped within a shared-memory communicator. One process per node is assigned as the node-leader process and another communicator is created to include all the node-leader processes. In [18]–[21], the authors have demonstrated the performance and scalability of such approaches. These algorithms involve the following stages :

- Intra-node phase: All the processes within the same compute node write their buffers into the explicitly created shared-memory region. This operation involves no data movement across the network links.
- Network phase: This phase of communication only involves the node-leader processes and the data is moved across the network.
- Intra-node phase: Depending on the nature of the collective operation, there might be an optional phase in which the non-leader processes read the data that was written into the shared-memory regions by the leader process after the second step.

III. Related work

Several researchers have explored the possibility of saving power in the high performance computing domain. These studies have involved minimizing power

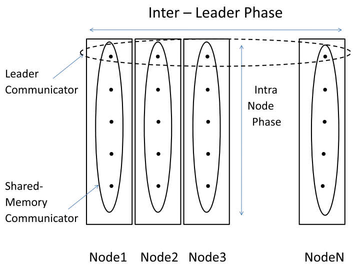

Fig. 1. Multi-Core Aware Collective Algorithms in MVAPICH2

consumed by both the CPU and the network. In [11], [12], [22], the authors proposed efficient designs to conserve the power drawn by the network fabric by shutting down the network links when the network was not being stressed. Using DVFS to save CPU power during communication phases have been proposed by researchers in [6]–[8], [10]. Some of these ideas involved automatically detecting the communication phases and scaling the voltage and frequency of the CPU around these regions to conserve power. In [9], the authors proposed theoretical models to analyze the power consumption characteristics of applications on multi-core architectures. However, all communication phases were considered as an overhead in the models that were proposed in that paper. In [14], [17], the authors have studied the potential for saving CPU power by using DVFS and CPU throttling operations in clusters that use RDMA based networks. In [10], the authors have also proposed a tool - PowerPack to efficiently profile the energy consumption characteristics of applications on clusters. In [23], researchers have presented a fine-grained component-level description of the amount of power consumed by scientific appliciations. These approaches treat communication operations as a "black-box" and try to conserve CPU power during these regions. However, in this paper, we propose a set of models to analyze the power consumption of the state-of-the-art algorithms for collective message exchange operations and we also propose power-aware algorithms that leverage both DVFS and CPU Throttling to deliver fine-grained power savings to applications.

IV. CPU Usage in Current Collective Algorithms

In this section, we study the characteristics of the state-of-the-art multi-core aware algorithms for a few important collective operations.

A. Alltoall Personalized Exchange Operation

The Alltoall Personalized Exchange algorithm is the most dense collective communication operation defined in the MPI Standard. In this operation, every process in the process group exchanges distinct messages with every other process. The performance and scalability issues with MPI_Alltoall were discussed in [19], [24]. In MVAPICH2, the hypercube algorithm [25] for small messages and the pair-wise exchange algorithms for large messages is used. In Figure 2(a), we show the scalability of the Alltoall operation for large messages. We have run the OSU Alltoall benchmark [26] with 32 processes in 4-way and 8-way configurations (4 processes per node, across 8 nodes and 8 processes per node, across 4 nodes, respectively). We have also shown the theoretical estimate of the latency involved for the MPI_Alltoall operation with 32 processes for various message sizes. We can see that even though the size of the Alltoall job is the same, the change in the process allocation pattern has lead to a performance difference of almost 54% with large messages. This could only be attributed to contention at various levels of the system - switches, network links, the HCAs and the internal system bus. We expect the effects of contention to become even more severe as the number of cores per node increase further. We would also like to emphasize that these experiments were performed with InfiniBand QDR network, which currently offers the highest bandwidth when compared to other InfiniBand networks. Since we use the "polling" mode, as the time required for an MPI_Alltoall operation increases, the amount of power consumed by each compute core during the operation also increases as the CPU spends a considerable amount of time in the busy-wait state, as the communication progresses. Hence, it is necessary to design efficient algorithms for the MPI_Alltoall operation to address these concerns.

B. Broadcast and Reduction Operations

In MVAPICH2, multi-core aware shared-memory based algorithms are being used for several collective operations. The nature of these algorithms were described in Section II-D. The entire collective operation can be viewed as being comprised of two phases - network communication phase and an intra-node communication phase. The network communication phase will involve regular eager and rendezvous point-to-point MPI operations. As indicated in [27], as the scale of the systems increases, the network phase will also involve data being moved across multiple switches. These network flows will also contend for network bandwidth with other applications that are using other compute nodes in the system. The network time will also be affected by process skews and network noise [28], [29]. For the intra-node communication phase, processes use the explicitly created shared-memory buffers to read/write data. Since this phase of communication happens completely within a compute node, it does not experience any network contention. Depending on the memory organization of the system, a degree of memory contention could be experienced. However, the impact of network contention is expected to be more significant when compared to the effects of intra-node memory contention. Overall, we expect the costs associated with the network communication phase to be higher than that of the intra-node communication phase. We have instrumented the collective algorithms to profile the amount of time spent in the network phase of the collective operation and the amount of time spent in the intra-node phase of the communication. In Figures 2(b) and (c), we present the comparison between the overall time taken for a collective operation and just the network phase

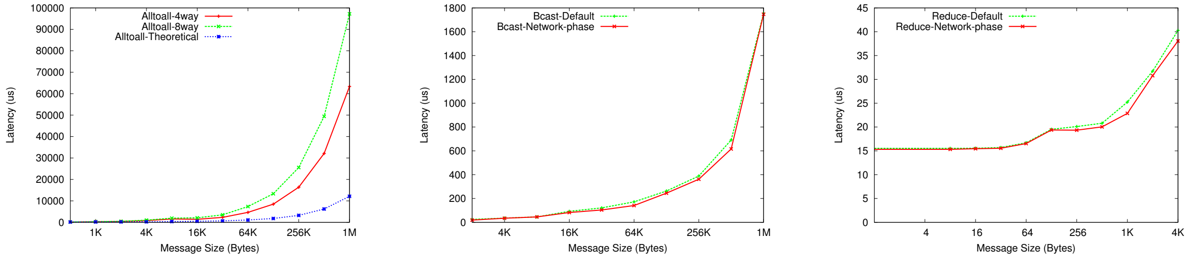

Fig. 2. (a) Alltoall scalability with 32 processes across 4-way and 8-way configurations, (b) Bcast Overall Time Vs Network Time (c) Reduce Overall Time Vs Network Time

of the collective operation for various message sizes for MPI_Bcast and MPI_Reduce with 64 processes. These figures indicate that the network communication phase accounts for most of the overall time required for these collective operations. Since only one process per compute-node is involved in the network communication phase and we are using the "polling" mode, this also leads us to the observation that rest of the processes are continuously using the CPU, as they wait for the network communication phase to complete and are not doing any useful work. This implies that there is a strong potential to conserve the power consumed by the non-leader processes in every node with the shared-memory based multi-core aware collective operations. *Broadly, we are interested in leveraging the low latency power saving operations offered by the Intel "Nehalem" architecture to design power-aware collective algorithms.*

V. Designing Power-Aware Collective Algorithms

In this section, we describe our research methodology and also present our power-aware algorithms for collective operations. Since modern architectures such as the Intel "Nehalem" allow us to perform DVFS operations and CPU throttling operations with very small overheads, we have chosen to perform the DVFS operations on a *per-call* basis. At the start of each collective operation, we scale the frequency of all the compute-cores to the minimum possible frequency and at the end of the operation, we again scale the frequency up to the peak frequency supported by the architecture. We would like to note that the methods proposed by authors in [6]–[9] are definitely applicable on these modern architectures, as well. However, we have also re-designed some of the important collective algorithms to deliver fine-grained power savings to applications with small performance overheads. We have also focused mainly on the power consumed by the CPU in this paper. Some of the existing literature in this field [23] indicate that the CPU accounts for more than 50% of the overall power consumption of a compute node.

A. Power-Aware MPI_Alltoall algorithm

In Section IV-A, we discussed about the performance impact of network contention on the performance of an MPI_Alltoall operation. In this section, we propose our novel algorithm to schedule the inter-node exchanges in an efficient manner and we also leverage the concept of CPU throttling to conserve power with very little performance overheads. Consider an MPI_Alltoall operation being performed using the pair-wise exchange algorithm, across *P=N*c* processes, such that we have *N* compute nodes, each node has *c* cores and *N*c* is a power of 2. In the pair-wise exchange algorithm, each process does *P* iterations to send and receive distinct messages from every other process in the system. The first c steps of this operation will involve intra-node message exchanges and the remaining *(P-c)* iterations will involve data being sent across the network. Since the time spent in the last *P-c* steps is almost comparable to the time consumed by the entire operation, we focus on conserving CPU power during this part of the operation. Within each node, we group processes that are on the same CPU socket as shown in Figure 3. We now have two process groups within each compute node - A and B. We scale down the frequency of each core to its minimum frequency, f_{min} at the start of the collective and schedule the entire communication operation in the following manner:

- *Phase 1*: All the processes perform c iterations to exchange data only with other processes that are on the same compute node.
- *Phase 2*: Throttle down all processes belonging to process group B to the lowest possible state allowed by the architecture. Allow only the processes in the process group A to participate in inter-node communication.
- *Phase 3*: Once all the processes in process group A are done with their message exchange operations, throttle all these processes down, throttle up the processes in process group B, and allow only these processes to participate in the inter-node communication.
- *Phase 4*: The final phase of the algorithm involves *N* iterations. In each iteration, we pair the compute nodes i and j such that $i < j$. Let A_i and B_i be the two process groups on node *i*. Similarly, A_j and B_j be the two process groups on node *j*. We first throttle down process groups B_i, A_j and allow process groups A_i, B_j to communicate. Then, we throttle down process groups A_i, B_j, throttle up process groups B_i and A_j and allow only B_i, A_j to communicate.

In Figure 3, we represent the socket that has been throttled down by shading it completely and we also indicate the throttling level as $T7$. Once the MPI_Alltoall operation is complete, we perform another DVFS operation to restore each core to its peak frequency, f_{max}. We perform the CPU throttling operations in the manner described above to selectively throttle down only the processes in the idle socket, while the processes in the active socket proceed with their inter-node communication steps. The communication in phases 2 and 3, can be viewed as an Alltoall pair-wise exchange operation with a system size of *(N*c)/2* processes. Since in each of these steps, only half the processes within each compute node are involved in inter-node communication, we can expect the overhead associated with the network contention to be less than the regular algorithm in this case. This algorithm provides us with an opportunity to schedule messages in a way that allows half the cores to be idle for half of the time required for the MPI_Alltoall operation which can be utilized to conserve power. Also, we would like to emphasize that a core that has been throttled down is not involved in any communication operation.

B. Power-Aware Algorithms for Shared-Memory Based Collectives

In Section IV-B, we observed that for shared-memory based collectives, the amount of time spent in the network communication phase is much higher when compared to the intra-node communication phase and the non-leader process keep the CPU busy as they wait for the network phase of the operation to complete. We also discussed that on large scale systems with processes being distributed across different parts of the systems, the non-leader processes will spend a considerable amount of time in the busy-wait state during collective operation. This gives us the opportunity to save CPU power by throttling down the cores of the non-leader processes with our shared-memory based algorithms. In our proposed power-aware algorithms, we throttle down the CPUs of the non-leader processes during the network phase of the collective operation. The current generation "Nehalem" architecture allows for the CPU throttling operation to be performed at the socket-level granularity. Suppose the node-leader process is on socket A, if we throttle this socket, this will invariably slow down the network phase of the communication and this will affect the performance of the collective operation. Hence, we throttle down the cores on this socket partially to allow for some power savings without sacrificing on the performance. However, the cores on socket B are not involved in any communication and can be throttled down to the lowest possible state offered by the architecture, as indicated in Figure 4. On architectures that allow for CPU throttling to be performed at the core-level granularity, we could easily throttle down the compute cores of all the non-leader processes to the lowest possible state, thereby leading to higher power savings and can also minimize the performance overheads as the leader-process can remain at the $T0$ state. Note that if were to use the default binomial exchange algorithm [30] in which every process is involved in the entire communication operation, we cannot directly use CPU throttling to conserve power, without observing significant overheads.

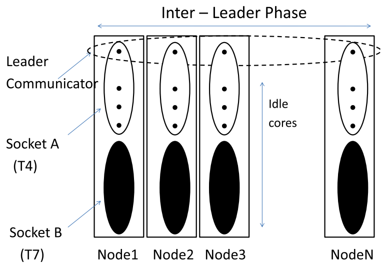

Fig. 4. Power-Aware Shared-Memory based Collective Algorithms

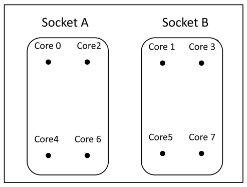

Fig. 5. Core-Socket Mapping on the Intel "Nehalem" Architecture

C. Impact of Process Affinity on Power Management

As shown in Figure 5, the Intel "Nehalem" architecture has cores 0,2,4,6 on socket A and cores 1,3,5,7 on socket B. MVAPICH2 binds processes to sockets such that processes 0,1,2,3 will be bound to the cores on socket A and processes 4,5,6,7 will be bound to the cores on socket B. This mapping is established at job launch time and the processes remain bound in this manner for the entire duration of the application's life-time. The power-aware algorithms that we have discussed above, rely on this mapping so that we can perform DVFS and CPU throttling operations in the manner that we have described If we were to choose a different process to core mapping pattern, it is possible to perform DVFS or the CPU throttling operations incorrectly and this will invariably affect the performance of our algorithms.

VI. Modeling Performance and Power for Collective Operations

In this section, we derive a set of theoretical models to analyze the performance and power consumption patterns of the collective algorithms that we discussed in Section V. In [30], the authors proposed a set of theoretical models to analyze some of the classical collective algorithms on traditional systems based on Ethernet networks and single-core compute nodes. We extend these models for analyzing power and performance on multi-core clusters. As discussed in Section IV, the time spent in pure intra-node communication operations is very small for both MPI_Alltoall and shared-memory based collectives and for brevity, we are not going to include these times in our models. We use the terms O_{dvfs} and $O_{throttle}$ to represent the overhead involved in performing the DVFS and CPU Throttling operations,

A. Modeling Performance

Let $t_{s-intra-node}$ be the start-up cost associated with an intra-node message exchange operation and $t_{w-intra-node}$ be the cost involved in sending a word of data to a peer process within the same node. Similarly, let $t_{s-inter-node}$ and $t_{w-inter-node}$ be the costs associated with an inter-node message exchange operation. We consider a system with *N* nodes, with each node having *c* cores and the size of the message to be *M* bytes. As discussed in Section IV-A, we need to account for the network contention while modeling the performance of collective operations. We introduce the parameter C_{net}, such that, $C_{net} > 1$, to account for the network contention effects.

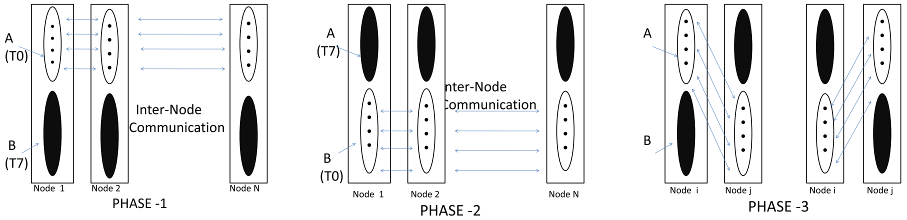

Fig. 3. Power-Aware Alltoall Personalized Exchange Algorithm

1) Default Algorithms: The cost of performing an MPI_Alltoall or MPI_Alltoalv operation with a message size of M across *N*c* processes, with the pair-wise exchange algorithm can be expressed as :

$$T_{Alltoall} = t_{w-inter-node}(P-c)(C_{net})M \quad (1)$$

As we discussed in Section II-D, for a collective like MPI_Bcast, the entire operation is performed across the leader and the shared-memory communicators. For the inter-leader operation with medium and large messages, the Scatter-Allgather algorithm is used to broadcast the data across all the leaders. If the size of the message to be broadcast is *M* bytes, first the data are scattered across all the leader processes in the leader communicator and then an alltoall-broadcast operation is done across all the leaders to achieve the desired result. Once all the node-level leader processes have the entire buffer, they broadcast the data to all the processes that are within the same compute node using the shared-memory communicator. Hence, the performance of a medium or large message MPI_Bcast operation across *N*c* processes can be modeled as:

$$t_{scatter} = M\left((N-1)/N\right)t_{w-inter-node}$$
$$t_{allgather} = M(N-1)t_{w-inter-node}$$
$$T_{Bcast} = M(N-1)t_{w-inter-node}(1+(1/N)) \quad (2)$$

2) Proposed Power-Aware Alltoall Algorithm: In Section V-A, we proposed our power aware algorithm for the MPI_Alltoall operation. The algorithm is designed to allow only half of the processes to be active for half of the time during the inter-node exchange steps, and we completely throttle down the idle cores. Since the volume of data exchanged by any node is now reduced by a factor of 2, we expect the network contention to also improve by 50%, along the lines of the discussion in IV-A. In phase-2, the cost of throttling down all the processes on socket-B can be hidden as the processes on socket-A would have started their communication operation. In phase-3, the processes on socket-B can be throttled up only after phase-2 is complete and each process incurs the cost $O_{throttle}$ once. In each iteration of phase-4, the cost associated with one of the CPU throttling operations can be hiddlen, but each process will incur the cost $O_{throttle}$ once and there are *N-1* iterations. Hence, we can model the performance of our proposed algorithm in the following manner:

$$\begin{aligned} t_{Phase-2} &= t_{w-inter-node}Nc(C_{net}/4)M \\ t_{Phase-3} &= t_{w-inter-node}Nc(C_{net}/4)M \\ t_{Phase-4} &= t_{w-inter-node}Nc(C_{net}/4)M \\ t_{Alltoall-power} &= (3/4)t_{w-inter-node}NcC_{net}M + \\ &\quad +2*O_{dvfs}+N*O_{throttle} \quad (3) \end{aligned}$$

In equation (3), we can see that the performance overhead associated with our proposed algorithm is linearly proportional to the number of nodes in the system. So, we do not expect to see much difference in the raw performance between our proposed algorithm and the default pair-wise exchange algorithm, even though we could theoretically see a maximum improvement of about 25% in the raw performance of the algorithm (without any power optimizations). However, we do expect to see benefits on slower networks such as the InfiniBand DDR or SDR [4], where the impact of contention can be more significant. We can also infer that if the costs associated with the DVFS and CPU throttling operations are further minimized in the next generation multi-core architectures, it is possible to minimize the performance overheads associated with our proposed algorithm.

3) Proposed Power-Aware Shared-Memory based Algorithms : As discussed in Section V-B, in our proposed power-aware shared-memory based collective algorithms, we throttle the CPUs of the non-leader processes when the leader processes are involved in the inter-leader communication operations. However, since the throttling operation is being performed at the socket-level granularity, and we are throttling down the socket on which the leader process is mapped to, we expect to see some performance degradation. If future architectures allow for the CPU throttling to be performed at core-level granularity, we could minimize the performance impact on the inter-leader operation by throttling only the non-leader processes. So, we introduce the parameter $C_{throttle} > 1$ to account for the performance overhead introduced by throttling the leader processes. Also, all the processes are throttled down at the start of the inter-leader operation and throttled up at the end of it. So, we can model the performance of our proposed MPI_Bcast algorithm in the following manner:

$$\begin{aligned} t_{Bcast-power} &= (M(N-1)t_{w-inter-node} \cdot \\ &\quad \cdot(1+1/N))C_{throttle}+2*O_{dvfs} \cdot \\ &\quad \cdot +2*O_{throttle} \quad (4) \end{aligned}$$

If the next generation multi-core architectures allow us to perform CPU throttling operations at the core-level granularity, then the inter-leader operations can proceed without any performance degradation and the overheads introduced by our algorithm will be a constant factor. Hence, on such systems, as the job size scales, our algorithms can deliver higher power savings with almost negligible performance overheads.

B. Modeling Power

Suppose there are no power optimizations being used, each core in the system will operate at its peak frequency, f_{max}, when the system is loaded.

If these collective operations were performed without any power optimizations, and each core will operate at its peak frequency, f_{max}. Let $p_{core,i(t)}$ be the instantaneous power drawn by the core *i* at time t. Suppose a collective operation occurred in the interval [$t1$, $t2$], such that the difference $(t2 - t1)$ is equivalent to the expressions derived in equations (1) and (2). Since we have a total of *N*c* cores in the system, the total power drawn by the system can be expressed as :

$$\int_{t1}^{t2}\sum_{i=1}^{N*c} p_{core,i,f_{max}}(t)\,dt \tag{5}$$

Suppose we perform only the DVFS operations for all the compute cores before and after each collective communication operations, each of the CPU cores will be running at their minimum frequency, f_{min} during the communication operations and at their peak frequency, f_{max}, during the computation phases of the application. Since the communication operations are now being performed at a lower frequency, the time required to complete these operations can be higher. Suppose, the collective operation in the time interval [$t1$,$t2'$], such that $(t2' > t2)$, to account for the possibility of incurring a performance overhead when operating at a lower frequency. The overall power consumed by the system during these collective operations with the DVFS operation can be expressed as:

$$\int_{t1}^{t2'}(\sum_{i=1}^{N*c} p_{core,i,f_{min}}(t))\,dt \tag{6}$$

If the i^{th} core is throttled to the T_j state, at a time instant t, we consider the power drawn by this core to be $c_j * p_{core,i(t)}$, where c_j is in the interval [0,1]. Since we have 8 different throttling levels [T1, T7], with T7 being the state where the CPU is only 12% active, we can say that $c_1 > c_7$.

1) Power-Aware MPI_Alltoall algorithm: In our algorithm, in phases 2 through 4, a given core will spend half of the time in the completely throttled state, and the remaining time in the T_0 throttled state, and the frequency of each core during the operation is f_{min}. If the time elapsed for the MPI_Alltoall operation is [$t1$,$t3'$] we can model the power consumption of our proposed algorithm in the following manner:

$$\int_{t1}^{t3'/2}(\sum_{i=1}^{(N*c)} p_{core,i,f_{min}}(t))dt + \\ + \int_{t3'/2}^{t3'}(\sum_{i=1}^{(N*c)} c_7 p_{core,i,f_{min}}(t))\,dt \tag{7}$$

On comparing equations (6) and (7), we can see that our proposed algorithm can deliver greater power savings, as the amount of power consumed by each core has been reduced by a factor of $c7$ for half of the time interval. Despite the fact that our algorithm has a slightly higher performance overhead, we are able to demonstrate power-savings with applications in Section VII.

2) Power-Aware Algorithms for Shared-Memory Based Collectives: In Section V-B, we proposed our power-aware algorithms for shared-memory based collectives. During the inter-node operation, we are throttling down Socket B to the T7 state, and socket A to T4 state. Suppose our power-aware MPI_Bcast operation was executed in the time interval [t1,t3'], we can model the power consumption of our proposed power-aware algorithm in the following manner:

$$\int_{t1}^{t3'}(\sum_{i=1}^{(N*c)/2} c_4 p_{core,i,f_{min}}(t) + \\ + \sum_{i=1}^{(N*c)/2} c_7 p_{core,i,f_{min}}(t))\,dt \tag{8}$$

From equations (6) and (8), we can see that with our proposed power-aware shared-memory based collective algorithms can deliver greater power savings as the amount of power consumed by the processors on socket B, is now lower by a factor of $c7$. The amount of processors on socket A is also lower by a factor of $c4$, but there is a performance overhead associated with our algorithm. We already discussed the potential advantages of architectures that allow core-level CPU throttling. On such systems, the amount of power consumed by all the non-leader processors can be lowered by a factor of $c7$, leading to greater power savings without additional performance overheads.

VII. Experimental Evaluation

In this section, we describe the experimental testbed used for our experimental methodology.

A. Experimental Testbed

We have used a cluster comprising of 8 compute nodes based on the Intel "Nehalem" architecture. Each node has 2 CPU sockets, with each socket having 4 compute cores that can operate in the frequency range 1.6GHz to 2.4GHz. The 8 compute nodes are connected together through InfiniBand QDR links and a Mellanox QDR Switch. We use a MASTECH MS2205 Digital Clamp Power Meter with RS232 Interface to measure the power drawn by a single compute node and log the data on a remote machine. We have repeated the experiments by changing the application's hostfile to measure the power consumed by the other compute nodes. The power meter generates instantaneous power consumption readings with intervals of 0.5s. We have used this power meter to measure the amount of power consumed by a single node with micro-benchmarks. However, we need a high resolution power meter to capture the power consumption data during every communication phase in applications. For this paper, we have profiled the applications to learn about how much time processes spend in various collective operations and we use the power consumption data gathered from the benchmark results to estimate the potential power benefits in applications.

B. Benchmarks and Applications

We have used OSU MPI Benchmarks, a freely available benchmark suite that is distributed with the MVAPICH/MVAPICH2 [31] software stacks to measure the performance and power consumption characteristics of communication operations. We have also used NAS Parallel Benchmarks [32] and the CPMD application [33], to analyze the performance and potential for power savings of our proposed power-aware collective algorithms. In the NAS suite, we have used the Class C FT and IS kernels in our experiments.

C. Polling Vs Blocking Power and Performance Characteristics

In Figure 6, we demonstrate the performance and power characteristics of the "blocking" and the "polling" message progression modes with MPI_Alltoall with 64 processes for medium and large message sizes. In Section II-C, we indicated that the latency of intra-node communication was very poor with the "blocking" mode. However, with the pair-wise exchange algorithm, we spend little time performing pure intra-node exchanges. So, the performance difference between the "blocking" and "polling" modes in Figure 6(a), can be considered to be purely due to the interrupt and scheduling overheads associated with the "blocking" mode. In Figure 6(b), we compare the amount of power consumed by one compute node during the benchmark execution with the "polling" and "blocking" modes. When we use the "polling" mode, each node consumes almost 2.3 KW of power at each sampling instant. However, with the "blocking" mode, each node consumes about 2 KW of power at each sampling point. This can be attributed to the fact that, each process yields the CPU after "polling" for a short time. The InfiniBand HCA generates an interrupt when a new message arrives and the OS schedules the task onto the CPU upon servicing the interrupt. We can see that the "blocking" mode has the potential for conserving power, but since the overall performance is poor, it is not a desirable option. Also, in Figure 6(b), we can see that the curve corresponding to "blocking" involves larger number of sampling points indicating that the benchmark has taken a longer time to complete, when compared to the "polling" mode.

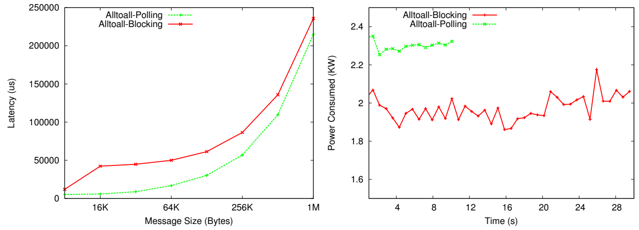

Fig. 6. Blocking Vs Polling with MPI_Alltoall 64 Processes: (a) Performance Performance and (b) Power

D. MPI_Alltoall and MPI_Alltoallv

In Figure 7(a), we compare the performance of the power-aware alltoall algorithm that we discussed in Section V-A with the original algorithm that does not involve any power-optimizations and the version in which we perform only the frequency scaling operation on a *per-call* basis. We can observe that the performance difference between the default version and the power-aware algorithms is only about 10% and that there is very little difference between our proposed algorithm and the algorithm that performs only the DVFS operations. This indicates that the overhead of the throttling operation is actually quite low and the factor $(N * O_{throttle})$ in equation (3) affects the performance by only a small amount. We would also like to stress that our methods can work well in conjunction with the methods proposed in [6] to minimize the overheads associated with the DVFS operations, leading to higher power savings. In Figure 7(b), we compare the power consumption characteristics of the three algorithms. In the default case, each node consumes about 2.3 KW of power at each sample point. With the algorithm that performs only the DVFS operations, we can see that the power consumption drops down to about 1.8 KW at each sampling point. However, we can see that with our proposed algorithm, we can minimize the amount of power consumed to about 1.6 KW at each sampling point. We can see similar results even in Figures 8(a) and (b) for the MPI_Alltoallv operation.

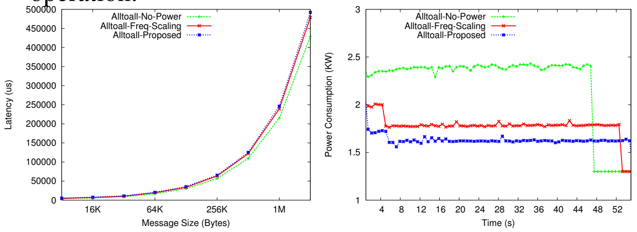

Fig. 7. Alltoall with 64 processes:(a) Performance and (b) Power

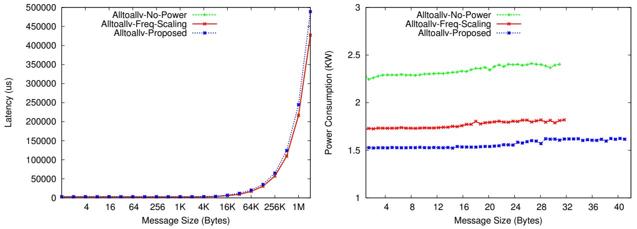

Fig. 8. Alltoallv with 64 processes:(a) Performance and (b) Power

E. MPI_Bcast

In Section V-B, we discussed our power-aware optimizations to the current set of shared-memory based collective algorithms in MVAPICH2. In Figures 9(a) and (b), we evaluate these designs from both the power and performance perspectives for the MPI_Bcast operation. In Figure 9(a), we compare the performance of the three designs - default case, the algorithm that performs only the DVFS operations and our proposed algorithm that throttles down the idle CPU's during the inter-leader phase of the broadcast operation. With either of the power-aware algorithms, we can see that there is an overhead of about 15% when the message size is 1MB. However, there is very little difference between the performance of the algorithm that performs only the DVFS operation and our proposed algorithm. In Figure 9(b), we compare the power consumption patterns of the three algorithms. In the default case, each compute node consumes about 2.3 KW of power at each sampling point. The algorithm that only performs the frequency scaling operation consumes about 1.8 KW of power at each point. But, with our proposed algorithm, we can minimize the power consumption to about 1.6 KW at each point. As discussed in Section V-B, if future architectures allow the throttling operation to be performed at the core-level granularity, we would be able to conserve more power by throttling down all the non-leader processes to the lowest state.

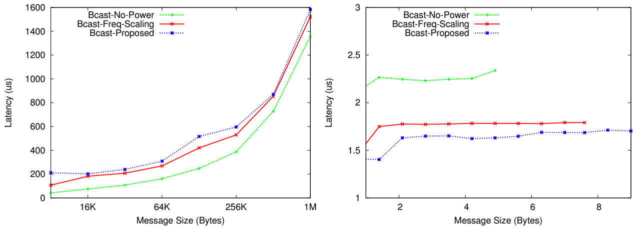

Fig. 9. Bcast with 64 processes. Performance and Power

F. CPMD Application

The CPMD code is a parallelized plane wave/pseudopotential implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamics [33]. We have used the datasets wat-32-inp-1, wat32-inp-2 and ta-inp-md available at [34] for our experimental evaluation. In Figure 10, we compare the performance of the default and the power-aware schemes with the CPMD application run across 32 and 64 processes in the strong-scaling mode. The profiling information available in the CPMD output logs indicate that the time spent in the MPI_Alltoall operation dominates the overall communication time. For brevity, we compare only the overall execution time and the amount of time spent in the MPI_Alltoall operation in these figures. We can see that with increasing the system size from 32 to 64 processes, the overall application run-time reduced by almost 50%, as we are using the same problem size. However, the amount of time spent in the MPI_Alltoall operations has changed by a small amount. This is because the cost associated with the pair-wise exchange algorithm is linearly proportional to both the system size and the message size. We can also see that with either of the power optimizations, there is a performance degradation of about 2 - 5% and there is very little difference between the version that performs only the DVFS operations and our proposed versions, indicating that the overhead of the throttling operations is very negligible. In Table I, we compare the total amount of power consumed with the three different data-sets and across 32 and 64 processes. We observe about 8% energy savings with ta-inp-md dataset with a system size of 64 processes.

G. NAS FT and IS Application Kernels

In Figure 11 and Table II, we compare the performance and energy consumption statistics of class C FT and IS NAS kernels. The performance and power characteristics are similar to what was observed with the CPMD benchmarks and we observe about 8% energy savings with the IS kernel.

VIII. Conclusion

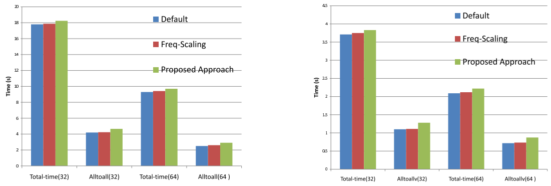

Fig. 11. NAS : Overall Execution time and Alltoall time with 32 and 64 Processes (a)FT Kernel (b) IS Kernel

TABLE II
NAS Application : Power Statistics

	32 Processes (KJ)		64 Processes(KJ)	
	FT	IS	FT	IS
Default (No-Power)	16.36	3.412	17.056	3.8456
Freq-Scaling	15.588	3.248	16.32	3.608
Proposed	15.472	3.16	16.16	3.52

Most applications spend a significant amount of their run-times performing collective operations. As the size of the systems continues to scale, various factors affect the performance of collective operations and several researchers have proposed various algorithms to improve the performance of collective operations on such systems. However, with the sharp growth in the amount of power being consumed by large scale supercomputers, it is also necessary to design collective algorithms in a power-aware manner to minimize the total amount of power consumed by the system with acceptable performance overheads. In this paper, we have proposed efficient power-aware algorithms for collective operations that utilize the Dynamic Voltage and Frequency Scaling concepts along with CPU throttling to deliver fine-grained power savings. We have demonstrated through micro-benchmarks, and application benchmark suites that our proposed methods can deliver higher power-savings than some of the existing power-aware algorithms. Our evaluations have shown that we are able to conserve about 8% of energy with applications such as CPMD and NAS class C kernels. We are keen on extending these power-aware optimizations to the topology-aware algorithms [27] to conserve power on large scale clusters by throttling down all the processes in a rack, during the inter-rank communication phases. Also, since the modern architectures allow for DVFS operations to be performed at the core-level granularity, we are also interested in exploring how intra-node point-to-point operations can be designed to conserve power. We also predicted the potential for power savings with our proposed algorithms if the next generation multi-core processors allowed CPU throttling to be performed at the core-level granularity. Also, the current generation InfiniBand networks do not allow us to dynamically shut-down the links during computation intensive phases of applications. We would also be interested in exploring the design challenges involved with conserving network power dynamically during application execution.

IX. Acknowledgments

We would like to thank Dr. K. Cameron, Dr. R. Teodorescu, Dr. K. Tomko and Hung-Ching Chang for their valueable inputs.

References

[1] Intel, “Intel Nehalem,” http://www.intel.com/technology/architecture-silicon/next-gen/.

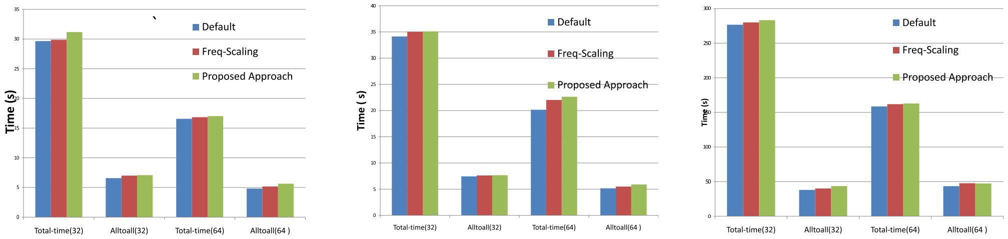

Fig. 10. CPMD Application : Overall Execution time and Alltoall time with 32 and 64 Processes (a)Wat-32-inp-1 (b) Wat-32-inp-2 (c)ta-inp-md

TABLE I
CPMD APPLICATION : POWER STATISTICS

	32 Processes(KJ)			64 Processes(KJ)		
	Wat-32-inp-1	Wat-32-inp-2	ta-inp-md	Wat-32-inp-1	Wat-32-inp-2	ta-inp-md
Default (No-Power)	28.4736	32.76	265.56	31.79	38.68	304.5312
Freq-Scaling	27.096	31.72	259.48	29.944	38.84	289.20
Proposed	27.20	31.36	258.96	29.49	38.13	281.04

[2] InfiniBand Trade Association, http://www.infinibandta.org/.

[3] IEEE 802.3 Ethernet Working Group, "IEEE 802.3," http://www.ieee802.org/3/.

[4] Mellanox Technologies, "InfiniBand and TCP in the Data-Center."

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

[6] Min Yeol Lim and Vincent W. Freeh and David K. Lowenthal, "Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs," in *Proc. of the 2006 ACM/IEEE Conf. on Supercomputing, Tampa, Florida*, 2006.

[7] Vincent W. Freeh and David K. Lowenthal, "Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster," in *Proc. of the 10th ACM SIGPLAN Symp. on Principles and Practice of Par. Prog.*, 2005, pp. 164–173.

[8] V. W. Freeh, F. Pan, N. Kappiah, and R. S. David K. Lowenthal, "Exploring the Energy-Time Tradeoff in MPI Programs on a Power-Scalable Cluster," in *Proc. of 19th IEEE Int. Par. and Dist. Processing Symposium (IPDPS'05)*, 2007.

[9] Rong Ge and Xizhou Feng and Kirk W. Cameron, "Modeling and Evaluating Energy-Performance Efficiency of Parallel Processing on Multicore based Power Aware Systems," in *Proc. of IEEE Int. Sym. on Par. and Dist. Processing*, Rome, Italy, 2009, pp. 1–8.

[10] K. W. Cameron, R. Ge, and X. Feng, "High-Performance, Power-Aware Distributed Computing for Scientific Applications," *Computer*, vol. 38, no. 11, pp. 40–47, November 2005.

[11] Seung Woo Son and Konrad Mlakowski and Guilin Chen and Mahmut Kandemir and Padma Raghavan, "Reducing Energy Consumption of Parallel Aparse Matrix Applications through Integrated Link/CPU Voltage Scaling," *J. of Supercomputing*, vol. 41, pp. 179–213, 2007.

[12] S. Conner, S. Akioka, M. J. Irwin, and P. Raghavan, "Link Shutdown Opportunities During Collective Communications in 3-D Torus Nets," in *Proc. of IEEE Int. Par. and Dist. Processing Sym., IPDPS*, 2007, p. 344.

[13] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda, "High Performance RDMA-Based MPI Implementation over InfiniBand," in *ACM Int'l Conference on Supercomputing*, June 2003.

[14] R. Zamani, A. Afsahi, Y. Qian, and C. Hamacher, "A Feasibility Analysis of Power-Awareness and Energy Minimization in Modern Interconnects for High-Performance Computing," in *Proc. of the 2007 IEEE Int. Conf. on Cluster Comp.*, 2007, pp. 118–128.

[15] MVAPICH2, http://mvapich.cse.ohio-state.edu/.

[16] Top500, "Top500 Supercomputing systems," November 2009, http://www.top500.org/list/2009/11/100.

[17] Jiuxing Liu and Dan Poff and Bulent Abali, "Evaluating High Performance Communication: A Power Perspective," in *Proc. of the 23rd int. conf. on Supercomputing*, 2009, pp. 326–337.

[18] A. R. Mamidala and R. Kumar and D. De and D. K.Panda, "MPI Collectives on Modern Multicore Clusters: Performance Optimizations and Communication Characteristics," in *Proc. of 8th IEEE Sym. on Cluster Comp. and Grid*, 2008, pp. 130–137.

[19] R. Kumar, A. R. Mamidala and D. K. Panda, "Scaling Alltoall Collective on Multi-core Systems," in *Proc. of IEEE Int. Sym. on Parallel and Dist. Processing*, 2008.

[20] A. R. Mamidala, A. Vishnu and D. K. Panda, "Efficient Shared Memory and RDMA Based Design for MPI-Allgather over InfiniBand," in *Recent Adv. in Par. Virtual Machine and Message Passing Interf., LNCS, Volume 4192/2006*, 2006, pp. 66–75.

[21] K. Kandalla, H. Subramoni, G. Santhanaraman, M. Koop and D. K.Panda, "Designing Multi-leader-based Allgather Algorithms for Multi-core Clusters," in *Proc. of IEEE Int. Sym. on Parallel and Dist. Processing*, Rome, Italy, 2009.

[22] G. Chen and K. Malkowski and M. Kandemir and P. Raghavan, "Reducing Power with Performance Constraints for Parallel Sparse Applications," in *Proc. of the 19th IEEE Int. Par. and Dist. Proc. Sym. (IPDPS'05)*, vol. 12, 2005.

[23] Shuaiwen Song and Rong Ge and Xizhou Feng and Kirk W. Cameron, "Energy Profiling and Analysis of the HPC Challenge Benchmarks, journal = "int. j. of high perf. computing app.", volume = "23", issue = "3", pages = 256-276, year = 2009,."

[24] S. Kumar, Y. Sabharwal, R. Garg, and P. Heidelberger, "Optimization of All-to-All Communication on the Blue Gene/L Supercomputer," in *Proc. of 37th Int. Conf. on Parallel Processing*, 2008, pp. 320–329.

[25] Jehoshua Bruck and Ching-Tien Ho and Shlomo Kipnis and Derrick Weathersby, "Efficient Algorithms for All-to-All Communications in Multi-Port Message-Passing Systems," in *Proc. of the 6th ACM Sym. on Par. Alg. and Arch.*, 1994, pp. 298–309.

[26] The Ohio State University, "OSU MPI Benchmarks," http://mvapich.cse.ohio-state.edu.

[27] K. Kandalla, H. Subramoni, and D. K.Panda, "Designing Topology-Aware Collective Communication Algorithms for Large Scale InfiniBand Clusters: Case Studies with Scatter and Gather," in *IEEE Int. Sym. on Par. and Dist. Proc., to appear*, 2010.

[28] T. Hoefler, T. Schneider, and A. Lumsdaine, "The Impact of Network Noise at Large-Scale Communication Performance," in *Proc. of IEEE Int. Sym. on Par. and Dist.*, 2009, pp. 1–8.

[29] P. Patarasuk and X. Yuan, "Efficient MPI Bcast across Different Process Arrival Patterns," in *Proc. of Int. Par. and Dist. Proc. Sym. (IPDPS)*, Miami, FL, 2008.

[30] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective Communication Operations in MPICH," in *Int. J. of High Perf. Comp. App.*, 2005, pp. (19)1:49–66.

[31] "MVAPICH2: High Performance MPI over InfiniBand and iWARP," 2010, http://mvapich.cse.ohio-state.edu.

[32] D. H. Bailey et al, "The NAS Parallel Benchmarks Summary and Preliminary Results," in *Proc. of ACM/IEEE Conf. on Supercomputing*, 1991, pp. 158–165.

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

[34] CPMD Data Sets, "Compute Canada," https://computecanada.org/?pageId=156.