Simulation-based Performance Analysis and Tuning for a Two-level Directly Connected System

Ehsan Totoni[*], Abhinav Bhatele[†], Eric J. Bohm[*], Nikhil Jain[*], Celso L. Mendes[*], Ryan M. Mokos[‡], Gengbin Zheng[*] and Laxmikant V. Kale[*]

[*]Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
[†]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551, USA
[‡]National Center for Supercomputing Applications, University of Illinois, Urbana, IL 61801, USA
E-mail: {totoni2, ebohm, nikhil, cmendes, gzheng, kale}@illinois.edu, bhatele@llnl.gov, mokos@ncsa.illinois.edu

Abstract—**Hardware and software co-design is becoming increasingly important due to complexities in supercomputing architectures. Simulating applications before there is access to the real hardware can assist machine architects in making better design decisions that can optimize application performance. At the same time, the application and runtime can be optimized and tuned beforehand. BigSim is a simulation-based performance prediction framework designed for these purposes. It can be used to perform packet-level network simulations of parallel applications using existing parallel machines. In this paper, we demonstrate the utility of BigSim in analyzing and optimizing parallel application performance for future systems based on the PERCS network. We present simulation studies using benchmarks and real applications expected to run on future supercomputers. Future petascale systems will have more than 100,000 cores, and we present simulations at that scale.**

Keywords-**simulation, performance prediction, mapping, system noise, collective communication**

I. Introduction and Motivation

Some of the largest supercomputers available today cost tens or even hundreds of millions of dollars. They include more than a hundred thousand processor cores, and complex interconnection topologies. Porting and tuning science and engineering applications to these machines, after their deployment, can easily take months to years. To run them with lower efficiencies than feasible in the intervening months, which is necessary to utilize the machine, represents a huge waste of resources. Although some aspects of porting and tuning can be carried out ahead of time, much effort depends on the specific features of the target machine.

BigSim aims to assist with this situation, using a unique "emulation followed by simulation" approach. The emulation phase allows the users to run their application at the target scale while using a much smaller machine. For example, a one million core application run can be emulated using ten thousand cores of an existing machine. This aspect is supported by an adaptive runtime system in CHARM++ and Adaptive MPI [1]. Emulation helps application developers identify scaling bugs in the data structures and code. More importantly, it records the dependencies between computations and messages. It also records salient features about the computational blocks. The obtained traces are then used by a multi-resolution simulator to produce performance traces and timings as if the application was run on the target machine.

BigSim's multi-resolution aspects cover both sequential execution and communication. For sequential execution, one can either use a simple scaling factor, or a detailed model based on performance counters, or even use timings based on cycle-accurate simulations of the processor. Another option is to get the timings by running the sequential blocks on an existing machine with the same processor. For communication, one can use a simple latency-bandwidth model or a detailed model of the network including all the switches and buffers. The BigSim methodology has been validated on older machines in the past [2], [3]. We are now using it to tune performance of some applications for the upcoming PERCS systems that will have a novel two-level directly connected interconnection topology and nodes based on IBM's POWER7 processors.

This paper describes some of the performance analysis and tuning experiments that we have carried out with BigSim targeting PERCS systems. The experiments include alternate schemes for mapping processes to processors for a simple application prototype, analysis and design of an all-to-all operation within a "supernode" – a hierarchical component of the system, and simulation-based analysis of the effect of noise on a few applications/kernels. These studies cover important applications, such as MILC and NAMD. Collectively, they demonstrate the utility of our BigSim framework.

The major contributions arising from this study in using BigSim to predict performance of a future system, include: (a) we show that a careful mapping of tasks to processors can improve overall application performance by 20% in the presented case; (b) we provide a practical technique to quantify the effect of system noise on application performance; and (c) we demonstrate that the level of detail produced by BigSim may provide insights leading to a more advanced algorithm for a collective operation, potentially resulting in a five-fold improvement in its performance.

Unlike other simulators that focus on specific parts of a system (e.g., cycle-accurate simulators or network simulators), BigSim has the unique feature of allowing the simulation of full applications on a future machine. For example, BigSim is not limited to analyzing certain communication patterns, or specific code fragments. Instead, it can simulate the actual computation and communication behavior expected on the target system, and thus provide a much more realistic prediction of application behavior and potential bottlenecks. This is particularly important on a large system, where applications and hardware may interact in very complex ways. Such interactions might be very hard to capture in analytical models, or to represent in simulators with a limited scope.

The rest of this paper is organized as follows. §II reviews related work in the area. §III and §IV briefly describe the PERCS systems and BigSim, respectively. §V presents results of BigSim validations for PERCS systems. §VI-A analyzes the gains that can be obtained with proper mapping of tasks on the machine. §VI-B discusses BigSim's capabilities to model the effects of system noise on applications. We present a concrete case of optimization in §VI-C, with an example of an important collective operation, and conclude our work in §VII.

II. Related Work

The Structural Simulation Toolkit (SST) [4] has been used to model the Red Storm system and perform "what if" analysis. However, the focus is on the effect of hardware details on MPI latency and bandwidth, and full application performance was not modeled. Furthermore, the framework works on instruction-level traces, which may not be needed and even become infeasible for simulations at large scale.

PSINS [5] is a trace-driven simulation framework for MPI applications similar to BigSim. It intercepts MPI calls of the application to produce traces of computation and communication. Although their traces may look like BigSim traces, those traces have to be produced on the same number of MPI processes (in contrast to the user-level threads of BigSim), which makes the approach intractable for large target supercomputers. In addition, it uses buses to model the network, and it does not consider different topologies. Dimemas [6] also uses buses to model the network, which is not accurate for our purposes. As will be seen in later sections, the topology, can be very important for tuning many aspects of the system and the applications.

IBM's MARS [7] (also called MERCURY) is a framework to simulate full HPC systems, down to the instruction and flit level. This approach is very useful for detailed network design and tuning, but it is an overkill for large-scale application studies. BigSim's level of abstraction for networks is at the packet level, which is efficient and sufficient for its purposes. Nevertheless, MERCURY is being used to validate the network simulation component of BigSim for PERCS network.

Hoefler *et al.* [8] show that simulation is necessary to realistically inspect noise's influence. However, their approach uses existing MPI traces, and it cannot generate traces for machines larger than those that currently exist. In addition, the largest simulation conducted for real applications is for 32,768 processors, which is relatively small considering the current petascale systems. Some of the other noise studies are focused only on collective operations [9]; however, as shown in [8], other communication patterns of applications also have crucial impact on performance.

III. The PERCS Architecture

PERCS is a supercomputer design by IBM that uses POWER7 processors and a two-level directly-connected network [10]. This was the intended design for the Blue Waters system at Illinois, however, the plans changed later. In the PERCS design, the system is divided into *supernodes* containing 32 nodes each. These nodes are evenly grouped into four drawers, yielding eight nodes per drawer. Each node is connected to the seven other nodes in its drawer with 24 GB/s LLocal (LL) links, and to the other 24 nodes in its supernode with 5 GB/s LRemote (LR) links. All supernodes are connected to one another with 10 GB/s D links. This unique, high level of connectivity requires at most three hops for a direct route between any pair of nodes (L-D-L), and at most five hops for an indirect route (L-D-L-D-L).

A compute node contains four POWER7 chips, each with eight processing cores, and a Hub chip. The four POWER7 chips, which form a Quad Chip Module (QCM), have access to 192 GB/s of bandwidth over four links (24 GB/s per link in each direction) for sending messages to the Hub chip. This Hub chip interfaces the QCM with the network through two Host Fabric Interface (HFI) components, and provides network switching via an Integrated Switch Router (ISR). The Hub chip also takes part in the cache coherency of the four POWER7 chips and improves collective performance with a Collective Acceleration Unit (CAU).

Figure 1 shows a high-level view of the PERCS network. Its architecture, and thus its performance characteristics, differ greatly from today's common topologies (such as meshes and fat-trees). This uniqueness presents several challenges for software developers. For example, applications, runtime environments, and common operations (like collectives) that have been optimized and tuned for typical networks may perform poorly due to inefficient use of the network. Moreover, the required theoretical and practical experience does not exist yet for guiding such optimizations. This, coupled with the extra complexity added by the advanced features of the POWER7

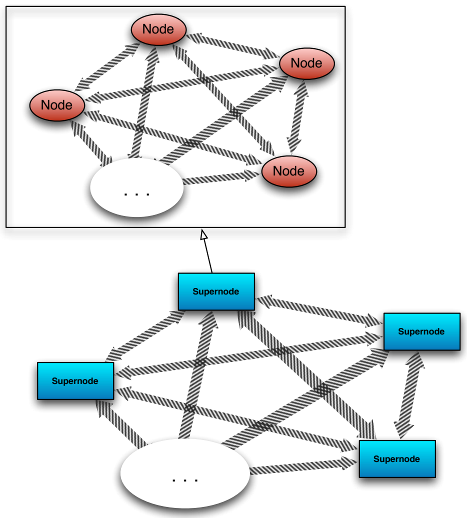


Fig. 1. High-level view of the two-level PERCS topology: 32 fully-connected nodes form supernodes, which are in turn fully connected.

and Hub chips, leaves simulation as the most effective means of analyzing and testing optimization strategies before the system comes online.

IV. BigSim Simulation Framework

BigSim is a simulation-based framework used for predicting the behavior of real applications on large parallel machines [3]. It is based on two major components, as illustrated in Figure 2. First, the Charm++ or MPI application is run on the BigSim emulator, which records the time required to process the code's Sequential Execution Blocks [2] (SEBs) for that particular machine and the occurrence of communication events. These events and time-stamps are written to trace files, which are then fed into the BigSim simulator. BigSim supports large simulations with different levels of fidelity. Being a parallel application itself, BigSim is built on Charm++, which allows it to use Charm++'s processor virtualization ability to simulate multiple target processors on each physical processor [1]. The emulation of parallel applications largely depends on the memory footprint of these applications. For applications that have a small memory footprint, such as NAMD [11], BigSim can emulate a machine with hundreds of thousands of processors on a few thousand physical processors, as demonstrated later in this paper. For applications with a large memory footprint, BigSim can employ various techniques such as out-of-core execution and memory aliasing.

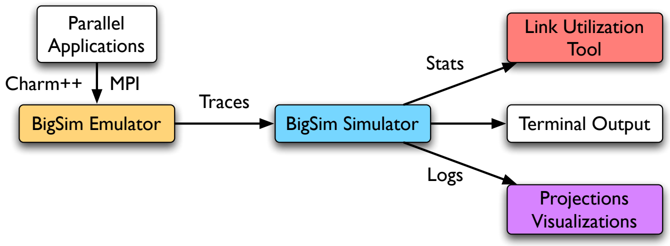


Fig. 2. Architectural overview of the BigSim framework

The BigSim simulator uses a network model, selected by the user, to adjust the send and receive times of the messages recorded in the logs, thereby producing the final simulation result [3]. These network models include objects that represent processors, nodes, network interface cards, switches, and network links, and they can simulate contention in the network. Several different topologies and routing algorithms are available, as well as virtual channel routing strategies and adaptive routing. The user may choose between direct and indirect routing, and may configure a number of network parameters such as link latencies, bandwidths, and buffer sizes. Finally, instead of selecting a particular network, the user may also select a simple linear model based on bandwidth and latency parameters for predicting communication times.

BigSim was designed to assist a user in investigating the behavior of an application on a particular network than to simply predict the application's execution time. As a result, it offers several forms of output including predicted execution time of the user code blocks, end-of-simulation network link utilization statistics over time, which can be graphically displayed using existing visualization tools.

One goal of the BigSim project is to give application programmers the opportunity to tune their codes for a new machine, even before it comes online. To that end, over the past couple years, in collaboration with IBM, we have built a BigSim model of the PERCS network and validated it against IBM's MERCURY simulator. Our model simulates processors, nodes, HFIs, ISRs, and all of the LL, LR, and D links. It implements virtual channels and Hub chip buffers, and it delays packets in the network appropriately if congestion occurs. Since it is a packet-level simulator rather than a flit-level one, it is efficient, while still correctly modeling link contention and buffers. Having such a model now allows us to look for bottlenecks, not only in applications that will eventually be run on PERCS systems, but even in simple, widely-used algorithms such as those for all-to-all. Our simulation approach has had prior validations on various machines, including Linux clusters and Blue Gene/P, with NAS Benchmarks and NAMD [2], [3].

To enable simulations of large systems with hundreds of thousands of processors, we extended BigSim with several new features that improve productivity significantly. For example, to reduce the need to re-run the emulation of an application, we created a parameter replacement scheme that can modify message sizes or sequential block durations when the application is simulated. Another recent improvement was the addition of "skip points", which are marks inserted in the application to allow the simulator to skip uninteresting parts of the execution.

V. Validation of the Simulator

Validation of a simulator, by a team distinct from the designers of the target system, prior to the construction of the system, presents several challenges. A modicum of confidence can be gained by comparisons against a simulator implemented by the design team. In later stages of the project, basic validation can be performed against hardware as it becomes available at various scales.

A. Comparison to MERCURY

We conducted several early tests to validate the PERCS network model in BigSim against IBM's MERCURY simulator. Those simulations can include any number of nodes or supernodes, and we selected network traffic patterns that could also be produced by MERCURY.

Several ping-pong experiments were performed, primarily to measure latency differences. Tiny, 2-byte messages were passed between nodes, exercising all possible combinations of LL, LR, and D links. MERCURY's results differed by 0.6% to 1.1% from BigSim's. Another test, primarily intended to examine bandwidth, involved executing an all-to-all communication pattern within a supernode. Each of the 1,024 cores sent a 51 KB message to all other cores. The results between the two simulators only differed by 0.5%.

B. Validation on a Power 775 Drawer

During recent months, we have periodically had remote access, for brief periods, to a prototype of a Power 775

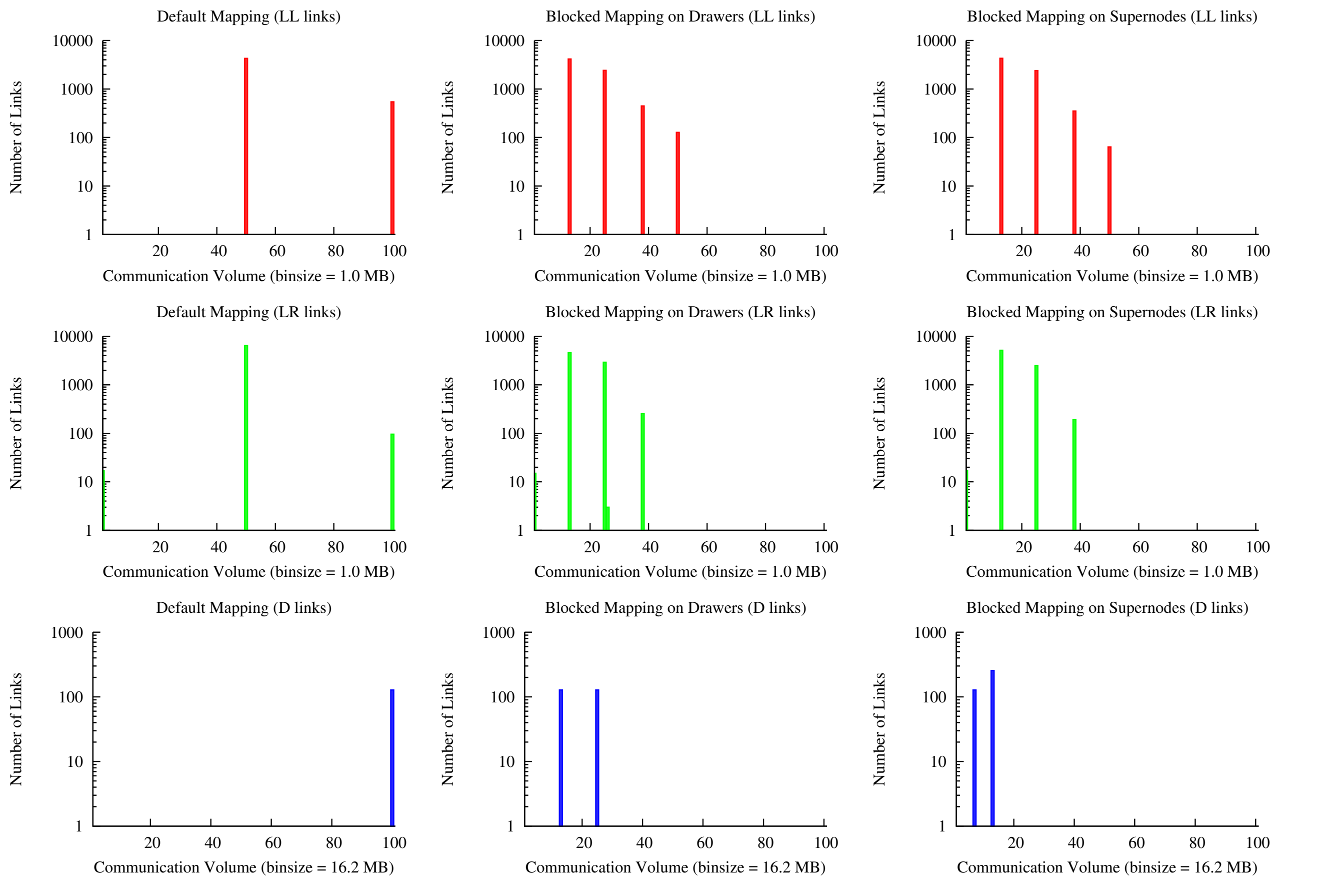


Fig. 3. Link utilization for different mappings of a 3D Stencil to 64 supernodes. Each column represents utilization of the LL, LR and D links under a given mapping; top row corresponds to LL links, middle row to LR links and bottom row to D links.

drawer installed at IBM. This prototype drawer contains eight nodes, with four POWER7 chips each, 256 cores in total, and a development version of the Hub interconnect. It must be noted that this prototype is still significantly different from the planned Power 775 drawer, both in terms of hardware configuration and of the software stack. Nevertheless, it is a good platform for minimal BigSim validation.

We conducted experiments to validate BigSim's results with those observed on this prototype drawer. One of those experiments involved the same all-to-all operation mentioned previously, with exchanges of size 51 KB. In this test, the simulation was based on a regular MPI program that contained a few calls to `MPI_Alltoall` followed by a barrier. We simulated this code in BigSim, for a target configuration of eight PERCS nodes (256 cores), and also ran it on the prototype drawer. The execution on the drawer resulted in a time of 22.6 ms for the `MPI_Alltoall`. Meanwhile, BigSim's trace-driven simulation of that code produced a prediction of 20.2 ms, a result that was within nearly 10% of what we observed on the actual drawer.

VI. Prediction Experiments

We now demonstrate the usefulness of BigSim's capabilities, through various simulation experiments aimed at providing insights about the behavior of applications on future PERCS systems.

A. *Topology Aware Mapping*

For any new machine, various design choices for job scheduling, routing and mapping of tasks to physical processors need to be made. Making these choices by running experiments on the real machine results in waste of both time and money. Using BigSim, we can answer the following questions before the machine is installed:

- Should the job schedulers be topology aware? Should the node allocation for a particular job be at the granularity of supernodes, drawers or nodes?
- Should the routing be direct or indirect? Can indirect routing alleviate congestion on the PERCS network?
- Should tasks be mapped in a topology aware fashion?

In this section, we demonstrate the utility of BigSim in making some of these choices. We used a simple three-dimensional seven-point stencil to study some of the questions raised above and simulate various mappings for 64 supernodes of the PERCS system. Each MPI task holds a data array of $256 \times 256 \times 256$ doubles and sends ghost layers to six neighbors for a Stencil-like exchange. Hence, the size of each exchange on the boundary is $(256 \times 256) \times 8$ bytes = 512 KB.

We tried three different mappings of the 65,536 MPI tasks onto the 64 supernodes. The first mapping is the default MPI rank ordered mapping where the first 32 tasks are placed on the first node, the next 32 on the second and so on. To compare with this, we tried two other mappings that block MPI tasks into 3D cuboids, which can nicely map onto the nodes and drawers of the machine. For example, the second mapping places blocks of $4 \times 4 \times 2$ on each node and blocks of $8 \times 8 \times 4$ on each drawer. In the third mapping, in addition to the blocking for node and drawer, tasks are also blocked on supernodes, with block dimensions being $16 \times 8 \times 8$.

Figure 3 presents histograms depicting the number of bytes sent over the LL, LR and D links in the 64 supernode subsystem. The first column represents the results for the default mapping and the second and third column for the two intelligent mappings described above. Each bin shows the number of links which had a certain range of bytes passing through them. It can be seen that the intelligent mappings for the 3D Stencil reduce the number of links that have a very high utilization. The aim is to reduce hot-spots that might appear on a few links, which can slow down the application. Intelligent mappings are able to lower the maximum data being sent over any link (as fewer bins on the right contain any links).

Figure 4 shows the time spent in communication and overall execution of one iteration of the 3D Stencil for different mappings. The communication reduces by 80% using an intelligent mapping and the overall time per iteration reduces by 20%. Hence, reducing congestion and hot-spots on the links translates into improvement in application performance also. This illustrates the use of BigSim as a tool to evaluate different mappings without access to the actual machine and decide on the best mapping to be used for actual runs.

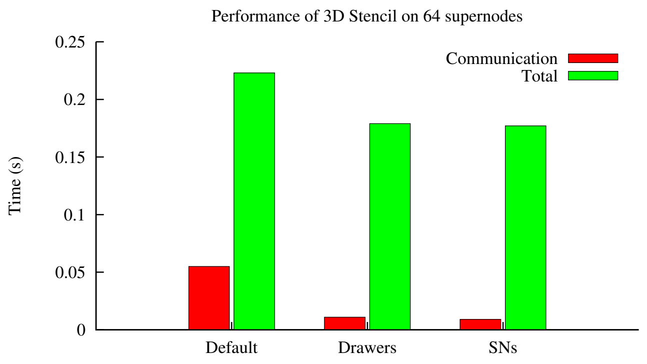


Fig. 4. Comparison of communication and overall execution time (in seconds) of 3D Stencil on 64 supernodes for different mappings

B. *Effects of System Noise*

Many supercomputers use full-fledged operating systems (OS) on their compute nodes. This is important for several applications that require a full kernel to execute. However, current supercomputers that made the same design choice (such as Jaguar and Ranger) demonstrate significant system noise on their compute nodes [12]. System noise can have sources other than the OS, such as architectural effects. It can affect certain categories of parallel applications – those that are fine-grained and also those that have long critical paths.

Using BigSim and a recently added feature in it, we can study the impact of system noise on applications that will run on large-scale supercomputers like those with the PERCS network. Different kinds of noise patterns can be introduced in BigSim simulations:

- Noise traces can be collected from an existing (similar) node and then a statistically similar noise can be introduced on different nodes of the full simulated system.
- Artificial noise can be introduced by simulating perturbations of different periodicities and durations.

Several noise studies can be done to assess the impact of noise on applications. The applications that we used for the studies are NAMD [11], which is a scalable molecular dynamics code, and MILC (MIMD Lattice Computation) [13], a quantum chromo-dynamics program. We also used two versions of a synthetic micro-benchmark (*kNeighbor*) that iteratively executes a nearest-neighbor communication followed by some amount of sequential computation. One version of kNeighbor contains an all-reduce at the end of each iteration, and the other version does not have the all-reduce. The goal is to illustrate how noise may affect the communication.

For NAMD, the number of target processors was 256K, and a 10 million-atom water dataset was used as the input; we only simulated a few time-steps of NAMD. The baseline time per step for NAMD is about 1.29 ms. In MILC, the simulation involved the entire `su3_rmd` code, with a lattice of size $4 \times 4 \times 3 \times 6$ on each core; a target configuration of 4K processors was used. Given the frequent global synchronization in MILC, even at this smaller machine configuration, it is possible to observe significant impact of noise on performance. The baseline value of execution time for MILC is 491.9 ms. kNeighbor was calibrated to have 1 ms of computation per step, and each processor communicated with eight neighbors. For the version with the all-reduce, the original simulated time per step was about 2.4 ms while in the other version it was around 1.6 ms. Each of the simulations conducted at that scale was completed in tens of minutes to a couple of hours running BigSim on just a single node, which shows the efficiency of this tool. As an example, the collection of trace files for NAMD were about 25 GB in size.

Figure 5 shows the effect of increasing the frequency of a noise pattern on application performance. With such a study, system designers can gain insight about the frequency where performance starts getting affected by noise of a certain duration (such as an OS daemon), and they can seek measures to avoid those effects in the future system. Meanwhile, application developers can characterize and compare their application's noise sensitivity and improve it if it is not acceptable. As Figure 5 shows, with a decrease in the noise period (i.e. increase in noise frequency), the overhead on execution time increases, as expected. However, many noise characteristics of applications are not easy to identify without detailed simulations. As an example, NAMD is tolerant to low-frequency noise (the curve is mostly flat near the left) but, at some frequencies, there is a sudden increase in execution time and then the curve becomes flat again. Thus, it is sensitive to a certain range of frequencies, suggesting that tuning of the OS daemons' frequencies (or any other source of noise) can have significant impact on application's performance.

For kNeighbor, with naive intuition, one would expect the version with the all-reduce to have at most 4% increase in

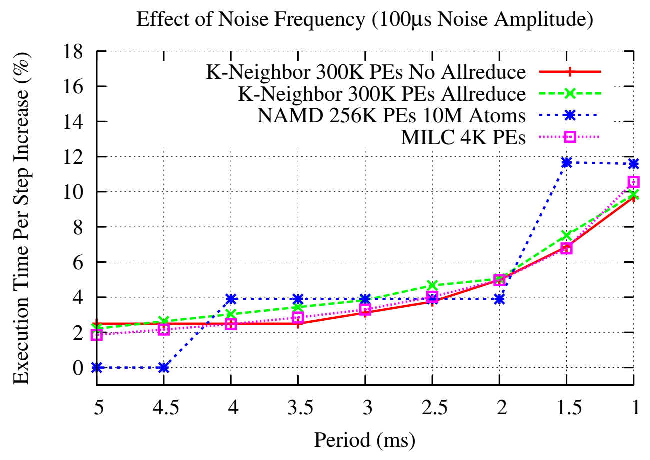


Fig. 5. Effect of increase in frequency of noise patterns on performance

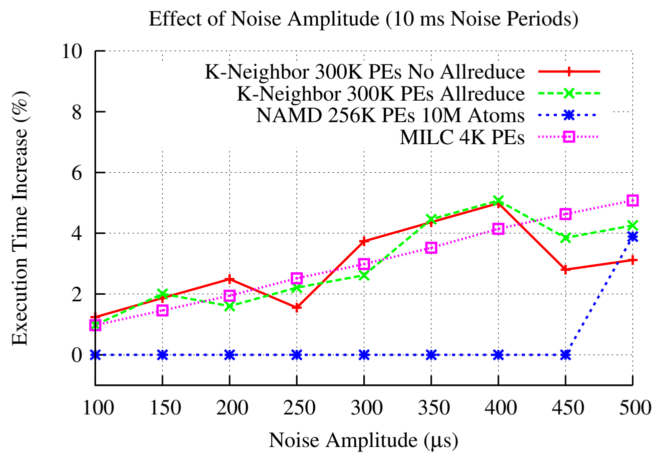


Fig. 6. Effect of increase in amplitude of noise patterns on performance

its execution time. This is because the computation quantum is 1 ms and, during that interval, there can be only one occurrence of a perturbation (since the perturbation periods in Figure 5 are always greater than 1 ms). Given that these perturbations have an amplitude of 0.1 ms, the maximum duration for an iteration would be extended from 2.4 ms to 2.5 ms, representing a 4% overhead. However, with the increase in noise frequency, there is a higher chance that very short MPI calls, used to implement the all-reduce, get perturbed by the noise too. This phenomenon is reflected towards the right end of Figure 5, and we confirmed this effect through detailed analysis of the BigSim traces. Meanwhile, because MILC is more sensitive to noise in general (due to a global-sum on each iteration of its conjugate-gradient solver), even executions with just 4K processors get affected by noise, as much as larger executions of other applications (e.g. the 256K processor run of NAMD). This shows that noise may be a concern for small-scale jobs on supercomputers as well.

Figure 6 shows a similar study that inspects different amplitudes of a fixed-frequency noise (10 ms period). In this figure, the NAMD curve is flat for a while, probably due to noise absorption (elaborated in previous works [8]). Comparing Figure 6 to 5, one can conclude that trading amplitude for frequency is beneficial for this type of application. On the other hand, MILC is sensitive to noise amplitude and is not in that category. This illustrates the usefulness of this technique for comparing design alternatives. Suppose there are two choices such as a high-amplitude, less-frequent noise, and a low-amplitude but more frequent noise. For instance, these could correspond to cluster monitoring daemon activities or different garbage collection strategies in runtime systems, such as full or incremental garbage collection. This choice of which noise to keep, can be made by using the two graphs discussed and comparing the two points on them, corresponding to those alternatives. In the case of NAMD, for example, our results indicate that low-frequency noise will win in many cases.

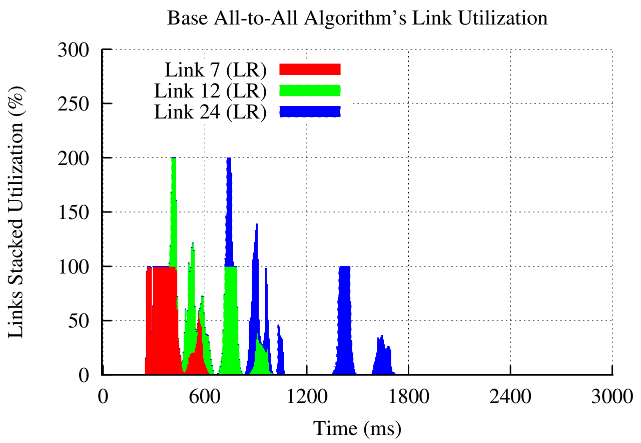
C. All-to-all Optimizations

`MPI_Alltoall` is an important collective operation, with extensive usage in applications, such as FFT and matrix transpose. It is also among the most communication intensive collective operations performed in modern day parallel applications. As a result, it may suffer from scaling problems for large data exchanges over large systems. Several algorithms have been proposed for it in the literature; most of them perform well for a certain range of data sizes exchanged. We narrow our focus to `MPI_Alltoall` for large data sizes and show how simulation can provide insight about internal behavior and details of a system. Moreover, we show how this insight may result in up to five-fold improvement of this important operation, before the future system arrives.

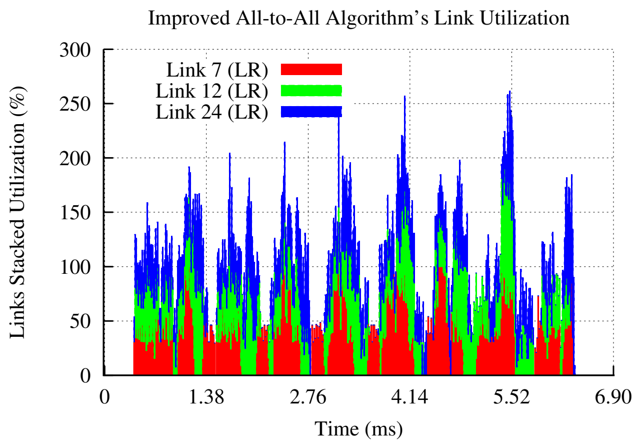
The pairwise-exchange algorithm [14] for an *all-to-all* has been found to achieve better results on most machines. In each step of the algorithm, $P/2$ pairs of tasks perform a tightly coupled send-receive operation. The communication pattern has been found to have minimal congestion for topologies like torus and fat-trees, with a small number of independent paths between nodes. However, as we show, direct implementations of the algorithm will perform poorly on the PERCS network.

We simulated `MPI_Alltoall` using the pairwise-exchange algorithm for a supernode of PERCS, with large data sizes being sent to each task. This scenario is of practical interest; as an example, it is desirable to allocate on the same supernode all the tasks of each sub-communicator in a typical FFT implementation, such that the all-to-all happens within supernodes. Figure 7(a) presents a stacked chart for link utilization of three arbitrarily selected links of a QCM during an `MPI_Alltoall` of size 1 MB. The three link utilizations are stacked to show the overlap of usage (hence it can go beyond 100%). One can observe that the utilization of these links is interleaved in time. A similar pattern occurs if all the links of a QCM are plotted. This observation shows that the pairwise-exchange algorithm causes the links to be used in a shifted manner. However, given the fully-connected nature of the network in a supernode, it is desirable to utilize all links stemming from a QCM simultaneously. This motivated us to consider a more advanced implementation of an all-to-all, to enable simultaneous data transfers on all links.

We propose the following carefully-determined ordering in which the sends from a task should be performed, to



(a) Baseline pairwise-exchange algorithm



(b) Proposed all-to-all algorithm

Fig. 7. Link utilization for three arbitrarily selected links of a QCM for the baseline and new all-to-all algorithm

(a) simultaneously utilize links stemming from a QCM, and (b) avoid the undesired link congestion. We describe the scheme assuming a node-level all-to-all network with n nodes, each containing c cores:

1) Consider a list of $t = n * c$ tasks running on n nodes with c cores each.
2) Each task has to send $t - 1$ messages, of which sets of c destination cores lie on a given node. Any core can reach a particular set of c cores by using the direct link between the destination node and its home node.
3) In phase i ($0 \leq i \leq n - 1$), core j ($0 \leq j \leq c - 1$) on every node sends data to the set of cores residing in the $((j + i) \bmod n)$th node.

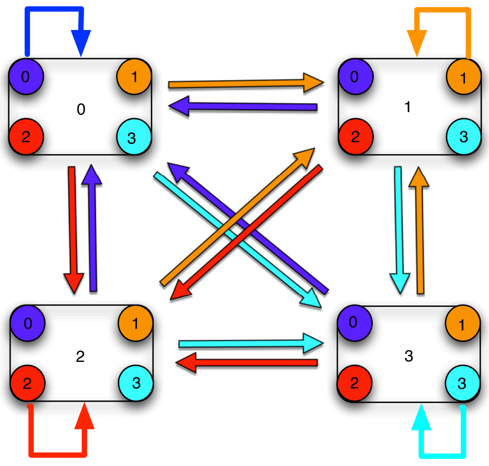


Fig. 8. Sends in first phase of the new all-to-all algorithm

This scheme ensures that different cores of a node use different links, for better link utilization. For example, consider the communication for phase 0, as shown in Figure 8. This figure assumes an application consisting of 16 tasks running on a four-node system, with four cores per node. The circles represent a core/task and a box represents a node containing the cores. An edge from a core to a box means that the source core sends data to all cores in that destination node. In phase 0, core 0 of each node sends data to the set of cores residing on node 0, as shown by blue edges. The sends to nodes 1, 2 and 3 are represented by orange, red and cyan edges, respectively. Such a communication pattern ensures that all the links of each node are being utilized simultaneously. On a PERCS system, for cases in which all the cores of a supernode execute the `MPI_Alltoall`, this scheme will use all 31 links going out from a QCM simultaneously.

Figure 7(b) presents a stacked chart with the new link utilizations, for the same three links shown earlier in Figure 7(a). Note that the new scheme is much faster, hence the scale of the time axis in Figure 7(b) is different. These results demonstrate that the new scheme overlaps the usage of links for most of the simulation period. Thus, the sequential usage of different links no longer occurs, and utilization improves significantly.

We consider an application with 1024 tasks running on one supernode of a PERCS system, with the amount of one-way data exchanged between two cores equal to m bytes. Consider the volume of data exchanged between two QCMs: QCM-1 contains 32 cores, each of which has to send m bytes to 32 cores in QCM-2. Thus, the total data communicated from QCM-1 to QCM-2 is $d = 1024 \times m$ bytes. A QCM sends d bytes to every other QCM over independent links. In the best scenario, the lower bound for the time taken for the `MPI_Alltoall` will be determined by the time taken to send this data on the slowest link. LR links, with a bandwidth of 5 GB/s, are the slowest links, and thus a lower bound for the time taken will be $d/5$ nanoseconds.

We present a comparison between our scheme and the default `MPI_AlltoAll` in Figure 9. The plot shows the simulation times for the baseline all-to-all, our new all-to-all and a bandwidth-based lower bound, for message sizes from 32 KB to 4 MB. Note that the impact of message startup time has been ignored in the theoretical numbers. We demonstrate 3x to 5x speedups for message sizes beyond 256 KB. More importantly, as the message size increases, the performance of our new scheme remains closer to the theoretical lower bound. One can also observe that the enormity of bandwidth capacity in the PERCS network results in nearly constant execution time for messages of size below 256 KB in the new scheme.

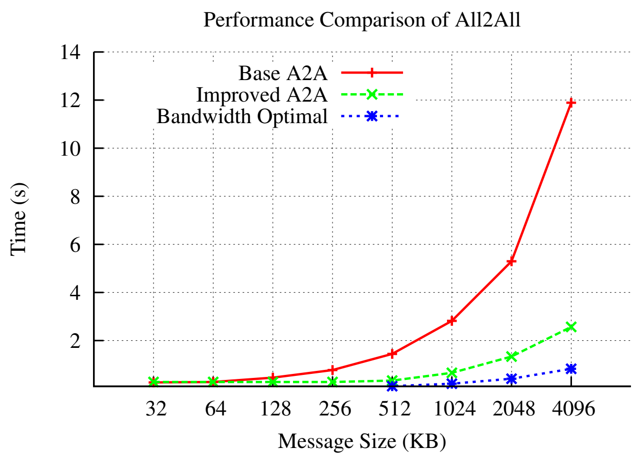


Fig. 9. Performance comparison of different algorithms for all-to-all

VII. Conclusion

Porting and tuning applications for a new system is typically a time-consuming task. This task becomes harder when the new system is significantly larger than existing systems. Simulation-based analysis is an effective technique to prepare applications for future machines. With a simulator such as BigSim, which has the capability to emulate a full application's behavior on a target system, one can predict the interactions between a given application and the underlying hardware of the machine. This technique can pinpoint potential bottlenecks or help programmers focus their attention on specific parts of the application where performance is expected to be problematic on an upcoming system.

In this paper, we have demonstrated the utility of the BigSim simulator to predict application performance on future PERCS systems. We show various benefits that BigSim can provide to a future user of a PERCS system, namely: (a) the effects of different mappings of tasks to processors across the machine: we show a 20% performance gain via an intelligent mapping; (b) the potential impact of system noise on applications: we show a practical technique to introduce noise in the simulation and assess its effects on application performance; and (c) the performance gains that can be achieved by changing the algorithm employed for certain collective operations with large data sizes: we show that a five-fold improvement is expected for an `MPI_Alltoall` within a PERCS supernode.

With a tool like BigSim, users can start tuning their applications before the system arrives. We plan to calibrate BigSim continuously to ensure that it accurately models PERCS systems as such machines become available to us. We also plan to work on the development of other network models for BigSim, such as one for the Blue Gene/Q machine.

Acknowledgments

This work was supported in part by the Blue Waters sustained-petascale computing project, which is funded by the NSF grant OCI 07-25070 and by the state of Illinois. It was also supported in part by a NSF HECURA grant NSF-0833188 and a DOE grant DE-SC0001845 for HPC Colony II. This document was released by Lawrence Livermore National Laboratory for an external audience as LLNL-CONF-500821. Runs for this paper were done on Blue Print and Ember, resources at NCSA, via a Director's allocation for Blue Waters and on TeraGrid resources under allocation ASC050039N.

References

[1] L. V. Kale and G. Zheng, "Charm++ and AMPI: Adaptive Runtime Strategies via Migratable Objects," in *Advanced Computational Infrastructures for Parallel and Distributed Applications*, M. Parashar, Ed. Wiley-Interscience, 2009, pp. 265–282.

[2] G. Zheng, G. Gupta, E. Bohm, I. Dooley, and L. V. Kale, "Simulating Large Scale Parallel Applications using Statistical Models for Sequential Execution Blocks," in *Proceedings of the 16th International Conference on Parallel and Distributed Systems (ICPADS 2010)*, no. 10-15, Shanghai, China, December 2010.

[3] G. Zheng, T. Wilmarth, P. Jagadishprasad, and L. V. Kalé, "Simulation-based performance prediction for large parallel machines," in *International Journal of Parallel Programming*, vol. 33, no. 2-3, 2005, pp. 183–207.

[4] K. Underwood, M. Levenhagen, and A. Rodrigues, "Simulating red storm: Challenges and successes in building a system simulation," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1 –10.

[5] M. M. Tikir, M. A. Laurenzano, L. Carrington, and A. Snavely, "Psins: An open source event tracer and execution simulator," *HPCMP Users Group Conference*, vol. 0, pp. 444–449, 2009.

[6] R. M. Badia, J. Labarta, J. Gimenez, and F. Escale, "DIMEMAS: Predicting MPI applications behavior in Grid environments," in *Workshop on Grid Applications and Programming Tools (GGF8)*, 2003.

[7] W. E. Denzel, J. Li, P. Walker, and Y. Jin, "A framework for end-to-end simulation of high-performance computing systems," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, ser. Simutools '08. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 21:1–21:10. [Online]. Available: http://portal.acm.org/citation.cfm?id=1416222.1416248

[8] T. Hoefler, T. Schneider, and A. Lumsdaine, "Characterizing the influence of system noise on large-scale applications by simulation," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11.

[9] R. Garg and P. De, "Impact of noise on scaling of collectives: An empirical evaluation," in *High Performance Computing - HiPC 2006*, ser. Lecture Notes in Computer Science, Y. Robert, M. Parashar, R. Badrinath, and V. Prasanna, Eds. Springer Berlin / Heidelberg, 2006, vol. 4297, pp. 460–471.

[10] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni, and R. Rajamony, "The PERCS High-Performance Interconnect," in *2010 IEEE 18th Annual Symposium on High Performance Interconnects (HOTI)*, August 2010, pp. 75 –82.

[11] A. Bhatele, S. Kumar, C. Mei, J. C. Phillips, G. Zheng, and L. V. Kale, "Overcoming scaling challenges in biomolecular simulations across multiple platforms," in *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2008*, April 2008.

[12] A. Bhatele, L. Wesolowski, E. Bohm, E. Solomonik, and L. V. Kale, "Understanding application performance via micro-benchmarks on three large supercomputers: Intrepid, Ranger and Jaguar," *International Journal of High Performance Computing Applications (IJHPCA)*, 2010, http://hpc.sagepub.com/cgi/content/abstract/1094342010370603v1.

[13] C. Bernard, T. Burch, T. A. DeGrand, C. DeTar, S. Gottlieb, U. M. Heller, J. E. Hetrick, K. Orginos, B. Sugar, and D. Toussaint, "Scaling tests of the improved Kogut-Susskind quark action," *Physical Review D*, no. 61, 2000.

[14] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, Spring 2005.