SIMULATION-BASED PERFORMANCE ANALYSIS AND TUNING FOR FUTURE SUPERCOMPUTERS

BY

EHSAN TOTONI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Adviser:

Professor Laxmikant V. Kalé

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 thesis, 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 peta-scale systems will have more than 100,000 cores, and we present simulations at that scale.

To my parents and sisters, for their love and support.

Acknowledgements

This work was supported in part by the Blue Waters sustained petascale computing project, which is funded by the National Science Foundation grant OCI 07-25070 and by the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign, its National Center for Supercomputing Applications, and the Great Lakes Consortium for Petascale Computation. Runs for this thesis were done on Blue Print and Ember, resources at the National Center for Supercomputing Applications, and on other TeraGrid resources under allocation TG-ASC050039N.

This thesis is based on a previous publication [1] and I thank all the co-authors. I highly appreciate help and support of my advisor, Professor Laxmikant V. Kalé, and all our group members. Sincere thanks to my family for their endless patience, love and support.

Table of Contents

List of Figures

1 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 and sophisticated 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.

The BigSim project is aimed at assisting with this situation. It uses a unique "emulation followed by simulation" approach. The emulation phase allows the users to run their applications at the target scale while using a much smaller machine. For example, a one million core application run can be emulated using 100,000 cores of an existing machine. This aspect is supported by an adaptive runtime system in Charm++ and Adaptive MPI [2]. Emulation helps application developers identify any scaling bugs in the data structures and code. More importantly, it records the dependencies between computations and messages. It also records a few salient features about each computational block. The obtained traces are then used by a multi-resolution simulator to produce performance traces and timings as if the whole application had been run on the target machine.

BigSim's multi-resolution aspects cover both sequential execution and communication. For sequential execution, one can use either a simple scaling factor, or a detailed model based on performance counters, or even the ability to plug-in timings based on cycle accurate simulations of the processor. Another option is to run the sequential blocks on an existing machine with the same processor and get the timings. For communication, one can use a simple latency-bandwidth model or a fully detailed model of the network including all the switches and buffers.

The BigSim methodology has been validated on older machines in the past [3, 4]. We are now using it to tune performance of some applications for the upcoming PERCS systems that will have a novel interconnection topology, and nodes based on IBM's POWER7 processors.

In this thesis, we describe 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 and latency on a few simple applications/kernels. The studies include some important applications, such as MILC and NAMD. Collectively, these studies 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; (c) we show how to use variations in network parameters of the simulator to quantify the overall impact on application performance; and (d) 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 of processors, or network simulators), BigSim has the unique feature of allowing the simulation of full applications on the future machine. As an example, BigSim is not limited to analyzing certain communication patterns, or specific code fragments. Instead, it will simulate the actual computation and communication behavior expected on the target system, and thus it can 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; those interactions might be very hard to capture in analytical models, or to represent in simulators with a limited scope.

The rest of this thesis is organized as follows. §2 reviews related work in the area. §3 and §4 briefly describe the PERCS systems and BigSim, respectively. §5 presents results of BigSim validations for PERCS systems. §6 analyzes the gains that can be obtained with proper mapping of tasks on the machine. §7 discusses BigSim's capabilities to model the effects of system noise on an application. §8 shows similar effects arising from variations in network parameters. We provide a concrete case of optimization in §9, with an example of an important collective operation, and conclude our presentation in §10.

2 Related Work

The Structural Simulation Toolkit (SST) [5] has been used to model the Red Storm system and perform "what if" analysis. However, the focus was on 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 unfeasible for simulations at large scale.

PSINS [6] 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 [7] also uses buses to model the network, which is not accurate for our purposes. As will be seen in later chapters, the topology, PERCS topology in particular, is very important for tuning many aspects of the system and applications.

IBM's MARS [8] (also called MERCURY) is a framework to simulate full HPC systems. It simulates the details of the system 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 was used to validate the network simulation component of BigSim for PERCS network.

Full execution-driven system simulators (like SIMICS [9]) have been used in different contexts. They model to a level of detail such that one can run the operating system on top of the simulator. In parallel computing, they are particularly useful in the cases of new chip designs, but they cannot simulate beyond small clusters, due to performance reasons. Moreover, large-scale HPC applications generally do not need those details (such as OS details) in most cases. Nevertheless, their simulation output for sequential code segments can be used in BigSim studies for future systems.

Hoefler *et al* [10] showed that simulation is necessary to realistically inspect noise's influence. However, their approach used existing MPI traces, and it cannot generate traces for machines larger than those currently existing. In addition, the largest simulation conducted for real applications was for 32K processors, which is relatively small considering the current peta-scale systems. Some of the other noise studies were only focused on collective operations [11, 12], but as shown in previous work [10], other communication patterns of applications also have crucial impact on performance.

3 PERCS Architecture

PERCS is a supercomputer design by IBM that uses POWER7 processors and a two-level directly-connected network [13]. 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; those nodes are evenly grouped into four drawers (hence, eight nodes per drawer). Each node is connected to the seven other nodes in its drawer with a 24 GB/s LLocal (LL) link, and to the 24 other nodes in its supernode with a 5 GB/s LRemote (LR) link. All supernodes are then connected to each other 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 between any pair of nodes (L-D-L-D-L).

Figure 3.1 shows a high-level view of the PERCS network. As can be seen, this network is not similar to any of today's common topologies (such as mesh or torus). In fact, it has very different performance properties, which have various implications. For example, existing applications and runtimes are designed and optimized for common networks and may not perform efficiently on this new machine. Moreover, there is no body of theory and legacy about optimizing applications or common operations (like collectives) on this network. Thus, effective simulations seem to be the only way to analyze and optimize applications and runtimes before the system comes online.

A compute node contains four POWER7 chips, each with eight processing cores. These 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. The Hub chip, depicted in Figure 3.2, interfaces the QCM with the network through two Host Fabric Interface (HFI) com-

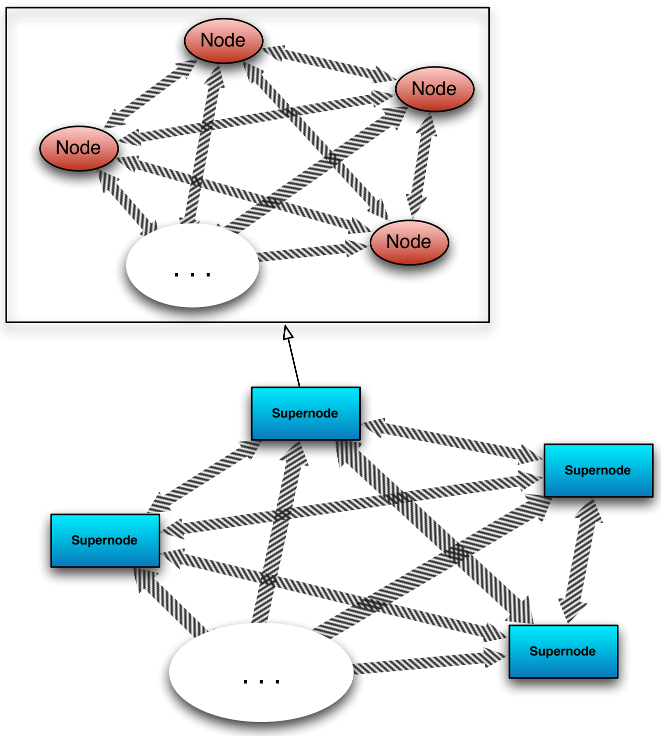


Figure 3.1: High-level view of the two-level PERCS topology: 32 fully-connected nodes form supernodes, which are in turn fully connected.

ponents, and provides network switching via the Integrated Switch Router (ISR). The Hub chip also takes part in the cache coherency of the four POWER7 chips, as well as in increasing the speed of collectives through the Collective Acceleration Unit (CAU). The presence of this advanced chip adds more complexity to the understanding of this network for analyzing applications, without detailed simulation.

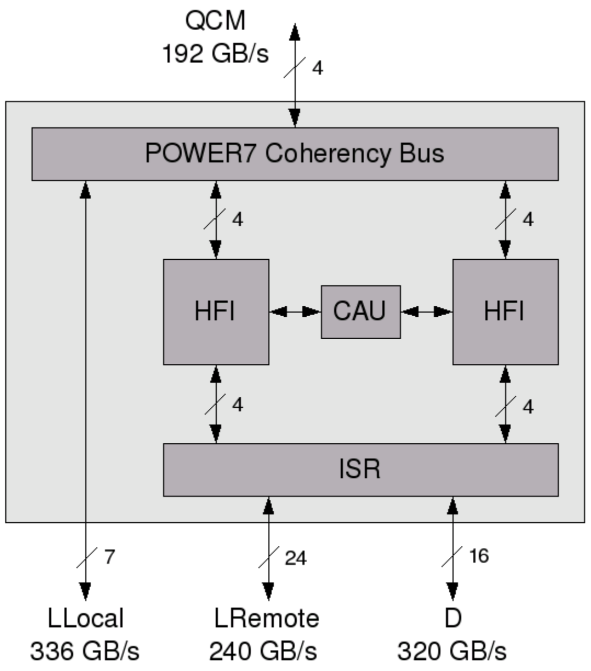


Figure 3.2: Components of the PERCS Interconnect Hub chip relevant for this thesis. Based on Figure 2 of the PERCS interconnect paper [13]. The labeled bandwidths are bidirectional.

4 BigSim Simulation Framework

BigSim is a simulation-based framework used for simulating the behavior of real applications on large parallel machines [14, 15]. To correctly predict the performance of an application, one must compute both the execution time of sequential portions of the code and the communication time. BigSim handles these predictions with two separate components. The application, which must be written in either Charm++ or MPI, is run on the BigSim emulator, which records the time required to process the code's SEBs for that particular machine [3] and the time for communication events. These times are written to trace files, which are then fed into the BigSim simulator. See Figure 4.1 for an architectural overview. It supports large simulations with different levels of fidelity. Being itself a parallel application, BigSim is built on Charm++, which allows it to use Charm++'s processor virtualization ability to simulate multiple target processors on each physical processor [16]. 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 [17], a molecular dynamics simulation program, BigSim can emulate a machine with hundreds of thousands of processors; those studies can be run on only a few thousand physical processors, as demonstrated later in this thesis. For applications with a large memory footprint, BigSim uses various techniques such as out-of-core execution.

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 [18]. 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

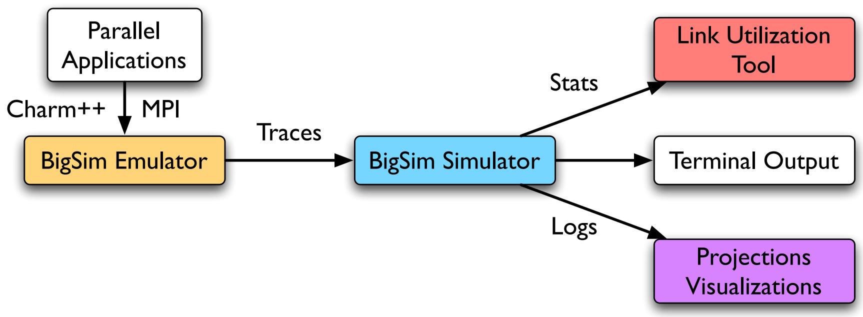


Figure 4.1: BigSim architecture overview

and adaptive routing. The user may choose between direct and indirect routing, and may configure a number of network parameters such as link latencies, link bandwidths, and buffer sizes. Finally, instead of selecting a particular network, the user may also select a simple bandwidth-latency calculation for predicting communication times.

BigSim was designed more to assist the user in investigating the behavior of an application on a particular network than to simply predict the execution time of the application. As a result, BigSim offers several forms of output: end-of-simulation network link utilization statistics; link utilization and contention traces; print statements inserted by the user in the application that are stamped with the simulation's current virtual time; and log files of when the SEBs are executed in virtual time, which can be graphically displayed using an existing visualization tool called Projections.

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 and 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 or contention exist. 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 tune and look for bottlenecks, not only in applications that will eventually be run on the

PERCS systems, but even in simple, widely-used algorithms such as MPI_Alltoall, which we explore later in this thesis. Our simulation approach has been validated before on various machines, including Linux clusters and Blue Gene/P, with NAS Benchmarks and NAMD [3, 4].

4.1 BigFastSim Simulator

BigFastSim is a new version of BigSim's simulator component, intended for easier development and higher efficiency. Because of the overheads and inefficiencies of PDES (like BigSim's original simulator BigNetSim), it is sequential and written in C++. This makes the code much simpler for development and creation of new features. In addition, the simulator will be efficient and will run with much less total resources.

For large-scale simulation studies, it is necessary to be able to run many simulations with minimum possible resources. Each of the analysis that we present later took many simulations to understand the behavior of the applications and the completely new network of the system. In each simulation, the number of target cores is enormous which increases the overheads of the PDES simulation.

Optimization to Avoid Irregular Memory Access Patterns According to performance data, BigSim simulator is memory access bound rather than computation bound. Thus, the emphasis should be on improving locality and avoiding irregular memory access patterns. The major memory footprint and access of the core of the simulator consists of SEBs. These SEBs have many-to-many dependencies to other local (on the same target PE) SEBs meaning that each SEB can have multiple backward and forward dependencies. Furthermore, they have message dependencies to SEBs of other target PEs. These dependencies are shown in Figure 4.2 with solid lines representing local dependencies and dashed lines showing message dependencies. For various reasons, such as flexibility of specific function time replacement and semantics of different languages, computation blocks are divided into separate local tasks in this way. One source of irregular memory accesses is checking backward dependencies and executing forward dependencies in chains. In the general case, for each SEB, all the backward dependencies should be checked and all the forward dependencies should be executed if had not executed before. However, in our observations, the common case is that most of the SEBs can be executed in sequence. So the algorithm tries to execute as much tasks (we use tasks and SEBs interchangeably) in sequence as possible and avoid checking many dependencies in different locations of memory and using pointers. To do this, one variable (*currentTask*) is assigned to each PE that shows the earliest SEB that needs to be executed. Each time a PE can execute SEBs, it tries to advance currentTask and execute the longest possible run. In this way, checking backward dependencies is not needed anymore and forward dependencies are likely to execute. For validity, the forward dependencies that were not done will be checked afterwards and executed in general mode. This variable can also be used in general mode to keep away from checking backward dependencies in many cases (if a backward dependency is before currentTask it has been done for sure and no memory check is needed). In addition, it can be used in "windowing scheme" to keep track of SEBs that are already executed.

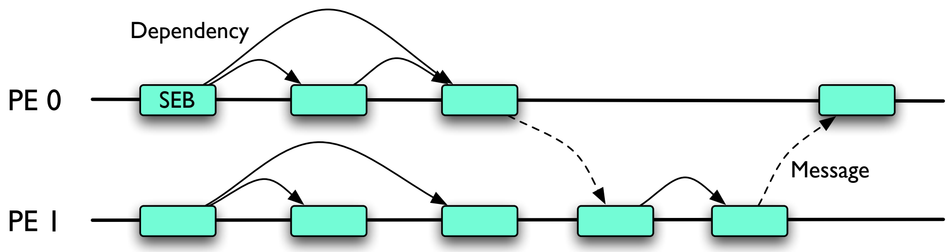


Figure 4.2: Example of dependencies between SEBs

Optimization to Target PE's Message Queue: Another critical optimization in that scale concerns message receipt of a target PE. When a PE receives a message, the task that is dependent on that message should be found. Without optimization it involves searching through tasks for each message which becomes bottleneck. The solution implemented in BigSim is to add a mapping data structure to keep destinations of messages. During initialization and loading of tasks, the message id that the tasks depend on is recorded for each task. This does not add significant overhead to the initialization. Thus, during execution, no search through tasks is needed to receive a message. This optimization has improved the performance of BigFastSim up to two orders of magnitude.

Other New Features: In the scale of peta-scale systems, trace files will be so big that they do not fit into memory. One solution which is implemented into BigSim is windowing scheme. By windowing, only a number of tasks (a window) are loaded in the memory at a time and the window shifts during the execution. Because of the dependencies among tasks, the window has to be adaptive to load more tasks if needed. Another useful feature to save execution time and memory is skip points. Skip points are marks in the program to skip uninteresting parts to accelerate the simulation. For example, an skip point after startup is useful in most of the cases. Another use case is having skip points before and after iterations of scientific applications to predict only certain iterations. To complement the flexibility and control of BigSim, a user can have “BgPrints” to print text and/or timings at desired points in the program to get insight of the execution on the target system.

To enable simulations of large systems with hundreds of thousands of processors, we added new features to BigSim that improve productivity significantly. For example, to reduce the need to re-run the emulation of user applications, we use a parameter replacement scheme. In this approach, the user can specify (on the command line) all the message sizes of the traces above a cut-off to be replaced by another size during the simulation. The purpose of the cut-off is to ignore the small messages of startup or synchronization and only consider data messages to preserve correctness. The user can also replace the duration times of Sequential Execution Blocks (SEBs) with a particular name and with a timing above a cut-off with another duration time. With this feature, re-emulation or changing the traces for each simulation is not necessary anymore in many cases. In addition, some large emulations that are constrained by memory usage or execution time are avoided. Replacements have been particularly used to accelerate the studies of all-to-all and topology mapping in later chapters, as they employ regular size messages.

5 Validation Results

Validation of a simulator, by a team distinct from the designers of the target system, prior to the construction of the system, presents several important challenges. In the absence of a physical environment to perform experiments, the simulator can be validated against analytical models to achieve a modicum of confidence in the result. Additional confidence can be gained by comparisons against a simulator implemented by the design team. Finally, in later stages of the project, basic validation can be performed against hardware as it becomes available at various scales. Presentation of data from these studies is complicated by the fact that fundamental components of the PERCS network design will remain undisclosed proprietary intellectual property of IBM, and part of the data obtained on pre-release hardware is covered by non-disclosure agreements, until sometime after the hardware itself becomes generally available.

5.1 Comparison to MERCURY

Several early tests were conducted to validate the PERCS network model in BigSim against IBM's MERCURY simulator. In all of those early tests, BigSim was used in its network traffic-pattern mode. In that mode, instead of performing a trace-driven simulation as usual, BigSim can reproduce one of various pre-programmed patterns representing how packets are sent over the network. Those simulations can include any number of nodes or supernodes, and patterns were selected such that they could be also 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 were 0.6% to 1.1% greater than 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 KByte message to all other cores. The results between the two simulators only differed by 0.5%.

5.2 Validation on a Power 775 Drawer

During recent months, our group has periodically had remote access, for brief periods, to a prototype of a Power 775 drawer installed at IBM. This prototype drawer contains eight nodes, with four POWER7 chips each (256 cores total), and a development version the 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 minimally validating some of the BigSim components.

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 KBytes. 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 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 for that operation. Hence, BigSim's prediction was within nearly 10% of what we observed on the actual drawer.

A major advantage of using BigSim is the level of information that one can gather from a particular simulation. BigSim can produce detailed traces of link utilization (for an example illustrating this, see §9). It can also generate detailed event logs with computation and communication events during the simulation. These logs can be viewed in our *Projections* visualizer, as if the application had actually been run on the target machine. As an example, Figure 5.1 shows timelines of the above-mentioned `MPI_Alltoall` operation being sim-

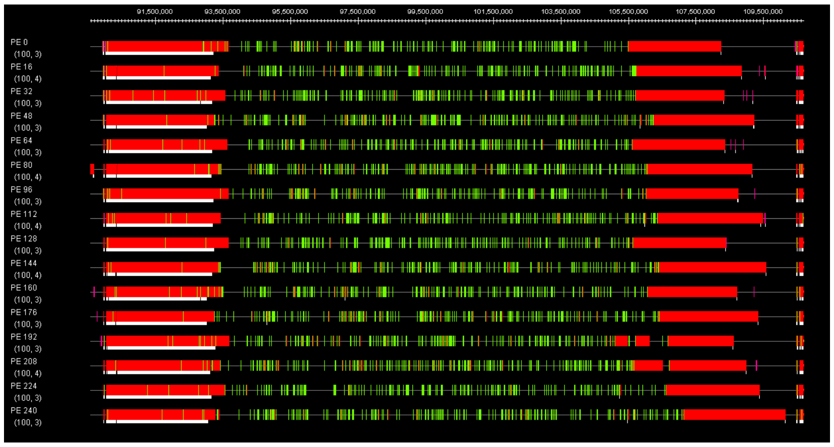


Figure 5.1: Detailed view of simulated `MPI_Alltoall` in a Power 775 drawer

ulated for a Power 775 drawer (only 16 of the 256 simulated cores are presented). The red boxes on the left correspond to the processor activity to send data in the all-to-all, and the red boxes on the right correspond to the respective receives. The middle region corresponds to the period of network activity. This kind of view can greatly help application developers understand what might be causing a bottleneck in their applications. Similarly, it may provide insights about how to change an algorithm in a certain phase to improve overall application performance.

6 Topology Aware Mapping

The PERCS topology provides a hierarchical two-level fully connected network topology. Each node is connected to every other node within the same supernode by a direct (LL or LR) link. Every supernode is connected to every other supernode by a D link. Two things make this topology interesting – 1. Link bandwidths on LL, LR and D links are different, so mapping and routing can significantly impact congestion and application performance also in some cases, and 2. There is only one D link connecting a pair of supernodes and hence, all traffic between a pair goes through a single D link in case of direct routing.

The PERCS network leads to various design choices for job scheduling, routing and mapping of tasks on to physical processors. Using simulations we can answer questions such as:

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

In this chapter, we demonstrate the utility of BigSim in making some of these design choices before the machine is installed. We use 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 machine. Each MPI task holds a data array of $256 \times 256 \times 256$ doubles and sends ghost layers to six neighbors for the jacobi exchange. Hence the size of each message 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 attempt to block MPI tasks into 3D cuboids that can nicely map onto the nodes and drawers of the PERCS 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 the node and drawer, tasks are also blocked on to supernodes with block dimensions being $16 \times 8 \times 8$.

Figure 6.1 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 that had a a certain range of bytes passing through them. It can be seen that the intelligent mappings for 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 links (as fewer bins on the right have any links).

Figure 6.2 shows the time spent in communication and overall execution of one iteration of 3D Stencil with different mappings. The communication reduces by 80% using an intelligent mapping and the overall time per iterations 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 to decide on the best mapping to be used for actual runs.

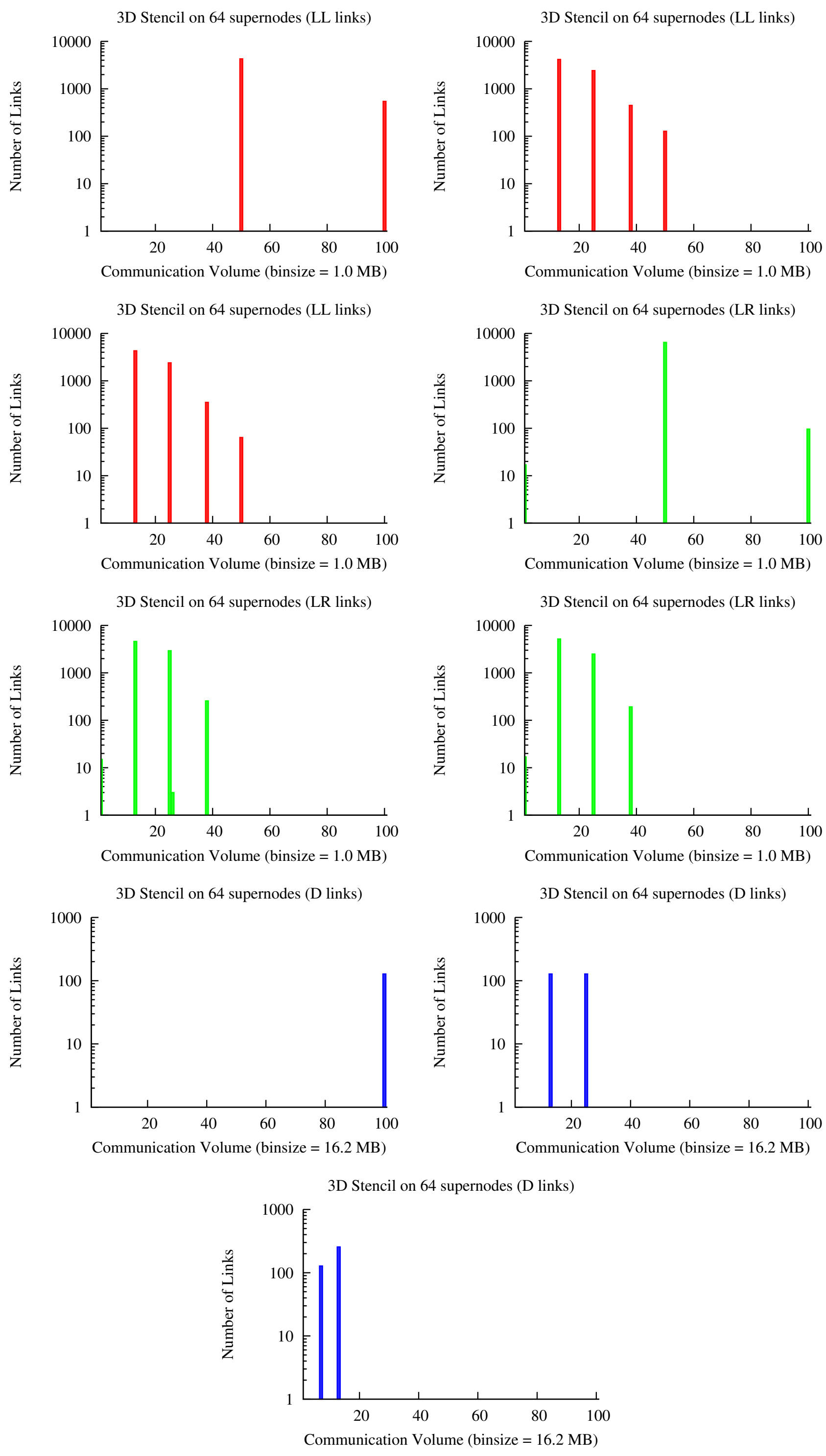


Figure 6.1: Link utilization for different mappings of a 3D Stencil on to 64 supernodes of the PERCS system. Each column represents utilization of the LL, LR and D links for a different mapping.

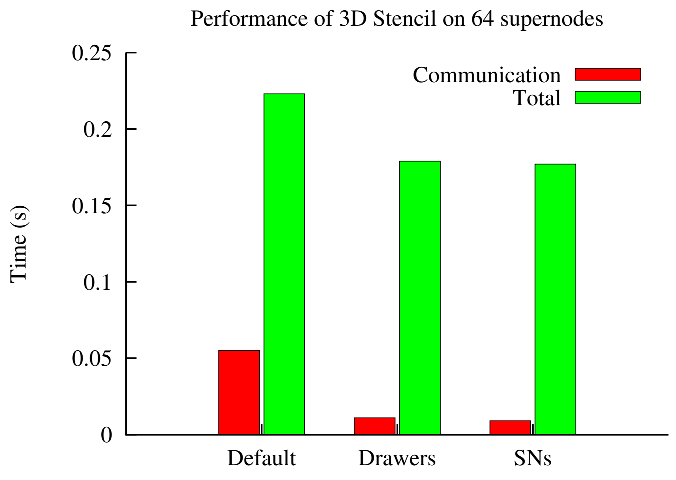


Figure 6.2: Communication and overall execution time (in seconds) of 3D Stencil on 64 supernodes

7 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 running on the compute nodes. However, current supercomputers that made the same design choice (such as Jaguar and Ranger) demonstrate significant OS jitter on their compute nodes. OS jitter can affect certain categories of parallel applications – those that are fine-grained and also those that have long critical paths.

Using BigSim, we can study the impact of OS jitter on applications that will run on large-scale supercomputers like those with the PERCS network. Different kinds of noise can be introduced in BigSim simulations:

- Noise traces can be collected from an existing POWER7 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.

7.1 Noise Simulation Support in BigSim

BigSim has comprehensive support for system noise to realistically simulate the effects on applications. It has two noise simulation modes to support different "what-if" analysis studies. The first mode uses previously captured noise from an existing system, and the second mode uses synthetic noise patterns, characterized by their periodicity and amplitude. Although this second mode only considers periodic kinds of noise, this is the common case for many types of noise sources, such as OS daemons and interrupts.

The captured-noise mode takes separate files with noise traces for each core of a node, because noise effects are different on distinct cores. This might be due to different OS daemons running on different cores or to architectural features. Each file contains all the perturbations that happened during the noise capture on an existing core, in the form of initial timestamp of the perturbation and its duration. This format is compatible with the output of many noise-capture programs, such as Netgauge [19]. A collection of captured-noise files, corresponding to all the cores of the node, is then created. During simulation, this collection is replicated on different nodes with different random offsets. It is also replicated in time if the application's simulated execution duration is longer that the noise capture duration.

BigSim can also use noise patterns. It reads separate pattern files for each core of a node. Each file can contain many noise patterns to be combined. A pattern is specified by its periodicity, amplitude and offset (phase). The simulator then considers the start time and duration of each Sequential Execution Block (SEB) and the various perturbations produced by the patterns in the underlying core. If a perturbation has occurred during the execution of an SEB, BigSim will extend the SEB's duration by an amount equal to the amplitude of that perturbation. The send time of each message in the SEB is adjusted accordingly if a perturbation has happened before the message is sent. Using these features, an application developer can "inject" noise, and examine its effects directly, without carrying out sophisticated analysis requiring detailed knowledge of the application's structure.

7.2 Examples of Noise Studies

Several noise studies can be done with this tool to assess the impact of noise on applications. The applications that we used for the studies are NAMD [20], which is a scalable molecular dynamics code, and MILC (MIMD Lattice Computation) [21], a quantum chromodynamics program. We also used two versions of a synthetic microbenchmark (K-Neighbor) that iteratively executes a nearest-neighbor communication followed by some amount of sequential computation;

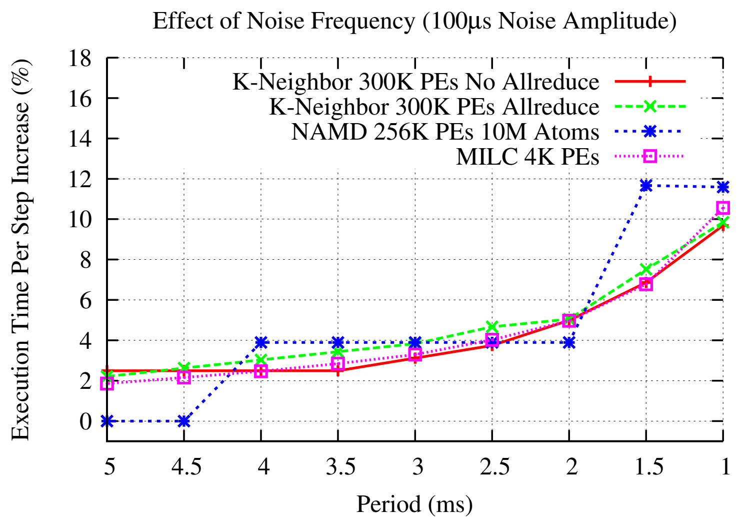


Figure 7.1: Effect of increase in frequency of noise patterns on application performance

one version of K-Neighbor contained an Allreduce at the end of each iteration, and the other version did not have the Allreduce. The goal was to illustrate the communication effect under the presence of noise.

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 timesteps 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 4×4×3×6 on each core; a target configuration of 4K processors was used. The baseline for MILC is 491.9 ms for the execution simulated. K-Neighbor was calibrated to have 1 ms of computation per step, and each processor communicated with eight neighbors. For the version with Allreduce, 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 in that scale was done in tens of minutes to a couple of hours using BigSim on just a single node, which shows the efficiency of this tool.

Figure 7.1 shows how increasing the frequency of a noise pattern can increase its effect on applications. With a study like this, system designers can gain insight about the frequency where applications get affected by noise of a certain duration (such as an OS daemon), and they can try to take measures to avoid those effects in the future system. Meanwhile, application developers can characterize and compare their applications' noise sensitivity and improve them if they are not acceptable. As one can see in Figure 7.1, with the decrease of the noise period (i.e. increase in noise frequency), the overhead on applications 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 of Figure 7.1) 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 K-Neighbor, with naive intuition one would expect the version with Allreduce to have at most 4% increase in 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 7.1 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 in the communication needed to implement the Allreduce, get perturbed by the noise too. This phenomenon is reflected towards the right end of Figure 7.1; 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 as affected by noise as much larger executions of other applications (e.g. the 300K processor run of NAMD). This shows that noise may be a concern for small-scale jobs of supercomputers as well.

Figure 7.2 shows a similar study that inspects different amplitudes of a fixed-frequency noise. In this figure, the NAMD's curve is flat up to some extent, probably due to noise absorption (elaborated in previous works [10]). Comparing Figure 7.2 to Figure 7.1, 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

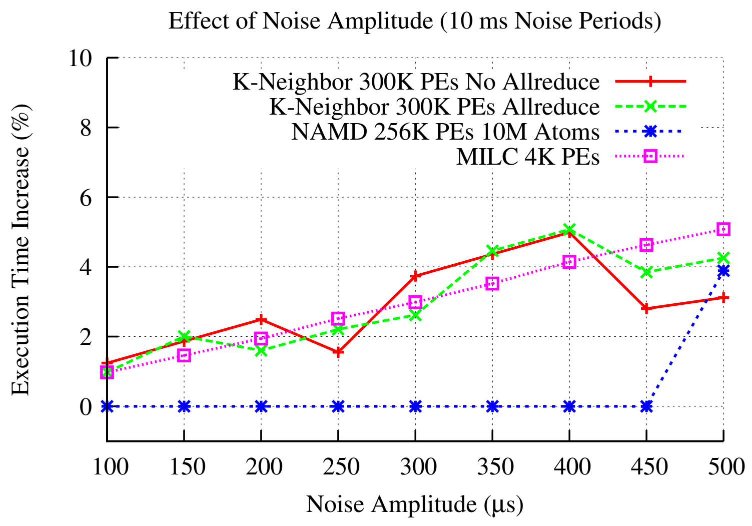


Figure 7.2: Effect of increase in amplitude of noise pattern on application performance

and is not in that category. This illustrates the usefulness of this technique for comparing design alternatives. Suppose there are two choices like a high-amplitude, less-frequent noise, and a low-amplitude but more frequent noise. For instance, these could correspond to different garbage collection strategies in runtime systems, such as full or incremental garbage collection which one wants to choose. This analysis can be done 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.

The effect of combining different noise patterns is another interesting analysis. One concern may be whether having two different noise patterns magnifies their effects on certain applications or not. Figure 7.3 shows the results of combining two noise patterns on our example applications. In this figure, the two noise patterns do not amplify or absorb each other, and the execution overhead is close to the sum of the individual overheads. Using these studies increases the level of certainty about applications' behavior on the future system.

One can also simulate the noise behavior of the future system by using early prototypes. For example, it is known that PERCS systems will have POWER7 processors, possibly running the Linux operating

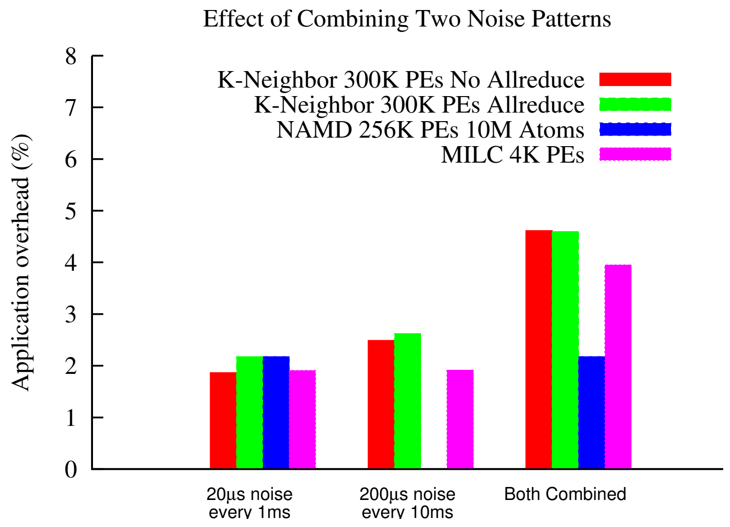


Figure 7.3: Effect of combining two noise patterns on application performance

system. One can capture noise on a POWER7 node and add it to simulation of different applications to measure the resulting overheads. Care must be taken to use noiseless emulation trace files for a more realistic modeling of this type. These traces can be collected by running BigSim emulations on noise-free machines (such as Blue Gene/P) or by using various BigSim tools to replace and normalize serial computations. Using this method, further tuning and optimizations can be done before the system comes online.

8 Effect of Network Parameters

One helpful use of a BigSim simulation is to analyze performance of applications under different values of network parameters. While designing networks, there are various tradeoffs of parameters that need to be addressed effectively. For example, a network designer may have the choice of increasing bandwidth at the cost of increasing latency, or improving latency while spending a relatively larger budget. Simulation can be used to determine what parameters are more critical to the applications under consideration. The designer can thus make better choices by having more knowledge about the sensitivity of applications to various network parameters.

To illustrate the use of BigSim in the study of application sensitivity to network parameters, we considered the same codes of the previous chapter. We conducted several BigSim simulations, varying the parameters of a simple network model used by the simulator. This network model assumes that communication time is a linear function of message size, characterized by latency and bandwidth values. The results of these experiments are in Figure 8.1, and reveal performance degradation in some applications when network latency increases. As can be seen, the K-Neighbor version with Allreduce is sensitive to latency, while the one without Allreduce remains unchanged. Thus, applications with collectives and global synchronizations are good candidates for this study. Also, comparing MILC and NAMD, we see that MILC is more sensitive to variations in latency, thus it should be studied more carefully in this regard. Note that although this analysis used a simple latency/bandwidth model and considered changes in the latency of the network as a whole, one could change the latency of specific parts of a network, such as PERCS different link types, for the purpose of tuning applications before the machine is deployed.

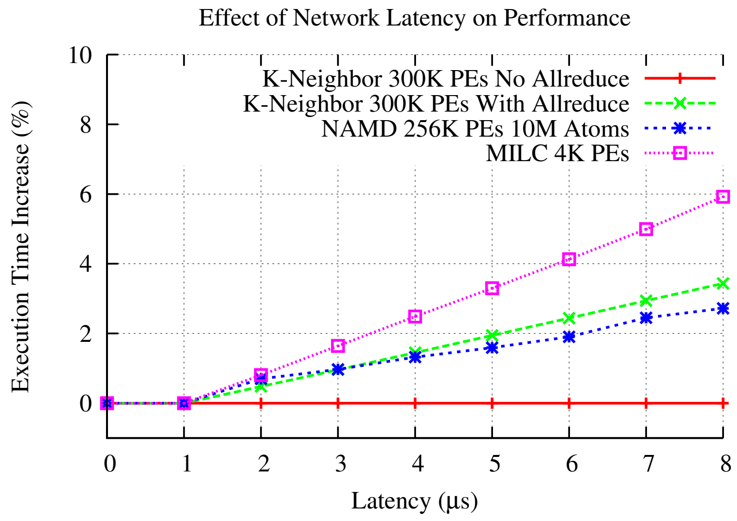


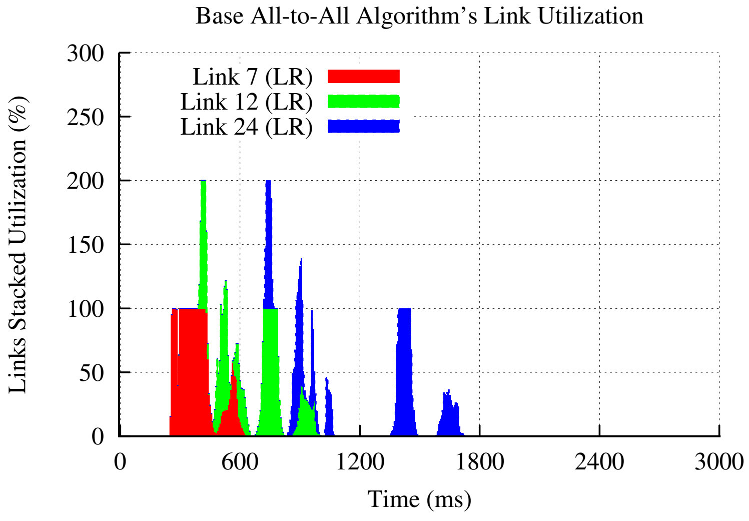
Figure 8.1: Effect of network latency on application performance

9 All-to-all Optimizations

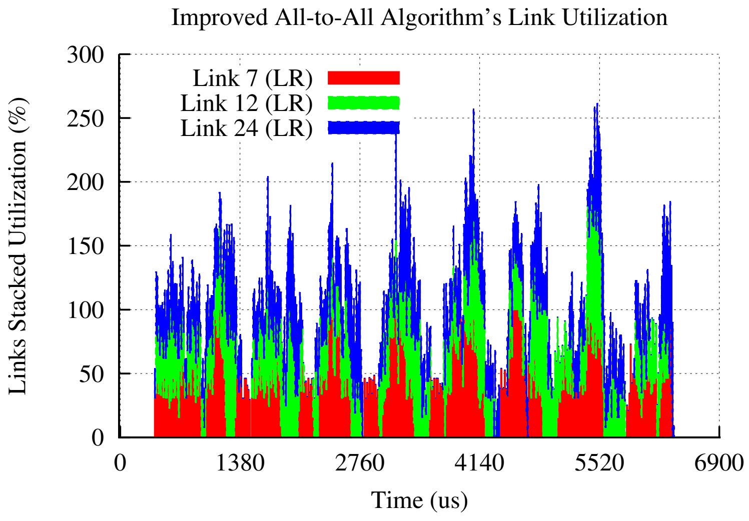
MPI_Alltoall is an important collective operation over a given set of tasks, with extensive usage in applications, such as FFT and matrix transpose. It is also among the most communication intensive collective operations performed in modern day parallel applications. As a result, it may suffer from scaling 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 size exchanged. We narrow our focus to MPI_AlltoAll for large data sizes. This chapter shows how simulation can provide insight about internal behavior and details of a system. Moreover, it shows how this insight may result in up to five-fold improvement of an important operation like all-to-all, before the future system arrives.

The Pairwise-Exchange algorithm [22] has been found to achieve better results on most machines. In each step of the algorithm, $\frac{P}{2}$ pairs of tasks perform a tightly coupled send-recv 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, current 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 system, 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 subcommunicator in a typical FFT implementation. Thus, the all-to-all only happens inside supernodes. Figure 9.1(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 links utilizations are stacked to show the overlap of usage (so it can go beyond 100%). One can observe that the utilization of these links is interleaved. A



(a) Base Pairwise-Exchange algorithm



(b) New all-to-all algorithm

Figure 9.1: Link utilization in base and new all-to-all

similar pattern is observed if all the links of a QCM are plotted. This observation shows that some bottleneck makes the links to be used in a shifted manner. However, given the contention-free, fully-connected nature of a supernode's network in PERCS, it is desirable to utilize all links stemming from a QCM simultaneously. This motivated us to consider a more advanced implementation of all-to-all, to enable simultaneous data transfers on all links.

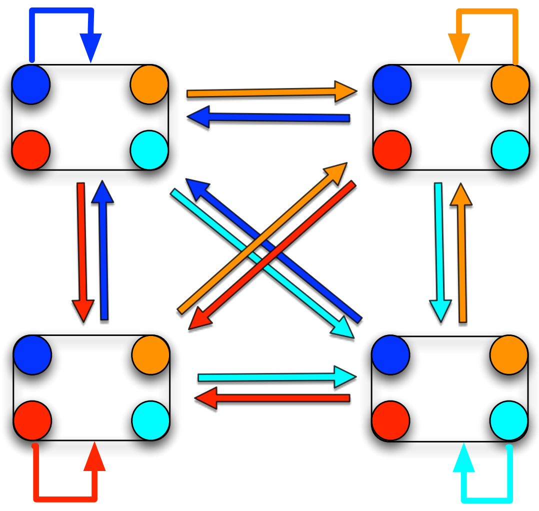


Figure 9.2: Sends in first phase of all-to-all

9.1 New Communication Scheme

We propose the following carefully-determined ordering in which the sends from a task should be performed, to (a) simultaneously utilize links stemming from a QCM of PERCS, and (b) avoid the undesired congestion and link contention. 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 in the $((j+i)\ mod\ n)^{th}$ node.

This scheme ensures that different cores of a node use different links, for better utilization. For example, consider the communication for phase 0, as shown in Figure 9.2. 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 node 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 PERCS system, for cases in which all the cores of a supernode execute MPI_Alltoall, this scheme will use all 31 links stemming from a QCM simultaneously.

Figure 9.1(b) presents a stacked chart for link utilization in the new scheme, for the same three links shown earlier in Figure 9.1(a). These results demonstrate that we have been able to overlap the usage of links for most of the simulation period. Thus, the sequential usage of different links no longer exists, and utilization improved significantly.

9.2 Performance Comparison

We consider an application with 1024 tasks running on one supernode of PERCS system. Let the amount of one-way data being exchanged between two cores be 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 $d = 1024 * m$ 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 MPI_Alltoall will be determined by the time taken to send this data on the slowest link. LR links, with bandwith of 5 GB/s, are the slowest links, and thus a lower bound for the time taken will be $\frac{d}{5}$ nanoseconds.

We present a comparison between our scheme and the default implementation of MPI_AlltoAll in Figure 9.3. The chart consists of simulation times for base 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 3× to 5× speedups for message sizes beyond 256 KB. More importantly, as the message size increases, the

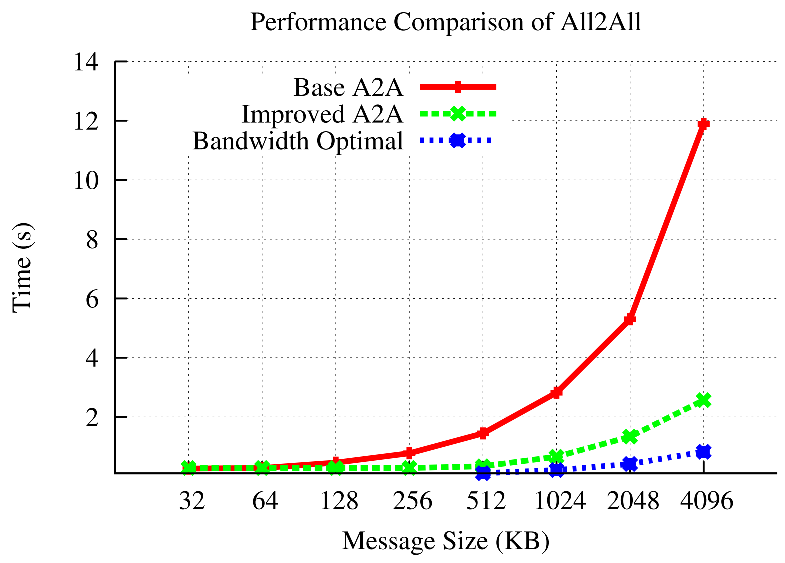


Figure 9.3: Performance comparison for all-to-all

performance in 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 near constant transmission time for messages of size below 256 KB.

10 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 is one of the most effective techniques 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 to specific parts of the application where performance is expected to be problematic on the upcoming system.

In this thesis, we have demonstrated the utility of the BigSim simulator to predict application performance on the upcoming PERCS systems. We showed various benefits that BigSim can provide to a future user of those systems, namely: (a) the effects of different mappings of tasks to processors across the machine: we showed that a 20% performance gain was achieved 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; (c) the assessment of application's sensitivity to network parameters, by rerunning the simulator with different values for a given parameter of interest; and (d) the performance gains that one can achieve by changing the algorithm employed for certain collective operations with large data sizes: we showed that a five-fold improvement is expected for an `MPI_Alltoall` in a PERCS supernode.

With a tool such as BigSim, users can concretely start preparing their applications before the system arrives. As we gain access to larger portions of the actual machine, we plan to continuously calibrate BigSim to ensure that it accurately models PERCS system.

A possible future work would be the development of other types of network models for BigSim, such as a model for Blue Gene/Q.

References

[1] E. Totoni, A. Bhatele, E. Bohm, N. Jain, C. Mendes, R. Mokos, G. Zheng, and L. Kale, "Simulation-based performance analysis and tuning for a two-level directly connected system," in *Proceedings of the 17th IEEE International Conference on Parallel and Distributed Systems*, December 2011.

[2] C. Huang, G. Zheng, S. Kumar, and L. V. Kalé, "Performance Evaluation of Adaptive MPI," in *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 2006*, March 2006.

[3] 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.

[4] T. L. Wilmarth, G. Zheng, E. J. Bohm, Y. Mehta, N. Choudhury, P. Jagadishprasad, and L. V. Kale, "Performance prediction using simulation of large-scale interconnection networks in pose," in *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation*, 2005, pp. 109–118.

[5] 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.

[6] 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.

[7] 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.

[8] 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. [Online]. Available: http://portal.acm.org/citation.cfm?id=1416222.1416248 pp. 21:1–21:10.

[9] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50 –58, Feb. 2002.

[10] 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.

[11] S. Agarwal, R. Garg, and N. Vishnoi, "The impact of noise on the scaling of collectives: A theoretical approach," in *High Performance Computing - HiPC 2005*, ser. Lecture Notes in Computer Science, D. Bader, M. Parashar, V. Sridhar, and V. Prasanna, Eds. Springer Berlin / Heidelberg, 2005, vol. 3769, pp. 280–289.

[12] 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.

[13] 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.

[14] G. Zheng, G. Kakulapati, and L. V. Kalé, "Bigsim: A parallel simulator for performance prediction of extremely large parallel machines," in *18th International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fe, New Mexico, April 2004, p. 78.

[15] 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.

[16] 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.

[17] 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.

[18] N. Choudhury, Y. Mehta, T. L. Wilmarth, E. J. Bohm, and L. V. . Kalé, "Scaling an optimistic parallel simulation of large-scale interconnection networks," in *Proceedings of the Winter Simulation Conference*, 2005.

[19] T. Hoefler, T. Mehlan, A. Lumsdaine, and W. Rehm, "Netgauge: A Network Performance Measurement Framework," in *Proceedings of High Performance Computing and Communications, HPCC'07*, vol. 4782. Springer, Sep. 2007, pp. 659–671.

[20] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten, "Scalable molecular dynamics with NAMD," *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.

[21] 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.

[22] 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.