Understanding application performance via micro-benchmarks on three large supercomputers: Intrepid, Ranger and Jaguar

Abhinav Bhatelé, Lukasz Wesolowski, Eric Bohm, Edgar Solomonik and Laxmikant V. Kalé

Department of Computer Science
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
{bhatele, wesolwsk, ebohm, solomon2, kale}@illinois.edu

Abstract

Emergence of new parallel architectures presents new challenges for application developers. Supercomputers vary in processor speed, network topology, interconnect communication characteristics and memory subsystems. This paper presents a performance comparison of three of the fastest machines in the world: IBM's Blue Gene/P installation at ANL (Intrepid), the SUN-Infiniband cluster at TACC (Ranger) and Cray's XT4 installation at ORNL (Jaguar). Comparisons are based on three applications selected by NSF for the Track 1 proposal to benchmark the Blue Waters system: NAMD, MILC and a turbulence code, DNS. We present a comprehensive overview of the architectural details of each of these machines and a comparison of their basic performance parameters. Application performance is presented for multiple problem sizes and the relative performance on the selected machines is explained through micro-benchmarking results. We hope that insights from this work will be useful to managers making buying decisions for supercomputers and application users trying to decide on a machine to run on. Based on the performance analysis techniques used in the paper, we also suggest a step-by-step procedure for estimating the suitability of a given architecture for a highly parallel application.

Keywords: performance analysis, micro-benchmarking, scientific applications, supercomputers, architecture

1 Introduction

A new era is taking shape with the emergence of very large parallel machines. 2008 saw the installation of two large machines: Ranger, a Sun Constellation Linux Cluster at Texas Advanced Computing Center (TACC) and Intrepid, a Blue Gene/P installation at the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory (ANL.) According to the TACC website, the 62,976-core Ranger is the largest computing system in the world for open science research. Intrepid was declared the fastest supercomputer in the world for open science in the Top500 List* for June, 2008. This list also announced the first Pflop/s (peak performance) supercomputer in the world – IBM's Roadrunner at Los Alamos National Laboratory (LANL).† Jaguar (Cray XT5) at Oak Ridge National Laboratory (ORNL) also broke the Pflop/s barrier in the November, 2008 Top500 List§. A *sustained* Pflop/s performance machine, Blue Waters is currently being designed by IBM for the National Center for Supercomputing Applications (NCSA). With such immense computing power at our disposal, it becomes essential that the end users (application developers and scientists) learn how to utilize it efficiently and share their experiences with each other.

This paper compares three of the largest supercomputers in the world - ANL's Intrepid (IBM Blue Gene/P), TACC's Ranger (SUN-Infiniband cluster) and ORNL's Jaguar (Cray XT4.) These machines were ranked fifth, sixth and eighth respectively in the Top500 List released in November, 2008. The three machines are architecturally diverse and present unique challenges to scientists working to scale their applications to the whole machine. Although benchmarking results are available for many of these machines (Alam et al. 2008; Alam et al. 2007), this is the first time in published literature that they are being compared using an application suite and micro-benchmarks. Among the top 25 systems in the Top500 list, there are six Blue Gene machines and five Cray XT machines. Infiniband (used in Ranger) is used in 28.4% of the Top500 machines and the XT4 SeaStar interconnect is used in 3.2% of them. Therefore, the machines in question form a significant percentage of the Top500 list both in terms of best performance and number of entries in the list. We present a comprehensive overview of the architectural details of each of these machines and compare them.

Another important aspect of this work is the use of full-fledged applications to compare performance across these machines. As part of the Track 1 request for proposals, NSF identified three categories of applications to serve as acceptance tests for the Track 1 machine: molecular dynamics (specifically NAMD), Lattice QCD calculation and a turbulence code. This paper uses NAMD (developed at the University of Illinois), MILC (developed by the MIMD Lattice Computation collaboration) and DNS (developed jointly by Sandia National Laboratory and Los Alamos National Laboratory). Together, these applications span a wide range of characteristics. In their domains, they vary from molecular dynamics to turbulence. In their programming paradigms, they vary from C++ using Charm++ to Fortran or C using MPI. Comparisons across these applications will provide us with an enhanced understanding of factors affecting performance. It should be noted that we did not tune the applications for the purpose of benchmarking for this paper. Also, we did not use the specifications of problem sizes to be used for these applications as outlined by NSF.

To understand the observed performance of these applications in terms of machine characteristics, we use multiple micro-benchmarks. We have selected a diverse set of micro-benchmarks from multiple sources to quantify message latencies, available network bandwidth, system noise, computation and communication overlap and on-chip vs. off-chip bandwidth. Characteristics of the machines reflected in the micro-benchmark results throw some

*http://www.top500.org/lists/2008/06

†This machine is not analyzed in this paper because we did not have access to it and also because the machine is sufficiently different as to require a significant porting effort for all the applications used in the paper.

§http://www.top500.org/lists/2008/11

light on the results obtained for the application codes.

The analysis of application performance using architectural details and micro-benchmarks suggests a step-by-step procedure for estimating the suitability of a given architecture for a highly parallel application. This method can be useful in a variety of scenarios: 1. an application user when applying for running time can choose the machine depending on this analysis, 2. project managers can make buying decisions depending on the suitability of a particular architecture to the class of applications they usually run, and 3. application developers can analyze the performance of applications on different platforms in a systematic manner.

Similar performance comparison work has been done using benchmarks, applications and performance modeling for Blue Gene/L, Red Storm and Purple (Hoisie et al. 2006) and through multiple scientific applications for Blue Gene/L, XT3 and Power5 machines (Oliker et al. 2007). Compared to these studies, we also outline a systematic analysis technique based on our methodology which can be used to qualitatively predict and understand application performance on new machines. In addition, we describe application characteristics which govern performance on the analyzed platforms. As the results will show, different machines appear to be better for different applications and sometimes even for different instances of the same application.

2 Architectures

A thorough understanding of the architectures of the machines being compared in this paper will guide us in our analysis. Below, we outline the major architectural features of the three supercomputers and discuss their expected behavior. In the rest of the paper, we will refer to the IBM system at ANL as Intrepid or BG/P, to the Sun machine at TACC as Ranger, and to the Cray at ORNL as Jaguar or XT4.

Intrepid: Intrepid* at Argonne National Laboratory (ANL) is a 40-rack installation of IBM's Blue Gene/P supercomputer (IBM Blue Gene Team 2008). Each rack contains 1024 compute nodes. A node is a 4-way SMP consisting of four PowerPC450 cores running at 850 MHz for a peak performance of 3.4 Gflop/s per core. There is a total of 2 GB of memory per node. Every core has a 64-way set-associative 32 KB L1 instruction cache and a 64-way set-associative 32 KB L1 data cache, both with 32-byte line sizes. The second level (L2R and L2W) caches, one dedicated per core, are 2 KB in size and fully set-associative, with a line size of 128 bytes. Each node is also equipped with two bank interleaved, 8-way set-associative 8 MB L3 shared caches with a line size of 128 bytes. The L1, L2 and L3 caches have latencies of 4, 12 and 50 cycles, respectively (IBM System Blue Gene Solution 2008). In terms of absolute time, the latencies are 4.7 ns, 14.1 ns and 58.8 ns, respectively. Memory latency is 104 cycles, or 122 ns.

Each node is attached to five communication networks: (1) six links to a 3D Torus network providing peer-to-peer communication between the nodes, (2) a collective network for MPI collective communication, (3) a global barrier/interrupt network, (4) an Optical 10 Gigabit Ethernet network for machine control and outside connectivity and (5) a control

*http://www.alcf.anl.gov/support/usingALCF/usingsystem.php network for system boot, debug and monitoring of the nodes. A midplane of 512 nodes forms a 3D torus of dimensions $8 \times 8 \times 8$. Larger tori are formed from this basic unit. The peak unidirectional bandwidth on each torus link is 425 MB/s which gives a total of 5.1 GB/s shared between 4 cores of each node. The nodes can be used in three different modes: (1) VN mode, where one process runs on each core, (2) DUAL mode where two processes run per node and multiple threads can be fired per process and (3) SMP mode where one process runs per node and multiple threads can be fired per process.

Comparing Blue Gene/P (BG/P) with its earlier counterpart, Blue Gene/L (BG/L), cores are 15% faster on BG/P and have twice as much memory (256 MB vs. 512 MB). Link bandwidth has increased from 175 MB/s to 425 MB/s but is now shared by twice as many cores. So bandwidth available per core has increased only slightly (1.05 GB/s to 1.275 GB/s.) However, the addition of a DMA engine to each node, offloads the communication load from the processors and improves performance.

Jaguar: Jaguar[†] at Oak Ridge National Laboratory (ORNL) comprises 7,832 compute nodes. Each node contains a quad-core 2.1 GHz Barcelona AMD Opteron processor and 8 GB of memory. Each processing core has a peak performance of 8.4 Gflop/s. Barcelona Opterons have an integrated memory controller and a three level cache system, with 2×64 KB, 512 KB and 2 MB L1, L2 and L3 cache sizes, respectively. L1 and L2 caches are private, while the L3 cache is shared. The L1 cache is 2-way set-associative and has a latency of 3 cycles. The L2 cache is 16-way set-associative and its latency is about 15 cycles. The 32-way set associative L3 cache has a latency of about 25 cycles. In terms of absolute time, the L1, L2 and L3 cache latencies are 1.4 ns, 7.1 ns and 11.9ns, respectively. The memory latency is about 143 ns[§]. The nodes run Compute Node Linux (CNL), which features low system overhead.

Each node is connected through a HyperTransport (HT) link to a Cray Seastar router. The routers form a 3-dimensional mesh of dimensions $21 \times 16 \times 24$. It is a mesh in the X dimension and a torus in the Y and Z dimensions. The rate at which packets can be injected on the network is bound by the bandwidth of the HT link (6.4 GB/s.) Each link on the torus has a unidirectional link bandwidth of 3.8 GB/s which gives a total of 45.6 GB/s shared between four cores. Comparing an XT4 node with an older XT3 node, there are twice as many cores per node. However, the link bandwidth has not increased and the same, admittedly high bandwidth is now shared by 4 cores instead of 2. This can negatively impact performance for communication bound applications.

Ranger: Ranger[‡] at Texas Advanced Computing Center (TACC) consists of 3,936 nodes connected using Infiniband technology in a full CLOS fat-tree topology. The machine comprises 328 12-node compute chassis, each with a direct connection to the 2 top-level switches. The nodes are SunBlade x6420 quad-socket blades with AMD quad-core Barcelona Opteron processors. The processors run at 2.3 GHz, with a peak performance of 9.2 Gflop/s per core. Architecturally, the processors are the same as the ones used in Jaguar, so the cache sizes and latencies are the same. Each node has 32 GB of DDR2/667 memory. The compute

[†] http://nccs.gov/computing-resources/jaguar/

[§] http://www.tacc.utexas.edu/services/training/materials/parallel_programming/T06_HW-2.pdf

[‡] http://www.tacc.utexas.edu/services/userguides/ranger/

	Intrepid	Ranger	Jaguar
Location, Year	ANL, 2007	TACC, 2008	ORNL, 2008
No. of Nodes (Cores per Node)	40,960 (4)	3,936 (16)	7,832 (4)
CPU Type (Clock Speed, MHz)	PowerPC 450 (850)	Opteron (2300)	Opteron (2100)
Peak Tflop/s (Gflop/s per Core)	557 (3.4)	579 (9.2)	260 (8.4)
Memory per Node (per Core), GB	2 (0.5)	32 (2)	8 (2)
Memory BW per Node (per Core), GB/s	13.6 (3.4)	21.3 (1.3)	10.6 (2.65)
Type of Network (Topology)	Custom (3D Torus)	Infiniband (full-CLOS)	SeaStar2 (3D Mesh)
Link Bandwidth GB/s	0.425	1	3.8
L1, L2, L3 Cache Sizes KB, KB, MB	64, 2, 8	128, 512, 2	128, 512, 2

Table 1: Specifications of the parallel machines used for the runs

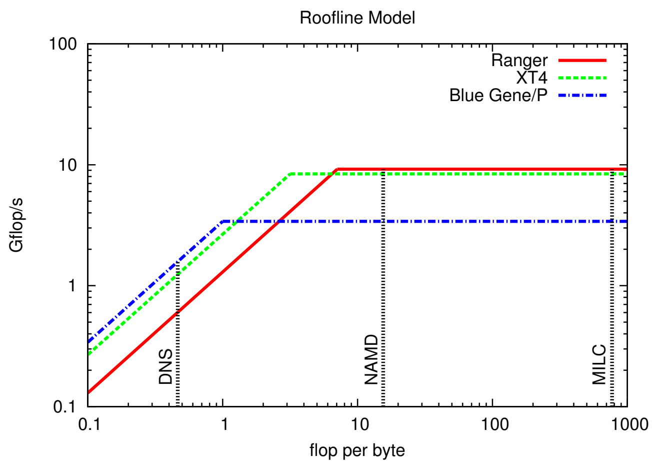

Figure 1: Roofline model

nodes run a Linux operating system. The peak peer-to-peer bandwidth on the network is 1 GB/s.

2.1 Comparison Across Machines

Table 1 presents the basic specifications for these systems. Based on these data, we can see that Ranger has the highest floating point performance per core (at 9.2 Gflop/s per core), followed by Jaguar (8.4 Gflop/s per core). These values are contrasted by Intrepid's 3.4 Gflop/s per core. This suggests that computation-bound applications will run faster on Jaguar and Ranger compared to Intrepid when the *same number of cores* is used on each machine.

Single core performance is also affected by the memory subsystem if the application is memory-bound. BG/P has the highest memory bandwidth to the DRAM per core (3.4 GB/s) followed by Jaguar (2.65 GB/s) and then Ranger (1.3 GB/s, see Table 1.) Plotting the attainable flop/s performance on a single core for a given value of arithmetic intensity (ratio of floating point operations computed per byte of memory bandwidth used), we get the roofline model (Williams et al. 2009) presented in Figure 1. This model gives a bound on what flop/s performance an application can achieve on a specific architecture given the application's theoretical flop per byte ratio. If the ratio is small the performance is likely to

System	Comm. Bandwidth per Core (MB/s)	Comm. Bandwidth per flop (bytes)	Mflop/s per W
Blue Gene/P	1,275	0.375	357
Ranger	1,000	0.109	217
XT4	11,400	1.357	130

Table 2: Comparison of network bandwidth and flop ratings for the parallel machines

be bound by the bandwidth (upward-sloped portion of the line). On the other hand, if the ratio is large enough the performance will be bound by the machine's flop/s performance (horizontal portion of the line). We can compare the shift between these two bounds on different machines. The plot suggests that memory-bound applications would likely achieve a larger portion of the machine's flop/s performance on BG/P - if the application can do a little more than one floating point computation for each byte loaded from DRAM, it can achieve close to the peak Gflop/s for each core. On the other hand, applications that can obtain significant reuse of data loaded from memory might achieve highest performance on Ranger despite its comparatively low bandwidth per core.

We mapped the three applications onto the roofline plot by calculating the bandwidth utilization to memory for the three applications. This was done using the PerfSuite toolkit (Kufrin 2005) on the NCSA Abe cluster. The applications were run on small problem sizes using just a single core of the machine. The flop per byte ratios of the three applications span a wide range in the plot. The performance of DNS is likely to be memory bandwidth bound on all three machines. On the other hand, both NAMD and MILC have high flop per byte ratios and are unlikely to experience a significant memory bottleneck. In fact, MILC runs almost completely in the L1 cache for the small input that was used. The flop per byte ratios should remain similar for weak scaling runs, although with strong scaling, the ratios will change for the three applications. Nevertheless, the plot allows us to roughly characterize the applications as being either memory or computation bound.

Communication-heavy applications are typically limited by the network bandwidth and latency characteristics of the machine even more than by the peak floating point performance. We can compare the network bandwidth per core on these machines. Intrepid, with its 3D Torus topology where each node is connected to two neighbors in each dimension through a dedicated link, has a bandwidth of 1.275 GB/s per core. The peer-to-peer bandwidth available on Ranger is 1 GB/s. The actual bandwidth available per core when using all 16 cores per node might be lower. The bandwidth per core on Jaguar in comparison is 11.4 GB/s but the limiting factor on Jaguar would be the Hyper Transport link which gives 1.6 GB/s per core.

Dividing the network bandwidth per core by the peak floating point performance per core in flop/s gives us a useful metric indicating the amount of data that can be transferred per operation in an application running at peak performance. The values for BG/P, Ranger and XT4 are 0.375, 0.109 and 1.357 bytes per flop. For network bandwidth-bound applications, Jaguar would perform the best, followed by BG/P and then Ranger.

As processor count and complexity of supercomputers grows over time, power consumption is becoming an increasingly important factor relating to the cost of executing code on these systems. Intrepid has the lowest total power consumption of the three systems considered, at 1.26 MW. Ranger, in comparison, consumes 2.00 MW of power and Jaguar consumes 1.58 MW. We can estimate the power efficiency of these systems by dividing the peak floating point performance by the power draw. This shows Intrepid as the most power-efficient, yielding 442 Gflop/s per kW. Ranger, at 290 Gflop/s per kW, is 34% less power-efficient than Intrepid, while Jaguar, at 165 Gflop/s per kW, is 63% less power-efficient than the BG/P machine. Table 2 summarizes these results.

Another important factor when evaluating a supercomputer is the cost of the machine. Cost is typically considered in a relative sense, by dividing the purchase price of a system by its performance. Delivered performance per dollar would be a better metric for evaluation than the performance per core metric used in this paper. Unfortunately, cost evaluation is difficult due to shifting or unavailable prices and varying maintenance costs across machines. As a result, we decided not to incorporate cost data into this paper.

3 Micro-benchmarks

Micro-benchmarking results are useful to characterize various aspects of parallel machines. Not only do they reveal the actual communication and computation characteristics of the supercomputers as opposed to the vendor-advertised values, but they also help to explain performance results of real applications. We have carefully chosen a range of micro-benchmarks to assess the communication and computation characteristics of the three machines under consideration. Of the five benchmarks presented here, the first, third, and fourth were written by us and the others were obtained from other sources, as indicated in the text. We used the vendor-supplied MPI implementations on BG/P and XT4 and MVAPICH (version 1.0.1) on Ranger.

3.1 Message Latencies: Absence and Presence of Contention

Latency of messages directly impacts the performance of parallel programs. Message latencies can vary significantly between the no-contention and contention scenarios. Two benchmarks (Bhatele and Kale 2009) are used to quantify these scenarios. The first benchmark, `WOCON`, sends a message from a pre-decided master rank to every other rank one by one. The average time for the message transmission is recorded and plotted against varying message sizes. The second benchmark, `WICON`, groups the ranks into pairs and each pair sends multiple messages to its partner. The pairs are randomly chosen so that messages are sent all over the job partition, hence creating contention. Once again, average latencies are plotted against varying message sizes[§].

Figure 2 shows the results of these two benchmarks on BG/P, Ranger and XT4. In the left plot, we see that for small messages, BG/P has the lowest latencies followed by Ranger and then XT4. Around 4 KB, the situation inverts and XT4 has the lowest latencies followed by Ranger and then BG/P. This is consistent with the higher bandwidth of XT4. In many parallel applications, there are several messages in flight at the same time, sharing multiple links which leads to network contention. The plot on the right presents results for

[§] The benchmarks can be downloaded here: http://charm.cs.illinois.edu/~bhatele/phd/MPI_Contention_Suite.tar.gz

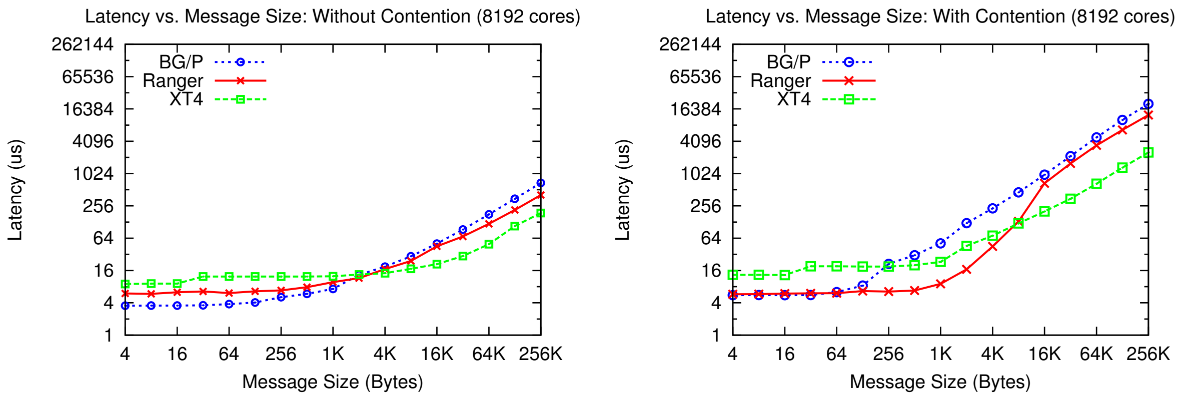

Figure 2: Latency Results from the `WOCON` and `WICON` Benchmarks

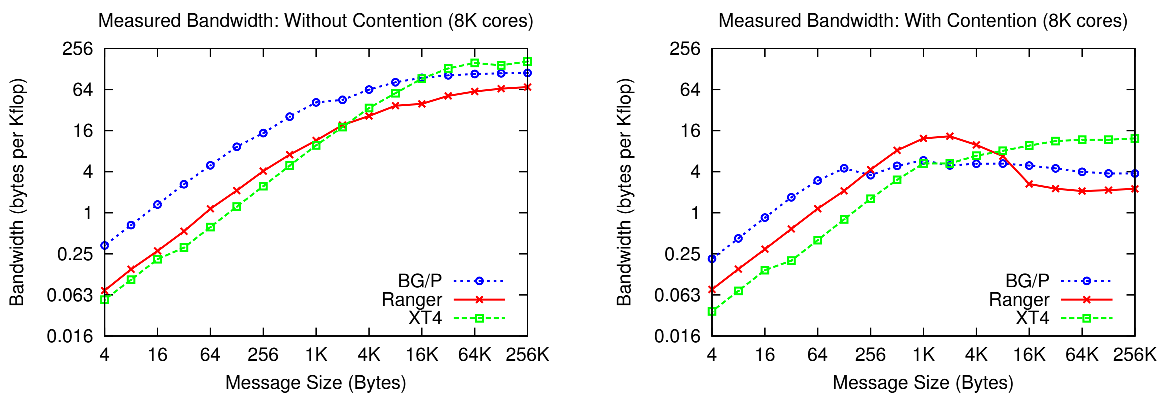

Figure 3: Bandwidth per flop/s comparisons using `WOCON` and `WICON` Benchmarks

the scenario where random contention is created. At message size of 1 MB, the latency for XT4 is 14 times that of the no-contention case, whereas it is 27 times for BG/P and 32 times for Ranger. This shows that for large messages (> 4 KB), XT4 is much more well behaved in presence of contention compared to BG/P and Ranger. On the other hand for very small messages (< 128 bytes), BG/P and Ranger perform better than XT4 as can be seen on the left side of the right plot.

Another metric to compare the communication characteristics of the interconnect fabric is bandwidth per flop/s, obtained by dividing the available bandwidth by the peak floating point execution rate of the processor. Figure 3 shows the bandwidth per flop/s metric plotted vs. message size for the WOCON and WICON benchmarks. Bandwidth per flop indicates the amount of data that can be transferred per floating point operation in an application executing at the peak floating point rate. Results show that although absolute bandwidth is higher on XT4 and Ranger than on BG/P for most message sizes, in terms of bandwidth per flop, BG/P is the best choice for small message sizes, and remains better than Ranger for large message sizes as well. Although XT4 has a poor bytes/flop ratio for small messages, it is the best for large message sizes, especially in the presence of contention. We do not have an explanation for the convex shape of Ranger's curve.

System	Max Ping Pong Latency μs	Random Order Ring Latency μs	Min Ping Pong Bandwidth MB/s	Natural Order Ring Bandwidth MB/s	Random Order Ring Bandwidth MB/s
Blue Gene/P	4.6	6.5	374.3	187.2	9.6
Ranger	7.3	35.5	504.2	201.4	18.2
XT4	11.6	35.2	1057.1	368.6	63.0

Table 3: Results from the HPCC Latency and Bandwidth Benchmark

3.2 Communication Bandwidth and Latency

This benchmark from the HPC Challenge suite measures communication bandwidth and latency for point-to-point and ring exchanges (Dongarra and Luszczek 2005). The benchmark consists of three separate experiments:

1. Ping pong exchange for a set of two randomly selected cores; when repeating the experiment, the ping pong exchanges are serialized so that no two exchanges happen simultaneously.

2. In a naturally ordered ring (built by the sequence of ranks in `MPI_COMM_WORLD`), each process sends in parallel a message to its neighbor; bandwidth and latency are measured per process.

3. For a set of 10 different and randomly ordered rings, each process sends a message to its neighbor. Ring creation is serialized, with each new ring created only after the completion of measurements for the previous ring.

The ping pong experiments were performed using several thousand pairs of cores randomly selected from a group of $8,192$ cores. The benchmark used 8 byte messages for measuring latency and $2,000,000$ byte messages for bandwidth measurements. MPI standard sends and receives were used for ping pong benchmarks. Ring patterns were performed in both directions on $8,192$ cores, where the best result of two implementations was used: (a) two calls to `MPI_Sendrecv`, (b) two non-blocking receives and two non-blocking sends (allowing duplex usage of network links). Results in Table 3 show BG/P having much lower communication latencies and bandwidth than the other two machines. XT4 has both the highest bandwidth and highest latency of the three systems. Results imply that BG/P would tend to be better for applications with small messages, while bandwidth-intensive applications would perform the best on XT4.

3.3 System Noise

Noise on a supercomputer can disrupt application performance but it is difficult to diagnose as the cause of performance degradation. To quantify hardware and operating system interference on the three parallel machines, we used a benchmark that runs a sequential computation for several thousand iterations and compares the running time for the computation across iterations and across cores. The sequential computation is a `for` loop that runs a specified number of times. There are two loops in the benchmark that control the execution of the sequential computation. The outer loop does 100 iterations and each iteration consists

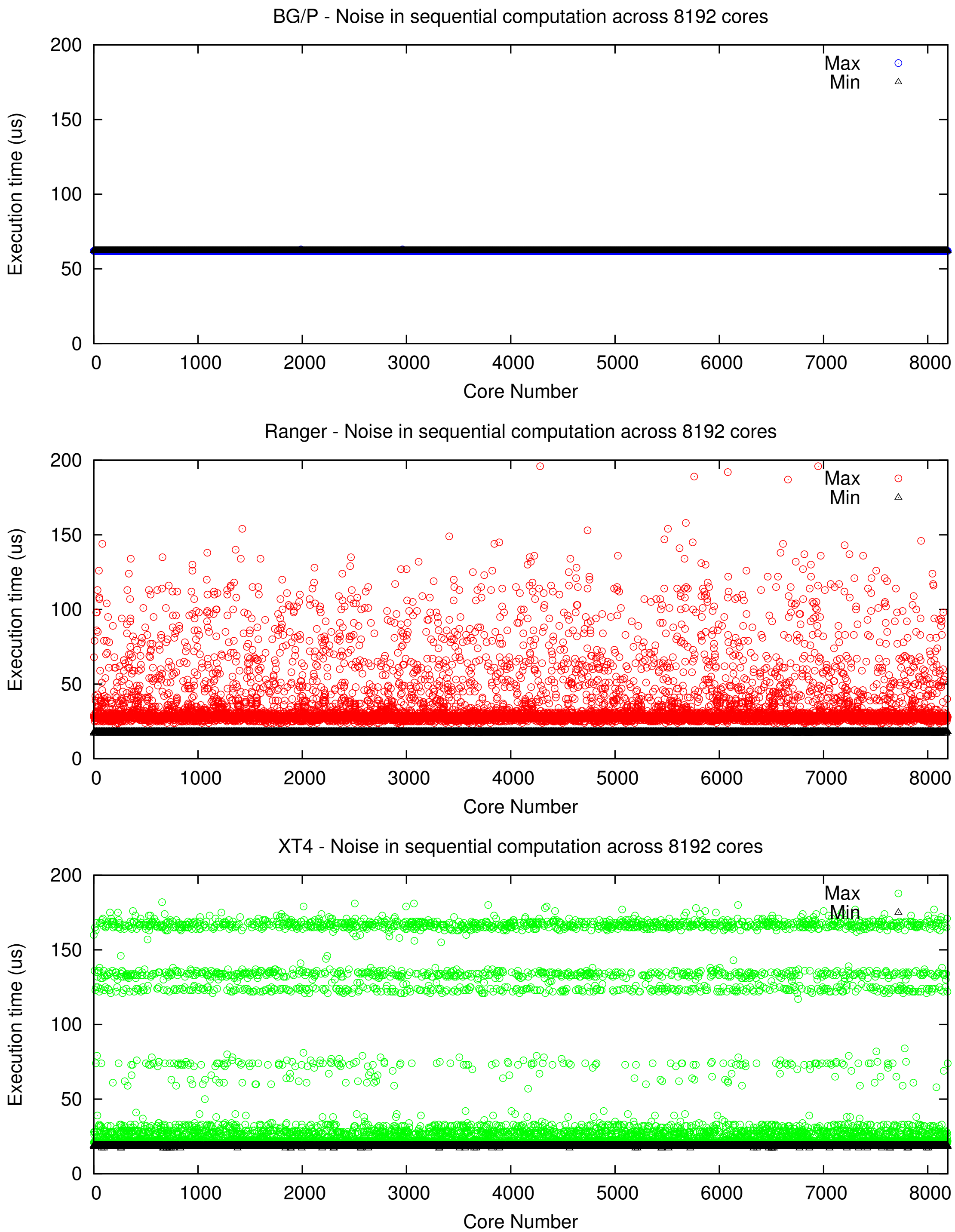

Figure 4: Plot showing system noise plotted against all ranks in a 8192-core run

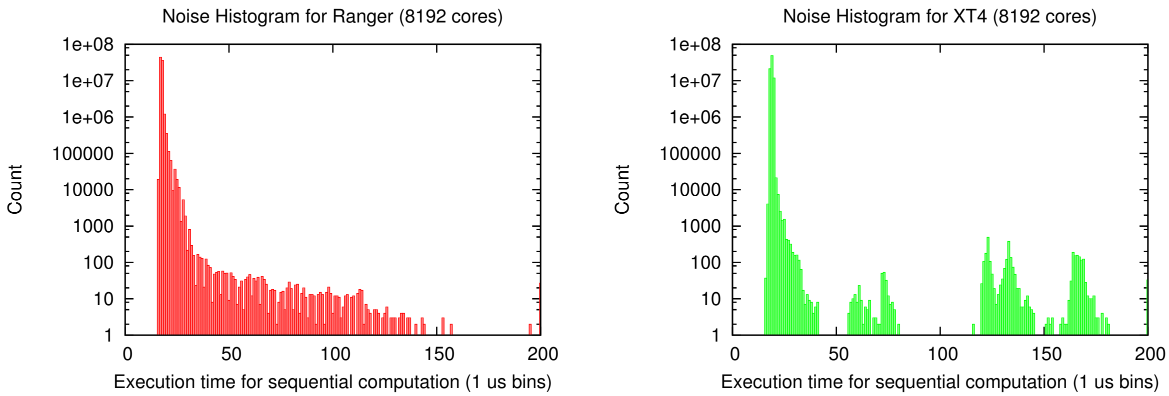

Figure 5: Histograms for the execution time of the sequential computation on Ranger and XT4

of running the inner loop and four `MPI_Allreduce` operations across all the cores. Each inner loop does 100 iterations of the sequential computation. So, each of the $8,192$ cores executes the sequential computation $10,000$ times in total. We record the minimum and maximum execution times for the sequential computation and the inner loop.

To see if specific cores were responsible for most of the noise, we plotted the minimum and maximum execution times for the $10,000$ executions of the sequential computation on each of the $8,192$ cores. The results shown in Figure 4 demonstrate that BG/P shows no noticeable noise behavior while both Ranger and XT4 suffer from significant noise problems. BG/P's behavior is expected because of its use of a micro-kernel instead of a full fledged operating system on the compute nodes. On Ranger and XT4, noise was observed on a large number of cores and did not seem to be limited to a certain region of the system. On XT4, the maximum execution times are clustered into regions which suggests that daemons taking a fixed amount of time might have interfered with the job. Based on these observations, we conclude that operating system interference is a serious problem on Ranger and XT4. Noise of this magnitude needs to be taken into account by application users, as it could cause poor performance, especially in applications using short time steps.

The plots in Figure 4 only show data points for up to 200 μs. The stretches are much longer in some cases on Ranger and XT4. We plotted histograms for the two machines to observe the phenomenon more closely (Figure 5). The histograms display the number of occurrences of the sequential computation in 1 μs bins. It is to be noted that the y-axis is logarithmic and more than 98% of the occurrences are clustered around the mean. The average times for Ranger and XT4 are 17.5 and 19 μs respectively. On Ranger, we see a spread from 16 to 157 μs with 27 occurrences in the last bin (> 200 μs). Stretches on Ranger for the computation are as large as 8 ms. Execution times on XT4 range from 16 μs to 174 μs. The largest stretch in the computation on XT4 is 778 μs and there are 33 occurrences of execution times being larger than 200 μs.

Figure 6 shows a plot of the "expected" minimum value and the maximum values for execution time of each iteration of the outer loop. Expected minimum value is the lowest observed execution time for any iteration of the outer loop. Maximum execution time is the most meaningful metric for noise in most cases, indicating the amount of time by which a lagging core may slow down the remaining cores in applications. Each outer loop of the noise benchmark executes the inner loop (100 executions of the sequential computation) and a few MPI collective operations. We included the collective operations to more closely model a typical iteration in a parallel application. As before, no noise is noticed on BG/P while Ranger and XT4 demonstrate noise across iterations of the outer loop. We do not know what was the specific source of the observed noise on Ranger and XT4.

Experiments (not shown in the paper) using a single core of XT4 did not show any noise whatsoever. However, when we used all four cores on a node, we started noticing some noise. This could be an indication that operating system interference is indeed responsible for the noise we observe in our large experiments, since any daemons which cause performance degradation could be executed on one of the idle cores in the single core experiment.

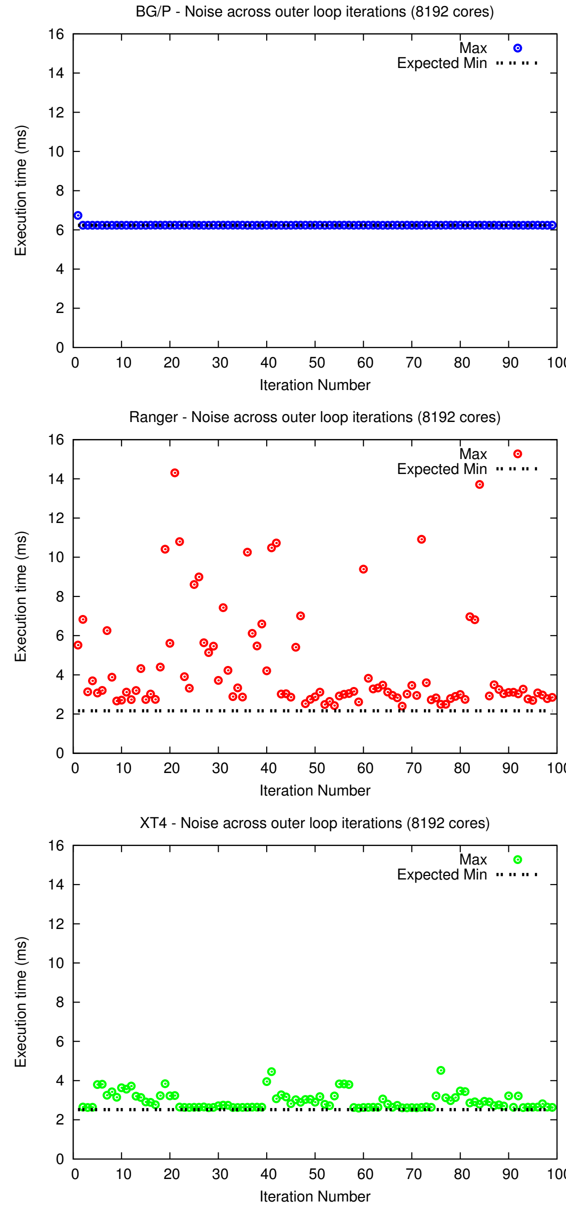

Figure 6: Plots showing system noise across iterations of the outer loop for the three machines

3.4 Overlap of Computation and Communication

Application performance is affected significantly by the capability of a machine to overlap communication with computation. We developed a simple benchmark which does the following: Two MPI processes on different nodes post an `MPI_Irecv`, then send a message to each other, do some amount of work and then post an `MPI_Wait` for the send from the other node. The total time from the posting of the `MPI_Irecv` to the completion of the `MPI_Wait` is recorded and plotted against the time for the computation. If communication is effectively overlapped with computation, we would expect to see a horizontal line while work is completely overlapped within the communication latency and then the total execution time should rise linearly with the work done.

Figures 7 and 8 (left) present the results obtained by running this benchmark on two nodes of each machine using one core per node. The x-axis is the amount of work in μs done between the message send and receive. The y-axis is the sum of the time for the message send and receive and the work interval. Three messages sizes were used: 16 bytes, 1 KB

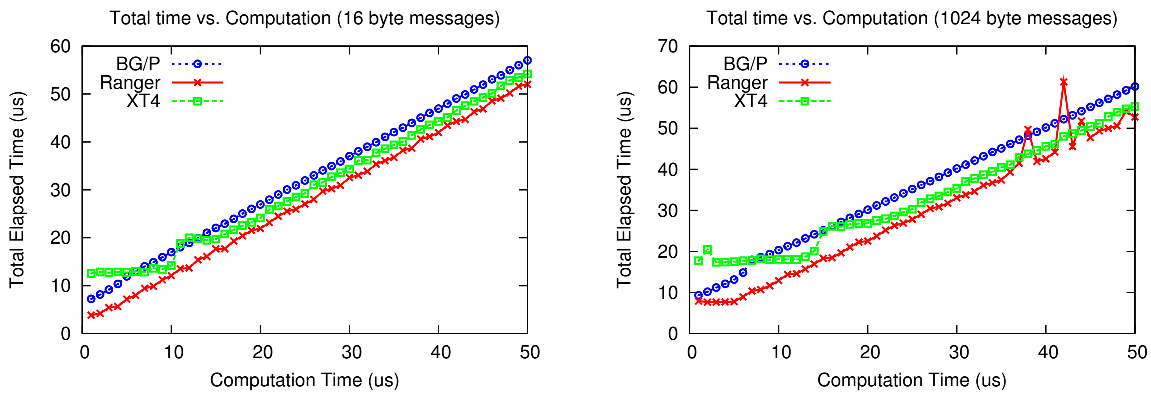

Figure 7: Results of the Communication-Computation Overlap benchmark for 16 and 1024 byte messages

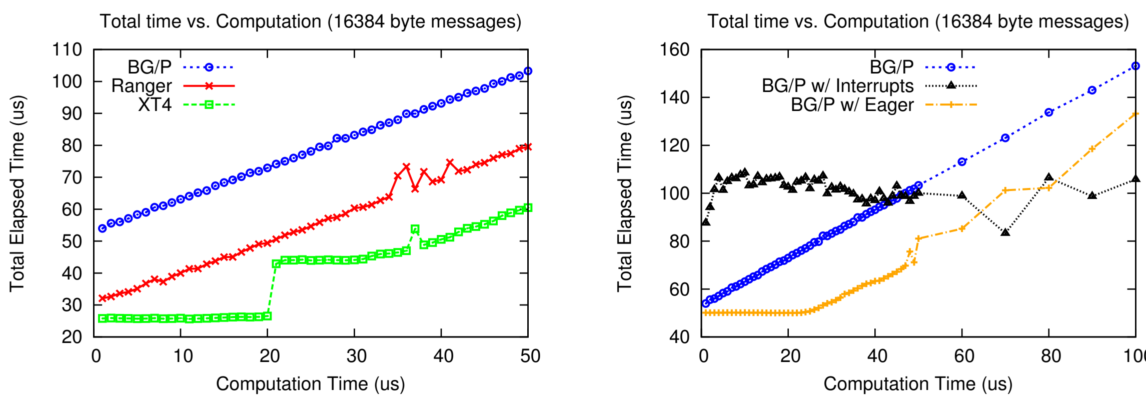

Figure 8: Results of the Communication-Computation Overlap benchmark for 16 KB messages (left) and various optimization possibilities on BG/P (right)

and 16 KB. For the 16 byte messages, only XT4 shows an overlap up to work intervals of 10 μs. This means, that of the 12 μs latency, 10 can used for doing useful computation on the processor. The Ranger curve may also have a flat portion at the left end, but since its latency is very low (4 μs), it is not seen. In other words: the ideal we would expect for Ranger is a flat line indicating overlap up to 4 μs, and it seems to be so. For 1 KB messages XT4 shows an overlap up to 14 μs and Ranger shows a small overlap up to 5 μs. For 16 KB messages, again only XT4 shows overlap.

It is interesting that BG/P, despite having DMA engines, shows no overlap. We reran the test on BG/P with the environment variable `DCMF_EAGER=1000000000`, which led to significant overlap for 16384 bytes messages (see the right plot in Figure 8). This is similar to what we see for XT4 in the left plot. Another option, `DCMF_INTERRUPTS=1` also increases the overlap but using this option increases the message latency to nearly 100 μs.

System	On-Chip Latency (Intranode) μs	Off-Chip Latency (Intranode) μs	Internode Latency μs	On-Chip Bandwidth (Intranode) MB/s	Off-Chip Bandwidth (Intranode) MB/s	Internode Bandwidth MB/s
BG/P	5.77	N/A	6.55	2287	N/A	374
Ranger	3.43	5.11, 5.79	10.25	1062	972, 864	902
XT4	1.70	N/A	11.45	1959	N/A	1252

Table 4: Communication Latency and Bandwidth for On-Chip and Off-Chip Communication

3.5 On-chip and Off-chip Communication Bandwidth and Latency

This benchmark[†] measures latency and bandwidth for a series of MPI ping pong programs with varying proximity of the processes performing the exchange. The cases explored are intra-socket communication for two cores in the same processor, inter-socket communication for two processors in the same node and inter-node communication. Standard `MPI_Send` and `MPI_Recv` calls are used for the ping pong exchange. Results are averaged over message sizes from 125 to 5000 bytes (in increments of 125 bytes), with 100 messages sent for each message size. The benchmark serves to quantify the benefit of placing two communicating processes on the same chip or node.

Table 4 summarizes our results for this benchmark. The on-chip latency on Cray XT4 is half of that on Ranger and one-third of that on BG/P. In terms of on-chip bandwidth, BG/P slightly edges out XT4, while Ranger has about half the bandwidth of XT4. In tests of internode latency, BG/P has the lowest latency, with a value only 14% higher than its maximum intranode latency. By comparison, latency on Ranger and XT4 suffers greatly when moving to internode communication. In tests of internode bandwidth, Ranger's bandwidth results are only slightly lower than intranode. XT4 has 36% lower bandwidth off-chip than on-chip, while BG/P has nearly 84% lower bandwidth off-chip compared to on-chip communication.

On Ranger, which contains four sockets (processor chips) per node, we also include results for intranode, but off-chip communication. The table contains two values for latency and bandwidth. The first corresponds to communication with a neighboring socket within the node, while the latter value represents communication between the two sockets in the node which are not directly connected through a Hyper Transport link.

4 Applications

This section describes the three applications: NAMD, MILC and DNS, and outlines the performance results obtained on the three machines. For each application, we use two input datasets of different sizes to understand scaling issues better. NAMD and DNS are run in strong scaling mode whereas MILC datasets are run in weak scaling mode as is typical for this application. Strong scaling refers to running a fixed size input on a varying number of processors. Weak scaling refers to a mode where the input size is increased to keep the total

[†]ftp://ftp.mcs.anl.gov/pub/mpi/mpi-test/mpich-test.tar.gz

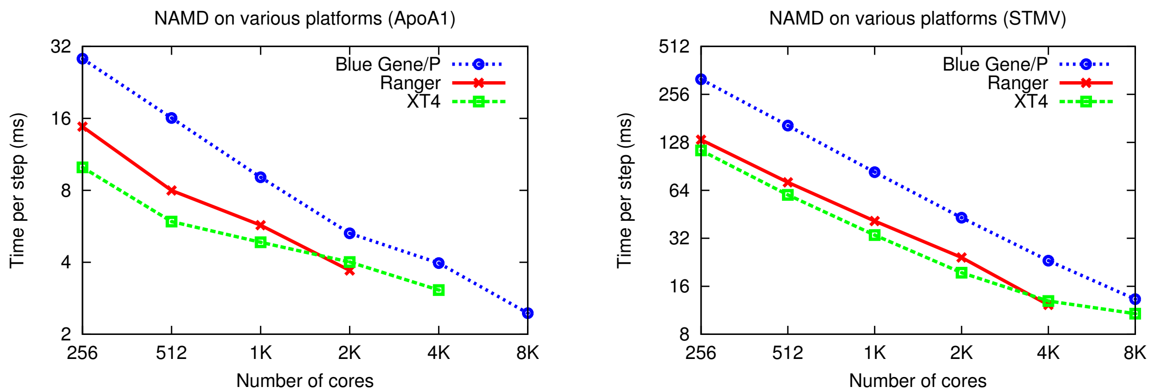

Figure 9: Performance (strong scaling) of NAMD on the three machines

work per processor constant as the number of processors is increased.

4.1 NAMD

NAMD is a highly scalable production Molecular Dynamics code (Phillips et al. 2005; Bhatele et al. 2008). It is used at most supercomputing centers in US and elsewhere and scales well to a large number of cores. NAMD provides us with an application which is used in the regime of strong scaling and hence stresses the machines. It runs in L2 cache on most machines and is latency-tolerant. NAMD is written in CHARM++ (Kalé and Krishnan 1993) and uses the CHARM++ runtime for parallelization and load balancing.

We use two simulation systems in this paper for benchmarking: one is a 92,222-atom system called Apolipoprotein-A1 (ApoA1) and the other is a 1-million atom simulation called STMV. Performance of NAMD on the three machines (Figure 9) is roughly consistent with the ratio of peak flop/s ratings of the three machines. As we scale the problems to a large number of cores the lines start converging, which can be attributed to the communication bottlenecks in that regime (Bhatele et al. 2008).

4.2 MILC

MILC stands for MIMD Lattice computation and is used for large scale numerical solutions to study quantum chromodynamics (Bernard et al. 2000). For benchmarking in this paper, we use the application *ks_imp_dyn* for simulating full QCD with dynamical Kogut-Susskind fermions. Two input datasets are used, one which is expected to fit in cache and the other to fit in memory. The smaller input is a grid with dimensions $4 \times 4 \times 4 \times 4$ on 1 core. The bigger input is a grid with dimensions of $8 \times 8 \times 8 \times 8$ on 1 core, 16 times bigger than the small input.

We run MILC in weak scaling mode, doubling the value along one of the dimensions when doubling the number of cores we run on. For weak scaling runs, a horizontal line on the plot indicates perfect scaling. Figure 10 presents MILC performance for the two inputs. For the small input, XT4 and Ranger have similar execution times, which are nearly half of

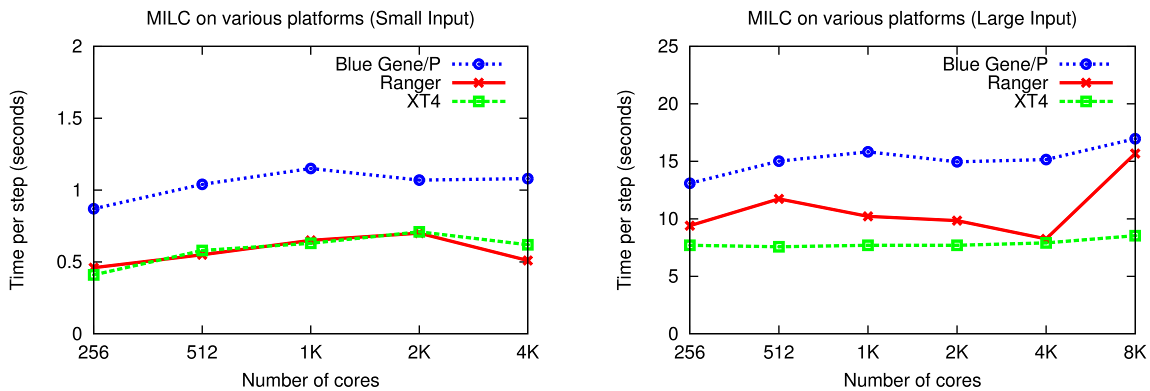

Figure 10: Performance (weak scaling) of MILC on the three machines

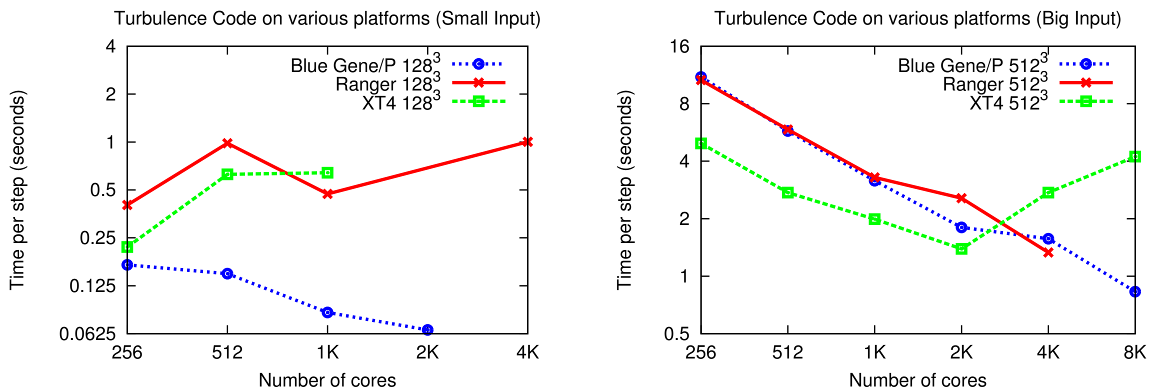

Figure 11: Performance (strong scaling) of DNS (Turbulence code) on the three machines

that on BG/P. For the bigger input, MILC is 33% faster on Ranger than on BG/P at 256 cores and the performance on XT4 is marginally better and more consistent than on Ranger.

4.3 DNS

DNS is a turbulence code developed at Sandia National Laboratory (Taylor et al. 2003). This code solves viscous fluid dynamics equations in a periodic rectangular domain (2D or 3D) with a pseudo-spectral method or 4th order finite differences and with the standard RK4 time stepping scheme. For benchmarking, we use the *purest* form of the code: Navier-Stokes with deterministic low wave number forcing. Results show strong scaling for two grid sizes: 128^3 and 512^3. We did not search the configuration space for the optimal parameters at each processor count.

Figure 11 shows strong scaling results of DNS for two input sizes. For the smaller input, BG/P has the best performance and scales reasonably well. On XT4 and Ranger, on the other hand, the application performs poorly and does not scale. For the larger input, the application shows scaling on all platforms, but Ranger performs roughly on the same level as BG/P despite having faster processors, and XT4, though it executes well on up to 2K cores, shows no scaling past that point.

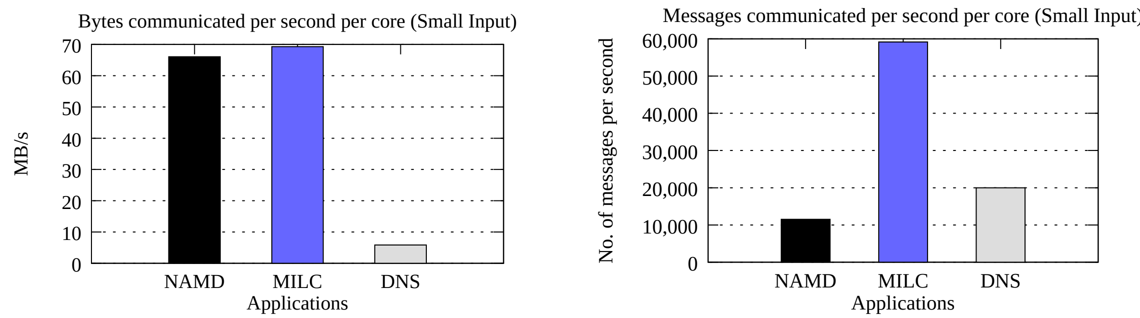

Figure 12: Communication characteristics of the three applications on XT4 using 512 cores (average values per core)

5 Performance Evaluation

In this section, we aim to explain the performance of the applications using their communication characteristics and results obtained from the presented micro-benchmarks. Communication profiles for the applications were obtained using the FPMPI interface on XT4. Figures 12 and 13 present the communication characteristics on Jaguar. The runs for Figure 12 were done with the smaller inputs and the numbers include all types of MPI messages including `MPI_Barrier`, `MPI_Wait`, `MPI_Waitall`, `MPI_Allreduce`, `MPI_Bcast`, `MPI_Isend` and `MPI_Irecv`. Runs for Figure 13 were done with both large and small inputs for the respective applications on 512 cores. These histograms only include `MPI_Isend`'s.

NAMD and MILC have a fairly large communication volume (measured in MB/s), nearly ten times that of DNS. Messages in NAMD are comparatively large in size. Hence, the number of messages sent per second in NAMD is about one-sixth of that in MILC and half of that in DNS (Figure 12). A large number of messages can lead to large overheads at the NIC or co-processor. To obtain a further breakdown, histograms showing the distribution of `MPI_Isend` message sizes were obtained for both the large and small inputs for each application. We will use these plots to explain the performance behavior of MILC and DNS.

A general trend which we noticed in the performance plots of all the applications is that BG/P performs as expected, achieving a high fraction of its peak performance. This can be attributed to some of the design features and characteristics of BG/P which govern performance in many cases: 1. Highest memory bandwidth per core (3.4 GB/s), 2. Biggest L3 cache (2 MB per core), 3. Smallest latencies for small-sized messages (4 to 1024 bytes), 4. No evidence of noise and 5. Well-implemented MPI collectives (see (Kumar et al. 2008)). Next, we compare the performance of Ranger and XT4 with that of BG/P for MILC and DNS. Performance of NAMD on these machines is as expected and hence, NAMD is not a part of the future discussion.

MILC: MILC shows unusual performance behavior (quite different from NAMD) which might throw some light on the performance characteristics of the machines (Figure 10). For the small input, XT4 and Ranger have similar execution times, which are nearly half of that on BG/P. For the bigger input, MILC is 33% faster on Ranger than on BG/P at 256 cores and the performance on XT4 is slightly better than on Ranger.

The small input stresses the communication fabric of the machine by sending a large

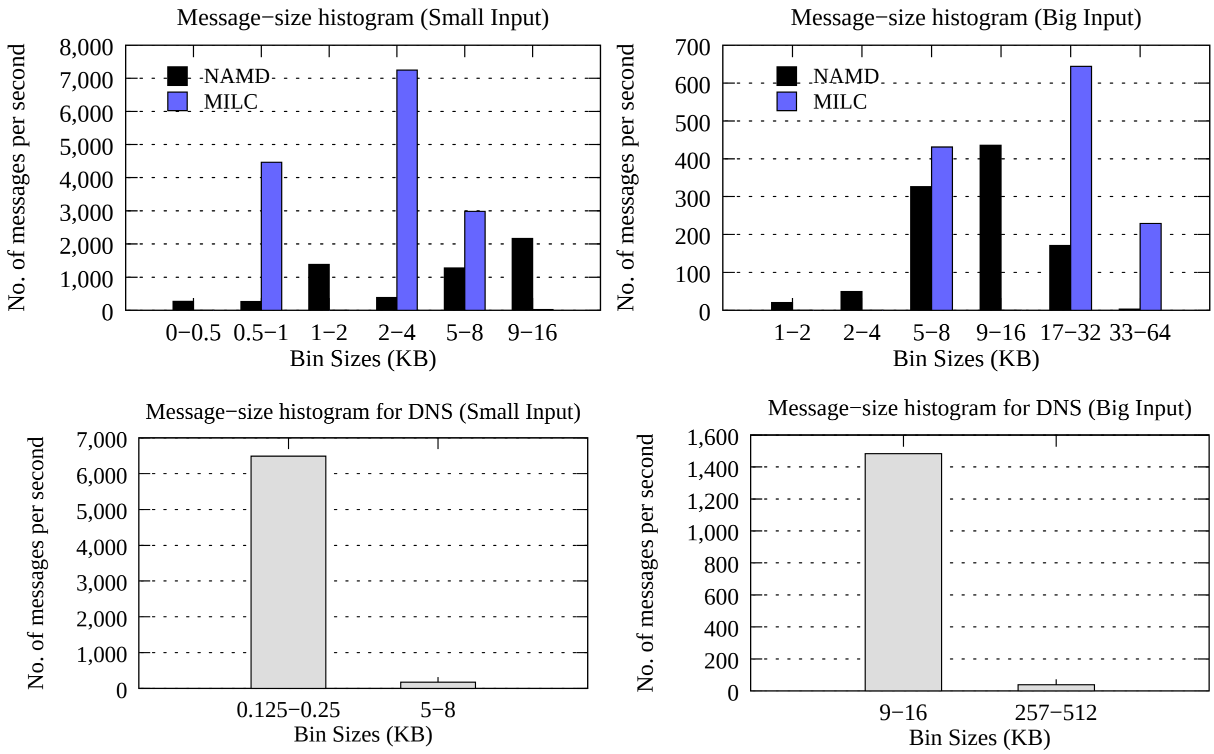

Figure 13: Message size histogram for the three applications on XT4 using 512 cores (average values per core)

number of messages (about 59,000 messages per second per core, see Figure 12 right). On XT4, this can lead to contention for the Hyper Transport link (shared between four cores). Since the messages range from 0.5-8 KB (see top left in Figure 13), there could be contention on the network, which would not be handled well by XT4 (refer to the WICON results in Figure 3). Performance for the small benchmark is primarily bound by small message latency and collective latency, which means that Ranger's faster CPUs cannot be utilized efficiently by its network under these conditions.

The large input stresses memory bandwidth in addition to the network. Ranger, which has the lowest memory bandwidth (see Figure 1), is thus able to perform only slightly better than BG/P while XT4, with reduced network contention, now outperforms the other two machines. Reducing the number of cores used per node for the runs eases the stress both on the memory subsystem and the network. Table 5 shows comparisons of runs on XT4 and Ranger using 4 vs. 2 cores and 16 vs. 8 cores per node respectively. On Ranger, using half the number of cores on each node improves performance by 32% and 40% for the small and large input respectively on 256 cores, although it involves leaving 256 cores idle. Greater improvement for the large input suggests that memory bandwidth is the major bottleneck for this application which prevents XT4 and Ranger from performing three times faster than BG/P (which is the ratio of their peak flop/s per core). However, network bandwidth is also a problem, as suggested by the improvement in performance of the small input runs. The performance of MILC for the large input size is a reflection of the three architectures' ability to handle bandwidth-bound applications.

Input	MILC (s/step) Small Input		MILC (s/step) Large Input		DNS (s/step) 128^3		DNS (s/step) 512^3	
#cores	**256**	**512**	**256**	**512**	**256**	**512**	**256**	**512**
XT4 4 cpn	0.41	0.58	7.70	7.57	0.219	0.626	4.96	2.74
XT4 2 cpn	0.35	0.43	5.25	5.68	115.46		121.70	
Ranger 16 cpn	0.46	0.55	13.09	15.02	0.402	0.983	10.63	5.86
Ranger 8 cpn	0.31	0.49	7.76	7.98	0.292	0.928	7.55	4.04

Table 5: Runs on XT4 and Ranger using fewer than all cores per node

DNS: Figure 11 shows strong scaling results of DNS for two input sizes. For the smaller input, BG/P has the best performance and scales reasonably well. On XT4 and Ranger, on the other hand, the application performs poorly and does not scale. For the larger input, the application shows scaling on all platforms, but Ranger performs roughly on the same level as BG/P despite having faster processors, and XT4, though it executes well on up to 2K cores, shows no scaling past that point.

DNS is a turbulence code and does a large number of small FFTs, which lead to all-to-all communication. It sends a large number of small messages (128 to 256 bytes) - approximately 6700 `MPI_Isend`'s per second per processor (see bottom left in Figure 13). Communication performance for small messages is determined by the overhead of message sending on the target machine. XT4 and Ranger have large overhead for small messages, as is apparent from the WOCON results in Figure 2. Secondly, a large number of `MPI_Waitall` calls (6700 calls per second per processor) indicates that the application could be susceptible to noise for the smaller dataset, as was demonstrated in Figure 6. For the large input, the number of messages sent per second drops by a factor of four and the size of the messages increases. As a result, the performance of both XT4 and Ranger is significantly better on the large input. The results also confirm our prior observation that applications running on BG/P seem to be less susceptible to latency and noise overheads.

Table 5 shows results similar to MILC for DNS using fewer cores per node on Ranger and XT4. On Ranger, there is a substantial benefit to using 8 instead of 16 cores per node for the 256-core runs for both the small and large input. Using fewer cores per node brings the performance closer to XT4. This can be attributed to higher memory bandwidth available per core for these runs. As is clear from the roofline plot 1, DNS has a very small flop/byte ratio and hence stresses the memory subsystem severely. Using fewer cores per node alleviates some of this stress. Leaving some of the cores idle on a node could also reduce OS noise, assuming the idle cores would be used for OS-related tasks, and also reduce network contention and communication volume resulting from the large number of messages. The benefit on Ranger is smaller when running on 512 cores. On XT4, using fewer cores per node produced performance three orders of magnitude worse than expected. We do not have an explanation for this anomaly.

Finally, Figure 14 shows the flop/s for NAMD, MILC and DNS on the three machines. The runs were done with the smaller inputs for the one core case (left plot) and with the larger inputs for the 512 core case (right plot). The flop counts for NAMD were obtained using the PerfSuite toolkit available on NCSA's Abe cluster. The numbers for MILC and DNS are

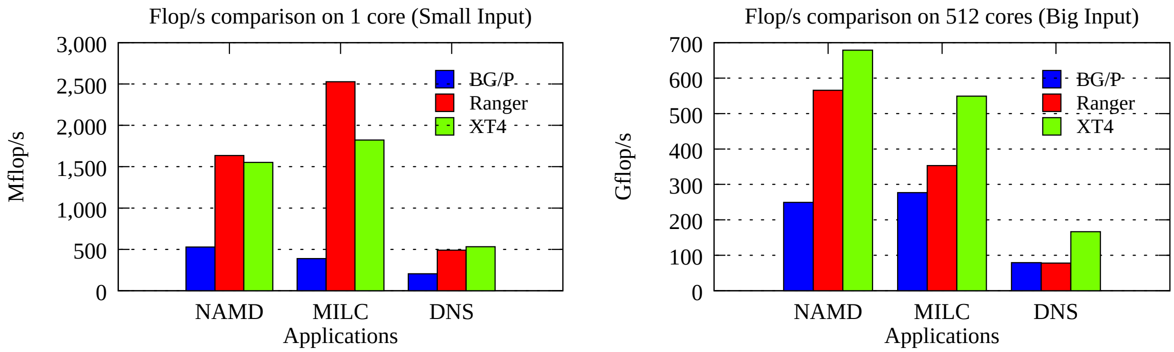

Figure 14: Flop/s comparisons of the three applications on BG/P, Ranger and XT4

estimates obtained from the outputs of actual runs. The one core performance for NAMD and DNS is as expected. In the case of MILC, Ranger's flop/s performance is 5 times that of BG/P. This gap appears inconsistent with the earlier results (Figure 10) where BG/P is only two times worse in performance on a larger number of cores. This change can be explained by the low network bandwidth per core and low memory bandwidth of Ranger compared to BG/P (see Figure 1), which start affecting performance on large runs. For further insight, we did MILC runs on the three machines from 1 to 256 cores which shows that performance on Ranger gradually degrades and becomes 7 times worse from 1 to 256 cores for weak scaling (Figure 15). There is a difference in the performance between the debug and batch queues of Ranger because of faster processors (2.6 GHz) and more importantly, because of lesser interconnect contention on the debug queue. Note that this is the only test for which we used Ranger's debug queue. For all other Ranger runs in the paper, we used the batch queue.

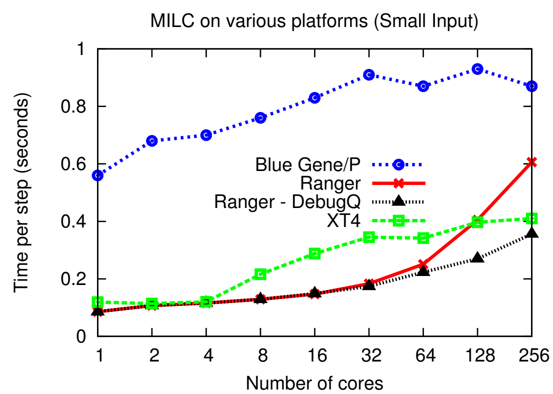

Figure 15: MILC Performance (weak scaling)

Looking at Figure 14, we notice a trend. Ranger performance is best on a single core (attributed to its high flop/s rating). XT4 gives the best performance on 512 cores (sometimes twice as that on Ranger, for MILC and DNS). Not only that, Ranger's performance drops quite close to BG/P. This can be attributed to the memory bandwidth and network contention issues in these two applications which are not handled well on Ranger. NAMD on the other hand runs in cache and is latency tolerant and hence is largely unaffected by these issues.

6 Systematic Analysis Technique

The analysis of application performance using architectural details and micro-benchmarks suggests a step-by-step procedure for estimating the suitability of a given architecture for a highly parallel application. This method can be useful in a variety of scenarios: 1. an application user when applying for running time can choose the machine depending on this analysis, 2. project managers can make buying decisions depending on the suitability of a particular architecture to the class of applications they usually run, and 3. application developers can analyze the performance of applications on different platforms in a systematic manner. We now outline the steps involved in this analysis technique.

Step 1: Obtain application and architecture information
We first need to identify various application characteristics which affect performance. It is useful to measure the flop per byte and communication requirements of the application. The architecture of the machines in question also needs to be carefully analyzed. It is important to take into account factors such as: 1. The peak flop/s rating 2. Memory bandwidth and cache sizes, and 3. Network bandwidth and latency.

Step 2: Check whether application is computation or memory-bandwidth bound
To ascertain if the application is computation bound or memory bandwidth bound, we can map the flop per byte ratio for the application onto the roofline model for the different machines in question. This calculation gives a rough idea of whether the application will be memory bandwidth or computation bound (if the value is in the slanted portion of the model, application will likely be memory bound). If the application is computation bound, we can choose a machine with high peak flop/s rating. If the program is memory bandwidth bound, we should try to estimate what portion of peak flop/s the application might achieve on the machines given the specific memory demands of the application.

Step 3: Determine communication characteristics
Based on the application specifications or a small parallel run on an available machine, one should be able to get a sense of whether the application is communication-bound. One way to do this is to build a histogram of message sizes, such as the one in Figure 13. A large number of messages over time, or frequent collective communication operations, may also indicate that the application will be hindered by network contention on the target machine. Based on the message size histogram and a rough estimate of whether contention will be a problem, we can use bandwidth and latency graphs on the target machine, such as were presented in Figures 2 and 3, to see which machine performs best for the range of message sizes used in the application. If the application does a significant amount of collective communication, an important factor to consider is the MPI implementation of collective operations on the machine. System noise is another factor to consider, especially for fine-grained applications with frequent barriers.

Step 4: Benchmark the application on the real machine
If access to the machine is available, we can obtain additional and more reliable information about the application performance and do a more concrete analysis. In this scenario (possibly useful to application developers), architecture or micro-benchmarking data can be used to explain the scaling performance of the application on that machine. The capability of an application to overlap communication with computation and the tendency to create network contention are important factors affecting communication performance, but they can only be

deduced through actual runs on a certain number of processors on the real machine. Grain size of the application is another important consideration; it can be used to estimate the effect of noise on application performance. Results in Figure 5 show noise of up to 8 ms on Ranger and up to 778 μs on XT4. For these systems, we believe that applications with short step sizes (on the order of tens of milliseconds or less), where the steps are separated by barriers, may suffer from system noise. If access to the machine is not available, then getting a better grasp of such issues often requires performance modeling using simulations (Zheng et al. 2004).

7 Conclusion

In this paper, we compared the performance of three diverse applications (NAMD, MILC and DNS) on three machines (Intrepid, Ranger and Jaguar) which represent the common architectures of the top supercomputers available for open science research. We analyzed the performance of those applications on both small and large problems and explained their performance variation across the platforms in the context of each application's characteristics, using both the vendor machine specifications and selected micro-benchmarks.

The micro-benchmarks chosen illustrate key differences in how each platform responds to various load conditions. The standard flop and bandwidth benchmarks are further illuminated by the WICON and WOCON benchmarks to provide insights into the performance problems encountered in the small input runs for the applications on Ranger and XT4. The noise benchmark demonstrates that significant performance variation occurs on both Ranger and Jaguar, despite negligible single core noise results on Jaguar. We cannot fully explain the inter node noise on XT4, though it seems likely that some of it is due to network contention with applications running at the same time that share links. This is due to the Cray scheduler not allocating dedicated partitions.

As expected, applications which are latency intolerant (such as DNS) scale best on the low latency Blue Gene architecture. However, those applications which are computation bound can achieve excellent performance on the Cray XT4 architecture. Ranger's memory bandwidth limitations restrict the performance for applications of this type. Although some of these performance issues can be ameliorated by running on fewer cores per node, system users will be charged for the entire node, so such users should prefer systems with higher bandwidth per core by design. Latency tolerant applications, such as NAMD or MILC in weak scaling, can make effective use of supercomputers based on commodity interconnects, like Ranger, but will experience their best performance for larger scale problems on Cray style architectures.

In summary, vendor specifications, benchmarking results and application characteristics can and should be combined to form a more complete picture of application performance to guide the expectations of the supercomputer user community. We hope that the systematic analysis technique presented in the previous section will contribute towards developing effective guidelines for application developers, users and others trying to do similar analyses.

Acknowledgments

This work was supported in part by the DOE Grant B341494 funded by CSAR; the DOE grant DE-FG05-08OR23332 through ORNL LCF and the NASA grant NNX08AD19G. The authors would like to thank Profs. Carleton DeTar and Steven Gottlieb for their advice on running MILC and on analyzing the performance results. They also wish to thank Dr. Mark Taylor for providing the turbulence code (DNS) and answering our queries.

This research used the Blue Gene/P machine at Argonne National Laboratory, which is supported by DOE under contract DE-AC02-06CH11357. Runs on Abe and Ranger were done under TeraGrid allocation grant ASC050040N supported by NSF. The research also used Jaguar at Oak Ridge National Laboratory, which is supported by the DOE under contract DE-AC05-00OR22725. Accounts on Jaguar were made available via the Performance Evaluation and Analysis Consortium End Station, a DOE INCITE project.

Biographies

Abhinav Bhatelé: Abhinav received a B. Tech. degree in Computer Science and Engineering from I.I.T. Kanpur (India) in May 2005 and a M. S. degree in Computer Science from the University of Illinois at Urbana-Champaign in 2007. He is a Ph.D. student at the Parallel Programming Lab at the University of Illinois, working with Prof. Laxmikant V. Kalé. His research is centered around topology aware mapping and load balancing for parallel applications. Abhinav has received the David J. Kuck Outstanding MS Thesis Award in 2009, Third Prize in the ACM Student Research Competition at SC 2008, a Distinguished Paper Award at Euro-Par 2009 and the George Michael HPC Fellowship Award at SC 2009.

Lukasz Wesolowski: Lukasz is a Ph.D. student in computer science at the University of Illinois at Urbana-Champaign, where he also received his B.S. and M.S. degrees in computer science. His areas of interest include tools and abstractions for parallel programming, general purpose graphics processing units, and large scale parallel applications.

Eric Bohm: Eric received a B.S. degree in Computer Science from the State University of New York at Buffalo in 1992. He worked as Director of Software Development for Latpon Corp from 1992 to 1995, then as Director of National Software Development from 1995 to 1996. He worked as Enterprise Application Architect at MEDE America from 1996 to 1999. He completed his time in industry as Application Architect at WebMD from 1999 to 2001. Following a career shift towards academia, he joined the Parallel Programming Lab at University of Illinois at Urbana-Champaign in 2003. His current focus as a Research Programmer is on optimizing molecular dynamics codes for tens of thousands of processors.

Edgar Solomonik: Edgar is an Undergraduate Computer Science student at the University of Illinois at Urbana-Champaign. His research for the Parallel Programming Lab involves work on parallel scientific applications and parallel algorithms.

Laxmikant V Kalé: Professor Laxmikant Kalé has been working on various aspects of parallel computing, with a focus on enhancing performance and productivity via adaptive runtime systems, and with the belief that only interdisciplinary research involving multiple CSE and other applications can bring back well-honed abstractions into Computer Science that will have a long-term impact on the state-of-art. His collaborations include the widely used Gordon-Bell award winning (SC'2002) biomolecular simulation program NAMD, and other collaborations on computational cosmology, quantum chemistry, rocket simulation, space-time meshes, and other unstructured mesh applications. He takes pride in his group's success in distributing and supporting software embodying his research ideas, including Charm++, Adaptive MPI and the ParFUM framework.

L. V. Kalé received the B.Tech degree in Electronics Engineering from Benares Hindu University, Varanasi, India in 1977, and a M.E. degree in Computer Science from Indian Institute of Science in Bangalore, India, in 1979. He received a Ph.D. in computer science in from State University of New York, Stony Brook, in 1985. He worked as a scientist at the Tata Institute of Fundamental Research from 1979 to 1981. He joined the faculty of the University of Illinois at Urbana-Champaign as an Assistant Professor in 1985, where he is currently employed as a Professor.

References

Alam, S., R. Barrett, M. Bast, M. R. Fahey, J. Kuehn, C. McCurdy, J. Rogers, P. Roth, R. Sankaran, J. S. Vetter, P. Worley, and W. Yu (2008). Early evaluation of IBM Blue Gene/P. In *SC 08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pp. 1–12. IEEE Press.

Alam, S. R., J. A. Kuehn, R. F. Barrett, J. M. Larkin, M. R. Fahey, R. Sankaran, and P. H. Worley (2007). Cray XT4: An early evaluation for petascale scientific simulation. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pp. 1–12. ACM.

Bernard, C., T. Burch, T. A. DeGrand, C. DeTar, S. Gottlieb, U. M. Heller, J. E. Hetrick, K. Orginos, B. Sugar, and D. Toussaint (2000). Scaling tests of the improved Kogut-Susskind quark action. *Physical Review D* (61).

Bhatele, A. and L. V. Kale (2009, May). An Evaluation of the Effect of Interconnect Topologies on Message Latencies in Large Supercomputers. In *Proceedings of Workshop on Large-Scale Parallel Processing (IPDPS '09)*.

Bhatele, A., S. Kumar, C. Mei, J. C. Phillips, G. Zheng, and L. V. Kale (2008, April). Overcoming Scaling Challenges in Biomolecular Simulations across Multiple Platforms. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2008*.

Dongarra, J. and P. Luszczek (2005). Introduction to the HPC Challenge Benchmark Suite. Technical Report UT-CS-05-544, University of Tennessee, Dept. of Computer Science.

Hoisie, A., G. Johnson, D. J. Kerbyson, M. Lang, and S. Pakin (2006). A performance comparison through benchmarking and modeling of three leading supercomputers: Blue

Gene/L, Red Storm, and Purple. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, New York, NY, USA, pp. 74. ACM.

IBM Blue Gene Team (2008). ewblock Overview of the IBM Blue Gene/P project. *IBM Journal of Research and Development 52*(1/2).

IBM System Blue Gene Solution (2008). Blue Gene/P Application Development Redbook. http://www.redbooks.ibm.com/abstracts/sg247287.html.

Kalé, L. and S. Krishnan (1993, September). CHARM++: A Portable Concurrent Object Oriented System Based on C++. In A. Paepcke (Ed.), *Proceedings of OOPSLA'93*, pp. 91–108. ACM Press.

Kufrin, R. (2005). Perfsuite: An Accessible, Open Source Performance Analysis Environment for Linux. In *In Proceedings of the Linux Cluster Conference*.

Kumar, S., G. Dozsa, J. Berg, B. Cernohous, D. Miller, J. Ratterman, B. Smith, and P. Heidelberger (2008). Architecture of the Component Collective Messaging Interface. In *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pp. 23–32. Springer-Verlag.

Oliker, L., A. Canning, J. Carter, C. Iancu, M. Lijewski, S. Kamil, J. Shalf, H. Shan, E. Strohmaier, S. Ethier, and T. Goodale (2007). Scientific Application Performance on Candidate PetaScale Platforms. In *Proceedings of IEEE Parallel and Distributed Processing Symposium (IPDPS)*.

Phillips, J. C., R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten (2005). Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry 26*(16), 1781–1802.

Taylor, M. A., S. Kurien, and G. L. Eyink (2003). Recovering isotropic statistics in turbulence simulations: The Kolmogorov 4/5th law. *Physical Review E* (68).

Williams, S., A. Waterman, and D. Patterson (2009). Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM 52*(4), 65–76.

Zheng, G., G. Kakulapati, and L. V. Kalé (2004, April). 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, pp. 78.