Cluster Computing manuscript No.
(will be inserted by the editor)

The Impact of System Design Parameters on Application Noise Sensitivity

Kurt B. Ferreira · Patrick G. Bridges · Ron Brightwell · Kevin T. Pedretti

the date of receipt and acceptance should be inserted later

Abstract Operating system (OS) noise, or jitter, is a key limiter of application scalability in high end computing systems. Several studies have attempted to quantify the sources and effects of system interference, though few of these studies show the influence that architectural and system characteristics have on the impact of noise at scale. In this paper, we examine the impact of three such system properties: platform balance, noisy node distribution, and the choice of collective algorithm. Using a previously-developed noise injection tool, we explore how the impact of noise varies with these platform characteristics. We provide detailed performance results that indicate that a system with relatively less network bandwidth is able to absorb more noise than a system with more network bandwidth. Our results also show that application performance can be significantly degraded by only a subset of noisy nodes. Furthermore, the placement of the noisy nodes is also important, especially for applications that make substantial use of tree-based collective communication operations. Lastly, performance results indicate that non-blocking collective operations have the ability to greatly mitigate the impact of OS interference. When combined, these results show that the impact of OS noise is not solely a property of application communication behavior, but is also influenced by other properties of the system architecture and system software environment.

Keywords Operating Systems Interference; Jitter; System Balance; Non-blocking Collectives

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Kurt B. Ferreira · Ron Brightwell · Kevin T. Pedretti
Scalable System Software Department
Sandia National Laboratories
Albuquerque, NM 87185–1319
E-mail: kbferre,rbbrigh,ktpedre@sandia.gov

Kurt B. Ferreira · Patrick G. Bridges
Computer Science Department
The University of New Mexico
Albuquerque, NM 87131
E-mail: kurt,bridges@cs.unm.edu

1 Introduction

Research has shown that operating system (OS) interference is a key limiter of application performance in large-scale systems [11,19,7], with much of this work focusing on how applications respond to different amounts and types of noise. However, the measured impact of noise has varied widely between systems, with some platforms showing relatively little performance impact from noise [3] and others showing substantial performance impacts [11,19,7].

In this paper, we study how a number of important architectural and system software design features as well as application behavior affect the noise sensitivity of high-performance computing (HPC) systems. In particular, we examine how the following important system features impact the sensitivity of applications to OS noise:

- The hardware *balance* of the system, the ratio of peak network bandwidth (bytes/second) to peak compute performance (floating point operations/second)[1,18].
- The isolation of "noisy" nodes running full-featured operating systems to a subset of the nodes on the system.
- The use of collective communication mechanisms that are relatively insensitive to noise.

Our results provide important guidance to HPC hardware and system software designers by demonstrating that:

1. The impact of noise is dependent on machine parameters - specifically network performance and the resulting balance of the hardware platform;
2. Isolating noise to only a subset of system nodes is not sufficient to mitigate the impact of noise at scale on certain key HPC applications;
3. The placement of noisy nodes in the system matters, with noisy nodes close to the root of the system collective communication tree (rank 0) having less impact on application performance than nodes further from rank 0;
4. In a very noisy environment, the choice of optimum collective algorithm can be different than that in a low noise environment; and,
5. Non-blocking collective reductions can substantially mitigate the impact of noise on applications running in HPC systems.

To our knowledge, this is the first such empirical study on the impact system design parameters have on an HPC applications sensitivity to OS noise.

The remainder of the paper is organized as follows. Section 2 provides background on OS noise, its effect on application performance in HPC systems, and the system features listed that we hypothesize affect the impact of noise on applications in HPC systems. Section 3 describes the hardware platform we use to test the impact of these system features on application noise sensitivity, how we vary these features on this hardware platform, and the applications we use to test how variations in system features impact application noise sensitivity. Section 4 then presents and analyzes the results of these experiments. Section 5 follows with discussion of related work in this area, and Sections 7 and 6 present directions for future work and conclude.

2 Background

2.1 OS Noise

The detrimental side effects of OS interference on massively parallel processing systems have been known and studied, primarily qualitatively, for nearly two decades [23]. Previous investigations have shown that the global performance cost of noise is, in many cases, due to the variance in the time for processes to participate in a collective operation, such as `MPI_Allreduce`, resulting in the accumulation of noise at scale [17]. These interruptions occur for a variety of reasons, from the periodic timer "tick" commonly used by many commodity operating systems to keep track of time, to the scheduling points used to replace the currently running process with another task or kernel daemon. In each of these cases, processor cycles are taken away for the duration of the noise event, which can typically vary from a few microseconds to a few milliseconds [19,3]. A number of recent studies [11,19,7] have shown that even relatively minimal OS noise (e.g. 2.5% OS overhead) can reduce the performance of applications at scale by orders of magnitude.

2.2 System Features Affecting Noise

While noise has been recognized as a substantial factor in application scaling in HPC systems, different platforms have seen dramatic differences in how much impact noise has had on applications, and a number of techniques have been proposed for mitigating the impact of system noise on applications [15]. For example, noise injection studies on two different systems, the Cray Red Storm system [7] and the IBM BG/L system [3] measured dramatically different slowdowns in `MPI_Allreduce` performance in the presence of small amounts (e.g. 2.5%) of injected noise. Exactly what accounts for the different noise sensitivities of these two systems is not clear; system differences that could affect noise propagation include the different ratios between compute and communication of the two systems, BG/L's isolation of operating systems with non-trivial amounts of noise to a subset of its nodes, or the noise-resistant collectives that BG/L includes in the form of a hardware collective communication network.

2.2.1 System Balance

System balance, the ratio of peak network bandwidth (bytes/second) to peak compute performance (floating point operations/second) [1,18] is one potential system hardware feature that we hypothesize could alter the impact of OS noise on application performance. Previous work has shown that a bytes-to-flops ratio of one results in the best performance for a typical HPC workload [18]. However, given the ever-increasing compute performance available from multi-core processors and the inability of network performance to keep pace, well-balanced machines are becoming increasingly difficult to design and deploy. Our supposition is that OS noise is less likely to impact applications on systems that are less balanced, since OS noise is more likely to be absorbed in an environment where there are potentially excess compute cycles available.

2.2.2 OS Noise Isolation

Isolating OS services to a subset of system nodes has been a popular technique mitigating the impact of necessary OS noise on application performance. The ASCI Q system, for example, was changed to run most system services on dedicated processors and dedicated nodes in response to the well-known study of the impact of noise on the SAGE application [19]. Similarly, IBM BG-series systems run a low-noise Compute Node Kernel (CNK) [16] on most compute nodes and distribute Linux OS service nodes throughout the system. System calls are forwarded from CNK-based nodes to these Linux service nodes in an effort to isolate OS noise in the system. Finally, we note that a similar strategy was proposed for supporting full-featured operating system services on an Intel Paragon system [12] at Sandia, but was never deployed in production. Cray has also recently taken a similar approach to isolating OS services to specific cores within a node. They refer to this capability as "core specialization". System services are bound to a small subset of specific cores that do not run application processes or threads. Our hypothesis is that this isolation of OS noise can substantially reduce the impact of noise to applications in the system, but that the placement of "noisy" nodes in the system also matters.

2.2.3 Noise-resistant Collectives

Finally, since collective communication primitives such as `MPI_Allreduce` have been shown to be a key factor in accumulating noise in HPC systems [7,9], a variety of noise-resistant collective communication primitives has been proposed as a possible means of mitigating this effect. A variety of collective communication algorithmic and implementation techniques could potentially impact noise sensitivity, including hardware-based collectives, alternative collective communication patterns, and non-blocking collectives.

Hardware-based collectives have become increasingly common, though they have had limited success in reducing noise impact. Hardware-assisted barriers on the ASCI Q system, for example, had little impact on improving SAGE performance in the presence of noise [19]. The more general hardware-based collectives on IBM BG series of systems [2], however, are generally regarded as an important source of that system's scalability.

The wide range of available collective communication algorithms [22] also potentially has different sensitivity to collective communications. Most prior research on noise sensitivity has focused on tree-based collective operations, which are frequently used for broadcasts and reductions in large systems because of their scalability to large nodes in terms of the number of messages that must be sent. However, each algorithm's sensitivity to noise also impacts its scalability, and, to our knowledge, this aspect of collective communication scaling has not been studied.

Finally, non-blocking collectives have been proposed to the recently reconvened MPI Forum for inclusion in MPI-3 [8] as another means of reducing the impact of OS noise on large-scale applications that require collective communication. Similar to non-blocking point-to-point operations, non-blocking collectives allow an application to hide the cost of the operation by overlapping the network communication with computation. The key to benefiting from a non-blocking operation is the size of the overlap portion of the computation. We hypothesize that the amount of overlap an application is actually able to achieve between computation and collective propagation is directly related to the ability of noise-resistant collective implementations to absorb noise.

3 Approach

In this section, we provide an overview of the hardware and software environment of the test system used to evaluate the impact of the system features described in Section 2.2 on noise sensitivity. This includes a description of the test platform, the changes to the platform we have implemented in system software on this platform, the three applications evaluated on this platform, and the benchmarks we have developed to evaluate the impact of various collectives algorithms on noise propagation and accumulation.

3.1 Hardware Platform

We used the Red Storm system located at Sandia National Laboratories as a test platform. Red Storm is a Cray XT3/4 series machine consisting of nearly 13 thousand nodes. For our experiments, we used a 3000-node subset of the machine in dedicated mode. Each compute node in this subset contains a 2.4 GHz dual-core AMD Opteron processor and 4 GB of main memory. Additionally, each node contains a Cray SeaStar [4] network interface and high-speed router. The SeaStar is connected to the Opteron via a HyperTransport link. The current-generation SeaStar is capable of sustaining a peak unidirectional injection bandwidth of more than 2 GB/s and a peak unidirectional link bandwidth of more than 3 GB/s. An important characteristic of

the SeaStar network on the Cray XT is that it is interrupt driven. When a message arrives at the SeaStar, it interrupts the host processor, the host OS performs the necessary protocol processing, and then programs the SeaStar's network DMA engines directly to deliver the incoming message to the appropriate buffer in destination process' address space. Red Storm is an ideal platform on which to explore system balance, as it is one of the more balanced modern-day systems.

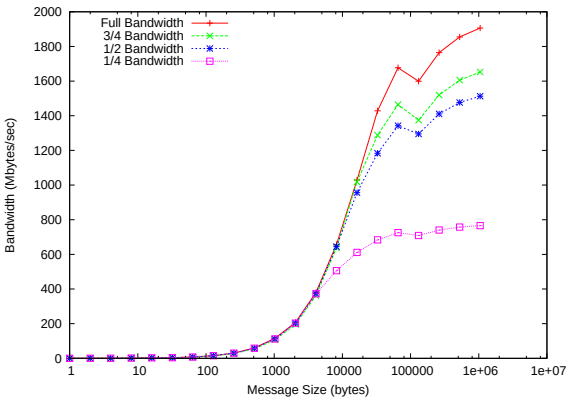
3.2 Noise Injection

For our experiments, we modified the system to run the Catamount lightweight kernel containing our noise injection framework described in [7] instead of the normal production version, along with additional modifications allowing us to control *which* nodes generated noise. The Catamount lightweight kernel is an ideal environment for noise studies due to its extremely low native noise signature and demonstrated record of scalability. All of our experiments were run using one process per node, thereby maximizing the overall balance of the system.

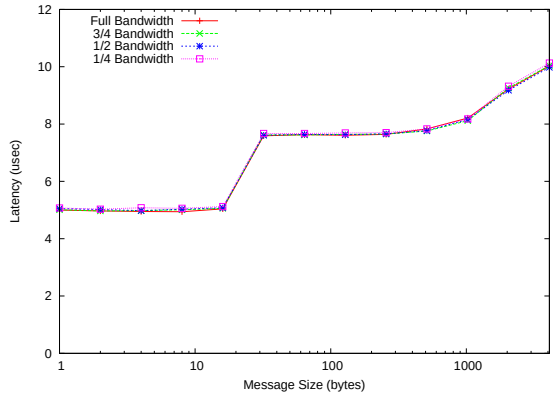
For this work, we used noise signatures that represents 2.5% net processor interference, focusing on a 10 Hz 2500 μs noise profile that is representative of kernel daemon interference. We focused on this 2.5% profile due to both specific measurements made on commodity systems and results of previous research that demonstrated the importance of this noise level [19, 7]. It is important to note that unloaded systems (e.g. those doing no communication, I/O, or memory management activities) can have lower noise signatures with corresponding lower overheads. We believe these unloaded noise patterns are not realistic for characterizing the behavior of real-world HPC applications, and this view is supported by recent results [17] that show significant OS overhead from scheduling and ACPI interrupts on loaded HPC Linux systems.

3.3 System Balance Modification

The SeaStar router supports a three-dimensional mesh topology with an optional torus in some or all dimensions. Each node in the mesh can be connected to one of its six possible neighbors by a point-to-point link. These links are configured at system boot time to operate at full speed, but the router supports a degraded mode that can be used to maintain communication in the presence of bad or failing connections. It is possible to manually configure the links to operate in this degraded mode at boot time by changing the router manager configuration file. Three degraded modes are supported that result in one-quarter, one-half, and three-quarters of full bandwidth performance.



(a) Bandwidth



(b) Latency

Fig. 1: MPI ping-pong bandwidth and latency result for degraded bandwidths.

To evaluate the impact of system balance on the noise sensitivity of HPC applications, we modified the balance of Red Storm using these degraded network bandwidth modes. We present results of three-quarters, one-half, and one-quarter bandwidth configurations that roughly correspond to the balance of the ASC Purple supercomputer located at Lawrence Livermore National Laboratories, a commodity single processor InfiniBand cluster, and a commodity dual-processor InfiniBand cluster similar to that of the Thunderbird cluster at Sandia National Laboratories, respectively. Note, that IBM's BG/L and BG/P machines are even further imbalanced towards excess computation than the configurations examined here. See [4] for a balance comparison of several recent large-scale computing systems.

Figure 1 shows the resulting MPI ping-pong bandwidth and latency numbers. For bandwidth in Figure 1a, with message sizes less than 4 KB the bandwidth of all scenarios are nearly equal due to message overhead, but for message sizes greater than 4 KB, the bandwidth

curves diverge, with 1 MB messages peaking at around 1900 MB/s, three-quarters bandwidth at slightly more than 1600 MB/s, one-half bandwidth at 1500 MB/s and one-quarter bandwidth at just under 800 MB/s. Figure 1b shows that while the network configuration de-tuning affects maximum bandwidth, MPI latency of small messages remains virtually unchanged. Together, these numbers demonstrate that the hardware mechanism allows for controlling network link bandwidth, and hence system balance, independent of latency.

3.4 Blocking Collective Algorithms

A great deal of research has been conducted on optimizing collective communication primitives for HPC systems because of their impact on the performance of many parallel applications. These algorithms vary from linear implementations, to binomial tress with recursive doubling or halving, to operations on rings [22]. The choice of optimal algorithm is typically dependent on the size of the collective operation and the number of nodes performing the operation.

Because blocking collectives operations have been shown to limit scalability of HPC applications in noisy environments [19, 7], we compare the performance of a linear and a tree-based `MPI_Allreduce` algorithm with and without OS noise injection. In the linear algorithm, all nodes send the data to be reduced to a root node, in this case rank 0. The root node reduces the data from each node, and once the reduction is complete sends the computed result to each node via a point-to-point operation. In the binomial tree algorithm, nodes are organized into a logical tree rooted at rank zero. Internal tree nodes wait for data from each of their children, perform the requested reduction and then send their node to their parent in the tree. Once the root node has computed the final reduction value, it similarly broadcasts it down the reduction tree.

3.5 Non-blocking Collectives

To test the impact of non-blocking collectives on noise propagation and accumulation, we implemented a non-blocking `MPI_Allreduce` collective operation. Again, the Allreduce operation was chosen due to the sensitivity of this collective operation shown in [7]. Although there are a number of non-blocking collective libraries currently in existence (most notably [8]), we chose to implement our own in order to take full advantage of the SeaStar interconnect on Red Storm.

Because using non-blocking collectives would require substantial application changes, we also implemented a bulk-synchronous micro-benchmark that uses this non-blocking collective library. This bulk-synchronous micro-benchmark allows for specifying the length of time of the compute phase before all nodes join in the Allreduce-based synchronize step. We set the synchronization step of the micro-benchmark to occur at a rate previously measured for the SAGE and POP applications, described below.

3.6 Test Applications

When appropriate, we used three applications that represent important HPC modeling and simulation workloads, CTH, SAGE, and POP, to evaluate the impact of hardware and system software changes on noise propagation and accumulation in applications. These applications represent a range of different computational techniques, all frequently run at very large scales (i.e. tens of thousands of nodes), and are key applications to both the United States Departments of Energy and Defense. We briefly describe these applications below.

CTH [6] is a multi-material, large deformation, strong shock wave, solid mechanics code developed by Sandia with models for multi-phase, elastic viscoplastic, porous, and explosive materials. CTH supports three-dimensional rectangular meshes; two-dimensional rectangular, and cylindrical meshes; and one-dimensional rectilinear, cylindrical, and spherical meshes, and uses second-order accurate numerical methods to reduce dispersion and dissipation and to produce accurate, efficient results. It is used for studying armor/anti-armor interactions, warhead design, high explosive initiation physics, and weapons safety issues.

SAGE, SAIC's Adaptive Grid Eulerian hydrocode, is a multi-dimensional, multi-material, Eulerian hydrodynamics code with adaptive mesh refinement that uses second-order accurate numerical techniques [13]. It represents a large class of production applications at Los Alamos National Laboratory. It is a large-scale parallel code written in Fortran 90 and uses MPI for inter-processor communications. SAGE routinely runs on thousands of processors for months at a time.

The Parallel Ocean Program (POP) [14] is an ocean circulation model developed at Los Alamos National Laboratory that is capable of ocean simulations as well as coupled atmosphere, ice, and land climate simulations. Time integration is split into two parts: baroclinic and barotropic. In the baroclinic stage, the three-dimensional vertically-varying tendencies are integrated using a leapfrog scheme. The baroclinic stage consists of a preconditioned conjugate gradient solver which is used to solve for the two-dimensional surface pressure.

4 Experimental Results

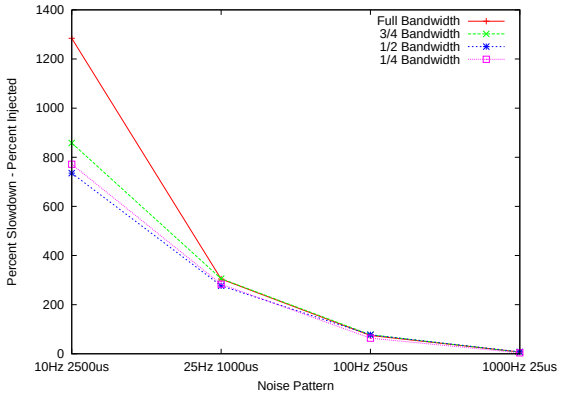
4.1 Changing System Balance

To measure the impact that changing system balance has on system noise sensitivity, we ran each of the three applications described in the previous section with different available network bandwidths and 2.5% injected noise ranging from low-frequency/high-duration profiles (10 Hz/2500 μs) to high-frequency/low-duration profiles (1000 Hz/25 μs). We then measured the amount of *excess* slowdown that each application experienced after subtracting out the 2.5% injected noise – that is, the amount of additional noise that *accumulated* during the application run. Each data point represents the average of three runs, and we ran each application at the largest system size that an available data set supported and for which we were able to get an allocation – 2500 nodes in the case of POP, 3360 nodes in the case of SAGE, and 2048 nodes in the case of CTH.

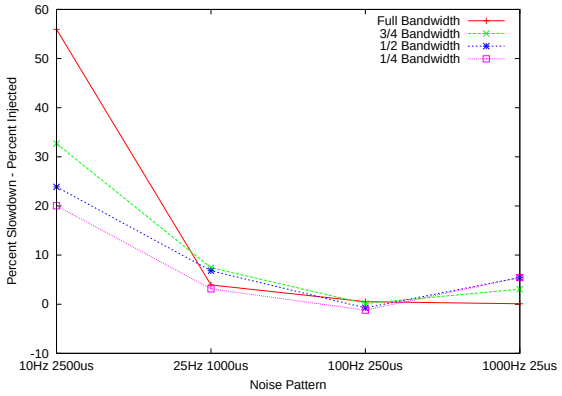
Figure 2 shows how noise accumulates for each of POP, SAGE, and CTH with varying network bandwidth and 2.5% noise profiles. Shifting the balance of the system in favor of computation by reducing the amount of network bandwidth available reduces the performance impact of noise on SAGE and POP, though POP in particular is still significantly impacted by low-frequency/high-duration noise similar to that caused by scheduling a kernel daemon. CTH also accumulates less noise on an unbalanced system, but because it accumulates little noise to begin with (approximately 12% in the worst case), this is less significant.

Similarly, Figure 3, shows the impact of scale on the accumulation of noise for a 10 Hz 2500 μs noise signature (CTH results not included in the figure due due to CTH's very low noise accumulation). This shows that, independent of scale, the system balance decreases the impact of OS noise on the these two HPC applications. The difference between the curves for POP and SAGE is mainly due to the fact that POP is a strong scaling application and SAGE is a weak scaling application.

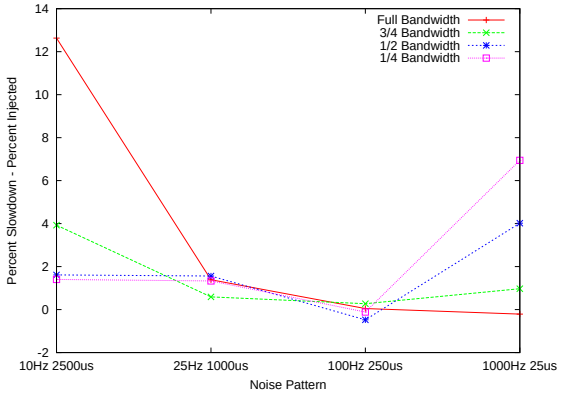
Noise accumulates under varying system balances for each of these applications because of differences in the amount of computation and types of communication they perform. Figure 4 shows breakdowns of how POP and SAGE divide time between computation and different communication primitives. POP, for example, spends the majority of time at scale in small `MPI_Allreduce` operations and relatively little time in computation. As a result, it still incurs substantial slowdown due to noise accumulation (more than 1000% on 2500 nodes) even when the system balance is tilted heavily in favor of computation. SAGE, on the other hand, can leverage an imbalanced system more effectively to reduce noise impact because of its larger computational demands. Finally, while noise has the least impact on CTH in any case, changes in system balance do significantly affect its performance, as CTH is network bandwidth limited [18].



(a) POP (2500 nodes)

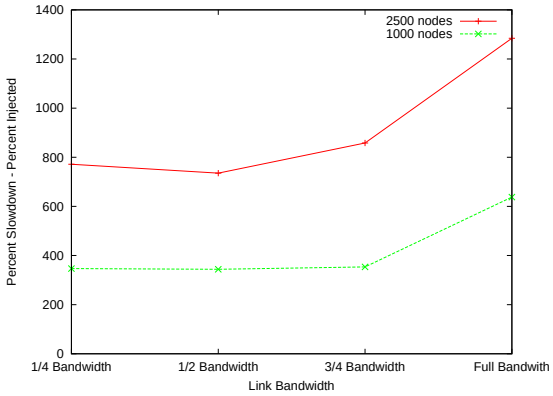


(b) SAGE (3360 nodes)

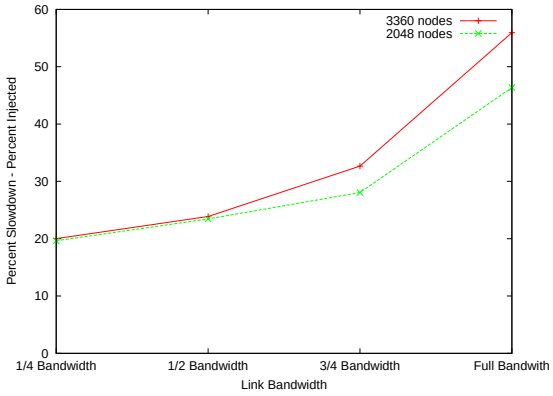


(c) CTH (2048 nodes)

Fig. 2: Accumulation of noise in application runtime with varying network bandwidth and different 2.5% net injected CPU noise profiles.

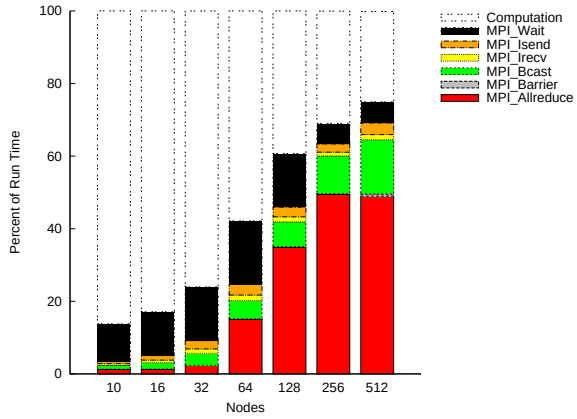


(a) POP

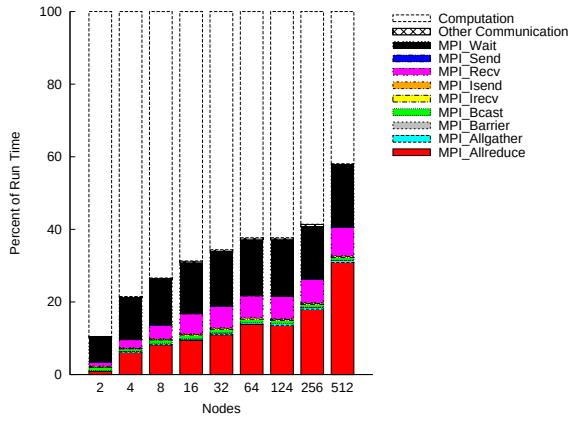


(b) SAGE

Fig. 3: Impact of node count on accumulation of noise in application runtime with varying network bandwidth for 10 Hz 2500 μs, 2.5% net injected CPU noise profile.



(a) POP



(b) SAGE

Fig. 4: Breakdown of application runtime between computation and various communication primitives with full network bandwidth and no injected noise.

4.2 Isolating Noise to a Subset of Nodes

To measure the impact of isolating noise-generating actions onto a subset of system nodes, we varied the percentage and location of noise-injecting nodes in the system and measured the application noise accumulation. Nodes injecting noise were generally chosen randomly using a Fisher-Yates permutation [5] to shuffle the list of MPI ranks of the job. We then choose the first N elements of the list as the ranks of noisy nodes. Each data point in the following graphs corresponds to an average of at least five data points (maximum of 6) with error bars shown.

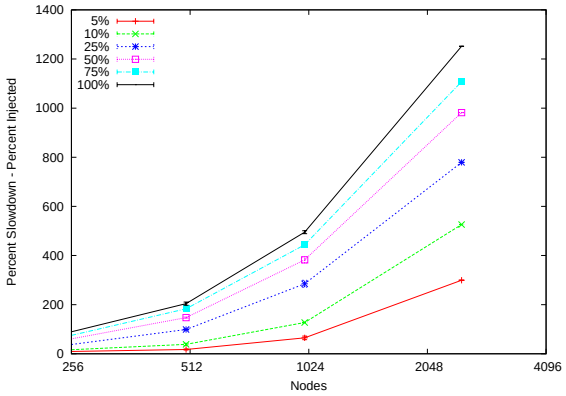
4.2.1 Varying percentage of nodes injecting noise

Figure 5 shows the impact on noise accumulation for the three applications of isolating noise generation to a varying percentage of system nodes. In the case of POP, reducing noise generation to just 5-10% of the system nodes still results in substantial application slowdown. In particular, we note the slowdown for POP is *not* related purely to the number of nodes injecting noise; for example, POP is slowed down by more than 500% on 2500 nodes when only 250 nodes are injecting noise, but slows down only nominally when 100% of the nodes are noisy in the 256-node case.

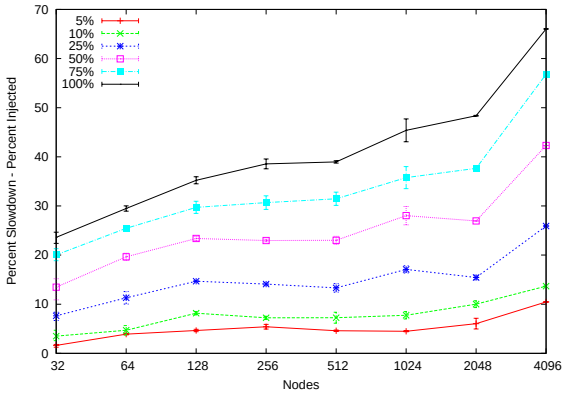
In contrast, noise isolation appears to be a very successful strategy for SAGE (as others have found [19]). For example, isolating noise to roughly 10% of the system nodes reduces noise accumulation in SAGE by a factor of three on 2048 nodes. Finally, noise isolation appears to be largely irrelevant to CTH. Also, in contrast to POP and SAGE, the slowdowns for CTH (though smaller than POP and SAGE) appear to be relatively constant, independent of the number of noisy nodes.

4.2.2 Placement of noisy nodes

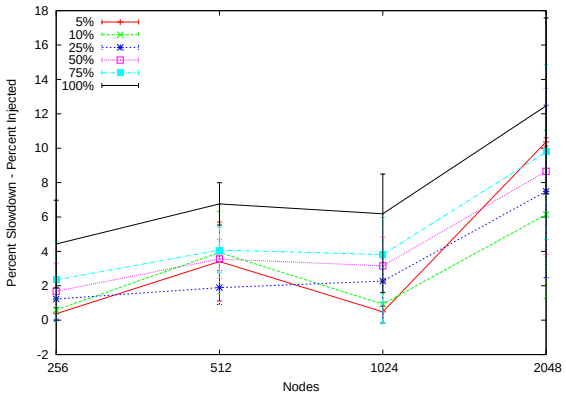
To study how changing the location of noise injection affected application performance, we used two different



(a) POP

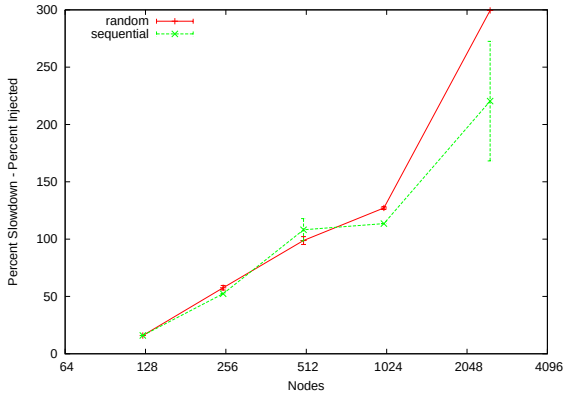


(b) SAGE

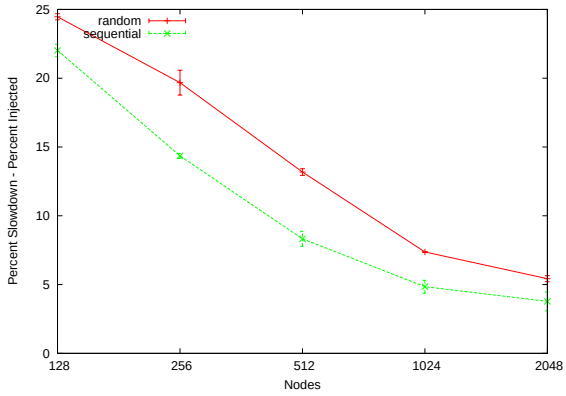


(c) CTH

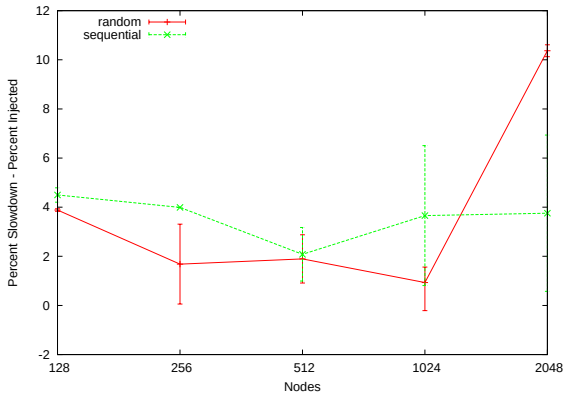
Fig. 5: Accumulation of noise in application runtime with varying percentages of nodes injecting 2.5% net CPU noise.



(a) POP



(b) SAGE



(c) CTH

Fig. 6: Accumulation of noise in application runtime with nodes injecting noise arranged randomly or sequentially around rank 0. 125 nodes are injecting noise with POP, while 128 nodes are injecting noise with SAGE and CTH.

policies to place 125-128 noisy nodes into a system run: random and sequential. Random uses the Fisher-Yates shuffle mentioned above, while sequential places noise-injection nodes as the first 125–128 nodes in the system starting at rank zero.

As seen in Figure 6, placement of noisy nodes in the system can have a substantial effect. Noisy nodes close to rank 0 result in less noise accumulation. We believe

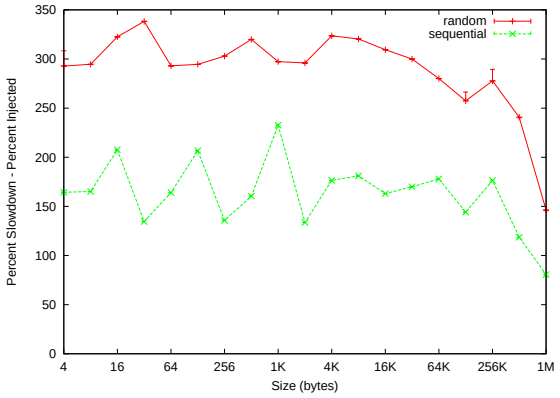


Fig. 7: Slowdown of `MPI_Allreduce` on 128 nodes with 6.25% of the nodes generating low-frequency, high-duration noise either distributed randomly throughout the application (random) or around and including rank zero (sequential)

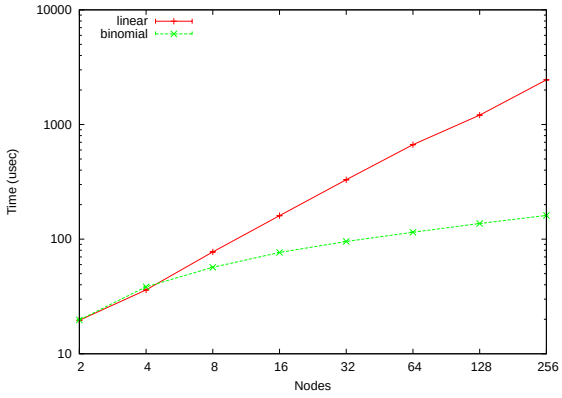
this is due to the collective communication primitives on the Sandia Red Storm system using tree-based algorithms, with the root of the tree at rank 0. As a result, placement of noisy nodes near the root limits the accumulation of noise when performing collectives, while randomly placing noisy nodes, including potentially at leaves in the collective tree, allows more noise to accumulate on average. To illustrate this difference, we use a `MPI_Allreduce` micro-benchmark on 128 nodes with eight noisy nodes either distributed randomly or around rank zero. From Figure 7 we see that, at this scale, random distribution leads to a nearly factor of two slowdown over the noisy nodes distributed around rank zero.

4.3 Noise-Resistant Collectives

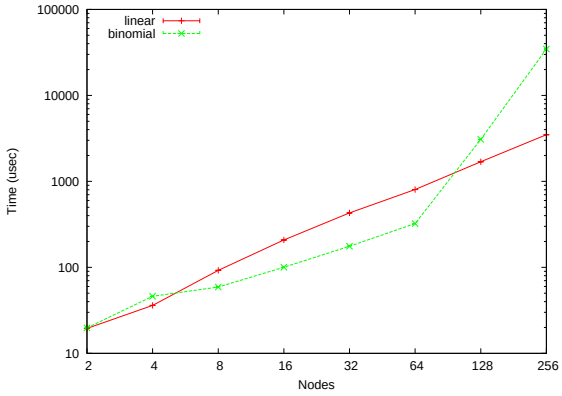
As described earlier, the different techniques and algorithms for implementing collective communication can potentially impact noise sensitivity. Here we describe the results of experiments to quantify the noise sensitivity of two such approaches to realizing collective communication in parallel systems: linear versus tree-based collective communication and non-blocking collective primitives.

4.3.1 Linear Versus Tree-based Collectives

To examine the performance of different collective algorithms in noisy environments, we used the two different `MPI_Allreduce` algorithms described previously: linear and binomial. We tested each algorithm using an `MPI_Allreduce` micro-benchmark that repeatedly performs `MPI_Allreduce` operations using the `MPI_SUM` operator. We ran this benchmark on node counts ranging



(a) No Injected OS Noise



(b) 5% Low Frequency / High Duration Noise Signature

Fig. 8: Accumulation of noise in Allreduce micro-benchmark using a linear or binomial-tree based implementation

from two to 256, and with both no noise and 5% injected low-frequency, high-duration OS noise.

Figure 8 shows the results of these tests, and demonstrates that the two algorithms have very different behavior in noise-free versus noisy environments. As expected, the binomial-tree based algorithm greatly outperforms the linear algorithm for node counts greater than four with no injected noise. With large amounts of OS noise, however, the linear `MPI_Allreduce` algorithm outperforms the tree-based algorithm for more than 64 nodes because the linear algorithm can more effectively absorb noise by performing partial reductions or handing requests from non-noisy nodes while waiting for data from nodes delayed due to injected OS noise.

4.3.2 Non-blocking Collectives

To examine the impact of non-blocking collectives on system noise sensitivity, we studied how increasing the amount of overlap between application computation and collective communication affected the noise sensitivity of the synthetic bulk-synchronous benchmark described

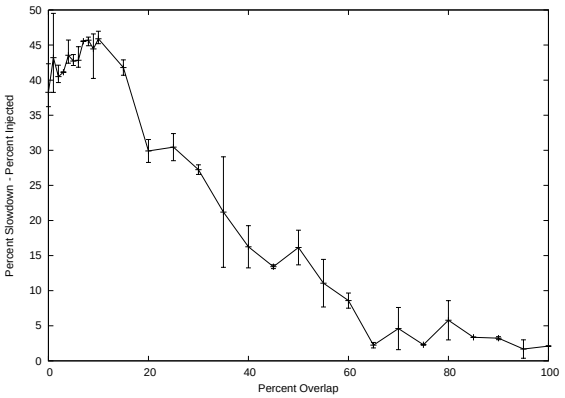


Fig. 9: Slowdown of non-blocking Allreduce in bulk-synchronous micro-benchmark as a function of overlap of communication with computation

in Section 3.5. We achieved this overlap using the non-blocking Allreduce collective also described above.

Figure 9 illustrates the slowdown of our non-blocking micro-benchmark with a low-frequency, high duration noise signature, similar to that of a periodic kernel thread or daemon. For this test, we specified the bulk-synchronous interval to be that of what we measured for SAGE [7] and the slowdown is in comparison to a run with no noise. Each data point in the figure corresponds to an average of ten runs with error bars shown. From this figure, we see that in a noisy environment with sufficient overlap, the slowdown due to noise can be reduced to nearly zero. In this case in particular, a 2.5% noise signature could be completely absorbed with 62% overlap between application computation and collective communication.

5 Related Work

Though the impact of OS noise on HPC systems has been studied for more than fifteen years [23], the seminal work of Petrini et al. [19] most recently raised the visibility of the impact of OS noise on application performance. This thorough study investigated performance issues from OS noise on a large-scale cluster built from commodity hardware components, running a commodity operating system, and running a cluster software environment designed for data center applications. While the findings of this paper from an OS perspective were largely well known, such as turning off unnecessary system daemons, the paper brought to light several important new findings relevant to OS noise. The authors developed a micro-benchmark specifically for measuring OS noise on a parallel machine; such benchmarks were previously non-existent. Second, the paper demonstrates the inability of communication micro-benchmarks to accurately reflect and/or predict application performance. Also, the authors offered a conjecture that OS noise is most damaging when the application resonates with OS noise. Finally, this work showed the advantage of dedicating a portion of hardware resources to performing system tasks.

Beckman et al. [3] investigated the effect of user-level noise on an IBM BG/L system. This system runs a custom lightweight OS like Catamount that demonstrates very little noise. This system contains a number of hardware facilities which allow for collective operations to be performed in hardware and therefore not sensitive to CPU noise. In addition, this system has a balance shifted towards excess computation in comparison to other systems like the Cray XT series. In this paper, the authors showed that a properly configured Linux kernel can have a noise signature similar to that of a lightweight kernel. Using a user-level injection mechanism built into the communication library and a series of micro-benchmarks, the authors showed that noise levels had to be very high in order to show any real impact. We believe this difference in noise impact with other studies is due to the difference in how noise is injected as well as the architectural differences of the BG/L system.

Nataraj et al. [17] used the KTAU toolkit to investigate the kernel activities of a general-purpose operating system. This toolkit instruments the Linux kernel to collect measurements from various kernel components including system calls, scheduling, interrupt handling, and network operations. The authors begin by showing the effectiveness of the KTAU toolkit for measuring the OS noise in Linux. In addition, they show how their toolkit can be used to track the accumulation and absorption of noise during the communication stages of an application. However, this work, unlike ours, only presents results from a 128-node development system that may or may not generalize to a massively parallel machine containing tens of thousands of nodes. More importantly, while their tool can be used to identify possible sources of noise, the authors do not relate the effects of noise to the performance of a large-scale application (e.g. the largest source of noise may not be the most harmful).

Recently, we examined the sensitivity of OS noise at scale for three real-world HPC applications using a kernel-level noise injection framework on a well balanced architecture [7]. In this paper we showed the importance of how noise is injected and the application communication characteristics that impact noise absorption. For example, we showed how the computation/communication ratios, collective communication sizes, and other characteristics of an application, relate

to their ability to amplify or absorb noise. Finally, this paper discussed the implications of our findings on the design of new operating systems, middleware, and other system software laying out how system software tasks can be constructed as to minimize impact on HPC applications.

Most recently, Hoefler et al. [9] used a LogGOPS-based simulator [10] to analyze the influence of OS noise on HPC applications. The simulator is able to simulate micro-benchmarks with up to one million ranks and full applications up to 32 thousand ranks. Using this simulator, the authors were able to reproduce the results of much of the previous work in OS noise. In addition, this simulator showed that noise will continue to limit the scalability of future HPC systems that do not utilize noise mitigation techniques.

A number of studies have been conducted regarding the implementation and performance of blocking and non-blocking collective implementations [20, 2, 22, 24]. Most notably, Hoefler et al. [8] described the implementation of the non-blocking collective library currently being considered for inclusion in the MPI-3 standard. In this paper, the authors showed that the performance benefit of non-blocking collectives is related to the ability of the system to overlap the communication cost of messages with computation of the application. In addition, this work outlined the importance of intelligent network interfaces like the SeaStar on ensuring independent network progress for HPC systems.

Finally, Alam and Vetter [1] characterized the system balance requirements for GYRO, a Office of Science fusion simulation code, and the POP climate modeling code investigated here. In this work, the authors measured the parallel efficiencies for these applications on a number of parallel systems: an SMP cluster, a shared-memory system, and a vector supercomputer. This work showed the sensitivity of POP to MPI latency and the bandwidth sensitivity of GYRO. Similarly, Pedretti et al. [18] investigated the sensitivity of HPC applications to link and injection bandwidth on the Cray Red Storm machine. Using similar hardware methods employed in this paper, the authors showed the sensitivity of CTH and PARTISN to link bandwidth and injection bandwidth (the bandwidth of the point-to-point HyperTransport link connecting a compute node's Opteron CPU to its SeaStar network interface) degradation.

6 Summary

In this paper, we showed how a number of important architectural and system software design features affect the impact of noise on HPC systems. Results indicate that these system characteristics have a significant impact on the performance of applications at scale.

We used a previously-developed, kernel-based noise injection utility to explore several important aspects of OS interference. We showed that the impact of noise is not solely a property of the communication characteristics of an application. Using a hardware mechanism to degrade network bandwidth performance, we showed that the relative peak compute-to-communication ratio of the system is also important. This particular analysis helps to explain the disparity in observed results of the impact of noise on systems with disparate balance characteristics. Our results show that, in general, systems with excess compute cycles tend to be less sensitive to noise.

In addition, we explored whether isolating noise to only a subset of nodes can lessen performance degradation. We used our noise injection tool to impact only a subset of compute nodes rather than affecting all nodes equally. Results showed that it takes a relatively small percentage of nodes – even as little as 5% – to have a significant impact on application performance. We also explored the distribution of the noisy nodes to determine whether placement makes any difference. Our results showed that placement of noisy nodes can also have a substantial impact on application performance. If noise is generated on nodes whose MPI rank is closer to rank zero, the impact of noise is much less than if the noise is generated on a subset of ranks further from rank zero. We validate our hypothesis that this rank distribution effect is the result of tree-based collective operations where the ranks nearest the root are able to more easily absorb noise, while nodes furthest from the root are not.

Also, we illustrated the performance of two collective algorithms in noisy environments We implemented a linear and binomial-tree based `MPI_Allreduce` collective operation. Our experiments revealed that, while the tree-based collective algorithm greatly outperforms the linear algorithm in a low noise environment, in a very noisy environment, the linear algorithm can absorb much of the injected noise and therefore outperform the tree-based implementation.

Finally, we investigated the ability of non-blocking collective operations to mitigate the impact of OS noise. We implemented a non-blocking Allreduce operation and a corresponding overlap benchmark. Combined with the noise injection utility, we were able to explore the amount of noise that a non-blocking collective operation could potentially absorb. Results showed that, for a typical noise signature, a relatively modest amount of overlap between computation and communication is enough to nearly eliminate the impact of noise.

Together, these results increase our understanding of how and why OS noise impacts applications. Deeper knowledge about the important characteristics of noise-sensitive applications and key system or architectural features that can mitigate the negative impact of noise. This knowledge greatly enhances the ability to design future-generation platforms, system software, and applications.

7 Future Work

There are several avenues of future work related to this study. First, we intend to analyze more applications in order to increase the understanding of application sensitivity to noise. While the set of applications in this study covers a range of important problems and scalable computational techniques, additional application experimentation would further increase the understanding of the relationship between OS noise and HPC hardware and system software design features. Obtaining access to large-scale applications, problem sets, appropriate application scientist expertise, and dedicated system time to run these applications has proved challenging, but we believe that this approach is key to understanding the overall impact of OS noise, especially as future programming models may require additional system services with additional system demands.

We are also interested in analyzing how basic OS services, for example memory management, can influence the generation and impact of noise. We are exploring modifications to our noise framework that allow Catamount's memory management strategy to be more representative of a general-purpose OS like Linux. Specifically, we are implementing a non-contiguous memory page allocation scheme that better resembles the method used by Linux to allocate and manage physical memory pages. There is evidence to suggest that these memory management strategies can significantly influence the scalability of certain HPC applications [21].

Acknowledgements The authors gratefully acknowledge a number of associates from Sandia National Laboratories for their assistance in this work. We thank Sue Kelly, Bob Ballance, and the entire Red Storm support staff for their tireless support during our dedicated system time. We also wish to thank Courtenay Vaughan for his help in configuring our three representative HPC applications. Lastly, we wish to thank the US Department of Energy's Office of Advanced Scientific Computing Research for their financial support of this work.

References

1. S. R. Alam and J. S. Vetter. An analysis of system balance requirements for scientific applications. In *ICPP '06: Proceedings of the 2006 International Conference on Parallel Processing*, pages 229–236, Washington, DC, USA, 2006. IEEE Computer Society.
2. G. Almási, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng. Optimization of MPI collective communication on BlueGene/L systems. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 253–262, New York, NY, USA, 2005. ACM.
3. P. Beckman, K. Iskra, K. Yoshii, and S. Coghlan. The influence of operating systems on the performance of collective operations at extreme scale. In *IEEE Conference on Cluster Computing*, September 2006.
4. R. Brightwell, T. Hudson, K. T. Pedretti, and K. D. Underwood. SeaStar Interconnect: Balanced bandwidth for scalable performance. *IEEE Micro*, 26(3):41–57, May/June 2006.
5. R. Durstenfeld. Algorithm 235: Random permutation. *Commun. ACM*, 7(7):420, 1964.
6. J. E.S. Hertel, R. Bell, M. Elrick, A. Farnsworth, G. Kerley, J. McGlaun, S. Petney, S. Silling, P. Taylor, and L. Yarrington. CTH: A software family for multi-dimensional shock physics analysis. In *Proceedings of the 19th International Symposium on Shock Waves, held at Marseille, France*, pages 377–382, July 1993.
7. K. B. Ferreira, R. Brightwell, and P. G. Bridges. Characterizing application sensitivity to OS interference using kernel-level noise injection. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (Supercomputing'08)*, November 2008.
8. T. Hoefler, A. Lumsdaine, and W. Rehm. Implementation and performance analysis of non-blocking collective operations for MPI. In *In proceedings of the 2007 International Conference on High Performance Computing, Networking, Storage and Analysis, SC07*. IEEE Computer Society/ACM, Nov. 2007.
9. T. Hoefler, T. Schneider, and A. Lumsdaine. Characterizing the influence of system noise on large-scale applications by simulation. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, Nov. 2010.
10. T. Hoefler, T. Schneider, and A. Lumsdaine. Loggopsim - simulating large-scale applications in the LogGOPS model. Jun. 2010. Accepted at the ACM Workshop on Large-Scale System and Application Performance (LSAP 2010).
11. T. Jones, W. Tuel, L. Brenner, J. Fier, P. Caffrey, S. Dawson, R. Neely, R. Blackmore, B. Maskell, P. Tomlinson, and M. Roberts. Improving the scalability of parallel jobs by adding parallel awareness to the operating system. In *Proceedings of SC'03*, 2003.
12. D. Katramatos, S. J. Chapin, P. Hillman, L. A. Fisk, and D. van Dresser. Cross-operating system process migration on a massively parallel processor. Technical Report CS-98-28, University of Virginia, 1998.
13. D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings. Predictive performance and scalability modeling of a large-scale application. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pages 37–48, Denver, CO, 2001. ACM Press.
14. D. J. Kerbyson and P. W. Jones. A performance model of the Parallel Ocean Program. *Int. J. High Perform. Comput. Appl.*, 19(3):261–276, 2005.

15. P. D. V. Mann and U. Mittaly. Handling OS jitter on multicore multithreaded systems. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–12, Washington, DC, USA, 2009. IEEE Computer Society.
16. J. Moreira, M. Brutman, J. Castanos, T. Gooding, T. Inglett, D. Lieber, P. McCarthy, M. Mundy, J. Parker, B. Wallenfelt, M. Giampapa, T. Engelsiepen, and R. Haskin. Designing a highly-scalable operating system: The Blue Gene/L story. In *Proceedings of the 2006 ACM/IEEE International Conference for High-Performance Computing, Networking, Storage, and Analysis (SC'06)*, Tampa, Florida, November 2006.
17. A. Nataraj, A. Morris, A. D. Malony, M. Sottile, and P. Beckman. The ghost in the machine: Observing the effects of kernel operation on parallel application performance. In *Proceedings of SC'07*, 2007.
18. K. T. Pedretti, C. Vaughan, K. S. Hemmert, and B. Barrett. Application sensitivity to link and injection bandwidth on a Cray XT4 system. In *Proceedings of the 2008 Cray User Group Annual Technical Conference*, May 2008.
19. F. Petrini, D. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Proceedings of the International Conference on High-Performance Computing and Networking*, Phoenix, AZ, 2003.
20. J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. Dongarra. Performance analysis of MPI collective operations. *Cluster Computing*, 10(2):127–143, 2007.
21. B. V. Straalen, J. Shalf, T. Ligocki, N. Keen, and W.-S. Yan. Scalability challenges for massively parallel AMR applications. In *Proceedings of the International Parallel and Distributed Processing Symposium*, May 2009.
22. R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications*, 19:49–66, 2005.
23. R. Zajcew, P. Roy, D. Black, C. Peak, P. Guedes, B. Kemp, J. LoVerso, M. Leibensperger, M. Barnett, F. Rabii, and D. Netterwala. An OSF/1 UNIX for Massively Parallel Multicomputers. In *Proceedings of the 1993 Winter USENIX Technical Conference*, pages 449–468, January 1993.
24. H. Zhu, D. Goodell, W. i. Gropp, and R. Thakur. Hierarchical collectives in MPICH2. In *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 325–326, Berlin, Heidelberg, 2009. Springer-Verlag.