Reducing Application Runtime Variability on Jaguar XT5

Sarp Oral Feiyi Wang David A. Dillow Ross Miller
Galen M. Shipman Don Maxwell

Oak Ridge National Laboratory Leadership Computing Facility
{oralhs,fwang2,dillowda,rgmiller,gshipman,maxwellde}@ornl.gov

Dave Henseler Jeff Becklehimer Jeff Larkin

Cray Inc.
{dah,jlbeck,larkin}@cray.com

1 Abstract

Operating system (OS) noise is defined as interference generated by the OS that prevents a compute core from performing "useful" work. Compute node kernel daemons, network interfaces, and other OS related services are major sources of such interference. This interference on individual compute cores can vary in duration and frequency, and can cause de-synchronization (jitter) in collective communication tasks and thus results in variable (degraded) overall parallel application performance. This behavior is more observable in large-scale applications using certain types of collective communication primitives, such as MPI_Allreduce.

This paper presents our effort towards reducing the overall effect of OS noise on our large-scale parallel applications. Our tests were performed on the quad-core Jaguar, the Cray XT5 at the Oak Ridge National Laboratory Leadership Computing Facility (OLCF). At the time of these tests, Jaguar was a 1.4 PFLOPS supercomputer with 149,504 compute cores and 8 cores per node. We aggregated OS noise sources onto a single core for each node. The scientific application was then run on six of the remaining cores in each node. Our results show that we were able to improve the MPI_Allreduce performance by two orders of magnitude. We demonstrated up to a 30% boost in the performance of the Parallel Ocean Program (POP) using this technique.

2 Introduction

OS noise is defined as interference generated by OS that prevents the CPU from performing "useful" work. This interference can vary in duration and frequency and can cause de-synchronization (jitter) in collective communication tasks. Recent research [3, 5, 7, 8, 11, 12, 13, 14] showed that OS noise can seriously vary and degrade overall parallel application performance. This behavior is more observable in large-scale applications using certain types of collective communication primitives, such as MPI_Allreduce.

Petrini et al. [11] described efforts towards identifying performance bottlenecks for the SAGE application on a then large-scale (8,192 nodes) machine. The modeled and observed performance for the SAGE on the ASCI Q supercomputer did not match. They established a correlation between the frequency (low/high) and duration (short/long) of the noise and application's granularity (fine/coarse). They concluded that fine-grained applications are more susceptible to high-frequency short-duration noise, while coarse-grained applications are more susceptible to low-frequency long-duration noise. They found that many MPI collective primitives are highly susceptible to OS noise, MPI_Allreduce in particular.

Agrawal et al. [2] studied the OS noise problem theoratically and concluded that it can impede the parallel performance drastically for applications heavily using collective communication primitives. They further stated that this negative impact is more pronounced for heavy-tailed and Bernoulli noise distributions.

Ferreira et al. [7] stated that micro-benchmarking the impact of noise on collective communication performance does not necessarily correlate with the impact of noise on application performance. They introduced a new classification for applications based on their OS noise responses as *absorbing* and *amplifying/accumulating*. Using POP,

SAGE, and CTH, they concluded that some collective primitives are OS noise absorbing (e.g. MPI_Wait), while some others are accumulating (e.g. MPI_Allreduce). They also observed that unloaded system noise presents different noise characteristics than loaded system noise. An unloaded system is defined as a quiet system (i.e. no application is running). They stated that the signature of unloaded system noise is realistic (e.g. application generated I/O interrupts are not OS noise sources as such are not included in the unloaded system noise profile). The authors concluded that POP spends most of its time in collective communications – up to 70% for 512 nodes – and that MPI_Allreduce is the dominant consumer of time for MPI primitives. In other words, MPI_Allreduce has been shown to have an OS noise *accumulating* effect on application performance. It was also shown that the smaller MPI_Allreduce message size, the more susceptible to low-frequency high-duration noise the primitive becomes.

Beckman et al. [4] also investigated the OS noise problem on large-scale platforms and suggested that using a global clock to synchronize timer interrupts or using a tickless configuration for the kernel can provide even higher levels of synchronization.

Beckman et al. [3] stated that duration of the noise interferance is important and they need to be quite large in order to significantly impact performance on extreme-scale architectures. Authors concluded that unless extra processes or interrupt processing dramatically desynchronizes a Linux cluster, OS noise does not cause significant performance degradation.

On Jaguar XT5, we identified the OS noise problem due to varying and degraded parallel application performance. This was especially true for the Parallel Ocean Program (POP) application [9]. As stated, POP heavily uses the MPI_Allreduce parallel communication primitive. We measured the effect of OS noise on our parallel applications and implemented a prototype solution to alleviate the performance degradation. We identified noise sources, aggregated them to a specific core on each compute node, and ran applications on select remaining cores.

Section 3 describes our methodology and prototype Reduced Noise kernel design. Section 4 presents our test and discusses results. Finally, Section 5 presents our conclusions.

3 Prototype Reduced Noise Kernel Design

We observed highly variable application performance on the Jaguar XT5 system, and worked to identify the source. Testing demonstrated very high levels of operating system activity on the compute cores of Jaguar when compared to other specialized HPC platforms. These high levels of activity were present regardless of the location of the node on the 3D torus network, and frequently interrupted the CPU, preventing the application from doing useful work.

Collaboration with Cray identified several sources of noise in the CLE architecture. Many sources come in the guise of interrupts, timer events, and related activities:

- TCP/IP protocol processing
- Time-of-Day clock maintenance
- Kernel work queues
- Non-fatal machine checks
- Flushing dirty data from page cache
- DVS protocol handling (read-ahead)
- Lustre protocol handling (lock and RPC timeouts)
- BEER helper threads for network reliability
- Virtual-to-physical mapping for received packets
- Other generic timer events

In addition, the CLE infrastructure has noise sources that live in userspace:

- Application Level Placement Scheduler (ALPS) daemons
- RCA (heartbeat, console)
- SSH (admin logins)
- NTP (time synchronization)

To reduce the effect of these noise sources, operating system services were largely moved to the first CPU – core 0 – on each node. Application processes can then run on the remaining cores with less system interference. This technique to reduce noise is not new, and was previously implemented on the Intel Paragon [6].

Cray prototyped this in the UNICOS 2.2 Reduced Noise kernel. The noise reduction is selectable on a per-job basis, and is selected by specifying that applications are restricted to cores 1 to 7 on each node – `aprun -cc 1-7 -N 7`. Core 0 is then reserved for system activities.

In the Reduced Noise kernel, nearly every service listed above is pinned to core 0. However, Lustre/DVS processing and the mapping of incoming packets are not restricted, as most of these interrupts are in response to traffic generated by the application and are not considered noise. Additionally, there is substantial performance benefit from spreading this work among all available cores.

Although it may seem paradoxical to sacrifice cores to gain more performance, as these "overhead cores" do not

participate in with the pool of "application cores" in synchronization, they do not cause an overall performance degradation for applications that are sensitive to OS noise. However, applications that do not make heavy use of collective communications – such as embarrassingly parallel applications – or use absorbing collective tasks – MPI_Wait for example – may see degraded performance due to the lost computational power.

4 Experiments

We targeted our intial testing at understanding the OS noise patterns of various computing platforms. We used the FTQ/FWQ benchmark [1, 10] to profile the systems in the OLCF center.

Kurtosis is one of the statistical measures used in these micro-benchmak programs to broadly assess the *noise level* of a system. In a nutshell, this number measures the "peakedness" of a given distribution. A pronounced or a relatively high peak in a distribution results in a high kurtosis. The following formula can be used for calculating kurtosis for a variable x, with x_i representing individual data points:

$$\text{kurtosis} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^4}{(n-1)s^4} \qquad (1)$$

Basically, the closer a given distribution to the normal distribution, lower the kurtosis becomes.

We compared an IBM BG/P (single socket quad-core 850 MHz CPU), a Cray XT4 (dual-socket dual-core 2.1 GHz CPU), a regular Linux cluster (quad-socket quad-core 2.0 GHz CPU), "Chester" a single cabinet Cray XT5 (dual-socket quad-core 2.4 GHz CPU), and Jaguar XT5. Our results showed that the IBM BG/P platform is very quiet and the cores behave similarly, if not identically, to each another, exhibiting a uniform noise pattern. On Cray XT4 system, we observed that while every core was quite noisy, core 0 had the highest levels of noise. The Linux cluster was the most noisy platform – almost half of the cores were extremely noisy. Both XT5 platforms were running the stock UNICOS 2.2 Stock kernel and exhibited slightly better noise levels than the Linux cluster. Chester was more quiet than Jaguar. We believe this difference is due to the difference in scale of the network traffic and interrupts between the XT5 systems. Overall, Cray CNL on XT4/5 was found to be very noisy compared to BG/P. It exhibited similar noise levels as a Linux cluster running an unmodified, off-the-shelf, Linux distribution with a stock kernel.

We then tested the UNICOS 2.2 Reduced Noise kernel on Chester. Figure 1 shows data collected using the FWQ benchmark from IBM BG/P (top), Chester with Stock kernel (middle), and Chester with Reduced Noise (RN) kernel (bottom) platforms, normalized with respect to each platform's CPU clock cycle. The FWQ benchmark measures

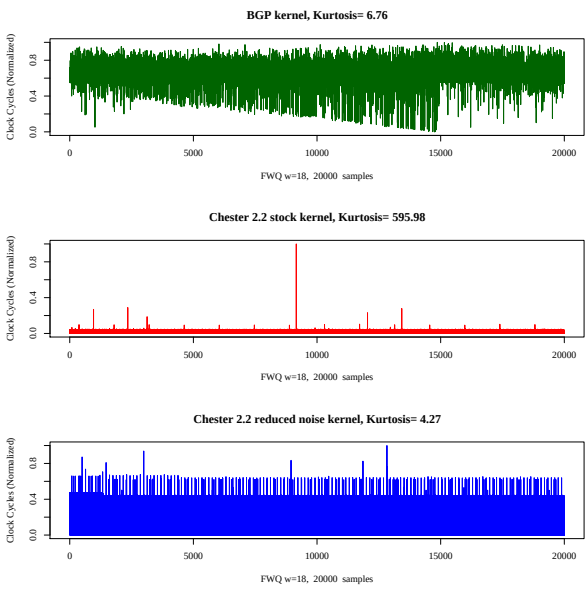


Figure 1. Kurtosis for XT5 (with and without the Reduced Noise kernel) and IBM BG/P platforms

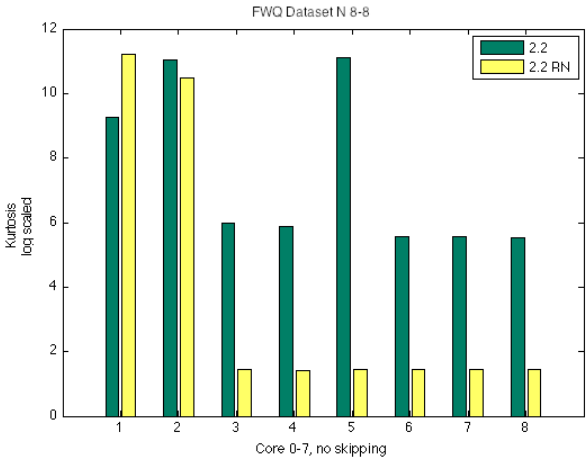
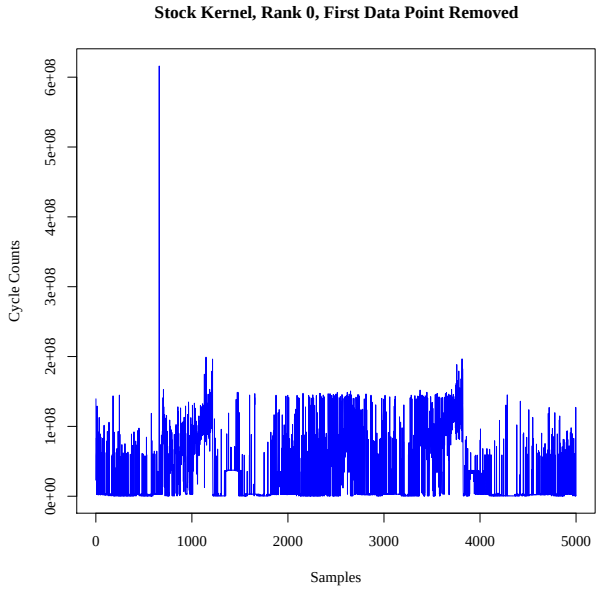
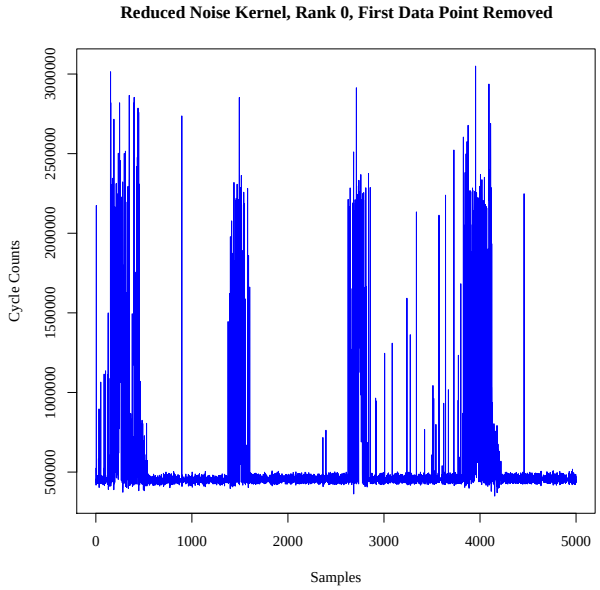


Figure 2. Threaded FWQ results on a XT5 node with 8 cores

the time spent to perform a fixed work quanta per each sample point. Therefore, a "low noise" kernel should exhibit a uniform FWQ sampling (in terms of amplitude and frequency), which means at each sample the CPU was interrupted more or less for the same amount of time – since all CPUs in the system will be interrupted in a roughly similar manner, the overall syncronization should not be greatly disturbed. The benchmark was ran for a w value of 18, and



(a) Rank 0 with Stock kernel



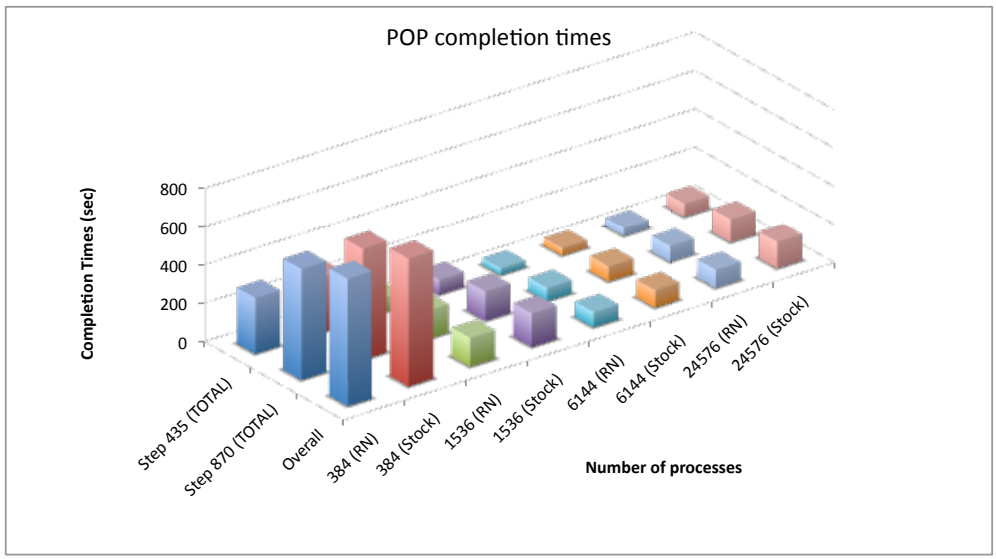
(b) Rank 0 with Reduced Noise kernel

Figure 3. MPI-FWQ completion times for Stock and Reduced Noise kernels

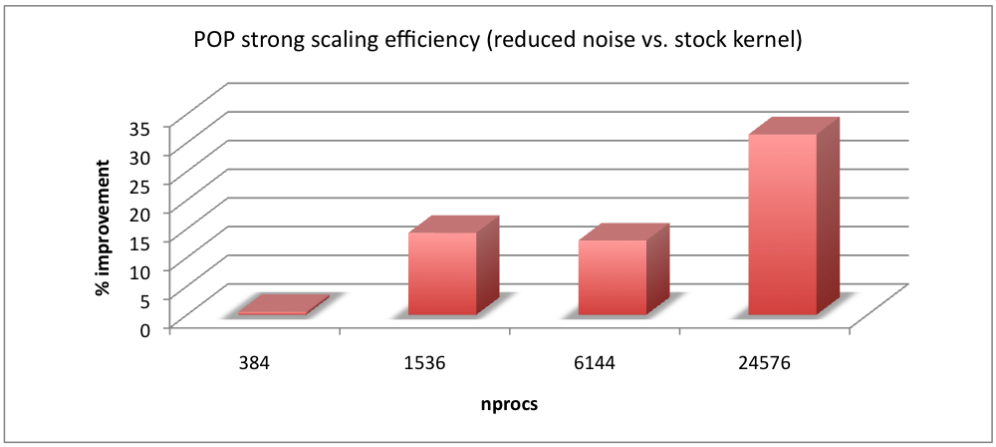
2,000 data samples were collected.

Kurtosis for each platform was then calculated from the FWQ data. Figure 1 also shows calculated kurtosis values for these three platforms. Based on the kurtosis values – 6.76, 595.98, and 4.27, for IBM BG/P, Chester with Stock kernel, and Chester with (RN) kernel, respectively – the IBM BG/P system provides a more uniform OS noise distribution compared to the XT5 system running a 2.2 stock kernel. However, the OS noise characteristics for the XT5 platform improved drastically with the 2.2 Reduced Noise kernel. Kurtosis was significantly reduced to 4.27.

A further look at the FWQ benchmark on a single XT5 node revealed an interesting behavior, shown in Figure 2. The results suggested that although noise levels on cores



(a) POP completion times



(b) Reduced Noise kernel efficiency

Figure 4. POP completion times and efficiency with Reduced Noise kernel

2-7 were significantly reduced with the RN kernel, kurtosis for cores 0 and 1 were similar to each other and they were four orders of magnitude higher than the remaining cores. Core 1's behavior was not expected at the time. Thus, we restricted our remaining tests to six cores (cores 2-7) to avoid masking any improvements with inconsistent behavior while using the prototype UNICOS 2.2 Reduced Noise kernel[1].

Tests were performed at scale on Jaguar XT5 in the summer of 2009, when Jaguar was in a quad-core configuration. We first ran the *MPI-FWQ* micro-benchmark to measure the performance of the MPI_Allreduce primitive. MPI-FWQ is an in-house code that combines the "work" segment of the FWQ benchmark with a user selectable MPI collective communication task. Each process executes a single threaded FWQ work quanta, and then measures the time required to complete the specified MPI collective task.

We ran MPI-FWQ with a w value of 18 and MPI_Allreduce as the collective task for 49,152 cores. Message size per task was 1 MB and rank 0 was the root of the collective. We used 6 cores out of the 8 available on a node, skipping cores 0 and 1 (`-N 6 -cc 2-7`). Figure 3 shows completion times at the root of the collective as the number of CPU cycles for the Stock kernel (Figure 3a) and the Reduced Noise kernel (Figure 3b). The first data point from each data set is discarded as it includes the overhead of starting the test. As can be seen, per iteration completion times is reduced on average by two orders of magnitude when running on the Reduced Noise kernel.

[1]This anomaly has since been tracked down to certain interrupts being inadvertantly pinned to core 1. This is fixed for the production Reduced Noise kernel.

Table 1. POP comparision for UNICOS 2.2 Reduced Noise and Stock kernels on OLCF's Jaguar. Step times are given in seconds and total run was for 1,000 steps.

Number of Processes	Reduced Noise kernel			Stock kernel		
	Step 435	Step 870	Step 1,000	Step 435	Step 870	Step 1,000
384	289.68	575.48	660.03	291	578.09	663.13
1,536	75.27	149.16	149.16	77.46	151.94	173.98
6,144	35.33	69.17	79.13	39.17	79.25	90.89
24,576	42.7	81.78	94.58	68.43	122.79	137.94

Table 2. POP comparision for UNICOS 2.2 Reduced Noise and Stock kernels on Cray's Shark. Step times are given in seconds and total runs were for 2,000 steps for both Reduced Noise and Stock kernels.

	Number of Processes	Step 2,000
Reduced Noise	7,168	379.03
Stock	8,192	499.00

We then ran POP at 384, 1,536, 6,144, and 24,576 processes to observe the strong scaling performance when running on the Reduced Noise kernel, and reran the same tests under the Stock kernel. For each test we used 6 cores out of the 8 available on a node, skipping cores 0 and 1 (`-N 6 -cc 2-7`). Our total mesh size was 3072 by 2048, *nx_global* and *ny_global*, respectively. Our block size (*nx_block, ny_block*) per process was 132 by 132, 68 by 68, 36 by 36, and 20 by 20 for 384, 1,536, 6,144, and 24,576, respectively.The *max_blocks_clinic* and *max_blocks_tropic* were set to 1. For each test the *stop_option* was set as *nstep* and we had 1,000 steps per run. We ran with balance for the baroclinic distribution, and cartesian for the barotropic distribution. The options for history, movie, tavg, and xdisply options were disabled. We maintained the same options for both OS kernels. Table 1 shows completion times for steps 435, 870, and 1,000 (total) for 384, 1,536, 6,144, and 24,576 processes for both UNICOS 2.2 Reduced Noise and Stock kernels.

Figure 4 plots the POP results given in Table 1. Figure 4a shows the POP completion times for each test configuration and POP step. Figure 4b shows the overall run time efficiency achieved with the UNICOS 2.2 Reduced Noise kernel. Our strong scaling results show that we realized a performance improvement of over 30% on the largest scale POP run tested (24,576 cores).

A second application test was conducted at Cray's facilities. POP was run on "Shark", a 12 cabinet XT5 system with 1065 dual-socket, quad-core nodes running at 2.4 GHz. This test was run with a mesh of 3584 by 2240, and 2000 steps per run. The block size per process was 32 by 44, and the simulation was run for 2000 steps. POP was run with 8192 processes on the Stock kernel to maximize processing power (`-N 8`), and with 7168 processes on the RN kernel to minimize OS noise (`-N 7 -cc 1-7`). In both cases, 1024 nodes were used for the computation; only the number of cores used on each node was varied. Even with the reduced computing capacity used for the RN kernel run, Table 2 shows a similar performance gain as on the Jaguar XT5 tests.

5 Conclusions

Operating system (OS) noise is a key limiting factor for large-scale parallel application performance. Interrupt sources for timers and network interfaces, kernel daemons, and other related OS services are major sources of OS noise interference. This noise can cause desynchronization (jitter) in collective communication tasks such as MPI_Allreduce.

We identified a major parallel application performance degredation on our Cray XT5 platform. Our tests indicated OS noise was the source of the problem, and we prototyped a Reduced Noise kernel for the XT5. This prototype kernel aggregated most OS noise sources onto a specific core on each compute node. It also provided a user controllable mechanism to prevent the scientific application from running on this core. Our results show that we were able to improve the performace of MPI_Allreduce by two orders of magnitude. We demonstrated up to a 30% boost in the performance of the Parallel Ocean Program (POP).

Acknowledgements

The authors would like to thank the staff and colleagues who have contributed material to this paper. Authors also would like to express their thanks and gratitude to George Ostrouchov, Jeff Kuhen, Terry Jones, Collin McCurdy, and

Vinod Tipparaju for their OS noise related ideas, discussions, and comments and to Jim Rosinski for his help with understanding and configuring POP application during test runs.

Research sponsored by the Mathematical, Information, and Computational Sciences Division, Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

References

[1] Advanced Simulation and Computing, Lawrence Livermore National Laboratory. ASC Sequoia Benchmark Codes. `https://asc.llnl.gov/sequoia/benchmarks/#ftq`, 2008.

[2] Saurabh Agarwal, Rahul Garg, and Nisheeth Vishnoi. The Impact of Noise on the Scaling of Collectives: A Theoretical Approach. In *Lecture Notes in Computer Science, High Performance Computing HiPC 2005*, pages 280–289. Springer Berlin/Heidelberg, 2005. ISBN 978-3-540-30936-9.

[3] Pete Beckman, Kamil Iskra, Kazutomo Yoshii, and Susan Coghlan. The Influence of Operating Systems on the Performance of Collective Operations at Extreme Scale. In *IEEE International Conference on Cluster Computing*, pages 1–12, 2006.

[4] Pete Beckman, Kamil Iskra, Kazutomo Yoshii, and Susan Coghlan. Operating system issues for petascale systems. *SIGOPS Oper. Syst. Rev.*, 40(2):29–33, 2006. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/1131322.1131332.

[5] Pete Beckman, Kamil Iskra, Kazutomo Yoshii, Susan Coghlan, and Aroon Nataraj. Benchmarking the effects of operating system interference on extreme-scale parallel machines. *Cluster Computing*, 11(1):3–16, 2008. ISSN 1386-7857. doi: http://dx.doi.org/10.1007/s10586-007-0047-2.

[6] T.H. Dunigan. Early Experiences and Performance of the Intel Paragon. Technical Report ORNL/TM-12194, Oak Ridge National Laboratory, 1993.

[7] Kurt B. Ferreira, Patrick Bridges, and Ron Brightwell. Characterizing application sensitivity to os interference using kernel-level noise injection. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press. ISBN 978-1-4244-2835-9. doi: http://doi.acm.org/10.1145/1413370.1413390.

[8] T.R. Jones, L.B. Brenner, and J.M. Fier. Impacts of Operating Systems on the Scalibility of Parallel Applications. Technical Report UCRL-MI-202629, Lawrence Livermore National Laboratory, 2003.

[9] Darren J. Kerbyson and Philip W. Jones. A Performance Model of the Parallel Ocean Program. *International Journal of High Performance Computing Applications*, 19:261–276, 2005.

[10] Collin B. McCurdy and Jeffrey S. Vetter. Understanding the Behavior of the FWQ System Noise Microbenchmark. Technical Report FTGTR-2008-10, Oak Ridge National Laboratory, Future Technologies Group, 2008.

[11] Fabrizio Petrini, Darren J. Kerbyson, and Scott Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 55, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 1-58113-695-1.

[12] D. Skinner and W. Kramer. Understanding the causes of performance variability in HPC workloads. In *Proceedings of the IEEE International Workload Characterization Symposium*, pages 137–149, 2005.

[13] William T, C. Kramer, and Clint Ryan. Performance Variability of Highly Parallel Architectures. In *Computational Science ICCS 2003*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003.

[14] Dan Tsafrir, Yoav Etsion, Dror G. Feitelson, and Scott Kirkpatrick. System noise, os clock ticks, and fine-grained parallel applications. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 303–312, New York, NY, USA, 2005. ACM. ISBN 1-59593-167-8. doi: http://doi.acm.org/10.1145/1088149.1088190.