Exploring the effect of noise on the performance benefit of nonblocking allreduce

Patrick Widener
Kurt B. Ferreira
Sandia National Laboratories*
Albuquerque, NM

Scott Levy
University of New Mexico
Albuquerque, NM

Torsten Hoefler
ETH Zurich
Zurich, Switzerland

ABSTRACT

Relaxed synchronization offers the potential of maintaining application scalability by allowing many processes to make independent progress when some processes suffer delays. Yet, the benefits of this approach in important parallel workloads have not been investigated in detail. In this paper, we use a validated simulation approach to explore the noise mitigation effects of nonblocking allreduce in workloads where allreduce is a major contributor to total execution time. Although a nonblocking allreduce is unlikely to provide significant benefit to applications in the low-OS-noise environments expected in next-generation HPC systems, we show that it can potentially improve application runtime with respect to other noise types.

Keywords

Collective operations; OS noise; Nonblocking collectives

1. INTRODUCTION

Nonblocking collective operations [13, 17], newly introduced in MPI-3.0 [21], allow application programmers to overlap collective communication with the application's computation. At scale, blocking collective operations can significantly degrade application performance because, in most cases, each application process must participate in the collective before any process can make further progress. As a result, a laggard process can slow the progress of all of its peers and consequently the entire application. Nonblocking collectives have the potential to alleviate the impact of process variability by allowing a process to make progress even if its peers are late entering a collective.

Delayed participation in a collective operation can be caused by several phenomena, including load imbalance, fault tolerance activities, and operating system *noise*. While noise exists on most computing platforms, it has a disproportionate effect in HPC systems because the applications that run on these systems tend to be highly synchronized. For many important HPC applications, operating system noise has been shown to have serious consequences for overall application performance [7].

The effects of operating system noise on high-performance computing (HPC) applications have been examined for insights into designing scalable hardware, system software, and applications [1, 22]. Recently, system designers have invested time and effort to reduce or eliminate sources of noise in the OS. These efforts have signficantly reduced the noise introduced by system software on HPC systems; there now exist operating systems that are essentially noiseless (e.g., IBM's CNK) [14, 25]. As a result, the need for nonblocking collectives to address the performance impact of OS noise appears to be waning[1].

Although the impact of OS noise may no longer be a significant issue, there are many potential sources of noise in future extreme-scale systems. For example, fault tolerance is projected to be a significant challenge on future systems. The dominant approach to fault tolerance is checkpoint/restart. Because checkpointing activities deprive the application of CPU cycles, they can be modeled as OS noise [20]. Other trends in extreme-scale system design (e.g., adaptive runtimes, in situ analytics) also have the potential to introduce noise-like events that impact application performance. As a result, even if OS noise is itself no longer an issue, emerging sources of noise mean that nonblocking collectives may still yield a performance benefit on next-generation extreme-scale systems.

In this paper, we investigate the impact of nonblocking collective operations on application performance at scale. In particular, we focus on `MPI_Allreduce()` because, as we will show, it is the dominant communication operation in many important parallel workloads. To examine the potential upside of a nonblocking `MPI_Allreduce()`, we use a validated simulator to compare the standard `MPI_Allreduce()` with an idealized cost-free `MPI_Allreduce()` version. Specifically, we show that:

- `MPI_Allreduce()` is a dominant contributor to application runtime for a set of important parallel workloads;
- in low-noise environments that are typical of current

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly-owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

[1]Other uses for nonblocking collectives include overlapping computation and communication as well as complex synchronization protocols [16].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
EuroMPI/ASIA '14, September 9-12 2014, Kyoto, Japan
Copyright 2014 ACM 978-1-4503-2875-3/14/09 ...$15.00.
http://dx.doi.org/10.1145/2642769.2642786.

and expected future systems, nonblocking allreduce operations are unlikely to provide significant benefits without algorithmic changes; and

- for other sources of noise such as resilience protocol overheads, nonblocking allreduce operations may have a signficant beneficial effect on application performance.

2. APPROACH

Our simulation-based approach is based on capturing and examining the communication structure of applications. This structure reflects synchronization of applications and exposes dependencies through the establishment of *happens-before* relations [19]. This is especially useful for the examination of asynchronous operations because of the possible formation of transitive dependencies between processes which do not communicate directly. For MPI programs, static analysis of communication structure is complicated by the difficulty of both offline message matching [3] and modeling interactions analytically. We use instead a discrete-event simulator and model application communication as events.

Our simulator framework comprises `LogGOPSim` [15] and the tool chain developed by Levy et al. [20]. `LogGOPSim` uses the LogGOPS model, an extension of the well-known LogP model [4], to simulate application traces that contain all exchanged messages and group operations. In this way, `LogGOPSim` reproduces all happens-before dependencies and the transitive closures of all delay chains of the application execution. It can also extrapolate traces from small application runs with p processes to application runs with $k \cdot p$ processes. The extrapolation produces exact communication patterns for all collective communications and approximates point-to-point communications. Noise injection into simulations is done by constructing a time-indexed list of detours and their durations; this list is given as input to the simulator which introduces the described delays into the execution of the simulated application.

`LogGOPSim` and its trace extrapolation features have been validated [14, 15]. The complete tool chain has been validated against experiments and established models [20].

2.1 HPC Workload Descriptions

We present results from the simulation and analysis of a set of workloads. These workloads represent scientific applications that are currently in use and computational kernels thought to be important for future extreme-scale computational science. They include:

- CTH, a multi-material, large deformation, strong shock wave, solid mechanics code [5],
- HPCCG, a conjugate gradient benchmark code that is part of the Mantevo [11] suite of mini-apps,
- LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [23], a classical molecular dynamics code developed at Sandia National Laboratories,
- LULESH, the Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics proxy application [6, 18], used by the Extreme Materials at Extreme Scale codesign center at Los Alamos National Laboratory [6],
- miniFE, the Mantevo [11] finite element mini-application which implements kernels representative of the implicit finite-element suite of mini-apps,
- AMG, an algebraic multigrid solver for linear systems arising from problems on unstructured grids [10].

Taken as a whole, these applications: are typically run (or are projected to be run) at extreme-scale, run for long periods of time, and represent a diverse mix of computations techniques and methods.

3. ALLREDUCE IN HPC APPLICATIONS

Previous investigations [7] have identified `MPI_Allreduce()` and other collectives as operations that are particularly sensitive to OS noise. Exploratory implementations of nonblocking collectives [13] have been developed to address these and other issues. To support our inquiry into the effect of noise on a nonblocking `MPI_Allreduce()`, we quantify the role that `MPI_Allreduce()` plays in our set of applications.

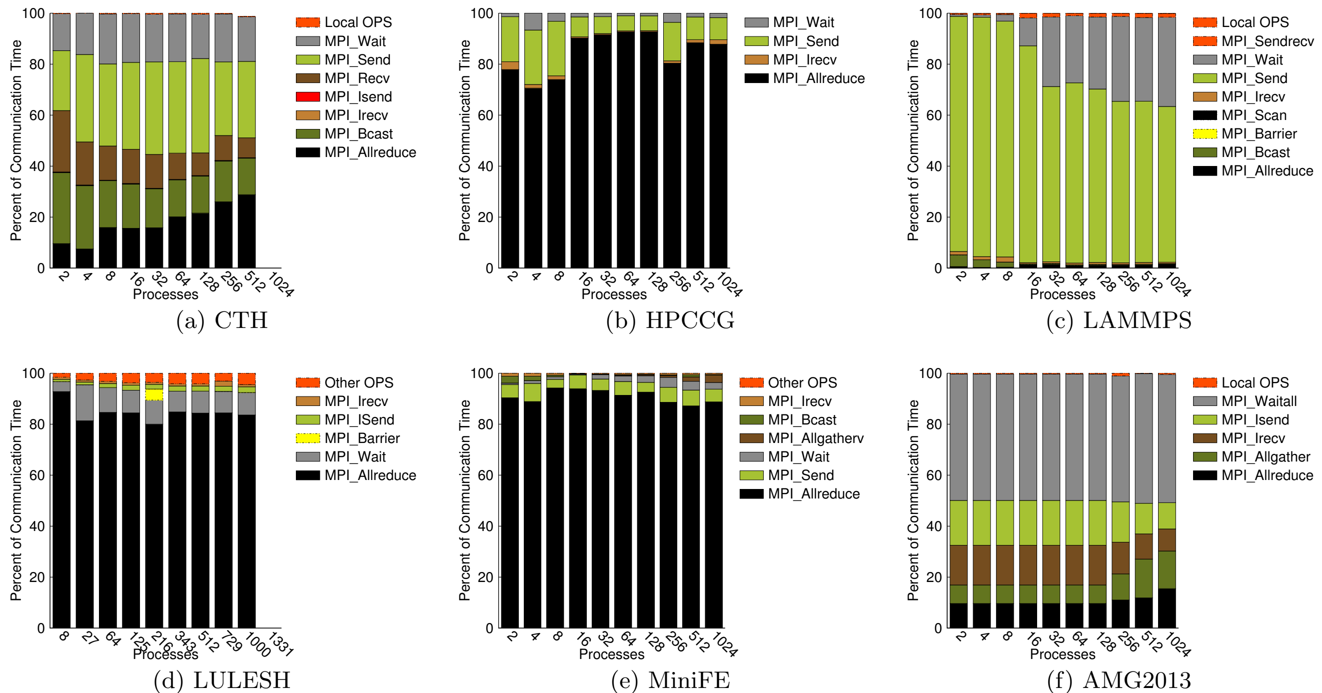
Figure 1 shows the percentage of communication time spent in various MPI functions for each of the applications we consider. These results were collected using the `mpiP` profiling library [24]. The communication time of LULESH and MiniFE is dominated by `MPI_Allreduce()` for process counts up to 1024. Similarly, HPCCG spends more than 80% of its communication time in `MPI_Allreduce()` at almost all the tested process counts. In contrast, CTH spends less than 30% of its time in `MPI_Allreduce()` (although the fraction increases significantly as the size of the application increases), and LAMMPS spends comparatively little time there. These results suggest that a nonblocking `MPI_Allreduce()` implementation might prove beneficial for at least HPCCG, LULESH, and MiniFE.

The amount of communication time is not by itself the only factor when considering the impact of noise. Another relevant perspective is the frequency of collective operations over the execution time of an application (Figure 2). Examining the performance of these applications we see that as execution proceeds, LULESH, HPCCG, and MiniFE perform `MPI_Allreduce()` operations at a high rate. These high amounts of interprocess communication are presumably susceptible to noise-based interference; moreover, any mitigation of noise effects by nonblocking collectives should certainly be observable for these applications.

4. NOISE, DETOURS, AND NONBLOCKING ALLREDUCE

Nonblocking collective operations allow the application to overlap collective communication with computation. As a result, they have the potential to mitigate the performance impact of system noise [7] by allowing the application to absorb noise events. However, it is not clear how much noise absorption nonblocking collectives enable or how much their noise absorption impacts application performance. Therefore, we investigate the performance impact of nonblocking collective using system noise patterns collected on current systems and a synthetic noise pattern we believe to be representative of future extreme-scale systems.

4.1 Characterizing system noise



(a) CTH (b) HPCCG (c) LAMMPS

(d) LULESH (e) MiniFE (f) AMG2013

Figure 1: Percentage of communication time spent in each MPI function for CTH, HPCCG, LAMMPS, LULESH, MiniFE, and AMG.

We measure the noise environment on current HPC systems and consider how might it might differ on future systems. Figure 3 shows the idle system noise patterns collected on three systems using the `selfish` [14] system noise measurement microbenchmark: *Volta*, a Cray XC30; *Muzia*, a Cray XE6; and *RedSky*, a SunBlade x6275 capacity system with Infiniband. Both Cray systems run the Cray Linux Environment (CLE), Cray's optimized Linux kernel, and *RedSky* uses CentOS-based software on the compute nodes. From the figure, we see that *Muzia* has the lowest system noise signature, *RedSky* has a slightly larger degree of noise, and *Volta* has the largest volume of the three. Most importantly, we see from this figure that all of these systems have comparatively low noise signatures, with all noise event durations less than 35 microseconds.

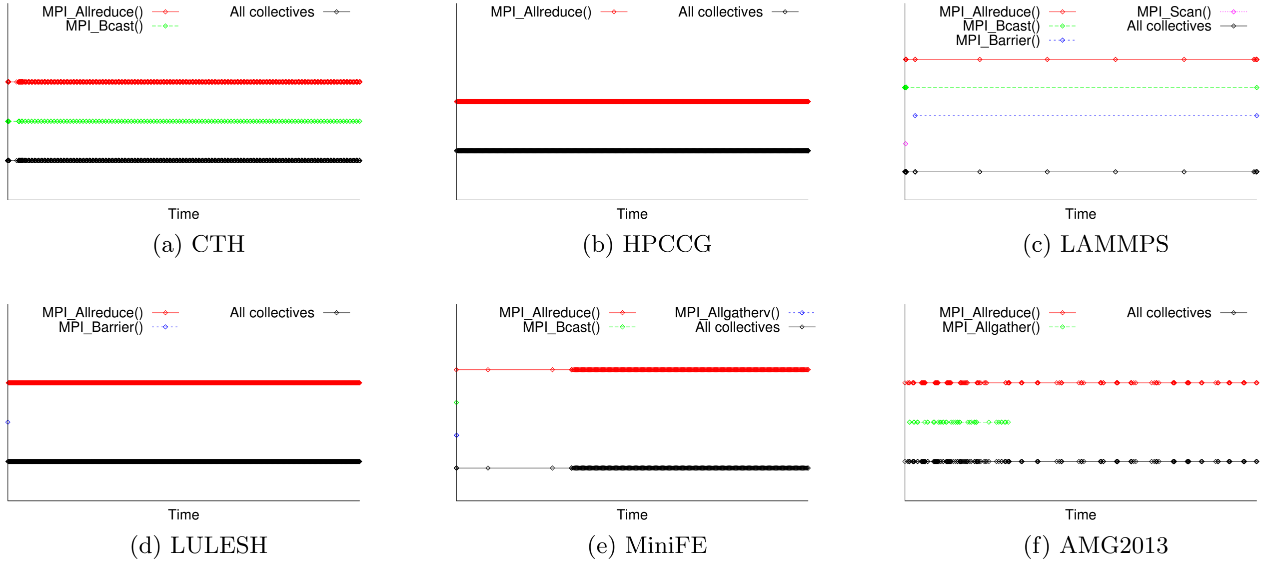
As previous work has shown that system noise events can significantly impact HPC performance [7, 14], great effort has been made to lower the duration of noise events. For example, Cray introduced OS core specialization in the CLE which allows the user to bind CPU cores to OS and system software tasks, thereby reducing the system noise on the application cores. Figure 4 shows the impact of this core specialization functionality on `selfish` noise traces collected on *Volta*, with 0, 1, and four cores dedicated to system software tasks, respectively. Core specialization can greatly "smooth" unpredictable spikes in noise duration, which is of great benefit to applications even if system noise is not notably reduced in general.

4.2 Simulating a nonblocking allreduce

By default, our simulator generates patterns of blocking communications to simulate collective operations. For example, when an `MPI_Allreduce()` is encountered in an application trace, the simulator generates SEND and RECV events using a dissemination algorithm similar to that used by MPICH and OpenMPI. We added to the simulator the ability to select different algorithms for implementing the `MPI_Allreduce()` operation. For this work, we emulate an optimal nonblocking implementation, but do so in a manner that does not require modification of the application. This operation works as follows: the nonblocking allreduce implementation assumes the the application was able to start the operation early enough and has sufficient overlapping progress such that when the `MPI_Allreduce()` operation is reached by a node, it completes immediately and incurs no further overhead. This assumes it is actually possible to re-factor the application in a manner that the nonblocking collectives can be fully utilized (cf. [9]). For some application workloads, this re-factoring may not be possible due to algorithmic details. Therefore, our strategy for nonblocking allreduce provides an optimistic upper bound on the performance speedups that can be realized, but this upper bound is not necessarily tight. In addition to nonblocking allreduce, this work may also be useful in studying systems where collectives are supported by targeted optimizations or specialized networking and hardware, for example.

4.3 Application simulation results

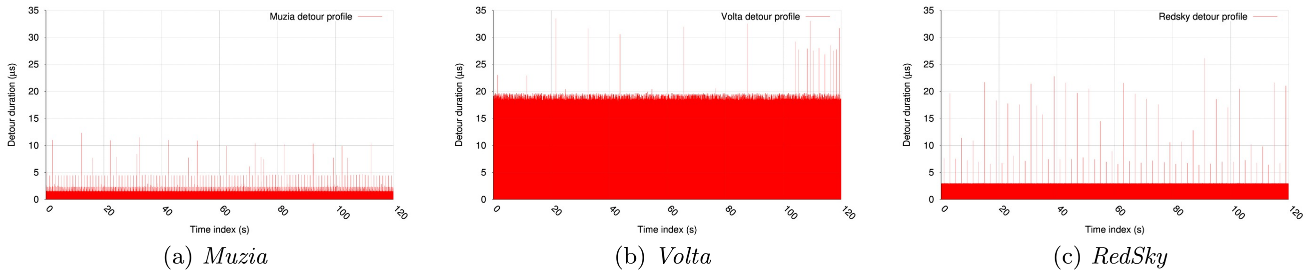
We conducted simulations of our target applications to examine how they would respond in the presence of different types of noise. We used three different noise traces: *system noise*, the worst-case idle system noise signature from *Volta* as found in Figure 4(a); *Daemon noise*, a synthetic noise trace with detour durations of 2.5ms at a frequency of



(a) CTH (b) HPCCG (c) LAMMPS

(d) LULESH (e) MiniFE (f) AMG2013

Figure 2: Frequency of collective operations for CTH, HPCCG, LAMMPS, LULESH, MiniFE, and AMG. For each application, each horizontal line represents either a timeline for a particular collective operation or a combined timeline of all collectives; the y-axis is not significant.



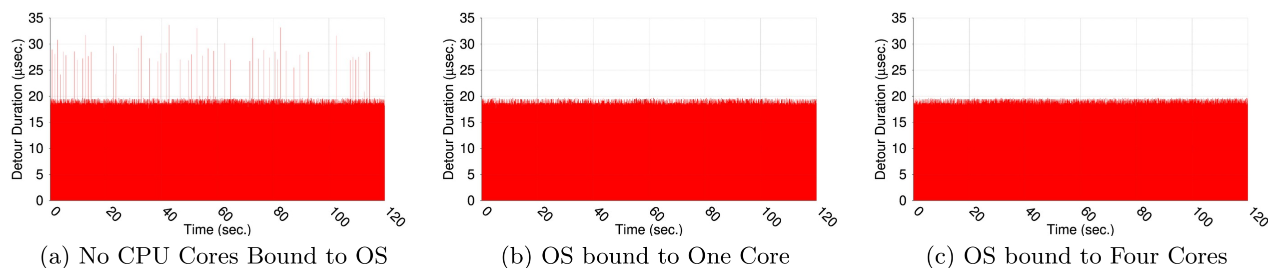
(a) *Muzia* (b) *Volta* (c) *RedSky*

Figure 3: Noise profiles for three Sandia cluster systems collected using the `selfish` detour collection tool.

10 Hz, designed to represent the types of detours that would be introduced by a typical system-level daemon process and similar to one used in previous work [7]; and *asynchronous checkpointing noise*, a 1-second duration, 120-second period application interference pattern similar to processes taking checkpoints in an uncoordinated checkpoint/restart resilience scheme at extreme scale [20]. In each case, we measured the application speedup of our idealized nonblocking `MPI_Allreduce()` implementation when compared to the use of a normal blocking `MPI_Allreduce()`.

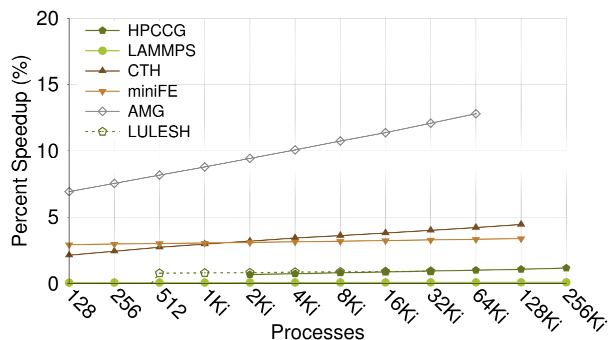
The speedup results are presented in Figure 5. Subfigure 5(a) shows that in the presence of system noise, a nonblocking `MPI_Allreduce()` produces almost no application speedup for any of the studied applications. Subfigure 5(b) shows roughly similar results, with only AMG and CTH showing over 10% speedup as process count increases. The behavior of AMG in these two noise regimes is largely explained by its unique (among this set of applications) pattern of collective inter-arrival times [8], as shown in Subfigure 2(f). These results indicate that a nonblocking `MPI_Allreduce()` will not provide much mitigation effect of system noise to applications, even for heavy allreduce users (as are HPCCG and MiniFE, for example). While it is dangerous to extrapolate this conclusion to other potential nonblocking collective implementations, these results suggest that whatever other benefits they may have for applications, mitigation of system noise is not among them.

Subfigure 5(c) shows a markedly different result, and in the process illustrates an important point about the nature of noise. This figure shows the response of our idealized nonblocking `MPI_Allreduce()` to asynchronous checkpointing noise. This particular type of noise introduces detours with very long durations compared to the OS or daemon noise traces (on the order of seconds as opposed to their millisecond- and microsecond-scale durations). Previous research [7] has shown that noise duration has the greatest impact on application performance, and the implications of this point are reflected in this figure. For many of our target applications, a nonblocking `MPI_Allreduce()` has significant potential for mitigating the effect of asynchronous checkpoint noise. HPCCG and miniFE both show large speedup factors, as does CTH. The outlier here is LAMMPS, which benefits little from this improvement because of its infrequent use of `MPI_Allreduce()`. This result suggests that other

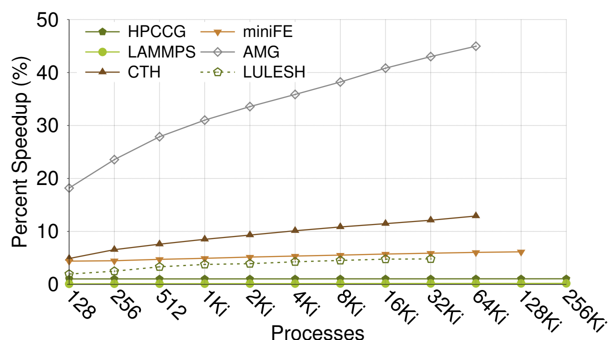


(a) No CPU Cores Bound to OS (b) OS bound to One Core (c) OS bound to Four Cores

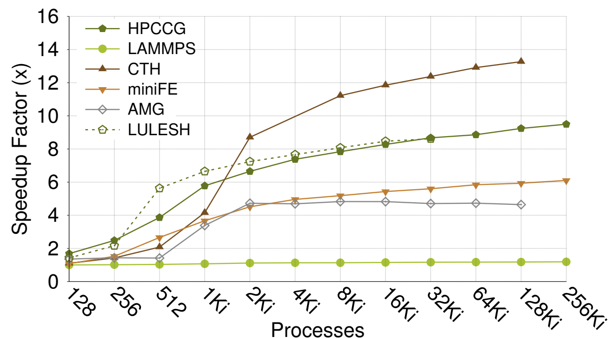
Figure 4: Impact of Cray's OS core specialization on the *Volta* cluster.



(a) System noise



(b) Synthetic noise (10 Hz, 2.5 ms)



(c) Asynchronous checkpointing noise (0.5 Hz, 1 s)

Figure 5: Observed application speedup achieved by switching from standard allreduce to an idealized nonblocking version in the presence of three kinds of noise.

nonblocking collectives may show similar benefits in applications using asynchronous checkpointing.

5. RELATED WORK

Techniques for improving the overlap of communication and computation in MPI applications have been thoroughly explored over the past decade. These techniques include offload [2] and nonblocking collectives. Based on results showing the potential performance benefit of nonblocking collectives [12], Hoefler et al. have argued for the inclusion of nonblocking collectives in the MPI standard [17]. They have also developed a library that implements nonblocking versions of all of the MPI collectives and characterized its impact on application performance [13]. Although the early work only briefly discusses system noise, Hoefler et al. subsequently made the connection between nonblocking collective operations and OS noise explicit [14].

Despite this body of research, our work is the first to examine the relationship between OS noise characteristics and nonblocking collectives. We are also the first to explore how nonblocking collectives could be used to improve the performance of emerging resilience techniques (e.g., uncoordinated checkpointing).

6. CONCLUSION

Nonblocking collective operations are of increasing interest, not only as enablers of continued application scalability but also as potential mitigators of OS noise effects. We have in this paper used a simulation approach to investigate how applications using an idealized nonblocking allreduce operation might respond in the presence of different noise environments, at scales likely to be encountered in future extreme scale systems. Our results indicate that, for noise caused by operating system activity, a nonblocking `MPI_Allreduce()` is unlikely to provide much benefit. We suggest that, by themselves, nonblocking collective operations of other types should not be automatically assumed to be able to mitigate such noise effects. However, we have also shown that the effect of other noise types, such as that caused by checkpoint/restart activity, might be usefully reduced by the introduction of nonblocking collectives. Application designers should consider carefully the characteristics of all the noise sources on their target systems to decide whether refactoring their code to take advantage of nonblocking collectives will be worthwhile.

7. REFERENCES

[1] P. Beckman, K. Iskra, K. Yoshii, S. Coghlan, and A. Nataraj. Benchmarking the effects of operating system interference on extreme-scale parallel machines. *Cluster Computing*, 11(1):3–16, 2008.

[2] R. Brightwell, R. Riesen, and K. D. Underwood. Analyzing the impact of overlap, offload, and independent progress for message passing interface applications. *Intl. Journal of High Performance Computing Applications*, 19(2):103–117, 2005.

[3] G. Bronevetsky. Communication-sensitive static dataflow for parallel message passing applications. In *Proceedings of the 7th annual IEEE/ACM Intl. Symposium on Code Generation and Optimization*, pages 1–12. IEEE Computer Society, 2009.

[4] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Logp: towards a realistic model of parallel computation. *SIGPLAN Not.*, 28(7):1–12, July 1993.

[5] J. E. S. Hertel, R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. McGlaun, S. V. PetneY, S. A. Silling, P. A. Taylor, and L. Yarrington. CTH: A software family for multi-dimensional shock physics analysis. In *Proceedings of the 19th Intl. Symp. on Shock Waves*, pages 377–382, July 1993.

[6] Exascale Co-Design Center for Materials in Extreme Environments (ExMatEx). `http://exmatex.lanl.gov/`. Retrieved 16 Jan 2014.

[7] K. B. Ferreira, P. Bridges, and R. Brightwell. Characterizing application sensitivity to os interference using kernel-level noise injection. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 19. IEEE Press, 2008.

[8] K. B. Ferreira, S. L. Levy, P. M. Widener, D. Arnold, and T. Hoefler. Understanding the effects of communication and coordination on checkpointing at scale. In *Proc. Inernational Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing)*, New Orleans, Louisiana, November 2014. IEEE/ACM. To appear.

[9] P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Computing*, 2013.

[10] V. E. Henson and U. M. Yang. Boomeramg: A parallel algebraic multigrid solver and preconditioner. *Appl. Num. Math.*, 41:155–177, 2002.

[11] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. Improving performance via mini-applications. Technical Report SAND2009-5574, Sandia National Laboratory, 2009.

[12] T. Hoefler, P. Kambadur, R. L. Graham, G. Shipman, and A. Lumsdaine. A case for standard non-blocking collective operations. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 125–134. Springer, 2007.

[13] T. Hoefler, A. Lumsdaine, and W. Rehm. Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI. In *Proc. of the 2007 Intl. Conference on High Performance Computing, Networking, Storage and Analysis, SC07*. IEEE Computer Society/ACM, Nov. 2007.

[14] 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.

[15] T. Hoefler, T. Schneider, and A. Lumsdaine. LogGOPSim - Simulating Large-Scale Applications in the LogGOPS Model. In *Proc. of the 19th ACM International Symp. on High Performance Distributed Computing*, pages 597–604. ACM, Jun. 2010.

[16] T. Hoefler, C. Siebert, and A. Lumsdaine. Scalable Communication Protocols for Dynamic Sparse Data Exchange. In *Proc. of the 2010 ACM Symposium on Principles and Practice of Parallel Programming (PPoPP'10)*, pages 159–168. ACM, Jan. 2010.

[17] T. Hoefler, J. Squyres, G. Bosilca, G. Fagg, A. Lumsdaine, and W. Rehm. Non-Blocking Collective Operations for MPI-2. Technical report, Open Systems Lab, Indiana University, Aug. 2006.

[18] I. Karlin, A. Bhatele, B. L. Chamberlain, J. Cohen, Z. Devito, M. Gokhale, R. Haque, R. Hornung, J. Keasler, D. Laney, E. Luke, S. Lloyd, J. McGraw, R. Neely, D. Richards, M. Schulz, C. H. Still, F. Wang, and D. Wong. Lulesh programming model and performance ports overview. Technical Report LLNL-TR-608824, December 2012.

[19] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[20] S. Levy, B. Topp, K. B. Ferreira, D. Arnold, T. Hoefler, and P. Widener. Using simulation to evaluate the performance of resilience strategies at scale. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2013 SC Companion:*. IEEE, 2013.

[21] Message Passing Interface Forum. MPI: A message-passing interface standard, version 3.0. `http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf`, Sept. 2012.

[22] 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 *Supercomputing, 2003 ACM/IEEE Conference*, pages 55–55. IEEE, 2003.

[23] S. J. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal Computation Physics*, 117:1–19, 1995.

[24] J. Vetter and C. Chambreau. mpip: Lightweight, scalable mpi profiling. *URL: http://www. llnl. gov/CASC/mpiP*, 2005.

[25] K. Yoshii, K. Iskra, H. Naik, P. Beckman, and P. C. Broekema. Characterizing the performance of "big memory" on blue gene linux. In *Parallel Processing Workshops, 2009. ICPPW'09. International Conference on*, pages 65–72. IEEE, 2009.