Mitigating System Noise With Simultaneous Multi-Threading*

Eli Rosenthal
Brown University
eli_rosenthal@brown.edu

Edgar A. León
Adam T. Moody
Lawrence Livermore National Laboratory
{leon,moody20}@llnl.gov

ABSTRACT

System noise on commodity clusters significantly impacts the scalability of scientific applications. As the number of nodes in a cluster grows, this impact outweighs the benefits of the increased system capability. In this work, we propose using Simultaneous Multi-Threading (SMT) to mitigate system noise, and we demonstrate its effectiveness using Fixed Work Quantum (FWQ) micro-benchmarks. This technique relies on using the floating-point and integer units of a core (shared SMT resources) for system processing when an application is stalled waiting for an out-of-core operation. Unlike core-specialization, which dedicates a core for system processing, our approach allows an application to use all cores of a node and proves significantly more effective in reducing system noise.

1. INTRODUCTION

The root cause of performance degradation for many scientific applications at scale stems from interference with system processes that run on each compute node. Because the application has little control over when and where these system processes run, the interference appears as *noise* to the application. Noise impacts a large number of scientific applications including global climate modeling, heart and organ simulations, visualization of mixing fluids, and detailed predictions of ecosystems.

Significant research and development over the last ten years [2] have virtually eliminated noise on leadership capability systems like Sequoia at Lawrence Livermore National Laboratory (LLNL). These gains, however, come with high costs associated with the research, development, and maintenance of specialized hardware and a radically simplified operating system (OS). Commodity systems, on the other hand, use inexpensive hardware and software designed for mass consumer markets. Since mass consumers do not commonly build large scale systems, designs for this hardware and software may disregard performance problems that arise at scale. Noise consumes intolerable amounts of execution time on today's commodity systems, and it will reach critical proportions in next-generation clusters if left unchecked.

2. IDENTIFYING NOISE

The effect of system noise on application performance depends on the schedule at which system processes execute across nodes [1]. The scalability of applications may improve if these processes are co-scheduled at the same time across the cluster. Thus, we first study whether system processes occur synchronously or asynchronously on a production Linux cluster at LLNL (*cab*). Compute nodes are comprised of two Intel Sandy Bridge sockets, with 8 cores per socket, and interconnected with an InfiniBand QDR fabric.

We use FWQ to quantify the impact of noise within each node and implement a global synchronous clock to correlate noise events across nodes. FWQ executes a fixed amount of work for a specified number of iterations and records the runtime (in processor cycles) of each iteration. This benchmark shows areas where the number of cycles to complete the same amount of wok is much higher, thereby indicating the impact of noise. Our global clock is based on Jones' synchronization algorithm [1]. We use *cab* for all of our experiments in this work.

Figure 1 shows a histogram of the duration of each FWQ iteration across time on 4 nodes with brighter colors indicating more populated bins. The median duration of a unit of work on a single core is about 1.02 ms. The spikes in this figure indicate a large amount of synchronous noise. At 256 nodes, noise is not synchronous as shown by Figure 2; areas of increased noise are broader and less distinct. Thus, without significant changes to the Linux kernel to co-schedule services (assuming they can be co-scheduled), system processes execute asynchronously across nodes, which increases the effect of noise on application performance.

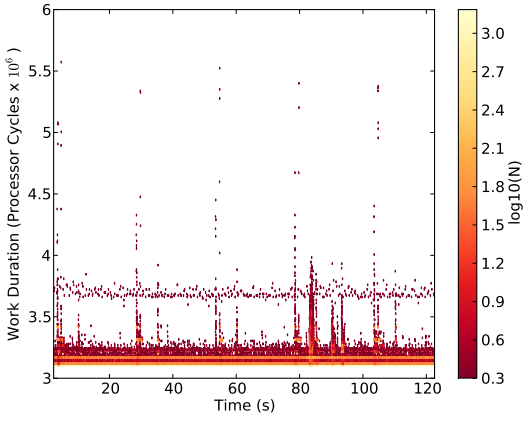

Figure 1: Work duration per iteration across time on 4 nodes. Well-defined spikes show areas of synchronous noise.

3. MITIGATING NOISE WITH SMT

Known methods of mitigating noise include dedicating one core per node for system tasks (core specialization). In this work, we propose using one thread of execution per core for processing system tasks on SMT architectures. SMT masks latencies of out-of-core operations. Thus, while an application's threads are stalled on a core, a different thread can process system tasks using the floating-point and integer

*This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-ABS-641767.

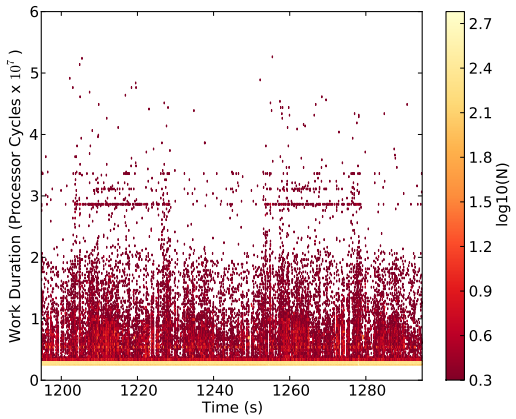

Figure 2: Work duration per iteration across time on 256 nodes. Noise at this scale is not synchronous.

units that are shared on that core. Unlike core specialization, an application can use all of the cores in the system. Intel uses the term Hyperthreading (HT) for their SMT chips. On *cab* each core features two hardware (hyper) threads.

Rajan et al. [3] observed significant improvements in application performance if one core is left free. We verified this behavior on *cab* and investigated the benefits of leaving core 0 free instead of another core. We ran FWQ on 256 nodes with 15 processes per node (PPN) and plotted the number of iterations (data points) as a function of their duration. As shown in Figure 3, leaving core 0 free for OS processing reduces noise more so than any other core (showing core 15 only); the tighter the spread of the points over the X axis and the closer to the left, the lower the noise. This is because certain sources of OS noise tend to execute on core 0 for portability reasons.

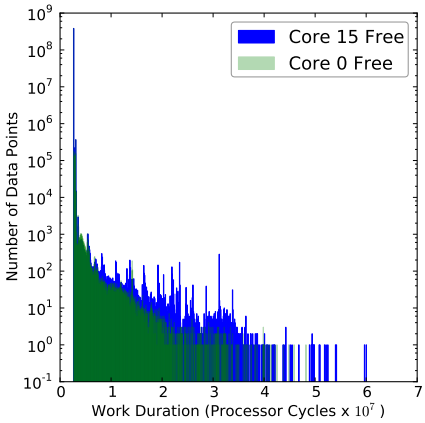

Figure 3: Work iterations based on their duration using 15 PPN leaving core 0 and core 15 free.

To measure the effect of HT on noise we ran FWQ on 256 nodes with 16 PPN (4096 total tasks) under two configurations: binding each MPI process to a core without HT and binding each process to one hardware thread per core with HT enabled (leaving the other thread idle for system processing). Figure 4 shows that the HT run resulted in a nearly 2000x decrease in the incidence of high-impact, low-frequency noise. Furthermore, when compared to core specialization (Figure 3) HT reduces this noise by over 1000x.

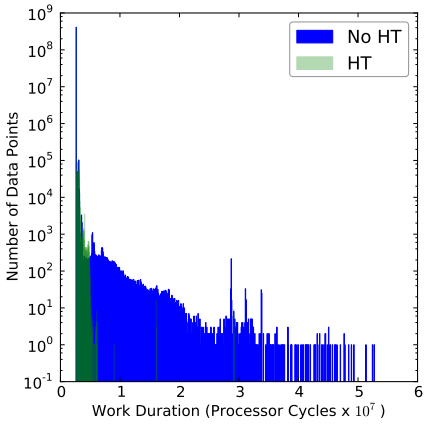

Figure 4: Work iterations based on their duration using 16 PPN with and without hyperthreading.

4. CONCLUSIONS AND FUTURE WORK

Despite significant advances in reducing system noise, scientific applications running on production, commodity clusters today continue to suffer the effect of noise on scalability. In this work, we propose using SMT to mitigate noise and demonstrate a dramatic reduction even when compared to core specialization. While these results are promising, they come at a cost, specifically an application can use at most $N-1$ hardware threads per core on an N-way SMT system. This may or may not affect application performance and, if so, the impact on performance and scalability is application dependent. Our future work includes evaluating the effectiveness of our SMT approach on the performance and scalability of a suite of applications from LLNL. We are interested in evaluating the tradeoffs of using *secondary* SMT threads for the application versus using them for mitigating system noise. We anticipate that scale will be an important factor in this study.

5. REFERENCES

[1] T. Jones. Linux kernel co-scheduling for bulk synchronous parallel applications. In *International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS'11, Tucson, AZ, May 2011. ACM.

[2] A. Morari, R. Gioiosa, R. W. Wisniewski, F. J. Cazorla, and M. Valero. In *International Parallel & Distributed Processing Symposium*, IPDPS'11, Anchorage, AK, May 2011. IEEE.

[3] M. Rajan, D. Doerfler, C. T. Vaughan, M. Epperson, and J. Ogden. Application performance on the Tri-Lab Linux Capacity Cluster–TLCC. *International Journal of Distributed Systems and Technologies*, 1(2), Apr. 2010.