2016 45th International Conference on Parallel Processing Workshops

The Effects of System Management Interrupts on Multithreaded, Hyper-threaded, and MPI Applications

Konstantin Macarenco, Kristina Frye, Benjamin Hamlin, and Karen L. Karavanic
Department of Computer Science
Portland State University
Portland, Oregon, USA
e-mail: karavan@pdx.edu

Abstract— **System Management Interrupts (SMIs) are a potential source of noise for parallel applications at all levels, and their use may be increasing. Building on a previous effort, we have conducted a study of SMI effects on parallel applications. In this paper we present initial results for NAS benchmarks, a multithreaded application, and the UnixBench benchmark. Findings reconfirm that performance degradation scales with the latency of the SMI. Additional results show that performance degradation increases when SMIs are enabled upon multiple communicating nodes, and may be amplified with HTT. This potential source of noise is of interest to tool developers both to detect its presence and also to accommodate the effect on measurement tools.**

Keywords-System Management Mode; runtime noise; parallel application performance; parallel performance tools

I. INTRODUCTION

System Management Mode (SMM) is a mode long available on x86 platforms that is more privileged than the operating system kernel or hypervisor. System Management Interrupts (SMIs) preempt the operations of applications, the operating system or hypervisor causing all processor cores to stop and enter SMM. Additionally, such preemption is invisible to the system software. This source of noise is different from OS noise, since it occurs at a lower level and in fact is not visible to the OS. This paper reports the results of a detailed study of the interactions of SMM with multithreaded, hyper-threaded, and parallel applications.

Originally, SMM was created to handle infrequent, simple activities such as power throttling, hardware emulation, and system health checks. More recently, with security compromises of hypervisors, such as Xen, SMM has been suggested as a mechanism to detect malicious code injection into hypervisors, as part of RIM (Runtime Integrity Measurement). The idea is for the RIM to periodically check the hypervisor and/or kernel code for changes by performing inspections from SMM. The use of SMM is advantageous as it uses a protected address space and is a much more challenging target for malware to compromise. Nevertheless, the amount of time needed to reside in SMM in order to perform security checks can be disruptive to the normal functioning of the application, hypervisor and operating system.

This work builds on a previous study to investigate the effects of SMM [7]. In this previous work it was concluded that while shorter length SMIs produce jitter, their effects upon performance are moderate. On the other hand, long SMIs are detrimental to performance. SMIs are shown to increase program run times, increase energy usage, cause time scaling discrepancies, reduce throughput, and produce audio and video distortions. In this paper we present results of a new study focusing specifically on parallel applications and their performance.

The remainder of the paper is organized as follows: In Section II we discuss background and related work. In Section III we present selected results and discussion for our MPI study. In Section IV we present results of a study of multithreaded applications. Finally, in Section V we summarize and conclude.

II. BACKGROUND AND RELATED WORK

A. System Management Mode (SMM)

In this section we summarize background information from previous work [7].

System Management Mode (SMM) is a special x86 processor mode that privileged software such as operating systems or hypervisors cannot access. This hardware feature was originally developed for operating system-independent functionality such as power throttling, hardware emulation, and running OEM code. The key distinction of SMM is its invisibility to the kernel and the hypervisor. The entry to SMM is through a System Management Interrupt (SMI), a unique type of interrupt that is much more disruptive than a traditional interrupt: When an SMI occurs, the standard behavior is all of the processor cores will enter System Management Mode [9][13]. The SMI handler will then perform the requested work, restore the interrupted context, and return. Because all CPU threads stay in SMM until the completion of the SMI's work, the severity of the impact increases with the number of cores. The x86 architecture features a variety of different types of exceptions and interrupts; SMIs are unique in that they are a higher priority interrupt than Nonmaskable Interrupts (NMIs) and device interrupts [14]. Because other interrupts cannot preempt SMM, other device interrupts will only be handled after it has finished its work. An Intel tool, BIOSBITS, warns if an interval of time spent in SMM exceeds 150 microseconds [15].

In the past it has been assumed that time spent in SMM would be relatively small and therefore its side effects were unimportant. Three recent trends have increased the importance of studying the performance impacts of SMM:

2332-5690/16 $31.00 © 2016 IEEE
DOI 10.1109/ICPPW.2016.55

Runtime Integrity Measurements or RIMs for security checking, virtualization, and the trend toward an increasing number of cores per socket. In the security realm, proposals to repurpose SMM for quick detection of malware and rootkits to limit their damage [10][16][17] as well as providing secure isolated execution environments [11] dramatically change expectations over its use. Since SMM completely pauses host software execution for the duration of its work and the time required for these new usages exceeds common SMI durations, there are clear performance concerns.

In previous work we have conducted a thorough investigation of the performance impacts of SMM. Our contributions included developing a measurement methodology for SMM-related studies. We identified a variety of slowdowns and problems caused by ramping up the rate or increasing the duration of time spent in SMM. Because the system software is unaware of time spent in SMM, the time is incorrectly attributed to whatever was running at the time of the SMI. Performance tools would similarly report the time incorrectly. The potential exists for an SMI to preempt time-sensitive code (e.g. code holding a global lock on one node in a cluster), resulting in delays well beyond what the software developer may have expected.

In this paper we focus specifically on the impacts for parallel applications.

B. Hyper-Threading Technology

Hyper-threading Technology (HTT) [1] is Intel's implementation of Simultaneous Multithreading (SMT). SMT is a technique that interleaves instructions from different threads, thereby introducing thread-level parallelism (TLP) in the CPU. Specifically, HTT exposes two logical cores to the operating system (OS) per physical core. The OS can then schedule different threads on these cores. A hyperthreaded CPU includes duplicate registers per physical core so that each logical core has its own set of registers. Each pair of logical cores also shares the same cache, which has a significant impact on performance [2][3]. In practice, the performance benefits of HTT are application-dependent. For HTT to benefit performance, there must be some gaps for HTT to fill, for example executing another thread when a thread is delayed by a cache miss. Leng et al. [4] found that applications performing intensive floating-point operations do not benefit from HTT because they already utilize CPU resources efficiently. I/O-bound applications, however, can benefit from HTT since one thread can execute while another is waiting on I/O. A NASA study concluded that structured applications, which "...access adjacent elements of the underlying data structures..." are usually cache-optimized by the compiler such that HTT has few opportunities to improve throughput by filling gaps [5].

Additionally, because each pair of logical cores share a cache, the cache behaviour of the two threads will affect performance. For example, two cache-friendly threads can compete with one-another and cause more cache misses than would otherwise occur with a single core and thread [6].

C. Application Runtime Noise

The effects of SMIs are quite unique in comparison to other interrupts occurring on the system. Traditional device interrupts can preempt running application code, however, they are able to provide acknowledgement to the hardware and set up mechanisms for future processing. SMIs cannot typically be deferred in this manner and all CPU threads leave the host environment and transition into the SMI handler upon receiving an SMI. Thus SMIs have a broader system impact than a traditional device interrupt. The operating system also isn't able to mask SMIs as with traditional device interrupts, which results in the potential for an SMI occurring at an unexpected point in time. Much of the focus on SMI performance comes from the field of Real Time Operating Systems (RTOS) and latency-sensitive computing. An Intel whitepaper noted that SMIs present serious complications to the XenoMai RTOS microkernel as they are invisible to the RTOS scheduler. [18] Latency-sensitive users attempt to disable SMIs to remove their impact [19][20] and others use tools to detect their occurrence [21].

Previous studies have examined the effect of *noise* from software heartbeats and system daemons [22], hardware interrupts [12], and network interrupts [23]. Ferreira et al. [24] have found that noise's effect on an application may be reduced by absorption; conversely, the impact of noise can be amplified when it occurs at a performance-sensitive time. Since SMIs are the highest priority interrupt, they affect the platform on a greater scale than these other types of noise. The operating system timer interrupt that Beckman et al. study [12] is itself at risk from asynchronous SMI noise.

III. SMM AND MPI APPLICATIONS

We designed and conducted the NAS Parallel Benchmark experiment presented here to check the effects of SMM intervals on parallel codes. Would the effects be amplified as we scaled up the problem size, number of MPI ranks or number of nodes? We measured the benchmarks on a small linux cluster, with varying amounts of time spent in SMM.

A. Hardware and Software

Experiments were run on 16 nodes of Wyeast, a small Linux cluster in our HPC Laboratory. All the nodes have the following configurations: Intel(R) Xeon(R) CPU E5520 @ 2.27GHz, cache size of 8192 KB, and 12Gb RAM. Hyperthreading was disabled or enabled as specified on all nodes. All machines ran CentOS 5.10, kernel version 3.0.4. We measured the EP, BT, and FT benchmarks

B. Triggering SMIs

To trigger SMIs in our experiments we used a modified version of the "Blackbox SMI" approach as described in [7]. This approach uses a driver that can be configured to produce two types of SMIs, named "short" and "long," with total time spent in SMM of 1-3 *ms* and 100-110 *ms*, respectively. No work is done while in SMM. The SMI driver uses the TSC counter to measure the average SMI latency. The driver triggers one SMI every *x* jiffies. In our system, one jiffy equals one millisecond.

C. NAS Parallel Benchmarks

To evaluate the effects of SMI noise on parallel codes, we conducted a detailed study using the MPI-based NAS parallel benchmarks. We included in our study three of the benchmarks: EP, BT, and FT; and three of the specified problem sizes: A, B, and C. For each, we recorded the resulting time, work completed, and MOPs (integer or floating

point operatings as relevant to the particular benchmark). We varied the number of nodes between 1 and 16, setting the number of MPI ranks per node to 1 or 4 for each. For each case we measured six runs and report the average. We repeated the entire set of measurements for the three cases: no SMI activity, short SMIs, and long SMIs, where short SMI has a duration of 1-3 *ms* and a frequency of 1/sec, and a long SMI has a duration of 100-110 *ms* and a frequency of 1/sec.

We show the results for BT in Table 1. BT is the Block Tri-diagonal solver. We see minor or no impact from short SMM intervals on any BT configuration. However, the long SMM intervals do result in noticeable delays. The impact of the long SMIs increases with the number of MPI ranks, for both the four ranks per node case and the single rank per node case.

We show the results for execution time for EP in **Table 2**. EP is the Embarassingly Parallel benchmark, with little synchronization between the MPI ranks. We would expect the effects of the SMI activity to be similar for each node, and not to grow as we scale up, due to the lack of synchronization. The SMM 0 column shows our base case, with no SMI activity added. The shaded columns of data show the % change from the no-SMM case for each of short (SMM=1) and long (SMM=2) SMM intervals. We see that the short SMM intervals do not significantly impact the runtimes, although there is no clear pattern and in two of the cases with 4 MPI ranks per node there are noticeable changes. However, with long SMM intervals we see increases in the execution times in every case, with a pattern of increasing perturbation as the number of nodes increases from 1 to 16.

For comparison, we show the results for the FT benchmark in **Table 3**. FT performs discrete 3D fast Fourier Transform, using MPI all-to-all communication. We note that in this case, 16 MPI ranks with 1 per node, or any number of ranks with 4 per node, are poor fits for the underlying platform, in the configuration used. The performance worsens as the number of MPI ranks increases. As in the EP case, we see that short SMIs do not significantly impact the benchmark runtimes. The long SMM intervals do have a significant impact, that scales up with the number of MPI ranks per node until 8, then decreases slightly. This is true even though the execution times are increasing as the number of ranks is increased.

During earlier testing (results not included here) of MPI applications and SMM, we were puzzled by anomalous results, and our investigations revealed that at the time of the experiments, unintentionally HTT was turned off on some nodes but enabled on some nodes. The indication that HTT might change our results motivated us to investigate this further. We conducted further testing to check the effects of Hyper-Threading Technology (HTT) on our results.

We show the results for EP and FT in **Table 4** and **Table 5**. We clearly see that our results are affected by HTT in the case of long SMM intervals. However, the impact does not follow a clear scaling pattern, and we do not see a similar impact for the short SMM intervals.

IV. SMM AND MULTITHREADED CODES

In this section we present a study designed to investigate the effects of SMM intervals on multithreaded codes. We investigate the effects of SMIs on a simple application kernel that performs convolution, and also on several of the UnixBench benchmarks. In addition to varying the duration of the SMM intervals, we also vary the number of available logical cores, to allow comparison of the single thread per core case (similar to no HTT) and the two threads per core case (similar to HTT).

A. Methodology

We tested the codes with two different length SMM intervals: 1-3 *ms* ("short") and 100-110 *ms* ("long"), respectively or "long"). We started with Delgado et al.'s SMI driver [7] and modified it slightly to allow us to vary SMI frequency. Testing various frequencies allowed us to chart the slope of SMI's impact and understand the interaction between various SMI configurations and the number of logical threads.

We performed all our tests on Dell Poweredge R410 servers. These use an Intel Xeon E5620 quad-core CPU with HTT in an Intel 5500 chipset with 4MB L1, 8MB L2, and 24MB L3 caches. The servers we ran our tests on had 12GB of main memory. The OS we tested on was Fedora with kernel version 3.17.4. All tests were run on a tickless kernel.

To vary the logical threads per core, we used the Linux *sysfs* interface to selectively offline specific logical cores, allowing us to vary the number of CPUs from 1–8. (Offlining a core's HTT sibling while leaving the physical core online causes the kernel to ignore the HTT sibling for scheduling purposes.) We tested 1-4 logical processor cores with all HTT siblings offlined, then selectively onlined the HTT siblings to test 5-8 logical processor cores. In this way we were able to evaluate configurations similar to HTT enabled and HTT disabled.

B. Convolve

We used a simple implementation of convolution ("Convolve") as our application kernel benchmark. Convolution is common in image processing and in computer vision algorithms such as SIFT. Given an NxN matrix P and an MxM matrix Q with M<N and M odd, convolving Q over P to produce a new matrix R (R=P*Q) involves, for each R[*i,j*], superimposing Q over P, centered at P[*i,j*], multiplying the superimposed elements, and summing the products. We parallelized this operation by splitting R up into blocks of a configurable size, *k*, and spawning a thread for each. Each thread calculated and wrote its submatrix R[*i..i+k-1,j..j+k-1*]. Note that there were no data dependencies, since each element of R depends only on P and Q, and those aren't modified.

We selected configurations for "cache-friendly" and "cache-unfriendly" experimentally using *cachegrind* [25]. Convolve simulates running a Gaussian filter over an image by dividing the image into subimages, which it then runs on as many threads as it has available. Each thread then performs convolution using the same kernel matrix. Each thread writes to thread-local memory, so there is no overhead from locking - time is spent exclusively in one of the following three activities: spawning threads; performing loads from shared memory and loads from and stores to each thread's writable memory; and performing integer multiplications and additions. Both the image and the kernel matrix are generated outside of the timed section, which uses the Linux `clock_gettime()` function with `CLOCK_MONOTONIC`. The size of the image, the size of each thread's subimage, the size of the kernel, and the

number of threads scheduled simultaneously are all configurable. This allowed us to tune the parameters of our tests for a variety of cache behaviors. For this experiment, we settled on two configurations: one with approximately 70% cache misses (CacheUnfriendly), and the other with approximately 1% cache misses (CacheFriendly), both out of approximately 20-million cache references. The parameters used are shown below. In both cases, we limited the number of threads to 24 and used SMI intervals of 50-1500 *ms* (in 50 *ms* increments).

	CF	CU
image size	0.5 megapixels	16 megapixels
subimage size	4x4 pixels	1 megapixel
kernel matrix size	61x61	3x3

The graphs in **Figure 1** detail the results of our Convolve experiments. The graphs on the left show the impact on execution time of increasing the frequency of SMIs. We present the mean of 3 runs. Each line represents a CPU configuration, and the x-axis shows the time between SMIs. For both cache-unfriendly and cache-friendly cases, the results show minimal or no impact of the time in SMM up to approximately 600 *ms* intervals. From this point up to the highest frequency (50*ms* intervals), we see a dramatic impact on execution time, with some configurations affected disproportionately. Our CacheUnfriendly configuration did not benefit greatly from HTT, as we see in the similarity of results for 4-8 logical cores. The latency introduced by frequent memory accesses should allow more opportunity for the CPU to interleave threads, however in this case *both* threads had high numbers of cache misses. In the CacheFriendly case, we see a similar performance degradation, although the variance at 50 *ms* intervals is not quite as high.

The graphs on the right detail the performance with different core configurations with a fixed SMI interval of 50 *ms*. For the CacheUnfriendly case (top) we see that although with 1-3 logical threads the performance scaled consistently, there is greater variance starting at 4 logical threads. Perhaps most straightforward is to compare 4 and 8 logical threads, since this case compares two homogeneous configurations. Due to variance we cannot conclude that performance is better or worse with 8 logical threads than the 4 logical thread configuration. The CacheFriendly configuration (bottom) shows minimal benefits from HTT. This is in keeping with the results of Saini, et al. [11]. The variance at the highest frequency is less marked, but still present.

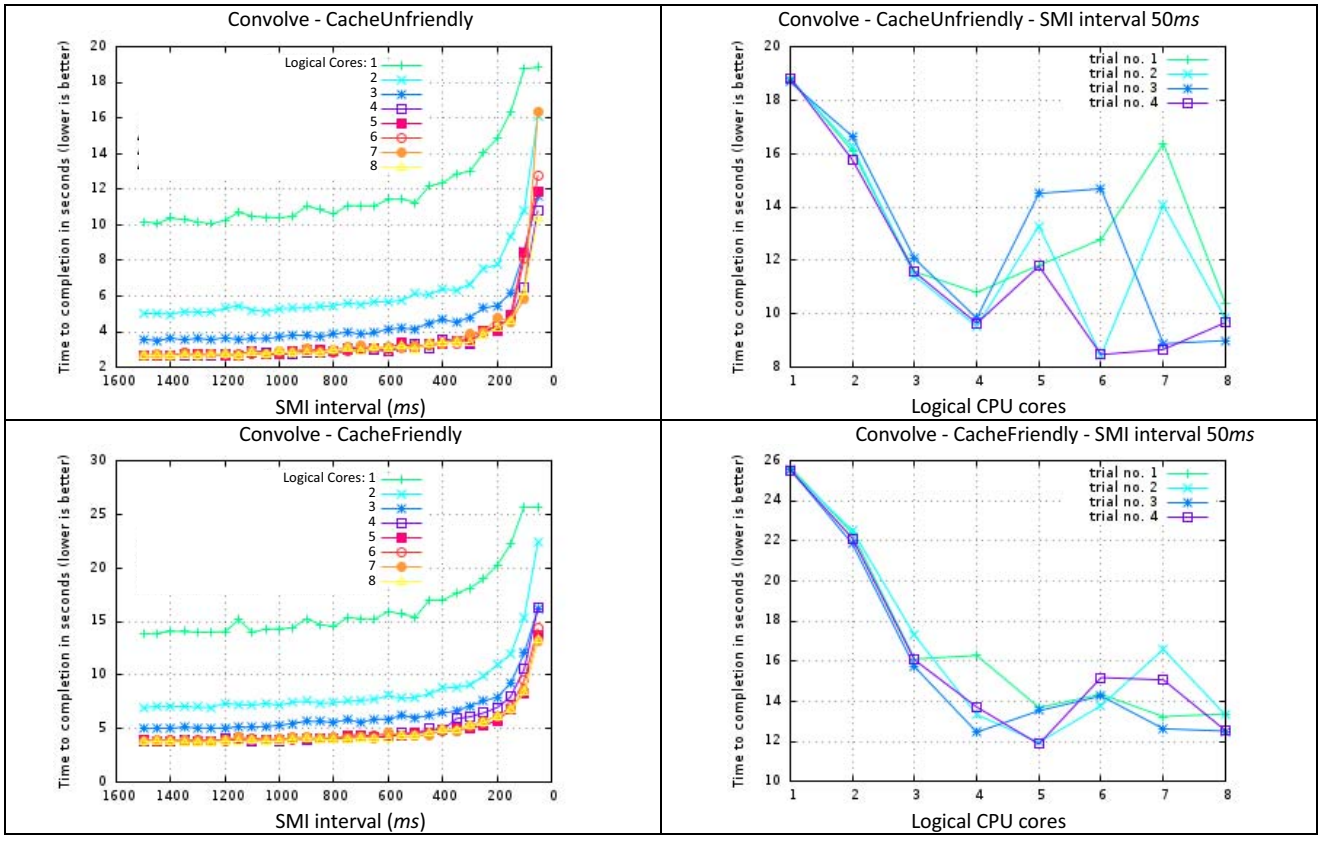

Figure 1: Convolve Experiments

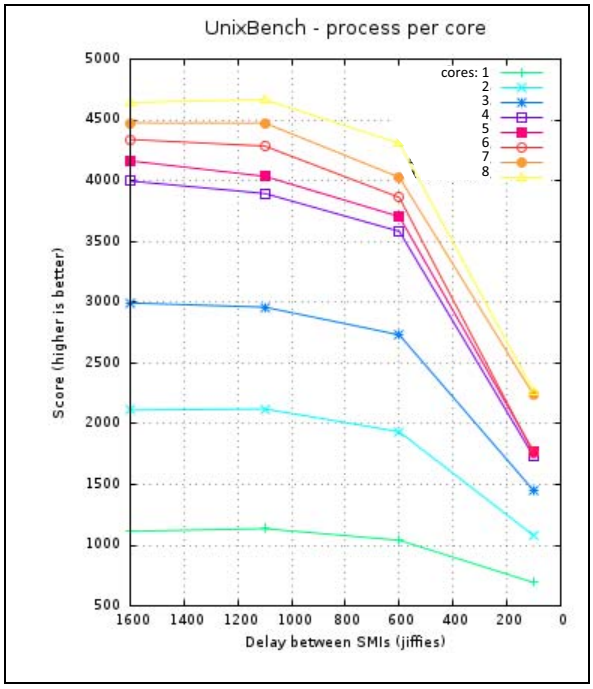

Figure 2 UnixBench Results

C. *UnixBench*

UnixBench [8] is a collection of benchmarks designed to measure a variety of different aspects of POSIX system performance. It generates an index score rating the system against a base system. We used the default configuration that runs each set of benchmarks twice: first with a single copy of the test program; then with one copy per core. We selected a subset of the benchmarks:

•Dhrystone - Performs various string manipulations.

•Whetstone - Measures floating point performance via various mathematical functions (sin, cos, sqrt, etc.)

•Pipe Throughput - Measures the performance of how well processes can read and write from a pipe.

•Pipe-based Context Switching - Measures the performance of two processes communicating through a pipe. The two processes pass an increasing integer to each other through a shared pipe.

•System Call Overhead - Measures how quickly processes can enter and exit system calls.

We ran UnixBench in a loop for each of the targeted CPU configurations. For each configuration, we measured SMI intervals from 100ms to 1600*ms* at 500 *ms* increments. We show the total index score for each iteration.

Our investigation of the effects of short SMIs did not show any change in the performance score as we increased the rate of SMIs. Hence, we conclude that the short SMIs do not have a noticeable effect on the UnixBench performance index.

We show the results for long SMIs in **Figure 2**. This graph shows the performance of UnixBench at different SMI frequencies and CPU configurations. The y-axis is the benchmark's score, so higher is better. The x-axis is the gap between SMIs, in jiffies or *ms (1 jiffy = 1ms on this system)*, therefore the larger the x-axis value, the lower the frequency of the SMIs. The benchmark shows performance gains from HTT, and the different CPU configurations are affected symmetrically, up to 600*ms* intervals. Long SMMs with SMI intervals shorter than 600 *ms* showed the greatest effect on UnixBench performance. As the number of cores increases, the effect of SMIs becomes greater.

D. *Discussion*

In this section we presented our study of the effects of SMIs on multithreaded codes. The results suggest minimal effects of short SMMs, and of long SMMs at intervals greater than 600 *ms*. However, our experiments demonstrated clear impacts of long SMIs at intervals less than 600 *ms*, with the impact mostly increasing with the number of logical threads.

V. Conclusions and Future Work

The results presented here represent our effort to characterize the possible effects on our applications and performance measurements of time spent in System Management Mode. This is an important potential source of noise, with effects that grow with increased number of cores and in some cases, with the use of HTT.

SMM time below a certain latency did not cause significant performance effects on MPI or multithreaded applications; however, longer time intervals spent in System Management Mode caused noticeable slowdowns for both MPI applications and multithreaded applications, that in some cases was amplified by the use of HTT. The impacts would not be reported correctly by the current generation of performance tools, since the system level software (kernel or hypervisor) are not aware of the time spent in SMM and attribute it in various incorrect ways.

As we continue our study of SMI noise, we hope to focus in more precisely on the cause of variance with HTT, and to test additional parallel applications at larger scales.

Acknowledgments

This research was sponsored in part by NSF award # 1528185. We acknowledge the contributions of Portland State CS533 students in tests and discussions that first showed a performance effect between SMM and HTT. We thank the PSU Maseeh College Computer Action Team (CAT) for their ongoing support of the lab facilities. Brian Delgado provided the SMI driver and instructions for its use.

References

[1] "Intel Hyper-Threading Technology Technical User's Guide." Intel, 2003.

[2] "NO_HZ: Reducing Scheduler Clock Ticks." https://www.kernel.org/doc/Documentation/timers/NO_HZ.txt Accessed June, 2015.

[3] W. Hassanein, L. Rashid, M. Mehri, and M. Hammad. "Characterizing the Performance of Data Management Systems on Hyper-Threaded Architectures." In *Proceedings of the 18th Inter- national Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD). 2006.

[4] T. Leng, R. Ali, J. Hsieh, V. Mashayekhi, and R. Rooholamini. "An Empirical Study of Hyper-Threading in High Performance Computing Clusters" Dell Computer Corp., USA, 2014.

[5] S. Saini, H. Jin, R. Hood, D. Barker, P. Mehrotra, and R. Biswas. "The Impact of Hyper-Threading on Proces- sor Resource Utilization in Production Applications" In *Procedings of the 18th International Conference on High Performance Computing* (HiPC). Bengaluru, India, December, 2011.

[6] J. Cieslewicz. "Improving database performance on simultaneous multi- threading processors" In *Proceedings of the International Conference on Very Large Data Bases* (VLDB), pages 49–60. 2005.

[7] B. Delgado and K. Karavanic. "Performance Implications of System Management Mode." In *2013 IEEE International Symposium on Workload Characterization* (IISWC), pages 163–173. Portland, OR, USA, 2013.

[8] UnixBench https://github.com/kdlucas/byte- unixbench. Accessed June, 2015.

[9] P. Stultz, "System and method for processing system management interrupts in a multiple processor system," U.S. Patent 7,200,701 B2, issued April 3, 2007.

[10] A. Azab, P. Ning, et al., "HyperSentry: enabling stealthy in-context measurement of hypervisor integrity," CCS. Chicago, IL, 2010.

[11] A. Azab, P. Ning, and X. Zhang, "SICE: A Hardware-Level Strongly Isolated Computing Environment for x86 Multi-core Platforms," CCS. Chicago, IL 2011.

[12] 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): pp. 3- 16, 2008.

[13] Camden Associates, "Device Driver and BIOS Development for AMD Multiprocessor and Multi-Core Systems,"Advanced Micro Devices, Inc. 2006.

[14] Intel, Intel® 64 and IA-32 Architectures Software Developer's Manual (Vol. 3).

[15] J. Triplett and B. Triplett, "BITS: BIOS Implementation Test Suite,"http://www.linuxplumbersconf.org/2011/ocw/system/presen tations/867/original/bits.pdf

[16] J. Wang, A. Stavrou, and A. Ghosh, "HyperCheck: a hardware-assisted integrity monitor," Lect. Notes Comput. Sci. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 6307 LNCS: 158-177, 2010.

[17] F. Zhang, K. Leach, K. Sun, and A. Stavrou, "SPECTRE: A dependable Introspection Framework via System Management Mode," DSN, Budapest, Hungary, 2013.

[18] A. Ugal, "Hard Real Time using Xenomai on Intel Multi-Core Procesors", white paper, Intel, 2009.

[19] R. Ghosh-Roy, "Disabling SMIs on Intel® ICH5 Chipsets," http://www.mathworks.com/matlabcentral/fileexchange/18832-disabling-smis-on-intelr-ich5-chipsets, accessed July 31, 2013.

[20] "Configuring and Tuning HP ProLiant Servers for Low-Latency Applications", white paper, HP, 2013.

[21] C. Williams, "Realtime in the Enterprise," RTLWS11, Dresden, Germany,https://www.osadl.org/fileadmin/dam/presentations/RTL WS11/clarkw-realtime-in-the-enterprise.pdf, accessed: July 30, 2013, 2009.

[22] F. Petrini, D.J. Kerbyson, and S. Pakin, "The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q," SC 2003. Phoenix, Arizona, 2003.

[23] D. Tsafrir, Y. Etsion, D. Feitelson, and S. Kirkpatrick, "System noise, OS clock ticks, and fine-grained parallel applications." Proceedings of the 19th annual international conference on Supercomputing. Cambridge, Massachusetts, ACM: 303-312 , 2005.

[24] K. Ferreira, P. Bridges, and R. Brightwell, "Characterizing application sensitivity to OS interference using kernel-level noise injection," Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press: 1-12. Austin, Texas, 2008.

[25] N. Nethercote and J. Seward, "Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation,"Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007), San Diego, California, USA, June 2007.

Table 1: BT Benchmark with no (0), short (1) and long (2) SMM intervals

		1 MPI rank per node							4 MPI ranks per node						
	MPI rks	**SMM 0**	**SMM 1**	**Δ**	**%**	**SMM 2**	**Δ**	**%**	**SMM 0**	**SMM 1**	**Δ**	**%**	**SMM 2**	**Δ**	**%**
A	1	86.87	86.89	0.02	0.02	96.24	9.37	10.79	24.89	24.88	-0.01	-0.04	27.55	2.66	10.69
	4	27.44	27.57	0.13	0.47	39.53	12.09	44.06	53.78	50.93	-2.85	-5.30	64.13	10.35	19.25
	16	48.51	48.93	0.42	0.87	95.23	46.72	96.31	103.27	102.39	-0.88	-0.85	173.93	70.66	68.42
B	1	369.7	369.55	-0.15	-0.04	409.36	39.66	10.73	103.44	103.4	-0.04	-0.04	114.52	11.08	10.71
	4	108.1	108.58	0.48	0.44	148.39	40.29	37.27	85.53	85.31	-0.22	-0.26	108.94	23.41	27.37
	16	123.79	124.44	0.65	0.53	179.56	55.77	45.05	173.78	174.77	0.99	0.57	262.97	89.19	51.32
C	1	1585.75	1585.95	0.2	0.01	1756.33	170.58	10.76	424.39	424.51	0.12	0.03	470.35	45.96	10.83
	4	419.75	420.67	0.92	0.22	537.73	117.98	28.11	219.86	218.9	-0.96	-0.44	281.38	61.52	27.98
	16	336.84	336.58	-0.26	-0.08	439.49	102.65	30.47	402.26	403.79	1.53	0.38	535.67	133.41	33.17

Table 2: EP Benchmark with no (0), short (1) and long (2) SMM intervals

		1 MPI rank per node							4 MPI ranks per node						
	MPI ranks	SMM 0	SMM 1	Δ	%	SMM 2	Δ	%	SMM 0	SMM 1	Δ	%	SMM 2	Δ	%
A	1	23.12	23.18	0.06	0.26	25.66	2.54	10.99	5.87	5.87	0	0.00	6.47	0.6	10.22
	2	11.69	11.6	-0.09	-0.77	13.15	1.46	12.49	2.93	2.93	0	0.00	3.35	0.42	14.33
	4	5.84	5.8	-0.04	-0.68	6.77	0.93	15.92	1.47	1.47	0	0.00	1.75	0.28	19.05
	8	2.92	2.94	0.02	0.68	3.5	0.58	19.86	0.73	0.74	0.01	1.37	0.95	0.22	30.14
	16	1.46	1.47	0.01	0.68	2.04	0.58	39.73	0.37	0.42	0.05	13.51	0.65	0.28	75.68
B	1	92.72	93.17	0.45	0.49	102.5	9.78	10.55	23.49	23.42	-0.07	-0.30	25.97	2.48	10.56
	2	46.35	46.59	0.24	0.52	52.58	6.23	13.44	11.71	11.66	-0.05	-0.43	13.27	1.56	13.32
	4	23.33	23.28	-0.05	-0.21	26.71	3.38	14.49	5.9	5.93	0.03	0.51	6.77	0.87	14.75
	8	11.67	11.74	0.07	0.60	13.51	1.84	15.77	2.96	2.95	-0.01	-0.34	3.58	0.62	20.95
	16	5.86	5.9	0.04	0.68	7.03	1.17	19.97	1.59	1.49	-0.1	-6.29	2.06	0.47	29.56
C	1	370.67	372.53	1.86	0.50	411.19	40.52	10.93	93.86	93.33	-0.53	-0.56	104	10.14	10.80
	2	185.1	185.87	0.77	0.42	210.03	24.93	13.47	46.96	46.85	-0.11	-0.23	53.01	6.05	12.88
	4	93.36	93.34	-0.02	-0.02	106.47	13.11	14.04	23.47	23.48	0.01	0.04	28.32	4.85	20.66
	8	46.9	47.09	0.19	0.41	53.59	6.69	14.26	11.78	12.61	0.83	7.05	13.66	1.88	15.96
	16	24.94	25.16	0.22	0.88	28.49	3.55	14.23	5.91	5.9	-0.01	-0.17	7.53	1.62	27.41

Table 3: FT Benchmark with no (0), short (1) and long (2) SMM intervals

		1 MPI rank per node							4 MPI ranks per node						
	MPI ranks	SMM 0	SMM 1	Δ	%	SMM 2	Δ	%	SMM 0	SMM 1	Δ	%	SMM 2	Δ	%
A	1	7.64	7.61	-0.03	-0.39	8.41	0.77	10.08	2.49	2.49	0	0.00	2.78	0.29	11.65
	2	6.22	6.21	-0.01	-0.16	7.96	1.74	27.97	3.34	3.34	0	0.00	4.21	0.87	26.05
	4	4.25	4.24	-0.01	-0.24	6.05	1.8	42.35	5.69	5.49	-0.2	-3.51	6.96	1.27	22.32
	8	2.22	2.22	0	0.00	4.32	2.1	94.59	9.51	9.22	-0.29	-3.05	13.6	4.09	43.01
	16	6.5	6.39	-0.11	-1.69	10.43	3.93	60.46	20.57	20.51	-0.06	-0.29	28.42	7.85	38.16
B	1	95.48	95.65	0.17	0.18	106.09	10.61	11.11	31.2	31.2	0	0.00	34.53	3.33	10.67
	2	76.35	76.31	-0.04	-0.05	91.46	15.11	19.79	40.46	40.38	-0.08	-0.20	49.97	9.51	23.50
	4	51.85	51.73	-0.12	-0.23	67.24	15.39	29.68	39.46	39.65	0.19	0.48	52.37	12.9	32.72
	8	26.74	26.74	0	0.00	41.52	14.78	55.27	56.19	58.01	1.82	3.24	74.52	18.3	32.62
	16	82.18	82.96	0.78	0.95	110.93	28.75	34.98	127.33	127.28	-0.05	-0.04	157.82	30.4	23.95
C	1	-	-	-	-	-	-	-	135.96	136.09	0.13	0.10	150.59	14.6	10.76
	2	-	-	-	-	-	-	-	163.06	165.12	2.06	1.26	200.84	37.7	23.17
	4	216.75	216.58	-0.17	-0.08	264.44	47.69	22.00	125.66	126.34	0.68	0.54	163.17	37.5	29.85
	8	111.31	111.44	0.13	0.12	145.04	33.73	30.30	107.47	107.88	0.41	0.38	141.09	33.6	31.28
	16	315.42	313.81	-1.61	-0.51	419.34	103.9	32.95	339	337.92	-1.08	-0.32	412.11	73.1	21.57

Table 4: Effect of HTT on EP with 4 MPI ranks per node

		SMM 0			SMM 1			SMM 2			
	MPI rks	ht=0	ht=1	Δ	ht=0	ht=1	Δ	ht=0	ht=1	Δ	%
A	1	5.87	5.81	-0.06	5.87	5.81	-0.06	6.47	6.78	0.31	4.79
	2	2.93	2.91	-0.02	2.93	2.93	0	3.35	3.45	0.1	2.99
	4	1.47	1.46	-0.01	1.47	1.46	-0.01	1.75	1.77	0.02	1.14
	8	0.73	0.74	0.01	0.74	0.74	0	0.95	0.99	0.04	4.21
	16	0.37	0.39	0.02	0.42	0.39	-0.03	0.65	0.88	0.23	35.38
B	1	23.49	23.3	-0.19	23.42	23.24	-0.18	25.97	26.94	0.97	3.74
	2	11.71	11.69	-0.02	11.66	11.7	0.04	13.27	13.56	0.29	2.19
	4	5.9	5.86	-0.04	5.93	6.67	0.74	6.77	6.85	0.08	1.18
	8	2.96	2.95	-0.01	2.95	2.94	-0.01	3.58	3.56	-0.02	-0.56
	16	1.59	1.48	-0.11	1.49	1.5	0.01	2.06	2.14	0.08	3.88
C	1	93.86	93.24	-0.62	93.33	93.33	0	104	108.2	4.2	4.04
	2	46.96	46.43	-0.53	46.85	47.18	0.33	53.01	53.94	0.93	1.75
	4	23.47	23.44	-0.03	23.48	23.49	0.01	28.32	27.39	-0.93	-3.28
	8	11.78	11.71	-0.07	12.61	11.76	-0.85	13.66	13.77	0.11	0.81
	16	5.91	5.91	0	5.9	5.93	0.03	7.53	7.58	0.05	0.66

Table 5: Effect of HTT on FT with 4 MPI Ranks Per Node

		SMM 0			SMM 1			SMM 2			
	MPI rks	ht=0	ht=1	Δ	ht=0	ht=1	Δ	ht=0	ht=1	Δ	%
A	1	2.49	2.49	0	2.49	2.49	0	2.78	2.89	0.11	3.96
	2	3.34	3.33	-0.01	3.34	3.33	-0.01	4.21	4.19	-0.02	-0.48
	4	5.69	5.63	-0.06	5.49	5.28	-0.21	6.96	6.97	0.01	0.14
	8	9.51	9.78	0.27	9.22	9.89	0.67	13.6	12.33	-1.27	-9.34
	16	20.57	20.21	-0.36	20.51	20.1	-0.41	28.42	25.69	-2.73	-9.61
B	1	31.2	31.08	-0.12	31.2	31.13	-0.07	34.53	35.94	1.41	4.08
	2	40.46	40.41	-0.05	40.38	40.3	-0.08	49.97	50.18	0.21	0.42
	4	39.46	39.78	0.32	39.65	39.41	-0.24	52.37	48.86	-3.51	-6.70
	8	56.19	57.09	0.9	58.01	56.23	-1.78	74.52	69.18	-5.34	-7.17
	16	127.33	127.74	0.41	127.28	129.95	2.67	157.82	154.64	-3.18	-2.01
C	1	135.96	135.59	-0.37	136.09	135.5	-0.59	150.59	157.04	6.45	4.28
	2	163.06	165.57	2.51	165.12	164.33	-0.79	200.84	206.55	5.71	2.84
	4	125.66	125.8	0.14	126.34	125.57	-0.77	163.17	160.26	-2.91	-1.78
	8	107.47	108.15	0.68	107.88	106.92	-0.96	141.09	134.8	-6.29	-4.46
	16	339	331.25	-7.75	337.92	330.41	-7.51	412.11	392.96	-19.15	-4.65