Performance Impacts and Limitations of Hardware Memory Access Trace Collection

Nicholas C. Doyle*, Eric Matthews*, Graham Holland*, Alexandra Fedorova† and Lesley Shannon*

*School of Engineering Science
Simon Fraser University
Burnaby, Canada
{ndoyle, ematthew, gmh7, lshannon}@sfu.ca

†Dept. of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
sasha@ece.ubc.ca

***Abstract*—In today's multicore architectures, complex interactions between applications in the memory system can have a significant and highly variable impact on application execution time. System designers typically use hardware counters to profile execution behaviours and diagnose performance problems. However, hardware counters are not always sufficient and some problems are best identified with full memory access traces. Collecting these traces in software is very expensive; our work explores using dedicated hardware for memory-access trace collection. We analyze the limitations of this approach and its impacts on application performance. Our study is performed on actual hardware using two very different CPU platforms: 1) the PolyBlaze multicore soft processor and 2) the ARM Cortex-A9. In both cases, the data collection is implemented on an FPGA. Using micro-benchmarks designed to test the bounds of memory access behaviour, we illustrate the operational regions of data collection and the impact on system performance. By examining the bandwidth bottlenecks that limit the rate of data collection, as well as hardware architecture choices that can aggravate the impact on application performance, we provide guidelines that can be used to extrapolate our analysis to other systems and processor architectures.**

I. Introduction

Multicore architectures are now popular for embedded systems as they improve throughput and power consumption for multi-application workloads. They share hardware resources, such as caches and memory controllers, between the CPU cores, which can introduce non-deterministic interactions between threads and processes in the memory system, and can have a significant and highly variable impact on performance. Applications with unpredictable memory access patterns and systems with complicated memory hierarchies, such as *Non-Uniform Memory Access* (NUMA) architectures, make these interactions very difficult to understand and diagnose. System designers often use hardware performance counters or software simulation to profile performance problems. Both methods have limitations. Profiling tools report aggregate statistics or samples of memory accesses, whereas software simulation does not provide runtime analysis.

This work explores an alternative: collection of memory access traces in hardware. Hardware tracing has the potential to collect traces at lower overhead. Yet due to the massive volumes of data this process can generate, the impact of trace collection can be non-negligible. As existing trace-collection hardware features on contemporary CPUs do not collect full memory-access traces, the impact of hardware memory-access trace collection is difficult to evaluate.

We use a *Field Programmable Gate Array* (FPGA)-based platform to implement an SMP architecture with an OS in hardware for our study. This allows us to capture the complex interactions in the memory system between the data collection and the applications. We evaluate the implications of hardware memory-access trace recording on system performance as well as its limitations and practical design considerations.

The contributions presented in this work include:

- A detailed study characterizing runtime hardware memory-access trace collection, including the impact on application performance and limits on data collection across a comprehensive range of memory behaviours and data collection configurations.
- A comparison of performance between two distinct CPU architectures to show how our results can be extrapolated across platforms and which design features affect data collection behaviour. Further, we provide guidelines for use in future systems.
- A FPGA-based demonstration of runtime memory-access trace collection on an SMP architecture with an OS using standard PARSEC [1] workloads.

Our system features *Performance Monitoring Co-processors* (PMCs) that collect trace data directly from CPU debug signals and store it using *Direct Memory Access* (DMA) to system memory. We use FPGAs to create two multicore CPU systems: (1) a simple, in-order PolyBlaze soft processor [2] and (2) a superscalar, ARM Cortex-A9 hard processor that has been used in many embedded applications. The two platforms allow us to develop a generic model for memory-access trace collection for distinct hardware architectures.

This paper is organized as follows. Section II discusses related works and Section III summarizes our methodology and design. Section IV describes our experiments and results. Finally, Section V outlines our conclusions and future work.

II. Related Work

Designers typically rely on either Hardware Performance Counters (HPCs) or simulation to profile their software performance. However, accurate modelling of non-deterministic

978-3-9815370-8-6/17/$31.00

memory system behaviour and process interactions in a multi-core system at fast execution speeds is challenging [3]. Thus, we focus on hardware-based approaches.

A. Hardware Performance Monitoring Units

CPU vendors include *Performance Monitoring Units* (PMUs) and *Hardware Performance Counters* (HPCs) as part of their architectures [4][5][6] to provide application execution information. PMUs use software-configurable counters for data collection on important internal system events, such as cache misses, branches, and data memory reads/writes. HPCs are also used by tools that help application developers produce functional and high performance code. Greathouse et al. [7] present a HPC-assisted race condition detector that is faster than instrumentation-based race detectors without sacrificing accuracy. Similarly, PBI [8] uses HPCs to detect sequential and concurrency bugs in production software. HPCs are also used to check for malware in static and dynamic execution of programs [9] and in kernel rootkit detection [10], but the accuracy of HPC data collection is a limiting factor.

There have been many examples of stand-alone hardware PMUs in academic literature [11][12][13][14], but they are all structured like HPCs and are not designed to collect high frequency data. The accuracy of collected data is important for software verification, yet studies have uncovered problems with event counts for even deterministic events for HPC collected data in commercial processors [15][16][17].

Intel VTune [18], AMD CodeXL [19], as well as *perf* on Linux [20] use data collected from HPCs as an aid to debugging and optimization. To enhance the collection of performance data, users can collect detailed data on individual instructions using the Instruction-Based Sampling (IBS) and Precise Event-Based Sampling (PEBS) techniques on AMD and Intel processors respectively [4], [5]. These are sample-based approaches that require software intervention through interrupts and tend to introduce significant overhead.

Recognizing the need for in-system data collection and addressing the limitations of sample-based approaches, major CPU vendors also support low-overhead collection of execution trace data. Techniques include *Processor Trace* (PT) [21] on Intel architectures, *Lightweight Profiling (LWP)* [22] on AMD architectures, and *Embedded Trace Macrocell* (ETM) [23] and CoreSight [24] on ARM architectures. The Intel and AMD approaches are integrated directly into the CPU architecture, with added instructions to access the trace functionality, using the CPU memory interface to store collected data. The ARM approach (CoreSight) has a separate hardware unit that receives test data from debugging signals from the CPU. CoreSight uses a dedicated memory pathway to collect trace data and store it via several interfaces, including a direct connection to system memory. At present, there is no commercial support to collect memory-access traces.

Chen et al. studied high speed collection and analysis of trace data in Log-Based Architectures (LBA) [25]. They extend SHIM's approach [26] by incorporating hardware enhancements into the CPU architecture to improve performance of runtime analysis on unused CPU resources. The primary focus of LBA has been on the data processing methods, while the performance of the hardware enhancements have been only evaluated with simulation [27]. These techniques can be applied to data collected with the hardware-based approaches investigated in this paper and further motivates adding this functionality to future CPU designs. Similar to Coresight, ABACUS [28][29], is a modular, CPU-independent, performance monitoring framework demonstrated with different soft processor architectures on FPGAs. We build on this previous work and ARM Coresight to investigate the impact of hardware memory-access traces.

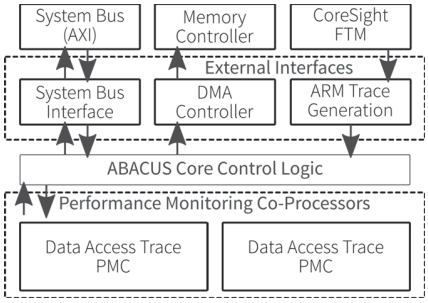

Fig. 1. Performance Monitoring Framework system diagram

III. Methodology

Existing CPU architectures do not offer runtime hardware collection of memory access traces to system memory. We developed *Performance Monitoring Co-Processors* (PMCs) implemented on FPGAs to perform hardware memory-access trace collection and study its performance impact. This was created as part of the *Performance Monitoring Framework* (PMF) shown in Figure 1 (based on ABACUS [2][29]), which we integrated into our CPU systems.

We use two CPU architectures for this analysis: a PolyBlaze [2][30] soft-processor implemented on a Xilinx Virtex 6 FPGA (Figure 2) and a Xilinx Zynq-7000 SoC with an ARM Cortex-A9 hard-block processor [31] (Figure 3). PolyBlaze, an SMP variant of the Xilinx MicroBlaze [32], is a relatively simple in-order processor developed for efficient implementation on FPGAs. For this work, it was configured with 64 KB, 4-way L1 instruction and data caches. The ARM Cortex-A9 is an advanced out-of-order, superscalar processor. The Cortex-A9 system has a 512 KB Level 2 (L2) cache, while the PolyBlaze has only the 64 KB Level 1 (L1) caches.

While the architectures have similar memory hierarchies, details such as cache and memory coherency policies, connection widths, and frequency play a critical role in performance. In the PolyBlaze system, a write-through memory policy results in a 2-word read-modify-write to occur in the memory controller for every write instruction, and creates the most memory congestion on this platform. In the ARM Cortex-A9 system, an L2 cache line eviction on a write results in an 8-word write, followed by an 8-word read, and creates the most memory congestion on this platform.

The Data Access Trace PMCs (Figure 1) are hardware units that collect memory-access trace information directly

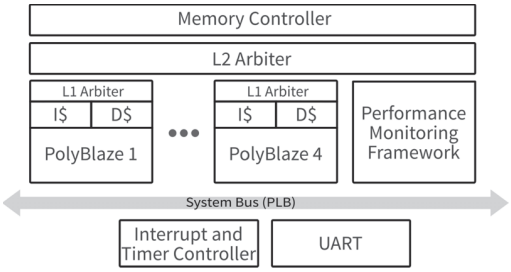

Fig. 2. PolyBlaze System Configuration

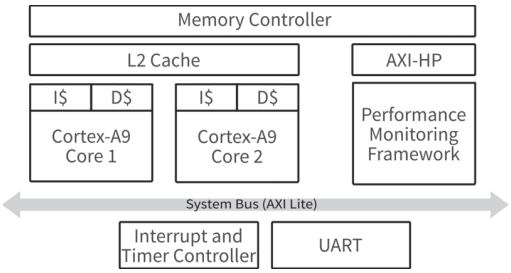

Fig. 3. ARM Cortex-A9 System Configuration

and store to system memory without software intervention. We characterize their impact on system behaviour using synthetic micro-benchmarks on a single CPU and use a case study to demonstrate their impact on real-world workloads running on an OS on SMP system. The PMCs can collect virtual and physical addresses from the memory stage of the pipeline, as well as generate timestamps. An internal 4 KB buffer helps absorb bursts of memory congestion, ensuring *data loss only occurs when the sustained data generation rate exceeds available memory bandwidth*.

As the PolyBlaze CPU is a soft-processor, the PMF can interface directly into debug signals from the CPU. However, data collection is not as simple for the ARM Cortex-A9 on the Xilinx Zynq-7000 SoC, as the FPGA fabric does not have direct access to the debug signals from the CPU. The regular nature of the synthetic micro-benchmarks makes generating representative debug signals straightforward. To extend our study to the this platform, we have created a *Trace Generation Module* (TGM) that can be programmed with the pattern of memory instructions for the particular benchmark being executed. A CPU signal generated from the branch instruction at the bottom of the benchmark loop is transmitted via CoreSight and used to synchronize the TGM with actual benchmark execution.

Our micro-benchmarks are written in assembly as a mixture of memory and non-memory instructions that can be varied from 0-100% for both instructions that access system memory and those that will be handled by the cache. As both CPU architecture have a single memory interface, they can only process a single memory instruction per cycle. To reflect this, the benchmarks target IPC of 1.0 when operating in cache.

IV. Experiments and Results

A. Synthetic Micro-Benchmark Characterization

Our micro-benchmarks are run on both hardware platforms using a variety of PMC configurations. We vary the number of PMCs simultaneously collecting data (0 to 16) and the sample rate of each (1 to 1/32). Due to issues with keeping an IPC of 1.0 for the 100% write operation benchmark on the ARM Cortex-A9 system, the tests on this hardware platform only go to 80%. The CPU and DDR memory frequencies are fixed at 100 MHz and 400MHz, respectively, for both the PolyBlaze and ARM systems.

We characterized both the PolyBlaze and ARM Cortex-A9 systems by investigating the impacts on benchmarks that operate entirely in the cache (i.e. using reads) and that access the memory system (i.e. using writes) as the percentage of memory accesses increases for the benchmarks. Figure 4 shows the impact of increasing data generation when the benchmarks operate entirely in the cache using "reads" on the loss of trace data for both the PolyBlaze and ARM Cortex-A9 systems. Figure 5 show the impact on data generation and decrease in IPC for the PolyBlaze when benchmarks access the memory system using "writes". We have only included the results for a sample rate of 1 (all data collected), with 2, 4, 8, and 16 PMCs active. However, our results show that doubling the number of active profiling units has the same impact on data generation as doubling the sample rate.

Memory system congestion caused by hardware trace data collection has no performance impact on benchmarks operating entirely in the cache. With the IPC at 1.0, we found that the generated data scales linearly with the increase in proportion of memory instructions (reads) and is the same for both systems. Trace data is lost when data collection saturates the available bandwidth to memory. Figures 4 (a) and (b) illustrate that the Cortex-A9 system has almost twice the data collection bandwidth (∼581 MB/s) as the PolyBlaze system (∼290 MB/s). This is a direct result of the Cortex-A9 system's 64-bit memory interface versus the PolyBlaze's 32-bit interface.

Conversely, the performance of benchmarks that access system memory are very sensitive to the impact of congestion in the memory system. Although data generation was similar for both PolyBlaze and Cortex-A9, the PolyBlaze system's lower memory bandwidth and write-through cache architecture had a significant impact on the system's IPC (Figure 5), whereas the impact on the IPC of the benchmarks run on the Cortex-A9 was negligible and thus not shown here.

The differences in cache structure and arbitration also play a critical role in the differences in behaviour between the two systems. Arbitration between the CPU and data collection in the PolyBlaze system occurs at the L2 arbiter, which operates at 200 MHz and has a 32-bit interface with memory. In the Cortex-A9 system, arbitration occurs directly in the memory controller, which operates at 400 MHz and has a 64-bit interface with memory.

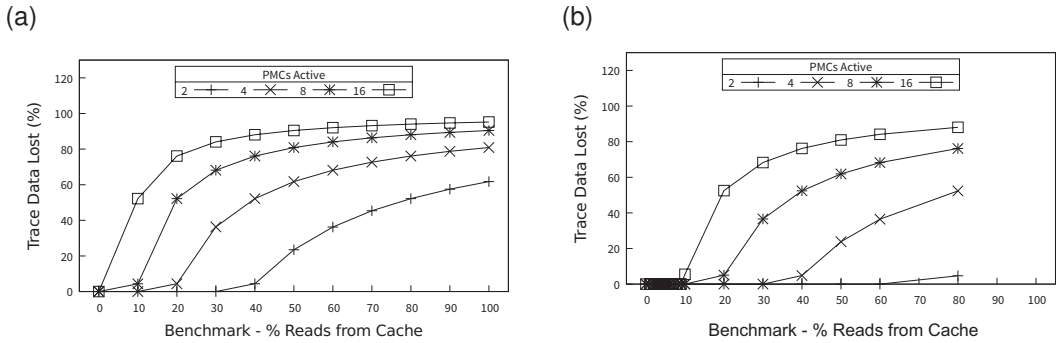

Fig. 4. Results for benchmark only accessing cache. X-axis: the % of memory access instructions in the benchmark. Y-axis: (a) and (b) % of trace data lost for PolyBlaze and ARM Cortex-A9 Systems, respectively.

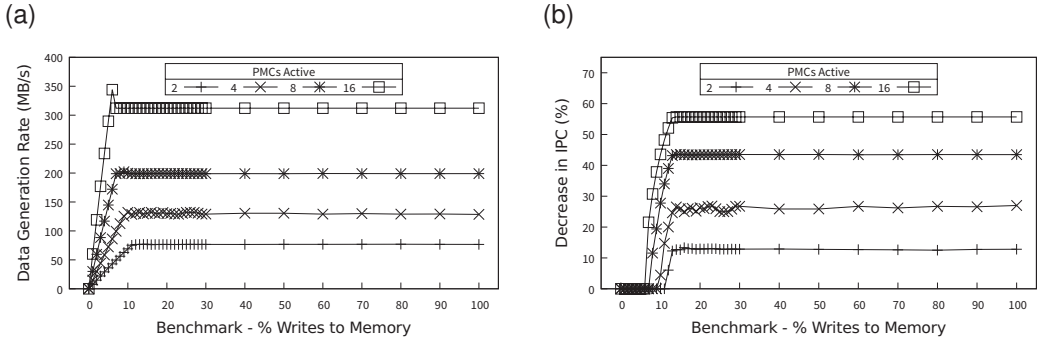

Fig. 5. Results for benchmark accessing system memory. X-axis: the % of memory access instructions in the benchmark. Y-axis: PolyBlaze System (a) Trace data generation rate in MB/s, and (b) Reduction of IPC relative to systems with no data collection.

B. Case Study

In the previous section, we used micro-benchmarks crafted to test the performance of data collection at the limits of system behaviour. In this section, we broaden our analysis by examining the performance of data collection using representative workloads from the PARSEC benchmark suite [1] executing on a Linux operating system in both a single-core and a 4-core SMP configuration.

Furthermore, we motivate the inclusion of low-overhead data trace collection in future CPU designs by showing an example of how existing tools would benefit from this functionality. Two existing software systems for *Non-Uniform memory-access* (NUMA) architectures, MemProf [33] and Carrefour [34], are examples of the software that could benefit from enhanced memory-access trace collection functionality. MemProf is a tool for profiling the remote memory-access patterns in programs, while Carrefour is a scheduling algorithm for operating systems running on NUMA architectures. Both currently use AMD *Instruction-Based Sampling* (IBS) to collect memory-access traces, which requires software intervention using interrupts. The authors of both papers noted the limitations caused by this approach, including high system overheads and a limit on the sample rate, as well as cases where they could not collect sufficient data for analysis. We use PMCs to collect the same information that was previously collected by MemProf and Carrefour using AMD IBS (virtual addresses, physical addresses, and time-stamps). We examine the quality of the data collected and the impact on application performance. Testing is conducted on both a single-core system (allowing for comparisons against our micro-benchmark results) and a 4-core SMP system (similar to the original MemProf and Carrefour research), with full implementation of memory-access trace collection using the PolyBlaze architecture.

Two benchmarks were selected from the PARSEC benchmark suite [1] for this analysis. The first is *streamcluster*, which was used in the previous MemProf and Carrefour work, providing a performance reference. This benchmark is known to have the highest level of remote memory access of all the PARSEC benchmarks. We have balanced this with *blackscholes*, which has the smallest working set (and thus highest execution speed) of the PARSEC benchmarks. These benchmarks can be run in 1 thread for the single core system or 4 threads pinned to different CPUs in the 4-core SMP system. For the multi-core system, we also consider a third case in which *streamcluster* is run with 2 threads on two CPUs and *blackscholes* is run with 2 threads on the other two CPUs.

Table I shows the captured baseline performance data, including the percentage of total instructions that are load and store instructions, and the percent of instructions resulting in a data cache miss. The IPC averaged over the entire execution and total execution time is also recorded. We omit instruction cache misses, as they were negligible in all cases. When the same benchmark was run with multiple threads in parallel, the benchmarks execute approximately 3-3.2 times as fast, with the same increase in IPC, although the proportion of memory-access remained fixed. *streamcluster* has a relatively large number of memory reads and experienced data caches misses, as compared *blackscholes*, and consequently sees a

TABLE I
BASELINE BENCHMARK PERFORMANCE CHARACTERISTICS

Benchmark	Load Instr.	Store Instr.	D-Cache Misses	IPC	Time (s)
1 CPU/Thread					
streamcluster	4.65%	20.07%	1.05%	0.084	1565
blackscholes	4.23%	4.33%	0.00%	0.192	3239
4 CPUs/Threads					
streamcluster	4.88%	20.48%	1.03%	0.277	486
blackscholes	4.23%	4.33%	0.00%	0.579	1072
Both	4.46%	12.22%	0.53%	0.495	1053

lower IPC. These values are averaged over the entire execution. Unlike the micro-benchmarks, these workloads will experience bursts of memory traffic, as well as interactions in the memory system between threads.

1) Results: To provide a comparison to data collection using AMD IBS by MemProf [33] and Carrefour [34], we enable 3 PMCs per CPU to collect virtual and physical memory addresses and a timestamp for each memory-access. We adjust sampling rate from 1 to 1/32, as well as 1/8,000, the highest sample rate used in MemProf [33]. We examine the performance of both a single-core system, to provide the most direct comparison to our micro-benchmark results, as well as a 4-core SMP system, which better reflects contemporary systems and the test platform for MemProf and Carrefour.

While the PMCs in the previous section had a fixed 4 KB internal buffer, we examined if altering the size of this buffer had an impact on the potentially bursty memory-access behaviour of real workloads. Adjusting the buffer size between 1 KB and 4 KB, we saw less than 1% impact in all cases.

For the single core system, we found that for both benchmarks the data generation rate was a maximum of 24 MB/s with our maximum sampling rate of 1, and decreased roughly linearly as the sampling rate decreased. As a result, no trace data was lost. The corresponding decrease in IPC was less than 5% for a sampling rate of 1 and less than 1% in all other cases. These results are consistent with the results of the micro-benchmarks, given the proportion of reads and writes from Table I.

When executed in four threads on an SMP system, Table I shows the benchmarks executing over three times as fast. This increases the amount of trace data being generated and collected (Figure 6 (a)), and results in more congestion in the memory system. The loss of data in Figure 6 (b), particularly for *blackscholes*, shows the impact of data operations occurring within the cache system. The high level of memory traffic from this data collection is partly responsible for the impact on IPC seen in Figure 6 (c).

The results show that hardware memory-access trace collection is very favourable compared to collection with AMD IBS. We are able to achieve four cycle resolution with less than 10% overhead and 16 cycle resolution with less than 5% in all cases. When tested with 8K cycle resolution, our system experienced less than 0.2% impact on IPC and increase in execution time in all cases, as compared to the 20% overhead reported with AMD IBS using the same sample rate. Furthermore, in all sample rates of 1/2 or less, we collected all data. By replacing data collection in these algorithms with a PMC, we have shown that PMCs addresses the problem of high overhead in techniques that sample HPCs. In addition to improving the performance of existing algorithms, a greatly improved sample rate allows for new types of analysis, such as detecting regions of different memory behaviour and faster reaction to bursty memory access behaviour.

As discussed in the previous section, the PolyBlaze test platform has a memory frequency fixed at 4 times the CPU frequency, while commercial CPUs often operate at higher frequencies than the memory system. However, these results show that even if this difference would result in 10x the data generation rate, data collection would still not result in system overhead with a 1/8,000 sampling rate.

These results also show the complicated interactions between processes and data collection in a multi-processor system. While the mixed benchmark showed intermediate IPC and data access behaviour in Table I, it showed the highest data generation rate and level of data collection loss in Figures 6 (a) and (b). *streamcluster* showed the highest impact on IPC and the mixed benchmark the lowest at a sample rate of 1 in Figure 6 (c), but the opposite was true at lower sample rates where no trace data was lost. Hardware capture of performance data on a running system gives access to this kind of valuable insight about system performance.

V. CONCLUSION AND FUTURE WORK

This work presents a detailed study of the performance impact of runtime hardware collection of memory-access traces. We show that the performance and overhead impact of data collection is related to application execution speed and the bandwidth between the CPU, data collection, and memory. In addition to the size of the memory interface, the operating frequency and data size at the point of memory arbitration between data collection and the CPU is critical. The issue rate between the CPU and the data collection at this point needs to be considered, as an imbalance can inadvertently give priority to data collection. Cache policy is important during worst-case memory-access behaviour, with systems using a write-through cache seeing higher execution speed (and sensitivity to interactions with data collection). The performance impact of cache line evictions illustrates why data collection should bypass the cache system to maximize system performance.

Our case study achieved sampling rates orders of magnitudes higher than previous work using AMD IBS with no performance impact. High sample rates with low system overhead for multi-threaded workloads in an SMP system show the value of hardware-based memory-access trace collection for software designers. Even on a high performance architecture with the CPU running faster than memory, data collection at the highest sample rates commercially possible would still have no impact on application performance.

Future work includes a more detailed characterization of relative CPU and memory system frequencies to further un-

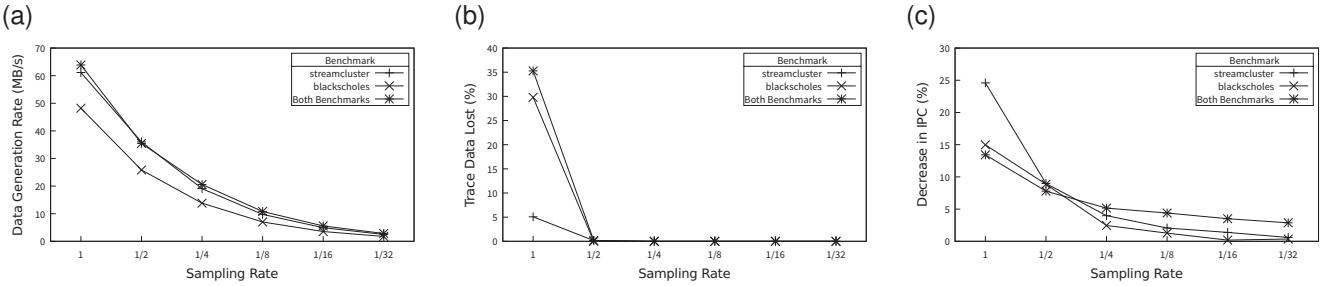

Fig. 6. Performance impact of benchmarks on a 4 CPU PolyBlaze system with different sampling rates. X-axis: Sampling rate used for data collection. Y-axis: (a) Trace data generation rate in MB/s, (b) Percentage of trace data lost, and (c) Reduction of IPC relative to a system with no data collection.

derstand impacts on memory system congestion. Also, the interactions between processes and the impact of data collection on unrelated processes should be characterized to fully understand the impact of data collection in multi-core architectures. The application of data compression techniques and previously explored adaptive sampling rates [33] to reduce memory bandwidth should also be considered.

REFERENCES

[1] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[2] E. Matthews, L. Shannon, and A. Fedorova, "Polyblaze: From one to many bringing the microblaze into the multicore era with linux smp support," in *Field Programmable Logic and Applications (FPL), 2012 22nd Int. Conf. on*, 2012, pp. 224–230.

[3] M. Paolieri, E. Quiñones, and F. J. Cazorla, "Timing effects of ddr memory systems in hard real-time multicore architectures: Issues and solutions," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 1s, pp. 64:1–64:26, Mar. 2013.

[4] Advanced Micro Devices, *AMD64 Architecture Programmer's Manual*, June 2015, vol. 2, ch. 13.

[5] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual*, June 2015, vol. 3B, no. 325462-055US, ch. 18.

[6] ARM Ltd., *ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition*, May 2014, no. ARM-DDI-0406C, ch. C12.

[7] J. L. Greathouse, Z. Ma, M. I. Frank, R. Peri, and T. Austin, "Demand-driven software race detection using hardware performance counters," in *ACM SIGARCH Computer Architecture News*, vol. 39, no. 3. ACM, 2011, pp. 165–176.

[8] J. Arulraj, P.-C. Chang, G. Jin, and S. Lu, "Production-run software failure diagnosis via hardware performance counters," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1. ACM, 2013, pp. 101–112.

[9] C. Malone, M. Zahran, and R. Karri, "Are hardware performance counters a cost effective way for integrity checking of programs," in *Proc. 6th ACM workshop on Scalable trusted computing*. ACM, 2011, pp. 71–76.

[10] X. Wang and R. Karri, "Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters," in *Proc. 50th Design Automation Conference*, ser. DAC '13. IEEE, 2013, pp. 1–7.

[11] M. Aldham, J. Anderson, S. Brown, and A. Canis, "Low-cost hardware profiling of run-time and energy in fpga embedded processors," in *Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE Int. Conf. on*, Sept 2011, pp. 61–68.

[12] A. Nair, K. Shankar, and R. Lysecky, "Efficient hardware-based non-intrusive dynamic application profiling," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 3, pp. 32:1–32:22, May 2011.

[13] N. Ho, P. Kaufmann, and M. Platzner, "A hardware/software infrastructure for performance monitoring on leon3 multicore platforms," in *Field Programmable Logic and Applications (FPL), 2014 24th Int. Conf. on*, Sept 2014, pp. 1–4.

[14] E. Gibert, R. Martinez, C. Madriles, and J. Codina, "Profiling support for runtime managed code: Next generation performance monitoring units," *Computer Architecture Letters*, vol. 14, no. 1, pp. 62–65, Jan 2015.

[15] V. M. Weaver and S. A. McKee, "Can hardware performance counters be trusted?" in *Workload Characterization, 2008. IISWC 2008. IEEE Int. Symp. on*. IEEE, 2008, pp. 141–150.

[16] D. Zaparanuks, M. Jovic, and M. Hauswirth, "Accuracy of performance counter measurements," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE Int. Symp. on*. IEEE, 2009, pp. 23–32.

[17] V. M. Weaver, D. Terpstra, and S. Moore, "Non-determinism and overcount on modern hardware performance counter implementations," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE Int. Symp. on*. IEEE, 2013, pp. 215–224.

[18] I. Corporation. (2016) Intel vtune amplifier 2016. [Online]. Available: https://software.intel.com/en-us/intel-vtune-amplifier-xe

[19] AMD. (2016) Codexl: Powerful debugging, profiling and analysis. [Online]. Available: http://developer.amd.com/tools-and-sdks/opencl-zone/codexl/

[20] R. Vitillo, "Performance tools developments," in *Future computing in particle physics*, 2011.

[21] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual*, June 2015, vol. 3C, no. 325462-055US, ch. 36.

[22] Advanced Micro Devices, "Lightweight profiling specification," Tech. Rep. 43724, August 2010.

[23] ARM Ltd., "Arm embedded trace macrocell architecture specification," Tech. Rep. ARM-IHI-0064D, February 2016.

[24] ——, "Coresight technical introduction," Tech. Rep. ARM-EPM-039795, August 2013.

[25] S. Chen, B. Falsafi, P. B. Gibbons, M. Kozuch, T. C. Mowry, R. Teodorescu, A. Ailamaki, L. Fix, G. R. Ganger, B. Lin, and S. W. Schlosser, "Log-based architectures for general-purpose monitoring of deployed code," in *Proc. 1st Workshop on Architectural and System Support for Improving Software Dependability*, ser. ASID '06. New York, NY, USA: ACM, 2006, pp. 63–65.

[26] X. Yang, S. M. Blackburn, and K. S. McKinley, "Computer performance microscopy with shim," in *Proc. 42nd Int. Symp. on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015, pp. 170–184.

[27] S. Chen, P. B. Gibbons, M. Kozuch, and T. C. Mowry, "Log-based architectures: Using multicore to help software behave correctly," *SIGOPS Oper. Syst. Rev.*, vol. 45, no. 1, pp. 84–91, Feb. 2011.

[28] E. Matthews, L. Shannon, and A. Fedorova, "A configurable framework for investigating workload execution," in *Field-Programmable Technology (FPT), 2010 Int. Conf. on*, Dec 2010, pp. 409–412.

[29] L. Shannon, E. Matthews, N. C. Doyle, and A. Fedorova, "Performance monitoring for multicore embedded computing systems on fpgas," *CoRR*, vol. abs/1508.07126, 2015. [Online]. Available: http://arxiv.org/abs/1508.07126

[30] E. Matthews, N. C. Doyle, and L. Shannon, "Design space exploration of l1 data caches for fpga-based multiprocessor systems," in *Proc. 2015 ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 156–159.

[31] Xilinx Inc., "Zynq-7000 all programmable soc technical reference manual," Tech. Rep. UG585, February 2015.

[32] ——, *MicroBlaze Processor Reference Guide*. [Online]. Available: www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/mb_ref_guide.pdf

[33] R. Lachaize, B. Lepers, and V. Quema, "Memprof: A memory profiler for numa multicore systems," in *Presented at the 2012 USENIX Annu. Tech. Conf.* Boston, MA: USENIX, 2012, pp. 53–64.

[34] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema, and M. Roth, "Traffic management: A holistic approach to memory placement on numa systems," in *Proc. 18th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '13. New York, NY, USA: ACM, 2013, pp. 381–394.