Memory-Efficient Hardware Performance Counters with Approximate-Counting Algorithms

Jingyi Xu, Sehoon Kim, Borivoje Nikolic, Yakun Sophia Shao

University of California, Berkeley

***Abstract*—Hardware performance counters are special registers on processors that track the hardware activities. While the performance counter data are useful for many applications, there are challenges in efficiently collecting many event statistics simultaneously, due to the limited number of performance counters on chip. We propose an efficient hardware performance counter design that uses approximate-counting algorithms to improve the number of events tracked on-chip without incurring significant memory overhead. These counters are more memory efficient because they increment counts according to a dynamic probability and approximate the exact counts. Compared with multiplexed hardware performance counters, our approximate hardware counters have a statistically provable memory-accuracy trade-off and are entirely managed in hardware.**

I. Introduction

The event statistics collected by hardware performance counters are valuable for various applications, e.g. workload characterization [1], performance optimization [2], task scheduling [3], and CPU power modeling [4]. The performance-counter-based analysis is typically most useful when multiple event counts are provided, to correlate the activities to performance and pinpoint the root cause of performance bottlenecks. However, due to area and power constraints, only a limited number of hardware performance counters are built into processors, typically between 4 and 20 [5] - [8]. In contrast, more than 200 monitoring events are supported by current generation processors, including instruction counts, memory accesses, CPU pipeline status [8]. Normally, one counter is only set to keep track of one event at any time of the program execution, i.e., "one-counter-one-event". In this case, collecting a large set of performance data requires rerunning a workload multiple times. In addition to incurring extra cost in time and resources, on cloud platforms for instance, rerunning workloads may not be able accurately capture the behaviors due to non-deterministic events. Therefore, it is not straightfoward to efficiently collect a large set of performance events.

The multiplexing technique [9] has been proposed to monitor a large number of events using a small number of hardware performance counters. By measuring the events only over a fraction of execution time, and extrapolating the overall counts from the sampled counts, the multiplexed counters are able to cover more events in a small number of runs. However, the multiplexed counters are inaccurate, and there is no guarantee on the error bound. Several techniques have been proposed to improve the accuracy of multiplexed counters, including rate-of-change event schedulers [10] and data-cleaning-based post-processing [11]. Despite their complexities, the error rate of these techniques are typically between 5% to 30%.

In this work, we propose applying approximate-counting algorithms to build efficient hardware performance counters. The approximate-counting algorithms are a subset of sketching algorithms that use a small amount of memory to count a large number of events. The clear memory-accuracy trade-off makes them a good fit for performance monitoring in system design. In particular, the approximate counters have been applied to distributed computing [12] and network monitoring [13], and have shown reduction in memory requirement and improved accuracy. Our implementation of approximate hardware performance counter achieves an average accuracy of 25% while using only half the memory of deterministic counters.

II. Background

Sketching algorithms are a family of algorithms that uses "sketches" as data structures to compactly store information about massive datasets, with statistically provable trade-offs between memory and accuracy. The data compression is typically done using probabilistic techniques [14].The sketches approximate the original datasets and act as surrogates of the original datasets for subsequent queries and computations. Each sketch is problem-specific, answers one particular question, and has unique structures. Many sketches require no revisits to the data, so they are particularly useful in streaming applications, e.g., software-defined traffic measurement [13].

Approximate-counting algorithms increment their counts probabilistically instead of deterministically to save memory bits. When the count values are queried, the counter can approximate the exact value with the value it stores and the increment probability with a specific error bound. While this paper focuses on the Morris' algorithm [15] invented by Robert Morris in 1977, there are many variants of it, e.g. Sampled-Log Approximate Counting [16], Flexible Approximate Counters [17]. These counters have been used ML topic modelling in distributed systems [12].

Morris' Algorithm uses compact data structures and simple update/query algorithms to count the number of events in a stream. The desired accuracy can be controlled by choosing the counter bit-width, and/or update base-exponent and by averaging, making it a good fit for hardware performance counters. When an event occurs, the Morris counter is incremented with probability $\frac{1}{\alpha^X}$, where α is a parameter typically between

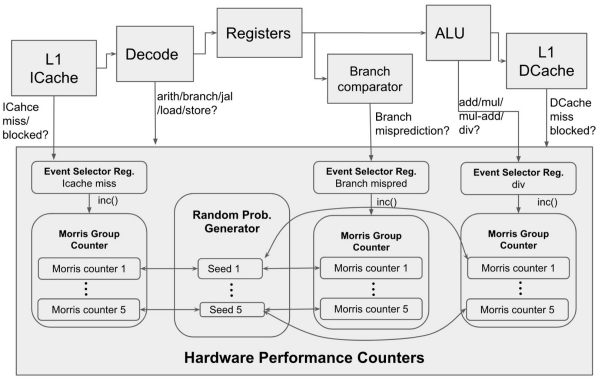

Fig. 1. Rocket core with Approximate Performance Counters.

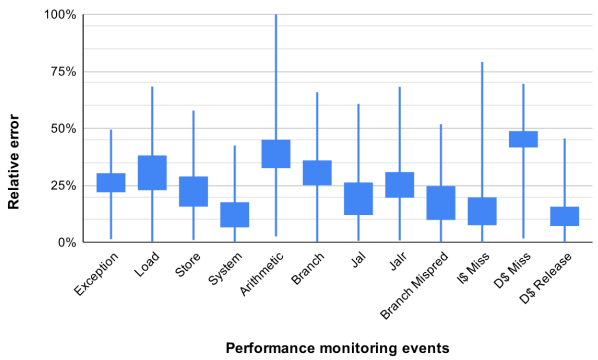

Fig. 2. Relative errors of the proposed approximate counters. The max 129% relative error of arithmetic is left out to better visualize the lower range.

1 and 2 and X is value in the counter before the update. When the number of events seen so far is queried, the value α^X is returned. This counter is an unbiased estimator of the actual count, however, it can have a high variance when α is close to 2. In particular, when $\alpha = 2$, the failure probability $P(|c' - c| > \epsilon c) < \frac{1}{2\epsilon^2}$, where c is the actual number of events and c' is the output of the counter.

One of the commonly-used strategies to decrease the variance is to run multiple independent Morris counters in parallel as a Morris group counter and average their outputs. The failure probability of a Morris group counter is $\frac{1}{groupsize}$ of a single Morris counter. It effectively improves the accuracy without adding excessive overhead on the hardware.

III. Methodology

We leverage the Rocket Chip generator [18] to implement Morris counters in hardware. By default, it generates a RISC-V in-order pipelined CPU core with 29 general 64-bit hardware performance counters, and 29 event selector registers.

Overall Architecture: Figure 1 illustrates the overall architecture of our probabilistic hardware performance counter. First, user registers the specific events to be monitored, e.g., I-cache miss and branch mispredcition into the event selector registers. Those event signals are tracked during the program execution and, if they occur, the system calls `inc()` of the corresponding Morris group counter. Then the Morris group counter forwards that signal to all the Morris counters to probabilistically update their internal states. As the Morris counter is connected with the random probability generator module, it can probabilistically accept or deny the increment signal according to its internal state. In particular, the Morris counters in the same Morris group counter do not share the same random probability generator so that they can behave independently. Finally, if user queries the Morris group counter, i.e., `query()`, it queries all the Morris counters and returns the average of their predictions.

Hardware Morris Counter: We use 6-bit Morris counters that can each store up to 2^{64} counts in expectation. Since a single Morris counter tends to produce a large variance, 5 Morris counters are grouped into one Morris group counter. All counters in a group counter count the same event and the average of the counter values is used. Comparing with the deterministic counter that requires 64-bit for counting up to 2^{64}, the Morris group counter only needs 30-bit, which allows $2\times$ improvement in terms of storage utilization.

Hardware Probability Generation: Since Morris counters rely on randomness for the probabilistic increment, we need a random probability generator module that takes as input X and outputs 1 with the probability of $\frac{1}{2^X}$. A random number of X-bit is generated by a Fibonacci linear-feedback shift register and then reduced to 1-bit with bitwise OR operation, which produces the output of 0 with the probability of $\frac{1}{2^X}$.

IV. Evaluation

We generate our Rocket Core with built-in deterministic-increment performance counters and approximate counters for RTL simulation. To gague the accuracy of our sketching hardware counter, we run two benchmarks from the standard RISC-V testbench library, spmv(sparse matrix-vector multiplication) and vvadd(vector-vector add). Each benchmark are first run on the standard RISC-V processor to set baselines, and the simulated 50 times on the processor with Morris counters.

The maximum, 3rd quartile, 1st quartile and minimum of relative errors of each performance events are aggregated in Figure 2. Overall, the average errors are between 10% and 30%. The minimum errors stay consistently within 5%. In contrast, there is more variability in the maximum errors, ranging from 40% to 120%. The relative error can get larger than 100% due to 2^X reconstruction. However, the theoretical error bound show that when $\alpha = 2$, $P(|c' - c| > 0.75c) < \frac{1}{2 \cdot 0.75^2} = 0.89$, so relative errors over 75% are less likely to occur.

V. Conclusion

We propose hardware performance counters with approximate-counting algorithms that use only half of the bit-width of regular performance counters and show 5% decrease in relative errors compared to previous works. For future work, we would like to experiment with techniques to further improve accuracy of approximate hardware counters.

VI. Acknowledgement

This work was supported in part by the NSF Award 1955450 and in part by ADEPT Lab industrial sponsors and affiliates.

REFERENCES

[1] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, and R. Hundt, "Google-wide profiling: A continuous profiling infrastructure for data centers," IEEE Micro, vol. 30, no. 4, pp. 65–78, 2010.

[2] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "Cpi2:cpu performance isolation for shared compute clusters," in Proceedings of European Conference on Computer Systems (EuroSys), 2013.

[3] S. Ziemba, G. Upadhyaya, and V. Pai.,2004. "Analyzing the Effectiveness of Multicore Scheduling Using Performance Counters".

[4] R. Zamani and A. Afsahi, "A study of hardware performance monitoring counter selection in power modeling of computing systems," 2012 International Green Computing Conference (IGCC), pp. 1-10, 2012.

[5] Intel, "Intel 64 and ia-32 architectures developer's manual," 2017. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and- technology/ 64-ia-32-architectures- software- developer- manual-325462.html

[6] Arm, "Arm cortex-a53 mpcore processor technical reference manual," 2014. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500d/DDI0500_cortex_a53_r0p2 trm.pdf

[7] I. Advanced Micro Devices, "Bios and kernel developer's guide for amd family 10h processors," 2013. [Online]. Available: http: //support.amd.com/TechDocs/31116.pdf

[8] Intel,"Intel® 64 and IA32 Architectures Performance Monitoring Events Guide," 2017. [Online]. Available: https://software.intel.com/sites/default/files/managed/8b/6e/335279 _performance_monitoring_events_guide.pdf

[9] J.M, May, "MPX: Software for multiplexing hardware performance counters in multithreaded programs," *Proc. 15th International Parallel and Distributed Processing Symposium. IPDPS 2001. IEEE*, 2001.

[10] R.V. Lim, D. Carrillo-Cisneros, W. Alkowaileet, and I. Scherson, "Computationally efficient multiplexing of events on hardware counters," *Linux Symposium*, 2014.

[11] Y. Lv, B. Sun, Q. Luo, J. Wang, Z. Yu, X. Qian, "Counterminer: Mining big performance data from hardware counters," *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE*, 2018.

[12] G.L. Steele and J. Tristan. "Adding approximate counters," *Proc. the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '16)*, 2016.

[13] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proc. 10th USENIX Conf. NSDI*,pp. 29–42, 2013.

[14] D. Ahfock, WJ. Astle, S. Richardson, "Statistical Properties of Sketching Algorithms," arXiv preprint arXiv:1706.03665, 2019. https://arxiv.org/abs/1706.03665.

[15] R.Morris, "Counting large numbers of events in small registers," *Communications of the ACM, 21 (1976), 840-842.*, 1976.

[16] A. Cvetkovski, "An algorithm for approximate counting using limited memory resources," *Proc. 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems,* pages 181–190, 2007.

[17] S. Mitchell and D. Day. "Flexible approximate counting," In IDEAS '11: Proc. 15th Symp. *International Database Engineering Applications,* pages 233–239, 2011.

[18] K. Asanovic, et al. "The Rocket Chip Generator," Technical Report No. UCB/EECS-2016-17, [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.pdf