Establishing a base of trust with performance counters for enterprise workloads

Andrzej Nowak, *CERN openlab and EPFL* Ahmad Yasin, *Intel* Avi Mendelson, *Technion* Willy Zwaenepoel, *EPFL*

Abstract

Understanding the performance of large, complex enterprise-class applications is an important, yet non-trivial task. Methods using hardware performance counters, such as profiling through event-based sampling, are often favored over instrumentation for analyzing such large codes, but rarely provide good accuracy at the instruction level.

This work evaluates the accuracy of multiple event-based sampling techniques and quantifies the impact of a range of improvements suggested in recent years. The evaluation is performed on instances of three modern CPU architectures, using designated kernels and full applications. We conclude that precisely distributed events considerably improve accuracy, with further improvements possible when using Last Branch Records. We also present practical recommendations for hardware architects, tool developers and performance engineers, aimed at improving the quality of results.

1. Introduction

Optimizing large codes is difficult. It requires dealing with millions of lines of code developed by many engineers, processing diverse data sets that run on complicated warehouse-scale systems [1]. Furthermore, it requires a deep understanding of the specific hardware architecture [3] or mandates use of recently published cycle-accounting methods suitable for out-of-order cores [27][36]. Modern processors feature counters that aim to assist users in understanding how well their application is performing. The hardware component in charge of gathering this information is usually called the Performance Monitoring Unit (PMU). The methodologies based on using these counters are well-established, especially in the HPC domain [2].

Many profilers [4][5][6] provide the means to narrow down the information gathered to select locations in the code that may cause an inefficiency. Methodologies that perform their analysis at basic-block granularity provide high source-level resolution, and are therefore the focus of this paper. Accurately obtaining basic block execution counts is a key problem facing the abovementioned profilers when analyzing enterprise-class, large-scale object-oriented workloads. These have challenging long-tail profiles with few hotspots where instrumentation is usually unsuitable [39].

This paper surveys Event Based Sampling (EBS) techniques and the parameters that influence measurement accuracy. We conduct our study on instances of modern enterprise processors. We develop a set of microbenchmarks, use a subset of the CPU2006 workloads, and use a large production workload from the CERN datacenter [7], running in deployments exceeding 300'000 cores. The contributions of this paper are:

- A first, to our knowledge, experimental evaluation of the accuracy of EBS techniques – serving as a necessary base for further work in this field.
- The conclusion that precise events considerably improve accuracy with little or no added cost, with further improvements resulting from the collection and analysis of Last Branch Records.
- Recommendations for hardware architects, tool developers and performance engineers working with EBS, aimed at improving the quality of results.

2. Motivation and background

2.1. The need for accurate basic-block profiles

Accurate and efficient basic block execution counts are important for a wide spectrum of use cases. Basic block graphs can rely on accurate basic block profiles. These can greatly improve the compiler's capability to make better decisions on inlining, while increasing code locality. Code level energy-efficiency monitors demand accuracy by using metrics such as Watts-per-instruction (WPI) [8][9]. Loop tripcounts are widely used for a variety of purposes [10][11][12], but are hard to obtain with pure EBS methods. In automatic or semi-automatic optimization of whole complex applications consisting of millions of lines of code, performance tuning must be driven by rather precise methods – e.g., basic block execution counts or precise function/method-granularity profiles.

2.2. Increasing processor complexity

Common tasks described in section 2.1 are made even more difficult by the ever-increasing complexity of modern processors. Out-of-order execution, superscalar pipelines, speculation and hardware prefetching increase performance, but complicate analysis [27]. At the same time, the core infrastructure of the PMUs has not progressed much, with only incremental updates between generations. Consequently, attributing samples to the exact program location that triggers a

performance event is becoming more and more difficult.

2.3. Growing code size and sophistication

The sheer number of lines of code (LOC), many of which are active, is steadily increasing across the application spectrum. An example study [14] focused on the SPEC CPU suite indicates the CPU2000 suite's codebase was below 1000 KLOC (thousand LOC) with no C++ code included, while CPU2006 reached over 3000 KLOC with only 25% of the benchmarks written in C++. Heavily object-oriented scientific toolkits, such as Geant4 [15, p. 4] and ROOT [16], are self-contained software packages that reach millions of LOC. Yasin et al. [13] demonstrate how abstractions incur significant performance overhead in data analytics and lead to code fragmentation with very small code-block sizes (typical ratios of instructions to branches taken around 6-12 with three lead JVMs).

2.4. Applicability of standard PMU methods

Traditional PMU-based methods as deployed in common profilers - such as perf [4], oprofile, VTune [5] or Code Analyst [6] - are rather designed for HPC and steady-state traditional workloads. They do not cope well with large object-oriented codes with highly fragmented profiles having thousands of entries and very few hotspots. A good tool has to consciously adjust the setup of the underlying generic hardware mechanisms for the best possible handling of the characteristics of a workload.

Our research indicates that more advanced methods, that can be particularly competitive and accurate, are infrequently used (e.g., PBA profiler [17]).

2.5. Related work

In recent years, the topic of PMU trust has been touched on by several publications. Works by Mytkowicz [18] and Weaver [3][19][20] analyze and identify CPU and OS effects that influence accuracy in *counting-mode* – some of which we also observe in our study of *sampling (i.e. EBS)*. Chen et al. have discussed some key sampling effects such as skid and shadow [21]. We go beyond these works and beyond Zaparanuks et al. [37] by focusing on the root causes of sampling inaccuracy, showing where to look for control over multiple events and how to identify optimal configurations for enterprise-like workloads.

3. Comments on sampling

3.1. Event Based Sampling

EBS exploits the capability of the PMU to count pre-defined hardware events and to generate an interrupt when the counted event is observed N specified times. N is called the sampling period, and the interrupts are called Performance Monitoring Interrupts (PMI). Instruction pointer locations are sampled on PMIs, and their distribution is used to generate profiles.

According to Chen et al. [21] and Levinthal [23], errors in the distribution of samples are the result of three major factors: (1) *synchronization* of monitored code with the sampling period, (2) the *sampling skid effect*, when the address reported by the hardware sample does not necessarily match the address of the instruction causing counter overflow, and (3) the *sampling shadow effect,* when instructions in the shadow of a long latency instruction get low sample counts.

When seeking more accurate profile information about basic block execution counts, tools average samples across all instructions in the same block. While helpful, such mitigation is still insufficient for short blocks, such as e.g., jump tables, as samples have higher chances of getting attributed to adjacent blocks.

3.2. Last Branch Records

Some processors feature a branch recording facility, such as Last Branch Record (LBR) in the x86 architecture, which records the addresses of the most recently executed branches. These facilities can be used for basic block execution counts, as suggested by Levinthal [23] and implemented in the emerging Gooda [26][36] and PBA [17] tools. These tools and their relevant heuristics are scarcely documented and not yet widely adopted in the community. Consequently, we implement our own version of LBR analysis, as documented below.

An LBR facility has a number of stacked entries, which represent source-target pairs $<S_i, T_i>$ of branches executed by the processor. When sampling on the *Taken Branches* event, branches between a target T_i and the next source S_{i+1} in the stack are not taken. Thus, all basic blocks between T_i and S_{i+1} are executed exactly once.

3.3. Profiling accuracy

In order to estimate the accuracy of PMU-based sampling, we cross-reference results with instrumentation-based basic block counts obtained through Pin [25] ("REF").

$$\text{Accuracy Error (x)} = \frac{\sum_{i \in BB} |(\text{BB}_{\text{x}}[\text{i}] - \text{BB}_{\text{REF}}[\text{i}])|}{\text{net\_instruction\_count}}$$

For a given sampling method *x*, our accuracy error is defined as the sum of all deviations between the *x* method and the REF method, of the number of instructions executed in each basic block. This error is normalized to the total number of instructions executed. Ideally, we would like the accuracy error as close as possible to zero.

3.4. Evaluation of existing methods

Out of the combinations of profiling approaches assessed and discussed in Section 4 (also see Appendix), we choose three particular groups of methods that exemplify how sampling can be used:

- *Classic Sampling* is a representative, widely employed method of the first group. It uses an imprecise counter, where the sampling period is fixed, even and not randomized. No filtering is applied. This is the default in mainline tools such as perf [4], where an architectural event is typically set to capture a sample every ~1 millisecond.
- *Precise Sampling* represents the advanced group of methods. It uses a general-purpose counter with a precise-sampling mechanism featured by the PMU, where the period is a prime number, preferably randomized.
- *LBR Sampling* is performed when using a retired Taken Branches event[1]. The full contents of each collected LBR stack serve as the basis for basic block execution counts, while the address that comes with the PMI is ignored.

4. Experimental environment

4.1. Hardware and software setup

We evaluate out-of-order processors from the x86 family, as implemented by Intel and AMD. From the AMD side, we choose a representative of the "Magny-Cours" family, the 12-core 6164 HE. Software support for this family was the most robust at the time of writing. On the Intel side, we choose the Xeon X5650, as the representative of the 1st Core i7 family, a.k.a. Westmere, and a Xeon E3-1265L, representing the 3rd generation Core family, a.k.a. Ivy Bridge. Again, stable software support and existing experience played a role in our choice. Frequency scaling and "turbo mode" are disabled.

To obtain PMU samples we use a modified version of the perf utility in Linux 3.6.6, on RHEL6 compatible systems. Perf has a very fluid codebase, which impacts measurement overheads much more than hardware does [38], and this particular version is used throughout the whole study. Non-essential services/daemons are disabled.

Each of our kernels, emphasizing specific difficulties leading to reduced accuracy, is measured five times.

4.2. PMU configurations and events

The methods described in this section are presented in more detail in Table 3 in the Appendix.

Magny-Cours does not feature LBRs, nor a fixed architectural counter. The latter could be an issue with the version of perf available at time of writing. The standard event of choice was RETIRED_INSTRUCTIONS. Instruction Based Sampling (IBS) is the precise mechanism offered by AMD. We program the PMU to sample with prime and non-prime periods. Due to perf limitations, software-based period randomization was unavailable, but the hardware randomizes the 4 least significant bits.

On Westmere, we choose to work with the fixed instructions retired counter and with the programmable instructions retired event supplemented with Precise Event Based Sampling (PEBS). LBRs are sampled with the BR_INST_EXEC:TAKEN event, with PEBS disabled.

On Ivy Bridge, we use the instructions retired event on the fixed counter (INST_RETIRED:ANY) as well as the recently added Precisely Distributed event (INST_RETIRED:PREC_DIST, a.k.a. PDIR, [24]). LBRs are sampled using BR_INST_RETIRED:NEAR_TAKEN, with PEBS disabled.

4.3. Workloads

4.3.1. *Latency-biased kernel*

The latency-biased kernel is the simplest form of emulation of workloads with basic blocks with non-uniform execution times. Here, a loop executes a relatively costly calculation when a certain condition is true:

```
while (n--) ((n%2) ? x /= y : x += y);
```

Such code occurs in practice, for example, when a pre-computed value is returned in the standard case, and re-computed otherwise. Typically, the PMU would bias samples towards the long latency instruction, thus distorting overall results [21].

4.3.2. *Call chain kernel*

This kernel is a simple 10-deep call chain enveloped by a loop. Since the functions do equal work, they are expected to produce equal numbers of samples.

This example serves as a vivid illustration of potential sampling bias on call chains - such as those commonly seen in object-oriented programming with frequently called short methods.

4.3.3. *G4Box test*

The G4Box micro-benchmark, written in C++, executes only two functions, with an even work split. It could be thought of as a heavier version of our Latency Biased kernel. The length of the main function depends on the input data.

This kernel is particularly difficult for hardware sampling, since it contains a chain of tests and branches that generates short basic blocks – a good case for LBR analysis.

[1] BR_INST_RETIRED.NEAR_TAKEN on Ivy Bridge

4.3.4. *Geant4 test40*

Test40 is a kernelized doppelganger of large Geant4 applications. In this test, an electron travels through a detector with a very simple geometry, triggering physics processes on its way. The signature workload is therefore a collection of small, fragmented methods, conditionally executed depending on where the particle is and what matter it interacts with.

4.3.5. *Applications*

First, we select a non-HPC subset of both INT and FP benchmarks from the SPEC2006 CPU suite: 429.mcf, 453.povray, 471.omnetpp and 483.xalancbmk. This subset, written in C/C++, has (to some extent) the characteristics of enterprise workloads [39], as described by Wong in [28] and [29]. Some enterprise vendors commonly use these benchmarks as proxies for real applications.

Second, the FullCMS application is based on parts of Geant4 and is designed to simulate complex physics events taking place in one of CERN's Large Hadron Collider particle detectors. It is similar to the enterprise class of workloads in the sense that it executes similar fragmented operations, albeit using floating point rather than integers. This production-grade workload has successfully served as an enterprise "proxy" in the past and runs on ~300'000 cores.

5. Results

5.1. Kernel results

The results in Table 1 present accuracy errors as defined in Section 3.3 of the various sampling methods defined in Section 4.2. Overall results show that:

- LBR-based methods are highly beneficial, significantly reducing errors by up to 18x (3-6x on average).
- Progressive improvements from randomization and period adjustment are observed as better techniques are applied.
- The precisely distributed event (PDIR) significantly improves results across all kernels and especially for Latency Biased.
- AMD systems are consistently burdened with high error rates, worsening when built-in hardware randomization was used.

On Latency Biased, we observe improvements introduced by the Ivy Bridge precisely distributed PDIR event. These accuracy boosts, on the other hand, are not observed on the Westmere microarchitecture, where that event is not featured.

Results for the Callchain kernel show how applying prime as well as randomized periods gradually improves accuracy as we move to the right with improvements. In the Ivy Bridge case, combining the LBR-based IP+1 fix with PDIR (see Appendix) gives the best results. While there is no definitive indication of the reason, it would appear that out-of-order clustering of uops, which causes uops to be retired in bursts, is responsible for this characteristic.

On testG4Box, medium error rates are reduced when LBR is employed. This is because this test case is dominated by very short basic blocks, which challenge sampling in general. The LBR-based technique addresses this issue by averaging samples across the last 15 uninterrupted basic block segments, which extends the effective number of instructions that the sample corresponds to.

On test40, we see that Westmere suffers from distribution problems linked to the sampling event, which disappear on Ivy Bridge. Employing LBRs alleviates these issues on both platforms.

5.2. Application results

Table 2 shows error averages for applications. The general observations are as follows:

Table 1: Sampling methods used on kernels and their errors according to our accuracy metric (lower is better).

		INST RETIRED default	Precise event or uops					Taken Branches Retired
		Even period	Even period		Prime period			
		Fixed	Fixed	Rand.	Fixed	Rand.	Rand. Fix IP+1 (LBR)	LBR
AMD	latencybias	0.84	N/A	0.88	N/A	0.84	N/A	N/A
	callchain	0.61		0.80		0.73		
	testG4Box	0.28		0.29		0.35		
	test40	0.60		0.91		0.85		
WSM	latencybias	1.57	1.57	1.57	1.59	1.59	1.57	0.09
	callchain	0.48	0.53	0.41	0.68	0.38	0.37	0.21
	testG4Box	0.32	0.36	0.29	0.75	0.30	0.42	0.05
	test40	0.58	0.57	0.57	0.55	0.55	0.54	0.16
IVB	latencybias	1.21	0.28	0.19	0.22	0.19	0.23	0.07
	callchain	0.59	0.48	0.19	0.24	0.15	0.10	0.12
	testG4Box	0.18	0.17	0.16	0.15	0.17	0.18	0.05
	test40	0.60	0.20	0.19	0.14	0.14	0.13	0.11

- Randomization has little to no impact on full applications. Full applications often do not have specific loop trip-counts to synchronize with sampling periods, as tight kernels can have.
- Using a counter with precise distribution and applying an LBR address fix provides good results.
- Pure LBR basic block counts further improve accuracy (except for FullCMS). Overall improvement is 4-5x over the classic case and 1-10x over the precise case.

The classic method registers high overall error rates, much improved with the precise event on IVB. Also, the LBR method is noticeably better than precise sampling, especially so in the case of mcf.

As a side note, our FullCMS experiments showed that already choosing an EBS method on a counter with precise distribution and applying an LBR-based IP offset correction (but not full LBR sampling) improves average per basic block accuracy by 5x over the leftmost classic case. Pure LBR-based results, however, do not bring improvement over the precise method in this case, since the workload has similar characteristics to the callchain kernel.

LBR, while of great benefit to performance monitoring activities, is not entirely free of issues. None of the methods produces the top 10 functions from the FullCMS profile in the right order.

Table 2: Errors per machine/app (lower is better).

		INST RETIRED Default	Precise; Fixed period	Precise; Prime period	LBR
AMD	mcf	0.66	1.225	1.212	N/A
	povray	0.48	0.545	0.557	
	omnetpp	0.80	1.018	1.013	
	xalancbmk	0.53	0.606	0.606	
	FullCMS	0.53	0.611	0.606	
WSM	mcf	0.44	0.432	0.446	0.194
	povray	0.50	0.511	0.514	0.177
	omnetpp	0.67	0.682	0.679	0.302
	xalancbmk	0.46	0.485	0.484	0.087
	FullCMS	0.55	0.547	0.547	0.177
IVB	mcf	0.50	0.306	0.305	0.034
	povray	0.49	0.164	0.161	0.123
	omnetpp	0.65	0.246	0.249	0.116
	xalancbmk	0.56	0.341	0.340	0.112
	FullCMS	0.55	0.179	0.177	0.120

6. Recommendations

6.1. Recommendations for tool developers

First, we recommend that tool developers make distinctions between similar performance events. Users of perf, PAPI or OProfile are presented with opaque tools that obscure events and make adjustments (such as those of the sampling period) hard. For example, in the case of standard perf, a recompilation and installation of an extra library (libpfm4) is required to obtain reasonable access to hardware performance events.

Second, sampling periods need to be chosen with a dose of care. Prime number periods reduce the risk of synchronizing with the workload, and randomization further improves results on artificial kernels, but neither produced noticeable improvements on our large benchmarks (unlike what is reported in [21]). As of today, neither perf nor major commercial tools support fixed period randomization.

Third, the LBR-based methods we evaluated allow for enhanced degrees of accuracy, which – with some post-processing – could serve as input to PGO, code coverage or other sensitive optimization techniques. Only a couple tools (PBA [17] and GOODA [26][36]) use LBRs to obtain basic block execution counts, and their documentation only sparsely describes the methods employed.

6.2. Recommendations for PMU hardware designers

The IP+1 inaccuracy fix in sample addresses based on an LBR sourced address (not the full LBR) can lead to good improvements, especially for branchy code with a high rate of calls or taken branches. Implementing such functionality in hardware would not only remove the workaround burden in drivers, but also avoid collisions on LBRs – a valuable single resource – with other filtered collections such as call-stack mode.

A precise instruction event in AMD's IBS is missing, which led us to use precise uops instead.

6.3. Recommendations for application optimizers

We have shown that the methods used to obtain performance data matter and influence potential conclusions. Overall, we recommend to sample on a modern platform with support for precise distributed events, while using a prime period. Kernel-like code additionally benefits from more frequent sampling periods and period randomization. For ultimate sampling performance, we recommend liaising with tool developers to employ LBR-based methods that maximize accuracy.

7. Conclusions and Summary

In this short survey of a somewhat underexplored area, we quantify the level to which choices related to performance monitoring methods influence results. The precise events introduced in the Ivy Bridge microarchitecture considerably improve accuracy with little or no added cost. Period randomization shows improvements on kernels, but not on complete applications. LBR-based methods improve results even more over precise counters, and work especially well on Westmere machines.

REFERENCES

[1] L. A. Barroso and U. Holzle, "The Case for Energy-Proportional Computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[2] P. J. Mucci, S. Browne, C. Deane, and G. Ho, "PAPI: A portable interface to hardware performance counters," in *Proc. Dept. of Defense HPCMP Users Group Conference*, 1999, pp. 7–10.

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

[4] A. Carvalho de Melo, "The New Linux 'perf' tools." Linux Kongress, 2010.

[5] Intel Corporation, "Intel VTune Amplifier XE 2013," 2012. [Online]. Available: http://software.intel.com/en-us/intel-vtune-amplifier-xe. [Accessed: 22-Nov-2012].

[6] P. J. Drongowski, A. M. D. C. A. Team, and B. D. Center, "An introduction to analysis and optimization with AMD CodeAnalyst Performance Analyzer," *Advanced Micro Devices, Inc*, 2008.

[7] S. Banerjee, "Readiness of CMS simulation towards LHC startup," *Journal of Physics: Conference Series*, vol. 119, no. 3, p. 032006, Jul. 2008.

[8] S. Schubert, D. Kostic, W. Zwaenepoel, and K. Shin, "Profiling Software for Energy Consumption," in *Proceedings of the IEEE International Conference on Green Computing and Communications (GreenCom)*, 2012.

[9] M. D. DeVuyst, "Efficient Use of Execution Resources in Multicore Processor Architectures," University of California, San Diego, 2011.

[10] P. Saxena, P. Poosankam, S. McCamant, and D. Song, "Loop-extended symbolic execution on binary programs," in *Proceedings of the eighteenth international symposium on Software testing and analysis*, New York, NY, USA, 2009, pp. 225–236.

[11] T. Sherwood and B. Calder, "Loop Termination Prediction," in *High Performance Computing*, M. Valero, K. Joe, M. Kitsuregawa, and H. Tanaka, Eds. Springer Berlin Heidelberg, 2000, pp. 73–87.

[12] K. Muthukumar and G. Doshi, "Software Pipelining of Nested Loops," in *Compiler Construction*, R. Wilhelm, Ed. Springer Berlin Heidelberg, 2001, pp. 165–181.

[13] A. Yasin, Y. Ben-Asher, and A. Mendelson, "Deep-dive Analysis of the Data Analytics Workload in CloudSuite," presented at the IISWC'14, 2014.

[14] J. L. Henning, "SPEC CPU suite growth: an historical perspective," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 65–68, Mar. 2007.

[15] J. Apostolakis, "Geant4—a simulation toolkit," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 506, no. 3, pp. 250–303, Jul. 2003.

[16] R. Brun and F. Rademakers, "ROOT — An object oriented data analysis framework," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 389, no. 1–2, pp. 81–86, Apr. 1997.

[17] "Intel Performance Bottleneck Analyzer." Intel Corporation, 2011.

[18] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Producing wrong data without doing anything obviously wrong!," in *ACM Sigplan Notices*, 2009, vol. 44, pp. 265–276.

[19] V. Weaver, "Can Hardware Performance Counters Produce Expected, Deterministic Results?," presented at the 3rd Workshop on Functionality of Hardware Performance Monitoring, 2010.

[20] V. Weaver, D. Terpstra, and S. Moore, "Non-determinism and overcount on modern hardware performance counter implementations," in *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, 2013.

[21] D. Chen, N. Vachharajani, R. Hundt, S. Liao, V. Ramasamy, P. Yuan, W. Chen, and W. Zheng, "Taming hardware event samples for FDO compilation," in *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, New York, NY, USA, 2010, pp. 42–52.

[22] A. Shye, M. Iyer, T. Moseley, D. Hodgdon, D. Fay, V. J. Reddi, and D. A. Connor, "Analysis of path profiling information generated with performance monitoring hardware," in *9th Annual Workshop on Interaction between Compilers and Computer Architectures, 2005. INTERACT-9*, 2005, pp. 34 – 43.

[23] D. Levinthal, "Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 processors." Intel Corporation, 2009.

[24] "Intel® 64 and IA-32 Architectures Software Developer Manuals," *Intel*. [Online]. Available: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html. [Accessed: 26-Sep-2014].

[25] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized

program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, New York, NY, USA, 2005, pp. 190–200.

[26] Google, *Gooda - a pmu event analysis package (http://code.google.com/p/gooda/)*. 2012.

[27] A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture," presented at the 2014 IEEE International Symposium Performance Analysis of Systems and Software (ISPASS), 2014.

[28] M. Wong, "C++ benchmarks in SPEC CPU2006," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 1, p. 77, Mar. 2007.

[29] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[30] D. Chen, N. Vachharajani, R. Hundt, X. Li, S. Eranian, W. Chen, and W. Zheng, "Taming Hardware Event Samples for Precise and Versatile Feedback Directed Optimizations," *IEEE Transactions on Computers*, vol. PP, no. 99, p. 1, 2011.

[31] T. Ball and J. R. Larus, "Optimally profiling and tracing programs," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 4, pp. 1319–1360, Jul. 1994.

[32] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. Sweeney, "We have it easy, but do we have it right?," in *IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008*, 2008, pp. 1 –7.

[33] K. Walcott-Justice, J. Mars, and M. L. Soffa, "THeME: a system for testing by hardware monitoring events," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, New York, NY, USA, 2012, pp. 12–22.

[34] M. L. Soffa, K. R. Walcott, and J. Mars, "Exploiting hardware advances for software testing and debugging: NIER track," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 888 –891.

[35] S. Narayanasamy, T. Sherwood, S. Sair, B. Calder, and G. Varghese, "Catching accurate profiles in hardware," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings*, 2003, pp. 269 – 280.

[36] A. Nowak, D. Levinthal and W. Zwaenepoel, "Hierarchical Cycle Accounting – a new method for application performance tuning" in *2015 IEEE International Symposium on Performance Analysis of Systems and Software,* 2015.

[37] D. Zaparanuks, M. Jovic and M. Hauswirth, "Accuracy of performance counter measurements" in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, 2009.

[38] G. Bitzes, A. Nowak, "The overhead of profiling using PMU hardware counters", *CERN openlab report,* 2014.

[39] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G. Wei and D. Brooks, "Profiling a Warehouse-Scale Computer" in *International Symposium on Computer Architecture (ISCA)*, Jun 2015.

APPENDIX:

Table 3: An overview of reviewed sampling methods

	Method	**Parameters**			**Comments**	**Drawbacks**
		Period size (Example value)	**Period Rando-mization**	**Event** (Intel nomenclature)		
Classic method	Default	Round (2'000'000)	No	INST_RETIRED.ANY (non-precise)	Used by default in many tools. Uses a fixed-function counter to free up general counters.	The period is fixed and round which increases the risk of synchronization, the hardware event is imprecise
Precise methods	Precise event	Round (2'000'000)	No	INST_RETIRED.ALL (precise)	Uses a precise mechanism to capture the event location (IP+1)	The distribution of samples is not guaranteed
	Precise event with randomization	Round (2'000'000)	Yes		A randomized sampling period to avoid synchronization risk	As above
	Precise event with prime period	Prime number (2'000'003)	No		The introduction of prime numbers reduces resonance which leads to improved accuracy	Lack of randomization and overall low accuracy in some cases alike the Latency-Biased kernel
	Precise event with randomized prime period	Prime number (2'000'003)	Yes		Randomization applied on the prime period further improves accuracy	Still overall low accuracy in some cases
	precise event with distribution fix plus IP+1 offset fix	Prime number (2'000'003)	Yes/No	INST_RETIRED. PREC_DIST (precisely distributed)	To remedy skid, the top address from the LBR backtrace is used to determine which basic block the trigger occurred in; thus fixing IP+1 and enhancing accuracy.	Good for large basic blocks. Some inaccuracies for small ones
LBR method	Last Branch Record	N/A	N/A	BR_INST_RETIRED. NEAR_TAKEN	Full LBR-based basic block execution count accounting with manageable errors per basic block	Errors can still reach 30-50% of basic block execution count (for some basic blocks). Overhead (in collection and post-processing)