Center for Embedded Computer Systems
University of California, Irvine

Metrology applied to performance analysis using hardware counters

Rosario Cammarota and Alexander Veidenbaum

Technical Report CECS-10-01
May 7, 2010

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{rosario.c,alexv}@ics.uci.edu
http://www.cecs.uci.edu/

Metrology applied to performance analysis using hardware counters

Rosario Cammarota and Alexander Veidenbaum

Technical Report CECS-10-01
May 7, 2010

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{rosario.c,alexv}@ics.uci.edu
http://www.cecs.uci.edu

Abstract

Performance evaluation and analysis are used to understand and improve computer system performance. One common approach is to measure performance indicators of interest during execution of a set of benchmark programs on a given system. Processor architecture, compiler, operating system, etc. and their interaction are often not predictable and can introduce random variability in such measurements. For instance, complex out-of-order processors with speculation may have a different memory utilization, instructions per cycle, pipeline stalls, branch prediction results, for each execution of a program. This raises the question of accuracy in collecting execution-based measurements.

Modern microprocessors contain hardware performance counters to make such measurements and to collect dynamic information on the state of a processor [13], [15]. A common practice is to collect multiple measurements per counter and to report the average value and the standard deviation. However, only a limited number of hardware counters can be collect in one execution of a program. Therefore, the number of times a benchmark suite needs to be executed to collect all the counters of interest may be quite large. The question then is, what is the minimum number of samples required required to measure all parameters with a required accuracy? Another question is if measurements performed in separate benchmark executions can be combined. Finally, there is an issue of how to deal with measurement errors in systems that can collect only a small number of counters per run. The are the issues addressed in this work.

We propose a methodology for hardware counter based measurements that answers the above

questions. We observe that metrology and the performance evaluation of computer systems deal with similar issues and use a metrological approach. In particular, the methodology introduces a criteria to identify whether multiple executions of a given benchmark are performed under "the same conditions". The proposed methodology is verified in a case study using SPEC CPU 2006 [7, 28] benchmarks executed on the Intel Core 2 Duo processor [15]. The methodology is shown to produce accurate and reproducible measurements.

Contents

Metrology applied to performance analysis using hardware counters

Rosario Cammarota and Alexander Veidenbaum

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA

{rosario.c,alexv}@ics.uci.edu
http://www.cecs.uci.edu

Abstract

Performance evaluation and analysis are used to understand and improve computer system performance. One common approach is to measure performance indicators of interest during execution of a set of benchmark programs on a given system. Processor architecture, compiler, operating system, etc. and their interaction are often not predictable and can introduce random variability in such measurements. For instance, complex out-of-order processors with speculation may have a different memory utilization, instructions per cycle, pipeline stalls, branch prediction results, for each execution of a program. This raises the question of accuracy in collecting execution-based measurements.

Modern microprocessors contain hardware performance counters to make such measurements and to collect dynamic information on the state of a processor [13], [15]. A common practice is to collect multiple measurements per counter and to report the average value and the standard deviation. However, only a limited number of hardware counters can be collect in one execution of a program. Therefore, the number of times a benchmark suite needs to be executed to collect all the counters of interest may be quite large. The question then is, what is the minimum number of samples required required to measure all parameters with a required accuracy? Another question is if measurements performed in separate benchmark executions can be combined. Finally, there is an issue of how to deal with measurement errors in systems that can collect only a small number of counters per run. The are the issues addressed in this work.

We propose a methodology for hardware counter based measurements that answers the above questions. We observe that metrology and the performance evaluation of computer systems deal with similar issues and use a metrological approach. In particular, the methodology introduces a criteria to identify whether multiple executions of a given benchmark are performed under "the same

conditions". The proposed methodology is verified in a case study using SPEC CPU 2006 [7, 28] benchmarks executed on the Intel Core 2 Duo processor [15]. The methodology is shown to produce accurate and reproducible measurements.

1 Introduction

Multi-core or Chip Multiprocessor (CMP) architectures have become ubiquitous. Their architecture and that of a single core are quite complex and are not easy to program. Understanding the program behavior on a given core architecture is important for a number of reasons, such as improving performance and/or reducing power consumption, and/or improving system utilization. In this context, performance analysis can play a critical role by suggesting either how to better utilize the resources available in a chip multi-processor or in a multi-core systems, or to improve existing programming models and techniques, or to introduce new compiler optimizations. This work deals with the accuracy of measurements for execution-based performance evaluation of computer systems. This is a challenging task [2], [5], [9], [8].

Modern microprocessors are equipped with a special monitoring hardware, including hardware event counters, to collect dynamic information on the state of the processor. However the number of counters that can be used concurrently is limited, and the exact number on depends on a particular architecture and its performance monitoring unit[1].

The use of performance counters is attractive since it is the only performance evaluation methodology that gives information on the execution of the entire system, including an operating systems, user level libraries, and an application. Given the limit on the number of simultaneously useable counters, it is necessary to execute a program multiple times to collect all required counter data. However the combined effect of imprecise sampling, microarchitecture features like speculative execution, and multiprogramming environment results in different counter values for each execution of a given benchmark. This requires collecting multiple samples for each event counter in order to give a quantitative evaluation of the performance parameters, according to a standard metrology-based approach [12]. Classic metrology addresses issues like reproducibility of measurements and compatibility among measurements. We adapt and use metrology to define our measurement methodology.

Since multiple executions of a given benchmark are necessary to complete a given set of measurements, and the state of the system can vary dynamically, the understanding of whether measurements are executed under the same conditions is one of the main problems addressed by our methodology.

[1] For instance, in the case of Intel Core 2 Duo processor the maximum number of counters that can be read concurrently is four.

2 Terminology

Given a physical quantity of interest (a *measurand*), metrology standardizes [19] the expression of each measurement from a collection of samples in terms of a sample mean and of absolute or relative uncertainty. It also provides the rules to propagate the uncertainty across elementary operations. A measurement process associates quantitative information to a measurand(s). The measure is expressed by assigning a value to the measured quantity and a related uncertainty. The uncertainty itself provides quantitative information on the dispersion of the values that could be reasonably attributed to the measurand.

Although counters are not related to any physical quantity recognized in metrology, in this technical report we consider a value of a counter as the quantity that we want to measure for a given workload. With this meaning the value of a counter relative to the execution of a workload is our measurand.

We call the measure of a counter a direct measure since measurement tools are able to access counters directly. All other measures obtained by performing elementary operations on counters are called indirect measures.

The uncertainty is usually expressed in terms of a confidence interval, that is a range of values where the measurand value is most likely to fall. The probability that the measurand value falls inside the confidence interval is called a confidence level. The confidence level indicates the probability that a measure is contained inside a certain interval obtained from the sample mean, the sample standard deviation and a slip factor.

The reading of each counter is modeled as a stochastic variable X assuming a positive integer value[2] with certain expected mean μ and variance σ.

According to the Guide to the expression of Uncertainty in Measurements (GUM) [19], standard uncertainty, that is uncertainty expressed as a standard deviation, can be evaluated either statistically, as an estimate of the standard deviation of the mean of a set of independent observations.

R executions (or runs) of a workload will produce R values of a given counter, $x_1, x_2, \ldots, x_R$[3]. For each counter, the average value and the standard deviation are computed from the samples. Let $\{x_1, x_2, \ldots, x_R\}$ being R samples of a counter, we use the following unbiased estimators of the mean and of the variance [19]:

$$\mu_R = \frac{\sum_1^R x_r}{R} \tag{1}$$

$$\sigma_R^2 = \frac{\sum_1^R (x_r - \mu_R)^2}{R-1} \tag{2}$$

μ_R is called the sample mean and σ_R is called the sample variance. According to [19], the expression

[2] We assume that overflow of counters doesn't happen or that is correctly managed by the tool that reads the values.

[3] We assume the samples are collected by independent runs, which solves the problem of whether two or more runs of a given application can be considered independent.

of the sample uncertainty is given by the following formula:

$$u_R = \frac{\sigma_R}{\sqrt{R}} \tag{3}$$

Given a slip (or confidence) factor $k = 1, 2, 3$ the measure of the counter is expressed as follows:

$$X = \mu_R \pm k \times u_R \tag{4}$$

[19] also indicates how the uncertainty propagates among indirect measures. An indirect measure is specified by a deterministic function called measurement equation, 5, that associates the measurand Y to the measured quantities X_r.

$$f : X_1, X_2, \ldots, X_R \rightarrow Y \tag{5}$$

An estimation of Y, denoted by y, is achieved from the measurement equation using input estimates x_r, of the R input quantities, as shown in 6.

Measurement equation	Indirect measures
$Y = aX$	$y = ax \pm au_x$
$Y = X_1 + X_2$	$y = (x_1 + x_2) \pm \sqrt{(u_1^2 + u_1^2)}$
$Y = \frac{X_1}{X_2}$	$Y = \left(\frac{x_1}{x_2}\right) \pm \frac{1}{x_2}\sqrt{(u_1^2 + \left(\frac{x_1}{x_2}\right)^2 u_2^2)}$

Table 1: Examples of indirect measurements

$$y = f(x_1, x_2, \ldots, x_R) \tag{6}$$

Let u_r be the uncertainty of the estimate x_r, and assume R independent observations over X_r. According to [10] and [19], the uncertainty of y can be obtained from the law of propagation of the uncertainty, as shown in 7.

$$u_y = \sqrt{\sum_1^R f_{X_r}(x_r) \times u_r^2} \tag{7}$$

where $f_{X_r}(x_r) = \frac{\partial f}{\partial X_r}(x_r)$. Composite uncertainty for simple expressions of f are reported in Table 1. All the definitions above introduced are dependent from the number of samples considered.
At the same time, the number of samples considered influences the time necessary to obtain any single measure.
In [12] the following algorithm is presented to estimate the number of samples needed to obtain a single measure, based on reasonable constraints. Given the sequence of results from the experiments x_r , with $r \in \{1, 2, \ldots, R\}$, and assuming a desired value σ_{ref} of uncertainty in our measurements, we repeat independent experiments, collect data and compute the sample mean and the sample variance for each new sample that is added. We continue adding samples until the sample variance over the square root of the number of sample is less than σ_{ref}, i.e. until $u_R \leq \sigma_{ref}$. Once the process is finished, R is the number of samples that must be considered, the sample mean

is considered as a good measure of the value of parameter we are looking for, with the accuracy given by the initial constraint.
The sample uncertainty related to the measure is estimated from the ratio between the sample variance and the square root of the number of samples.
Finally the measure is expressed by combining the sample mean with the sample uncertainty, for a given slip factor $k = 1, 2, 3$, as shown in 4. The slip factor indicates the probability or the confidence that a sample will fall into the interval $(\mu_R - k \times u_R, \mu_R + k \times u_R)$[4].
Measures obtained under the same conditions and expressed with the same confidence have confidence intervals overlapping. This last statement allow us to test whether the conditions of the system vary during the collection of the samples. In fact, variation in the state of the system during the execution time is most likely to occur due to a different distribution of the samples. Potentially, performance evaluation where samples are not distributed in the same way, will lead to an incorrect analysis.

3 The methodology

Most modern high-performance microprocessors use out-of-order and speculative execution. In the former case, instructions are executed in a different order than they appear in the original. In the latter case, a processor issues more instructions than the program needs to complete. For these microprocessors the Instructions "Retired" indicates the number of instructions that are executed to completetion. The count does not include partially processed instructions executed, for instance due to branch mispredictions.
For a given workload we collect multiple samples of instructions retired plus additional counters that can be read concurrently with the instructions retired. The observed variation in the number of Instructions Retired must be bounded if different executions of the workload are performed under the same conditions.
This approach reduces the number of samples that need to be collected, and by testing compatibility among different measures of the Instructions Retired allows us to identify if the measures are performed under the same conditions of the system or not.
The ideas discussed so far lead us to define the following methodology to collect performance hardware counters.
Let $B = \{b_1, b_2, \ldots, b_N\}$ be the set of workloads, and let the counter X_n representing Instructions Retired event for the benchmark b_n. Let $C = \{c_1, c_2, \ldots, c_M\}$ being the set of the other counters which are different from X_n. We iterate the following procedure for each benchmark, that is for $n = 1, 2, \ldots, N$.

1. Given a value for uncertainty σ_{ref} we compute the number R of iterations needed to measure the Instructions Retired X_n.

2. We consider a partition of C containing L subsets of events that can be collected concurrently

[4] k=1,2,3 indicates a confidence of 68.26%, 95.45%, 99.99% respectively.

with X_n. Let us call this set $P = \{p_1, p_2, \ldots, p_L\}$. Here $L \leq M$ and must be choosen to be a multiple of R [5].

3. We collect the counters reported in the set 8, by performing L independent executions of the benchmark b_n[6].

$$E_{n,L} = \{X_n\} \times P = \{(X_n, p_1), (X_n, p_2), \ldots, (X_n, p_L)\} \tag{8}$$

4. Given the L samples of the Instructions Retired for the benchmark b_n we group the measurements in $h = \frac{L}{R}$ subsets, and obtain h measures of the Instructions Retired.

5. If the h measures of the Instructions Retired are not compatible, the executions of the workload have not been done under the same conditions.

6. If they are compatible we assume that the execution have been done under the same conditions and the value of the counters obtained so far are considered as characterizing the n^{th} workload[7], and the counters can be used to analyze[8].

4 The experiments

The methodology described above is applied to an experimantal setup illustrated in table 4. The advanced power management of the reference architecture were disabled, both from the BIOS and from the operating system. This avoids possible false measurements due to the adaptation of the system to different (Voltage, Frequency) operating points.
The counters have been collected using Intel VTune Performance Analyzer for Linux SMP 64-bit [17], [18], [19], [20], [21], [22]. Intel VTune has been configured to use Event Base Sampling, and the sample after value has been selected automatically by running the benchmark suite one time with the auto-calibration option enabled. The resulting configuration of Intel VTune is reporter in table 2. The Sample after value is evaluated in order to produce 1000 of samples per second. The observation window, that is the temporal window inside which Intel Vtune collects the counters has been selected in order to be larger than the longest executing benchmark. The values in table 3 are used during the experiments, and the auto-calibration is disabled.
We used Linux kernel v2.6 [32] configured in text mode, however multiprogramming features were enabled. The system was isolated from any potential asynchronous sources of interference by disabling the network subsystem.
SPEC CPU 2006 [28] was used as the benchmark suite, with GNU compiler suite v4.2 [31] to build the benchmarks.

[5] The partitioning of the set of counters is totally architecture dependent.

[6] $E_{n,L}$ indicates the Execution of the n^{th} benchmark L times

[7] For events related to retired instructions, a methodology to estimate the uncertainty among ratios of counters over instructions retired is proposed in [1].

[8] E.g. starting from providing indirect measures like the Instruction Per Cycle, instruction breakdown, branch misprediction rate, cache misses rate, etc.

System Name	core4.ics.uci.edu
Processor	Intel(R) Core(TM)2 Duo CPU E7200, 2.53 GHz
Architecture	45 nm
Front side bus	1066 MHz
Intel VT	No
Main memory	2 GB
L1 I-Cache	32 KB, 64 B, 8-way
L1 D-Cache	32 KB, 64 B, 8-way
L2 I-Cache	3 MB, 64 B, 8-way
Compilers flags	-O2
OS	Linux Ubuntu server (kernel 2.6.22)

Table 2: Description of the system under test

Intel VTune parameter name	Intel VTune parameter value
Sample After Value	2526000
Observation Window	100000 [sec]

Table 3: Intel VTune sampling settings

4.1 Performance evaluation

We selected a subset of performance hardware counters, to capture the branch and cache miss behavior, to give a breakdown of the pipeline stalls and to report the Instruction Per Cycle (IPC).
The application of the methodology described in the previous section tells us that $R = 3$ samples of the Instructions Retired of each benchmark are required to estimate the Instructions Retired with an uncertainty that is less than $\sigma_{ref} \leq 1\%$.
A partitioning of the set of counters used is shown in 4.1.

p_1	$(CPU\_CLK\_UNHALTED.CORE)$
p_2	$(BR\_INST\_RETIRED.MISPRED, BR\_INST\_RETIRED.ANY)$
p_3	$(MEM\_LOAD\_RETIRED.L1D\_MISS)$
p_4	$(MEM\_LOAD\_RETIRED.L2\_MISS)$
p_5	$(RESOURCE\_STALLS.BR\_MISS\_CLEAR, RESOURCE\_STALLS.LD\_ST)$
p_6	$(RESOURCE\_STALLS.ROB\_FULL, RESOURCE\_STALLS.RS\_FULL)$

Table 4: A feasible partition of the set of counters, N=28, M=9, R=3, L=6, H=2, k=3

An explanation of the meaning of the counters is reported in [16], [17], [18], [19]. For each benchmark we considered the direct measures of $E_{n,l} = (INST\_RETIRED.ANY_n, p_l)$, with $N = 28$, the number of benchmark considered, and $L = 6$, the number of partition of C considered.

Since $R = 3$, we obtained $h = \frac{L}{R} = 2$ groups among which to control the compatibility of the measurements over the Instructions Retired for each benchmark. We choose to express the measurements with a coverage factor $k = 3$, that is for a confidence level of 99%. We found the measures compatible over all the set of measurements, therefore there was no need to repeat any set of measurements.
The maximum uncertainty evaluated on the Instructions Retired, among the benchmarks was less than 1%, as expected. The maximum uncertainty we observed on the measurands related to $L1D$ and $L2$ events, and stall events, was less than 8%, whether the maximum uncertainty we found among the branch events was less than 7%.
These observations provided an estimation of the $CPU\_CLK\_UNHALTED.CORE$ with maximum uncertainty among the benchmarks that is less 5%.
This additional set of measurements provided a measure of the Instructions Retired that was compatible with the previous two. Thus we computed the IPC and expressed it accordingly with 4.
We observed that the uncertainty obtained for the counters $INST\_RETIRED.ANY_n$ and $CPU\_CLK\_UNHALTED.CORE$ was relatively small, but their ratio that represents the IPC was computed with a resultant uncertainty that was somehow larger than 8%.
The proposed methodology allowed us to performe 28×6 executions instead of performing $c \times 28 \times 3 \times 9$ executions[9] and we obtain accurate description of the architectural behavior for the system used.

4.2 Performance analysis

From the counters we derive the following performance metrics or the actual measurand:

- *IPC* = $\frac{INST\_RETIRED.ANY}{CPU\_CLK\_UNHALTED.REF}$
- *Branch misprediction* = $\frac{BR\_INST\_RETIRED.MISPRED}{BR\_INST\_RETIRED.ANY}$
- *L1D cache misses* = $\frac{MEM\_LOAD\_RETIRED.L1D\_MISS}{INST\_RETIRED.ANY}$
- *L2 cache misses* = $\frac{MEM\_LOAD\_RETIRED.L2\_MISS}{INST\_RETIRED.ANY}$
- *Load/Store stalls* = $\frac{RESOURCE\_STALLS.LD\_ST}{INST\_RETIRED.ANY}$
- *ReOrder Buffer stalls* = $\frac{RESOURCE\_STALLS.ROB\_FULL}{INST\_RETIRED.ANY}$
- *Reservation Station stalls* = $\frac{RESOURCE\_STALLS.RS\_FULL}{INST\_RETIRED.ANY}$

The performance, in terms of IPC, is somewhat low - the average IPC is below 1.5 and it is less than 1 in many cases.
The larger contribution to the stalls inside the pipeline comes from reservation station stalls. This contribution accounts for the 23% of stall time, on average, among the benchmarks. It is followed by the load/store stall contribution, that accounts as the 17.2% in the average, either when the

[9]The constant c accounts the number of samples necessary to measure counters different from the Instructions Retired with a certain precision.

pipeline has exceeded load or store queue limits or when it is waiting to commit all stores. The reorder buffer stalls account for 11.8%. Branch misprediction, that results in a flush of the pipeline, accounts for only 4.8%, on average. This is in agreement with the low rate of branches mispredicted over branches retired, which is 8%, on average.
The memory behavior and the caches miss penalty have a major impact on performance in terms of IPC.

5 Conclusion

This technical report described a methodology to address the problem of how to deal with measurement errors in computer systems that can collect only a very small number of performance counters per run.
The methodology has been successfully applied to a modern out-of-order and speculative microprocessor.
The methodology was developed by approaching the problem of hardware counter data collection from a metrological perspective.
The main contributions of this methodology are 1) that it limits the number of counter samples that need to be collected to produce accurate measurements, and 2) the application of the concept of compatibility between measurements to test whether the measurements are performed under the same conditions. The latter helps us to determine whether a variation in the system setup occurred and impacted the measures.
The importance of having measures performed under the same conditions resides in the necessity to execute an application multiple times to collect all the counters necessary to analyze the program behavior. Thus the counters may be combined together to produce the measurands and the analysis is conducted as if all the counters were collected in a single execution.

References

[1] R. Cammarota, A metrology based approach to performance analysis using hardware counters, Master Thesis, University of California Irvine, 2008

[2] Mytkowicz, T., Diwan, A., Hauswirth, M., and Sweeney, P. F. 2009. Producing wrong data without doing anything obviously wrong!. ASPLOS '09

[3] Mirghafori, N., Jacoby, M., and Patterson, D. Truth in SPEC benchmarks. SIGARCH 1995)

[4] Ofelt, D. and Hennessy, J. L. 2000. Efficient performance prediction for modern microprocessors. SIGMETRICS Perform. Eval. Rev. 28, 1 (Jun. 2000), 229-239

[5] Henning, J.L., SPEC CPU2000: Measuring CPU Performance in the New Millennium, IEEE Computer, Vol. 33 (7), July 2000, pp. 28-35

[6] Ye, D. et al, Performance Characterization of SPEC CPU2006 Integer Benchmarks on x86-64 Architecture, Proc. of the IEEE International Workshop on Workload Characterization, Oct. 2006, pp. 120 127

[7] Performance Characterization of SPEC CPU Benchmarks on Intel's Core Microarchitecture based processor
www.spec.org/workshops/2007/austin/papers/

[8] Kuck, D., Davidson, E., Lawrie, D., Sameh, A., Zhu, C., Veidenbaum, A., Konicek, J., Yew, P., Gallivan, K., Jalby, W., Wijshoff, H., Bramley, R., Yang, U. M., Emrath, P., Padua, D., Eigenmann, R., Hoeflinger, J., Jaxon, G., Li, Z., Murphy, T., and Andrews, J. 1993. The cedar system and an initial performance study. SIGARCH 1993

[9] Carbone, P.; Buglione, L.; Mari, L.; Petri, D., "Metrology and Software Measurement: a Comparison of some Basic Characteristics," IMTC 2006.

[10] Bondavalli, A., Ceccarelli, A., Falai, L., and Vadursi, M. 2007. Foundations of Measurement Theory Applied to the Evaluation of Dependability Attributes. In Proceedings of the 37th Annual IEEE/IFIP international Conference on Dependable Systems and Networks (June 25 - 28, 2007)

[11] Sheldon M. Ross, Simulation, 4th Ed, Academic Press, San Diego, 2006

[12] BIMP, IEC, IFCC, ISO, IUPAC, IUPAP, and OIML., Guide to the expression of uncertainty in measurement, second edition,1995

[13] Sprunt, B., "The basics of performance-monitoring hardware," Micro, IEEE , vol.22, no.4, pp. 64-71, Jul/Aug 2002

[14] Olukotun, K. and Hammond, L. 2005. The Future of Microprocessors. Queue 3, 7 (Sep. 2005), 26-29

[15] Gochman et Al., Introduction to Intel Core Duo Processor Architecture, in Intel Technology Journal, Volume 10, Issue 2, 2006

[16] Cycle Accounting Analysis on Intel© Core™ 2 Processors
http://softwarecommunity.intel.com/isn/downloads/

[17] IA-32 Intel© Architecture Optimization Reference Manual, in
http://www.intel.com/design/Pentium4/manuals/248966.htm

[18] IA-32 Intel© Architecture Software Developers Manual Volume 3A: System Programming Guide, Part 1, in
ftp://download.intel.com/design/Pentium4/manuals/25366819.pdf

[19] Using Intel© VTune™ Performance Analyzer Events/ Ratios & Optimizing Applications
http://software.intel.com/en-us/articles/

[20] Aart J. C. Bik, David L. Kreitzer, Xinmin Tian: A Case Study on Compiler Optimizations for the Intel© Core™ 2 Duo Processor. International Journal of Parallel Programming 36(6): 571-591 (2008)

[21] An Introduction to Sampling and Time
http://www.intel.com/cd/software/products/ijkk/jpn/219345.htm

[22] Hardware based performance monitoring with VTune Analyser under linux
http://hassan.shojania.com/

[23] Schneider, F. T., Payer, M., and Gross, T. R. 2007. Online optimizations driven by hardware performance monitoring. In Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (San Diego, California, USA, June 10 - 13, 2007). PLDI '07. ACM, New York, NY, 373-382

[24] Itzkowitz, M., Wylie, B. J., Aoki, C., and Kosche, N. 2003. Memory Profiling using Hardware Counters. In Proceedings of the 2003 ACM/IEEE Conference on Supercomputing (November 15 - 21, 2003). Conference on High Performance Networking and Computing. IEEE Computer Society, Washington, DC, 17.

[25] Yong Luo; Cameron, K.W., "Instruction-level characterization of scientific computing applications using hardware performance counters," Workload Characterization: Methodology and Case Studies, 1998 , vol., no., pp.125-131, 1999

[26] Rose, L. D. 2001. The Hardware Performance Monitor Toolkit. In Proceedings of the 7th international Euro-Par Conference Manchester on Parallel Processing (August 28 - 31, 2001). R. Sakellariou, J. Keane, J. R. Gurd, and L. Freeman, Eds. Lecture Notes In Computer Science, vol. 2150. Springer-Verlag, London, 122-131

[27] Kyung, H., Park, G., Kwak, J. W., Jeong, W., Kim, T., and Park, S. 2007. Performance monitor unit design for an AXI-based multi-core SoC platform. In Proceedings of the 2007 ACM Symposium on Applied Computing (Seoul, Korea, March 11 - 15, 2007). SAC '07. ACM, New York, NY, 1565-1572

[28] SPEC CPU 2006
http://www.specbench.org/cpu2006/

[29] KleinOsowski, A. J. and Lilja, D. J. 2002. MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research. IEEE Comput. Archit. Lett. 1, 1 (Jan. 2002), 7

[30] Fadi N. Sibai, Al Ain, Gauging The OpenSourceMark Benchmark in Measuring CPU Performance, Seventh IEEE/ACIS International Conferente, 2008, pp. 433- 438

[31] GCC, the GNU Compiler Collection
http://gcc.gnu.org/

[32] The Linux Kernel Archives
http://www.kernel.org/