Just how accurate are performance counters?

Wendy Korn and Patricia J. Teller
The University of Texas at El Paso
The Department of Computer Science
El Paso, TX 79968-0518
{wcard,pteller}@cs.utep.edu

Gilbert Castillo
IBM
Austin, TX
castillog@netzero.com

Abstract

The wide use of performance counters by application developers and benchmarking teams gives evidence that performance counters are well worth the silicon and design time required to include them on modern microprocessors. These counters provide rudimentary performance measurements that may or may not be accurate. This paper presents our methodology for determining the accuracy of these counters as well as preliminary results of a study that, using this methodology, evaluates the accuracy of the R12000 performance counters with respect to eight of 30 measurable events. The results indicate that care must be taken when using data generated by performance counters because, in some cases, this data may lead to erroneous conclusions. This can occur when the granularity of the measured code is not sufficient to ensure that the overhead introduced by counter interfaces does not dominate the event counts.

1.0 Introduction

Performance counters are used to measure events that occur during program execution. Some events that can be measured are the number of instructions, loads, stores, and cycles executed, and the number of cache misses. Due to the counters being on-chip, counter updates do not perturb program execution or the recorded event counts. The MIPS R12000 is one of several processors that support this hardware feature. Other processors with performance counters include the DEC Alpha Series [7], the IBM PowerPC series [10, 11], the Intel P6 and Pentium series [9], and the Sun UltraSPARC series [12]. An event–monitoring device driver needs to be provided by the operating system in order to use any of these counters. In the case of the MIPS R12000, IRIX 6.5.2 provides software interfaces such as perfex and libperfex to access the counters.

For several years, on-chip performance counters have been used with confidence for performance analysis of applications [3, 4, 11, 13]. For example, in [3], different methods, including performance counters used in combination with SpeedShop [11], are used to tune the performance of an Alternating Direction Implicit (ADI) method for solving three-dimensional partial differential equations. In [13], cache results generated by performance counters that profile cache misses and other measurement methods are used to tune the application and attain a speedup of 1.3. In the following quotation, the authors summarize the perceived importance of performance counters: “the counters were well worth the silicon and design time to include them.” The authors believe that “counter data is important for application developers, and that counters should be documented and supported by high-level tools.” Addressing this need, [3] makes available to the computer science and engineering communities an API, or application programming interface, that facilitates access to on-chip hardware counters. These research efforts and others that utilize performance counters focus on enhancing the performance of applications by using the counters, assuming that they afford a reasonable degree of accuracy. In contrast, this paper investigates their accuracy.

Our interest in performance counters relates to our goal of studying application resource demands and formulating models that predict application execution time. These models are for use in the Performance Oriented End-to-end Modeling System (POEMS), a problem-solving environment that spans application software, operating system and runtime system software, and hardware architecture for end-to-end performance analysis of complex parallel/distributed systems [1]. To use performance counters in this work, we must understand their degree of accuracy. Since there is a lack of published documentation on the accuracy of this hardware feature, we developed a

methodology for determining the accuracy of on-chip performance counters.

In this paper we demonstrate our methodology by presenting: 1) three microbenchmarks used to study eight of 30 measurable MIPS R12000 events, 2) our experimental design, 3) predicted and collected data, and 4) analysis of results. The results indicate that under certain circumstances, performance counter data can be very inaccurate. Such a situation can occur when the granularity of the measured code is not sufficient to ensure that the event counts associated with the counter interface do not dominate the reported total event counts (i.e., the event counts associated with the measured code and those associated with the interface).

The paper is organized as follows. Section 2 presents the methodology used to evaluate performance counter accuracy, including a description of the studied events, a discussion of the microbenchmarks for estimation and measurement of the events, an overview of the MIPS R12000 performance counters and the perfex and libperfex interfaces, and a brief description of sim-outorder [4], the simulator used to collect comparative event counts. The predicted and collected data are presented and analyzed in Section 3. Finally, in Section 4, conclusions are made about the accuracy of the performance counters and future research is discussed.

2.0 Methodology

The methodology used to study the accuracy of performance counters is an extension of that used in [6], a master's project report of one of the authors. [6] effectively studies the accuracy of only some of the targeted events of the MIPS R10000, i.e., issued instructions, loads, and stores, cycles, and decoded branches. For other events, i.e., cache misses, performance data was compromised because of perturbations introduced by multiprogramming and network services. As a result, the study discussed in this paper uses the same methodology but in a single-user, stand-alone environment based on an SGI O2 running IRIX 6.5.2 on a MIPS R12000 with a clock speed of 250 MHz. The compiler is gcc, version 2.8.1.

The methodology comprises five phases:

1. creation of microbenchmarks for studying event counter accuracy,
2. prediction of event performance data,
3. collection of event performance data from the MIPS R12000 performance counters via perfex and libperfex,
4. collection of event performance data via a MIPS R12000 simulator, and
5. comparison of event performance data from phases 2, 3, and 4.

The measured events and the microbenchmarks are presented and discussed in Sections 2.1 and 2.2, respectively. The MIPS R12000 performance counters and the interfaces used to access the counters are described in Section 2.3 and the MIPS R12000 simulator is discussed in Section 2.4.

2.1 Measured Events

The two criteria used to select the measured events of interest were 1) the events are reasonably predictable and 2) they are measurable by the MIPS R12000 simulator. To be reasonably predictable means that, for a given set of benchmarks, the event counts can be estimated easily. For example, given the code size and structure of a benchmark as well as the size and organization of the primary (L1) cache, the analyst can estimate the number of L1 cache misses that program will generate. To be measurable by the simulator means that the simulator can be configured to generate the targeted event count.

Table 1: Event Numbers

Event Type	Event Number
decoded instructions	1
decoded loads	2
decoded stores	3
conditional resolved branches	6
primary (L1) data cache (D-cache) misses	25
L1 instruction cache (I-cache) misses	9
secondary (L2) D-cache misses	26
L2 I-cache misses	10
Translation Lookaside Buffer (TLB) misses	23

According to these criteria, nine out of the available 30 events [8], defined in Table 1, were selected for study. Since the MIPS R12000 L2 cache is unified, events 10 and 26 are combined when evaluating the accuracy of the performance counters.

2.2 Microbenchmarks

The microbenchmarks used in this study were designed to facilitate relatively easy prediction of the event counts and to vary the number of event occurrences. Evidence provided in [6] indicates that the accuracy of the counters depends on the number of events generated by the measured program as compared to the number of events contributed by the counter interface. By varying the number of event occurrences, the influence of the interface on the event counts can be determined and, when possible, the counts can be adjusted accordingly.

2.2.1. Linear Microbenchmark

The Linear Microbenchmark is used to measure only the L1 I-cache miss event. It was designed initially to measure the number of decoded instruction, load, and store events, as well as the L1 I-cache miss event. However, as discussed in Section 3, the Loop Microbenchmark is more effective in measuring the decoded instruction, load, and store events.

```
a = 1;
b = 1;
c = 1;
a = b + 1;
b = a + 1;
c = a + b;
a = b + c;
b = a + c;
c = a + b;
```

Figure 1. Linear Microbenchmark Code Segment

As shown in Figure 1, the Linear Microbenchmark consists of a repeated sequence of simple add instructions that use three variables. In order to allow accurate L1 I-cache count prediction, the benchmark was designed to include no branches, avoiding the effects of speculative execution, and to force data dependences that ensure in-order instruction execution and prevent instruction elimination by the compiler.

Several versions of the benchmark were used to determine its effectiveness; the versions differ in terms of the number of times the instruction sequence is repeated.

2.2.2 Loop Microbenchmark

The Loop Microbenchmark is used to measure the number of decoded instruction, load, and store events, as well as the number of resolved conditional branch events. The benchmark consists of a for-loop with a body equivalent to the Linear Microbenchmark with 100 instructions. Several versions of the benchmark are used in the study; the versions differ in terms of the number of loop iterations. Predicting the number of decoded instructions, loads, and stores and resolved branches is elementary using the grep and wc (word count) Unix tools (e.g., grep lw testx.s | wc).

2.2.3 Array Microbenchmark

The Array Microbenchmark is used to measure the number of L1 D-cache, L2 cache, and TLB miss events. The other two microbenchmarks do not access enough data elements to generate a sufficient number of these events. This benchmark consists of a for-loop with a body that reads, increments, and writes elements of an integer array in sequence. Figure 2 presents the general format of the benchmark. Several versions of the benchmark were used in the study; the versions differ in terms of array size, which is passed as an input parameter.

```
#define MAXSIZE 1000000
int main (int argc, char *argv[]) {
int a[MAXSIZE], ARRAYSIZE, i;
  ARRAYSIZE = atoi(argv[1]);
  for (i=0; i<ARRAYSIZE;i++)
    a[i] = a[i] + 1;
  }
}
```

Figure 2. Array Microbenchmark

2.3 MIPS R12000 Performance Counters

The R12000 performance counters are two 32-bit registers that can count up to 30 unique events. The counters can be accessed via the IRIX operating system using two interfaces: perfex and libperfex [8]. perfex is the command-line interface used for counting events during the execution of an entire binary file,

while libperfex provides easy to access C and Fortran functions that are inserted in the program code to count events in specific code sections. The overall functionality of libperfex is a subset of that provided by perfex.

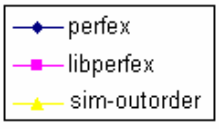
In the study presented in this paper, perfex and libperfex were used to collect data generated by the performance counters during execution of the microbenchmarks described in Section 2.2. For each benchmark type, i.e., Linear, Loop, and Array, approximately 25 versions were created and run. As mentioned above, the versions differ in terms of number of instructions, number of iterations, and number of array elements, respectively. Because event counts vary with each execution of a version of a microbenchmark, each version was run 100 times and the results averaged. The accuracy of event counts was evaluated by comparing the average event counts to the predicted event counts.

2.4 Sim-outorder

The simulator used in this study is sim-outorder, which is included in the SimpleScalar [5] tool set. sim-outorder allows complex modeling of the processor core, memory hierarchy, and branch prediction. It is capable of simulating superscalar processors such as the MIPS R12000. As shown in Section 3, the sim-outorder model of the MIPS R12000 is very effective in verifying our predicted event counts.

3.0 Study Results

Comparisons of the predicted and collected event counts for the three microbenchmarks are presented in Sections 3.1-3.5, Table 2, and Figures 3-10. Section 3.1, Table 2, and Figures 3-6 relate to the events for which the Loop Microbenchmark is most effective in demonstrating agreement between predicted counts and counts generated by the performance counters, i.e., decoded instruction, decoded load, decoded store, and conditional resolved branch counts. Section 3.2 and Figure 7 relate to the event for which the Linear Microbenchmark is used most effectively, i.e., the L1 I-cache miss counts. And, Sections 3.3-3.5 and Figures 8-10 relate to the events for which the Array Microbenchmark is most effective, i.e., L1 D-cache miss, L2 cache miss, and TLB miss counts. The legend for Figures 3-10 appears below.



3.1 Decoded Instructions, Loads, and Stores and Conditional Resolved Branches

For the Linear Microbenchmark, there are very large discrepancies between the libperfex and perfex decoded instruction, decoded load, decoded store, and conditional resolved branch event counts and the predicted counts. For example, the difference between libperfex decoded instruction event counts and the predicted counts is between 57% and 925% and the difference between perfex decoded instruction event counts and the predicted counts is even larger. For the

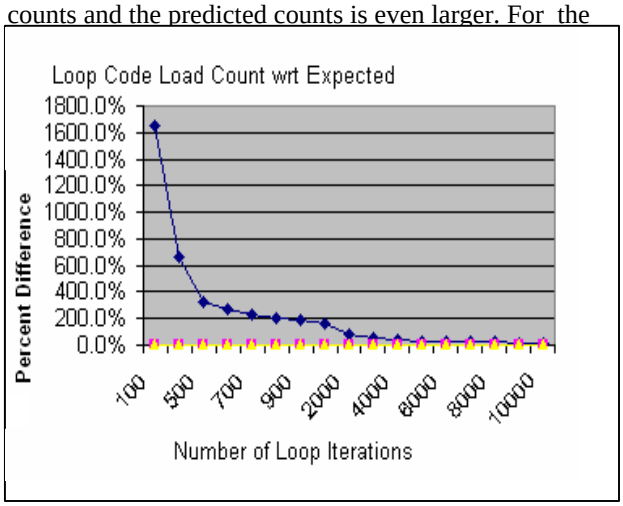
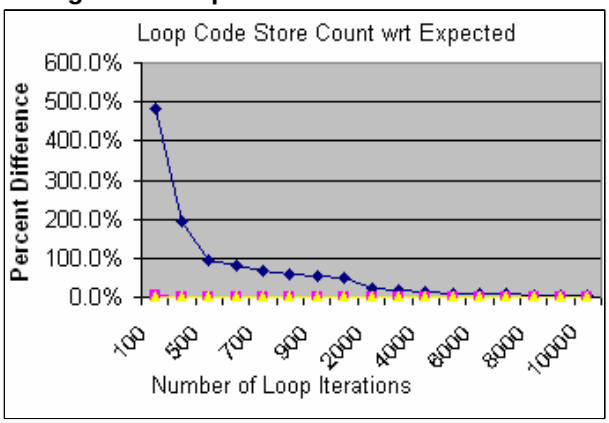


Figure 4. Loop Microbenchmark: Decoded Loads

Figure 5. Loop Microbenchmark: Decoded



Stores

decoded load event, at a code size of 100,000 instructions, the difference between predicted and libperfex (perfex) counts is 71.8% (75.9%). For decoded stores, the perfex error begins at 10,000% and decreases as the code size increases, achieving a difference of 34% when the code comprises 50,000 instructions. For conditional resolved branches, perfex counts are very inflated; the error, w.r.t. predicted counts, ranges from –43.3% to 66,339%.

In contrast, as depicted in Figures 3-6 and Table 2, the differences between the predicted counts and the decoded instruction, decoded load, decoded store, and conditional resolved branch counts for the Loop Microbenchmark as measured by perfex, libperfex, and sim-outorder decrease as the number of iterations executed increases. For the first three events, the counts generated by libperfex and sim-outorder agree very well with the predicted counts, while the counts generated by perfex do not come within 10% of the predicted counts until the number of loop iterations in the benchmark is fairly large. For the conditional resolved branch event, when the number of loop iterations is greater than 2,000, the difference between libperfex and predicted counts ranges from –42% to –50%; in these cases, libperfex has an event count of approximately ½ of the predicted counts.

Table 2 presents more detailed comparative data generated using the Loop Microbenchmark for the decoded instruction, decoded load, decoded store, and conditional resolved branch events. Using the decoded instruction event as an example, Table 2 should be read as follows:

1) The counts generated by libperfex are within 5% of the predicted counts when the number of loop iterations in the benchmark is at least 250, i.e., the value in column 1.

2) The counts generated by perfex are within 5% of the predicted counts when the number of loop iterations in the benchmark is at least 100,000, i.e., the value in column 2.

3) The counts generated by perfex are within 10% of the predicted counts when the number of loop iterations in the benchmark is at least 30,000, i.e., the value in column 3.

4) For all or almost all versions of the benchmark, sim-outorder event counts are within .3%, i.e., the value in column 4, of the predicted counts.

5) The error range for perfex is 9.7-1086% for 30,000 to 250 iterations (value in column 5).

6) The perfex overhead is approximately 830,000 (value in column 6) instructions for 10,000 loop iterations.

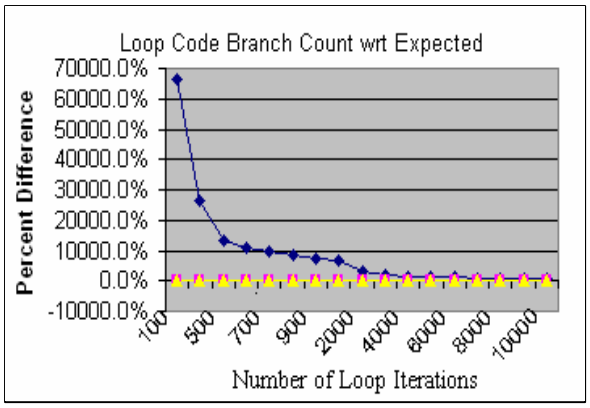


Figure 6. Loop Microbenchmark: Conditional Resolved Branches

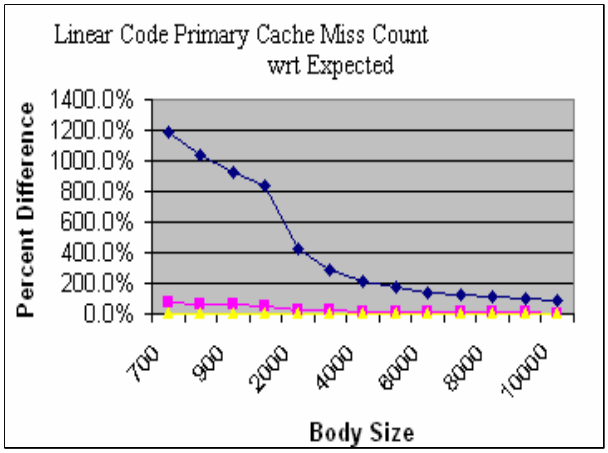


Figure 7. Linear Microbenchmark: L1 I-cache Misses

3.2 L1 I-cache Misses

The MIPS R12000 L1 I-cache is a 32-KB two-way set-associative cache with 16-word blocks. Since the instruction size is four bytes, the expected number of L1 I-cache misses is $\lceil n/16 \rceil$, where n is the number of instructions that comprises the Linear Microbenchmark. Figure 7 compares the L1 I-cache miss counts for the Linear Microbenchmark as measured by perfex, libperfex, and sim-outorder to the predicted counts. Like the counts for decoded instructions, loads, and stores, the counts for L1 I-cache misses generated by sim-outorder agree very well with the predicted counts, whereas the counts

generated by libperfex (perfex) do not come within 10% of the predicted counts until the number of instructions is fairly large, i.e., 6,000 (100,000). In all cases sim-outorder event counts are within less than 10% of the predicted counts. libperfex counts are within 600% of the predicted counts in all cases, while perfex counts have an error from 8,061% to 1.7% between 100 and 500,000 instructions. According to the results, perfex has an overhead of approximately 1,600 misses.

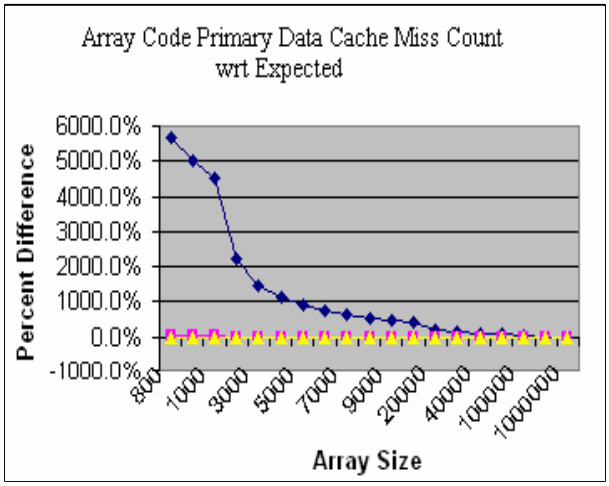


Figure 8. Array Microbenchmark: L1 D-Cache Misses

3.3 L1 D-cache Misses

The MIPS R12000 L1 D-cache is a 32-KB two-way set-associative cache with eight-word blocks. Since the Array Microbenchmark sequentially accesses four-byte words that are stored consecutively in the memory hierarchy, a cache miss is generated upon the access of element i*8 of the array, where i = 0, …, *n*-1 (where *n* is the size of the array). Thus, the expected number of L1 D-cache misses is ⌈ n/8 ⌉. Figure 8 compares the L1 D-cache miss counts for the Array Microbenchmark as measured by perfex, libperfex, and sim-outorder to the predicted counts. The sim-outorder counts are within 5% of the predicted counts for array sizes from 250 to 1,000,000. From array sizes 2,000 to 1,000,000 the error is less than 1%. libperfex counts are never within 10% of the predicted counts. After the array size reaches 3,000 the libperfex counts are within 25% of the predicted counts. Only when the array size becomes 100,000 or greater do the perfex counts come within 25% of the predicted counts.

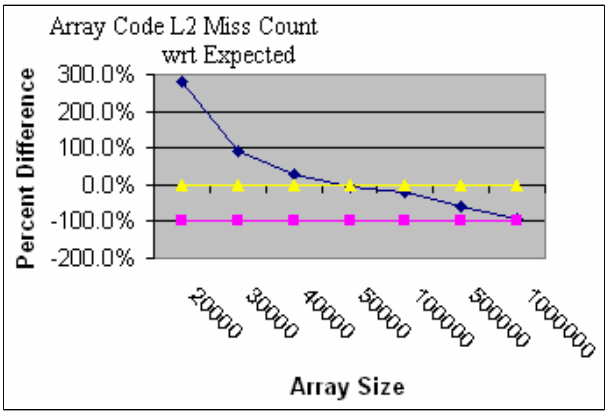


Figure 9. Array Microbenchmark: L2 Misses

3.4 L2 Cache Misses

The MIPS R12000 L2 cache is a 1-MB two-way set-associative unified cache with 32-word blocks. The L2 cache is large enough to store all instructions in the Array Microbenchmark. Every primary instruction (data) cache miss that misses the L2 cache causes two (four) L1 I- (D-) cache blocks to be brought into the L2 cache. Therefore, the expected number of L2 cache misses is equal to ⌈(L1 I-cache misses)/2 + (L1 D-cache misses)/4⌉, assuming that the instruction and data misses do not conflict. Figure 9 compares the L2 cache miss counts for the Array Microbenchmark as measured by perfex, libperfex, and sim-outorder to the predicted counts. The sim-outorder counts are within 5% of the predicted counts for array sizes from 800 to 7,000. From array sizes 100 to 700 the error ranges from 44% to 5.1%. libperfex generates a count that is within 10% of the predicted count only for the array size of 250. From array size 500 to array size 7,000 the difference ranges from –54% to 96%. Only when the array size is from 30,000 to 50,000 does the perfex counts come within 30% of the predicted counts.

3.5 TLB Misses

The R12000 translation-lookaside buffer, TLB, contains 64 entries, each of which maps a pair of virtual pages and can select a page size ranging from 4-KB to 16-MB, inclusive, in powers of 4 [8]. The *getpagesize*() function indicates that the page size for the experimental platform is 4-KB. Note that the man page for *getpagesize* states that the page size returned

by the function is a system page size, which may not be the same as the underlying hardware page size (frame). In addition, the man page states that in systems with multiple page sizes, like the R12000 with seven, the base page size is returned; the base page size is the smallest page size used by the system. Figure 10 compares the TLB miss counts for the Array Microbenchmark as measured by perfex, libperfex, and sim-outorder to the predicted counts. The sim-outorder counts are within 5% of the predicted counts for array sizes from 30,000 to 1,000,000. libperfex generates a count that is within 5% only for an array size of 100,000. With this exception, the error for libperfex ranges from 31% to 2712% for array sizes from 1,000,000 to 1,000. Only when the array size becomes 1,000,000 does the perfex count come within 6% of the predicted counts. Since predicted and sim-outorder event counts both are based on a fixed size page, it is not possible to determine the accuracy of the TLB miss event counts generated by the performance counters.

4.0 Conclusions and Future Research

As this paper demonstrates, performance counters can be very accurate or very inaccurate. As shown, their accuracy depends upon the interface used, the application being measured, and the event being measured. For the MIPS R12000 performance counters, this paper demonstrates that three of the eight studied events can be measured accurately with the libperfex interface and the Loop Microbenchmark: decoded instructions, loads, and stores. One event, i.e., the L1 I-cache miss event, can be measured accurately with the libperfex interface and the Linear Microbenchmark if the body size is at least 6,000 instructions. Conditional resolved branches cannot be measured accurately by either libperfex or perfex. The L1 D-cache and L2 cache miss events are not measured accurately by the R12000 performance counters via libperfex and the Array Microbenchmark. The accuracy of the counters via the perfex interface depends heavily on the granularity of the code that is being measured. The accuracy of perfex and libperfex counts for the TLB miss event cannot be determined since multiple page sizes are used and prediction and simulation counts are based on a fixed page size.

To ascertain how performance counters behave on large applications, for which we cannot predict event counts, in [6] the event counts of the R10000 performance counters were compared to the event counts generated by sim-outorder for Sweep3D, a neutron transport application used as a benchmark in the ASCI program [2]. For the decoded instruction event, libperfex counts were approximately 25% higher than sim-outorder counts—this error is higher than for the Loop Microbenchmark. For the decoded load (store) event, libperfex counts were approximately 33% (.5%) higher (lower) than sim-outorder counts—again this error is higher than for the Loop Microbenchmark. For the conditional resolved branch event, libperfex counts were approximately .8% lower than sim-outorder counts—a much smaller error than for the Loop Microbenchmark. For the L1 I-cache (D-cache) miss event, libperfex counts were approximately 100% (12%) higher than sim-outorder counts—much higher (lower) than for the Linear (Array) Microbenchmark. And, finally, for the L2 cache miss event, libperfex counts were approximately 40% higher than Sim-outorder counts—as error prone as the counts for the Array Microbenchmark.

As our results indicate, this is a topic that requires further investigation. Accordingly, our future research includes using an extended methodology to study and compare the accuracy of the performance counters of other microprocessor architectures. The extended methodology will include an enhanced set of microbenchmarks, event count comparisons such as the one described above using Sweep3D, and the use of the PAPI [14] interface, which some commonality among the varying architectures.

Acknowledgements

This work was supported by DARPA/ITO under contract N66001-97-C-8533, "End-to-End Performance Modeling of Large Heterogeneous Adaptive Parallel/Distributed Computer/Commmunication Systems," 10/97-12/00,

and by National Science Foundation under contract CDA-9522207, "CISE Minority Institutions Infrastructure: Building Affinity Groups to Enable and Encourage Student Success in Computing," 9/95-8/01, and contract EIA-9729990, "A Multiprocessor Platform for Cross-Disciplinary Research in Parallel Systems."

Table 2: Loop Benchmark Comparative Data

	# of loop iterations at which libperfex vs. predicted counts are w/in 5%	# of loop iterations at which perfex vs. predicted counts are w/in 5%	# of loop iterations at which perfex vs. predicted counts are w/in 10%	upper bound error Sim-outorder vs. predicted for majority of # of loop iterations	perfex vs. predicted error range for 30,000-250 loop iterations	approx. perfex overhead in # of instructions for 1,0000 loop iterations
Decoded Instructions	250	100,000	30,000	0.3%	9.7-1086%	830,000
Decoded Loads	250	40,000	20,000	0.1%	5.6-663%	230,000
Decoded Stores	250	10,000	5,000	0.1%	1.7-193.5%	69,000
Conditional Resolved Branches	NA	NA	NA	3.0%	172.4-26585.8%	219,000

References

[1] Adve, V., R. Bagrodia, J. Browne, E. Deelman, A. Dube, E. Houstis, J. Rice, R. Sakellariou, D. Sundaram-Stukel, P. Teller, and M. Vernon, "POEMS: End-to-end Performance Design of Parallel Adaptive Computational Systems," *Transactions on Software Engineering*, November 2000.

[2] The ASCI Sweep3D Benchmark: http://www.llnl.gov/asci_benchmarks/asci/limited/sweep3d/

[3] Browne, S., J. Dongarra, N. Garner, K. London, and P. Mucci, "A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters," *Proceedings of SC '00*, November 2000.

[4] Buck, B., and J. Hollingsworth, "Using Hardware Performance Monitors to Isolate Memory Bottlenecks" *Proceedings SC '00*, November 2000.

[5] Burger, D., and T. Austin, "The SimpleScalar Tool Set, Version 2.0," http://www.cs.wisc.edu/

[6] Castillo, G., "A Feasibility Study on the Use of the MIPS R10000 Processor Performance Counters," Master's Project Report, Department of Computer Science, The University of Texas at El Paso, January 4, 2000.

[7] DEC Alpha 21164 Hardware Reference Manual: http://ftp.digital.com/pub/Digital/DECinfo/semiconductor/literature/l64hrm.pdf

[8] MIPS R12000 Reference Manual: http://www.sgi.com/processors/r12k/manual.html

[9] Pentium III Intel Architecture Software Developer's Manual: http://www.intel.com/design/PentiumIII/manuals/

[10] PowerPC 604e RISC Microprocessor User's Manual: http://www.chips.ibm.com/products/powerpc/chips/six04.html

[10b] PowerPC 705 RISC Processor Technical Summary: http://www.chips.ibm.com:80/products/powerpc/chips/750_ts.pdf

[11] Silicon Graphics Inc., SpeedShop User's Guide, 1998.

[12] Sun Microsystem's Dual Processor System Controller (DSC) User's Manual: http://www.sun.com/microelectronics/manuals/802-7511.pdf

[13] Zagha, M., Larson, B., Turner, S., Itzkowitz, M., "Performance Analysis Using the MIPS R10000 Performance Counters", Proceedings of Supercomputing '96, November 1996.

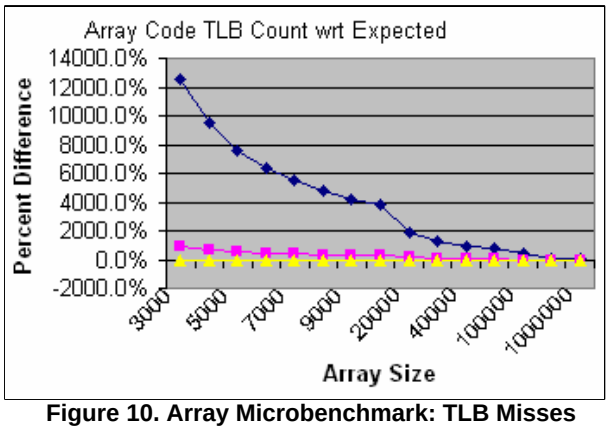


Figure 10. Array Microbenchmark: TLB Misses