Can Hardware Performance Counters be Trusted?

Vincent M. Weaver and Sally A. McKee
Computer Systems Laboratory
Cornell University
{vince,sam}@csl.cornell.edu

Abstract

When creating architectural tools, it is essential to know whether the generated results make sense. Comparing a tool's outputs against hardware performance counters on an actual machine is a common means of executing a quick sanity check. If the results do not match, this can indicate problems with the tool, unknown interactions with the benchmarks being investigated, or even unexpected behavior of the real hardware. To make future analyses of this type easier, we explore the behavior of the SPEC benchmarks with both dynamic binary instrumentation (DBI) tools and hardware counters.

We collect retired instruction performance counter data from the full SPEC CPU 2000 and 2006 benchmark suites on nine different implementations of the x86 architecture. When run with no special preparation, hardware counters have a coefficient of variation of up to 1.07%. After analyzing results in depth, we find that minor changes to the experimental setup reduce observed errors to less than 0.002% for all benchmarks. The fact that subtle changes in how *experiments are conducted can largely impact observed results is unexpected, and it is important that researchers using these counters be aware of the issues involved.*

1 Introduction

Hardware performance counters are often used to characterize workloads, yet counter accuracy studies have seldom been publicly reported, bringing such counter-generated characterizations into question. Results from counters are treated as accurate representations of events occurring in hardware, when, in reality, there are many caveats to the use of such counters.

When used in aggregate counting mode (as opposed to sampling mode), performance counters provide architectural statistics at full hardware speed with minimal overhead. All modern processors support some form of counters. Although originally implemented for debugging hardware designs during development, they have come to be used extensively for performance analysis and for validating tools and simulators. The types and numbers of events tracked and the methodologies for using these performance counters vary widely, not only across architectures, but also across systems sharing an ISA. For example, the Pentium III tracks 80 different events, measuring only two at a time, but the Pentium 4 tracks 48 different events, measuring up to 18 at a time. Chips manufactured by different companies have even more divergent counter architectures: for instance, AMD and Intel implementations have little in common, despite their supporting the same ISA. Verifying that measurements generate meaningful results across arrays of implementations is essential to using counters for research.

Comparison across diverse machines requires a common subset of equivalent counters. Many counters are unsuitable due to microarchitectural or timing differences. Furthermore, counters used for architectural comparisons must be available on all machines of interest. We choose a counter that meets these requirements: number of retired instructions. For a given statically linked binary, the retired instruction count *should* be the same on all machines implementing the same ISA, since the number of retired instructions excludes speculation and cache effects that complicate cross-machine correlation. This count is especially relevant, since it is a component of both the Cycles per Instruction (CPI) and (conversely) Instructions per Cycle (IPC) metrics commonly used to describe machine performance.

The CPI and IPC metrics are important in computer architecture research; in the rare occasion that a simulator is actually validated [19, 5, 7, 24] these metrics are usually the ones used for comparison. Retired instruction count and IPC are also used for vertical profiling [10] and trace alignment [16], which are methods of synchronizing data from various trace streams for analysis.

Retired instruction counts are also important when generating basic block vectors (BBVs) for use with the popular SimPoint [9] tool, which tries to guide statistically valid partial simulation of workloads that, if used properly, can greatly reduce experiment time without sacrificing accuracy in simulation results. When investigating the use of DBI tools to generate BBVs [26], we find that even a single extra instruction counted in a basic block (which represents the code executed in a SimPoint) can change which simulation points the SimPoint tool chooses to be most representative of whole program execution.

All these uses of retired instruction counters assume that generated results are repeatable, relatively deterministic, and have minimal variation across machines with the same ISA. Here we explore whether these assumptions hold by comparing the hardware-based counts from a variety of machines, as well as comparing to counts generated by Dynamic Binary Instrumentation (DBI) tools.

2 Related Work

Black et al. [4] use performance counters to investigate the total number of retired instructions and cycles on the PowerPC 604 platform. Unlike our work, they compare their results against a cycle-accurate simulator. The study uses a small number of benchmarks (including some from SPEC92), and the total number of instructions executed is many orders of magnitude fewer than in our work.

Patil et al. [18] validate SimPoint generation using CPI from Itanium performance counters. They compare different machines, but only the SimPoint-generated CPI values, not the raw performance counter results.

Sherwood et al. [20] compare results from performance counters on the Alpha architecture with SimpleScalar [2] and the Atom [21] DBI tool. They do not investigate changes in counts across more than one machine.

Korn, Teller, and Castillo [11] validate performance counters of the MIPS R12000 processor via microbenchmarks. They compare counter results to estimated (simulator-generated) results, but do not investigate the `instructions_graduated` metric (the MIPS equivalent of retired instructions). They report up to 25% error with the `instructions_decoded` counter on long-running benchmarks. This work is often cited as motivation for *why* performance counters should be used with caution.

Maxwell et al. [14] look at accuracy of performance counters on a variety of architectures, including a Pentium III system. They report less than 1% error on the retired instruction metric, but only for microbenchmarks and only on one system. Mathur and Cook [13] look at hand-instrumented versions of nine of the SPEC 2000 benchmarks on a Pentium III. They only report relative error of using sampled versus aggregate counts, and do not investigate overall error. DeRose et al. [6] look at variation and error with performance counters on a Power3 system, but only for startup and shutdown costs. They do not report total benchmark behavior.

3 Experimental Setup

We run experiments on multiple generations of x86 machines, listed in Table 1. All machines run the Linux 2.6.25.4 kernel patched to enable performance counter collection with the perfmon2 [8] infrastructure. We use the entire SPEC CPU 2000 [22] and 2006 [23] benchmark suites with the reference input sets. We compile the SPEC benchmarks on a SuSE Linux 10.1 system with version 4.1 of the gcc compiler and `-O2` optimization (except for `vortex`, which crashes when compiled with optimization). All benchmarks are statically linked to avoid variations due to the C library. We use the same 32-bit, statically linked binaries for all experiments on all machines.

We gather Pin [12] results using a simple instruction count utility via Pin version pin-2.0-10520-gcc.4.0.0-ia32-linux. We patch Valgrind [17] 3.3.0 and Qemu [3] 0.9.1 to generate retired instruction counts. We gather the DBI results on a cluster of Pentium D machines identical to that described in Figure 1. We configure `pfmon` [8] to gather complete aggregate retired instruction counts, without any sampling. The tool runs as a separate process, enabling counting in the OS; it requires no changes to the application of interest and induces minimal overhead during execution. We count user-level instructions specific to the benchmark.

We collect at least seven data points for every benchmark/input combination on each machine and with each DBI method (the one exception is the Core2 machine, which has hardware problems that limit us to three data points for some configurations). The SPEC 2006 benchmarks require at least 1GB of RAM to finish in a reasonable amount of time. Given this, we do not run them on the Pentium Pro or Pentium II, and we do not run `bwaves`, `GemsFDTD`, `mcf`, or `zeusmp` on machines with small memories. Furthermore, we omit results for `zeusmp` with DBI tools, since they cannot handle the large 1GB data segment the application requires.

4 Sources of Variation

We focus on two types of variation when gathering performance counter results. One is inter-machine variations, the differences between counts on two different systems. The other is intra-machine variations, those found when running the same benchmark multiple times on the same system. We investigate methods for reducing both types.

4.1 The `fldcw` instruction

For instruction counts to match on two machines, the instructions involved must be counted the same way. If not, this can cause large divergences in total counts. On Pentium 4 systems, the `instr_retired:nbogusntag` per-

Processor	Speed	Bits	Memory	L1 I/D Cache	L2 Cache	Retired Instruction Counter / Cycles Counter
Pentium Pro	200MHz	32	256MB	8KB/8KB	512KB	inst_retired cpu_clk_unhalted
Pentium II	400MHz	32	256MB	16KB/16KB	512KB	inst_retired cpu_clk_unhalted
Pentium III	550MHz	32	512MB	16KB/16KB	512KB	inst_retired cpu_clk_unhalted
Pentium 4	2.8GHz	32	2GB	12Kμ/16KB	512KB	instr_retired:nbogusntag global_power_events:running
Pentium D	3.46GHz	64	4GB	12Kμ/16KB	2MB	instr_completed:nbogus global_power_events:running
Athlon XP	1.733GHz	32	768MB	64KB/64KB	256KB	retired_instructions cpu_clk_unhalted
AMD Phenom	2.2GHz	64	2GB	64KB/64KB	512KB	retired_instructions cpu_clk_unhalted
Core Duo	1.66GHz	32	1GB	32KB/32KB	1MB	instructions_retired unhalted_core_cycles
Core2 Q6600	2.4GHz	64	2GB	32KB/32KB	4MB	instructions_retired unhalted_core_cycles

Table 1. Machines used for this study.

benchmark	fldcw instructions	% overcount
482.sphinx3	23,816,121,371	0.84%
177.mesa	6,894,849,997	2.44%
481.wrf	1,504,371,988	0.04%
453.povray	1,396,659,575	0.12%
456.hmmer retro	561,271,823	0.03%
175.vpr place	405,499,739	0.37%
300.twolf	379,247,681	0.12%
483.xalancbmk	358,907,611	0.03%
416.gamess cytosine	255,142,184	0.02%
435.gromacs	230,286,959	0.01%
252.eon kajiya	159,579,683	0.15%
252.eon cook	107,592,203	0.13%

Table 2. Dynamic count of `fldcw` instructions, showing all benchmarks with over 100 million. This instruction is counted as two instructions on Pentium 4 machines but only as one instruction on all other implementations.

formance counter counts `fldcw` as two retired instructions; on all other x86 implementations `fldcw` counts as one. This instruction is common in floating point code: it is used in converting between floating point and integer values. It alone accounts for a significant divergence in the `mesa` and `sphinx3` benchmarks. Table 2 demonstrates occurrences in the SPEC benchmarks where the count is over 100 million. We modify Valgrind to count the `fldcw` instructions, and use these counts to adjust results when presenting Pentium 4 data. It should be possible to use statistical methods to automatically determine which type of opcode causes divergence in cases like this; this is part of ongoing work. We isolated the `fldcw` problem by using a tedious binary search of the `mesa` source code.

4.2 Using the Proper Counter

Pentium 4 systems after the model 6 support a `instr_completed:nbogus` counter, which is more accurate than the `instr_retired:nbogusntag` counter found on previous models. This newer counter does not suffer the `fldcw` problem described in Section 4.1. Unfortunately, all systems do not include this counter; our Pentium D can use it, but our older Pentium 4 systems cannot. This counter is not well documented, and thus it was not originally available within the `perfmon` infrastructure. We contributed counter support that has been merged into the main `perfmon` source tree.

4.2.1 Virtual Memory Layout

It may seem counterintuitive, but some benchmarks behave differently depending on where in memory their data structures reside. This causes much of the intra-machine variation we see across the benchmark suites. In theory, memory layout should not affect instruction count. In practice, both `parser` and `perlbench` exhibit this problem. To understand how this can happen, it is important to understand the layout of virtual memory on x86 Linux. In general, program code resides near the bottom of memory, with initialized and uninitialized data immediately above. Above these is the heap, which grows upward. Near the top of virtual memory is the stack, which grows downward. Above that are command line arguments and environment variables.

Typical programs are insensitive to virtual address assignments for data structures. Languages that allow pointers to data structures make the virtual address space "visible". Different pointer values only affect instruction counts if programs act on those values. Both `parser` and `perlbench` use pointers as hash table keys. Differing table layouts can cause hash lookups to use different numbers of instructions, causing noticeable changes in retired instruction counts.

There are multiple reasons why memory layout can vary from machine to machine. On Linux the environment variables are placed above the stack; a differing number of environment variables can change the addresses of local variables on the stack. If the addresses of these local variables are used as hash keys then the size and number of environment variables can affect the total instruction count. This happens with `perlbench`; Mytkowicz et al. [15] document the effect, finding that it causes execution time differences of up to 5%.

A machine's word size can have unexpected effects on virtual memory layout. Systems running in 64-bit mode can run 32-bit executables in a compatibility mode. By default, however, the stack is placed at a higher address to free extra virtual memory space. This can cause inter-machine variations, as local variables have different addresses on a 64-bit machine (even when running a 32-bit binary) than on a true 32-bit machine. Running the Linux command `linux32 -3` before executing a 32-bit program forces the stack to be in the same place it would be on a 32-bit machine.

Another cause of varied layout is due to virtual memory randomization. For security reasons, recent Linux kernels randomize the start of the text, data, bss, stack, heap, and mmap() regions. This feature makes buffer-overrun attacks more difficult, but the result is that programs have different memory address layouts each time they are run. This causes programs (like `parser`) that use heap-allocated addresses as hash keys to have different instruction counts every time. This behavior is disabled system wide by the command:

```
echo 0 >
/proc/sys/kernel/randomize_va_space
```

It is disabled at a per-process level with the `-R` option to the `linux32` command. For our final runs, we use the `linux32 -3 -R` command to ensure consistent virtual memory layout, and we use a shell script to force environment variables to be exactly 422 bytes on all systems.

4.3 Processor Errata

There are built-in limitations to performance counter accuracy. Some are intended, and some are unintentional byproducts of the processor design. Our results for our 32-bit Athlon exhibit some unexplained divergences, leading us to investigate existing errata for this processor [1]. The errata mention various counter limitations that can result in incorrect total instruction counts. Researchers must use caution when gathering counts on such machines.

4.3.1 System Effects

Any Operating System or C library call that returns nondeterministic values can potentially lead to divergences. This includes calls to random number generators; anything involving the time, process ID, or thread synchronizations; and any I/O that might involve errors or partial returns. In general, the SPEC benchmarks carefully avoid most such causes of non-determinism; this would not be the case for many real world applications.

OS activity can further perturb counts. For example, we find that performance counters for all but the Pentium 4 increase once for every page fault caused by a process. This can cause instruction counts to be several thousands higher, depending on the application's memory footprint. Another source of higher instruction counts is related to the number of timer interrupts incurred when a program executes; this is possibly proportional to the number of context switches. The timer based perturbation is most noticeable on slower machines, where longer benchmark run times allow more interrupts to occur. Again, the Pentium 4 counter is not affected by this, but all of the other processors are. In our final results, we account for perturbations due to timer interrupt but not for those related to page faults. There are potentially other OS-related effects which have not yet been discovered.

4.4 Variation from DBI Tools

In addition to actual performance counter results, computer architects use various tools to generate retired instruction counts. Dynamic Binary Instrumentation (DBI) is a fast way to analyze benchmarks, and it is important to know how closely tool results match actual hardware counts.

4.4.1 The `rep` Prefix

An issue with the Qemu and Valgrind tools involves the x86 `rep` prefix. The `rep` prefix can come before string instructions, causing the the string instruction to repeat while decrementing the `ecx` register until it reaches zero. A naive implementation of this prefix counts each repetition as a committed instruction, and Valgrind and Qemu do this by default. This can cause many excess retired instructions to be counted, as shown in Table 3. The count can be up to 443 billion too high for the SPEC benchmarks. We modify the DBI tools to count only the `rep` prefixed instruction as a single instruction, as per the relevant hardware manuals.

4.4.2 Floating Point Rounding

Dynamic Binary Instrumentation tools can make floating point problematic, especially for x86 architectures. Default x86 floating point mode is 80-bit FP math, not commonly found in other architectures. When translating x86 instructions, Valgrind uses 64-bit FP instructions for portability. In theory, this should cause no problems with well written programs, but, in practice, it occasionally does. The move to SSE-type FP implementations on newer machines decreases the problem's impact, although new instructions may also be sources of variation.

The `art` benchmark. The `art` benchmark uses many fewer instructions on Valgrind than on real hardware. This is due to the use of the "==" C operator to compare floating point numbers. Rounding errors between 80-bit and 64-bit versions of the code cause the 64-bit versions to finish with significantly different instruction counts (while still generating the proper reference output). This is because a loop waiting for a value being divided to fall below a certain limit can happen faster when the lowest bits are being truncated. The proper fix is to update the DBI tools to handle 80-bit floating point properly. A few temporary workarounds can be used: passing a compiler option to use only 64-bit floating point, having the compiler generate SSE rather than x87 floating point instructions, or adding an instruction to the offending source code to force the FPU into 64-bit mode.

The `dealII` benchmark. The `dealII` SPEC CPU 2006 benchmark is problematic for Valgrind, much like `art`. In this case, the issue is more critical: the program enters an infinite loop. It waits for a floating point value to reach an epsilon value smaller than can be represented with 64-bit floating point. The authors of `dealII` are aware of this possibility, since source code already has a `#define` to handle this issue on non-x86 architectures.

benchmark	rep counts	% overcount
464.h264ref sss_main	443,109,753,850	15.7%
464.h264ref fore_main	45,947,752,893	14.2%
482.sphinx3	33,734,602,541	1.2%
403.gcc s04	33,691,268,130	18.8%
403.gcc c-typeck	30,532,770,775	21.7%
403.gcc expr2	26,145,709,200	16.3%
403.gcc g23	23,490,076,359	12.1%
403.gcc expr	18,526,142,466	15.7%
483.xalancbmk	15,102,464,207	1.2%
403.gcc cp-decl	14,936,880,311	13.6%
450.soplex pds-50	11,760,258,188	2.5%
453.povray	10,303,766,848	0.9%
403.gcc 200	10,260,100,762	6.1%

Table 3. Potential excesses in dynamic counted instructions due to the `rep` prefix (only benchmarks with more than 10 billion are shown).

4.4.3 Virtual Memory Layout

When instrumenting a binary, DBI tools need room for their own code. The tools try to keep layout as close as possible to what a normal process would see, but this is not always possible, and some data structures are moved to avoid conflicts with memory needed by the tool. This leads to perturbations in the instruction counts similar to those exhibited in Section 4.2.1.

5 Summary of Findings

Figure 1 shows the coefficient of variation for SPEC CPU 2000 benchmarks before and after our adjustments. Large variations in `mesa`, `perlbmk`, `vpr`, `twolf`, and `eon` are due to the Pentium 4 `fldcw` problem described in Section 4.1. Once adjustments are applied, variation drops below 0.0006% in all cases. Figure 2 shows similar results for SPEC CPU 2006 benchmarks. Larger variations for `sphinx3` and `povray` are again due to the `fldcw` instruction. Once adjustments are made, variations drop below 0.002%. Overall, the CPU 2006 variations are much lower than for CPU 2000; the higher absolute differences are counterbalanced by the much larger numbers of total retired instructions. These results can be misleading: a billion-instruction difference appears small in percentage terms when part of a three trillion instruction program, but in absolute terms it is large. When attempting to capture phase behavior accurately using SimPoint with an interval size of 100 million instructions, a phase's being offset by one billion instructions can alter final results.

5.1 Intra-machine results

Figure 3 shows the standard deviations of results across the CPU 2000 and CPU 2006 benchmarks for each machine and DBI method. DBI results are shown, but not incorporated into standard deviations. In all but one case the standard deviation improves, often by at least an order of magnitude. For CPU 2000 benchmarks, `perlbmk` has large variation for every generation method. We are still investigating the cause. In addition, the Pin DBI tool has a large outlier with the `parser` benchmark, most likely due to issues with consistent heap locations. Improvements for CPU 2006 benchmarks are less dramatic, with large standard deviations due to high outlying results. On AMD machines, `perlbench` has larger variation than on other machines, for unknown reasons. The `povray` benchmark is an outlier on all machines (and on the DBI tools); this requires further investigation. The Valgrind DBI tool actually has worse standard deviations after our methods are applied due to a large increase in variation with the `perlbench` benchmarks. For the CPU 2006 benchmarks, similar platforms

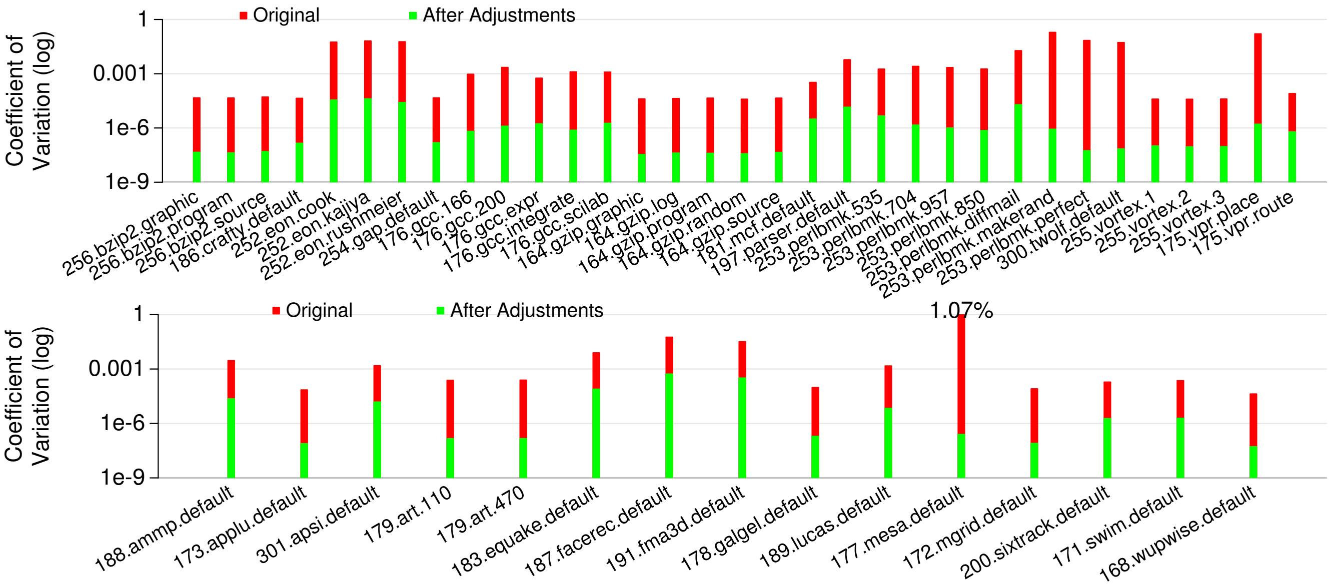

Figure 1. SPEC 2000 Coefficient of variation. The top graph shows integer benchmarks, the bottom, floating point. The error variation from `mesa`, `perlbmk`, `vpr`, `twolf` and `eon` are primarily due to the `fldcw` miscount on the Pentium 4 systems. Variation after our adjustments becomes negligible.

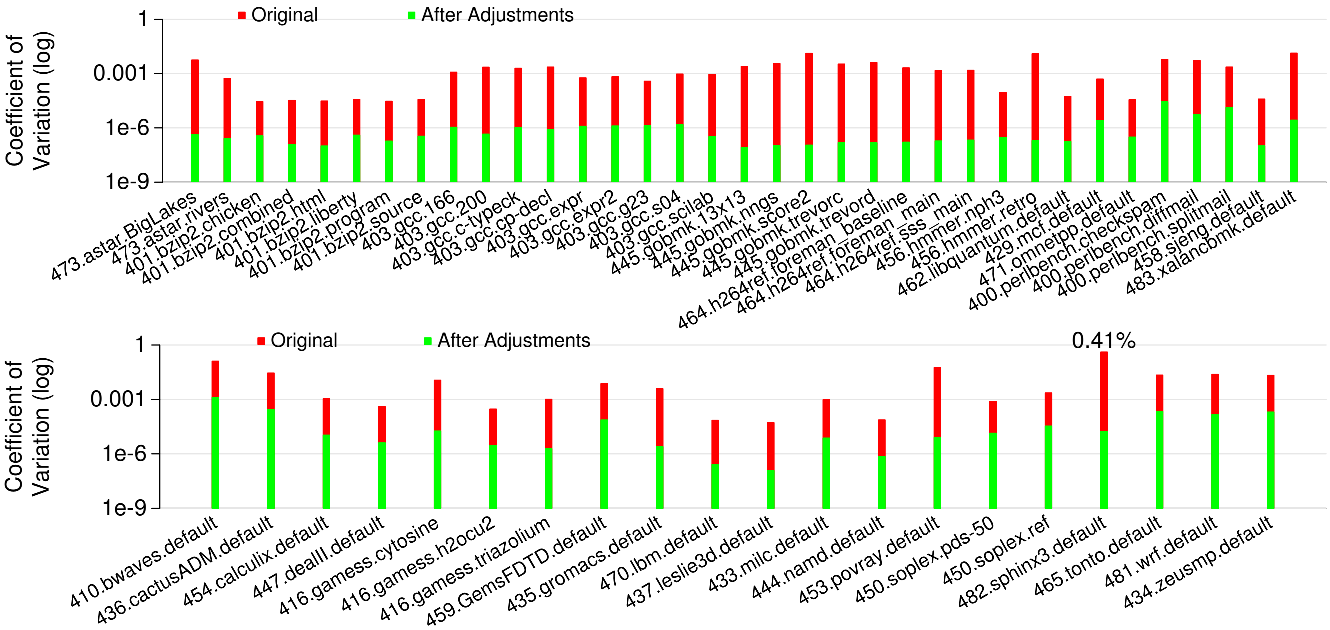

Figure 2. SPEC 2006 Coefficient of variation. The top graph shows integer benchmarks, bottom, floating point. The original variation is small compared to the large numbers of instructions in these benchmarks. The largest variation is in `sphinx3`, due to `fldcw` instruction issues. Variation after our adjustments becomes orders of magnitude smaller.

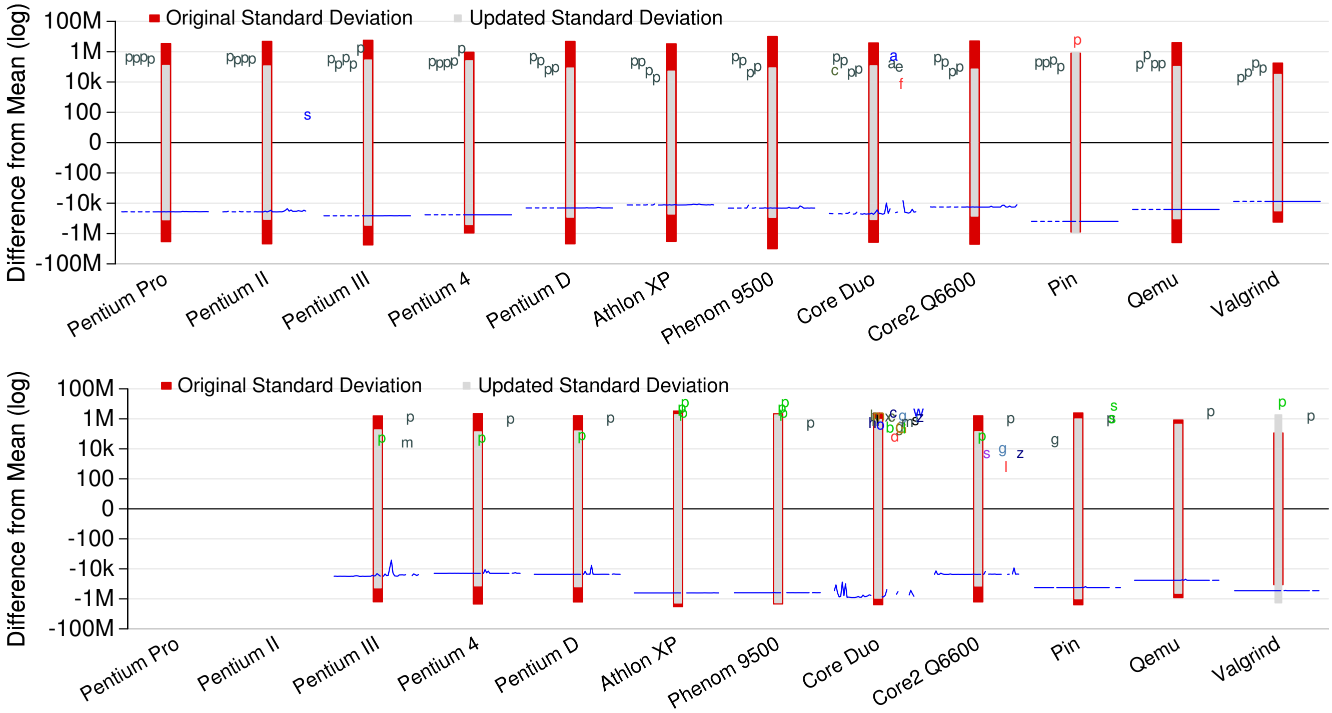

Figure 3. Intra-machine results for SPEC CPU 2000 (above) and CPU 2006 (below). Outliers are indicated by the first letter of the benchmark name and a distinctive color. For CPU 2000, the `perlbmk` benchmarks (represented by grey 'p's) are a large source of variation. For CPU 2006, the `perlbench` (green 'p') and `povray` (grey 'p') are the common outliers. Order of plotted letters for outliers has no intrinsic meaning, but tries to make the graphs as readable as possible. Horizontal lines summarize results for remaining benchmarks (they're all similar). The message here is that most platforms have few outliers, and there's much consistency with respect to measurements across benchmarks; Core Duo and Core2 Q6600 have many more outliers, especially for SPEC 2006. Our technical report provides detailed performance information — these plots are merely intended to indicate trends. Standard deviations decrease drastically with our updated methods, but there is still room for improvement.

have similar outliers: the two AMD machines share outliers, as do the two Pentium 4 machines.

5.2 Inter-machine Results

Figure 4 shows results for each SPEC 2000 benchmark (DBI values are shown but not incorporated into standard deviation results). We include detailed plots for five representative benchmarks to show individual machine contributions to deviations. (Detailed plots for all benchmarks are available in our technical report [25].) Our variation-reduction methods help integer benchmarks more than floating point. The Pentium III, Core Duo and Core 2 machines often over-count instructions. Since they share the same base design, this is probably due to architectural reasons. The Athlon frequently is an outlier, often under-counting. DBI results closely match the Pentium 4's, likely because the Pentium 4 counter apparently ignores many OS effects that other machines cannot.

Figure 5 shows inter-machine results for each SPEC 2006 benchmark. These results have much higher variation than the SPEC 2000 results. Machines with the smallest memories (Pentium 3, Athlon, and Core Duo) behave similarly, possibly due to excessive OS paging activity. The Valgrind DBI tool behaves poorly compared to the others, often overcounting by at least a million instructions.

6 Conclusions and Future Work

Even though originally included in processor architectures for hardware debugging purposes, when used correctly, performance counters can be used productively for

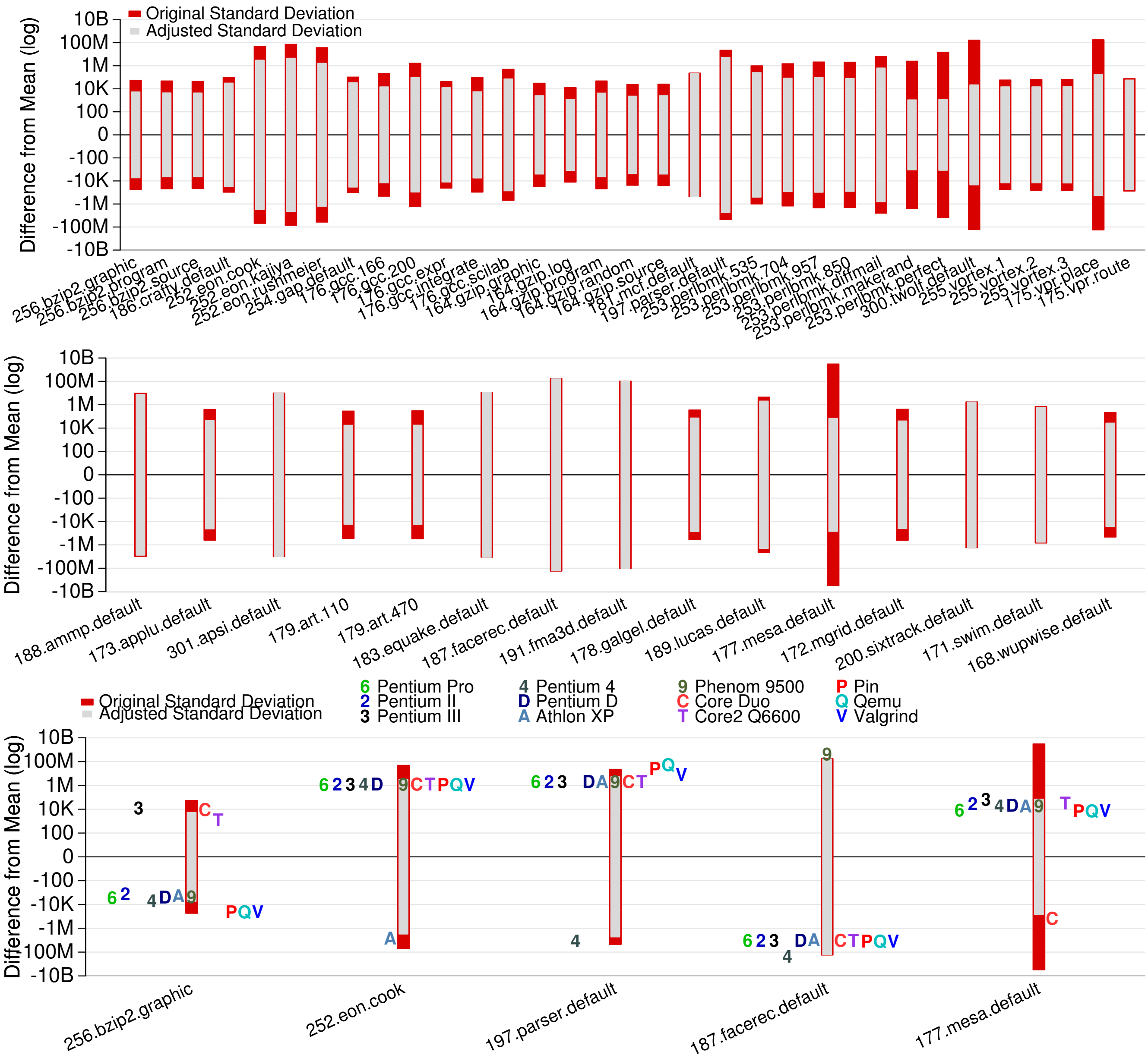

Figure 4. Inter-machine results for SPEC CPU 2000. We choose five representative benchmarks and show the individual machine differences contributing to the standard deviations. Often there is a single outlier affecting results; the outlying machine is often different. DBI results are shown, but not incorporated into standard deviations.

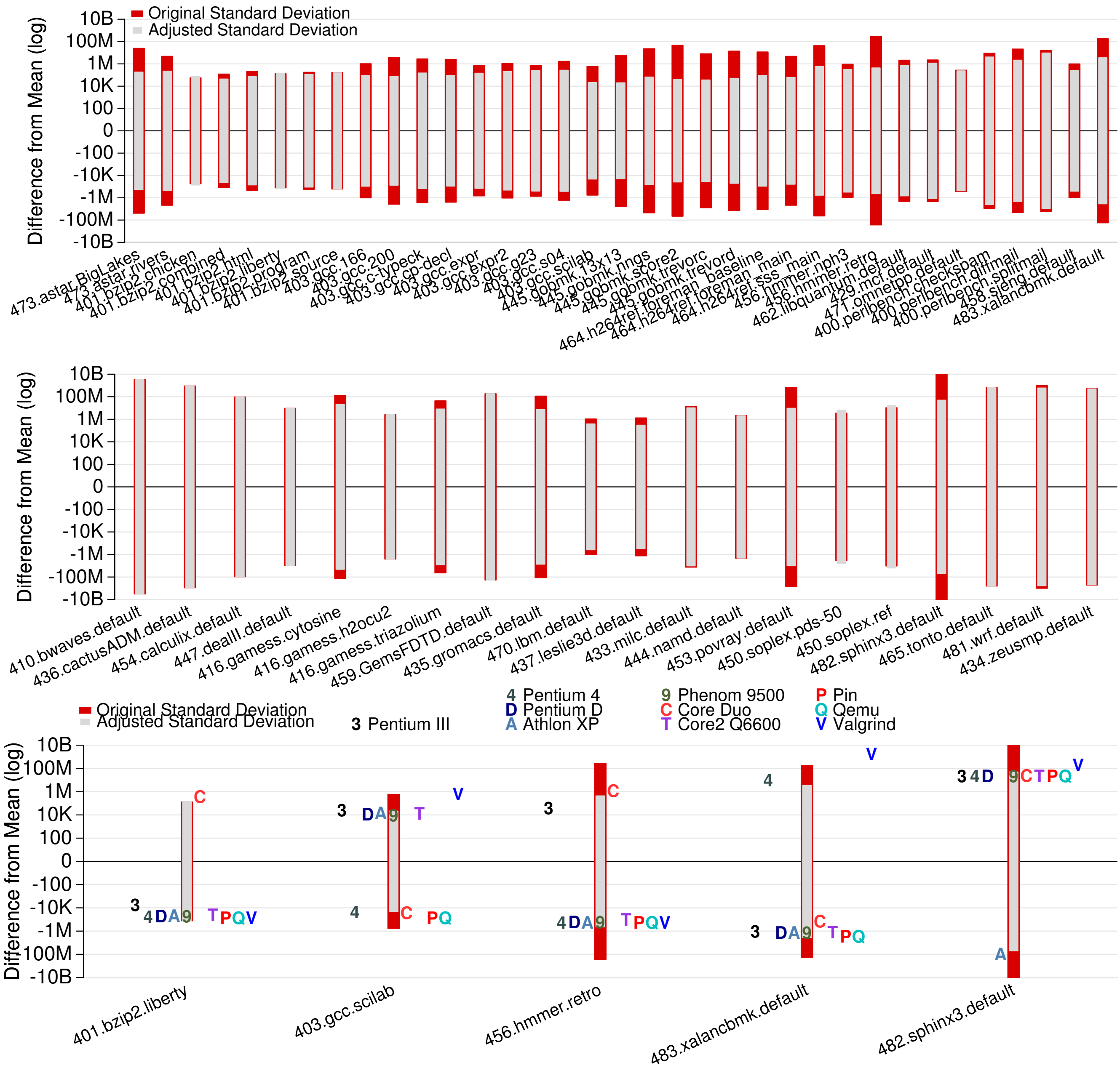

Figure 5. Inter-machine results for SPEC CPU 2006. We choose five representative benchmarks and show the individual machine differences contributing to the standard deviations. Often there is a single outlier affecting results; the outlying machine is often different. DBI results are shown, but not incorporated into the standard deviations.

many types of research (as well as application performance debugging). We have shown that with some simple methodology changes, the x86 retired instruction performance counters can be made to have a coefficient of variation of less than 0.002%. This means that architecture research using this particular counter can reasonably be expected to reflect actual hardware behavior. We also show that our results are consistent across multiple generations of processors. This indicates that older publications using these counts can be compared to more recent work.

Due to time constraints, several unexplained variations in the data still need to be explored in more detail. We have studied many of the larger outliers, but several smaller, yet significant, variations await explanation. Here we examine only SPEC; other workloads, especially those with significant I/O, will potentially have different behaviors. We also only look at the retired instruction counter; processors have many other useful counters, all with their own sets of variations. Our work is a starting point for single-core performance counter analysis. Much future work remains involving modern multi-core workloads.

Acknowledgments

We thank Brad Chen and Kenneth Hoste for their invaluable help in shaping this article. This work is supported in part by NSF CCF Award 0702616 and NSF ST-HEC Award 0444413.

References

[1] Advanced Micro Devices. *AMD Athlon Processor Model 6 Revision Guide*, 2003.

[2] T. Austin. Simplescalar 4.0 release note. http://www.simplescalar.com/.

[3] F. Bellard. QEMU, a fast and portable dynamic translator. In *Proc. 2005 USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, Apr. 2005.

[4] B. Black, A. Huang, M. Lipasti, and J. Shen. Can trace-driven simulators accurately predict superscalar performance? In *Proc. IEEE International Conference on Computer Design*, pages 478–485, Oct. 1996.

[5] G. Contreras, M. Martonosi, J. Peng, R. Ju, and G. Lueh. XTREM: A power simulator for the intel XScale core. In *Proc. ACM Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 115–125, 2004.

[6] L. DeRose. The hardware performance monitor toolkit. In *Proc. 7th International Euro-Par Conference*, pages 122–132, Aug. 2001.

[7] R. Desikan, D. Burger, and S. Keckler. Measuring experimental error in multiprocessor simulation. In *Proc. 28th IEEE/ACM International Symposium on Computer Architecture*, pages 266–277, June 2001.

[8] S. Eranian. Perfmon2: a flexible performance monitoring interface for Linux. In *Proc. 2006 Ottawa Linux Symposium*, pages 269–288, July 2006.

[9] G. Hamerly, E. Perelman, J. Lau, and B. Calder. Simpoint 3.0: Faster and more flexible program analysis. In *Workshop on Modeling, Benchmarking and Simulation*, June 2005.

[10] M. Hauswirth, A. Diwan, P. F. Sweeney, and M. C. Mozer. Automating vertical profiling. In *Proc. 20th ACM Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 281–296, 2005.

[11] W. Korn, P. J. Teller, and G. Castillo. Just how accurate are performance counters? In *20th IEEE International Performance, Computing, and Communication Conference*, pages 303–310, Apr. 2001.

[12] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. Reddi, and K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 190–200, June 2005.

[13] W. Mathur and J. Cook. Improved estimation for software multiplexing of performance counting. In *Proc. 13th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 23–34, Sept. 2005.

[14] M. Maxwell, P. Teller, L.Salayandia, and S. Moore. Accuracy of performance monitoring hardware. In *Proc. Los Alamos Computer Science Institute Symposium*, Oct. 2002.

[15] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. Sweeney. We have it easy, but do we have it right? In *NSF Next Generation Systems Workshop*, pages 1–5, Apr. 2008.

[16] T. Mytkowicz, P. F. Sweeney, M. Hauswirth, and A. Diwan. Time interpolation: So many metrics, so few registers. In *Proc. IEEE/ACM 41st Annual International Symposium on Microarchitecture*, 2007.

[17] N. Nethercote and J. Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 89–100, June 2007.

[18] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi. Pinpointing representative portions of large Intel Itanium programs with dynamic instrumentation. In *Proc. IEEE/ACM 37th Annual International Symposium on Microarchitecture*, pages 81–93, Dec. 2004.

[19] D. Penry, D. August, and M. Vachharajani. Rapid development of a flexible validated processor model. In *Proc. Workshop on Modeling, Benchmarking, and Simulation*, pages 21–30, June 2005.

[20] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proc. 10th ACM Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, Oct. 2002.

[21] A. Srivastava and A. Eustace. ATOM: a system for building customized program analysis tools. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 196–205, June 1994.

[22] Standard Performance Evaluation Corporation. SPEC CPU benchmark suite. `http://www.specbench.org/osg/cpu2000/`, 2000.

[23] Standard Performance Evaluation Corporation. SPEC CPU benchmark suite. `http://www.specbench.org/osg/cpu2006/`, 2006.

[24] V. Weaver and S. McKee. Are cycle accurate simulations a waste of time? In *Proc. 7th Workshop on Duplicating, Deconstructing, and Debunking*, June 2008.

[25] V. Weaver and S. McKee. Can hardware performance counters be trusted? Technical Report CSL-TR-2008-1051, Cornell University, Aug. 2008.

[26] V. Weaver and S. McKee. Using dynamic binary instrumentation to generate multi-platform simpoints: Methodology and accuracy. In *Proc. 3rd International Conference on High Performance Embedded Architectures and Compilers*, pages 305–319, Jan. 2008.

A Extended Results

This Appendix includes expanded results that could not be included with the original paper.

A.1 Miscounts due to Virtual Memory

In Section 4.2 we discuss various ways that changes in virtual memory addresses can affect the amount of retired instructions. We have found at least one additional cause of variation, which is optimized memory copy routines.

Many processors offer means of copying large blocks of memory at once, which is faster than doing individual word-sized loads and stores. Often these block memory copies are done using the SIMD or floating point units. These copies often have strict memory alignment rules, often of relatively large power-of-two (64 or 128) byte alignments. These alignment rules are stricter than the stack alignment rules which are often only 8 or 16 byte aligned. Thus when copying memory on the stack, the stack offset can affect how many instructions are retired, especially if extra code is needed at the beginning or end to take care of values that are not properly aligned.

A.2 Algorithmic Variations

Some of the SPEC benchmarks have code paths that cause variation in the retired instruction count, leading the results to be non-deterministic. We attempt to determine the causes of these variations in order to compensate for them.

A.2.1 perlbench

The SPEC CPU 2006 benchmark `perlbench` uses the address of a local variable as a key into a hash table, introducing dependencies on stack addresses (which cause dependencies on stack alignment and environmental variables, as described in Section 4.2.1).

This occurs in the code in the function `Perl_gv_fetchpv()` in the file `gv.c`:

```
char *tmpbuf;
...
gvp=(GV**)hv_fetch(stash,tmpbuf,len,add);
```

The variable `tmpbuf` is local, so is allocated on the stack, and it is passed as a key to the `hv_fetch` hash function.

A.2.2 parser

The SPEC CPU 2000 benchmark `parser` uses the address of a heap address as a key into a hash table. This can cause variation between runs if heap randomization is turned on, as described in Section 4.2.1.

This occurs in `parse.c` where the function `hash()` has the following code:

```
int hash(int lw, int rw, Connector *le,
         Connector *re, int cost) {
...
i = i + (i<<1) + randtable[
    (((long) le + i) %
    (table_size+1)) &
    (RTSIZE - % 1)];
```

The variable `le` is on the heap, and the pointer to it is cast to a long and used as a hash table index.

A.2.3 Others

There are variations in other benchmarks that need further investigation: `povray`, `gcc`, and `perlbmk`. The `gcc` based variation is eliminated by the methods described in this paper, but `povray` and `perlbmk` need further analysis.

A.3 Interrupt Related Overcounts

We investigate how interrupts affect the retired instruction counts on various machines. We are still determining the root cause of this source of variation: is it inherent in the counters, an artifact of the perfmon2 interface, or caused by the operating system itself? The fact that the Pentium 4 is immune indicates it might be a hardware issue.

Possibly all interrupts, both software and hardware, cause this variation. It is difficult to obtain per-process interrupt statistics under Linux. On most x86 systems the timer interrupt generates an order of magnitude more interrupts than any other sources, so we use it as a base for evaluating interrupt-caused variation. Current Linux developments, such as dynamic frequency scaling and tickless timers (no periodic clock interrupt) potentially affect this analysis.

Figure 6 shows the results of our investigation. In Linux, the timer interrupt is programmed to happen at an interval known as HZ, which is typically 100, 250, or 1000. We ran the SPEC CPU 2000 benchmarks on machines configured with those values. We then created a baseline using the 100Hz results, and attempted to estimate the Hz value for the others solely using the excess retired instruction counts. For all of the machines *except* the Pentium 4 the instruction overhead closely follows the HZ value, indicating that this should be accounted for when determining retired instruction count.

A.4 Cycles Performance Counter

In addition to retired instructions, each processor investigated also has a total cycles performance counter. We un-

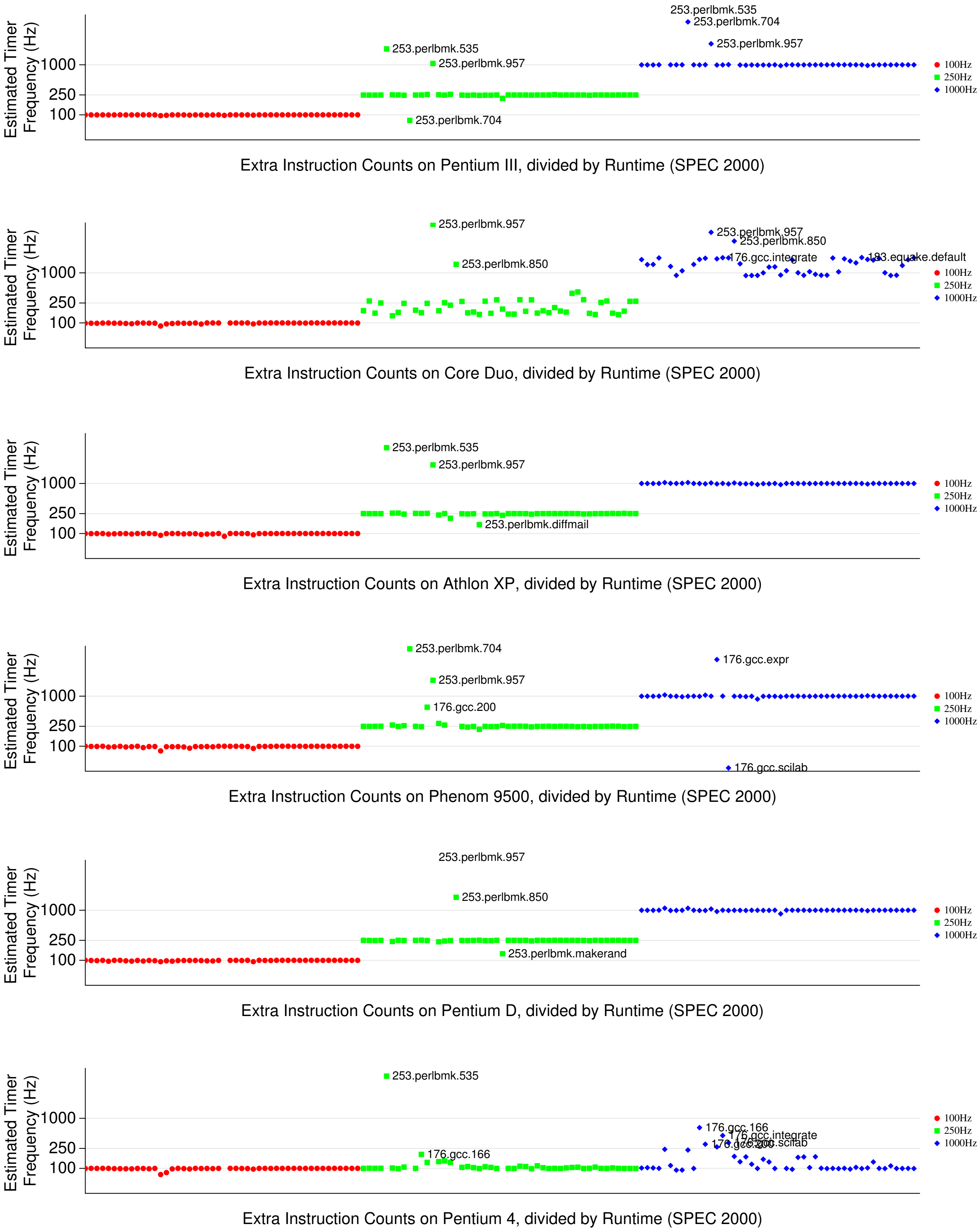

Figure 6. SPEC CPU 2000 results run on the same machines with different scheduling (“HZ”) intervals. A baseline value is calculated based on the 100Hz results, and the predicted Hz value based on benchmark run-time is plotted. With the exception of the Pentium 4, the machines show that overhead is relative to timer interrupt frequency.

Machine	Actual MHz	Derived Mean MHz	Standard Deviation	% Error
Pentium Pro	199	196	2	1.2%
Pentium II	401	397	5	0.9%
Pentium III	547	541	11	1.2%
Pentium 4	2800	2760	70	1.4%
Pentium D	3467	3435	67	0.9%
Athlon XP	1665	1645	30	1.2%
Phenom 9500	2200	2111	281	4.1%
Core Duo	1663	1635	61	1.7%
Core2 Q6600	2400	2353	113	1.9%

Table 4. Estimated cycle counts based on full SPEC 2000 and 2006 results. The Phenom was undergoing unrelated frequency scaling experiments (where some cores were clocked to 1.1GHz) during this preliminary study, which potentially accounts for the larger error.

dertook preliminary investigations of this counter, as it can be used in conjunction with retired instructions to calculate the CPI and IPC metrics. Table 4 shows our findings.

We found that the cycle count divided by time closely matched the actual clock cycle of the processor, with less than 2% error in all cases but the Phenom chip. The Phenom results are off, most likely due to unrelated research being done on the same machine by another researcher that occasionally forced various cores to run at a slower (1.1GHz) frequency.

These results, in conjunction with the retired instruction results shown earlier, show that CPI and IPC calculated with performance counters can be expected to be reasonably accurate.

A.5 Complete Final Results (Graphical)

Due to space limitations, the IISWC version of this paper only had detailed plots for a limited number of the inter-machine results. Figures 7 through 10 contain the complete results.

A.6 Complete Results (Tabular)

In addition to the graphical results, we generate tabular results which show more detail. Tables 5 through 12 contain these detailed results.

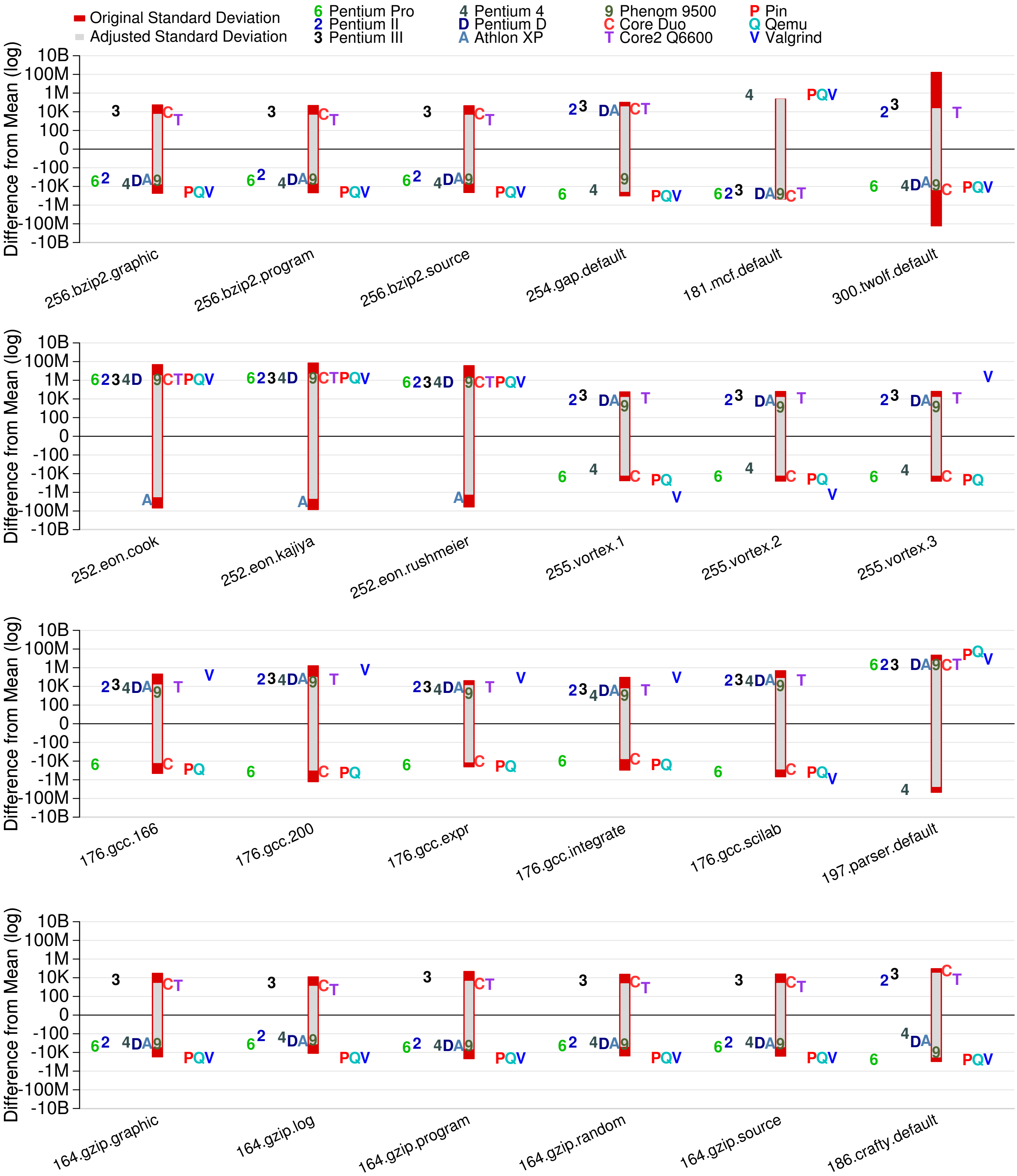

Figure 7. Complete inter-machine results for SPEC CPU 2000, part 1.

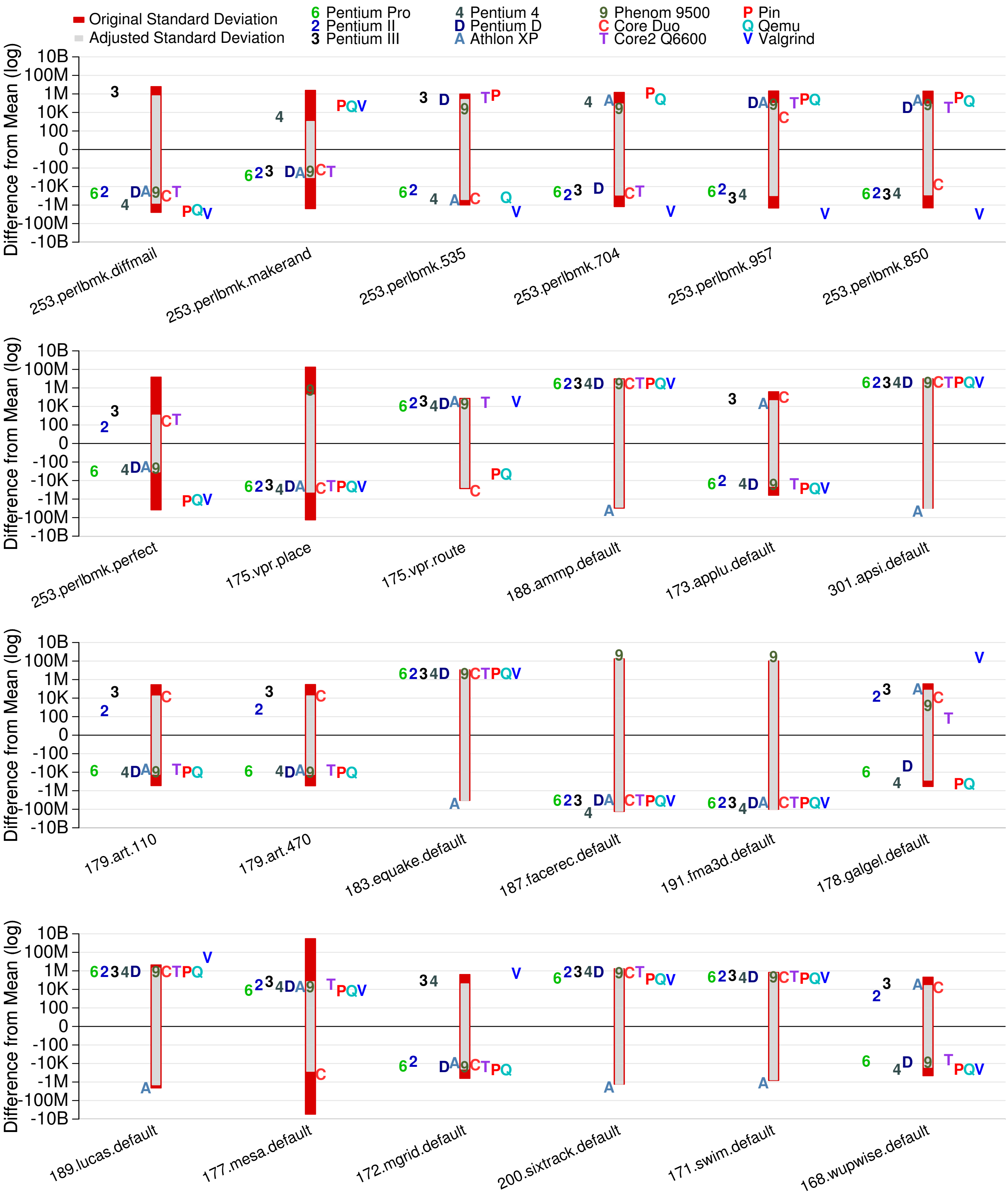

Figure 8. Complete inter-machine results for SPEC CPU 2000, part 2.

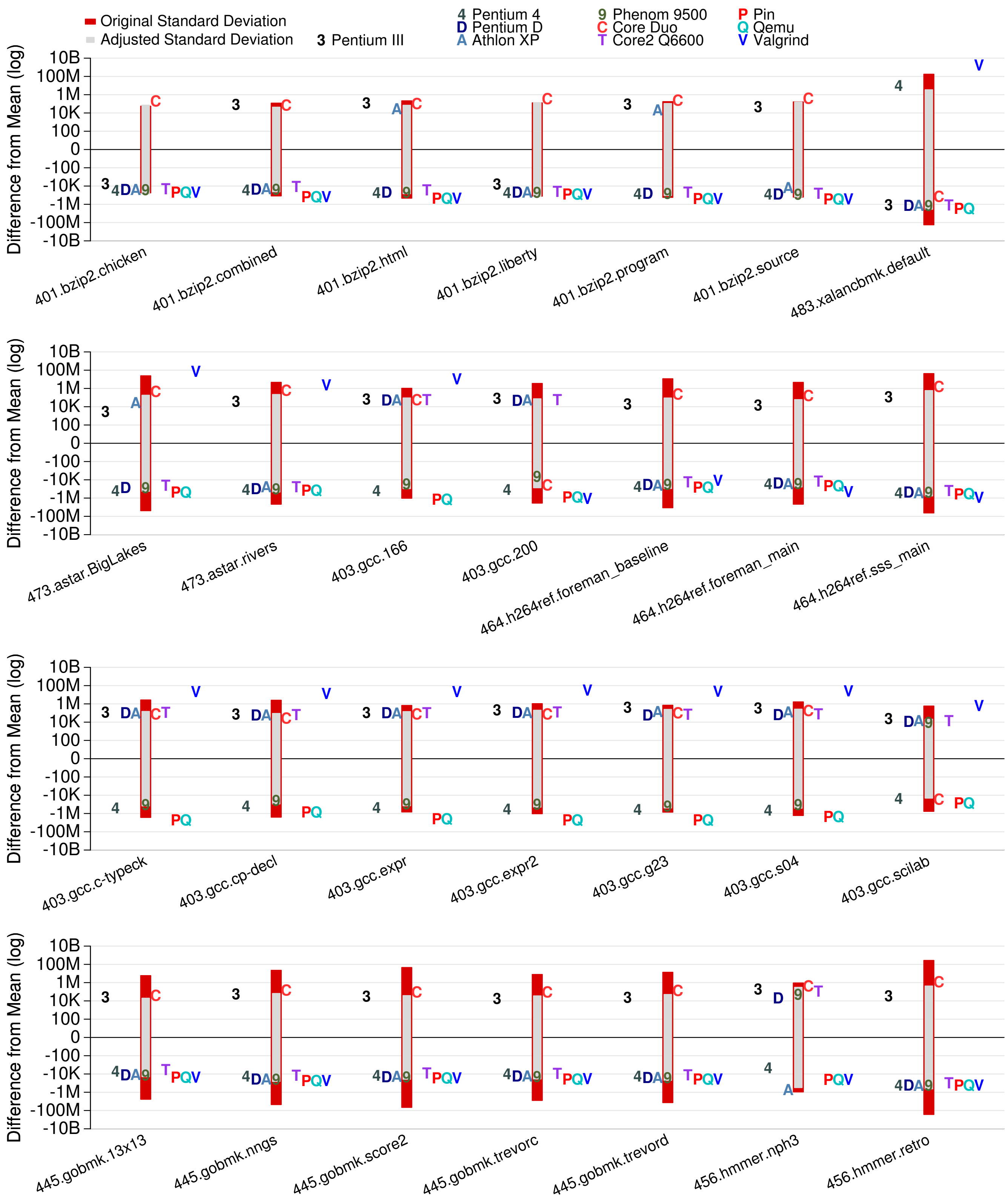

Figure 9. Complete inter-machine results for SPEC CPU 2006, part 1.

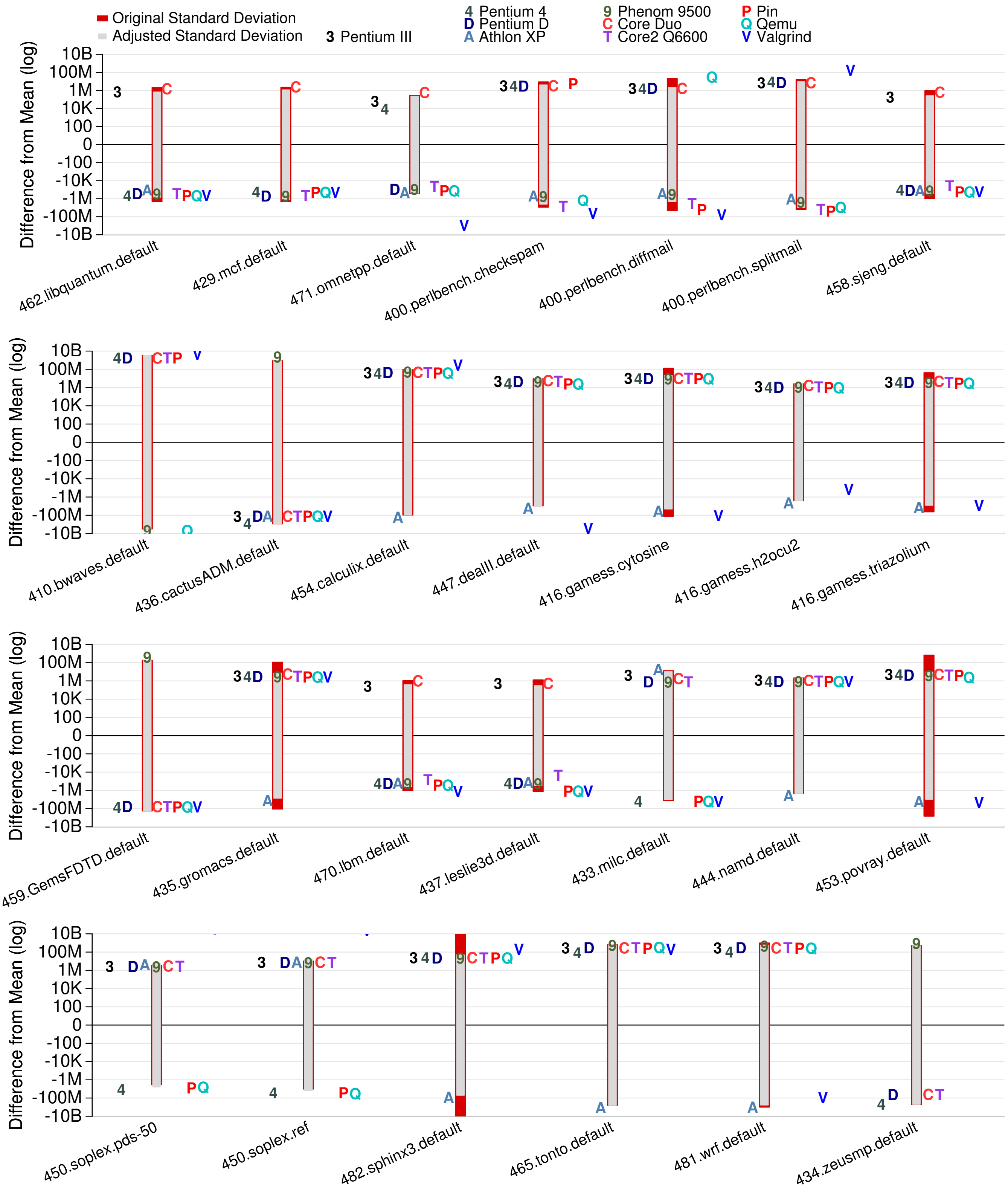

Figure 10. Complete inter-machine results for SPEC CPU 2006, part 2.

Benchmark	Retired Instructions (mean)	Pin	Qemu	Valgrind	Pentium Pro 200MHz	Pentium II 400MHz	Pentium III 550MHz	Pentium 4 2.8GHz	Pentium D 3.46GHz	Athlon XP 1.6GHz	Phenom 9500 2.2GHz	Core Duo 1.6GHz	Core2 Q6600 2.4GHz
188.ammp	333,169,537,028	-242k	-239k	-204k	946k	305k	238k	10M	10M	-22M	-151k	-103k	-129k
173.applu	554,510,334,050	-355k	-352k	-352k	885k	236k	203k	-293k	-343k	-25k	-246k	-164k	-251k
301.apsi	648,604,335,955	2M	2M	2M	4M	3M	3M	2M	2M	-26M	3M	3M	3M
179.art 110	117,968,083,815	-244k	-241k	-59G	635k	87k	70k	-242k	-241k	139k	-204k	-17k	-226k
179.art 470	121,326,239,332	-239k	-235k	-57G	676k	98k	86k	-232k	-235k	155k	-197k	-130k	-219k
256.bzip2 graphic	117,529,031,695	-96k	-93k	-92k	119k	38k	28k	-48k	-48k	-3k	-34k	-15k	-35k
256.bzip2 program	103,252,333,989	-89k	-86k	-85k	103k	34k	25k	-41k	-41k	-7k	-29k	-13k	-30k
256.bzip2 source	86,640,141,078	-88k	-84k	-84k	97k	31k	23k	-40k	-39k	-3k	-28k	-11k	-29k
186.crafty	215,657,969,685	-154k	-151k	-151k	189k	88k	50k	-82k	-75k	-54k	-76k	22k	-63k
252.eon cook	85,169,592,822	-22M	-22M	-22M	-22M	-22M	-22M	84M	84M	-31M	-22M	-22M	-22M
252.eon kajiya	109,376,544,142	-34M	-34M	-34M	-33M	-33M	-33M	125M	125M	-47M	-34M	-33M	-34M
252.eon rushmeier	62,991,805,854	-18M	-18M	-18M	-17M	-18M	-18M	65M	65M	-22M	-18M	-18M	-18M
183.equake	144,982,210,025	3M	3M	3M	4M	3M	3M	3M	3M	-30M	3M	3M	3M
187.facerec	309,893,884,667	4M	4M	4M	6M	6M	6M	-262M	-167M	7M	389M	6M	6M
191.fma3d	320,967,865,356	-21M	-21M	-21M	-20M	-20M	-20M	-97M	-46M	-20M	267M	-20M	-20M
178.galgel	370,731,076,558	-474k	-465k	185M	668k	275k	237k	-404k	-400k	66k	-211k	-14k	-216k
254.gap	221,616,872,611	-222k	-218k	-218k	163k	82k	67k	-119k	-118k	-26k	-76k	90k	-63k
176.gcc 166	22,311,256,009	-355k	-410k	-199k	-89k	216k	-466	360k	-160k	-176k	-45k	-282k	177k
176.gcc 200	72,618,732,571	370k	-528k	-840k	1M	1M	-1M	1M	760k	1M	-2M	-1M	650k
176.gcc expr	7,287,029,242	-13k	-33k	75k	-12k	38k	-74k	30k	-3k	63k	-24k	-30k	13k
176.gcc integrate	7,295,086,426	25k	16k	120k	31k	-65k	-146k	141k	-37k	20k	46k	-91k	101k
176.gcc scilab	39,175,303,719	-562k	1M	2M	-155k	-118k	-186k	757k	292k	716k	-232k	-474k	-598k
164.gzip graphic	73,929,764,146	-74k	-71k	-70k	64k	25k	15k	-25k	-24k	-16k	-21k	1k	-18k
164.gzip log	29,339,120,362	-57k	-54k	-54k	26k	12k	7k	-9k	-8k	-6k	-9k	-4k	-8k
164.gzip program	105,592,114,915	-90k	-87k	-87k	88k	53k	30k	-42k	-41k	-28k	-34k	5k	-31k
164.gzip random	60,368,112,438	-67k	-64k	-63k	50k	21k	13k	-18k	-18k	-11k	-16k	-6k	-14k
164.gzip source	56,026,965,586	-69k	-66k	-66k	49k	28k	16k	-20k	-20k	-13k	-17k	-6k	-14k
189.lucas	299,121,290,857	-1M	-1M	20M	-732k	-1M	-1M	6M	6M	-7M	-1M	-1M	-1M
181.mcf	69,384,664,851	415k	418k	418k	288k	20k	21k	420k	-241k	55k	-192k	-171k	-201k
177.mesa	284,456,293,470	-1G	-1G	-1G	-1G	-1G	-1G	5G	5G	-1G	-1G	-1G	-1G
172.mgrid	502,690,704,775	-382k	-379k	95k	896k	275k	244k	-257k	-361k	-118k	-306k	-67k	-304k
197.parser	372,122,213,309	-17M	-21M	-9M	5M	-1M	33M	-42M	-4M	-8M	21M	13M	-17M
253.perlbmk 535	54,501,662,824	-546k	1M	-6M	-382k	-416k	-555k	1M	2M	-541k	-216k	-548k	-703k
253.perlbmk 704	57,747,126,088	-482k	1M	-4M	-1M	-291k	-388k	2M	2M	-884k	-240k	-959k	-1M
253.perlbmk 957	95,768,874,544	-586k	1M	-10M	-866k	-708k	-1M	3M	4M	-1M	-86k	-1M	-1M
253.perlbmk 850	110,761,427,791	-1M	914k	-12M	-922k	-1M	-931k	3M	4M	-980k	-989k	-1M	-655k
253.perlbmk diffmail	32,818,634,327	-9M	-11M	-11M	5M	-5M	3M	-11M	-6M	5M	-1M	4M	5M
253.perlbmk mkrnd	1,266,339,997	-1M	-1M	-1M	-1M	-1M	-1M	4M	4M	-1M	-1M	-1M	-1M
253.perlbmk perfect	21,366,930,787	-8M	-7M	-8M	-7M	-9M	-7M	26M	25M	-7M	-5M	-7M	-7M
200.sixtrack	907,226,974,641	-128k	-125k	-136k	732k	801k	741k	529k	422k	-4M	379k	567k	443k
171.swim	301,164,029,263	-169k	-166k	-135k	1M	389k	405k	-149k	-162k	-1M	-58k	39k	-51k
300.twolf	311,952,993,905	-84M	-84M	-84M	-83M	-84M	-84M	294M	294M	-84M	-84M	-84M	-84M
255.vortex 1	144,373,990,150	-107k	-104k	-3M	112k	57k	39k	-54k	-54k	-14k	-35k	-18k	-31k
255.vortex 2	162,519,469,808	-107k	-103k	-2M	130k	57k	39k	-58k	-58k	-19k	-40k	-16k	-35k
255.vortex 3	160,888,183,904	-117k	-114k	1M	128k	62k	42k	-63k	-61k	-18k	-42k	-10k	-37k
175.vpr place	110,384,628,255	-90M	-90M	-90M	-90M	-90M	-90M	315M	315M	-90M	-89M	-90M	-90M
175.vpr route	93,441,470,424	-61k	-58k	-29k	133k	55k	44k	-44k	-44k	34k	-26k	-126k	-26k
168.wupwise	502,204,722,870	-221k	-218k	-218k	422k	121k	97k	-219k	-218k	-51k	-145k	138k	-144k

Table 5. Initial retired instruction counts for SPEC CPU 2000 before taking actions described in the text. The individual machine results are shown as deltas against the global mean. Light grey indicates differences of 1 million to 10 million, medium grey differences of 10 million to 1 billion, dark grey indicates over 1 billion. The Valgrind difference with `art` is due to floating point issues (described in Section 4.4.2). The extra differnces with the Pentium 4 and Pentium D with the `mesa`, `twolf`, `vpr` and `eon` benchmarks are due to the `fldcw` instruction problem (described in Section 4.1).

Benchmark	Overall Standard Deviation (mean)	Pin	Qemu	Valgrind	Pentium Pro 200MHz	Pentium II 400MHz	Pentium III 550MHz	Pentium 4 2.8GHz	Pentium D 3.46GHz	Athlon XP 1.6GHz	Phenom 9500 2.2GHz	Core Duo 1.6GHz	Core2 Q6600 2.4GHz
188.ammp	9M	10	9	0	1k	8k	1k	21	8	2k	3k	162k	2k,4
173.applu	393k	1	1	0	1k	2k	2k	3	4	1k	209	58k	330,4
301.apsi	9M	10	9	0	5k	4k	1k	9	5	1k	636	488k	3k,4
179.art 110	284k	0	6	0	28k	5k	12k	8	6	2k	11k	337k	230,4
179.art 470	298k	0	6	0	22k	4k	12k	6	6	1k	11k	79k	250,4
256.bzip2 graphic	54k	1	1	0	932	1k	240	2	3	531	52	22k	98,4
256.bzip2 program	47k	1	1	0	1k	1k	159	2	3	276	58	20k	85,4
256.bzip2 source	44k	4	4	0	1k	1k	157	1	1	338	56	19k	87,4
186.crafty	94k	10	9	0	1k	3k	495	83	41	50	682	133k	122,4
252.eon cook	48M	140	34	19	3k	1k	630	273	269	262	250	14k	1k,4
252.eon kajiya	71M	21	30	16	971	2k	436	288	201	287	430	18k	1k,4
252.eon rushmeier	37M	162	35	127	1k	696	327	230	188	241	264	8k	249,4
183.equake	11M	36	33	0	1k	3k	1k	3	4	2k	592	143k	20,4
187.facerec	176M	3	5	6	2k	4k	1k	33	16	5k	6k	57k	6k,4
191.fma3d	103M	1	6	0	2k	2k	2k	120	87	4k	4k	529k	11k,4
178.galgel	353k	10	13	5	3k	7k	1k	10	3	526	658	405k	1k,4
254.gap	103k	13	12	0	1k	3k	1k	17	19	609	213	302k	398,4
176.gcc 166	210k	503k	47	51	905k	843k	527k	740k	815k	324k	711k	486k	1M,4
176.gcc 200	1M	5M	70	87	3M	2M	5M	6M	5M	3M	2M	2M	6M,4
176.gcc expr	41k	147k	57	67	119k	135k	60k	94k	85k	99k	102k	48k	102k,4
176.gcc integrate	93k	333k	42	56	337k	333k	300k	76k	460k	135k	340k	266k	393k,4
176.gcc scilab	485k	1M	55	83	2M	3M	2M	1M	2M	1M	1M	2M	1M,4
164.gzip graphic	30k	7	6	0	1k	1k	337	11	16	32	42	32k	2k,4
164.gzip log	12k	7	6	0	259	721	119	11	10	8	8	4k	19,4
164.gzip program	47k	7	6	0	1k	3k	381	6	7	28	52	61k	108,4
164.gzip random	23k	4	4	0	1k	1k	135	13	14	22	19	7k	71,4
164.gzip source	25k	4	4	0	569	1k	240	12	17	16	47	9k	1k,4
189.lucas	4M	4	1	5	946	3k	443	5	3	2k	340	383k	56,4
181.mcf	231k	10	9	0	1k	2k	1k	8	8	2k	701	39k	929,4
177.mesa	3G	7	6	0	666	4k	591	26	9	422	529	73k	630,4
172.mgrid	409k	1	1	0	4k	9k	6k	5	2	2k	332	149k	505,4
197.parser	22M	10	27M	0	23M	33M	39M	26	32M	22M	69M	26M	35M,4
253.perlbmk 535	990k	356k	147k	507k	365k	232k	284k	256k	89k	269k	169k	36k	235k,4
253.perlbmk 704	1M	294k	303k	195k	344k	267k	789k	451k	498k	586k	479k	680k	88k,4
253.perlbmk 957	2M	329k	526k	1M	455k	567k	453k	676k	233k	247k	661k	1M	209k,4
253.perlbmk 850	2M	33k	304k	466k	206k	484k	441k	239k	175k	166k	489k	93k	29k,4
253.perlbmk diffmail	6M	11	5	35	522	430	669k	1M	1k	1k	1k	13k	935,4
253.perlbmk mkrnd	2M	7	2	5	40	23	52	46	59	26	6	800	2,4
253.perlbmk perfect	14M	9	34	25	325	444	50	32	1k	47	19	10k	461,4
200.sixtrack	1M	8	7	0	1k	6k	2k	53	41	345	1k	293k	846,4
171.swim	689k	10	9	0	19k	6k	3k	4	6	2k	3k	261k	2k,4
300.twolf	167M	1	1	0	8k	5k	2k	17	6	1k	3k	634k	1k,4
255.vortex 1	57k	9	8	0	971	1k	449	87	101	103	100	68k	19,4
255.vortex 2	63k	9	8	0	1k	750	254	210	44	54	40	83k	56,4
255.vortex 3	64k	9	8	0	2k	667	312	93	1k	163	57	84k	89,4
175.vpr place	178M	10	9	0	2k	1k	474	5	11	552	615	46k	106,4
175.vpr route	75k	10	10	0	1k	558	254	49	14	805	204	80k	229,4
168.wupwise	213k	3	3	0	2k	8k	1k	3	6	686	284	523k	6k,4

Table 6. Initial overall and per-machine standard deviations for SPEC CPU 2000. Most benchmarks are run 7 times; if fewer runs exist than the total number is listed after the variation. Light grey indicates deviation of 1k to 10k, medium grey 10k to 100k, dark grey over 100k. The slower machines are more sensitive to run-time related variation (due to number of interrupts). `parser`'s high variation is due to the heap-location issues described in Section 4.2.1. `perlbmk` and `gcc` variation might be due to programming issues, we are still investigating. The Core Duo machine consistently has high variation, we are still investigating.

Benchmark	Retired Instructions (mean)	Pin	Qemu	Valgrind	Pentium III 550MHz	Pentium 4 2.8GHz	Pentium D 3.46GHz	Athlon XP 1.6GHz	Phenom 9500 2.2GHz	Core Duo 1.6GHz	Core2 Q6600 2.4GHz
473.astar BigLakes	435,525,622,608	-14M	-14M	45M	-14M	35M	35M	-14M	-14M	-14M	-14M
473.astar rivers	870,946,649,357	-3M	-3M	-1M	-2M	6M	6M	-2M	-3M	-2M	-3M
410.bwaves	2,494,425,851,533	1G	-5G	3G	N/A	1G	1G	N/A	-5G	1G	1G
401.bzip2 chicken	199,232,705,385	-77k	-74k	-74k	95k	-51k	-50k	58k	-22k	-680	-28k
401.bzip2 combined	364,136,194,713	-265k	-262k	-261k	235k	-105k	-104k	73k	-49k	5k	-55k
401.bzip2 html	706,417,217,673	-420k	-417k	-416k	383k	-201k	-200k	166k	-91k	49k	-105k
401.bzip2 liberty	346,361,904,358	-139k	-136k	-135k	210k	-112k	-112k	152k	-59k	-9k	-69k
401.bzip2 program	593,333,246,889	-381k	-378k	-377k	309k	-162k	-161k	111k	-90k	88k	-96k
401.bzip2 source	452,012,760,241	-374k	-371k	-370k	281k	-152k	-152k	131k	-82k	65k	-90k
436.cactusADM	3,150,039,559,978	-124M	-124M	-124M	-120M	-1G	-376M	-122M	1G	-122M	-123M
454.calculix	8,687,234,268,125	24M	25M	211M	33M	41M	41M	-215M	36M	31M	29M
447.dealII	2,334,571,013,694	1M	1M	N/A	4M	3M	3M	-21M	3M	3M	3M
416.gamess cytosine	1,143,080,276,304	-65M	-65M	-222M	-64M	189M	189M	-119M	-65M	-64M	-65M
416.gamess h2ocu2	867,682,160,546	626k	634k	-300k	1M	854k	881k	-5M	892k	1M	828k
416.gamess triazolium	4,215,218,572,760	-21M	-21M	-35M	-18M	61M	61M	-42M	-21M	-20M	-21M
403.gcc 166	85,717,377,824	-880k	1M	10M	916k	-1M	-86k	1M	92k	-637k	-772k
403.gcc 200	166,629,701,342	-1M	212k	-3M	-2M	5M	2M	-1M	-5M	1M	-531k
403.gcc c-typeck	140,813,669,344	-7M	16k	4M	1M	-468k	1M	4M	-943k	-4M	-1M
403.gcc cp-decl	109,536,887,364	-304k	4M	15M	-1M	1M	-43k	4M	716k	-2M	-3M
403.gcc expr	118,135,701,275	-5M	-4M	26M	45k	-63k	-284k	155k	1M	-224k	-947k
403.gcc expr2	160,293,781,230	-9M	-5M	19M	710k	1M	-2M	-223k	331k	-518k	779k
403.gcc g23	193,775,636,958	-6M	-6M	20M	655k	-610k	-951k	551k	-607k	120k	842k
403.gcc s04	179,205,608,854	-6M	-3M	11M	2M	1M	-792k	-303k	-1M	634k	-2M
403.gcc scilab	64,699,183,368	338k	-2M	-1M	-129k	411k	30k	-1M	623k	317k	-77k
445.gobmk 13x13	238,223,728,722	-3M	-3M	-3M	-3M	8M	8M	-3M	-3M	-3M	-3M
445.gobmk nngs	631,500,778,072	-13M	-13M	-13M	-12M	33M	33M	-13M	-13M	-13M	-13M
445.gobmk score2	345,180,118,022	-26M	-26M	-26M	-26M	66M	66M	-26M	-26M	-26M	-26M
445.gobmk trevorc	236,510,344,415	-4M	-4M	-4M	-4M	11M	11M	-4M	-4M	-4M	-4M
445.gobmk trevord	340,197,007,561	-8M	-8M	-8M	-7M	20M	20M	-8M	-8M	-8M	-8M
459.GemsFDTD	2,511,629,771,597	-85M	-85M	-84M	N/A	-89M	-82M	N/A	329M	-75M	-81M
435.gromacs	2,929,336,245,188	-64M	-64M	-64M	-60M	166M	166M	-81M	-63M	-62M	-63M
464.h264ref forebase	564,686,760,152	-7M	-7M	-6M	-6M	17M	17M	-6M	-6M	-6M	-6M
464.h264ref foremain	323,104,250,969	-2M	-2M	-3M	-2M	6M	6M	-2M	-2M	-2M	-2M
464.h264ref sss	2,814,699,342,583	-26M	-26M	-26M	-23M	62M	62M	-25M	-25M	-25M	-25M
456.hmmer nph3	1,039,885,071,862	-419k	-416k	-415k	374k	1M	1M	-825k	-192k	-1M	-174k
456.hmmer retro	2,212,959,503,256	-160M	-160M	-160M	-159M	400M	400M	-160M	-160M	-160M	-160M
470.lbm	1,495,738,692,277	-1M	-1M	-2M	1M	-1M	-1M	1M	-685k	566k	-512k
437.leslie3d	2,534,172,444,247	-2M	-2M	-2M	2M	-1M	-1M	687k	-747k	38k	-546k
462.libquantum	3,884,594,828,362	-1M	-1M	-1M	4M	-1M	-1M	1M	-1M	112k	-1M
429.mcf	449,896,007,119	-773k	-770k	-769k	N/A	-766k	-1M	N/A	-881k	3M	-970k
433.milc	1,386,819,554,851	-18M	-18M	-19M	7M	-18M	-18M	17M	3M	4M	3M
444.namd	2,895,739,443,120	281k	285k	289k	1M	302k	302k	-4M	577k	1M	577k
471.omnetpp	764,012,914,003	-554k	-551k	-1G	476k	110k	-23k	125k	-296k	-92k	-300k
400.perlbench checkspam	148,058,875,551	8M	-15M	-62M	2M	-9M	-14M	2M	5M	4M	10M
400.perlbench diffmail	401,941,912,714	-53M	-25M	-99M	-21M	6M	-684k	-4M	-20M	-1M	42M
400.perlbench splitmail	714,310,897,663	-23M	-24M	155M	-1M	35M	476k	-2M	-13M	-6M	-12M
453.povray	1,204,553,566,871	-395M	-395M	-427M	-394M	1G	1G	-420M	-395M	-394M	-396M
458.sjeng	2,530,950,917,182	-903k	-899k	-899k	993k	-779k	-670k	-317k	-517k	1M	-498k
450.soplex pds-50	450,971,154,301	-13M	-8M	25G	181k	-5M	6M	1M	-606k	-255k	-557k
450.soplex ref	459,069,286,338	-37M	-41M	17G	-1M	-13M	21M	-1M	-1M	-1M	-1M
482.sphinx3	2,834,665,823,811	-6G	-6G	-6G	-6G	17G	17G	-6G	-6G	-6G	-6G
465.tonto	2,895,396,300,252	187M	230M	135M	210M	134M	227M	-1G	433M	209M	207M
481.wrf	4,117,369,090,579	-208M	-208M	-537M	-188M	1G	1G	-1G	-56M	-191M	-192M
483.xalancbmk	1,313,537,753,450	-106M	-106M	1G	-102M	263M	254M	-103M	-104M	-103M	-104M
434.zeusmp	2,397,598,208,777	10M	10M	N/A	N/A	-533M	-309M	N/A	818M	11M	11M

Table 7. Initial retired instruction counts for SPEC CPU 2006 before taking actions described in the text. The individual machine results are shown as deltas against the global mean. Light grey indicates differences of 1 million to 10 million, medium grey differences of 10 million to 1 billion, dark grey indicates over 1 billion. Entries marked N/A are benchmarks that could not be run due to memory constraints.

Benchmark	Overall Standard Deviation (mean)	Pin	Qemu	Valgrind	Pentium III 550MHz	Pentium 4 2.8GHz	Pentium D 3.46GHz	Athlon XP 1.6GHz	Phenom 9500 2.2GHz	Core Duo 1.6GHz	Core2 Q6600 2.4GHz
473.astar BigLakes	24M	4	3	0	5k	20	20	2k	17k	766k	975,3
473.astar rivers	4M	4	4	0	1k	28	9,6	2k	41k	1M	7k,3
410.bwaves	3G	1	1	0	N/A	28	18	N/A	150k	2M	631,3
401.bzip2 chicken	56k	1	1	0	382	5	7	685	7k	12k	168,3
401.bzip2 combined	121k	1	1	0	2k	15	12	812	15k	47k	294,3
401.bzip2 html	215k	1	1	0	1k	20	11	2k	32k	59k	22k,3
401.bzip2 liberty	130k	1	1	0	771	5	5	1k	13k	27k	396,3
401.bzip2 program	176k	4	3	0	1k	8	13	1k	21k	48k	50,3
401.bzip2 source	164k	1	1	0	1k	19	16	877	20k	60k	1k,3
436.cactusADM	895M	6	8	0	22k	102	87	5k	22k	1M	867,3
454.calculix	95M	1	0	0	27k	27	14	4k	289k	2M	1k,3
447.dealII	9M	7	5	N/A	12k	38	15	3k	74k	620k	2k,3
416.gamess cytosine	131M	1	1	0	5k	25	229	1k	35k	316k	1k,3
416.gamess h2ocu2	2M	9	7	0	1k	12	11	585	26k	392k	570,3
416.gamess triazolium	43M	9	7	0	5k	54	19	2k	128k	353k	277,3
403.gcc 166	1M	2M	2	0	2M	3M	1M,6	2M	3M	2M	2M,3
403.gcc 200	3M	13M	2	0	7M	15M	10M	11M	11M	10M	10M,3
403.gcc c-typeck	2M	5M	6	0	3M	2M	3M	3M	8M	8M	509k,3
403.gcc cp-decl	2M	5M	3	0	5M	3M	3M	2M	5M	6M	2M,3
403.gcc expr	682k	2M	6	0	1M	1M	1M	2M	3M	1M	360k,3
403.gcc expr2	1M	2M	4	0	3M	2M	2M	1M	1M	2M	1M,3
403.gcc g23	719k	561k	2	0	738k	763k	603k	1k	550k	568k	645k,3
403.gcc s04	1M	10M	2	0	4M	4M	3M	2k	5M	4M	4M,3
403.gcc scilab	586k	1M	7	0	1M	1M	1M	1M	865k	584k	1M,3
445.gobmk 13x13	5M	3	3	0	1k	119	172	91	12k	13k	920,3
445.gobmk nngs	22M	5	5	0	1k	230	133	216	33k	41k	62,3
445.gobmk score2	45M	6	6	0	842	83	152	196	17k	52k	174,3
445.gobmk trevorc	7M	2	1	0	626	125	130	96	12k	59k	177,3
445.gobmk trevord	13M	1	1	0	537	96	186	142	16k	81k	300,3
459.GemsFDTD	184M	4	4	0	N/A	49	17	N/A	9k,6	10M	471,3
435.gromacs	113M	60	67	34	7k	54	55	5k	209k	946k	130k,3
464.h264ref forebase	11M	0	7	0	1k	15	31	402	4k	62k	191,3
464.h264ref foremain	4M	1	7	0	2k	71	58	105	13k	30k	547,3
464.h264ref sss	42M	1	7	0	26k	372	286	1k	117k	189k	1k,3
456.hmmer nph3	930k	5	4	0	1k	72	28	1k	38k	76k	1k,3
456.hmmer retro	273M	13	13	0	6k	28	23	3k	64k	440k	1k,3
470.lbm	1M	12	9	0	21k	23	13	13k	55k	1M	28,3
437.leslie3d	1M	62	74	97	6k	73	124	11k	20k	425k	657,3
462.libquantum	2M	8	6	0	10k	12	9	15k	4k	1M	67,3
429.mcf	2M	3	2	0	N/A	15	6	N/A	56k	3M,6	229,3
433.milc	13M	6	5	0	97k	36	11	3k	591	1M	111,3
444.namd	2M	3	2	0	11k	10	5	1k	65k	1M	324,3
471.omnetpp	271k	3	2	0	8k	27	25	22k	38k	209k	648,3
400.perlbench checkspam	8M	4k	3k	4k	100k	92k	2M	101k	8k	91k	443,3
400.perlbench diffmail	21M	19	16	10	1k	27	16	20M	12k	30k	629,3
400.perlbench splitmail	16M	8k	38k	8k	23k	8k	8k	495k	24k	47k	316,3
453.povray	683M	1M	1M	827k	1M	1M	1M	1M	298k	289k	1M,3
458.sjeng	988k	1	1	0	14k	518	689	1k	51k	4M	1k,3
450.soplex pds-50	3M	0	2M	0	2k	344	108	4k	27k	446k	684,3
450.soplex ref	10M	1M	5M	0	13k	341	147	6k	41k	195k	58k,3
482.sphinx3	11G	2	2	0	32k	1k	601	6k	50k	729k	638,3
465.tonto	634M	121	122	237	7k	124	168	7k	103k	2M	3k,3
481.wrf	1G	417	376	380	9k,6	395	281	2k	117k	759k	4k,3
483.xalancbmk	177M	2	2	4	3k,6	7k	1k	2k	48k	144k	1k,3
434.zeusmp	512M	1	1	N/A	N/A	31	32	N/A	118k	534k	75k,3

Table 8. Initial overall and per-machine standard deviations for SPEC CPU 2006. Most benchmarks are run 7 times; if fewer runs exist than the total number is listed after the variation. Light grey indicates deviation of 1k to 10k, medium grey 10k to 100k, dark grey over 100k. The slower machines are more sensitive to run-time related variation (due to number of interrupts). Variation in `perlbench` is due to stack-related issues described in Section 4.2.1. `gcc` variation might be due to programming issues, we are still investigating. The Core Duo machine consistently has high variation, we are still investigating.

Benchmark	Retired Instructions (mean)	Pin	Qemu	Valgrind	Pentium Pro 200MHz	Pentium II 400MHz	Pentium III 550MHz	Pentium 4 2.8GHz	Pentium D 3.46GHz	Athlon XP 1.6GHz	Phenom 9500 2.2GHz	Core Duo 1.6GHz	Core2 Q6600 2.4GHz
188.ammp	333,166,801,392	2M	2M	2M	2M	2M	2M	2M	2M	-20M	2M	2M	2M
173.applu	554,510,063,746	-85k	-85k	-84k	-26k	-12k	57k	-26k	-30k	16k	-30k	82k	-30k
301.apsi	648,604,012,858	3M	3M	3M	3M	3M	3M	3M	3M	-26M	3M	3M	3M
179.art 110	117,967,850,842	-11k	-11k	-59G	-7k	367	41k	-12k	-10k	-5k	-10k	11k	-6k
179.art 470	121,326,013,826	-13k	-13k	-57G	-9k	515	41k	-9k	-12k	-7k	-11k	14k	-7k
256.bzip2 graphic	117,528,986,436	-50k	-50k	-50k	-3k	-1k	10k	-6k	-2k	-2k	-2k	7k	1k
256.bzip2 program	103,252,294,950	-50k	-50k	-49k	-2k	-648	9k	-5k	-2k	-1k	-1k	4k	1k
256.bzip2 source	86,640,103,449	-50k	-50k	-49k	-2k	-902	8k	-5k	-2k	-1k	-2k	4k	1k
186.crafty	215,657,884,928	-70k	-69k	-69k	-68k	4k	22k	-107	-907	-638	-10k	48k	5k
252.eon cook	85,145,647,912	997k	997k	999k	997k	1M	1M	1M	1M	-8M	1M	1M	1M
252.eon kajiya	109,341,034,194	1M	1M	1M	1M	1M	1M	1M	1M	-11M	1M	1M	1M
252.eon rushmeier	62,973,164,545	530k	531k	531k	530k	538k	545k	537k	537k	-4M	536k	537k	539k
183.equake	144,982,069,897	3M	3M	3M	3M	3M	3M	3M	3M	-30M	3M	3M	3M
187.facerec	309,913,035,961	-15M	-15M	-15M	-13M	-12M	-12M	-281M	-12M	-12M	370M	-12M	-12M
191.fma3d	320,970,437,842	-23M	-23M	-23M	-23M	-23M	-23M	-100M	-23M	-23M	265M	-23M	-23M
178.galgel	370,730,834,889	-232k	-232k	185M	-11k	13k	80k	-169k	-2k	78k	1k	9k	53
254.gap	221,616,777,789	-127k	-127k	-127k	-78k	15k	37k	-27k	10k	11k	-1k	16k	16k
176.gcc 166	22,310,934,566	-92k	-91k	129k	-25k	7k	13k	5k	6k	7k	2k	-24k	7k
176.gcc 200	72,618,408,814	-208k	-208k	483k	-156k	48k	60k	40k	43k	53k	26k	-165k	47k
176.gcc expr	7,287,033,719	-41k	-41k	67k	-28k	7k	7k	4k	6k	6k	1k	-12k	6k
176.gcc integrate	7,295,128,587	-28k	-28k	75k	-11k	3k	4k	885	2k	2k	993	-7k	3k
176.gcc scilab	39,177,380,041	-197k	-197k	-933k	-154k	39k	45k	31k	36k	41k	9k	-87k	38k
164.gzip graphic	73,929,737,445	-47k	-47k	-47k	-2k	-987	5k	-992	-1k	-1k	-1k	1k	1k
164.gzip log	29,339,109,169	-46k	-46k	-46k	-1k	-190	2k	-284	-701	-727	-563	1k	419
164.gzip program	105,592,072,365	-48k	-48k	-47k	-3k	-1k	10k	-1k	-2k	-2k	-2k	1k	1k
164.gzip random	60,368,092,367	-47k	-47k	-46k	-2k	-922	4k	-960	-1k	-1k	-1k	3k	684
164.gzip source	56,026,943,532	-47k	-47k	-47k	-3k	-981	5k	-968	-1k	-1k	-1k	2k	876
189.lucas	299,119,239,955	620k	620k	22M	670k	678k	720k	627k	668k	-5M	694k	679k	668k
181.mcf	69,384,512,278	567k	567k	568k	-73k	-64k	-28k	570k	-72k	-67k	-71k	-128k	-64k
177.mesa	282,923,956,697	5k	5k	6k	6k	25k	58k	15k	17k	15k	16k	-182k	26k
172.mgrid	502,690,381,546	-59k	-59k	415k	-23k	-7k	76k	63k	-27k	-10k	-27k	-17k	-26k
197.parser	372,094,065,184	21M	45M	6M	1M	1M	1M	-13M	1M	1M	1M	1M	1M
253.perlbmk 535	54,500,597,555	565k	-179k	-6M	-47k	-26k	361k	-257k	201k	-335k	19k	-236k	319k
253.perlbmk 704	57,746,243,838	1M	222k	-6M	-45k	-97k	-27k	109k	-18k	160k	22k	-65k	-38k
253.perlbmk 957	95,767,667,750	212k	197k	-10M	-38k	-22k	-190k	-84k	92k	90k	67k	2k	82k
253.perlbmk 850	110,760,317,639	310k	131k	-11M	-72k	-57k	-80k	-64k	29k	164k	61k	-7k	27k
253.perlbmk diffmail	32,815,554,959	-6M	-4M	-10M	-61k	-45k	1M	-1M	-47k	-39k	-48k	-123k	-45k
253.perlbmk mkrnd	1,265,083,274	42k	35k	39k	-752	-393	-253	2k	-292	-398	-278	-189	-298
253.perlbmk perfect	21,360,587,472	-2M	-1M	-1M	-1k	53	2k	-823	-466	-418	-546	225	310
200.sixtrack	907,226,745,184	100k	100k	89k	135k	622k	673k	708k	608k	-4M	543k	545k	628k
171.swim	301,163,730,733	129k	129k	159k	185k	214k	272k	146k	181k	-1M	184k	163k	200k
300.twolf	311,868,486,658	-14k	-14k	-14k	-11k	7k	51k	-9k	-8k	-4k	-8k	-24k	6k
255.vortex 1	144,373,937,241	-54k	-54k	-3M	-23k	7k	20k	-4k	5k	5k	1k	-23k	10k
255.vortex 2	162,519,411,996	-49k	-49k	-2M	-23k	7k	20k	-3k	4k	5k	1k	-23k	9k
255.vortex 3	160,888,123,221	-57k	-57k	1M	-23k	7k	21k	-4k	5k	5k	1k	-22k	10k
175.vpr place	110,294,461,470	-54k	-54k	-53k	-52k	-48k	-33k	-110k	-51k	-50k	474k	-78k	-46k
175.vpr route	93,441,411,107	-2k	-2k	26k	8k	19k	32k	9k	16k	26k	16k	-151k	21k
168.wupwise	502,204,554,585	-53k	-53k	-53k	-6k	1k	38k	-53k	-8k	28k	-8k	12k	-4k

Table 9. Final average retired instruction counts for SPEC CPU 2000 after taking actions described in the text. The individual machine results are shown as deltas against the global mean. Light grey indicates differences of 1 million to 10 million, medium grey differences of 10 million to 1 billion, dark grey indicates over 1 billion. The Valgrind difference with `art` is due to floating point issues (described in section 4.4.2). Remaining error in *eon* and *facerec* are still unexplained.

Benchmark	Overall Standard Deviation (mean)	Pin	Qemu	Valgrind	Pentium Pro 200MHz	Pentium II 400MHz	Pentium III 550MHz	Pentium 4 2.8GHz	Pentium D 3.46GHz	Athlon XP 1.6GHz	Phenom 9500 2.2GHz	Core Duo 1.6GHz	Core2 Q6600 2.4GHz
188.ammp	7M	0	0	0	2k	3k	1k	26	772	631	2k	31k	2k,6
173.applu	42k	3	0	0	1k	8k	705	7,6	54	716	101	219k	59,6
301.apsi	9M	0	0	0	1k	14k	1k	500	164	872	254	585k	77,6
179.art 110	17k	0	6	0	1k	4k	1k	6	56	1k	136	12k	96,6
179.art 470	17k	0	6	0	1k	9k	1k	5	87	1k	166	16k	154,6
256.bzip2 graphic	5k	3	0	0	544	297	281	4	44	209	35	7k	691,6
256.bzip2 program	4k	0	0	0	197	1k	197	5	81	160	187	6k	139,6
256.bzip2 source	4k	0	0	0	965	936	195	4	77	230	111	5k	606,6
186.crafty	30k	1	0	0	512	3k	254	26	18	118	94	112k	60,6
252.eon cook	3M	23	158	142	379	636	182	182	284	233	296	5k	295,6
252.eon kajiya	4M	23	118	108	290	587	191	241	171	234	244	6k	298,6
252.eon rushmeier	1M	51	166	34	121	284	232	316	240	180	303	3k	263,6
183.equake	11M	0	0	0	972	2k	425	10	370	880	41	154k	95,6
187.facerec	164M	7	0	0	1k	3k	570	42	1k	1k	6k	61k	4k,6
191.fma3d	102M	3	0	0	1k	3k	428	217	1k	991	4k	47k	4k,6
178.galgel	72k	16	2	2	1k	6k	1k	10	131	726	61	14k	491,6
254.gap	34k	4	0	0	449	2k	345	11	62	219	543	8k	175,6
176.gcc 166	14k	107	29	62	190	774	251	162	80	122	207	7k	203,6
176.gcc 200	91k	55	58	99	367	1k	284	108	324	685	7k	14k	708,6
176.gcc expr	12k	62	53	35	150	908	159	45	68	106	131	2k	74,6
176.gcc integrate	5k	42	54	77	107	415	139	92	30	95	55	3k	57,6
176.gcc scilab	71k	83	53	94	421	1k	390	139	122	1k	1k	15k	427,6
164.gzip graphic	2k	0	0	0	158	199	65	20	8	42	48	4k	188,6
164.gzip log	1k	3	0	0	48	928	74	16	3	9	26	1k	100,6
164.gzip program	4k	3	0	0	249	314	183	19	7	47	112	5k	120,6
164.gzip random	2k	5	0	0	131	240	59	17	7	200	92	3k	13,6
164.gzip source	2k	6	0	0	228	223	65	17	19	24	34	3k	67,6
189.lucas	2M	3	0	0	757	4k	650	4	32	481	59	13k	60,6
181.mcf	215k	2	0	0	952	4k	742	17	135	600	934	9k	596,6
177.mesa	69k	3	0	0	786	2k	551	47	51	211	528	9k	2k,6
172.mgrid	40k	3	0	0	1k	8k	877	5	22	638	114	15k	127,6
197.parser	5M	6M	0	0	720	3k	892	14	129	341	483	13k	360,6
253.perlbmk 535	252k	376k	181k	26k	476k	516k	419k	281k	501k	263k	521k	487k	424k,6
253.perlbmk 704	84k	352k	663k	52k	420k	343k	227k	233k	280k	243k	275k	317k	233k,6
253.perlbmk 957	95k	440k	193k	192k	491k	425k	490k	246k	87k	72k	64k	99k	68k,6
253.perlbmk 850	80k	266k	199k	94k	360k	358k	217k	306k	112k	33k	126k	124k	115k,6
253.perlbmk diffmail	637k	18	0	7	420	296	1M	1M	166	612	1k	1k	692,6
253.perlbmk mkrnd	1k	21	0	0	4	20	54	6	3	5	28	60	19,6
253.perlbmk perfect	1k	7	0	0	99	101	140	73	10	111	35	1k	27,6
200.sixtrack	1M	1	0	0	1k	4k	947	40	132	269	577	29k	84,6
171.swim	582k	1	0	0	1k	37k	1k	5	101	1k	261	15k	491,6
300.twolf	21k	3	0	0	771	8k	1k	26	108	900	276	26k	97,6
255.vortex 1	14k	0	0	0	280	441	178	49	23	145	111	7k	38,6
255.vortex 2	14k	2	0	0	795	549	197	74	5	55	167	6k	36,6
255.vortex 3	15k	2	0	0	285	454	255	32	13	134	141	7k	26,6
175.vpr place	179k	0	0	0	300	1k	307	2	80	197	92	44k	34,6
175.vpr route	57k	2	0	0	299	569	181	52	124	225	304	7k	132,6
168.wupwise	26k	2	0	0	877	4k	498	7	29	335	268	18k	5k,6

Table 10. Final overall and per-machine standard deviations for SPEC CPU 2000. Most benchmarks are run 7 times; if fewer runs exist than the total number is listed after the variation. Light grey indicates deviation of 1k to 10k, medium grey 10k to 100k, dark grey over 100k. The `gcc` variations seen in Table 6 have been removed, but the `perlbmk` variations remain (this needs investigating). The Core Duo still has high amounts of variation, which also needs investigating.

Benchmark	Retired Instructions (mean)	Pin	Qemu	Valgrind	Pentium III 550MHz	Pentium 4 2.8GHz	Pentium D 3.46GHz	Athlon XP 1.6GHz	Phenom 9500 2.2GHz	Core Duo 1.6GHz	Core2 Q6600 2.4GHz
473.astar BigLakes	435,510,972,009	-267k	-267k	60M	2k	-184k	-85k	21k	-88k	387k	-52k
473.astar rivers	870,943,440,840	-166k	-166k	1M	31k	-145k	-113k	-75k	-112k	486k	-71k
410.bwaves	2,494,425,636,829	1G	-5G	3G	N/A	1G	1G	N/A	-5G	1G	1G
401.bzip2 chicken	199,232,687,985	-60k	-60k	-60k	-6k	-36k	-31k	-30k	-31k	162k	-25k
401.bzip2 combined	364,136,119,274	-189k	-189k	-189k	70k	-32k	-27k	-25k	-27k	55k	-13k
401.bzip2 html	706,417,076,260	-278k	-278k	-278k	104k	-62k	-57k	21k	-56k	83k	-32k
401.bzip2 liberty	346,361,856,485	-91k	-91k	-90k	-7k	-67k	-61k	-59k	-61k	306k	-49k
401.bzip2 program	593,333,166,362	-300k	-300k	-300k	78k	-83k	-79k	17k	-78k	207k	-60k
401.bzip2 source	452,012,702,211	-316k	-316k	-315k	40k	-97k	-92k	-18k	-92k	335k	-75k
436.cactusADM	3,150,074,895,066	-160M	-160M	-160M	-158M	-1G	-159M	-158M	1G	-156M	-159M
454.calculix	8,687,229,110,374	30M	30M	216M	34M	29M	33M	-211M	41M	38M	34M
447.dealII	2,334,570,289,677	1M	1M	-3G	3M	2M	3M	-21M	3M	4M	3M
416.gamess cytosine	1,143,007,252,827	7M	7M	-149M	7M	7M	7M	-46M	7M	8M	7M
416.gamess h2ocu2	867,682,034,605	752k	752k	-183k	957k	802k	863k	-5M	944k	1M	891k
416.gamess triazolium	4,215,194,263,261	2M	2M	-11M	3M	2M	2M	-18M	2M	3M	2M
403.gcc 166	85,720,744,289	-1M	-1M	9M	55k	-198k	42k	41k	-35k	47k	46k
403.gcc 200	166,630,876,529	-966k	-966k	-1M	65k	-147k	38k	44k	-4k	-45k	48k
403.gcc c-typeck	140,819,836,173	-6M	-6M	17M	100k	-292k	83k	76k	-123k	65k	89k
403.gcc cp-decl	109,542,535,643	-872k	-872k	10M	66k	-189k	46k	48k	-43k	21k	50k
403.gcc expr	118,135,968,710	-4M	-4M	16M	93k	-286k	80k	74k	-110k	61k	85k
403.gcc expr2	160,294,356,532	-6M	-6M	24M	182k	-417k	93k	87k	-112k	66k	100k
403.gcc g23	193,775,908,083	-6M	-6M	19M	339k	-453k	47k	111k	-186k	85k	56k
403.gcc s04	179,205,032,366	-2M	-2M	20M	299k	-537k	54k	99k	-123k	144k	62k
403.gcc scilab	64,696,667,990	-88k	-88k	520k	20k	-27k	9k	11k	7k	-32k	10k
445.gobmk 13x13	238,220,190,813	-29k	-29k	-29k	22k	-6k	-15k	-14k	-16k	33k	-4k
445.gobmk nngs	631,487,392,799	-68k	-68k	-67k	49k	-21k	-46k	-44k	-47k	128k	-17k
445.gobmk score2	345,153,298,166	-33k	-33k	-32k	26k	-17k	-25k	-24k	-25k	76k	-9k
445.gobmk trevorc	236,505,629,198	-43k	-43k	-42k	15k	-12k	-22k	-21k	-23k	75k	-11k
445.gobmk trevord	340,188,823,216	-46k	-46k	-45k	19k	-20k	-30k	-29k	-31k	108k	-15k
459.GemsFDTD	2,511,627,666,076	-82M	-82M	-82M	N/A	-93M	-79M	N/A	331M	-78M	-79M
435.gromacs	2,929,269,770,446	2M	2M	2M	2M	2M	2M	-16M	2M	4M	2M
464.h264ref forebase	564,679,835,977	-83k	-83k	-14k	16k	-68k	-39k	-48k	-37k	199k	-20k
464.h264ref foremain	323,101,422,186	-56k	-56k	-261k	12k	-34k	-28k	-34k	-29k	133k	-18k
464.h264ref sss	2,814,673,630,472	-426k	-426k	-1M	102k	-324k	-283k	-329k	-281k	1M	-184k
456.hmmer nph3	1,039,884,702,457	-50k	-50k	-49k	159k	-2k	16k	-655k	44k	341k	95k
456.hmmer retro	2,212,798,906,108	-214k	-214k	-214k	30k	-214k	-212k	-226k	-210k	956k	-122k
470.lbm	1,495,737,916,162	-343k	-343k	-1M	214k	-235k	-235k	-211k	-233k	789k	-88k
437.leslie3d	2,534,171,622,239	-1M	-1M	-1M	423k	-219k	-202k	-181k	-221k	429k	-27k
462.libquantum	3,884,593,703,235	-616k	-616k	-615k	581k	-588k	-378k	-140k	-380k	1M	-377k
429.mcf	449,895,474,978	-241k	-241k	-240k	N/A	-237k	-626k	N/A	-622k	2M	-571k
433.milc	1,386,822,074,952	-20M	-20M	-21M	3M	-20M	626k	13M	615k	1M	632k
444.namd	2,895,739,055,062	669k	669k	673k	954k	687k	691k	-4M	696k	875k	773k
471.omnetpp	764,012,528,890	-169k	-169k	-1G	52k	6k	-112k	-262k	-112k	483k	-55k
400.perlbench checkspam	148,063,263,885	4M	-1M	-53M	2M	2M	2M	-655k	-753k	2M	-8M
400.perlbench diffmail	401,908,801,828	-19M	23M	-77M	1M	1M	1M	-359k	-387k	1M	-4M
400.perlbench splitmail	714,321,332,736	-30M	-12M	140M	5M	6M	5M	-1M	-3M	5M	-19M
453.povray	1,204,154,429,572	3M	1M	-27M	4M	3M	3M	-21M	2M	3M	3M
458.sjeng	2,530,950,250,697	-236k	-236k	-236k	149k	-137k	-161k	-162k	-162k	522k	-49k
450.soplex pds-50	450,968,408,662	-9M	-8M	25G	2M	-14M	2M	3M	2M	2M	2M
450.soplex ref	459,061,461,724	-35M	-40M	17G	6M	-35M	5M	5M	5M	5M	5M
482.sphinx3	2,827,860,491,335	18M	18M	159M	18M	18M	18M	-111M	18M	19M	18M
465.tonto	2,895,381,102,397	202M	245M	150M	223M	67M	222M	-1G	448M	224M	222M
481.wrf	4,116,952,684,945	208M	208M	-120M	224M	77M	223M	-1G	359M	226M	223M
483.xalancbmk	1,313,434,962,755	-3M	-3M	1G	-1M	7M	-1M	-1M	-1M	-173k	-1M
434.zeusmp	2,397,662,229,146	N/A	N/A	N/A	N/A	-597M	-52M	N/A	754M	-51M	-52M

Table 11. Final average retired instruction counts for SPEC CPU 2006 after taking actions described in the text. The individual machine results are shown as deltas against the global mean. Light grey indicates differences of 1 million to 10 million, medium grey differences of 10 million to 1 billion, dark grey indicates over 1 billion. Entries marked N/A are benchmarks that could not be run due to memory constraints. `zeusmp` **has a 1GB data segment size, so the DBI tools cannot run it while reserving memory for their own use.**

Benchmark	Overall Standard Deviation (mean)	Pin	Qemu	Valgrind	Pentium III 550MHz	Pentium 4 2.8GHz	Pentium D 3.46GHz	Athlon XP 1.6GHz	Phenom 9500 2.2GHz	Core Duo 1.6GHz	Core2 Q6600 2.4GHz
473.astar BigLakes	183k	1	1	0	958	38	118	583	344	550k	267
473.astar rivers	221k	1	1	0	1k	22	148	808	1k	716k	9k
410.bwaves	3G	9	1	0	N/A	47	171	N/A	2k	1M	1k
401.bzip2 chicken	72k	4	1	0	327	13	59	389	104	432k	496
401.bzip2 combined	43k	4	4	0	633	20	64	370	362	93k	111
401.bzip2 html	70k	4	1	0	777	10	87	537	791	147k	4k
401.bzip2 liberty	136k	4	1	0	663	10	53	531	324	757k	1k
401.bzip2 program	110k	1	1	0	806	8	56	311	430	226k,**6**	191
401.bzip2 source	156k	4	4	0	687	12	94	371	332	735k	135
436.cactusADM	883M	16	92	0	24k	70	17k	3k	2k	2M	767
454.calculix	93M	5	0	0	11k	67	387	2k	4k	3M	949
447.dealII	9M	4	0	0	4k	32	119	2k	920	896k	1k
416.gamess cytosine	20M	10	9	2	1k	31	109	589	950	339k	171
416.gamess h2ocu2	2M	1	2	2	1k	11	72	336	2k	503k	431
416.gamess triazolium	8M	1	2	4	5k	55	183	1k	2k	1M	421
403.gcc 166	92k	0	8	0	438	39	35	231	257	27k	85
403.gcc 200	75k	0	8	0	452	212	48	165	438	62k	251
403.gcc c-typeck	150k	5	6	0	630	269	61	290	669	29k	1k
403.gcc cp-decl	90k	0	8	0	454	154	78	203	65	11k	124
403.gcc expr	145k	5	6	0	222	164	100	295	573	14k	35
403.gcc expr2	204k	219k	11	0	2k	275	67	300	558	14k	165
403.gcc g23	251k	0	8	0	3k	136	72	803	720	61k	138
403.gcc s04	268k	0	8	0	3k	257	107	434	146	137k	360
403.gcc scilab	21k	14	2	0	196	138	38	139	1k	42k	87
445.gobmk 13x13	19k	2	8	0	331	38	42	174	220	48k	181
445.gobmk nngs	65k	8	4	0	1k	173	41	376	597	268k	127
445.gobmk score2	38k	6	2	0	521	71	24	152	342	124k	39
445.gobmk trevorc	35k	4	6	0	410	120	34	214	260	133k	95
445.gobmk trevord	50k	4	6	0	600	51	162	153	296	201k	132
459.GemsFDTD	185M	2	1	8	N/A	20	318	N/A	1k	1M	548
435.gromacs	7M	53	42	56,**6**	6k	38	362	2k	1k	2M	34k
464.h264ref forebase	91k	10	11	0	688	52	29	234	394	292k	178
464.h264ref foremain	61k	10	9	0	587	87	23	142	285	239k	124
464.h264ref sss	593k	10	9	0	4k	179	440	637	4k	1M	583
456.hmmer nph3	311k	27	20	0	1k	99	109	3k	10k	2M	747
456.hmmer retro	431k	3	10	0	3k	27	114	807	1k	1M	149
470.lbm	384k	1	6	0	5k	18	385	6k	2k	1M	1k
437.leslie3d	298k	98	96	123	6k	85	322	4k	1k	723k	24k
462.libquantum	681k	0	0	0,**6**	10k	6	287	5k	86	1M	102
429.mcf	1M	3	3	0	N/A	5	201	N/A	1k	2M	454
433.milc	10M	4	1	0	74k	34	249	2k	111	1M	104
444.namd	2M	12	0	0	2k	20	148	597	3k	284k	844
471.omnetpp	236k	9	2	0,**6**	1k	45	234	1k	481	1M	374
400.perlbench checkspam	4M	4k	4k	13M	71k	71k	101k	5M	5M	95k	93k
400.perlbench diffmail	2M	8	3	18	928	16	30	2M	2M	99k	43
400.perlbench splitmail	9M	27k	9k	8k	9k	8k	8k	12M	12M	263k	8k
453.povray	9M	1M	3M	1M	1M	1M	1M	430	914k	562k	1M
458.sjeng	256k	4	4	0	3k	1k	288	1k	3k	594k	28k
450.soplex pds-50	6M	1M	10	0	3k	405	323	1k	2k	385k	1k
450.soplex ref	15M	7M	3	0	13k	639	1k	10k	8k	153k	14k
482.sphinx3	49M	122	371	0	7k	2k	541	4k	4k	1M	614
465.tonto	631M	146	129	165	4k	124	362	2k	1k	2M	1k
481.wrf	594M	406	421	422	13k	407	342	1k	4k	3M	989
483.xalancbmk	3M	5	15	5	4k	5k	724	1k	3k	2M	891
434.zeusmp	483M	N/A	N/A	N/A	N/A	54	187	N/A	3k	2M	28k

Table 12. Final overall and per-machine standard deviations for SPEC CPU 2006. Most benchmarks are run 7 times; if fewer runs exist than the total number is listed after the variation. Light grey indicates deviation of 1k to 10k, medium grey 10k to 100k, dark grey over 100k. The slower machines are more sensitive to run-time related variation (due to number of interrupts). Variation in `perlbench` is due to stack-related issues described in Section 4.2.1. The `gcc` variation seen in Table 8 has been mitigated. There is still some `perlbench` related variation (needs investigation). `povray` also has some unexplained variation. The Core Duo machine consistently has high variation (also needs investigation).