Performance Counters and Development of SPEC CPU2006

John L. Henning
Sun Microsystems
Contact: john dot henning (at) acm dot org

Introduction

Performance counters provide the means to track detailed events that occur on a CPU chip. These events are of interest to both performance analysts and compiler developers. Counting them provides essential clues to guide performance improvement. For example, a tester who sees that a program has a high cache miss rate on a particular system may experiment with compilation options that improve prefetching. A compiler developer who sees the same thing may realize that the code generator's machine model is missing some crucial detail of behavior on that particular system.

The SPEC CPU subcommittee has also used performance counters during development of the SPEC CPU2006 suite. Here, though, the interest has not been performance improvement for a particular machine; rather, the interest is in better understanding the benchmark candidates. This article reviews some of the challenges of using performance counters, describes approaches taken by the subcommittee to deal with these challenges, and compares a set of CPU2006 vs. CPU2000 event counts for a particular system.

Difficulties

Although they provide useful information, performance counters can be difficult to interpret.

1. Perspective. Sometimes, counters record events that are more useful to chip architects than to users. For example, it is not uncommon for counters to record how many times a chip unit does an operation, without distinguishing between operations that were done intentionally vs. user-unintended speculative attempts, retries, or trap replays.

2. Overabundance. CPU chips are often implemented with more event types than the typical performance analyst would want to use. Using them has been likened to "drinking from a fire hose". Nevertheless, the analyst may be afraid to leave any of the events unexplored, for fear that the one with the innocent-sounding or unusual-sounding name is the event that actually eats the most time. ("Why would the chip architects have implemented a counter for 'bus burst enqueue overload double retry failure' if they didn't think this was important and at least somewhat likely to happen?")

3. Multiplexing. Although a chip may implement many performance events, often it is only possible to count a few of them at a time. If a performance analyst, after experience, has narrowed the set of interesting events down to, say, 14, but can only count 2 of them at a time, what should be done? Should the program be run 7 times? Or will it be sufficient to "multiplex" the events: run the program just once, and switch which are counted at regular time intervals?

4. Idiosyncrasies. Performance counters and their associated software are not as heavily used as some other features in systems, and so receive less testing. For example, a performance counter implementation was once observed that failed if a process was moved from one CPU to a another just as the 32-bit counter overflowed. Users of that system had to be aware of this problem, and had to develop workarounds. Because of such - let us call them - idiosyncrasies, the user base for performance counters has often been limited to those with a certain level of patience and expertise.

5. Comparability. During benchmark development, SPEC's primary interest in performance counters is to aid understanding of benchmark candidates. But lessons learned about a benchmark on one machine do not necessarily carry over to other machines. Branch mispredict rates vary with details of branch predictors. Cache miss rates and TLB miss rates vary with capacity, associativity and compiler optimization level. Floating point operation rates may depend on whether the machine implements a fused multiply-add (fma) -- and on whether an fma counts as one or two flops.

6. Compiler effects. If one is counting the percentage of floating point operations, the compiler optimization level is of central importance. Perhaps counter-intuitively, the tendency is that the fp op% goes up as the optimization level goes up. For example, with more optimization, the total number of instructions for 175.vpr falls from 173 billion to 65 billion (Figure 1), but the floating point instructions in the multiply pipe and add pipes show only minor variation. Presumably, to get the required answers, a set of floating point calculations must be done; but the surrounding support code, such as loads, stores, address arithmetic, pointer resolution, subroutine calls, and so forth are cut back as the compiler does more analysis. The overall time is less, but the floating point operations remain, and therefore the %fp increases.

175.vpr is from SPECint2000. Another benchmark from that set is 252.eon, whose fp op% varies up to a factor of 5x depending on the level of optimization. Perhaps the variability by optimization level for these 2 benchmarks helps to account for how they made it into the CPU2000 integer suite, despite not meeting the SPEC website-published criterion that integer benchmarks should be less than 1% floating point [1]. In any case, for CPU2006, SPEC CPU subcommittee members made a point of measuring hardware statistics at several optimization levels, with multiple compilers and hardware platforms.

7. Confidentiality. Because performance event counts are so closely tied to the details of individual system hardware and software, subcommittee members tended to view their full counter datasets as proprietary.

Figure 1: 175.vpr workload #2

	Compilation options		
Event	**-g**	**-O**	**-fast**
Instr	173B	81B	65B
FMpipe	0.1B	0.1B	0.0B
FApipe	4.9B	4.8B	5.2B
fp op%	2.9%	6.0%	8.0%

Disclaimer

This list of difficulties is not meant to paint an overly bleak picture. Performance counters are of irreplaceable importance: if you can't see which CPU component has poor performance, you can't cure the problem. Hardware implementations have improved over time. New software tools, such as SPOT [2], make counting easier. But CPUs are complex; many detailed events can be counted; it is hard to write software to present this detail clearly; it is hard to compare them across systems; and their behavior is closely tied to optimizers, so one must understand them both together.

Dealing with the challenges

From the perspective of SPEC CPU2006 development, it was not necessary for the whole subcommittee to know the solutions to each of the problems above. Rather, the subcommittee relied on the expertise of its members regarding the systems, their counter details, and their idiosyncrasies. Also, many details could be ignored, as concerns were basic: does this benchmark place a stress on the memory system? Does it place more of a stress than the typical CPU2000 benchmark? Is it an integer code? Does it have a large enough instruction footprint to cause Icache miss activity?

Although performance events are not strictly comparable across architectures, it is often straightforward to construct counts that are roughly comparable. For example, SPEC wanted reports on usage of instructions in three categories, namely floating point, load/store, and integer. All three of these were constructed for the UltraSPARC-III+ processor: (1) a flop count as the sum of the Floating Add and Floating Multiply pipes; (2) a load/store count as the sum of the L1 data cache reads and data cache writes; (3) an integer count as the total instruction count minus the flop count and the load/store count.

To minimize variation from compiler effects, members used tuning to reflect approximately the tuning for a base submission (SPECint_base2006, SPECfp_base2006).

The problem of confidentiality was probably the biggest challenge. Two tactics were used: (1) Although details of performance counter datasets were typically deemed proprietary, it was often possible to obtain permission to release summary data, or to release normalized data. For example, a member might report that benchmark candidate `997.lightweight` uses few system resources, and generally is less of a stress than the typical CPU2000 benchmark. (2) After several development versions of the new suite were built, various voting members of the SPEC CPU subcommittee released data to a trusted third party: non-voting participants from the Laboratory for Computer Architecture at the University of Texas. The University researchers prepared normalized summaries of the data, performed clustering analysis, and presented benchmark similarity dendograms such as the ones shown at [3].

If normalized data from a member showed that a benchmark used few resources, or if analysis from the university researchers showed that two benchmark candidates behaved similarly, this alone was not sufficient to exclude a candidate. But it was a factor that was considered, along with other factors such as application area, coding style, and size of user base.

Performance Counters: UltraSPARC-III+

Performance counter data was also provided to University of Texas researchers using the released V1.0 kit. The rest of this article presents the data for one such system:

- Sun-Blade 2000
- 8GB memory, 4-way interleaved
- Solaris 10
- 2x 1200 Mhz UltraSPARC-III+, each with:
  - 64KB L1 Data cache on chip
  - 32KB L1 Instruction cache on chip
  - 8MB L2 cache, unified, 2-way set associative, off chip
  - ITLB: 16 entry fully associative + 128 entry two-way set associative
  - DTLB: 16-entry fully associative + 2x 512 entry two-way set associative

The processor provides two of the larger DTLBs so that each can be dedicated to different page sizes. For the tests reported here, only the default 8KB page sizes were used.

The compiler was Sun Studio 11 with tuning `-fast`. The same system under test was also used for two historical submissions [4], which see for more description. The only differences in the tuning between [4] and the runs here is that this study does not use 4MB pages, and it adds the performance monitoring command shown in Figure 2. The `cputrack` [5] command uses multiplexing via the 9 occurrences of `-c`, and selects 18 out of the 72 possible events. [6] The information is written to the file `cputrack.all` in the

Figure 2: Performance monitoring command from the configuration file

```
monitor_wrapper = cputrack -fe -T 1 -o cputrack.out \
                    -c pic0=Instr_cnt,pic1=DC_rd_miss,sys \
                    -c pic0=Cycle_cnt,pic1=DC_wr_miss,sys \
                    -c pic0=DC_rd,pic1=DTLB_miss,sys \
                    -c pic0=IC_ref,pic1=EC_misses,sys \
                    -c pic0=DC_wr,pic1=FM_pipe_completion,sys \
                    -c pic0=EC_ref,pic1=IC_miss,sys \
                    -c pic0=FA_pipe_completion,pic1=ITLB_miss,sys \
                    -c pic0=IU_Stat_Br_count_taken,pic1=IU_Stat_Br_count_untaken,sys \
                    -c pic0=IU_Stat_Br_miss_taken,pic1=IU_Stat_Br_miss_untaken,sys \
                  $command;\
                  cat cputrack.out >> cputrack.all
```

run directory, which is post-processed to create the metrics shown in Figure 3. It is hoped that their meaning, and their derivation, will be apparent given the definitions in [6] plus the notes above mapping instruction types to counters. However, the reader may wonder why there are 2 metrics for %L2 misses. The reason is that for this processor, the L2 reference count includes speculative accesses that turn out to be resolved in the L1 cache. Therefore, it is more meaningful to compare the number of L2 misses to the number of L1 misses (`DC_rd_miss + DC_wr_miss + IC_miss`).

Figure 4 has event counts for the CPU2006 and CPU2000 integer benchmarks. The most obvious difference between the suites is for TLB activity, with remarkably more DTLB activity, and noticeable ITLB activity.

The activity here can be considered in the light of what is known of source code size [7], profile [8] and memory activity[9]. In some cases, the data make intuitive sense. For example, the very high "CWSS" (Core Working Set Size, in the terminology of [9]) for 429.mcf matches its high DTLB miss rate, and 483.xalancbmk's large number of source lines matches its high ITLB miss rate. Another (possibly) intuitive point may be that three of the four benchmarks with >15 mispredicts/Kins are derived from game-playing codes, which presumably are finding paths through possible scenarios of play; and the worst benchmark for mispredicts is, explicitly, a trip/path planning program.

Other statistics here could lead to interesting further investigation. For example: why does 471.omnetpp show a large DTLB miss rate even though its CWSS is small? Why does 403.gcc have no more L2 misses than 176.gcc, despite a much higher DTLB miss rate?

Figure 3: Metrics

time	seconds, as reported by the SPEC tools
int op%	percentage integer operations
fp op%	percentage floating point operations
mem op%	percentage load/store operations
branch/Kins	branches per 1000 instructions
mispre/Kins	mispredicted branches per 1000 instr
ICmiss/Kins	L1 instr cache misses per 1000 instr
DCmiss/Kins	L1 data cache misses per 1000 instr
L2miss/Kins	L2 cache misses per 1000 instructions
L2miss %L2acc	L2 miss rate expressed as % of L2 accesses
L2miss %L1miss	L2 miss rate expressed as % of L1 misses
ITLBm/Minst	Instr TLB misses per million instr
DTLBm/Minst	Data TLB misses per million instr

A key difference for CPU2006 vs. CPU2000 is shown in the fp op% column. The only report over 1% in the new suite is for 464.h264ref, but this number is not actually due to floating point calculations. As can be seen at [8], this program spends substantial time in `memcpy` – which uses floating point registers for 64-byte "block load/store" instructions, but does not actually do fp calculations. The fp content formerly seen in CINT2000 has, effectively, been eliminated for CINT2006.

Finally, we can note that there are 3 new programs with worse branch behavior than any of the old programs. Nevertheless, the median number of mispredicts is on the order of 10 in both suites (with CPU2000 actually slightly higher).

Figure 4: Counters for the Integer Benchmarks

		operations			branches		L1 misses		L2miss			TLB	
benchmark	time	int op%	fp op%	mem op%	branch /Kins	mispre /Kins	ICmiss /Kins	DCmiss /Kins	/Kins	%L2 acc	%L1 miss	ITLBm /Minst	DTLBm /Minst
CINT2006													
400.perlbench	2395	72	0	27	165	8	7.5	10	0	0	2	23	52
401.bzip2	2656	64	0	35	151	8	0.2	24	1	0	2	0	4
403.gcc	2248	68	0	30	190	7	4.4	101	3	2	3	44	257
429.mcf	2288	62	0	37	182	27	0.4	91	23	7	25	0	3028
445.gobmk	2629	70	0	29	133	23	19.3	13	1	0	2	1	9
456.hmmer	2075	62	0	37	34	1	0.1	51	0	0	1	0	1
458.sjeng	2758	78	0	21	159	17	5.9	5	0	0	4	0	296
462.libquantum	9775	79	0	20	220	14	0.3	73	30	15	41	0	271
464.h264ref	4464	51	5	42	56	3	1.9	65	6	2	9	0	7
471.omnetpp	2416	67	1	31	163	11	5	48	9	3	17	0	2404
473.astar	2332	65	0	34	137	19	0.3	31	5	2	14	0	2161
483.xalancbmk	1720	72	0	26	211	8	8.7	25	2	1	7	661	548
CINT2000													
164.gzip	308	73	0	26	146	9	0.1	26	0	0	1	0	5
175.vpr	278	57	6	35	108	13	1.1	23	2	1	6	0	141
176.gcc	180	64	2	32	150	10	6	81	3	1	3	3	9
181.mcf	291	64	0	35	195	18	0.3	165	16	4	10	0	1620
186.crafty	158	75	0	24	100	13	6.7	3	0	0	1	0	0
197.parser	387	71	0	28	157	13	0.4	28	1	0	2	0	11
252.eon	227	41	10	47	60	4	2	6	0	0	0	0	0
254.gap	274	66	0	33	133	7	1.1	38	3	1	8	0	35
253.perlbmk	320	73	0	26	137	7	7.8	23	0	0	1	3	69
255.vortex	329	70	0	29	151	6	8	12	1	0	4	0	295
256.bzip2	281	69	0	30	120	12	3.1	20	0	0	2	0	3
300.twolf	485	67	0	31	108	12	0.2	61	0	0	1	0	0

Figure 5: Counters for the FP benchmarks

		operations			branches		L1 misses		L2miss			TLB	
benchmark	time	int op%	fp op%	mem op%	branch /Kins	mispre /Kins	ICmiss /Kins	DCmiss /Kins	/Kins	%L2 acc	%L1 miss	ITLBm /Minst	DTLBm /Minst
CFP2006													
410.bwaves	3256	31	24	44	33	1	0.1	41	17	4	41	0	132
416.gamess	4987	45	17	36	63	2	1.7	4	0	0	1	0	0
433.milc	4482	9	42	47	13	1	0.3	87	24	6	27	0	1010
434.zeusmp	2625	18	45	36	18	1	0.1	45	5	1	11	0	448
435.gromacs	2035	14	48	36	30	2	0.1	14	0	0	3	0	3
436.cactusADM	8892	29	28	41	67	6	0.8	70	3	1	4	0	16191
437.leslie3d	2354	13	40	46	19	1	0.1	86	16	4	19	0	149
444.namd	2055	27	42	29	53	2	0	5	0	0	2	0	2
447.dealII	2657	62	7	29	102	4	0.3	16	2	1	15	5	159
450.soplex	3539	51	14	34	124	9	2.4	76	19	6	24	20	465
453.povray	1209	47	19	32	104	11	8.8	7	0	0	0	104	0
454.calculix	2200	13	54	32	28	1	0.2	11	1	0	4	0	5
459.GemsFDTD	3380	11	44	43	17	1	0.3	48	22	6	46	0	1202
465.tonto	2520	39	20	39	55	3	2.5	41	1	0	1	14	5
470.lbm	4898	9	66	24	12	0	0.1	92	25	7	27	0	237
481.wrf	2551	18	41	40	28	1	1.6	63	6	2	10	10	61
482.sphinx3	2715	30	37	32	57	3	0.3	21	2	1	11	0	150
CFP2000													
168.wupwise	345	44	27	28	84	4	0.1	19	3	1	16	0	37
171.swim	134	5	52	41	13	0	0	102	2	1	2	0	15
172.mgrid	298	11	51	36	8	0	0.1	52	11	3	21	0	145
173.applu	403	5	50	44	8	0	0.1	71	7	2	10	0	60
177.mesa	232	53	14	31	77	7	2.2	21	0	0	1	0	3
178.galgel	136	17	42	39	16	0	0	38	0	0	1	0	38
179.art	252	11	27	61	45	0	0.1	261	1	0	0	0	2
183.equake	541	18	35	46	22	1	0.1	60	23	5	38	0	182
187.facerec	119	33	26	39	50	2	0.2	92	4	1	5	0	76
188.ammp	474	31	35	33	91	4	0.1	31	2	1	6	0	285
189.lucas	602	45	26	28	88	2	0.5	144	12	6	8	0	22071
191.fma3d	452	28	30	40	38	2	0.7	34	8	2	23	4	94
200.sixtrack	262	14	62	23	25	1	0.2	9	0	0	0	0	1
301.apsi	383	25	28	45	33	1	0.1	47	3	1	7	0	50

Figure 5 has the counters for the floating point benchmarks. There are more benchmarks with DTLB activity than in CPU2000, though the worst CPU2000 offender, 189.lucas, is not matched by any of the CPU2006 benchmarks. For ITLB misses, only 453.povray shows significant activity; given its large number of source lines [7] and relatively flat profile [8], this makes sense. The L2 miss rates are higher for CPU2006, but L1 miss rates and branch mispredict rates are not strikingly different between the suites. As with CPU2000, the floating point benchmarks generally have fewer branches than the integer benchmark – unless the language is C++. Three of the four C++ benchmarks have >100 branches/Kins.

Acknowledgments

Thank you to Darryl Gove and to Miriam Blatt for advice on usage and interpretation of UltraSPARC counters.

References

[1] http://www.spec.org/spec/glossary/#integer
[2] http://cooltools.sunsource.net/spot/
[3] A.Phansalkar, A. Joshi and L. John, "Subsetting the SPEC CPU2006 Benchmark Suite", Computer Architecture News, vol. 35, no. 1, March 2007.
[4] See the two result submissions for the Sun Blade 2000 at www.spec.org/cpu2006/results/res2006q3
[5] The manpage for `cputrack` is available under User Commands at http://docs.sun.com/app/docs/coll/40.10
[6] UltraSPARC III Cu User's Manual, Version 2.2.1, January 2004, Chapter 14.
[7] J.Henning, SPEC CPU Suite Growth: An Historical Perspective", Computer Architecture News, vol. 35, no. 1, March 2007.
[8] R.Weicker and J.Henning, "Subroutine Profiling Results for the CPU2006 Benchmarks", Computer Architecture News, Vol. 35, no. 1, March 2007.
[9] D.Gove, "CPU2006 Working Set Size", Computer Architecture News, vol. 35, no. 1, March 2007.