SPEC CPU2000: Measuring CPU Performance in the New Millennium

The SPEC consortium's mission is to develop technically credible and objective benchmarks so that both computer designers and purchasers can make decisions on the basis of realistic workloads.

John L. Henning
Compaq Computer Corp.

Computers perennially become more powerful, as do the software applications that run on them, and it seems almost human nature to want the biggest and fastest toy we can afford. But how do you know if it is? Even if your application never does any I/O, it's not just the speed of the CPU that dictates performance—cache, main memory, and compilers also play a role—and different software applications have differing performance requirements. And whom do you trust to provide this information?

The Standard Performance Evaluation Corporation (SPEC) is a nonprofit consortium whose members include hardware vendors, software vendors, universities, customers, and consultants. SPEC's mission is to develop technically credible and objective component- and system-level benchmarks for multiple operating systems and environments, including high-performance numeric computing, Web servers, and graphical subsystems. Members agree on benchmark suites that are derived from real-world applications so that both computer designers and computer purchasers can make decisions on the basis of realistic workloads. By license agreement, members agree to run and report results as specified by each benchmark suite.

On June 30, 2000, SPEC retired the CPU95 benchmark suite. Its replacement is CPU2000, a new CPU benchmark suite with 19 applications that have never before been in a SPEC CPU suite. By continually evolving these benchmarks, SPEC aims to keep pace with the breakneck speed of technological innovation. But how does SPEC develop a benchmark suite and what do these benchmarks do? We can get a sense of the process by looking over SPEC's shoulders on one specific day in the benchmark development process.

SPEC BENCHATHON

It is 6 a.m. on a cool Thursday morning in February 1999. A Compaq employee prepares to shut off the alarm at SPEC headquarters in Manassas, Virginia, and start the day. But he finds that the alarm is already off, because two IBM employees are still inside from the night before. A weeklong SPEC ritual is in progress: A subcommittee is in town for a "benchathon," and technical activity is happening at all hours. The Compaq employee goes to the back room, which is about 85 degrees Fahrenheit (30 degrees Celsius), thanks to its collection of workstations by Sun, HP, Siemens, Intel, SGI, Compaq, and IBM. He opens the window to the cool air, and opens windows on his workstations to review the results of running Kit 60 of what will eventually be known as SPEC CPU2000. (SPEC will release it 10 months later, after building Kit 98.)

The primary goal at this stage is portability for the candidate benchmarks. As other subcommittee members arrive, the Compaq employee updates the Porter's Progress spreadsheet in preparation for today's meeting. As of this Thursday of the benchathon week, the spreadsheet shows test results for

- 34 candidate benchmarks,
- 18 platforms from seven hardware vendors (11

0018-9162/00/$10.00 © 2000 IEEE

Comparable Work

In order to provide a level playing field for competition, SPEC wants to ensure that comparable work is done across all tested platforms. But consider the following code fragment, which is extracted (without ellipses) from a much larger section of 187.facerec:

```
If ((NewSim – OldSim) > SimThresh) Then
  CoordX (IX, IY) = NewX
  CoordY (IX, IY) = NewY
  Hops = Hops + 1
  Improved = .TRUE.
EndIf
Sweeps = Sweeps + 1
  If ((.NOT. Improved) .OR.
    (Sweeps >= Params%Match%MaxSweeps))
Exit
```

The loop exit depends on a floating-point comparison indicating an improvement in face similarity. The comparison is affected by differences in the order and accuracy of floating-point operations as implemented by different compilers and platforms.

If two systems correctly recognize a face but do a different number of iterations, are they doing the same work? Although one could argue that in some sense the work is equivalent—one platform just takes a different code path to get the answer—SPEC has traditionally preferred similar code paths. But SPEC would also prefer to avoid artificially recoding the author's original algorithm—for example, by changing the above loop to use a fixed number of iterations.

Thus, 187.facerec faced a dilemma in March 1999. The solution to this dilemma was to use a new feature of the SPEC CPU2000 tool suite—namely, file-by-file validation tolerances. SPEC modified facerec to output detailed data about the number of iterations for individual faces to one file and a summary of total iterations to another file. The two files are validated as follows:

	Detail	Summary
reltol	0.2	0.001
abstol	5	2.e – 7
skiptol	4	0

That is, for individual faces, the tools will accept the reported number of iterations if it matches the expected number of iterations within 20 percent (reltol = 0.2), or they will accept the reported number of iterations if it is no more than five iterations different from the expected number of iterations (abstol = 5). If both of these checks fail, then up to four times the tools will accept any difference whatsoever (skiptol = 4). But for the overall run, the number of iterations must match within one-tenth of 1 percent (reltol = .001), and all iterations must be checked (skiptol = 0). Therefore, two platforms executing 187.facerec may in fact do different amounts of work in the task of matching a single face, but they must do substantially similar amounts of work in the task of matching all the faces.

with 32-bit and seven with 64-bit address spaces), and
- 11 versions of Unix (three of them Linux) and two versions of Windows NT.

SPEC tests a wide variety of systems, although there are basically only two operating system families represented at this benchathon: Unix and NT. SPEC relies on the efforts of those who choose to participate and cannot mandate participation for any platform.

Unfortunately, only 19 of 34 candidate benchmarks are successful on all platforms. Portability is difficult because SPEC CPU suites are not abstracted loop kernels but are programs for real-world problems, with real-world portability challenges. Portability challenges can be roughly categorized by source code language.

Fortran. The Fortran-77 applications are the easiest to port because the language contains relatively few machine-dependent features, and the ANSI standard is more than 20 years old. Nevertheless, there are issues. For example, a particular application has 47,134 lines of code, 123 source files, and hard-to-debug wrong answers when optimization is enabled for one SPEC member's compiler. Later, the compiler will be blamed, the application will be exonerated, and the 200.sixtrack benchmark will ship with CPU2000.

Several F77 applications allocate 200 megabytes of memory. When the allocation is static, a member complains that executables take too much disk space; when it is dynamic, another member's OS stack limits are exceeded. Eventually, SPEC will choose dynamic allocation but will allow static allocation for those who need it.

The Fortran-90 applications are more difficult to port because F90 implementations are less common and less mature than F77 implementations. One application author is a "language lawyer:" He aggressively uses as many features as he can from the Fortran-90 standard. In February 1999, only three platforms succeed on his application. Later, it will work on all tested F90 compilers, and the author of 187.facerec will be proud that he uncovered bugs in many of these compilers. But facerec itself will also be adjusted (see the "Comparable Work" sidebar).

C and C++. The C applications are harder to port than the ones in either Fortran dialect. The portability issues are not uncommon: How big is a long? How big is a pointer? Does this platform implement `calloc`? Is it little endian or big endian? But the applications take differing approaches to these issues, and SPEC has its own requirements. For example, SPEC prefers the ANSI standard, but some programs still have widespread K&R vestiges. Eventually, the vestiges that actually cause problems will be prioritized for removal, and most of the rest will remain unchanged. Some programs use a tailoring process, often called *configure*, but SPEC prefers to minimize source code differences. SPEC avoids configure, and looks for other portability methods, such as additional #ifdef directives.

Table 1. February 1999 benchathon results.

	19 Feb	26 Feb
Compile errors	22	2
Runtime errors	18	6
Validation errors	60	41
Total	100	49

The C++ applications are the biggest challenge. The standard is new, runtime functions (class libraries) are diverse and hard to compare, and SPEC received only two C++ applications. One of these has already been voted out: It worked with gnu g++ but proved impractical to port to ANSI C++. The other—252.eon—is far more portable, works on all 17 C++ compilers tested in February 1999, and will later ship with CPU2000 (in December 1999).

As Table 1 shows, in this particular week in February 1999, the benchathon solves just over half the outstanding problems. The point of a benchathon is to gather as many as possible of the project leaders, platforms, and benchmarks in one place and have them work collectively to resolve technical issues involving multiple stakeholders: At a benchathon, it is common to see employees from different companies looking at the same screen, helping each other.

Project leaders and the tool master

A project leader shepherds each candidate benchmark and owns resolution of all portability problems. One ambitious soul has 10 benchmarks, though the subcommittee will later lighten this load.

Another project leader has only three benchmarks, but each one has challenges. One benchmark is a simulator that never seems to get the same answers on different systems and will later be dropped. Another—186.crafty—requires 64–bit integers, which are not standardized in C89, but all tested compilers will prove to have some means of specifying them. The third benchmark has mysterious validation errors, and members closely examine one another's results. The application constructs a pixel map with different colored areas. For a few border pixels, the color choice is sensitive to minor differences of floating-point results, and different platforms make different decisions that are nearly invisible to the eye. After discussion, the members agree that for the purposes of this application the differences are irrelevant.

One of the subcommittee members serves as tool master and writes tools to control benchmark compilation, execution, and validation. This tool master suggests a new feature—skiptol—to allow a predefined number of different answers. The feature is implemented during this week in February, and 177.mesa will be released in CPU2000, with skiptol allowing six differences out of 3.9 million numbers printed.

BENCHMARK SELECTION

Porting is a clearly technical activity, with a reasonably simple completeness criterion: Does the benchmark work? By contrast, benchmark selection involves multiple, sometimes conflicting criteria, which may lack simple answers. The CPU2000 benchmarks (Table 2) were selected from a much larger collection of candidates, submitted by members and by the general public through a search process announced at SPEC's Web site. Ultimately, benchmarks are selected by vote, and members may weigh the criteria differently. Often, a candidate benchmark may attract "yes" votes if it

- has many users,
- exercises significant hardware resources,
- solves an interesting technical problem,
- has generated results that are published in recognized journals, or
- adds variety to the suite.

A candidate benchmark may attract "no" votes if it

- cannot be ported in a reasonable time,
- does too much I/O and therefore does not remain compute-bound,
- has previously appeared in a SPEC CPU suite and the workload is largely unchanged,
- appears to be a code fragment rather than a complete application,
- is redundant with other candidates, or
- appears to do different work on different platforms.

Objective criteria

Since SPEC wants to produce a suite with technical credibility, access to objective technical data is highly desirable. But SPEC members are often employees of companies that compete with each other, and vendor confidentiality limits what can be said, for both business and legal reasons.

The solution to this problem for CPU2000 was that all members simultaneously provided some amount of objective data, and the subcommittee kept this data confidential, thereby reducing management concerns. The process worked well: SPEC gained objective data about candidate benchmarks' I/O, cache and main-memory behavior, floating-point operation mixes, branches, code profiles, and code coverage.

Three subjective criteria

One subjective criterion is confidence in benchmark maintainability. Some candidate benchmarks have errors that are difficult to diagnose. Some have regres-

Table 2. CPU2000 integer and floating-point benchmark suite.

Benchmark	Language	KLOC	Resident size (Mbytes)	Virtual size (Mbytes)	Description
SPECint2000					
164.gzip	C	7.6	181	200	Compression
175.vpr	C	13.6	50	55.2	FPGA circuit placement and routing
176.gcc	C	193.0	155	158	C programming language compiler
181.mcf	C	1.9	190	192	Combinatorial optimization
186.crafty	C	20.7	2.1	4.2	Game playing: Chess
197.parser	C	10.3	37	62.5	Word processing
252.eon	C++	34.2	0.7	3.3	Computer visualization
253.perlbmk	C	79.2	146	159	Perl programming language
254.gap	C	62.5	193	196	Group theory, interpreter
255.vortex	C	54.3	72	81	Object-oriented database
256.bzip2	C	3.9	185	200	Compression
300.twolf	C	19.2	1.9	4.1	Place and route simulator
SPECfp2000					
168.wupwise	F77	1.8	176	177	Physics: Quantum chromodynamics
171.swim	F77	0.4	191	192	Shallow water modeling
172.mgrid	F77	0.5	56	56.7	Multigrid solver: 3D potential field
173.applu	F77	7.9	181	191	Partial differential equations
177.mesa	C	81.8	9.5	24.7	3D graphics library
178.galgel	F90	14.1	63	155	Computational fluid dynamics
179.art	C	1.2	3.7	5.9	Image recognition/neural networks
183.equake	C	1.2	49	51.1	Seismic wave propagation simulation
187.facerec	F90	2.4	16	18.5	Image processing: Face recognition
188.ammp	C	12.9	26	30	Computational chemistry
189.lucas	F90	2.8	142	143	Number theory/primality testing
191.fma3d	F90	59.8	103	105	Finite-element crash simulation
200.sixtrack	F77	47.1	26	59.8	Nuclear physics accelerator design
301.apsi	F77	6.4	191	192	Meteorology: Pollutant distribution

sions: A problem is solved and then mysteriously reappears. Some benchmarks are repeatedly submitted with errors that are easy to figure out but require subcommittee time. Although factors such as these may not be decisive, they contribute to the confidence level. SPEC desires real applications, with real, nontrivial source code, but that source code needs to be perceived by the committee as maintainable.

Another subjective consideration is transparency. A transparent benchmark becomes stable quickly enough so that members have time to analyze it. It may have complex code, but it does not give the impression of being intentionally misleading. It has a workload that can be sensibly described both in ordinary English and in the technical language of the application domain. Reasoned arguments support it.

A final subjective criterion is vendor interest. If a vendor employs a subcommittee member, is there a temptation to vote according to competitive effect? Yes, the temptation exists, but two factors greatly reduce its influence:

- Generally, members do not know the performance of candidate benchmarks on competitors' hardware. Even with an unlimited budget and unlimited time, it would be impossible to buy the competitors' unreleased next-generation hardware and unreleased compilers. You cannot really know the competitive position of a benchmark until after the suite is released; it's better to just vote on the basis of technical merit.
- It is nearly impossible to argue in the subcommittee, "you should vote for 999.favorite because it helps my company." Blatant efforts along these lines would backfire; subtle attempts may raise concerns about transparency.

Of course, SPEC members who are vendor employees keep their employer's interests in mind. For example, an employee of a company that makes big-endian Unix systems makes sure that the playing field is not tilted in favor of little-endian NT systems. Arguments to level the playing field are always welcome and quickly attract support. But attempts to tilt the playing field just don't work.

PERFORMANCE RESULTS

The CPU2000 suite should tell us something about performance that we cannot learn simply by knowing a chip's clock speed. If performance depended only on megahertz, there would be no performance differences between two identical chips, and a faster chip would always win.

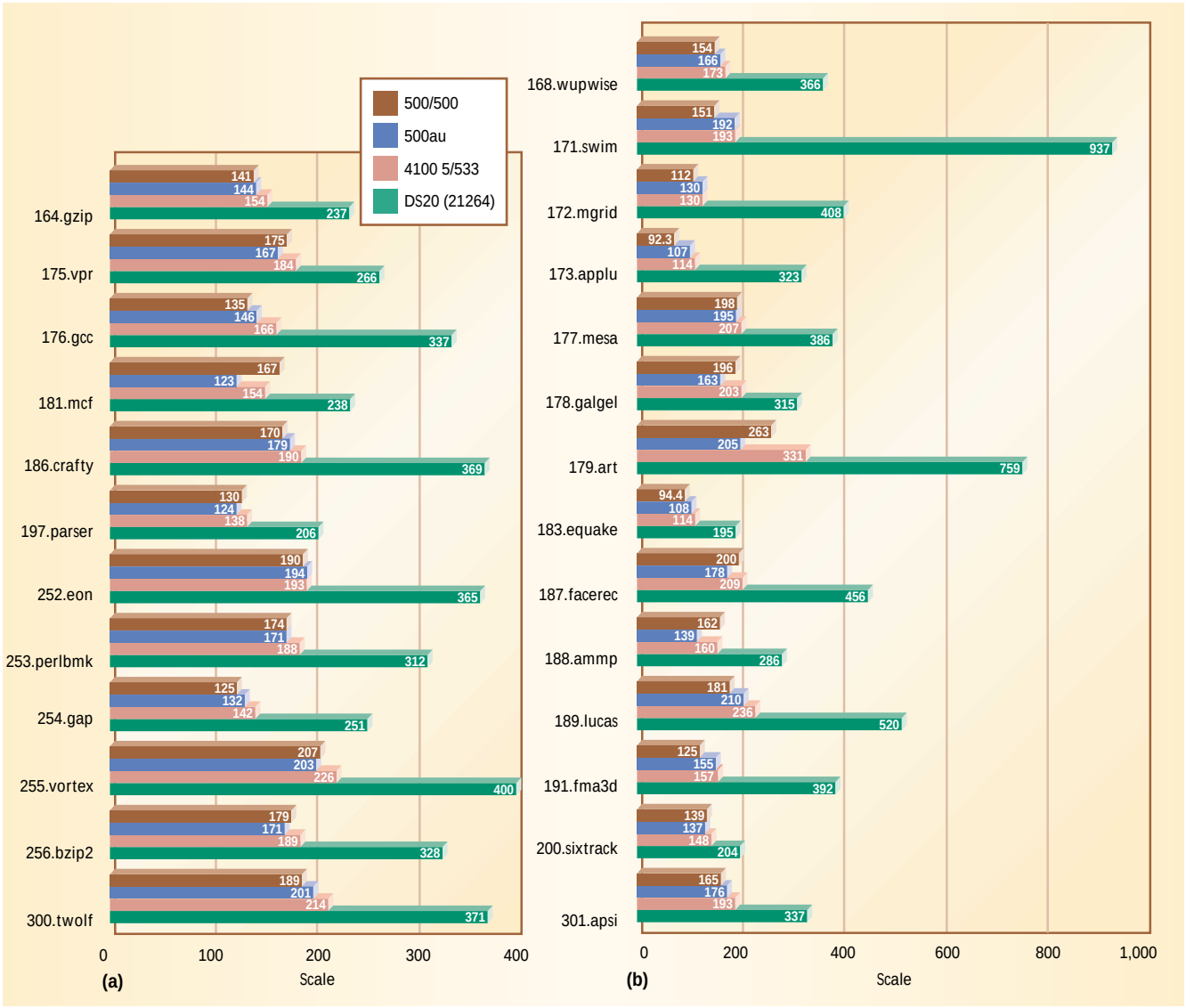

Figure 1. Integer (a) and floating-point (b) performance for three Alpha 21164 systems—the 500-MHz AlphaStation 500/500, the 500-MHz Personal Workstation 500au, and the 533-MHz AlphaServer 4100 5/533—and the Alpha 21264-based 500-MHz AlphaServer DS20 6/500.

Figure 1 shows integer and floating-point performance for four differently configured systems using the Alpha 21164 and 21264 chips. Performance is stated relative to a reference machine, a 300-MHz Sun Ultra5_10, which gets a score of 100.

Let's focus first on the three 21164 systems. As you can see, performance depends on more than megahertz:

- Performance of the 26 benchmarks on the 21164 systems ranges from 92.3 to 331.
- The 500-MHz 21164 systems (AlphaStation 500/500 and Personal Workstation 500au) differ by more than 5 percent on 17 of the 26 benchmarks.
- The 533-MHz AlphaServer 4100 5/533—with a 7 percent megahertz advantage—wins by more than 10 percent three times, by less than 3 percent three times, and loses to a 500-MHz system three times.

To understand these results, we need to consider the differences between the systems, particularly the memory hierarchies shown in Table 3. Notice that each 21164 memory system has some characteristic superior to the others:

- AlphaStation 500/500 has the largest L3 cache;
- Personal Workstation 500au has the best cache latency and main-memory latency, especially when measured in processor cycles; and
- AlphaServer 4100 5/533 has the highest main-memory bandwidth.

Cache effects. The 4100 5/533 wins for most benchmarks, with the largest margin on 179.art. In Table 2, we see that art's resident size almost fits in a 4-Mbyte cache, but not in a 2-Mbyte cache. The miss rates in Table 4 confirm this. The 4100 5/533 wins because 179.art fits in its cache, which is faster than the 500/500 cache.

A benchmark with especially interesting cache effects is 181.mcf, where the 500/500 outperforms the usually

Table 3. CPU and memory characteristics.

CPU	Alpha 21164			Alpha 21264
System	500/500	500au	4100 5/533	DS20
CPU MHz	500	500	**533**	500
L1 cache on chip	8 Kbytes (instruction) + 8 Kbytes (data)			64 Kbytes (instruction) + 64 Kbytes (data)
L2 cache on chip	96 Kbytes			None
Off-chip cache				
Size (Mbytes)	**8**	2	4	4
Latency (ns)	82	**58**	62	32
Latency (processor cycles)	41	**29**	33	16
Main memory				
Latency (ns)	341	**247**	248	184
Latency (processor cycles)	171	**124**	132	92
Bandwidth (Mbytes/s)	200	238	**272**	1,232

Latencies are measured for dependent loads (also called *pointer chasing*) in a stride of 128 bytes; lower is better. Bandwidth is measured using the McCalpin Stream triad,[1] which calculates bandwidth in millions of bytes per second; higher is better. For the three systems that use the Alpha 21164, bold type indicates the best value in each row.

faster 4100 5/533. According to a profile from DCPI (Compaq Continuous Profiling Infrastructure, formerly known as Digital Continuous Profiling Infrastructure),[2] the 4100 5/533 spends substantial time on code such as the following, where the subtract instruction requires on average 117 cycles per issue (CPI):

```
ldq a1, 64(s1)
... [two instructions elided]...
ldq a2, 88(t6)
ldq v0, 32(a1)
... [one instruction elided] ...
subq a2, v0, v0
```

This code loads registers `a1` and `a2` from memory, loads register `v0` from 32 bytes past the address contained in `a1`, subtracts `v0` from `a2`, and leaves the result in `v0`.

The subtract instruction, therefore, depends on the completion of three loads in the immediately preceding six instructions. The exact breakdown of time would require a low-level performance simulation, but we can sketch a plausible scenario why 117 cycles are required. The subtract typically waits for the dependent load of `v0`, finding it in the L3 cache 15 percent of the time (33 cycles) and in main memory 85 percent of the time (132 cycles). DCPI shows 253 million L3 cache misses for the subtract instruction. By contrast, the subtract shows a CPI of 95 on the 500/500 (much less than its 171-cycle trip to memory) and only 161 million L3 misses. For 181.mcf, an 8-Mbyte cache is helpful.

The 500au is victorious only once, on 252.eon, and its margin of victory is so small as to introduce doubt as to whether or not run-to-run variation may be larger than the seeming victory. But it is plausible that the faster cache contributes to eon's good performance on the 500au, because eon has the smallest resident size.

Main memory. Cache effects are important, but many scientific applications stride through large arrays and thus depend on main-memory bandwidth. The 4100 5/533 has the best main-memory bandwidth of the three 21164 systems, so its bandwidth advantage is a likely contributor to its victories. As Table 4 shows, three of the top five SPECfp2000 miss generators do not significantly improve when the cache size increases from two to four megabytes—namely, 171.swim, 189.lucas, and 173.applu. Therefore, we would hope to see an improvement from the 500au to the 4100 5/533 that is due to memory bandwidth and not cache effects. In fact, the 14 percent bandwidth improvement appears to have provided improvements of 1 percent, 12 percent, and 7 percent for these three benchmarks, respectively.

Table 4. L3 cache misses per 1,000 issues. Top 5 SPECfp2000.

Benchmark	2 Mbytes	4 Mbytes	8 Mbytes
179.art	29.1	0.5	0.4
171.swim	23.9	23.7	23.6
183.equake	23.6	22.2	21.0
189.lucas	19.6	19.3	18.9
173.applu	14.2	14.0	13.0

Although it's not clear why swim has such a small benefit, we can form a reasonable hypothesis as to why lucas has a large benefit: The 21164 can have only two simultaneous outstanding memory-load requests. Therefore, the 4100 5/533 bandwidth advantage will be most evident for applications where loads are nicely spread through the code rather than being bunched together. When they are spread, the processor is more likely to overlap computation with memory activity; when they are bunched, the processor will stall after the third load so that overlap is reduced.

Lucas was hand-unrolled by its author, long before submission to SPEC. DCPI shows that the routine fft_square has the bulk of the L3 cache misses

SPEC/Academia

SPEC encourages research and academic usage of the new CPU2000 suite, as described in section 4.5 of the run rules (http://www.spec.org/cpu2000/docs/runrules.txt). During the year 2000, SPEC is offering two incentives to the academic community:

- SPEC has reduced the cost of membership for associate members, as described at http://www.spec.org/news/y2kspecial.html.
- Universities can obtain a free copy of SPEC CPU2000 (or certain other SPEC products) through 31 December 2000, by following the instructions at the above URL.

(approximately 2.6 billion), but no single instruction has more than 86 million of these. If we extract the virtual addresses of the 40 instructions with the highest miss rates (which together comprise 84 percent of the fft_square misses), the average distance between these instructions is 823 bytes, or 205 instructions. In short, 189.lucas has its memory activity nicely spread.

Up to this point, we have considered three systems at about 500 megahertz, using the same CPU with differing memory hierarchies, and have observed differences in how the CPU2000 benchmarks respond. If you want good performance, it's not just megahertz that matters.

Processor performance

Let's change the processor but stay at 500 megahertz. Figure 1 also compares performance results for the three 21164 500-MHz systems against the AlphaServer DS20 6/500 using the new Alpha 21264 processor. The DS20 provides an entirely new memory system as shown in Table 3. Cache latency improves by a factor of 1.8, memory latency by 1.3, and bandwidth by 4.5. The memory bandwidth is better both because of the DS20 memory system and because of the 21264 CPU, which supports up to eight outstanding loads and eight outstanding writes.

Although we have held megahertz constant, performance has changed radically. The three integer benchmarks with the greatest improvements are 252.eon (1.89 times as fast as the 4100 5/533), 186.crafty (1.94), and 176.gcc (2.03). Eon and crafty have small memory footprints; so it is likely that they have benefited from the improved cache latency and from the CPU itself, which provides out-of-order issue (the 21164 has in-order issue).

Why does gcc show the greatest improvement among the integer benchmarks, and why does the CPU2000 version gcc show a much greater improvement than the CPU95 version (2.03 versus 1.61)?[3] Put simply, it's the workload. For CPU95, gcc does minimal optimization; in the 79 seconds it runs on the DS20, it compiles 56 programs and never uses more than 47 megabytes of virtual memory. The CPU2000 version compiles much larger programs, does aggressive inlining, and then does extensive optimization; it runs for 327 seconds on the DS20 while compiling only 5 programs and consuming 156 megabytes of virtual memory. The DS20's improved memory bandwidth contributes to the improved performance of CPU2000 gcc, as the compiler shuffles through its tuples in search of the best optimizations.

On the floating-point side, we would expect 171.swim, 189.lucas, and 173.applu (see Table 4) to benefit from main-memory bandwidth improvements, and the evidence seems to confirm that expectation for two of three benchmarks when considering the 21164 systems. The 21264-based system, with 4.5 times the memory bandwidth, appears to have helped all three benchmarks, with performance improvements of 4.9, 2.2, and 2.8 times, respectively.

Despite its high miss rate, 183.equake improves by a factor of only 1.7. According to its description in the CPU2000 documentation, it uses "an unstructured mesh that locally resolves wavelengths, using a finite element method." According to DCPI, the cache misses are concentrated in a routine that does a matrix vector product using indirect addressing. In short, equake is latency bound rather than bandwidth bound.

Compiler effects. All results in this article use a single set of compilers and "base" tuning: no more than four optimization switches, with the same switches applied to all benchmarks of a given language in a suite. Different tuning, or a different set of compilers, would produce different results. This is intentional. The CPU suites are designed to test three things: the CPU, the memory hierarchy, and the compilers.

Space does not permit exploration of the important effects from compilers, but we can briefly mention two items:

- With more than 400,000 lines of new code in the CPU2000 benchmark suites, compilers need to be robust. For example, if you turn on "–fast" for SPECfp_base2000, it must produce correct answers and acceptable performance for all 10 Fortran benchmarks.
- Some benchmarks are sensitive to compiler strategies such as unrolling methods or loop transformations. For example, when 178.galgel is built using peak rules, which permit individualized tuning, its performance on the DS20 6/500 is 70 percent faster than with base tuning.

As compilers continue to evolve and improve, it is likely that CPU2000 benchmark performance will change. SPEC encourages study of the benchmarks by compiler developers, but optimizations must be implemented in ways that benefit general applications, not just the applications in the SPEC CPU suite.

This discussion of benchmarks barely scratches the surface. SPEC encourages industry and academia to undertake further study of the CPU2000 benchmarks because we believe these research efforts will result in performance improvements that will be of general benefit to end users.

With CPU2000 out the door, it is already time to begin thinking about CPU200x, the successor to CPU2000. SPEC encourages those who would like to contribute to the next suite to come forward. Universities, consultants, compiler vendors, hardware vendors, ISVs, and end users are all welcome to become members of SPEC (see the "SPEC/Academia" sidebar). Please seriously consider how your contributions of code, time, or effort might help to shape better benchmarks and, therefore, help the computer industry to design and validate better computer systems. ✷

References

1. J. McCalpin, "Stream: Sustainable Memory Bandwidth in High Performance Computers;" http://www.cs.virginia.edu/stream/.
2. J. Anderson et al., "Continuous Profiling: Where Have All the Cycles Gone?" *Proc. 16th ACM Symp. Operating Systems Principles*, http://www.unix.digital.com/dcpi/publications.htm.
3. CPU95 gcc results, http://www.spec.org/cpu95/results/res97q4/cpu95-971027-02208.asc; and http://www.spec.org/cpu95/results/res99q1/cpu95-981221-03231.asc.

John L. Henning is secretary for the SPEC CPU Subcommittee. He is also a senior member of the technical staff in the benchmark performance engineering group, High Performance Servers Division, Compaq Computer Corporation. Henning studied philosophy at the University of New Hampshire and the University of Colorado. He is a member of the AAAS, the ACM, and the IEEE Computer Society. Contact him at j.henning@computer.org.