

## Benchmarking

We have already established that if performance were merely a factor of clock speed, then choosing a computing system would be very straightforward. In practice many factors impact computing performance.

**Ideal** – provide quantitative measure for *computing effectiveness* on the *workload* experienced by each user.

**Computing Effectiveness** → Performance as measured by the ability to effectively complete a computing workload.

**Workload** → mixture of programs and operating system commands necessary to support the user tasks.

**Pragmatics** – not in a position to evaluate the performance of a system on your specific application scenario (i.e. the operating system – application – (multi-)user mix).

**Reality** – rely on a *suitable* collection of measurable tasks – a **benchmark** – in order to guide your selection.

**Suitable** → representative of ‘real-world’ application scenarios, whilst also being specific enough for unambiguous, repeatable measurement.

**Benchmark** → the representative collection of programs on which the measurements are made.

Needless to say, there are a lot of potential problems here. For example,

- What mix of tasks should be included within any one benchmark?
  - Real programs are most representative, but even then how difficult should a task be on a real program in order to be representative of what users typically do?
  - Does the rule of thumb “10% of code is responsible for 90% of execution time” enable us to provide a more efficient way of constructing a benchmark?
- How much should be included within a benchmark measurement?
  - CPU alone; CPU plus I/O; CPU, I/O plus network?
  - The more that is included, the more sensitive the measurement becomes to the system mix (e.g. type and amount of RAM, cache and hard-disk). However, this is also a question of the intended application area. There is little point in testing CPU alone when the intended application is transaction-processing!
- What compiler and optimizations do you allow?
  - Compilers have a significant influence on computing effectiveness, how do we ensure that optimizations do not become un-representative of those typically employed?
- What measurements should be made?
  - How do you go about expressing performance in the benchmark (see additional comments on ‘faster than’, MIPS, FLOPS etc.).

## Establishing Benchmark Objectives

**Ideal** – set explicit limits on what it is that the benchmark is attempting to measure, where this should also reflect a system view on what provides a working computing system.

Unfortunately this is not necessarily obvious. For example, a useful CPU benchmark is actually designed to test: the CPU, the memory hierarchy, and compiler. **Why?**

**Pragmatics** – availability of useful benchmarks (see “selecting benchmark programs”) is divided up in terms of three general market sectors, each of which are supported by an independent standards body:

- Desktop systems: SPEC (www.spec.org) or Winstone (www.etestinglabs.com/benchmarks);
- Server systems: SPEC (www.spec.org) or TPC (www.tpc.org);
- Embedded systems: EEMBC (www.eembc.org).

### SPEC Benchmarks

- 11 benchmark categories defined

SPECapc	<ul style="list-style-type: none"> <li>- One of 2 graphics intensive benchmarks for single desktop computing systems.</li> <li>- Based purely on graphics intensive applications.</li> <li>- <a href="http://www.spec.org/gpc/apc.static/apcfaq.htm">http://www.spec.org/gpc/apc.static/apcfaq.htm</a></li> </ul>
SPECviewperf	<ul style="list-style-type: none"> <li>- Second of 2 graphics intensive benchmarks for single desktop computing systems.</li> <li>- Based on the performance in a series of 3D rendering problems running under the OpenGL graphics library.</li> <li>- <a href="http://www.spec.org/gpc/opc.static/opcview70.html">http://www.spec.org/gpc/opc.static/opcview70.html</a></li> </ul>
SPEC HPC96	<ul style="list-style-type: none"> <li>- One of 2 multi-processor benchmarks.</li> <li>- Emphasis on multi-processor systems evaluated with real industrial applications (might take several days to complete a run).</li> <li>- <a href="http://www.spec.org/hpg/">http://www.spec.org/hpg/</a></li> </ul>
SPEC OMP2001	<ul style="list-style-type: none"> <li>- Second of 2 multi-processor benchmarks.</li> <li>- Based on the OpenMP Application Program Interface for multi-platform, shared-memory parallel programming in C/ C++ and Fortran. <ul style="list-style-type: none"> <li>▪ <b>Why use C, C++ and Fortran rather than Java?</b></li> </ul> </li> <li>- Application emphasis on real scientific and engineering applications.</li> <li>- <a href="http://www.spec.org/hpg/omp2001/">http://www.spec.org/hpg/omp2001/</a></li> </ul>
SPEC CPU2000	<ul style="list-style-type: none"> <li>- Benchmark emphasizes desktop CPU performance.</li> </ul>

	<ul style="list-style-type: none"> <li>- Comprises of 11 integer (10 in C, 1 in C++) and 14 floating point (6 in Fortran 77, 5 in C, 3 in Fortran 90) benchmarks (<b>table 1</b>).</li> <li>- All programs are modified applications in which the I/O component has been minimized.</li> <li>- <a href="http://www.spec.org/osg/cpu2000/">http://www.spec.org/osg/cpu2000/</a></li> </ul>
SPEC JBB2000	<ul style="list-style-type: none"> <li>- One of 2 Java based application benchmarks, in this case emphasizing server-side Java business applications.</li> <li>- Provides an emulation of a 3-tier server-side Java application. Emphasis on business logic and object manipulation.</li> <li>- <a href="http://www.spec.org/osg/jbb2000/">http://www.spec.org/osg/jbb2000/</a></li> </ul>
SPEC JVM98	<ul style="list-style-type: none"> <li>- One of 2 Java based application benchmarks, in this case emphasizing performance of client-side Java Virtual Machines.</li> <li>- <a href="http://www.spec.org/osg/jvm98/jvm98/doc/benchmarks/index.html">http://www.spec.org/osg/jvm98/jvm98/doc/benchmarks/index.html</a></li> </ul>
SPEC MAIL2001	<ul style="list-style-type: none"> <li>- Benchmark designed to measure system suitability as a mail server based on the SMTP and POP3 Internet protocols.</li> <li>- Specific emphasis on ISP class mail servers (user counts in the range 10 thousand to 1 million).</li> <li>- <a href="http://www.spec.org/osg/mail2001/">http://www.spec.org/osg/mail2001/</a></li> </ul>
SPEC SFS97_R1	<ul style="list-style-type: none"> <li>- File server benchmark designed to measure NFS performance using a script of file server requests.</li> <li>- Naturally emphasizes evaluation of the disk and networked file system as well as the CPU.</li> <li>- <a href="http://www.spec.org/osg/sfs97r1/">http://www.spec.org/osg/sfs97r1/</a></li> </ul>
SPEC WEB99	<ul style="list-style-type: none"> <li>- One of 2 Web server benchmarks, in this case designed to simulate a multi-client environment in which both static and dynamic pages are requested from the server, and clients post information on the server.</li> <li>- <a href="http://www.spec.org/osg/web99/">http://www.spec.org/osg/web99/</a></li> </ul>
SPEC WEB_SSL	<ul style="list-style-type: none"> <li>- Second of 2 Web server benchmarks, in this case designed to include web server load associated with servicing encrypted requests. Otherwise it follows the same environment as SPEC WEB99.</li> <li>- <a href="http://www.spec.org/osg/web99ssl/">http://www.spec.org/osg/web99ssl/</a></li> </ul>

## Selecting benchmarking program suites

*Ideal* – a benchmark program should be selected for the following reasons [1],

- Task is utilized by many users;
- Exercises *significant* hardware resources;
  - significant** → a high utilization of a single computing element, or a lower utilization across multiple computing elements, as dictated by the overall benchmark objective. However, it should only exercise the set of computing resources of interest to the benchmark objective.
- Is demonstrated to exercise computing resource combinations that other tasks in the same benchmark do not cover (see example in [1]).
- Should not be *strongly biased* by specific system choices.
  - strongly biased** system choice → tasks designed to perform significantly better under specific operating systems are not desirable (little-endian and big-endian operating systems or architectures).

In practice, the programs (tasks) comprising a benchmark come in one of five forms that satisfy these objectives to varying degrees.

Real Applications	<ul style="list-style-type: none"> <li>- Complete application programs selected with a particular market segment in mind.</li> <li>- Input and output options (data) for each application require specification.</li> <li>- Portability problems encountered. Porting typically requires modification of the source code, the task might favor a specific platform.</li> </ul>
Modified Applications	<ul style="list-style-type: none"> <li>- Application programs still form the basis for tasks, but have components ‘stripped away’ to remove the significance of unwanted (with respect to the benchmark objective) resource calls. (e.g. remove or minimize I/O for a CPU only benchmark.)</li> </ul>
Kernels	<ul style="list-style-type: none"> <li>- relies on the rule of thumb that “10% of code is responsible for 90% of run time.”</li> <li>- Useful for isolating performance of individual features of a computing system (e.g. CPU alone). Specific examples include Livermore Loops and Linpack.</li> <li>- As the computing system performance as a whole is not considered (by definition only a small amount of the computer is ‘loaded’), kernels are only useful as a diagnostic!</li> </ul>
Toy benchmarks	<ul style="list-style-type: none"> <li>- task now limited to 10 – 100 lines of code and is therefore completely unrepresentative of tasks a user will actually be interested in using!</li> </ul>

Synthetic benchmarks	<ul style="list-style-type: none"> <li>- artificially created code that attempt to mimic instruction loads experienced in kernels.</li> <li>- Unfortunately are not able to represent the load of real applications.</li> </ul>
----------------------	---

**Pragmatics** – the only useful benchmarks are those that actually reflect application performance. Thus, real or modified applications are the only benchmarks that are worth basing a decision on.

**SPEC CPU 2000** – identifies the CPU as the target for the benchmarking activity, thus I/O should be minimized. As a consequence **Modified Applications** are utilized to formulate the suite of benchmark programs.

**Modified Applications** → in the case of SPEC CPU 2000 this means that they are modified to maximize **portability** between different platforms and minimize the roll of I/O in the benchmarks.

**Portability** → in the specific case of SPEC CPU 2000, 18 different platforms are supported. This represents a significant constraint on the programs employed within a suite of benchmark programs.

**Table 1** details the general characteristics of the programs utilized by the SPEC CPU 2000 suite. Note, the majority are Fortran and C programs, compilers for other languages not being robust enough across a sufficient number of different platforms to provide reliable performance.

## Case Study – SPEC CPU2000 on Alpha 21164 and Alpha 21264 based systems

Interested in ranking the performance of the following systems,

- Alpha 21164 CPU at 500Mhz (AlphaStation) ref – 500/500;
- Alpha 21164 CPU at 500Mhz (Personal Workstation) ref – 500au;
- Alpha 21164 CPU at 533Mhz (Alpha Server) ref – 4100 5/533;
- Alpha 21264 CPU at 500Mhz (Alpha Server) ref – DS20 6/500.

**Table 2** details the CPU and memory characteristics – are you able to predict on which applications each system will perform better?

**Table 2 : Alpha system CPU and Memory Characteristics [1].**

CPU	Alpha 21164			Alpha 21264
System	500/500	500au	4100 5/533	DS20
CPU (MHz)	500	500	533	500
L1 on chip cache	8 Kbytes (instruction) + 8 Kbytes (data)			64 Kbytes (inst.) + 64 Kbytes (data)
L2 on chip cache	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	174
Latency (processor cycles)	171	124	132	92
Bandwidth (Mbytes/s)	200	238	272	1,232

**Figure 1** summarizes performance over the 12 integer and 14 floating point programs comprising the SPEC CPU 2000 benchmark. Performance is expressed relative to that of a reference machine (300 Mhz Sun Ultra 5\_10) at a score of 100 across all programs.

With respect to the three 21164 based machines some general observations are made.

- Compare the two 500MHz machines, **figure 2**: differ by 5% or more in 17 of the 26 programs;

- Compare the best 500MHz 21164 machine on any benchmark program with that of the 533MHz machine (6.6% clock speed advantage), **figure 3**: the 533MHz machine performs better by 10% in 3 occasions, by less than 5% in 8 occasions and worse in 3 occasions.

Naturally the reason for these differences must lie in the memory hierarchy. From **table 2** it is apparent that,

- AlphaStation 500/500 has the larger cache;
- Personal Workstation (500au) has the best (lowest) cache and main memory latency, especially when measured in terms of processor cycles (**Why is it important to measure memory latency both in term of time and clock cycles?**);
- AlphaServer (4100 5/533) has the highest main memory bandwidth.

**Cache Contributions:**

Art	<p>From <b>figure 3</b>, the AlphaServer provides the largest performance improvement w.r.t. the 500MHz systems on application ‘art’.</p> <p>From <b>table 1</b> we know that the ‘art’ application has a 3.7Mbyte footprint; fitting entirely within the AlphaServer and AlphaStation cache.</p> <p>AlphaServer cache however is 20% faster than that of the AlphaStation, as reflected in the performance difference in <b>figure 3</b>.</p>
mcf	<p>From <b>figures 1 and 3</b>, the 533MHz AlphaServer is some 8% slower than the 500MHz AlphaStation on application ‘mcf’.</p> <p>Only performing a profile on the operation of the two systems is able to identify the source of this discrepancy.</p> <p>The AlphaServer (AlphaStation) has a CPI of 117 (95) for instruction sequences that involve a dependency between six load instructions and a subtraction operation.</p> <p>e.g.</p> <pre>R1 ← Mem[]; R2 ← Mem[]; R3 ← Mem[R1+32]; R2 ← R2 – R3;</pre> <p>Cycles are lost in the AlphaServer due to the smaller cache only being hit 15% of the time on this particular dependency–code profile; whereas the 8Mbyte cache of the AlphaStation is able to provide a much higher cache hit ratio.</p>
eon	<p>From <b>figure 3</b>, the only time that the Personal Workstation performs better than the AlphaServer is in the case of ‘eon’.</p> <p>Note that the Personal Workstation has the lowest cache latency times and ‘eon’ is the only application small enough to fit within a 2Mbyte cache. Thus, this is the only time that the memory hierarchy of the Personal Workstation is favored.</p>

**Main Memory:**

**Table 3** lists the first 5 floating point programs that cause the most cache misses. These do not necessarily correspond to the programs with largest memory requirements, **table 1**. Instead a large number of cache misses are caused by the manner in which memory addressing is performed. We would therefore expect the system with largest main memory bandwidth to benefit (AlphaServer).

**Table 3: Top five SPECfp2000 L3 cache misses / 1,000 issues [1].**

Benchmark	2 Mbytes	4 Mbytes	8 Mbytes
Art	29.1	0.5	0.4
Swim	23.9	23.7	23.6
Equake	23.6	22.2	21
Lucas	19.6	19.3	18.9
Applu	14.2	14	13

From **figure 3** it is apparent that the 14% bandwidth advantage of the AlphaServer results in a 0.5%, 5.5% and 11% improvement on ‘swim,’ ‘equake,’ and ‘lucas’.

lucas	The 21164 CPU shared by the three systems (AlphaStation, Personal Workstation, AlphaServer) is limited to 2 outstanding memory requests before stalls take effect. The bandwidth advantage of the AlphaServer is therefore most apparent when the processor can overlap memory references with computation. This occurs when the memory loads are distributed evenly through the code as opposed to being densely bunched in specific locations, where this is satisfied in the case of ‘lucas’.
-------	--

**Processor Performance:**

So far we have not considered the case of the 500MHz 21264 CPU architecture. From **table 2** it is apparent that this represents a significant improvement over the older 21164. Moreover, up to 8 outstanding loads and writes may be held and out of order execution is possible.

Irrespective of the application and configuration of the older architecture, the 21264 system is always significantly faster. The integer applications with largest improvement are,

Eon	1.89	Both applications have a small memory footprint, implying that most benefit is due to CPU architectural advantages and lower cache latency in the 21264.
Crafty	1.94	
Gcc	2.03	Here the principle factor is the much better memory bandwidth as the ‘gcc’ application requires 156 Mbytes of virtual memory. Note however that this is less than the factor of raw bandwidth improvement between the two systems.

Floating point applications also receive speed ups in line with the significance of bandwidth and cache latencies. With all three worst case cache tests (swim, lucas, applu) being significantly improved. In the case of 'equake', the organization of the memory references are such that the application is latency bound as opposed to cache bound.

## **Additional Reference Material**

[1] Henning J.L., SPEC CPU2000: Measuring CPU Performance in the New Millennium. IEEE Computer Magazine. pp 28-35, July 2000.

## Appendix A – Figures

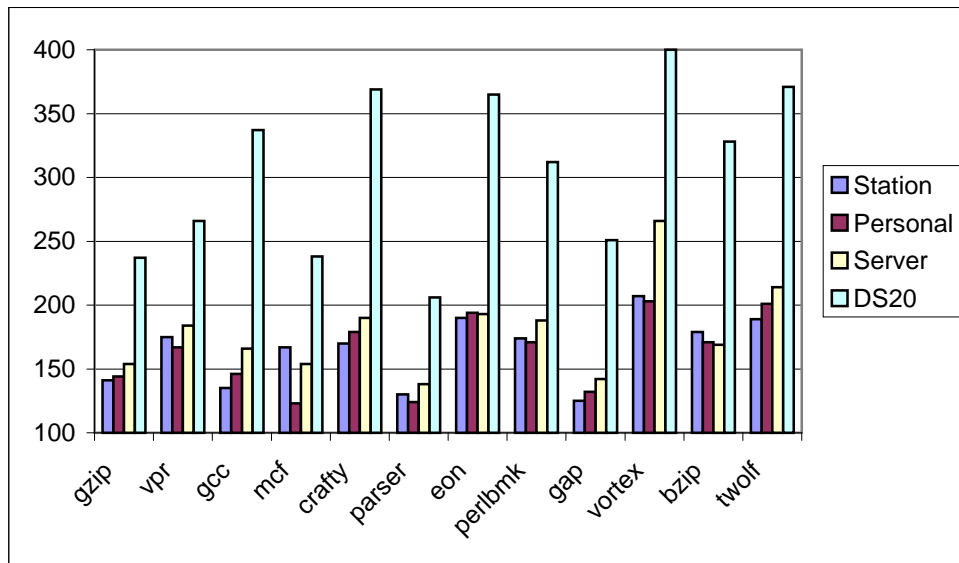


Figure 1(a) – Performance of Alpha systems on SPECint2000 (relative to 300 Mhz Sun Ultra 5\_10).

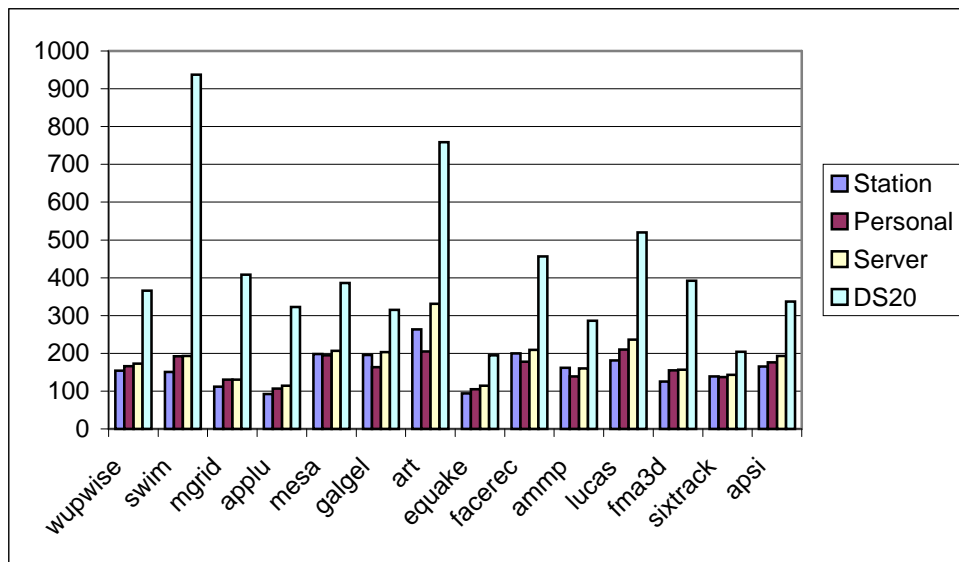


Figure 1(b) – Performance of Alpha systems on SPECfp2000 (relative to 300 Mhz Sun Ultra 5\_10).

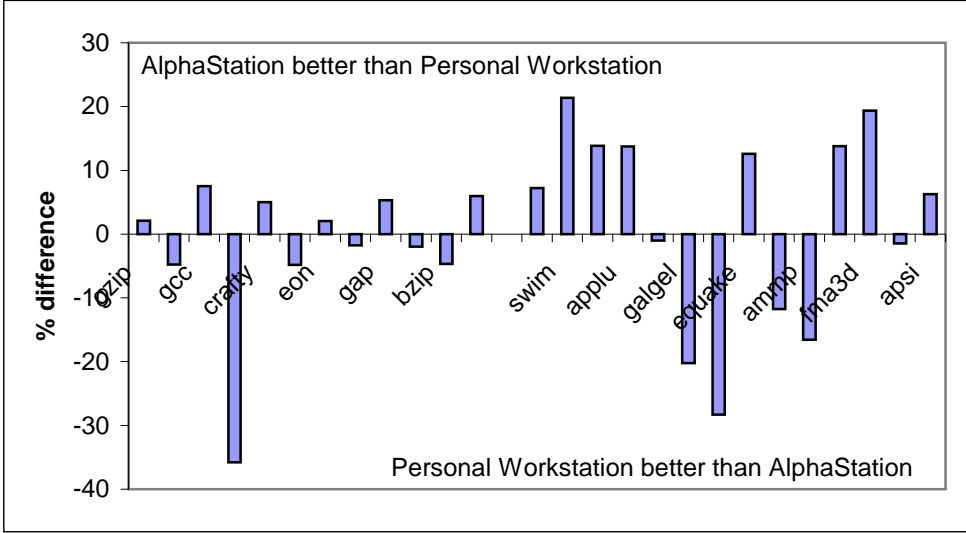


Figure 2 – AlphaStation against Personal Workstation alone.

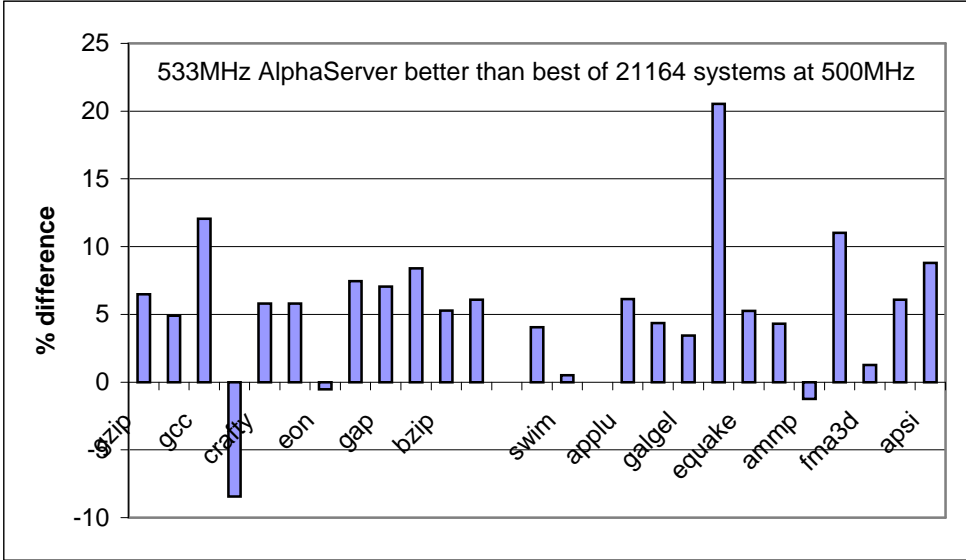


Figure 3 – 533MHz AlphaServer against the best of either the 500MHz AlphaStation or Personal Workstation.

## Appendix B – Table 1

Benchmark	Language	KLOC	Resident (Mbytes)	Virtual Memory (Mbytes)	Description
SPECint2000					
Gzip	C	7.6	181	200	Compression
Vpr	C	13.6	50	55.2	FPGA circuit place and route
Gcc	C	193	155	158	C programming compiler
Mcf	C	1.9	190	192	Combinatorial Optimization
Crafty	C	20.7	2.1	4.2	Game playing: Chess
Parser	C	10.3	37	62.5	Word Processing
Eon	C++	34.2	0.7	3.3	Computer visualization
Perlbmk	C	79.2	146	159	Perl programming language
Gap	C	62.5	193	196	Group theory, interpreter
Vortex	C	54.3	72	81	Object orientated database
Bzip2	C	3.9	185	200	Compression
Twolf	C	19.2	1.9	4.1	Place and route simulator
SPECfp2000					
Wupwise	F77	1.8	176	177	Physics: Quantum chromodynamics
Swim	F77	0.4	191	192	Shallow water modeling
Mgrid	F77	0.5	56	56.7	Multigrid solver: 3D potential field
Applu	F77	7.9	181	191	Partial differential equations
Mesa	C	81.8	9.5	24.7	3D graphics library
Gaigel	F90	14.1	63	155	Computational fluid dynamics
Art	C	1.2	3.7	5.9	Image recognition/ neural networks
Equake	C	1.2	49	51.1	Seismic wave propagation simulation
Facerec	F90	2.4	16	18.5	Image processing: Face recognition
Ampmp	C	12.9	26	30	Computational chemistry
Lucas	F90	2.8	142	143	Number theory/ primary testing
Fma3d	F90	59.8	1.3	105	Finite-element crash simulation
Sixtrack	F77	47.1	26	59.8	Nuclear physics accelerator design
Apsi	F77	6.4	191	192	Meteorology: Pollutant distribution