9

Overview of the SPEC Benchmarks

Kaivalya M. Dixit

IBM Corporation

“The reputation of current benchmarketing claims regarding system performance is on par with the promises made by politicians during elections.”

Standard Performance Evaluation Corporation (SPEC) was founded in October, 1988, by Apollo, Hewlett-Packard,MIPS Computer Systems and SUN Microsystems in cooperation with *E. E. Times*. SPEC is a nonprofit consortium of 22 major computer vendors whose common goals are “to provide the industry with a realistic yardstick to measure the performance of advanced computer systems” and to educate consumers about the performance of vendors’ products. SPEC creates, maintains, distributes, and endorses a standardized set of application-oriented programs to be used as benchmarks.

9.1 Historical Perspective

Traditional benchmarks have failed to characterize the system performance of modern computer systems. Some of those benchmarks measure component-level performance, and some of the measurements are routinely published as system performance. Historically, vendors have characterized the performances of their systems in a variety of confusing metrics. In part, the confusion is due to a lack of credible performance information, agreement, and leadership among competing vendors.

Many vendors characterize system performance in millions of instructions per second (MIPS) and millions of floating-point operations per second (MFLOPS). All instructions, however, are not equal. Since CISC machine instructions usually accomplish a lot more than those of RISC machines, comparing the instructions of a CISC machine and a RISC machine is similar to comparing Latin and Greek.

9.1.1 Simple CPU Benchmarks

Truth in benchmarking is an oxymoron because vendors use benchmarks for marketing purposes. There are three types of popular benchmarks: kernel, synthetic, and application.

Kernel benchmarks are based on analysis and the knowledge that in most cases 10 percent of the code uses 80 percent of the CPU resources. Performance analysts have extracted these code fragments and have used them as benchmarks. Examples of kernel benchmarks are Livermore Loops and Linpack benchmarks [Weicker, 1991]. The fundamental problems with kernel benchmarks are that they are usually small, fit in cache, are prone to attack by compilers, and measure only CPU performance.

Synthetic benchmarks are based on performance analysts' experience and knowledge of instruction mix. Examples of synthetic benchmarks are Dhrystone and Whetstone. New technologies and instruction set architectures make some older assumptions regarding instruction mix obsolete. Synthetic benchmarks offer problems similar to those of kernel benchmarks.

From a user's perspective, the best benchmark is the user's own application program. Examples of application benchmarks are Spice (circuit designers) and GNU compiler (software developers using GNU environments). Unfortunately, there are thousands of applications, and many of them are proprietary. A benchmark suite with a large number of applications is also impractical because of difficulties in porting and evaluation and long runtime.

9.1.2 Aging CPU Benchmarks

Popular benchmarks, such as Dhrystone, Linpack, and Whetstone, suffer from the following problems, which make their use in comparing machines difficult:

- Small, fit in cache
- Obsolete instruction mix
- Uncontrolled source code
- Prone to compiler tricks
- Short runtimes on modern machines
- Single-number performance characterization with a single benchmark
- Difficult to reproduce results (short runtime and low-precision UNIX timer)

Dhrystone was developed by Reinhold Weicker in 1984. This synthetic benchmark spends significant time on string functions. It was designed to measure the integer performance of small machines with simple architectures. RISC machines generally beat CISC machines on this benchmark because of RISC machines' large number of registers and the localities of code and data. The performance metric is Dhrystones per second [Weicker, 1991].

Linpack is a collection of linear algebra subroutines authored by Jack Dongarra in 1976. It is used to characterize the floating-point performance of machines. This benchmark operates on a 100x100 matrix, which was considered to be a large matrix in 1976. The program fits comfortably in the caches of many machines. Its performance is characterized by millions of floating-point operations per second (MFLOPS). The performance metrics are single- and double-precision MFLOPS [Dongarra, 1983].

Whetstone is a popular synthetic floating-point benchmark originally written by H. J. Curnow and B. A. Wichman in 1976. It contains ten modules that perform a variety of numerical computations (e.g., arrays, trigonometric functions). The benchmark is small and sensitive to the ordering of library modules and size of caches. The performance metrics are single- and double-precision Whetstones per second [Curnow, 1976].

The above benchmarks have had long lives, considering the dynamics of technological advances in the computer industry, but it is time to gracefully retire them.

9.1.3 Evolution of SPEC Benchmark Suites

A computer system's performance cannot be characterized by a single number or a single benchmark. Characterization of a system's performance with a single benchmark is akin to the fabled blind man's description of an elephant. Many users (decision makers), however, are looking for a *single-number* performance characterization. The customer faces a plethora of confusing performance metrics and reluctance by the press to publish complete information on performance and configuration. There are no simple answers. Both the press and the customer, however, must be informed about the danger and the folly of relying on either a single performance number or a single benchmark [Dixit, 1991a].

Unfortunately, computer vendors have, until recently, been unable or unwilling to agree on a set of standard benchmarks, which has made it virtually impossible for customers to evaluate and compare competing systems. The lack of standards in benchmarking has also created problems for vendors and hardware, software, and system designers. There is no absolute objective truth in comparing benchmarking results.

SPEC started to address these issues by selecting and developing real application benchmarks that would exercise major system components. SPEC's members struggled with questions like, Do we need one benchmark or many benchmarks? What workload should be used to show a particular system in the best light? What is system performance and how can we compare different computer systems in an objective manner?

SPEC wanted to compare system performance across many different technologies, architectures, implementations, memory systems, I/O subsystems, operating systems, clock rates, bus protocols, compilers, libraries, and application software. Additional problems, such as scaling system performance with multiple processors and peripherals, had to be considered. And other factors, like graphics and networking, complicated matters.

The raw hardware performance depends on many components: CPUs, floating-point units (FPUs), I/Os, graphics and network accelerators, peripherals, and memory systems. However, system performance as seen by the user depends on the efficiency of the operating system, compiler, libraries, algorithms, and application software.

With so many variables, SPEC's major goal was that the same source code (machine independent) would run on all members' machines, which required that all benchmarks be ported to SPEC members' machines. This turned out to be a nontrivial task.

Portability conflicts are resolved by SPEC members at SPEC *bench-a-thons*. A bench-a-thon is an intense five-day workshop during which engineers from SPEC's member companies develop benchmarks and tools that are portable across operating systems and platforms.

SPEC chose a simple measure, elapsed time, to run the benchmark. A simple speed metric and machine-independent code were keys to providing a comprehensive and fair comparison between competing machines.

SPEC measures performance by determining the times required to run a suite of applications and then comparing the time for completion of each with the time of a reference machine (DEC VAX 780). The individual results are called the SPECratios for that particular machine. There is general consensus that the use of a single number is not the best way to represent the results of a suite of benchmarks. Unfortunately, pressures exist that force the development of just such a number. By definition, all SPEC metrics for DEC VAX 780 are one. (For definitions of SPEC metrics, see Appendix D. of this chapter).

The experts seem to be split between those who favor the use of the weighted arithmetic mean and others who believe the geometric mean of the numbers provides the best composite number [Hennessy and Patterson, 1990]. Following extensive debate, SPEC selected the geometric mean—a composite metric—for the following reasons:

- It provides for ease of publication.
- Each benchmark carries the same weight.
- SPECratio is dimensionless, so use of the harmonic mean did not make sense.
- It is not unduly influenced by long running programs.
- It is relatively immune to performance variation on individual benchmarks.
- It provides a consistent and fair metric.

The SPEC{mark89, int89, fp89, int92, fp92} metrics are an overall (geometric mean) statistical measure of performance on SPEC CPU benchmark suites. However, SPEC strongly recommends that all SPECratios be published and used to make informed comparisons.

9.2 History of SPEC Products

SPEC has developed a series of CPU benchmark suites: SPEC Release 1, CINT92, and CFP92. CINT92 and CFP92 replace the SPEC Release 1 CPU benchmark suite [Dixit, 1992]. The system development multitasking (SDM) suite was developed to measure the throughput capacity of machines [Dronamraju, Naseem, & Chelluri, 1990].

SPEC Release 1 was announced in October, 1989, and contains four C (integer computation) benchmarks and six FORTRAN (double-precision floating-point computation) benchmarks (SPEC Benchmark Suite Release 1.0, 1990). SPEC Release 1 characterized CPU performance with the speed performance metrics: *SPECint89, SPECfp89,* and a composite *SPECmark89*.

In May, 1991, SPEC announced its SDM Release 1, which contains two system-level benchmarks. The benchmarks in CPU suites (SPEC Release 1, CINT92, and CFP92) spend over 99 percent of their time in the “user mode” and do not stress either the operating system or I/O subsystem. SDM Release 1 characterizes the peak throughput performance of the system in *scripts per hour* [Dronamraju, Balan, & Morgan, 1991]. The SDM suite was developed to address the throughput capacity of machines and does not replace CPU suites. The prefix “S” in SDM Release 1 identifies it as a system-level benchmark suite.

In January, 1992, SPEC released two component-level CPU suites (CINT92 and CFP92) to replace the SPEC Release 1.2b suite. The prefix “C” in CINT92 and CFP92 identifies them as component benchmark suites.

CINT92 contains six integer benchmarks, and CFP92 contains 14 floating-point benchmarks. CINT92 measures the integer speed performance of the CPU with metric *SPECint92,* and CFP92 measures the floating-point speed performance of the CPU with metric *SPECfp92*.

SPEC defined a new methodology and developed two new capacity metrics in June of 1992: SPECrate_int92 and SPECrate_fp92 [Carlton, 1992].

9.3 SPEC Release 1

SPEC Release 1 provides CPU speed performance measurements that are analogous to a sports car's lap times (elapsed time) on ten different race tracks (ten benchmarks) [Mashey, 1990]. SPEC Release 1 is a CPU benchmarking suite of four integer benchmarks written in C and six double-precision floating-point benchmarks written in FORTRAN. SPEC Release 1 is now obsolete.

SPEC Release 1 characterized CPU performance with the speed performance metrics *SPECint89*, *SPECfp89*, and a composite *SPECmark89*. SPEC released a throughput version of SPEC Release 1 as SPEC Release 1.2b in March, 1990. SPEC Release 1.2b characterizes the multiprocessor (MP) systems with the capacity metrics *SPECintThruput*, *SPECfpThruput* and *SPECThruput* [Greenfield, Raynoha, & Bhandarkar, 1990]. Table 9.1 summarizes the SPEC Release 1 suite.

Table 9.1 SPEC Release 1 benchmarks.

Benchmarks	Type	Application Description
001.gcc	INT[1]	GNU C compiler
008.espresso	INT	PLA optimizing tool
013.spice2g6	FP[2]	Circuit simulation and analysis
015.doduc	FP	Monte Carlo simulation
020.nasa7	FP	Seven floating-point kernels
022.li	INT	LISP interpreter
023.eqntott	INT	Conversions of equations to truth table
030.matrix300	FP	Matrix solutions
0.42fpppp	FP	Quantum chemistry application
047.tomcatv	FP	Mesh generation application

Notes: 1. INT = Integer intensive benchmark.
2. FP = Double-precision floating-point benchmark.

Experience has shown that useful information is obscured by the monochromatic (SPECmark89—a single number) view of performance and that the complete spectrum—all ten SPECratios—should be used to make fair comparisons.

Over 150 SPEC Release 1 results on a variety of platforms ranging from personal computers to high-end servers have been reported in SPEC newsletters. Tables B.1 and B.2 (Appendix B, this chapter) provide SPEC Release 1 results and SPEC Thruput Method A results extracted from recent issues of the SPEC newsletter.

9.4 Reasons for New CPU Suites CINT92 and CFP92

CPU performance more than doubles every year. SPEC Release 1 appeared in 1989, and since then improvements in architecture, hardware, clock rates, and software have dramatically reduced the runtimes of SPEC Release 1 benchmarks (e.g., 030.matrix300, 042.fpppp, and 047.tomcatv), thereby increasing the potential for errors in measurement. SPECmark89 was significantly influenced by the floating-point bias (e.g., six of the ten benchmarks in FORTRAN) in the suite and especially the peak floating-point performance (achieved by using vector preprocessors) on 030.matrix300 and 020.nasa7—kernel benchmarks [Keatts, Balan, & Bodo, 1991].

To alleviate the above problems, SPEC announced two new benchmark suites—CINT92 and CFP92. These suites build on the real end-user benchmarks in SPEC Release 1. They contain several additional application benchmarks and enhancements that enable them to keep pace with faster hardware and smarter optimizing compilers.

In spite of the popularity of SPECmark89, SPEC's members voted not to combine the results of two fundamentally different workloads (integer and floating-point), and so SPEC has not defined SPECmark92. CINT92 and CFP92 provide CPU speed performance measurements that are analogous to a sports car's lap times (elapsed time) on 20 different race tracks (20 benchmarks).

SPEC strongly encourages users and vendors to publish the results they have obtained with the new CINT92 and CFP92 suites. CINT92 measures the integer speed performance of the CPU with metric *SPECint92*, and CFP92 measures the floating-point speed performance of the CPU with metric *SPECfp92*. In June of 1992, SPEC defined a new methodology and two new capacity metrics for both uniprocessors and multiprocessors: *SPECrate_int92* and *SPECrate_fp92*. For a detailed understanding of the SPECrate, the reader is directed to the article by Carlton [1992]. See Appendix D. (this chapter) for definitions of various metrics.

9.4.1 Integer Suite, CINT92

The new integer suite, CINT92, contains six CPU intensive integer benchmarks written in C [Dixit, 1991b]. CINT92 retains three benchmarks (008.espresso, 022.li, and 023.eqntott) from SPEC Release 1. Two new benchmarks, 026.compress and 072.sc, are applications commonly available on UNIX systems. The longer running 085.gcc replaces 001.gcc from SPEC Release 1. The CINT92 suite supersedes SPEC Release 1. Tables A.1 and A.2 (Appendix A, this chapter) show recently reported results on CINT92. Table 9.2 summarizes applications in the CINT92 suite.

Table 9.2 The CINT92 suite.

Benchmarks	LOC[1]	Object[2]	Application Description

008.espresso	14,800	276	Programmable logic array (PLA) optimizing tool
022.li	7,700	235	LISP interpreter
023.eqntott	3,500	397	Conversion of equations to truth table
026.compress	1,500	510	Text compression algorithm
072.sc	8,500	343	Spreadsheet application
085.gcc	87,800	884	GNU compiler application

Notes: 1. LOC = lines of source code.
2. Object = Static size (KB) of SPARC executable.

9.4.1.1 Descriptions of the CINT92 Benchmarks

008.espresso is a tool for the generation and optimization of programmable logic arrays (PLAs). It characterizes work in the electronic design automation market and the logic-simulation and routing-algorithm areas [UC Berkeley, 1988]. The elapsed time to run a set of four different input models characterizes 008.espresso's performance. The program manipulates arrays in small loops, and its performance is sensitive to the storage allocation algorithm (e.g., *malloc*) and cache size. This benchmark represents circuit design applications.

022.li is a LISP interpreter that solves nine queens' problem. The backtracking algorithm is recursive [Betz, 1988]. This benchmark is based on XLISP 1.6, which is a small implementation of LISP. The backtracking algorithm poses a challenge for register window architectures. This benchmark represents object-oriented programming applications.

023.eqntott translates a logical representation of an equation to a truth table. The primary computation is a sort operation [UC Berkeley, 1985]. The program fits comfortably in the instruction cache of most machines, but the large data size may cause data cache misses on some machines. A small loop with several conditional branches poses a challenge to minimize bubbles in the pipeline. This benchmark represents circuit design applications.

026.compress is a text compression and decompression utility that reduces the size of the named files using adaptive Lempel-Ziv coding. This benchmark compresses and decompresses a 1-MB file 20 times. The amount of compression that is obtained depends on the size of the input, the number of bits per code, and the distribution of common substrings. Two processes (compress and uncompress) communicate via *pipes*, which is unique to this benchmark in the benchmark suite. This benchmark represents data compression applications.

072.sc is a spreadsheet benchmark that calculates budgets, SPECmarks, and 15-year amortization schedules. It performs standard spreadsheet operations: cursor movement, data entry, data movement, file handling, row and column operations, operation on ranges, numeric expressions, and evaluations. All output is directed to a file to alleviate problems of window sizes and output validation. An efficient *curses* package should improve the performance on this benchmark. 072.sc performs a small (less than 1 percent) amount of floating-point computation. This benchmark represents spreadsheet applications.

085.gcc is a popular C compiler used as a benchmark [Stallman, 1989]. It is a CPU intensive C benchmark that spends about 10 percent of its run-time in I/O operations and executes code spread over many subroutines. Its performance is sensitive to cache size and the speed of the I/O device [Phillip, 1992]. The benchmark measures the time it takes for the GNU C compiler to convert 76 preprocessed source files into optimized SUN-3 assembly (.s files) language output. This benchmark characterizes work done in a workstation-based software development environment.

9.4.1.2 Instruction Profile of the CINT92 Suite

Instruction profiling is used to understand benchmark behavior and locate potential sources of bottlenecks in hardware and software. It is very important to remember that the number of instructions executed and the instruction mix are highly dependent on the architecture, implementation, instruction set, preprocessors (e.g., KAP, VAST), compiler efficiency, optimization level, and runtime libraries. The instruction profiles will be different for other platforms (e.g., HP, IBM, DEC, Intel, Motorola) [Austin & Sohi, 1992]. The following profiling information relates to the CINT92 benchmark suite compiled for SPARCstation 2 (SS-2) with the SUN C 1.1 compiler.

The benchmarks were analyzed with *SpixTools*, a set of SUN internal performance measurement tools. *SpixTools* assume infinite cache and trace instructions only in the user mode. The instruction mix shown in Table 9.3 does not include code executed in the operating system. Most of the benchmarks spend 95–99 percent of their time in the user mode, so instruction trace is fairly accurate. 085.gcc spends about 10 percent of its time in the kernel mode, and those instructions are not reflected in the instruction counts. The data represented in Table 9.3 were extracted from the output of *SpixTools*. The salient runtime behavior (e.g., memory references, branch operations, and instruction counts) is detailed in Table 9.3.

Table 9.3 CINT92—SPARC instruction counts.

Benchmarks	Memory %	Branch %	Other %	Millions of Instructions
008.espresso	28.1	20.4	51.5	2,931
022.li	**33.3**	23.7	*43.0*	**4,661**
023.eqntott	*16.5*	**26.5**	57.0	1,322
026.compress	25.7	*16.6*	**57.7**	2,699
072.sc	24.1	21.8	54.1	*1,255*
085.gcc	26.6	20.2	53.2	4,624
TOTAL	27.3	23.3	49.4	17,692

Memory = loads + stores; Other = total – memory – branch.
Italics indicates the lowest number in the column.
Bold indicates the highest number in the column.

The programs executing high percentages of memory (load + store) and branch operations are more likely to cause cache misses and pipeline stalls. In the CINT92 suite, 022.li is the most memory intensive (33 percent) and 023.eqntott is the most branch intensive (27 percent) benchmark. The CINT92 suite exhibits an interesting range of instruction mix:

• Memory operations:	17–33 percent
• Branch operations:	17–27 percent
• Other operations:	43–58 percent

These numbers vary with different preprocessors' and compilers' options.

9.4.2 Floating-Point Suite, CFP92

The component-level CPU intensive floating-point benchmark suite contains 14 benchmarks that represent applications in circuit theory, nuclear simulation, quantum chemistry, electromagnetic particle-in-cell simulation, mesh generation, optical ray tracing, neural network, simulation of human ear, weather prediction, quantum physics, astrophysics, and several kernels used by NASA Ames.

CFP92 contains 12 floating-point benchmarks written in FORTRAN and two benchmarks written in C. CFP92 retains three benchmarks (013.spice2g6, 015.doduc, and 047.tomcatv) from SPEC Release 1. The new 093.nasa7 replaces 020.nasa7, and the longer running 094.fpppp replaces 042.fpppp. The suite contains two kernel benchmarks—015.doduc and 093.nasa7. Nine benchmarks represent double-precision applications, and five benchmarks represent single-precision applications. The CFP92 suite supersedes SPEC Release 1. See Appendix D. (this chapter) for definitions of CFP92 metrics (SPECfp92 and SPECrate_fp92). Tables A.1 and A.2 in Appendix A show recently reported results on CFP92. Table 9.4 summarizes applications in the CFP92 suite.

Table 9.4 The CFP92 suite.

CFP92	LOC[1]	Object[2]	Application Description
013.spice2g6	18,900	8,422	Circuit design (FDP)[3]
015.doduc	5,300	408	Nuclear simulation, Monte Carlo (FDP)
034.mdljdp2	4,500	457	Molecular dynamics—Chemistry (FDP)
039.wave5	15,100	14,634	Maxwell's equation (FSP)[4]
047.tomcatv	200	3,800	Mesh generation—Airfoil (FDP)
048.ora	500	202	Optical ray tracing (FDP)
052.alvinn	300	581	Neural network—Robotics (CSP)[5]
056.ear	5,200	212	Simulation of human ear (CSP)
077.mdljsp2	3,900	418	Molecular dynamics—Chemistry (FSP)

078.swm256	500	3,826	Shallow water model, weather prediction (FSP)
089.su2cor	2,500	4,367	Quantum physics, Quark-Gluon theory (FDP)
090.hydro2d	4,500	555	Astrophysics, Navier Stokes equations (FDP)
093.nasa7	1,200	3,062	Seven NASA Ames kernels (FDP)
094.fpppp	2,700	564	Quantum chemistry (FDP)

Notes:
1. LOC = Lines of source code.
2. Object = Static size (KB) of SPARC object code.
3. (FDP) = Double-precision floating-point benchmark in FORTRAN.
4. (FSP) = Single-precision floating-point benchmark in FORTRAN.
5. (CSP) = Single-precision floating-point benchmark in C.

9.4.2.1 Descriptions of the CFP92 Benchmarks

013.spice2g6 is an analog circuit simulation application that is heavily used in electronic design automation (EDA) markets. This application stresses the data cache (UC Berkeley, 1987). It is written mostly in FORTRAN, and the UNIX interface of the program is written in C. *013.spice2g6* runs five copies of the "grey code" circuit and spends about 4 percent of its runtime performing floating-point computations. The data accesses cause high data cache misses. More than 80 percent of assignments are memory-to-memory transfers. This benchmark represents circuit design applications.

015.doduc is a Monte Carlo simulation of the time evolution of a thermo-hydraulical model ("hydrocode") for a nuclear reactor's component. It includes little I/O, has an abundance of short branches and loops, and executes code spread over many subroutines. This benchmark is difficult to vectorize. 015.doduc is a large kernel benchmark that represents electronic computer-aided design (ECAD) and high-energy physics applications (Doduc, 1989).

034.mdljdp2 solves equations of motion for a model of 500 atoms interacting through the idealized Lennard-Jones potential. This would be a typical system used to model the structure of liquid argon. At each time step in the calculation, the position and velocity of each particle in the model system are used to calculate the configurational energy and pressure through equations of statistical mechanics. The density and temperature for the model are supplied through an input file. This benchmark represents quantum chemistry applications.

039.wave5 solves Maxwell's equations and particle equations of motion on a Cartesian mesh with a variety of field and particle boundary conditions. It is a two-dimensional, relativistic, electromagnetic particle-in-cell simulation code used to study various plasma phenomena. The benchmark problem involves 500,000 particles on 50,000 grid points for five time steps. This benchmark computes in single-precision and causes high data cache misses. This benchmark represents scientific and engineering applications.

047.tomcatv is a highly (90–98 percent) vectorizable program for the generation of two-dimensional coordinate systems around general geometric domains such as airfoils and cars. It uses two Poisson equations to produce a mesh that adapts to the physical region of interest. The transformed nonlinear equations are replaced by a finite difference approximation, and the resulting system is solved using successive line over-relaxation. It causes high data cache misses on several platforms. This benchmark represents computer-aided design (CAD) and other engineering applications.

048.ora traces rays through an optical system composed of spherical and plane surfaces. The design of high-quality lenses, such as those used in 35 mm reflex cameras, is a compute-intensive task. The many types of image evaluation that can be computed for optical systems, several of which include diffraction effects, are also based on ray tracing. Much of the computation time is consumed by ray tracing, that is, tracing the precise simulation of light passing through the optical system. 048.ora is the most floating-point intensive application in the suite and favors architectures that support "square-root" instruction in hardware. This benchmark represents optical ray tracing applications.

052.alvinn trains a neural network called ALVINN (autonomous land vehicle in a neural network) using backpropagation. It is a CPU intensive single-precision robotic application in C. 052.alvinn is designed to take as input sensory data from a video camera and a laser range finder and compute as output directions for a vehicle to travel in order to stay on the road. The 1220 input units comprise the two retinas, one from a video camera and one from a laser range finder. This benchmark represents neural network applications.

056.ear simulates the human ear. It is a CPU intensive single-precision floating-point benchmark written in C. 056.ear takes a sound file as input and produces a cochleagram—a representation that roughly corresponds to spectral energy as a function of time. It makes extensive use of complex fast Fourier transform (FFT) and other *math library* functions. The program creates a 1-MB output file. This benchmark represents engineering and scientific applications.

077.mdljsp2 solves the equations of motion for a model of 500 atoms interacting through the idealized Lennard-Jones potential. This would be a typical system used to model the structure of liquid argon. At each time step in the calculation, the position and velocity of each particle in the model system are used to calculate the configurational energy and pressure through equations of statistical mechanics. The density and temperature for the model are supplied through an input file. 077.mdljsp2 is a single-precision version of 034.mdljdp2. This benchmark represents quantum chemistry applications.

078.swm256 solves the system of shallow water equations using finite difference approximations on a 256x256 grid. It is a memory and floating-point intensive, single-precision FORTRAN benchmark. This benchmark represents weather prediction and other scientific and engineering applications.

089.su2cor solves for masses of elementary particles that are computed in the framework of the quark-gluon theory. The data are computed with a Monte Carlo method taken from statistical mechanics. The configuration is generated by the warm bath method. The benchmark is highly vectorizable (90–98 percent). This benchmark represents an area of quantum physics.

090.hydro2d solves hydrodynamical Navier Stokes equations to compute galactical jets. Almost all of the benchmark code is vectorizable (95–99 percent). This benchmark represents an area of astrophysics.

093.nasa7 is a collection of seven kernels that calculate matrix multiplication, complex radix 2 FFT, Cholesky decomposition, block tridiagonal matrix, vortex solution, vortex creation, and inversion of three matrix pentadiagonals [Bailey & Barton, 1985]. This benchmark represents a variety of engineering and scientific applications commonly used by NASA Ames.

094.fpppp measures performance on one style of computation (two electron integral derivative) which occurs in the Gaussian series of programs. This benchmark has a very large basic block and is difficult to vectorize. The amount of computation is proportional to the fourth power of the number of atoms (the benchmark solves the problem for 16 atoms). It is a memory and floating-point intensive double-precision FORTRAN benchmark, and it represents an area of quantum chemistry.

9.4.2.2 Instruction Profile of the CFP92 Suite

The benchmarks were analyzed with *SpixTools*, a set of SUN internal performance measurement tools. The data shown below were extracted from the output of *SpixTools*. The salient runtime behavior (i.e., memory references, floating-point operations, branch operations, and instruction mix) is detailed in Table 9.5.

Table 9.5 CFP92 - SPARC instruction counts.

Benchmarks	Memory %	Fpop %	Branch %	Other %	Millions of Instructions
013.spice2g6	25.0	*4.2*	**13.6**	**57.2**	**22,878**
015.doduc	29.4	25.9	8.4	36.2	*1,303*
034.mdljdp2	26.7	47.7	2.0	23.7	3,217
039.wave5	33.7	21.9	7.4	37.0	4,432
047.tomcatv	45.0	32.9	1.8	20.3	1,406
048.ora	*13.2*	**56.3**	7.4	23.1	1,695
052.alvinn	**48.3**	27.6	5.1	18.9	3,506
056.ear	35.3	26.4	6.2	32.1	17,768
077.mdljsp2	21.4	52.8	2.4	23.5	3,017
078.swm256	43.2	46.6	*0.6*	*9.5*	10,810
089.su2cor	33.3	31.8	4.3	30.6	4,915
090.hydro2d	28.5	38.0	3.9	29.6	7,891
093.nasa7	38.9	28.0	5.0	28.1	5,917
094.fpppp	47.2	34.7	3.0	15.1	5,076
TOTAL	33.2	27.2	6.5	33.1	93,741

Memory = loads + stores; Fpop = floating-point operations;

Other = total – memory – fpop – branch
Italics indicates the lowest value in the column; bold indicates the highest value.

The programs that execute high percentages of memory (load + store), floating-point operations, and branch operations are more likely to cause cache misses and pipeline stalls. In the CFP92 suite, 052.alvinn is the most memory intensive (48 percent) benchmark, 048.ora is the most floating-point intensive (56 percent) benchmark, and 013.spice2g6 is the most branch intensive (14 percent) benchmark. It is obvious that 013.spice2g6 is not very floating-point intensive (4 percent) on the input circuit model used in the benchmark. The CFP92 suite exhibits an interesting range of instruction mix:

• Memory operations:	13–48 percent
• Floating-point operations:	4–56 percent
• Branch operations:	1–14 percent
• Other operations:	10–57 percent

It is important to remember that floating-point benchmarks contain significant integer computational components as seen above. These numbers vary with different preprocessors' and compilers' options.

9.5 The System Development Multitasking (SDM) Suite

CPU suites (SPEC Release 1, CINT92, and CFP92) provide insights into the speeds (how fast) of machines; they do not provide adequate information about the throughput (how much) of machines. The benchmarks in the CPU suites are single-threaded, CPU (integer and floating-point) intensive programs. SPEC Release 1, CINT92, and CFP92 benchmarks spend over 99 percent of their time in the user mode and do not stress either the operating system or I/O subsystem.

The SPECthruput metric, also in SPEC Release 1, provided means to measure and report performance results of multiprocessors. This methodology did not stress additional system resources (e.g., system calls, I/O, and interprocessor communications), because it used a CPU intensive workload. However, the methodology still provided a level playing field for all existing vendors of multiprocessors, and many vendors have reported results.

The SDM suite was developed to address the throughput capacities of machines.

9.5.1 SDM Benchmarks

The SDM Release 1 benchmark suite contains two multitasking system-level benchmarks for UNIX systems: 057.sdet and 061.kenbus1. Both benchmarks stress a variety of system components (e.g., CPU, I/O, memory, operating system, and many UNIX utilities). These benchmarks enable system developers to compare different releases of hardware and software, including the UNIX operating system. They allow end-users to compare systems based on the behavior of each system's throughput as the load varies on the system.

This release addresses a simple question from a user's perspective: how much work can I get done on my system? Unfortunately, typical work is difficult to define because it varies with users' applications and environments.

As mentioned earlier, CPU suites CINT92 and CFP92 provide CPU performance measurements analogous to a sports car's lap times on 20 different race tracks. By contrast, the SDM suite provides (using two different loads) measurements of system capacity (e.g., acceleration, capacity) analogous to a loaded truck going up two different hills.

SDM Release 1 measures performance by systematically increasing the workload on the system. The gradual increase of the workload is achieved by increasing the number of concurrent copies (scripts) of the benchmark. A script under execution is commonly known as a *user process* in UNIX parlance. The terms *script*, *simulated user*, *user*, *user process* and *clone* are synonyms in this context.

As the system's workload increases, throughput typically increases to some maximum and then either stays constant or decreases. At a minimum, the workload on the system is increased until the system thrashes, the system's resources are exhausted (e.g., swap, proc table overflows, disk I/O bottleneck) and the system's performance degrades to 70 percent of the peak throughput value, or the system is subjected to twice the workload of the peak throughput value. SDM measures and reports this maximum throughput (peak value).

Obtaining the peak performance is an iterative process. System resources and tuning parameters are changed to obtain optimum (peak) system performance.

9.5.1.1 SPEC Metrics for SDM Benchmarks

SDET peak throughput is the maximum value of the throughput reached by monotonically increasing the number of concurrently executed SDET scripts while running the 057.sdet benchmark. It is measured in SDET scripts per hour.

KENBUS1 peak throughput is the maximum value of the throughput reached by monotonically increasing the number of concurrently executed KENBUS1 scripts while running the 061.kenbus1 benchmark. It is measured in KENBUS1 scripts per hour.

The performance metric is defined as the total amount of work done in a given time (scripts per hour). The scripts per hour metrics of two benchmarks cannot be compared, because their workloads are different and unrelated.

9.5.1.2 057.sdet

This benchmark is derived from an AT&T proprietary benchmark called Gaede. The workload is characterized as a software development in commercial UNIX environments. The benchmark is designed to find the throughput of a system when several processes are executed concurrently; each process executes a script from the scripts directory. Each script is a random collection of UNIX commands, and the frequency and number of UNIX commands are the same for all the scripts, as shown in Table 9.6. Each script runs this set of 141 UNIX commands in a random order.

Table 9.6 UNIX commands per workload: 057.sdet.

Command	Frequency	Command	Frequency	Command	Frequency
nroff/spell	1/1	echo	23	cat	4
ed	12	ls	6	ps	1
make/cc	1/6	cp	1	find	7
mkdir	4	mv	1	time	1
rm	21	pr	4	spell	1
rmdir	1	sh	1	touch	2
cd	33	diff	1	who	1
grep	2	wc	2	cpio	3

9.5.1.3 061.kenbus1

This benchmark is derived from the Monash University suite for benchmarking the UNIX system (MUSBUS Ver 5.2), which was developed by Ken J. McDonell. The workload is characterized as software development in a research and development UNIX environment.

In contrast to 057.sdet, this benchmark includes a concept of *think time*. The rate at which human users *peck* at keyboards is approximately three characters per second, based on a recent human-factors study by NCR. The frequency and the total number of UNIX commands are the same for all the scripts, as shown in Table 9.7. Each script runs this set of 18 UNIX commands in a random order.

Table 9.7 UNIX commands per workload: 061.kenbus1.

Command	Frequency	Command	Frequency	Command	Frequency
rm	5	cc	2	chmod	2
cp	2	ed	2	cat	1
export	1	grep	1	ls	1
mkdir	1				

9.5.2 Reading SDM Results

SDM benchmarks stress system resources: CPU, memory, disk I/O, and UNIX software. At peak throughput workloads, one or more of these resources become a bottleneck, and the only way to improve performance is to remove the bottleneck. To obtain optimal results on these benchmarks requires tuning the operating system and the disk I/O subsystem. Unlike the CINT92 and CFP92 metrics, SDM performance is highly dependent on the system's configuration (number of CPUs, controllers, disks, and the operating system). Table C.1 (Appendix C) shows recently reported results on SDM benchmarks.

System performance is characterized by a throughput curve. The SDM performance characterization provides more insights to the strengths and weaknesses of the system. The shape of the curve is especially telling. For example, on an ideal machine (infinite resources), the system's performance should never degrade.

A machine with a flat curve following a high peak is the closest a machine can come to being an ideal machine. In practice, a trapezoidal shape is expected. Obviously, high scripts per hour indicates a system that has excellent multitasking capabilities.

A curve that quickly (on light load) becomes nonlinear may indicate a poorly tuned system. A slow-rising curve may indicate a poorly tuned disk subsystem (possibly poor overlapping disk performance). More disks and better load distribution may improve the performance.

A sharp peak may indicate two possible bottlenecks in the system, namely memory contention (paging or swapping) or less overlap (concurrency) in disk I/O operations. Additional memory and disks may improve both the peak value and the shape of the curve.

A steep drop in throughput after a peak may indicate that the system is unable to sustain heavier workloads due to thrashing or other bottlenecks. It may also indicate that disk contention might be very high and/or that the system is paging and swapping heavily.

The width of a flat curve after the peak throughput value indicates the system's resiliency to sustained overloads. Thus, the length of the plateau is an indicator of CPU power. If the peak does not improve with a change to a faster CPU, or larger memory or faster controllers and disks, then there might be a problem with the operating system.

The workloads of 057.sdet and 061.kenbus1 are different, and the results from the two benchmarks should not be compared. The most exciting attribute of the SDM suite is that it provides a convenient framework for handling many different workloads.

9.6 SPEC Organization

SPEC has grown since October, 1988. Today, members include AT&T/NCR, Bull S.A., Compaq Computer Corporation, Control Data Corporation, Data General Corporation, Digital Equipment Corporation, Electronic Data Systems, Fujitsu Limited, HaL Computer Systems Inc., Hewlett-Packard Company, IBM Corporation, Intel Corporation, Intergraph Corporation, MIPS Computer

Systems, Motorola Inc., NeXT Computer Inc., Olivetti, Siemens Informationssysteme AG, Silicon Graphics Inc., Solbourne Computer, SUN Microsystems Inc., and Unisys Corporation.

SPEC's organization includes board members AT&T/NCR, HP, IBM, Intel, and SUN and an eight-member steering committee (AT&T/NCR, DG, DEC, HP, IBM, Intel, Siemens Nixdorf, and SUN). The steering committee's members are required to dedicate at least one full-time engineer to help develop SPEC benchmarks. The general membership votes for the board of directors, the steering committee's chair, and the final selection of the benchmarks in a suite.

Additionally, SPEC has a planning committee, an analysis committee, and release managers assigned to manage product development efforts. The planning committee works with the steering committee to determine product priorities and global issues for the future. The analysis committee examines the strengths and weaknesses of the benchmarks and makes recommendations to the steering committee. The product release managers work with the project leaders, the planning committee, the steering committee, and the publication organization to release quality SPEC products.

For additional information on SPEC products, please contact:
SPEC
c/o NCGA
2722 Merrilee Drive, Suite 200
Fairfax, VA 22031
Phone: (703) 698-9600, x318   FAX: (703) 560-2752

9.7 The Benchmark Development Process

Anyone with a sponsor can submit a benchmark to SPEC for its consideration (any steering committee member can be a sponsor). The sponsor works in cooperation with the submitter and assigns a project leader, who is responsible for solving porting problems and selling the merits of the benchmark initially to the steering committee and subsequently to the SPEC membership.

The steering committee members, along with the other SPEC members, develop, analyze, port, and test benchmarks on member platforms. Porting conflicts are resolved at bench-a-thons hosted by SPEC members. A bench-a-thon is an intense effort—five to six days and nights of working sessions during which engineering professionals from 10 to 15 member companies set aside their egos and resolve porting conflicts. When all porting problems are resolved, the steering committee votes on each benchmark. A two-thirds majority vote of the steering committee is necessary for a benchmark to be included in a suite. The suite is provided to the general membership for 30–60 days for further testing and evaluation. The general membership then votes on each benchmark for inclusion in the final product. All reasons for rejections are documented, and any rejected benchmark can be resubmitted for the next release.

9.8 Future Benchmarks

SPEC is working with the LADDIS (see below) group to release SPEC's version of the system file server (SFS) benchmark in 1993. A preliminary version of this benchmark was released in 1992 to obtain critical feedback from users. LADDIS was formed by representatives of Legato, Auspex, Data General, Interphase, and SUN Microsystems.

Written in C, the LADDIS benchmark is loosely based on Legato's 1989 *nhfsstone* benchmark. SFS minimizes NFS clients' platform sensitivity, offers improved accuracy of generated load, and normalizes such NFS implementation issues as file attribute and data caching across different vendors' platforms. The multiclient capability of SFS allows collections of NFS clients to be used to generate an aggregate load on an NFS server using one or more TCP/IP networks. This is important since a single client or a network may become saturated before a given server becomes saturated. Interoperability testing and porting efforts are in the final stages. At present, run and reporting rules for the SFS benchmarks are being debated, as are price/performance issues and models [Keith, 1992].

SPEC is also working on I/O intensive benchmarks, networking benchmarks, and new workloads to run under the SDM framework.

9.9 Concluding Remarks

The CINT92 and CFP92 suites provide a wider "SPECtrum" (20 colors) of performance characterization by adding new application benchmarks to both the integer and floating-point suites. SPEC strongly discourages a monochromatic view of performance characterization (e.g., SPECmark) and advocates reporting (full disclosure) of all 20 SPECratios and complete details of configuration (hardware, software, etc.). The benchmark results are reviewed by the steering committee for clarity, sanity, and rule compliance.

The measurement methodology of SDM Release 1 is not impacted by the number of CPU(s) and memory technologies. The unique feature of the SDM benchmarks is that their implementation provides a simple mechanism for introducing a variety of new workloads (e.g., network, database, and graphics applications).

Work with the SDM benchmarks showed that performance bottlenecks in the systems can be moved rather than removed. For example, adding CPUs will not help if disks are the bottlenecks, adding disks will not help if either the controller or CPU is the bottleneck, adding memory will not help if the operating system is the bottleneck, and nothing will help if the benchmark can't scale properly.

Even today, many people define performance as "hardware performance" (improvement in clock rate or cache organization). In fact, the user sees only the performance delivered by the system (combination of hardware, software—compilers, libraries, operating systems). Some performance *pundits* claim that performance improvement obtained by software is *cheating*. This is simply not true.

The fundamental difference between SPEC benchmark suites and other benchmarks is that vendors *tune* other benchmarks to show performance improvements, whereas with SPEC benchmark suites, vendors are challenged to improve compilers, preprocessors, and operating systems to improve performance. In fact, many performance improvements on SPEC are direct results of better optimizing compilers that benefit many other user applications.

Decision makers should not depend on a single benchmark or a single performance number in selecting systems. Their selection criteria should be based on the performance spectrum of speed, capacity, throughput, response time information on several standard benchmark suites, and performance on users' applications on a system configuration under consideration. A system's price, upgrade price, scalability, cost of support (hardware, software), interoperability, and future products are other important selection criteria.

SPEC's membership has debated for a long time whether it should implement a price/performance metric. The basic problem in doing so is that price is a transient marked by announcements and requires extensive investment in pricing models. Even a simple model (e.g., the list price of the reported configuration) requires much work and frequent updates. Dollars per SPECmetric is sometimes reported by vendors. For server configurations and system benchmarks (e.g., SFS), SPEC is formulating a simple price/performance metric.

SPEC is a very open and dynamic organization that listens to its critics and customers. It solicits new ideas and application benchmarks from critics of kernel and synthetic benchmarks and provides simple and inexpensive benchmark suites to users to validate performance claims and make informed comparisons.

9.10 Credits and Caveats

My sincere thanks to SPEC members who worked hard at developing these benchmark suites. I am grateful to Bob Cmelik (SUN) who developed SpixTools. I am indebted to release managers Dr. Sivram (Ram) Chelluri (AT&T/NCR) and Dr. Krishna Dronamraju (AT&T/NCR) for SDM, Jeff Reilly (Intel) for CINT92 and CFP92, Bruce Keith (DEC) for SFS/LADDIS, Bill Keatts (CDC) and John Freeman (CDC) for I/O suite.

I want to thank steering committee chairs Bob Novak (MIPS), Jack McClurg (HP), Bud Funk (Unisys), and Walter Bays (SUN) who focused on difficult issues and have helped solve many problems. I can't adequately thank bench-a-thon hosts for their very generous contributions: John Mashey (MIPS), John Laskowski (IBM), Jeff Garelick (IBM), Rod Skinner (Intel), Dr. Chelluri (AT&T/NCR) and John Dooley (Motorola), Frank Lee (Compaq), Larry Gray (HP), Doug Fears (NCR), Bhagyam Moses (DEC), Paula Smith (DEC), and Tom Morgan (DG). I also want to thank Dr. Reinhold Weicker (Siemens Nixdorf) who chaired the benchmark analysis committee. Most of SPEC's development work consisted of creating usable tools, and for that effort I extend thanks to Anne Skamrock (Solbourne), Vicki Scott (MIPS), Jeff Garelick (IBM), Todd Clayton (IBM), and Jeff Reilly (Intel).

Most of the data presented herein are from SPEC newsletters, and I want to thank the authors and the editors: Subra Balan (IBM), Walter Bays (SUN), and Alexander Carlton (HP). I am grateful to SUN Microsystems and IBM Corporation for supporting this effort.

A number of my colleagues and friends read drafts and helped improve readability. Thanks are due to Varun Mehta, Walter Bays, and Nhan Chu at SUN and to Mike Zarrillo, Sohel Saiyed, and Jeanette McWilliams at IBM. I want to thank Jim Gray (DEC), Bob Klingensmith (copyeditor), and Carol Leyba (Production Editor, Morgan Kaufmann) who translated this chapter from *Sanskrit* to *English*. Finally, I want to thank my wife Evelyn for enduring lonely evenings and weekends while I worked on this chapter.

Performance numbers change frequently, so users should always obtain the latest available results from SPEC. Appendix Tables A.1, A.2, B.1, B.2 and C.1 show recently reported results (December, 1991, March, 1992, June, 1992, September, 1992, and December, 1992). Appendix D contains definitions of various SPEC metrics.

The opinions expressed in this article are not necessarily endorsed by SPEC or its members.

References

Austin, Todd M. & Sohi, Gurindar (1992). Dynamic Dependency Analysis Of Ordinary Programs, pages 342–347, 1992 ACM.

Bailey, D. H. & Barton, J. T. (1985). The NAS kernel benchmark program. NASA Tech. Memo. 86711, Aug. 1985.

Berry, M., Cybenko, G. & Larson, J. (1991). Scientific benchmark characterizations. *Parallel Computing*, 17:1173–1194 (December 1991).

Betz, David Michael (1988). 1.6, Revision 1.0, November 1988.

Carlton, Alexander (1992). CINT92 and CFP92 homogeneous capacity method offers fair measure of processing capacity. *SPEC Newsletter*, 4 (2), June 1992.

Curnow H. J. & Wichmann, B. A . (1976). A synthetic benchmark. *The Computer Journal*, 19 (1).

Dixit, Kaivalya M. (1991a). Truth in SPECmarking. SunWorld, 4 (9), September 1991.

Dixit, Kaivalya M. (1991b). CINT92 and CFP92 benchmark descriptions. SPEC Newsletter, 3 (4), December 1991.

Dixit, Kaivalya M. (1991c). The SPEC benchmarks. Parallel Computing, 17:1195–1205, December 1991.

Dixit, Kaivalya M. (1991d). Interpreting SPEC Release 1 benchmark results. SPEC Newsletter, 3 (4), December 1991.

Dixit, Kaivalya M. (1992). New CPU benchmark suites from SPEC. compcon92, Digest of papers, pages 305–310.

Dixit, Kaivalya M. & Novak, Robert (1990). SPEC refines reporting format to ensure level playing field. SPEC Benchmark Results, 2 (1), Winter 1990.

Dixit, Kaivalya M., & Reilly, Jeff (1991). SPEC developing new component benchmark suites. SPEC Newsletter, 3 (4), December 1991.

Doduc, Nhuan (1989). FORTRAN execution time benchmark. Framentec, V29, March 1989.

Dongarra, J. J. (1983), Performance of various computers. Computer Architecture News, 1983.

Dronamraju, S. Krishna, Balan, Subra, & Morgan, Tom (1991). System analysis and comparison using SPEC SDM 1. SPEC Newsletter, 3 (4), December 1991.

Dronamraju, S. Krishna, Naseem, Asif, & Chelluri, Sivram (Ram) (1990). System level benchmarking. SPEC Newsletter, 2 (4), Fall 1990.

Flemming, P. J. & Wallace, J. J. (1986). How not to lie with statistics: The correct way to summarize benchmark results. Communications of the ACM 29, 3 (Mar. 1986): 218–221.

Greenfield, Mike, Raynoha, Paul, & Bhandarkar, Dileep (1990). SPEC adopts new methodology for measuring system throughput. SPEC Newsletter, 2 (2), Spring 1990.

Hennessy, J. L. & Patterson, D. A. (1990). Computer Architecture, A Quantitative Approach. Morgan Kaufmann Publishers, Inc.

Jain, Raj (1991). The Art of Computer Systems Performance Analysis, pp. 186–192. John Wiley & Sons.

Keatts, Bill, Balan, Subra, & Parady, Bodo (1991). matrix300 Performance Concerns. SPEC Newsletter, 3 (4), December 1991.

Keith, Bruce (1992). LADDIS file server benchmark. SPEC Newsletter, 4 (1), March 1992.

Mashey, John (1990). SPEC results help normalize vendor Mips-ratings for sensible comparison. SPEC Newsletter, 2 (3), Summer 1990.

Novak, Robert E. (1991). SPEC defines two new measures of performance. SPEC Newsletter, 3 (1), Winter 1991.

Phillip, Michael J. (1992). Performance issues for the 88110 RISC microprocessor. compcon92, Digest of papers, pp. 163–168.

Savedra-Barrera & Rafael H. (1990). The SPEC and Perfect Club benchmarks: Promises and limitations. Hot Chips-2 Symposium, August 20, 1990.

Smith, J. E. (1988). Characterizing computer performance with a single number. Communications of the ACM, 10 Oct. 1988, pp. 1202–1206.

SPEC Benchmark Suite Release 1.0 (1990). Staff Report, SPEC Newsletter, 2 (2), Spring 1990.

Stallman, Richard M. (1989). GNU CC Compiler Version 1.35. Free Software Foundation, 675 Mass Ave., Cambridge, MA, April 1989.

Stephens, Chriss, Cogswell, Bryce, Heinlein, John, Palmer, Gregory, & Shen, John (1991). Instruction level profiling and evaluation of the IBM RS/6000. Computer Architecture News, 19 (3):180–189, May 1991.

U. C. Berkeley (1987). CAD/IC Group, EECS/ERL of U. C. Berkeley, SPICE2G.6, Release date March 1987.

U. C. Berkeley (1988). EECS Department of U. C. Berkeley, Espresso Version #2.3, Release date 01/31/88.

U. C. Berkeley (1985). The Industrial Liaison Program, Eqntott #V9, released 1985.

Weicker, Reinhold P. (1991). A detailed look at some popular benchmarks. Parallel Computing, 17:1153–1172, December 1991.

Weicker, Reinhold, & Bays, Walter (1992). Point: Why SPEC should burn (almost) all flags: Counterpoint: Defending the flag. SPEC Newsletter, 4 (2), June 1992.

Trademark Information

SPEC{ratio, int89, fp89, mark89, Thruput89, intThruput89, fpThruput89} are trademarks of Standard Performance Evaluation Corporation (SPEC).
SPEC{int92, fp92} are trademarks of SPEC.
SPECrate_{int92, fp92} are trademarks of SPEC.
UNIX is a registered trademark of Unix System Laboratories (USL).
VAST is a registered trademark of Pacific Sierra Research Corporation.
KAP is a registered trademark of Kuck and Associates.
SPARC, SunOS, SPARCstation 2 are registered trademarks of SUN Microsystems.
AIX, RISC System 6000, POWER are registered trademarks of IBM Corporation.
PA-RISC and HP-UX are registered trademarks of Hewlett-Packard/Apollo.
AViion and DG/UX are registered trademarks of Data General Corporation.
Sinix is a registered trademark of Siemens Nixdorf Information Systems.
DECsystem, VAX, VMS are registered trademarks of Digital Equipment Corporation.
Intel, i860 and i486 are registered trademarks of Intel Corporation.
DESKPRO 486/50L is a registered trademark of Compaq Corporation.
EPIX and InfoServer are registered trademarks of Control Data Corporation.

Appendix A

Table A.1 CINT92 and CFP92 results (Page of).

Model	MHz	Cache	MB	SPEC int92	SPEC fp92
ALR Business 486 ASX	20	8I+D+64(E)	16	10.7	4.9
Compaq Deskpro/66M	33/66	8I+D+256(E)	16	32.2	16.0
Compaq Deskpro/50M	25/50	8I+D+256(E)	16	25.7	12.2
Compaq 486/33L	33	8I+D+128(E)	16	18.2	8.3
Compaq Deskpro/i	25	8I+D+64(E)	25	14.2	6.7
Compaq 486s/16M	16	8I+D	16	9.3	4.3
CDC 4680-312	80	64I+16D+2M(E)	128	-	61.9
CDC 4680-313	80	64I+16D+2M(E)	256	61.9	-
CDC 4330-300	33	32I+32D	128	24.5	25.7
CDC 4360-300	33	64I+64D	256	24.9	26.7
CDC 4680	60	64I+64D+512(E)	256	40.6	45.1
DG AV/4100	20	16I+16D	64	13.1	-
DG AV/4300	25	16I+16D	128	17.4	-
DG AV/4600	33.3	16I+16D	128	22.6	-
DG AV/4605	33	64I+32D	128	26.1	-
DG AV/5225 (#2)	25	64I+64D	256	20.3	-
DG AV/6240 (#4)	25	64I+64D	320	20.1	-
Decstation/20	20	64I+64D	40	13.7	14.8
Decstation/25	25	64I+64D	16	15.8	17.5
DECstation 5000/33	33	64I+128D	24	20.9	23.4
DECstation 5000/125	25	64I+64D	32	16.0	17.5
DECstation 5000/133	33	64I+128D	64	20.1	23.5
DECstation 5000/240	40	64I+64D	64	27.3	29.9
DECsystem 5900	40	64I+64D	128	27.3	29.9
DEC VAX 4000/60	18ns	256I+D	104	-	10.4
DEC VAX 4000/90	71.43	2I+8D	32	-	30.2
DEC VAX 6000/410	28ns	128I+D	256	-	7.1
DEC VAX 6000/510	16ns	512I+D	256	-	13.3
DEC VAX 6000/610	12ns	2M I+D	256	-	39.2
HP 9000/705	35	32I+64D	16	21.9	33.0
HP 9000/710	50	32I+64D	16	31.6	47.6

HP 9000/720	50	128I+256D	16	36.4	58.2
HP 9000/730	66	128I+256D	64	52.4	86.8
HP 9000/750	66	256I+256D	32	48.1	75.0
HP 9000/807	32	32I+64D	16	20.2	-
HP 9000/817	48	64I+64D	32	31.4	-
HP 9000/837	48	256I+256D	64	34.9	-
HP 9000/857	48	256I+256D	64	34.8	-
HP 9000/867	64	256I+256D	64	45.6	-
HP 9000/877	64	256I+256D	64	45.8	-
HP 9000/887	96	256I+256D	192	78.2	-
HP 9000/887S	96	256I+256D	192	-	141.6
IBM 6000/220	33	8I+D	32	15.9	22.9
IBM 6000/320H	25	8I+32D	32	20.9	39.4
IBM 6000/340	33	8I+32D	32	27.7	51.9
IBM 6000/350	41.67	8I+32D	32	34.6	65.0
IBM 6000/520H	25	8I+32D	32	20.9	39.6
IBM 6000/530H	33	8I+64D	32	28.2	57.5
IBM 6000/560	50	8I+64D	64	42.1	85.5
IBM 6000/970	50	32I+64D	64	47.1	93.6
IBM 6000/580	62.5	32I+64D	64	59.1	124.8
IBM 6000/980	62.5	32I+64D	64	59.1	124.8
XPRESS 486DX50	50	8I+D+256(E)	56	30.1	14.0
SGI Crimson	50	8I+8D+1M (E)	256	58.3	61.5
SGI Indigo R4000	50	8I+8D+1M (E)	64	57.6	60.3
SGI IRIS Indigo	33	32I+32D	32	22.4	24.2
SNI MX300-75	50	8I+D+256 (E)	64	28.4	-
Solbourne 5E/901	40.1	128I+D	128	22.2	23.4
Solbourne 6/901	33.33	20I+16D+1M(E)	128	44.0	52.5
SPARCstation 10/30	36	20I+16D	64	44.1	49.4
SPARCstation 10/30	36	20I+16D+256(E)	64	45.2	-
SPARCstation 10/41	40.33	20I+16D+1M (E)	64	53.2	63.4
SPARCstation IPX	40	64I+D	32	21.8	21.5
SPARCstation LT	25	64I+D	16	15.3	13.7
SPARCserver 490	33	128I+D	32	21.7	19.5
SPARCstation ELC	33	64I+D	24	18.2	17.9
SPARCstation IPC	25	64I+D	24	13.8	11.1
SPARCstation 2	40	64I+D	32	21.8	22.7

Notes: 1. Separate instruction and data cache values are reported in kilobytes, e.g., 32I + 64D.
2. Combined instruction and data cache values are reported in kilobytes, e.g.,

128I+D.

3. External cache are reported in kilobytes, e.g., 256(E); or in megabytes 1 M(E).
4. Memory is reported in megabytes (MB).

Table A.2 CINT92 and CFP92 Capacity Results (SPECrates).

Model	MHz	Cache	MB	SPECrate_ int92	SPECrate_ fp92
CDC 4680	60	64I+64D+512(E)	256	-	1130.4
CDC 4680(#2)	60	64I+64D+512(E)	256	-	2231.7
CDC 4680-312 (#2)	80	64I+16D+2M(E)	128	2674.0	2676.0
CDC 4680-312 (#4)	80	64I+16D+2M(E)	128	5300.0	5523.0
DG AV/4320 (#2)	25	16I+16D	128	732.6	-
DG AV/4620 (#2)	33	16I+16D	128	933.4	-
DG AV/4625(#2)	33	64I+32D	128	1097.9	-
DG AV/5225(#2)	25	64I+64D	256	868.1	-
DG AV/6240(#4)	25	64I+64D	384	1590.7	-
DG AV/6280(#8)	25	64I+64D	384	3245.0	-
HP 9000/807S	32	32I+64D	64	523.0	876.0
HP 9000/817S/827S	48	64I+64D	64	816.0	1335.0
HP 9000/837S/847S	48	256I+256D	64	890.0	1483.0
HP 9000/867S/877S	64	256I+256D	64	1201.0	1937.0
HP 9000/870/100	50	512I+512D	64	835.4	-
HP 9000/870/200(#2)	50	512I+512D	64	1514.9	-
HP 9000/870/300(#3)	50	512I+512D	64	2051.3	-
HP 9000/870/400(#4)	50	512I+512D	64	2478.8	-
HP 9000/890	60	2M I+2M D	256	1215.0	1180.0
HP 9000/890 (#2)	60	2M I+2M D	256	2253.0	2360.0
HP 9000/890 (#3)	60	2M I+2M D	256	3306.0	3529.0
HP 9000/890 (#4)	60	2M I+2M D	256	4301.0	4685.0
HP 9000/887S	96	256I+256D	192	1854.0	3490.0
SNI MX500-90/2 (#4)	50	8I+D+512(E)	128	1919.0	-
SNI MX500-90/2 (#8)	50	8I+D+512(E)	128	3474.0	-
SNI MX500-90/2 (#12)	50	8I+D+512(E)	128	4870.0	-
SNI RM600 15/25 (#4)	37	256I+256D+4M(E)	256	2298.0	2016.0
SNI RM600 15/25 (#8)	37	256I+256D+4M(E)	256	4506.0	3952.0
SNI RM600 15/25 (#12)	37	256I+256D+4M(E)	256	6527.0	5403.0
Solbourne 6/901	33.33	20I+16D+1M(E)	128	1028.0	1236.0
Solbourne 6/908 (#8)	33.33	20I+16D+1M(E)	256	6664.0	8693.0
SPARCstation 2	40	64I+D	32	517.0	537.0

SPARCstation 10/30	36	20I+16D	64	1046.0	1253.0
SPARCstation 10/41	40.33	20I+16D+1M (E)	64	1263.0	1503.0
SPARCserver600/120 (#2)	40	64I+D	128	1043.0	1066.0
SPARCserver600/140 (#4)	40	64I+D	128	1847.0	1930.0

Note: (#N) = Number of CPUs; e.g., (#3) = three CPUs.

Appendix B

Table B.1 SPEC Release 1 results (Page of).

Vendor Model	MHz	Cache	SPEC mark89	SPEC int89	SPEC fp89
Compaq 486/50L	50	8I+D+256E	19.3	27.6	15.2
DEC DECst. 5000-133	33	64I+128D	25.5	20.9	29.1
DEC DECst. 5000-240	40	64I+64D	32.4	27.9	35.8
DEC DECstation 20	20	64I+64D	16.3	13.5	18.4
DEC DECstation 25	25	64I+64D	19.1	15.7	21.7
DEC DECstation 33	33	64I+128D	26.5	23.3	29.0
DEC DECsystem 5900	40	64I+64D	32.8	28.4	36.2
DEC VAX 4000-90	71.43	2I+8D	32.7	26.7	37.4
DEC VAX 4000-200	35ns	64I+D	5.6	5.4	5.8
DEC VAX 4000-300	28ns	128I+D	9.2	8.4	9.8
DEC VAX 4000-400	16ns	128I+D	22.3	17.2	26.6
DEC VAX 4000-500	14ns	128I+D	30.7	24.9	35.4
DEC VAX 4000-600	12ns	512I+D	41.1	31.8	48.7
DEC VAX 6000-410	28ns	128I+D	8.5	7.8	9.0
DEC VAX 6000-510	16ns	512I+D	15.6	14.7	16.3
DEC VAX 6000-610	12ns	2M I+D	42.1	31.5	51.1
DEC VAX 7000-610	11ns	4M I+D	46.6	34.0	57.6
DG AV 530	33	16I+16D	18.2	24.1	15.1
HP 700-705	35	32I+64D	34.6	24.6	43.4
HP 700-710	50	32I+64D	49.7	35.4	62.4
HP 700-720	50	128I+256D	66.5	42.2	89.9
HP 700-750	66	256I+256D	86.6	55.7	116.1
IBM 6000/220	33	8I+D	25.9	17.5	33.7
IBM 6000/320H	25	8I+32D	43.4	21.8	68.8
IBM 6000/340	33	8I+32D	56.6	28.8	88.7
IBM 6000/350	42	8I+32D	71.4	36.2	112.3
IBM 6000/520H	25	8I+32D	43.5	21.8	58.9
IBM 6000/530H	33	8I+64D	59.9	29.2	96.7
IBM 6000/550	42	8I+64D	75.9	36.8	123.0
IBM 6000/560	50	8I+64D	89.3	43.8	143.6
IBM 6000/970	50	32I+64D	100.3	49.3	160.9
IBM 6000/580	62.5	32I+64D	126.4	61.3	204.8
IBM 6000/980	62.5	32I+64D	126.4	61.3	204.8
Ingr InterServe 6605	40	128I+D	40.0	23.1	57.7
Intel AL860XP-50	50	16I+16D	31.4	53.1	43.0
SGI Crimson	50	8I+D+1M (E)	70.4	60.6	77.8
SGI INDIGO R4000	50	8I+D+1M (E)	70.2	61.1	77.0

SNI MX300-60/5	50	8I+D +256(E)	17.0	24.8	13.2
SNI Targon/31-25	25	4I+4D	8.8	14.3	6.4
SPARCserver 490	33	128I+D	22.7	20.7	24.1
SPARCstation 2	40	64I+D	25.0	21.7	27.4
SPARCstation ELC	33	64I+D	20.3	18.0	22.0
SPARCstation IPC	25	64I+D	13.5	12.8	14.0
SPARCstation LT	25	64I+D	16.7	14.9	18.1
SPARCstation IPX	40	64I+D	24.4	21.7	26.5

Table B.2 SPEC Thruput89 Method A* results.

Vendor Model	**MHz**	**Cache**	**SPEC thruput89**	**SPECint thruput89**	**SPECfp thruput89**
CDC Infoserver 4375(#2)	40	64I+64D+1M(E)	2@28.8	2@24.3	2@32.2
CDC Infoserver 4375(#4)	40	64I+64D+1M(E)	4@28.2	4@24.0	4@31.4
CDC Infoserver 4375(#8)	40	64I+64D+1M(E)	8@23.1	8@20.7	8@24.0
CDC Infoserver 4680(#2)	60	16I+64D+512(E)	2@54.4	2@44.3	2@62.4
CDC Infoserver 4680(#3)	60	16I+64D+512(E)	3@53.6	3@44.1	3@61.1
CDC Infoserver 4680(#4)	60	16I+64D+512(E)	4@51.4	4@42.3	4@58.5
DG AV 4320(#2)	25	64I+16D	2@12.7	2@17.2	2@10.3
DG AV 4620(#2)	33.3	16I+16D	2@16.6	2@22.7	2@13.5
DG AV 4625(#2)	25	64I+64D	2@15.0	2@20.1	2@12.3
DG AV 6240(#4)	25	64I+64D	4@12.7	4@18.9	4@9.8
DEC VAX 6000-660(#6)	12ns	2M I+D	6@32.8	6@24.0	6@40.4
DEC VAX 7000-620 (#2)	11ns	4M I+D	2@42.0	2@30.7	2@51.7
DEC VAX 7000-640 (#4)	11ns	4M I+D	4@39.4	4@28.1	4@49.2
Solbourne 5/7021(#2)	33.3	128 I+D	2@18.0	2@18.2	2@17.9
Solbourne 5/7041(#4)	33.3	128 I+D	4@17.4	4@17.4	4@17.3
Solbourne 5E/903(#3)	40.1	128 I+D	3@20.4	3@21.0	3@20.1
Solbourne 5E/908(#8)	40.1	128 I+D	8@17.3	8@18.8	8@16.4
SPARCserver 600/120(#2)	40	64 I+D	2@25.4	2@21.4	2@28.5
SPARCserver 600/140(#4)	40	64 I+D	4@22.8	4@19.4	4@25.4

Notes: 1. (#N) = Number of CPUs, e.g., (#6) = 6 CPUs.
2. * = Method A = Homogeneous Load.

Appendix C

Table C.1 SPEC SDM 1 results.

Vendor Model	MHz	Cache	057.sdet scripts/hr	061.kenbus1 scripts/hr	#Contr.& #disks
AT&T/NCR Serv E 660 (#4)	66	8I+D+512(E)	770.1	-	4/13
Compaq 486/50L	50	8I+D+256(E)	188.6	-	1/3
CDC 4330-300	33	32I+32D	222.4	-	1/3
CDC 4350-300	33	64I+64D	242.9	-	2/5
CDC 4360-300	33	64I+64D	246.2	1773.2	2/5
CDC Infoserver 4680	60	64I+16D+512(E)	397.4	-	2/8
CDC Infoserver 4680(#2)	60	64I+16D+512(E)	686.3	3763.4	5/17
CDC Infoserver 4680(#4)	60	64I+16D+512(E)	1042.9	-	4/13
Dell 450-DE	50	8I+D+128(E)	206.2	1296.5	2/4
DEC DECstation/20	20	64I+64D	-	301.8	1/3
DEC DECstation/25	25	64I+64D	62.3	566.3	1/3
DEC DECstation/133	33	64I+128D	90.6	703.7	1/5
DEC DECstation/240	40	64I+64D	222.5	1212.3	1/5
DEC DECsystem 5900	40	64I+64D	233.4	1299.8	1/5
DEC 433MP (#3)	33	256 I+D	298.3	1406.4	3/15
DEC 433MP (#4)	33	256 I+D	374.0	1712.3	3/15
HP 9000/867S	64	256I+256D	-	2244.0	5/16
IBM 6000/320H	25	8I+32D	140.1	-	1/5
IBM 6000/340	33	8I+32D	184.6	-	1/5
IBM 6000/350	41.67	8I+32D	227.4	1479.7	1/5
IBM 6000/520	25	8I+64D	140.3	-	1/5
IBM 6000/530H	33	8I+64D	188.9	-	1/5
IBM 6000/550	41.67	8I+64D	236.2	-	1/5
IBM 6000/560	50	8I+64D	271.3	1899.0	2/6
IBM 6000/970	50	32I+64D	326.0	2238.9	2/8
Intel XPRESS 486DX50	50	8I+D+256(E)	290.8	-	2/4
Intel XPRESS 486DX50	50	8I+D+256(E)	269.1	-	3/8
SGI 4D/35	36	64I+64D	236.5	-	1/8
SGI Indigo R4000	50	8I+8D+1M(E)	232.1	-	1/3
SGI IRIS Indigo 4D/RPC	33	32I+32D	182.6	-	1/7
Unisys U6000/65	33	8I+D+256(E)	-	902.0	1/7
Unisys U6000/65(#2)	33	8I+D+256(E)	-	1595.0	1/6

Appendix D. Definition of SPEC Metrics

SPEC Metrics for CINT92 and CFP92 Suites

By definition the SPECint92 and SPECfp92 for a VAX-780 are 1.

SPEC Reference Time is the time it takes a DEC VAX 780 to run each individual benchmark in the suites. The SPEC Reference Time for both suites is shown below:

CINT92 Benchmark Names	SPEC Reference Time Seconds
008.espresso	2,270
022.li	6,210
023.eqntott	1,100
026.compress	2,770
072.sc	4,530
085.gcc	5,460

CFP92 Benchmark Names	SPEC Reference Time Seconds
013.spice2g6	24,000
015.doduc	1,860
034.mdljdp2	7,090
039.wave5	3,700
047.tomcatv	2,650
048.ora	7,420
052.alvinn	7,690
056.ear	25,500
077.mdljsp2	3,350
078.swm256	12,700
089.su2cor	12,900
090.hydro2d	13,700
093.nasa7	16,800
094.fpppp	9,200

SPECratio for a benchmark is the quotient derived from dividing the SPEC Reference Time by a particular machine's elapsed time.

Geometric Mean (GM) of *n* items is defined as the *n*th root of the product of *n* items. For example, the GM of the following six SPECratios—24.8, 25.0, 24.4, 28.7, 26.2 and 27.6—is the sixth root of the product (24.8 * 25.0 * 24.4 * 28.7 * 26.2 * 27.6) = 26.1.

SPECint92 is the GM of the six SPECratios (008.espresso, 022.li and 023.eqntott, 026.compress, 072.sc and 085.gcc) of the CINT92 suite.

SPECfp92 is the GM of the 14 SPECratios (013.spice2g6, 015.doduc, 039.wave5, 042.fpppp, 047.tomcatv, 048.ora, 052.alvinn, 056.ear, 077.mdljsp2, 078.swm256, 089.su2cor, 093.nasa7 and 094.fpppp) of the CFP92 suite.

SPECrate is a capacity measure for an individual benchmark. It is a function of the number of copies run, a reference factor for normalization and the elapsed time. This provides a measure in jobs per second and is then scaled to jobs/week (which allows the reference machine DEC VAX 780 to have a measure somewhat greater than 1). The reference factor for a given benchmark is calculated by dividing the SPEC reference time into the longest SPEC reference time. 25,500 seconds (for benchmark 056.ear). The number of seconds in a week is 604,800.

Thus for a given benchmark, the SPECrate is calculated as:

SPECrate = {# of copies run * (benchmark reference time / 25,500 seconds) * (604,800 / elapsed time) }.

SPECrate_int92 is the GM of the SPECrates from the six benchmarks in CINT92.

SPECrate_fp92 is the GM of the SPECrates from the 14 benchmarks in CFP92.

SPEC Metrics for SPEC SDM Release 1 Suite

SDET Peak Throughput is the maximum value of the throughput reached by monotonically increasing the number of concurrently executed SDET scripts, while running the 057.sdet benchmark. It is measured in SDET scripts/hour.

KENBUS1 Peak Throughput is the maximum value of the throughput reached by monotonically increasing the number of concurrently executed KENBUS1 scripts, while running the 061.kenbus1 benchmark. It is measured in KENBUS1 scripts/hour.

SDET scripts/hour and KENBUS1 scripts/hour are not comparable.

SPEC Metrics for SPEC Release 1 Suite

By definition the SPECint89, SPECfp89, and SPECmark89 for a VAX 780 are 1.

SPEC Reference Time is the time it takes a DEC VAX 780 to run each individual benchmark in the suite. The SPEC Reference Time for SPEC Release 1 is shown below:

SPEC Release 1 Benchmark	SPEC Reference Time

Names	Seconds
001.gcc	1,481.5
008.espresso	2,266.0
013.spice2g6	23,951.4
015.doduc	1,863.0
020.nasa7	20,093.1
022.li	6,206.2
023.eqntott	1,100.8
030.matrix300	4,525.1
042.fpppp	3,038.4
047.tomcatv	2,648.6

SPECint89 is the GM of the four SPECratios (001.gcc, 008.espresso, 022.li and 023.eqntott) of the SPEC Release 1 suite.

SPECfp89 is the GM of the six SPECratios (013.spice2g6, 015.doduc, 020.nasa7, 030.matrix300, 042.fpppp and 047.tomcatv) of the SPEC Release 1 suite.

SPECmark89 is the GM of the ten SPECratios of the SPEC Release 1 suite.

SPECintThruput89 is the GM of the four Thruput Ratios from the four integer benchmarks of the SPEC Release 1 suite.

SPECfpThruput89 is the GM of the six Thruput Ratios from the six floating-point benchmarks of the SPEC Release 1 suite.