Parallel Computing 17 (1991) 1195–1209
North-Holland

The SPEC benchmarks

Kaivalya M. Dixit

Sun Microsystems, Inc., Mountain View, CA, USA

Received April 1991
Revised June 1991

Abstract

Dixit, K.M., The SPEC benchmarks, Parallel Computing 17 (1991) 1195–1209.

This paper characterizes problems with popular benchmarks used to measure system performance and describes the history, structure, mission, future, and workings of an evolving standard in characterizing systems performance.

Systems Performance Evaluation Cooperative (SPEC) represents an effort of major computer companies to create a yardstick to measure the performance of computer systems. This paper compares results of three architectures with traditional performance metrics and metrics developed by SPEC. It discusses strengths and weaknesses of SPEC and warns users about the dangers of *single number* performance characterization.

Keywords. Benchmarks; system performance; performance metric; SPEC; performance results.

1. History

SPEC is a new, evolving standard in benchmarking. SPEC, a non-profit corporation, was founded in October 1988 by Apollo, Hewlett-Packard, MIPS and Sun Microsystems. Today, it is endorsed by 21 major computer vendors whose common goal is "To provide the industry with a realistic yardstick to measure the performance of advanced computer systems" – and to educate the consumer about their products' performance characteristics [17].

SPEC creates, maintains, distributes, and endorses a standardized set of application oriented programs to be used as benchmarks.

1.1. Structure

SPEC has significantly grown since October 1988. Today SPEC numbers include: AT & T, Bull, CDC, Compaq, DG, DEC, Fujitsu, HP, IBM, Intel, Intergraph, MIPS, Motorola, NCR, Prime, SGI, Siemens, Solbourne, Stardent, Sun Microsystems and Unisys.

SPEC is organized as: Board Members (HP, IBM, Intel, MIPS and Sun), Steering Committee (SC) members (AT & T, CDC, DEC, HP, IBM, MIPS, Motorola, Sun and Unisys), and general members. SC members are required to dedicate at least one full-time person to developing SPEC benchmarks. SC members are elected by the general membership. SPEC members hold monthly meetings, open to the public. It costs $15 000 to join SPEC which includes first year dues of $5000.

1.2. SPEC's Mission

SPEC's mission and vision is to provide the industry with a realistic yardstick to measure the performance of advanced computer systems. SPEC strives to achieve this with a few simple and pragmatic rules:

- Defend, attack or compete with no one
- Not favor any particular architecture
- Pursue open debate and disclosure.

Within a year of its inception, the SPEC Release 1.0 benchmark suite was announced in October 1989. This suite comprises four integer and six floating point benchmarks. These benchmarks were selected after evaluating and analyzing about 45 candidates submitted or sponsored by the member companies. The steering committee and general members unanimously voted to include these ten benchmarks.

Release 1.0 is available in Unix versions and VAX/VMS. SPEC also provides information on the application profiles of each of the individual benchmarks in the suite.

2. Why SPEC?

Traditional benchmarks have failed to characterize the system performance of modern computer systems. Some of these benchmarks measure component level performance. These measurements are routinely published as system performance. Historically vendors have characterized performance of their systems in a variety of confusing metrics. This, in part, is due to lack of credible performance information, agreement and leadership amongst competing vendors. The system performance is a complex measure of the system's ability to perform a variety of operations like executing integer, floating-point, and I/O operations.

Many vendors report performance results on the popular benchmarks like: *Dhrystones, Whetstones, Linpack, Perfect Club Suite, Livermore Loops, NAS kernels etc.* Below we describe a few of the popular benchmarks.

2.1. Dhrystones

This is a popular benchmark developed by Reinhold Weicker in 1984 [23]. Even though the original version was written in *Ada*, the most popular versions are in the *C* language. This benchmark spends a significant time on *string* functions. It was designed to measure the integer performance of small machines with simple architectures. RISC [1] machines generally beat CISC machines on this benchmark because of the large number of registers and the locality of code and data. The working set of the program fits in the cache of most modern machines. Almost all instruction and data accesses are cache hits; therefore the benchmark yields an inflated measure of the CPU and cache performance [22]. Some optimizing compilers produce code specific for this benchmark (e.g. in-lining *string functions*) that further inflates performance [25]. The new version corrects some of the problems but vendors continue to publish the *best* numbers with the older version. It is popular with vendors because it overstates the performance of their system. The performance metric is *Dhrystones / second.*

2.2. Whetstones

This is a popular floating point benchmark originally written in *Algol 60* by H.J. Curnow and B.A. Wichman in 1976 [5]. Today the most popular versions are in the *Fortran* and *C*

[1] Refer to the glossary in Appendix A.

language. It is a synthetic benchmark. The benchmark contains ten modules that perform a variety of numerical computations (e.g. arrays, trigonometric functions etc.). Unfortunately many versions exist in the field and the performance between the versions can vary significantly. Optimizing compilers produce code specific to this benchmark that avoids the intended computations (e.g. change *sqrt* to *multiplication by 2*) [10]. A significant portion of the run-time is spent in the math library and trigonometric functions and so optimized implementation of these routines. The benchmark is small and sensitive to the ordering of library modules and cache. The instruction mix does not represent today's programs [10]. The performance metrics are *Single Precision or Double Precision Whetstones/second.*

2.3. Linpack

Linpack is a collection of linear algebra subroutines authored by Jack Dongarra in 1976 [8]. It was written in *Fortran* and is widely used to characterize the floating point performance of machines. This benchmark operates on a 100 × 100 matrix (*a large matrix in 1976*). Today, the program fits comfortably in cache of many machines. Linpack code that executes on matrices of the order 300 × 300 or 1000 × 1000 alleviate the small size problem. The performance is characterized by Millions of Floating Point Operations per Second (*MFLOPS*) executed by the benchmark. The benchmark is sensitive to cache size, memory (size and organization), compiler optimizations and register allocation tricks used by smart preprocessors [3]. The code is highly vectorizable and is like many scientific applications. It does not accurately characterize the floating point performance because of the differences in instruction set architectures and its sensitivity to memory systems. LINPACK can be characterized as having a high percentage of floating point operations. The cache size and memory latency determines the execution time and hence the performance. Studies indicate that some computer systems spend about 50% of the execution time in memory accesses. This benchmark is as much a memory system benchmark as it is a floating point benchmark. If the memory latency is reduced by a smart preprocessor or a special library, the measure of the floating point performance increases due to reduction in the executing time [14]. It is popular with many vendors. The performance metrics are *Single Precision or Double Precision MFLOPS.*

These benchmarks have had a long life considering the dynamics and rapid technological advances in the computer industry. They have served us well.

2.4. And others...

Additionally, there are many in-house vendor benchmarks tuned to the vendor's architecture, compiler etc. which makes them unsuitable for system comparisons. There are many *proprietary benchmarks* that do not provide full disclosure and suffer from lack of critical review. Unfortunately, the strengths and weaknesses of these benchmarks cannot be openly debated. Lack of critical review makes these benchmarks unsuitable for fair comparison. For example, Sun Microsystem, Inc. internally uses *iobench* to measure the I/O performance of Sun products.

Many vendors characterize system performance in units of *Million of Instructions per Second* (MIPS). The problem is that *not all instructions are created equal.* Comparing a million instructions of a CISC machine and a RISC machine is similar to comparing *apples and oranges.* This comparison is more confusing because there are a half dozen different dialects of MIPS [13]:

- *Native mips:* native instructions/second, on some unspecified benchmark suite
- *VAX-Dhrystone mips:* obtained by running the Dhrystone 1.1 and dividing the number by 1757. VAX-780 is considered a 1 mip machine.
- *IBM mips:* IBM 370/158 was considered a 1 mip machine.

Table 1
Popular benchmark results

Machines	Dhrystone 1.1 MIPS	Linpack DP MFLOPS
SS-490	22.6	3.8
IBM-320	27.5	7.4
M/2000	24.1	3.9

The floating point performance in (*MFLOPS*) is obtained by running the Linpack program. The *MIP* is obtained from the Dhrystone 1.1 benchmark. *Table 1* summarizes the performance of three machines: *Sun SPARCserver 490 (SS-490), MIPS M/2000 and IBM RS/6000 Model 320* in terms of the popular benchmarks. The primary reason for selecting these machines is that their SPECmarks are roughly similar. Appendix B describes the configuration of these three machines. Later we will examine the performance inflation caused by these benchmarks.

3. System performance

SPEC members struggled with questions like: Do we need one benchmark or many benchmarks? How long they should run and what is the measure of performance? 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 members wanted to compare system performance across many different architectures, implementations, memory systems, operating systems, clock rates, bus protocols, compilers, libraries and system software.

The system performance depended on many system components: CPU, floating-point-unit, I/O, bus, graphics and network accelerators, peripherals and memory systems. The software, operating system, compiler, and libraries significantly influenced the system performance. With so many variables SPEC decided to keep common source code across all machines. This meant that the benchmarks had to be ported to all member machines. As was later discovered this was easier said than done due to subtle differences between SunOS, System V and other variants of Unix.

Defining system performance as function of several variables, we have:

```
Performance  =Path Length × CPI × Cycle Time
Path Length  =F (Architecture, Compiler, Libraries, Operating System, I/O)
CPI          =F (Architecture, Compiler, Libraries, Technology, Implementa-
              tion)
Cycle time   =F (Technology, Implementation)
```

where *Path Length* is the number of instructions executed, *CPI* is cycles per instruction.

The best performance is obtained when the path length, the CPI and the cycle time are minimized. A good benchmark should exercise several components of the system. No single program can effectively exercise all system components and provide a meaningful result.

The discussions so far should indicate that there is *no absolute objective truth* in comparing benchmarking results. SPEC's mission was to select and develop real applications that would exercise all major system components.

4. Benchmark selection criteria

The best benchmark is an application program because most statistical models (kernels, synthetic programs, simulations etc.) are incomplete. These models do not compute anything that a user could use [9,15]. Unfortunately, there are thousands of applications and many are proprietary. A benchmark suite that contains too many applications is impractical due to difficulties in porting, evaluation and long run-times. SPEC started with the following criteria for accepting and evaluating candidate benchmarks:

- Public domain applications were desired.
- The benchmark should run for a minimum of one minute on an eight Dhrystone 1.1 MIP machine.
- The same source code should run without change on all member machines.
- Open debate of strengths and weaknesses of benchmarks.
- Clearly document reasons for rejecting benchmarks.
- Results must be correct and reproducible.

The goal of common source code across different versions of Unix, C and Fortran required a substantial porting effort. Additionally, many different tools had to be either ported or developed for ease of use across member platforms.

4.1. Rules and guidelines

SPEC provides execution rules, rules for reporting results and other guide lines that give performance results uniformity, clarity, credibility, and above all, reproducibility. For example, the run-rules prohibit changes to validation, benchmark-specific libraries and preprocessors, or source code changes that enhance performance.

4.2. Portability changes

SPEC permits portability changes to the source code. These are minimal (performance neutral) changes necessary for correct execution of the benchmark. The reasons for any changes must be justified, documented, and approved by the steering committee. The benchmark results are annotated with an asterisk (*) to indicate when a source code change was made for portability purposes. Portability conflicts are resolved at five day workshops where engineers from SPEC member companies modify and run the benchmarks so they are portable across operating systems and platforms.

5. SPEC Release 1.0 Suite

Release 1.0 of the SPEC Benchmark Suite is CPU (*Integer and Floating Point*) intensive and performs little I/O. These programs represent Computer Aided Software Engineering (CASE), Electronic Design Automation (EDA), and other engineering and scientific areas. Four programs are written in *C* and six in *Fortran*.

5.1. gcc

It is based on the GNU C compiler version 1.35 distributed by the Free Software Foundation. The benchmark measures the time it takes for the GNU C compiler to convert 19 preprocessed source files into optimized Sun-3 assembly language (.s) files. *gcc* is a CPU intensive benchmark written in C. It spends about 10% of its execution time performing disk I/O.

This benchmark characterizes work done in a workstation based software engineering environment. *gcc* is a popular compiler, a real application program, and an excellent system benchmark because it exercises memory and I/O. Performance is sensitive to cache size and the speed of the I/O device. *gcc* shows the highest cache miss ratio amongst all four integer benchmarks [12].

5.2. *espresso*

espresso is an integer benchmark written in C [20]. *espresso* is used for the generation and optimization of *Programmable Logic Arrays* (PLAs). It characterizes work done in the Electronic Design Automation market and Logic-Simulation and Routing-Algorithm areas. *espresso* performance is characterized by elapsed time to run a set of four different input models. The EECS department of the University of California at Berkeley distributes *espresso* through the Industrial Liaison Program. It is a relatively small program that manipulates arrays in loops. This program exercises storage allocation in computing the PLAs and is sensitive to cache size [12].

5.3. *spice2g6*

spice2g6 is an analog circuit simulation tool. It was developed by the Integrated Circuit group of the Electronics Research Laboratory and the EECS department of the University of California at Berkeley [21]. It is written mostly in Fortran. The Unix interface of the program is written in C.

spice2g6 is a CPU intensive floating-point (double precision) application. It is a *real* application heavily used in the EDA markets. It is a large program that runs five copies of the input model. Recent analyses have shown that the program is not floating point intensive and it spends a significant amount of time in a small loop that searches for an index in a table [15]. The pattern of data accesses causes a high cache miss rate. More than 80% of the assignments require memory to memory transfer. SPEC is working with other input circuits that would make this a more floating point intensive benchmark. The current circuit served to reduce the floating point bias in the SPEC suite.

5.4. *doduc*

doduc is a Monte Carlo simulation of the time evolution of a thermo-hydraulical model for a nuclear reactor's component. It is a double precision floating-point benchmark. The benchmark was developed by Nhuan Doduc [7].

doduc is a synthetic benchmark that represents high-energy physics applications. *doduc* is a non-vectorizable Fortran benchmark. It is a large kernel extracted from the original program. It has little I/O, an abundance of short branches, short loops, and executes code spread over many subroutines.

5.5. *nasa7*

nasa7 is a synthetic benchmark originally written for Cray machines. It is a collection of seven kernels. The original benchmark was developed by David H. Bailey and John T. Barton of NASA Ames. *nasa7* is written in Fortran and computes in double-precision floating-point.

The *nasa7* kernels test computations like matrix multiply, complex radix 2 FFT, Cholesky decomposition etc. The benchmark contains 95% vectorizable code [1].

5.6. *li*

li is a *Lisp* interpreter written in C. It is a CPU-intensive *integer* benchmark. The performance is measured by the time *li* takes to solve the 9-queens problem. This benchmark was originally developed by David Michael Betz and is based on XLISP 1.6, a small implementation of *LISP* with object oriented programming [2]. The backtracking algorithm used is recursive and poses a challenge for register window architectures [4].

5.7. *eqntott*

eqntott is a CPU-intensive *integer* benchmark written in C. The benchmark translates a logical representation of a boolean equation into a truth table. The original source is from the Industrial Liaison Program of the University of California at Berkeley [22]. The primary computation performed is a sort operation using the *quicksort* algorithm. The program fits comfortably in the instruction cache of most machines but the large data size can cause data cache misses on some machines [12].

5.8. *matrix300*

matrix300 is a floating-point intensive *Fortran* benchmark. This code exercises the Linpack routine [8] `saxpy` on matrices of order 300. The benchmark originated at the Los Alamos National Laboratory. The code is highly vectorizable. *matrix300* was selected because it stressed the memory subsystems. The smart preprocessors have significantly reduced the memory traffic on this benchmark [3]. This benchmark appears to favor superscalar and vector processors.

5.9. *fpppp*

fpppp is a quantum chemistry benchmark that measures performance on one style of computation (two electron integral derivative). The original benchmark is from H.B. Schlegel. It is a *Fortran* benchmark. The amount of computation is proportional to the fourth power of number of atoms (solves for eight atoms). This benchmark has one module that consists of a long piece of the straight-line code. This large basic block is a challenge to many compilers. This benchmark appears to favor superscalar and, to some degree, vector processors.

5.10. *tomcatv*

tomcatv is a highly (90–98%) vectorizable mesh generation program. The program is floating-point intensive. The original *Fortran* source is from Dr. Wolfgang Gentzsch. This benchmark favors superscalar and vector processors. This benchmark causes high data cache misses on several platforms.

Three benchmarks (*doduc, nasa7 and matrix300*) are synthetic. Other benchmarks are applications used as benchmarks.

6. SPEC results

SPEC chose a simple measure-elapsed time to run all the benchmarks. A simple metric and machine independent code were key to providing a comprehensive and fair comparison

Table 2
Release 1.0 benchmarks

Benchmarks	Type	Description	SS-490 run-time seconds	IBM-320 run-time seconds	M/2000 run-time seconds
gcc	INT	GNU C Compiler	70.1	108.0	76.7
espresso	INT	PLA optimizing tool	136.7	139.3	119.8
spice2g6	FP	Circuit simulation & analysis	1455.6	1208.7	1720.9
doduc	FP	Monte Carlo simulation	108.3	86.3	108.9
nasa7	FP	Seven floating point kernels	846.1	756.0	1072.7
li	INT	LISP interpreter	282.8	397.0	258.8
eqntott	INT	Converts equations to truth table	56.2	62.1	59.4
matrix300	FP	Solves matrices of order 300	184.2	258.4 [a]	322.4
fpppp	FP	Quantum chemistry application	160.1	70.8	130.3
tomcatv	FP	Mesh generation application	169.9	46.9	149.8

INT - Integer benchmark in C. FP - Floating point benchmark in Fortran.
[a] Recently IBM has reported new run-time of 14 seconds.

between competing machines. *Table 2* summarizes the run-times on three platforms (Sun, IBM, and MIPS) with similar performance.

The Dhrystone 1.1 results from *Table 1* suggest that the IBM-320 should beat the Sun SS-490 and MIPS M/2000 on integer benchmarks. The run-times on *gcc*, and *li* demonstrate that both the SS-490 and M/2000 are faster than the IBM-320. Recent studies indicate that cache misses on these integer applications are relatively small on current sets of machines [4,12]. The real life instruction mix appears to be different than the instruction mix of the synthetic benchmark (e.g. Dhrystone 1.1).

matrix300 is similar to *Linpack* ($n = 300$), but IBM-320's performance is not twice as good as either SS-490 or M/2000. In reality, SS-490 beats IBM-320 on this benchmark [2]. On the other hand, IBM-320 clearly wins on *fpppp* and *tomcatv* by larger margins than suggested by *Table 1*. The moral of the story is that *no single benchmark result, including the SPECmark, should be used to predict performance.*

6.1. Dynamic instruction profiles

Both the MIPS Computer's pixie and the Sun Microsystems' tool spix were used to analyze the ten SPEC benchmarks. These tools report a count of executed instructions for each benchmark. Both *pixie* and *spix* count instructions executed in *user mode* and generate other statistical information on instruction and subroutine profiles. Both models assume a perfect memory system; cache misses will increase execution time above the predictions.

Table 3 reports the data collected for each SPEC benchmark on the MIPS M/2000. *Table 4* reports similar data for the Sun Microsystems' SS-490. Similar tools were not available at the time for the IBM-320.

The instruction profiles show that the SPEC benchmarks provide a reasonable diversity of computational problems. The instruction mix varies considerably across different benchmarks for the MIPS M/2000 and the Sun Microsystems' SS-490 architectures and compiler versions. Additional analysis of the suite showed a variety of load, branch, memory, floating point, and other pipe line stalls in the execution of the benchmarks for both architectures [6]. Other architectures may suffer from similar types of pipe line stalls.

[2] Recently, IBM has announced their results with the Kuck Associate's preprocessor that significantly reduced the run-time from 258.4 seconds to about 14 seconds (almost 18.5 times faster) on the *matrix300*. The SPEC members have voted to exclude *matrix300* in future releases.

Table 3
MIPS - instruction profile

Benchmarks	Instructions billion	Loads %	Stores %	NOPS %	FOPS %	Others %
gcc	1.1	19.8	11.6	13.3	0.0	55.3
espresso	2.8	18.4	7.3	12.0	0.0	62.3
spice2g6	21.5	24.6	5.1	14.6	5.1	50.6
doduc	1.6	27.8	8.9	7.3	23.7	32.3
nasa7	9.3	36.8	14.1	1.5	23.0	24.6
li	6.0	22.1	11.9	21.9	0.0	44.1
eqntott	1.2	16.5	0.9	14.5	0.0	68.1
matrix300	2.8	31.4	15.7	0.0	15.6	37.3
fpppp	2.3	48.7	13.2	8.4	25.8	3.9
tomcatv	1.8	36.6	13.0	0.9	27.3	22.2

Loads - Both integer and floating point.
Stores - Both integer and floating point.
NOPS - No operation.
FOPS - Floating point operations.
Others - All other arithmetic, logic, branch, etc., instructions.

It is interesting to note that both the Sun Microsystems' SS-490 and the MIPS M/2000 executed roughly 50 billion instructions on these benchmarks. Direct comparison of instruction counts between different architectures can be misleading due to differences in cycles/ instruction. Because *gcc* spends up to 10% of its time in the *UNIX kernel*, and tools only count instructions in the *user mode*, the real instruction count would be higher. Additionally, time spent in cache misses and register window overflow/underflow (a feature of Sun Microsystems' architecture) is not reflected in either instruction profile. Instruction profiles can reveal potential performance bottlenecks.

7. SPEC metrics

SPEC chose to use *elapsed time* as the metric for the Release 1.0 suite. SPEC has defined the following set of metrics for reporting results [18].

Table 4
SPARC instruction profile

Benchmarks	Instructions billion	Loads %	Stores %	NOPS %	FOPS %	Others %
gcc	1.2	18.5	8.1	1.0	0.0	72.4
espresso	2.9	23.3	4.9	0.3	0.0	71.5
spice2g6	22.8	21.0	4.0	0.5	4.2	70.3
doduc	1.3	22.9	6.5	2.6	25.9	42.1
nasa7	6.6	28.4	11.2	0.0	31.4	29.0
li	4.6	22.9	10.3	1.4	0.0	65.4
eqntott	1.3	15.4	1.2	0.1	0.0	83.3
matrix300	1.7	25.7	12.8	0.0	25.5	36.0
fpppp	1.4	40.9	9.0	0.2	41.0	8.9
tomcatv	1.6	29.7	11.6	0.8	30.8	27.1

7.1. SPEC reference time

For Release 1.0, the SPEC Reference Time is the time (in seconds) it takes a DEC VAX 11/780 to run each benchmark in the suite.

7.2. SPEC ratio

The SPEC Ratio for a benchmark is the quotient derived from dividing the SPEC Reference Time by a particular machine's run time. For example, if the SPEC Reference Time for *gcc* was 1482 seconds and machine A's run time for *gcc* was 100 seconds, the SPEC Ratio for *gcc* on A would be:

$$\text{SPEC ratio} = \frac{\text{Elapsed time on VAX-11/780}}{\text{Elapsed time on A}} = \frac{1482}{100} = 14.8$$

7.3. SPECmark

The SPECmark is the Geometric Mean (GM) of the ten SPEC Ratios.

$$\text{SPECmark} = \sqrt[10]{\text{Product of 10 SPEC ratios}}$$

SPEC chose the geometric mean for two reasons:
(1) it is not unduly influenced by long running programs and
(2) it provides a consistent and fair metric [16,19].

The SPECmark is an overall measure of CPU performance on the Release 1.0 suite. However, SPEC strongly recommends that all ten SPEC Ratios be published. No single number can adequately characterize today's complex systems. As we will see later, even the SPECmark by itself can give a skewed picture of the performance. Unfortunately, some press accounts report the SPECmark without the corresponding SPEC Ratios.

7.4. Reading results

Figure 1 illustrates SPEC Benchmark results for three different architectures: the *Sun SPARCserver 490 (SS-940), the MIPS M/2000 and the IBM RS/6000 Model 320 (IBM-320).* Even though the SPEC-marks of these machines are similar their performance on each benchmark varies significantly. As discussed earlier, a single number performance characterization can be misleading.

The ratio of maximum to minimum SPECratios for all three architectures: 1.57 for SS-490, 1.71 for M/2000 and 4.12 for IBM-320, illustrates the difficulties in predicting performance of new applications and the fallacy of single number performance characterization.

As indicated earlier, the superscalar architecture (IBM-320) does very well on most of the vectorizable benchmarks (e.g. *nasa7, fpppp and tomcatv*). Additionally, the performance on the integer benchmarks is relatively lower. Seven of the ten benchmarks perform at a lower level than the SPECmark (22.3).

The SPARC architecture appears to do reasonably well on two integer benchmarks (*gcc, eqntott*). Its performance on *fpppp and tomcatv* needs improvement. Five of the ten benchmarks perform at a lower level than the SPECmark (19.4).

The MIPS machine does well on two integer benchmarks (*espresso, li*) and outperforms the Sun SS-490 on two floating point benchmarks (*fpppp, tomcatv*). Four of the ten benchmarks perform at a lower level than the SPECmark (18.3).

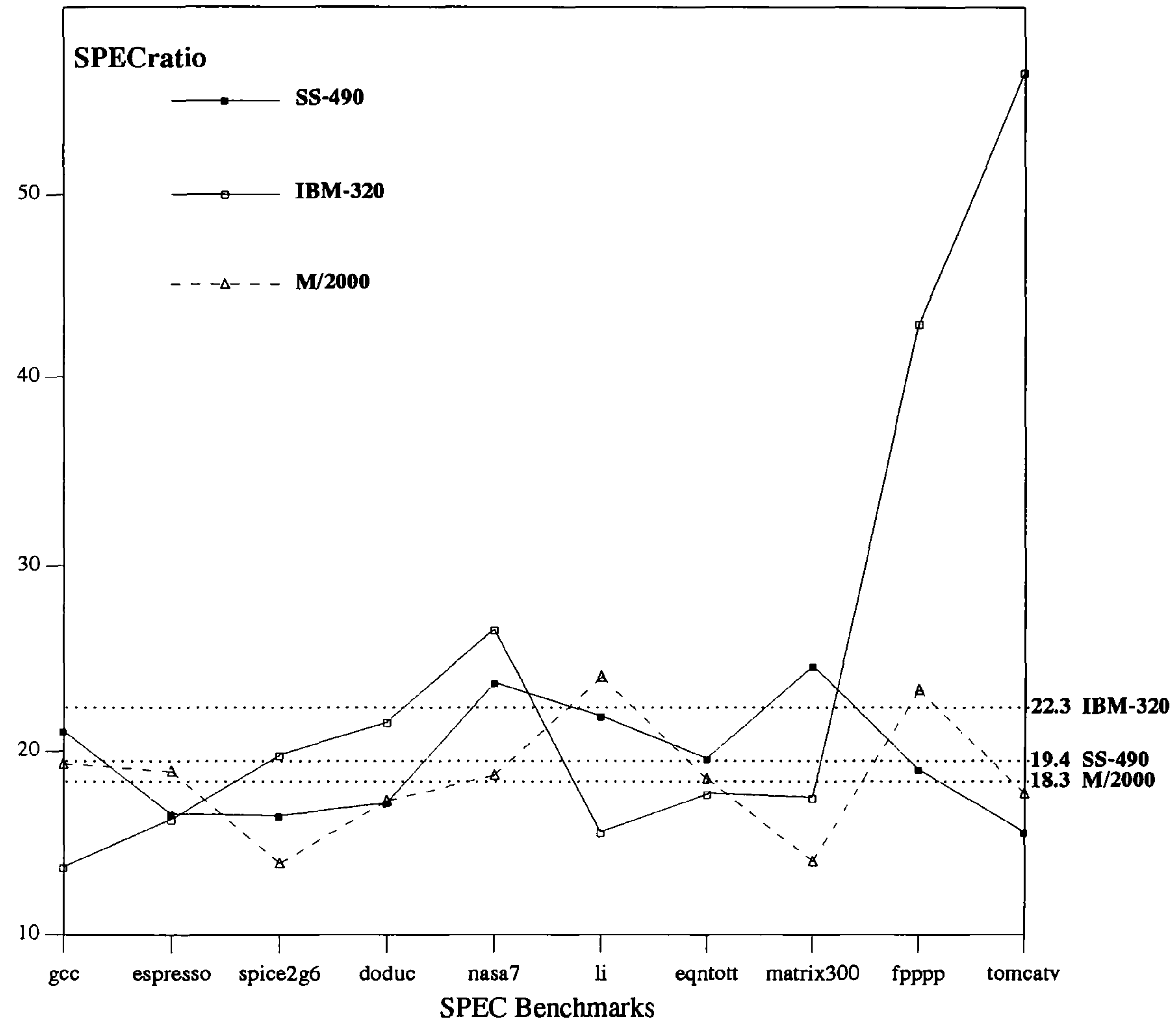

Fig. 1. SPEC benchmark performance results.

Table 5
Release 1.0 benchmarks

Benchmarks	SS-490 SPECratio	IBM-320 SPECratio	M/2000 SPECratio
gcc1.35	21.1	13.7	19.3
espresso	16.6	16.3	18.9
spice2g6	16.5	19.8	13.9
doduc	17.2	21.6	17.3
nasa7	23.7	26.6	18.7
li	21.9	15.6	24.0
eqntott	19.6	17.7	18.5
matrix300	24.6	17.5 [a]	14.0
fpppp	19.0	42.9	23.3
tomcatv	15.6	56.5	17.7
SPECint	19.7	15.8	20.1
SPECfp	19.1	28.0	17.2
SPECmark	19.4	22.3	18.3

[a] Recently IBM has reported SPECratio of 323.2 for the matrix300. These numbers frequently change – always obtain the latest data from SPEC.

The strength of the results is that the combination of SPECratios and SPECmark shows strengths, computational balance relative to VAX-780 (between integer and floating point).

The information in *Fig. 1* was derived from *Table 5*, which was itself obtained from the published SPEC newsletters.

The new SPEC reporting format includes the SPECint (geometric mean of four integer benchmarks) and the SPECfp (geometric mean of six floating point benchmarks) to highlight the different computational balance (integer versus floating point).

It is clear from the SPECratios that both the SS-490 and the M/2000 have superior integer performance to the IBM-320. The SPEC integer benchmarks reduce the popular Dhrystone 1.1 MIPS rating by 14.7% for SS-490, 74% for IBM-320 and 19.9% for M/2000. The IBM-320 wins in five of the six floating point intensive benchmarks.

A result in the SPEC Newsletter contains complete configuration (*hardware and software*) information, portability changes (if any), and system software environment.

Appendix B describes the hardware and software configurations for three platforms.

8. What's wrong with SPEC Release 1.0?

SPEC benchmarks are studied, analyzed and debated at technical forums by many organizations, vendors, and universities. This public interest, awareness and debate has revealed strengths and weaknesses of the Release 1.0 suite. The suite is biased in favor of double precision floating point processing. Most benchmarks do not stress the instruction cache because of the 10/90 rule (10% of the code executes 90% of the time). Only three benchmarks heavily use the data cache. Some benchmarks run too long and others now run a bit too short. Workload characterization for multiprocessors is contrived (e.g. minimal I/O) and SPEC Thruput can be optimistic. Most of the Release 1.0 benchmarks make few *system* and *procedure calls*.

We learned that large and long running programs are desirable and necessary characteristics but not always sufficient. Careful analysis of the dynamic execution properties of the program is an important necessity. SPEC results may be optimistic because the measurements are made in single user mode. The majority of the SPEC benchmarks do fit in split 128KB(instruction) and 128KB(data) caches thereby reducing cache misses close to zero. Real applications run in multiuser mode that exercise other system components (e.g. cache, memory, paging, swapping etc.) more severely than the current suite.

It is hoped that some of these problems will be avoided in future releases. SPEC is not a panacea but even SPEC's critics agree that this suite provides a level playing field for all vendors and reported results are reproducible.

9. Summary

The best benchmark is an application program used as a benchmark. The advantage of running an unmodified application as a benchmark is that results can be validated and the run-time comparison between computer systems is both meaningful and believable. Primary uses of benchmarks are:

- Performance prediction for new applications, architectures and technologies.
- Performance tracking – improvements in hardware and software.
- Used for quality assurance and regression testing.
- Marketing/sales product strategy and promotion.

Performance predictions for new applications are difficult. The SPEC suite needs more diverse applications. The current suite can be used to predict, and track improvements in compiler technology. The new Sun compiler (SPARCcompiler 1.0) has shown significant improvement over its predecessor on the Release 1.0. SPECmarks for the SS-490 improved from 17.6 to 19.4, M/2000 improved from 17.6 to 18 and IBM-320 improved from 22.3 to 32.8. Results from other vendors indicate a similar trend.

The major benefit of these compiler improvements is that they globally improve performance of many applications similar to the Release 1.0 suite. The SPEC Release 1.0 suite provides a method for objective speed comparison between platforms. SPEC has leveraged bright ideas, hard work, and resources into a product that no single company or individual can match. It is both notable and commendable that SPEC participants were able to convince their respective management and marketing organizations to accept the new metric that discounted the published MIPS ratings of their products.

SPEC hopes that all vendors will use SPECratios to characterize performance rather than MIPS and MFLOPS.

9.1. Future of SPEC

Release 1.0 from SPEC is popular. The primary reason for its current success and popularity is its acceptance among member companies who put their differences aside and focused on making it a useful benchmark. Release 1.0 is a first significant step towards better benchmarking. It is also enjoying wide spread acceptance because it is fair, simple to use and results are reproducible. SPEC has put a simple and inexpensive tool in the hands of users to verify vendor's claims and reduce dependence on performance pundits. It has provided both hardware and software vendors a good reference to track and improve their products. SPEC performance metrics are routinely provided at new product announcements by many companies.

Recently SPEC announced a new benchmarking suite. The System Development Multitasking Release 1.0 (SDM1) consists of two multitasking system level benchmarks. These benchmarks heavily exercise CPU, memory, disk I/O and operating system services. The SPEC SDM1 complements the current SPEC Release 1.0 that primarily exercise CPU and FPU (floating point unit).

SPEC is facing important business challenges due to diverse membership interests, and increased bureaucracy. The technical problems are analogous to releasing a product approved by: nine development, release engineering and marketing divisions of member companies.

Additional information about the SPEC source code, benchmark results, and the newsletter is available from:

SPEC

c/o Franson & Hagerty	c/o NCGA
181 Metro Drive, suite 300	2722 Merrilee Drive, Suite 200
San Jose, CA 95110, USA	Fairfax, VA 22031-4499
Phone: (408) 453-5220	Phone: (703) 698 9600 X318
FAX: (408) 453-8723	FAX: (703) 560 2752

SPEC publishes a quarterly newsletter that provides insights into the benchmarks, SC decisions and results.

10. Credits and caveats

I thank Vicki Scott from MIPS Computer, who provided the latest benchmark statistics. I thank all my fellow SPEC members who are responsible for the current popularity of SPEC. My special thanks to Earl Killian (MIPS) and Robert Cmelik (Sun) for *pixie and spix*. I am indebted to Varun Mehta, Maneesh Dhir, Sanjay Jain, Sadegh Gharavi, and Nhan Chu for their assistance. I am thankful to Jack Dongarra and Dave Levine for making this a readable paper. The opinions expressed in this article are those of the author and do not necessarily reflect those of SPEC or other SPEC members.

Appendix A. Glossary

- CISC - Complex Instruction Set Computer. Examples are VAX-780, IBM-370, Motorola 68xxx, Intel 80x86.
- RISC - Reduced Instruction Set Computer. Examples are IBM RISC/6000, MIPS M/2000, SPARCserver 490.
- Synthetic Benchmarks are programs created for benchmarking various machines. They claim to represent average characteristics of real programs. Examples are Dhrystones and Whetstones.
- Kernel Benchmarks are critical code fragments extracted from real programs. For example Livermore Loops, Linpack routines and Nasa7.
- Propietary Benchmarks are created by the vendors for profit. Examples are benchmark suites from Work Station Laboratory, Neal Nelson and AIM technology.
- MIPS - Millions of instructions per second. A popular metric for describing the integer performance of computer systems.
- MFLOPS - Millions of floating point operations per second. A popular metric for describing the floating point performance of computer systems.
- EDA - Electronic Design Automation. These are application programs that assist in design of electronic circuits.
- ECAD - Electronic Computer Aided Design. Similar to EDA.

Appendix B. Configuration information

SPARCserver 490 (SS-490)
  33 MHz CY7C601 IU, 33 MHz TI ACT8847 FPP
  128 KB (I + D) VAC, write back, 32 byte line
  32MB RAM
  avail mem = 31137792
  IPI Controller with 8/1.2 GB CDC disk
  SunOS Release 4.1_PSR_A (GENERIC)#1
  SunOS 4.1_PSR_A
IBM - RS 320 RS/6000 Model 320
  CPU 20 MHz IBM 2032, FPU – integrated
  Cache (8K I + 32 K D)
  Memory 16 MB, 320MB SCSI
  OS AIX Ver 3.1, AIX XL C/6000 &
  AIX XL FORTRAN/6000
MIPS M/2000 M/2000
  CPU 25 MHz R3000, FPU 25 MHz R3010
  Cache (64KB I + 64KB D)
  Memory 64MB
  Disk SMD, 1 × 663MB
  OS RISC/os 4.50, CC & F77 Release 2.11

References

[1] D.H. Bailey and J.T. Barton, The NAS Kernel benchmark program, NASA Tech. Memo. 86711, Aug. 1985.
[2] D.M. Betz, XLISP 1.6, Rev. 1.0, November 1988.
[3] S. Carr and K. Kennedy, Blocking linear algebra codes for memory hierarchies, in: *Fourth Siam Conf. on Parallel Processing for Scientific Computing*, Chicago, IL (Dec. 1989).
[4] T.M. Conte and W.M.W. Hwu, Benchmark characterization, *IEEE Comput.* (Jan. 1991).
[5] H.J. Curnow and B.A. Wichman, A synthetic benchmark, *Comput. J.* (1976) R.M. Stallman, GNU CC Compiler Version 1.35, Free Software Foundation, 675 Mass Ave., Cambridge MA, April 1989.
[6] R.F. Cmelik, S.I. Kong, D.R. Ditzel and E.J. Kelly, An analysis of MIPS and SPARC instruction set utilization on the SPEC benchmarks, SMI, 1991.
[7] N. Doduc, FORTRAN execution time benchmark, Framentec, V29, March 1989.
[8] J.J. Dongarra, Performance of various computers using standard linear equation software, Oct. 12, 1989.
[9] J.L. Hennessy and D.A. Patterson, *Computer Architecture, A Quantitative Approach* (1990).
[10] J.L. Hennessy and D.A. Patterson, *Computer Architecture, A Quantitative Approach*, Ch. 2 (1990) 73–74.
[11] J.L. Hennessy and D.A. Patterson, *Computer Architecture, A Quantitative Approach* (1990).
[12] M.D. Hill, Cache performance of the Integer SPEC benchmarks on a RISC, *Comput. Architecture News* 18 (2) (June 1990).
[13] J.R. Mashey, Your mileage may vary...but if your car were a computer, it would vary more, Issue 2.0, 05/18/90 MIPS Computer Systems.
[14] A.K. Portfield, Software methods for improvement of cache performance on supercomputer applications, PhD thesis, Rice University, 1989.
[15] R.H. Savedra-Barrera, The SPEC and Perfect Club benchmarks: Promises and limitations, in: *Hot Chips-2 Symp.* (Aug. 1990).
[16] J.E. Smith, Characterizing computer performance with a single number, *Commun. ACM* (1988) 1202–1206.
[17] K.M. Dixit, Speculations: Defining the SPEC benchmark, *SunTech J.* 4 (1) (Jan. 1991).
[18] SPEC Staff, Benchmark result reporting format terminology, *SPEC Newsletter* 3 (1) (Winter 1991).
[19] K. Dixit and R. Novak, SPEC refines reporting format to ensure level playing field, *SPEC Benchmark Results* 2 (1) (Winter 1990).
[20] EECS Department of U.C. Berkeley, Espresso Version #2.3, Release Date 01/31/88.
[21] CAD/IC Group, EECS/ERL of U.C. Berkeley, SPICE2G.6, Release Date March 1987.
[22] The Industrial Liaison Program of U.C. Berkeley, Eqntott #V9, Released 1985.
[23] R. Weicker, Dhrystone: A synthetic systems programming benchmark, *Commun. ACM* (1984).
[24] R. Weicker, An overview of common benchmarks, *Microprocessor Forum* (Fall 1989).
[25] R.P. Weicker, Understanding variations in Dhrystone performance, *Microprocessor Report* (May 1989).