Table 1.3. Real versus normalized floating-point (FP) operations.

Real FP operations	Normalized FP operations
ADD, SUB, COMP, MULT	1
DIV, SQRT	4
EXP, SIN,...	8

MIPS and MFLOPS are useful for comparing members of the same architectural family. They cannot be used to compare computers with different instruction sets and different clock cycles. This is because programs may be translated into different numbers of instructions on different computers.

1.3.5. Other Performance Measurements

Besides execution time, MIPS and MFLOPS, other measurements are often used to obtain a better characterization of a computer system. The most commonly used are *throughput*, *utilization*, *memory bandwidth*, *memory access time*.

- **Throughput** of a processor is a measure that indicates the number of programs (tasks or requests) that the processor can execute per unit of time.
- **Utilization** of a processor refers to the fraction of time the processor is busy executing programs. It is the ratio of busy time and total elapsed time over a given period.
- **Memory bandwidth** indicates the number of memory words that can be accessed per unit of time.
- **Memory access time** is the average time that it takes the processor to access the memory, usually expressed in terms of nanoseconds.

1.3.6. Benchmark Programs

One of the most utilized measures of computer performance is the time of executing a set of representative programs on that computer. This time can be the total execution time of the programs from the set, the arithmetic mean or the geometric mean of the execution times, or other similar measure. A set of actual programs that are representative for a particular computing environment can be used for performance evaluation. Such programs are called *benchmarks* and are run by the user on the computer being evaluated.

The best type of programs to use for benchmarks are real applications. These may be applications that the user employs regularly or applications that are typical. For example, if the users are primarily engineers, one might use a set of benchmarks containing several typical engineering or scientific applications.

1.3.6.1. Comparing and Summarizing Performance

After selecting the programs to use as benchmarks, the problem is how to summarize the performance of a group of benchmarks. The users often prefer to have a single number to compare performance. Table 1.4 illustrates a situation when it is difficult to compare the performance of two machines.

Table 1.4. Execution times of two programs on two different computers.

	Computer A	Computer B
Program 1 (s)	1	10
Program 2 (s)	1000	100
Total time (s)	1001	110

For program 1, computer *A* is 10 times faster than *B*. For program 2, computer *B* is 10 times faster than *A*. Using these measurements, the relative performance of computers *A* and *B* is unclear.

The simplest method to summarize relative performance is to use total execution time of the two programs. Thus, computer *B* is $1001/110 = 9.1$ times faster than *A* for programs 1 and 2.

The average of the execution times is the *arithmetic mean* (*AM*):

$$AM = \frac{1}{n}\sum_{i=1}^{n} t_{E\,i} \qquad (1.19)$$

where t_{Ei} is the execution time for the i^{th} program of a total of n in the benchmark set. A smaller mean indicates a smaller average execution time and thus improved performance.

The arithmetic mean indicates execution time by assuming that the benchmark programs from the set are each run an equal number of times. If it is not the case, we can assign a weighting factor w_i to each program to indicate the frequency of the program execution in the set. This is called the *weighted arithmetic mean*. One method of weighting programs is to choose weights so that the execution time of each benchmark is equal on a reference machine.

Other method of presenting machine performance is to normalize execution times to a reference machine, and then take the average of the normalized execution times. However, if we compute the arithmetic mean of the normalized execution time values, the result will depend on the choice of the machine we use as a reference. For example, in Table 1.5 the execution times from Table 1.4 are normalized to both *A* and *B*, and the arithmetic mean is computed.

Table 1.5. Execution times of two programs on machines *A* and *B*, normalized to each machine, the arithmetic mean and geometric mean of the execution times.

			Normalized to A		**Normalized to B**	
	Time on A	**Time on B**	**A**	**B**	**A**	**B**
Program 1	1	10	1	10	0.1	1
Program 2	1000	100	1	0.1	10	1
Arithmetic mean	500.5	55	1	5.05	5.05	1
Geometric mean	31.6	31.6	1	1	1	1

When we normalize to machine *A*, the arithmetic mean indicates that *A* is faster than *B* by 5.05/1. When we normalize to *B*, the arithmetic mean indicates that *B* is faster than *A* by 5.05. Only one of these results can be correct. The difficulty arises from the use of the arithmetic mean of execution times.

Instead of using the arithmetic mean, the normalized execution times should be combined with the *geometric mean* (*GM*). The formula for the geometric mean is:

$$GM = \sqrt[n]{\prod_{i=1}^{n} t_{Ei}} \tag{1.20}$$

where t_{Ei} is the execution time, normalized to the reference machine, for the *i*th program of a total of *n* in the benchmark set.

The geometric mean is independent of which data series we use for normalization, because it has the following property:

$$\frac{GM(X_i)}{GM(Y_i)} = GM\left(\frac{X_i}{Y_i}\right) \tag{1.21}$$

meaning that taking either the ratio of the means or the means of the ratios produces the same results.

Thus the geometric mean produces the same result whether we normalize to machine *A* or *B*, as we can see in Table 1.5. When execution times are normalized, only a geometric mean can be used to combine the normalized results.

The advantage of the geometric mean is that it is independent of the running times of the individual programs, and it doesn't matter which machine is used for normalization. However, the disadvantage of using geometric means of execution times is that they do not predict execution time. The geometric means in Table 1.5 suggest that for programs 1 and 2 the performance is the same for machines *A* and *B*. The arithmetic mean of the execution times suggests that machine *B* is 9.1 times faster then machine *A*.

1.3.6.2. The Evolution of Benchmark Programs

While it seems obvious today that the best solution is to develop a set of real applications that can be used as standard benchmarks, this was a difficult task until relatively recent times. Variations in operating systems and language standards made it hard to create large programs that could be moved from one machine to another simply by recompiling. Instead, after creating the MIPS and MFLOPS metrics, the next step was the development of artificial or *synthetic benchmark* programs. The goal was to create a single benchmark program where the execution frequency of instructions matches the instruction frequency in a large set of benchmarks. *Whetstone* and *Dhrystone* are the most popular synthetic benchmarks.

The *Whetstone* synthetic benchmark was created based on measurements of scientific and engineering applications written in ALGOL. This program was later converted to FORTRAN, and was widely used to characterize scientific program performance. *Dhrystone*, which was inspired by *Whetstone*, was created more recently as a benchmark for systems programming, and was based on a set of published frequency measurements. *Dhrystone* was originally written in *Ada* and later converted to *C*, after which it became popular.

Because synthetic benchmarks are not real programs, they usually do not reflect programs behavior. Furthermore, compiler and hardware optimizations can amplify the performance of these benchmarks, beyond what the same optimizations would achieve on real programs. For example, optimizing compilers can easily discard 25% of the *Dhrystone* code.

Kernel benchmarks are small, time-intensive pieces extracted from real programs. They were developed primarily for benchmarking high-end machines, especially supercomputers. *Livermore Loops* and *Linpack* are the best known examples. The *Livermore Loops* benchmark consists of a series of 21 small loop fragments. *Linpack* consists of a portion of a linear algebra subroutine package. Kernels are best used to isolate the performance of individual characteristics of a machine and to explain the reasons for differences in the performance of real programs. They are mostly used to characterize performance of scientific applications.

An important step in performance evaluation was the formation of the SPEC (*Standard Performance Evaluation Corporation*) group in 1988. SPEC is a non-profit corporation which develops and maintains standardized sets of benchmarks based on real programs. The benchmark suites contain source code and tools for generating performance reports, and are extensively tested for portability before release.

The first benchmark set (called SPEC89) was released in 1989. It contained six floating-point benchmarks and four integer benchmarks. A single metric was computed, SPECMark, using the geometric mean of execution times normalized to the VAX-11/780 computer. This measure favored machines with strong floating-point performance.

In 1992, a new benchmark set (called SPEC92) was introduced. It incorporated additional benchmarks, and provided separate metrics (SPECINT and SPECFP) for integer and floating-point programs.

Today, the SPEC organization consists of three groups, each with their own benchmarks:

- *Open Systems Group* (OSG): Component- and system-level benchmarks in an UNIX / NT / VMS environment.
- *High Performance Group* (HPG): Benchmarking for high-performance numeric computing.
- *Graphics Performance Characterization Group* (GPCG): Benchmarks for graphical subsystems, OpenGL and *Xwindows*.

1.3.6.3. CPU95

The CPU95 benchmarks were introduced by SPEC in 1995 as a replacement for the older CPU92 benchmarks. They were developed by the *Open Systems Group*, which includes more than 30 computer vendors, systems integrators, publishers and consultants. These benchmarks measure the performance of CPU, memory system, and compiler code generation. They normally use UNIX as the portability vehicle, but they have been ported to other operating systems as well. The percentage of time spent to execute operating system functions and I/O operations is generally negligible.

The CPU95 benchmarks are internally composed of two collections:

- CINT95: integer programs, representing the CPU-intensive part of system or commercial application programs;
- CFP95: floating-point programs, representing the CPU-intensive part of numeric-scientific application programs.

The CPU benchmarks can be used for two types of measurement:

- Speed measurement;
- Rate (throughput) measurement.

Speed Measurement

The result of each individual benchmark, called *"SPEC ratio"*, is expressed as the ratio of the time to execute one single copy of the benchmark, compared to a fixed SPEC reference time. For the CPU95 benchmarks, a *Sun SPARCstation* 10/40 with 128 MB of memory was chosen as the reference machine.

The different SPEC ratios for a given machine can vary widely. Users should consider those benchmarks that best approximate their applications. SPEC has also defined the following averages for speed measurements with the CPU95 benchmarks:

- *SPECint_base95*: geometric mean of the 8 SPEC ratios from CINT95 when compiled with conservative optimization for each benchmark;

- *SPECfp_base95*: geometric mean of the 10 SPEC ratios from CFP95 when compiled with conservative optimization for each benchmark;
- *SPECint95*: geometric mean of the 8 SPEC ratios from CINT95 when compiled with aggressive optimization for each benchmark;
- *SPECfp95*: geometric mean of the 10 SPEC ratios from CFP95 when compiled with aggressive optimization for each benchmark.

SPEC CPU95 incorporates run and reporting rules that permit both “baseline” and optimized results for the CINT95 and CFP95 suites. The “baseline” results are obtained by conservative optimization, and the optimized results are obtained by aggressive optimization. The “baseline” rules restrict the number of compiler optimizations that can be used for performance testing.

Rate Measurement

With this measurement method, several copies of a given benchmark are executed. This method is particularly suitable for multiprocessor systems. The result of a benchmark, called *“SPEC rate”*, express how many jobs of a particular type can be executed in a given time. The SPEC reference time is one 24-hour day. The execution times are normalized with respect to the SPEC reference machine. The SPEC rates therefore characterize the capacity of a system for compute-intensive jobs of similar characteristics.

Similar as with the speed metric, SPEC has defined the following averages:

- *SPECint_rate_base95*: geometric mean of the 8 SPEC rates from CINT95 when compiled with conservative optimization for each benchmark;
- *SPECfp_rate_base95*: geometric mean of the 10 SPEC rates from CFP95 when compiled with conservative optimization for each benchmark;
- *SPECint_rate95*: geometric mean of the 8 SPEC rates from CINT95 when compiled with aggressive optimization for each benchmark;
- *SPECfp_rate95*: geometric mean of the 10 SPEC rates from CFP95 when compiled with aggressive optimization for each benchmark

1.3.6.4. CPU2000

CPU2000 is the latest version of the standard for evaluating computer system performance, released at the end of 1999. The new benchmarks can be used across several versions of UNIX and *Microsoft* operating systems. They reflect the advances in microprocessor technologies, compilers and applications that have taken place over the last five years.

The new release replaces SPEC CPU95, which was phased out in July 2000, when SPEC stopped publishing CPU95 results. Performance results from CPU2000 cannot be compared to those from CPU95, since new benchmarks have been added and existing ones changed.

SPEC CPU2000 comprises two sets of benchmarks: CINT2000 for measuring compute-intensive integer performance, and CFP2000 for compute-intensive floating point performance. Improvements to the new sets include longer run times and larger problems for benchmarks, more application diversity, and standard development platforms that will allow SPEC to produce additional releases for other operating systems.

SPEC CPU2000 provides performance measurements for system speed and throughput. The speed metrics, *SPECint2000* and *SPECfp2000*, measures how fast a machine completes running all of the benchmarks from the CINT2000 and CFP2000 sets, respectively. The throughput metrics, *SPECint_rate2000* and *SPECfp_rate2000*, measures how many tasks a computer can complete in a given amount of time.

SPEC selected the *Sun Microsystems Ultra 5/10* workstation with a 300-MHz SPARC processor and 256 MB of memory as a reference machine. All benchmark results are computed as ratios against the reference machine, which has a *SPECint2000* and *SPECfp2000* score of 100. Each benchmark was run and measured on the *Ultra* 5/10 to establish a reference time.

SPEC considered the following criteria in the process of selecting applications to use as benchmarks:

- Portability to all SPEC hardware architectures (32-bit and 64-bit architectures including *Alpha*, *Intel Architecture*, Rxx00, SPARC, etc.);
- Portability to various operating systems, particularly UNIX and Windows;
- Benchmarks should not include measurable I/O, networking or graphics activities;
- Benchmarks should run in 256 MB of RAM memory without swapping;
- No more than five percent of benchmarking time should be spent processing code not provided by SPEC.

1.3.7. Quality Factors

In addition to the performance measurements, a number of quality factors also have influence over the success of a computer. Some of these factors are *generality*, *ease of use*, *expandability*, *compatibility*, and *reliability*.

- **Generality** is a measure that determines the range of applications for an architecture. Some architectures are good for scientific purposes and some for business applications. The architecture is more profitable when it supports a variety of applications.
- **Ease of use** is a measure of how easy it is for the system programmer to develop software for the architecture.