Edgar H. Sibley
Panel Editor

The controversy surrounding single number performance reduction is examined and solutions are suggested through a comparison of measures.

CHARACTERIZING COMPUTER PERFORMANCE WITH A SINGLE NUMBER

JAMES E. SMITH

Reducing computer performance to a single number has become one of the more controversial (and confusing) subjects in performance evaluation. At best, the need for a single performance number is seen as a necessary evil; many argue that performance is multidimensional and can only be accurately represented with a series of performance numbers. Despite any such arguments, the fact remains that single numbers will be used for performance comparisons. This being the case, it is important that the best *single-number* measure be used. Properties of *good* measures are proposed in this article, and commonly used measures are studied and compared to determine which have the desired properties.

An area where performance is of considerable interest and the temptation to use a single performance number seems greatest is in scientific (floating point intensive) applications. Indeed, this is one of the earliest computer application areas, and is often used as a yardstick for measuring overall progress in the art of computer design. Consequently, discussion and examples in this article are drawn from the context of central processor performance on scientific (floating point intensive) applications. For example, units of millions of floating point operations per second (mflops) are used. The central idea, however, is the reduction of performance data as measured for a series of programs to a single number. Hence, extensions to other performance environments are straightforward; for example, one could consider transactions or logical inferences per second just as easily.

Many of the problems of reducing performance to a single number, at least in scientific computing, have become apparent with the introduction of vector computers. The problem will be exacerbated as highly parallel computers come into widespread use. As will be made clear in this article, the problem results from the considerable disparity between performance on fast and slow programs. This difference can be two or more orders of magnitude, and not all single-number measures are well suited for dealing with differences of this scale. We will focus on how the wide variance in performance over individual programs distorts commonly used aggregate measures.

The use of peak performance as a single-number performance measure is an extreme example of the problems that can occur. The peak performance is the performance that can be achieved under absolutely optimum conditions, and then perhaps only for a brief period of time. With vector and parallel processing, extremely high peak performance numbers are possible. Peak performance, however, is generally recognized as being of no value for predicting actual performance on the variety of real programs that are actually used. In fact, the fallacies of using peak performance are so widely recognized that we will not deal with it any further. Rather, we will concentrate on attempts to summarize the *typical* performance that will be observed on real programs. Problems with single-number measures are more subtle here than with peak performance, yet they can result in gross distortions of actual performance.

We consider benchmarking the preferred method for determining typical performance. Our goal is not to provide means for accurate benchmarking, but to study ways of reducing benchmark performance results to a single number that maintains the accuracy of the original benchmarks. In situations where the workload is clearly understood, both in terms of the actual programs to be run and their relative usage, benchmark results summarized by a single performance number can actually be a very accurate predictor of performance.

A fundamental premise of this article is that *the time required to perform a specified amount of computation is the ultimate measure of computer performance*. When a set of programs is actually run on a specific computer, each will execute for a certain amount of time, and the computer will be busy for the total time consumed by the programs. If the same set of programs is run on several computers, the computer that finishes the set of programs first (the one consuming the less total time) is available to do more work. Simply put, we consider one computer to be faster than another if it executes the same set of programs in less time.

Example

Let us consider a simple example. Say that a typical workload to be evaluated consists of two component programs, so two benchmarks are used to model the workload. Further assume that we have successfully run the two benchmarks on each of three computers. The results are summarized in Table I.

TABLE I. Performance of Three Computers on Two Benchmarks

Benchmark	Millions of floating pt. ops.	Computer 1 time (secs.)	Computer 2 time (secs.)	Computer 3 time (secs.)
Program 1	100	1	10	20
Program 2	100	1000	100	20
Total Time		1001	110	40

The columns in the table represent the number of floating point operations in each benchmark and the time required for each. For simplicity, we have chosen example programs that do the same amount of work (floating point operations). For now, we will use versions of performance measures that give all the programs equal weight. More general, weighted performance measures will be discussed later.

Now we are faced with the problem of summarizing the performance of each computer with a single number. One way to do this is to simply add the individual benchmark times to arrive at the total time. The total times are given at the bottom of Table I. Another commonly used way to summarize performance is to first express performance as a rate (mflops), and then reduce the rates for the individual benchmarks to a single number. The arithmetic mean, geometric mean, and harmonic mean are commonly used. Details on calculating each are given in later sections. Table II contains the performance data from Table I converted to mflops and the mean performance measures as expressed by the arithmetic, geometric, and harmonic means.

First consider the total times from Table I. Computer 3 is almost three times faster than Computer 2, and 25 times faster than Computer 1. Using arithmetic mean mflops, Computer 1 is roughly 10 times faster than Computers 2 and 3; Computer 2 is slightly faster than Computer 3. Using geometric mean, Computer 3 is faster than Computers 1 and 2; Computers 1 and 2 have the same performance. Finally, with harmonic mean, relative performance is similar to that achieved with total time. The three mean mflops measures are in total disagreement with one another, not just in terms of absolute performance but in relative performance as well. The only agreement between any of the measures is between total time and the harmonic mean.

TABLE II. Performance of Benchmarks in Mflops

Benchmark	Computer 1	Computer 2	Computer 3
Program 1	100.0 mflops	10.0 mflops	5.0 mflops
Program 2	.1 mflops	1.0 mflops	5.0 mflops
Arith. Mean	50.1 mflops	5.5 mflops	5.0 mflops
Geom. Mean	3.2 mflops	3.2 mflops	5.0 mflops
Harm. Mean	.2 mflops	1.8 mflops	5.0 mflops

Properties of Good Performance Measures

The principle behind benchmarking is to model a real job mix with a smaller set of representative programs. If a set of benchmarks is chosen well, each program in the real job mix has the same performance characteristics as one or more of the benchmark programs. In addition, weights can be used to proportion each benchmark type with similar types of computation in the underlying workload. In this way, a weighted performance measure can be used for the benchmarks such that the performance measure is directly proportional to performance on the mix of real programs.

As stated earlier, we consider the ultimate performance measure to be the actual time needed to perform a specific amount of work. If we use time as the unit of performance, we arrive at the following property.

Property 1. A single-number performance measure for a set of benchmarks expressed in units of time should be directly proportional to the total (weighted) time consumed by the benchmarks.

Even though time is a good unit for performance measurement, it is common in practice to measure performance as operations per unit time rather than time itself. In scientific computing, performance is often expressed in millions of floating point operations per second (mflops).

When benchmarks are run and performance is calculated in mflops, performance should be highly correlated with the time measure. That is, if computer 1 is t times faster than computer 2 when time is used, then computer 1 should also be t times faster than computer

2 when rate (mflops) is used. Since rate is inversely proportional to time for a given amount of computation, this gives us

Property 2. A single-number performance measure for benchmarks expressed as a rate should be inversely proportional to the total (weighted) time consumed by the benchmarks.

SINGLE-NUMBER PERFORMANCE MEASURES

In light of the preceding discussion, the most obvious single-number performance measure is total time. This measure is not only accurate, but has considerable intuitive appeal.

Using time as a benchmark measure has another, less obvious, advantage. If one were to quote the performance of a computer as 10 seconds, the obvious question is: For what program(s)? That is, it forces a clear explanation of the benchmark used to arrive at the performance figure. On the other hand, if we use a rate measure like mflops and quote performance as 10 mflops, the same question does not seem to come as easily, although it is equally important as when time is used as a measure.

When performance is measured as a rate, single-number performance measures can become quite misleading, as our earlier example showed. It is common practice to measure mflops for each component program in a benchmark suite, then use these numbers to calculate an aggregate performance number. This is exactly what is done in Table II for arithmetic, geometric, and harmonic means. We will now define and discuss the properties of each of these means to see how well they satisfy Property 2.

We first define the means assuming equal weights; that is, each benchmark performs the same amount of *work* as measured by the number of floating point operations. (Weighted means are discussed in a later section). We assume a total of n benchmarks. For benchmark i, let M_i be the performance measured in mflops. Although the individual benchmarks may perform different numbers of floating point operations, using mflops has the effect of scaling out these differences. In general, we let F_i be the number of floating point operations in benchmark i, but because we are weighting the benchmarks equally in this section, we can simply let F be a constant number of floating point operations, and use scaled benchmark times T_i so that $M_i = F/T_i$.

Arithmetic Mean

Arithmetic mean mflops is defined in the following way:

$$A\text{-mean} = \sum_{i=1}^{n} \frac{M_i}{n}.$$

If we substitute F/T_i for M_i, we arrive at

$$A\text{-mean} = \sum_{i=1}^{n} \frac{F}{T_i} \Big/ n.$$

By inspecting this second equation, we see that arithmetic mean expresses performance in a way that is directly proportional to the sum of the inverses of the times. It is not inversely proportional to the sum of the times. Consequently, the arithmetic mean fails Property 2. The example in Section 1.1 clearly illustrates the inappropriateness of using arithmetic mean as applied to mflops. The arithmetic mean mflops bear no relationship to the time consumed by the benchmarks when they are actually executed.

The uselessness of arithmetic mean as a performance predictor cannot be emphasized enough. Giving additional statistics such as standard deviation to supplement the arithmetic mean does not mitigate the situation. Using arithmetic mean with a standard deviation is similar to saying: Here is a meaningless performance measure (i.e., arithmetic mean), and here is a measure (i.e., standard deviation) of just how meaningless it is.

On the other hand, the arithmetic mean of benchmark *times* does satisfy Property 1 since, for a given number of benchmarks, the arithmetic mean is total time divided by a constant. Hence, arithmetic mean *time is an accurate performance measure, although it is not often used.*

Geometric Mean

The unweighted geometric mean mflops is expressed as

$$G\text{-mean} = \left(\prod_{i=1}^{n} M_i\right)^{1/n}.$$

Again, performance is not accurately summarized since it does not have the correct inverse relationship with total time. Geometric mean has been advocated for use with performance numbers that are normalized with respect to one of the computers being compared [2]. The geometric mean has the property of performance relationships consistently maintained regardless of the computer that is used as the basis for normalization. The geometric mean does provide a consistent measure in this context, but it is consistently wrong. The solution to the problem of normalizing with respect to a given computer is not to use geometric mean, as suggested in [2], but to always normalize results *after* the appropriate aggregate measure is calculated, not before. This last point can be illustrated by using an example from [2]. Table III is taken from Table IX in [2].

The performance values are expressed in an unspecified unit of time. Since time is used, calculating a weighted sum or arithmetic mean and then (optionally) normalizing the results is an accurate method for determining aggregate performance. The authors make this same point (Rule 3) and use Table IX to illustrate it [2]. If we apply the (weighted) geometric mean as advocated in [2] (Rule 2.2), however, we get contradictory results. The geometric mean numbers have been added to the bottom of Table III. Using the arithmetic mean leads to the conclusion that Processor Y is slowest. Using geometric mean leads to the contradictory conclusion that processor Y is fastest. Normalizing the benchmark results with respect to any of the processors and then finding the geometric mean will give the same relative results as taking the geometric mean of the raw

TABLE III. Example Taken From Table IX of Reference [2]

Benchmark	Weight	Processor X	Processor Y	Processor Z
Program 1	0.6	20	10	40
Program 2	0.4	40	80	20
Weighted arithmetic mean		28	38	32
Normalized to X		1.00	1.36	1.14
Weighted geometric mean		26.4	23.0	30.3
Normalized to X		1.00	.87	1.15

data. None of these geometric means, however, provide performance that is proportional to the actual time consumed when running the weighted benchmark mix. Only the arithmetic mean of the times has this desirable property.

Harmonic Mean

Harmonic mean mflops are expressed as

$$H\text{-mean} = n \Big/ \sum_{i=1}^{n} \frac{1}{M_i},$$

The formula for harmonic mean may appear at first to lack intuitive appeal, but it is equivalent to taking the total number of floating point operations and dividing by the total time. This can be illustrated in the following way. By substituting F/T_i for M_i,

$$H\text{-mean} = n \Big/ \sum_{i=1}^{n} \frac{T_i}{F}.$$

This is equivalent to $n/[(1/F)\sum_{i=1}^{n} T_i]$. This, in turn, is equivalent to $nF/\sum_{i=1}^{n} T_i$. Now, since nF is the total number of floating point operations, and $\sum_{i=1}^{n} T_i$ is the total time, we have just shown that the harmonic mean is the total number of operations divided by total time.

Using the second of the above expressions for harmonic mean, we see that it is inversely proportional to the sum of the benchmark times, and thus satisfies Property 2. In the example in Section 1.1, the harmonic mean performance numbers are entirely consistent with the total time performance measures.

When we run a single program and calculate its performance as a rate, we determine the total number of operations, then divide by the total time. Why should it be any different when the workload is divided into two different programs? If we take the total number of operations over both programs and divide by the total time, we get the actual performance for the work done.

Weighted Means

The arithmetic, geometric, and harmonic means were discussed earlier using equal weightings to clearly show their properties. To properly weight the means, we assign a weight w_i to each of the benchmarks. The weights add to 1 and represent the fraction of the workload done by benchmark i. Weighted versions of each of the means are as follows:

$$A\text{-mean} = \sum_{i=1}^{n} w_i M_i$$

$$G\text{-mean} = \prod_{i=1}^{n} (M_i)^{w_i}$$

$$H\text{-mean} = 1 \Big/ \sum_{i=1}^{n} \frac{w_i}{M_i}.$$

The weighted versions of the means have properties similar to the unweighted ones. Specifically, the harmonic mean is the only one that provides an accurate relationship to the actual time required to execute the benchmark programs.

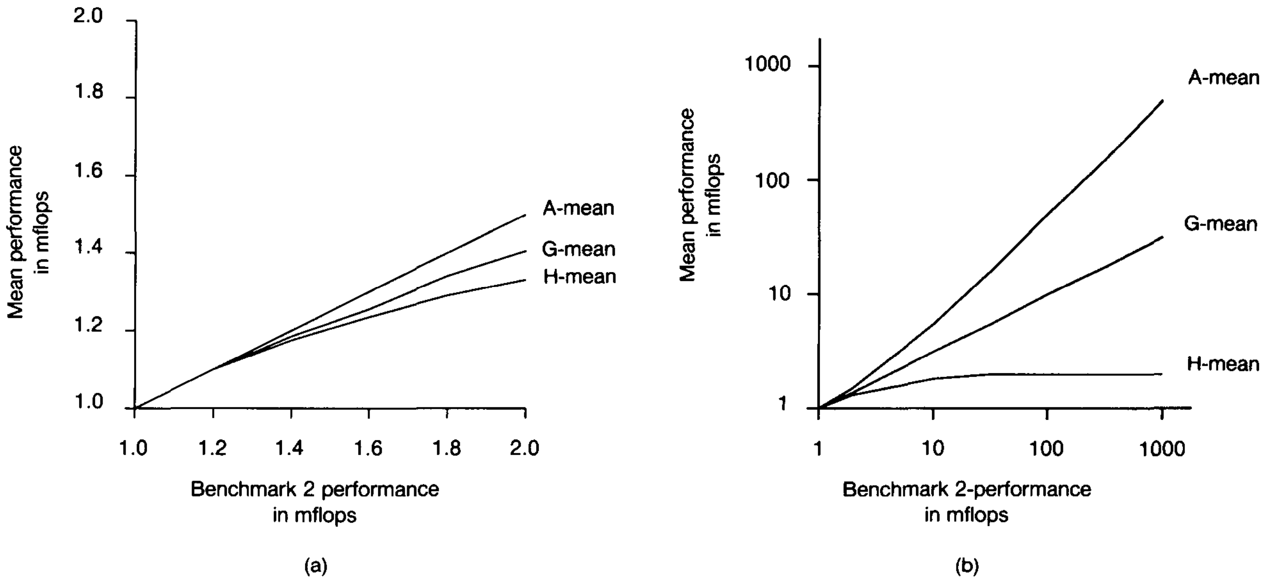

FIGURE 1. Mean Performance on Two Benchmarks When Benchmark 1 is a Constant 1.0 mflops. (a) Benchmark 2 varies from 1.0 to 2.0 mflops. (b) Benchmark 2 varies from 1 to 1000 mflops.

DISCUSSION AND CONCLUSIONS

Initially, this article claimed that problems with some of the single-number performance measures have become more evident with the introduction of vector and parallel computers. We can now show this graphically. Let us assume two benchmarks are run, and benchmark 1 always performs at a constant rate of 1.0 mflops. We vary the performance of benchmark 2 and plot the arithmetic, geometric, and harmonic means of the two benchmarks. First, in Figure 1a, we let the performance of benchmark 2 vary from 1.0 to 2.0 mflops. That is, performance on the two programs is approximately equal. We see that there is relatively little difference between the three means. They are equal when the performance of benchmark 2 is 1.0; otherwise, the arithmetic mean is slightly higher than the geometric mean, which is slightly higher than the harmonic mean. In Figure 1b, the performance of benchmark 2 is varied from 1 to 1000 mflops. We now see substantial differences in the three means, literally orders of magnitude. Note that Figure 1b uses logarithmic axes. The arithmetic mean increases linearly with the performance of benchmark 2. The geometric mean increases as the square root of the performance of benchmark 2, and the harmonic mean asymptotically approaches the constant 2.0 mflops. The harmonic mean is in accord with Amdahl's law, which, when applied to this example, asserts that making the second program infinitely fast will only have the total time used by both programs. (An interesting discussion of Amdahl's law and several related issues can be found in [3].) We see that geometric mean does not overstate true performance as much as arithmetic mean, but still can overstate performance substantially, especially as the performance difference between fast and slow programs becomes large.

In this article we have reached the following conclusions:

(1) Arithmetic mean can be used as an accurate measure of performance expressed as time. On the other hand, it should not be used for summarizing performance expressed as a rate such as mflops.

(2) Geometric mean should not be used for summarizing performance expressed as a rate or as a time.

(3) Harmonic mean should be used for summarizing performance expressed as a rate. It corresponds accurately with computation time that will actually be consumed by running real programs. Harmonic mean, when applied to a rate, is equivalent to calculating the total number of operations divided by the total time.

(4) If performance is to be normalized with respect to a specific machine, an aggregate performance measure such as total time or harmonic mean rate should be calculated *before* any normalizing is done. That is, benchmarks should not be individually normalized first.

This article has only examined the problem of reducing data to a single number that accurately characterizes performance. Issues related to acquiring the benchmark data are, at least, as important, and are much more difficult. In the realm of scientific computing, [1, 4] contain excellent discussions of the entire benchmarking process.

REFERENCES

1. Dongarra, J., Martin, J.L., and Worlton, J. Computer benchmarking: Paths and pitfalls. *IEEE Spectrum 24*, 7 (July 1987), 38–43.
2. Fleming, P.J., and Wallace, J.J. How not to lie with statistics: The correct way to summarize benchmark results. *Commun. ACM 29*, 3 (Mar. 1986), 218–221.
3. Hack, J.J. Peak vs. sustained performance in highly concurrent vector machines. *Comput. 19*, 9 (Sept. 1986), 11–19.
4. Worlton, J. Understanding supercomputer benchmarks. *Datamation 30*, 4 (Sept. 1, 1984), 121–130.

CR Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Measurement techniques, Performance attributes

General Terms: Measurement, Performance

Additional Key Words and Phrases: Arithmetic mean, benchmarking, geometric mean, harmonic mean

Received 2/88; accepted 4/88

ABOUT THE AUTHOR:

JAMES E. SMITH is system architect for a series of high performance scientific computer systems being developed by the Astronautics Corporation of America in Madison, Wisconsin. He has also been on the faculty of the University of Wisconsin–Madison since 1976 and is currently an associate professor in the Department of Electrical and Computer Engineering. His research interests include high speed processor and system design, particularly multiprocessor systems. Author's present address: James E. Smith, Astronautics Technology Center, 5800 Cottage Grove Road, Madison, WI 53716-1387.