In More Depth: MFLOPS as a Performance Metric

Another popular alternative to execution time is *million floating-point operations per second*, abbreviated *megaFLOPS* or *MFLOPS* but always pronounced "megaflops." The formula for MFLOPS is simply the definition of the acronym:

$$\text{MFLOPS} = \frac{\text{Number of floating-point operations in a program}}{\text{Execution time} \times 10^6}$$

A *floating-point operation* is an addition, subtraction, multiplication, or division operation applied to a number in a single or double precision floating-point representation. Such data items are heavily used in scientific calculations and are specified in programming languages using key words like *float*, *real*, *double*, or *double precision*.

Clearly, a MFLOPS rating is dependent on the program. Different programs require the execution of different numbers of floating-point operations (see Exercise for an example). Since MFLOPS were intended to measure floating-point performance, they are not applicable outside that range. Compilers, as an extreme example, have a MFLOPS rating near 0 no matter how fast the computer is, because compilers rarely use floating-point arithmetic.

Because it is based on operations in the program rather than on instructions, MFLOPS has a stronger claim than MIPS to being a fair comparison between different computers. The key to this claim is that the same program running on different computers may execute a different number of instructions but will always execute the same number of floating-point operations. Unfortunately, MFLOPS is not dependable because the set of floating-point operations is not consistent across computers, and the number of actual floating-point operations performed may vary. For example, the Cray-2 has no divide instruction, while the Motorola 68882 has divide, square root, sine, and cosine. Thus several floating-point operations are needed on the Cray-2 to perform a floating-point division, whereas on the Motorola 68882, a call to the sine routine, which would require performing several floating-point operations on most computers, would require only one operation.

Another potential problem is that the MFLOPS rating changes according not only to the mixture of integer and floating-point operations but to the mixture of fast and slow floating-point operations. For example, a program with 100% floating-point adds will have a higher rating than a program with 100% floating-point divides. The solution to both these problems is to define a method of counting the number of floating-point operations in a high-level language program. This counting process can also weight the operations, giving more complex operations larger weights, allowing a computer to achieve a high MFLOPS rating even if the program contains many floating-point divides. These MFLOPS might be called *normalized MFLOPS*. Of course, because of the counting and weighting, these normalized MFLOPS may be very different from the actual rate at which a computer executes floating-point operations.

Like any other performance measure, the MFLOPS rating for a single program cannot be generalized to establish a single performance metric for a computer. The use of the same term to refer to everything from peak performance (the maximum MFLOPS rate possible for any code segment), to the MFLOPS rate for one benchmark, to a normalized MFLOPS rating, only increases the confusion. The worst of these variants of MFLOPS, peak MFLOPS, is unrelated to actual performance; the best variant is redundant with execution time, our principal measure of performance. Yet, unlike execution time, it is tempting to characterize a computer with a single MFLOPS rating without naming the program or input.

4.30 [5] <§4.3> Find the MFLOPS ratings for each of the two programs on each computer in Figure 4.8.3, assuming that each floating-point operation counts as 1 FLOP. How do the MFLOPS ratings for programs 1 and 2 compare for each computer? Does the example illustrate one of the problems discussed above?

4.31 [15] <§4.3> If performance is expressed as a rate, such as MFLOPS, then a higher rating and a higher average indicate better performance. When performance is expressed as a rate, the average that tracks total execution time is the *harmonic mean* (HM):

$$\mathrm{HM} = \frac{n}{\sum_{i=1}^{n} \frac{1}{\mathrm{Rate}_i}}$$

Each Rate_i is $1/\mathrm{Time}_i$, where Time_i is the execution time for the ith of n programs in the workload. Prove that the harmonic mean of a set of rates tracks execution time by showing that it is the inverse of the arithmetic mean of the corresponding execution times.

4.32 [4 hours] <§4.3> Devise a program in C or Fortran that determines the peak MFLOPS rating for a computer. Run it on two computers to calculate the peak MFLOPS. Now run a real floating-point program on both computers. How well does peak MFLOPS predict performance of the real floating-point program?