Measuring Parallel Processor Performance

Many metrics are used for measuring the performance of a parallel algorithm running on a parallel processor. This article introduces a new metric that has some advantages over the others. Its use is illustrated with data from the Linpack benchmark report and the winners of the Gordon Bell Award.

Alan H. Karp and Horace P. Flatt

There are many ways to measure the performance of a parallel algorithm running on a parallel processor. The most commonly used measurements are the elapsed time, price/performance, the speed-up, and the efficiency. This article defines another metric which reveals aspects of the performance that are not easily discerned from the other metrics.

The elapsed time to run a particular job on a given machine is the most important metric. A Cray Y-MP/1 solves the order 1,000 linear system in 2.17 seconds compared to 445 seconds for a Sequent Balance 21000 with 30 processors [2]. If you can afford the Cray and you spend most of your time factoring large matrices, then you should buy a Cray.

Price/performance of a parallel system is simply the elapsed time for a program divided by the cost of the machine that ran the job. It is important if there are a group of machines that are "fast enough." Given a fixed amount of money, it may be to your advantage to buy a number of slow machines rather than one fast machine. This is particularly true if you have many jobs to run and a limited budget. In the previous example, the Sequent Balance is a superior price/performer than the Cray if it costs less than 0.5 percent as much. On the other hand, if you can't wait 7 minutes for the answer, the Sequent is not a good buy even if it wins in price/performance.

These two measurements are used to help you decide what machine to buy. Once you have bought the machine, speed-up and efficiency are the measurements often used to let you know how effectively you are using it.

The speed-up is generally measured by running the same program on a varying number of processors. The speed-up is then the elapsed time needed by 1 processor divided by the time needed on p processors, $s = T(1)/T(p)$. (Of course, the correct time for the uniprocessor run would be the time for the best serial algorithm, but almost nobody bothers to write two programs.) If you are interested in studying algorithms for parallel processors, and system A gives a higher speed-up than system B for the same program, then you would say that system A provides better support for parallelizing this program than does system B.

An example of such support is the presence of more processors. A Sequent with 30 processors will almost certainly produce a higher speed-up than a Cray with only 8 processors.

The issue of efficiency is related to that of price/performance. It is usually defined as

$$e = \frac{T(1)}{pT(p)} = \frac{s}{p}. \quad (1)$$

Efficiency close to unity means that you are using your hardware effectively; low efficiency means that you are wasting resources. As a practical matter, you may buy a system with 100 processors that each takes 100 times longer than you are willing to wait to solve your problem. If you can code your problem to run at high efficiency, you'll be happy. Of course, if you have 200 processors, you may not be unhappy with 50 percent efficiency, particularly if the 200 processors cost less than other machines that you can use.

Each of these metrics has disadvantages. In fact, there is important information that cannot be obtained even by looking at all of them. It is obvious that adding processors should reduce the elapsed time, but by how much? That is where speed-up comes in. Speed-up close to linear is good news, but how close to linear is good enough? Well, efficiency will tell you how close you are getting to the best your hardware can do, but if your efficiency is not particularly high, why? The new metric defined in the following section is intended to answer these questions.

NEW METRIC

We will now derive our new metric, the experimentally determined serial fraction, and show why it is useful. We will start with Amdahl's Law [1] which in its simplest form says that

$$T(p) = T_s + \frac{T_p}{p}, \tag{2}$$

where T_s is the time taken by the part of the program that must be run serially and T_p is the time in the parallelizable part. Obviously, $T(1) = T_s + T_p$. If we define the fraction serial, $f = T_s/T(1)$ then equation (2) can be written as

$$T(p) = T(1)f + \frac{T(1)(1-f)}{p}, \tag{3}$$

or, in terms of the speed-up s

$$\frac{1}{s} = f + \frac{1-f}{p}. \tag{4}$$

We can now solve for the serial fraction, namely

$$f = \frac{1/s - 1/p}{1 - 1/p}. \tag{5}$$

The experimentally determined serial fraction is our new metric. While this quantity is mentioned in a large percentage of papers on parallel algorithms, it is virtually never used as a diagnostic tool the way speed-up and efficiency are. It is our purpose to correct this situation.

The value of f is useful because equation (2) is incomplete. First, it assumes that all processors compute for the same amount of time, i.e., the work is perfectly load balanced. If some processors take longer than others, the measured speed-up will be reduced giving a larger measured serial fraction. Second, there is a term missing that represents the overhead of synchronizing processors.

Load-balancing effects are likely to result in an irregular change in f as p increases. For example, if you have 12 pieces of work to do that take the same amount of time, you will have perfect load balancing for 2, 3, 4, 6, and 12 processors, but less than perfect load balancing for other values of p. Since a larger load imbalance results in a larger increase in f, you can identify problems not apparent from speed-up or efficiency.

The overhead of synchronizing processors is a monotonically increasing function of p, typically assumed to increase either linearly in p or as $\log p$. Since increasing overhead decreases the speed-up, this effect results in a smooth increase in the serial fraction f as p increases. Smoothly increasing f is a warning that the granularity of the parallel tasks is too fine.

A third effect is the potential reduction of vector lengths for certain parallelizations of a particular algorithm. Vector processor performance normally increases as vector length increases except for vector lengths slightly larger than the length of the vector registers. If the parallelization breaks up long vectors into shorter vectors, the time to execute the job can increase. This effect then also leads to a smooth increase in the measured serial fraction as the number of processors increases. However, large jobs usually have very long vectors, vector processors usually have only a few processors (the Intel iPSC-VX is an exception), and there are usually parallelizations that keep the vector lengths fixed. Thus, the reduction in vector length is rarely a problem and can often be avoided entirely.

In order to see the advantage of the serial fraction over the other metrics, look at Table I which is extracted from Table 2 of the Linpack report [2]. The Cray Y-MP shows speed-ups ranging from 1.95 to 6.96. Is that good? Even if you look at the efficiency, which ranges from 0.975 to 0.870, you still don't know. Why does the efficiency fall so rapidly? Is there a lot of overhead when using 8 processors? The serial fraction, f, answers the question; the serial fraction is nearly constant for all values of p. The loss of efficiency is due to the limited parallelism of the program.

The single data point for the Sequent Balance reveals that f performs better as a metric as the number of processors grows. The efficiency of the 30 processor Sequent is only 83 percent. Is that good? Yes, it is since the serial fraction is only 0.007.

The data for the Alliant FX/40 shows something different. Here the speed-up ranges from 1.90 to 3.22 and the efficiency from 0.950 to 0.805. Although neither of these measurements tells much, the fact that f ranges from 0.053 to 0.080 does; there is some overhead that is increasing as the number of processors grows. We can't tell what this overhead is due to—synchronization cost, memory contention, or what—but at least we know it is there. The effect on the FX/80 is much smaller although there is a slight increase in f, especially for fewer than 5 processors.

Even relatively subtle effects can be seen. The IBM 3090 has a serial fraction under 0.007 for 2 and 3 processors, but over 0.011 for 4 or more. Here the reason is most likely due to the machine configuration; each set of 3 processors is in a single frame and shares a memory management unit. Overhead increases slightly when two of these units must coordinate their activities. This effect also shows up on the 3090-280S which has two processors in two frames. Its run has twice the serial fraction as does the run on the 3090-200S. None of the other metrics would have revealed this effect.

Another subtle effect shows up on the Convex. The 4 processor C-240 shows a smaller serial fraction than does the 2 processor C-220. Since the same code was presumably run on both machines, the actual serial fraction must be the same. How can the measured value decrease? This appears to be similar to the "superlinear" speed-ups reported on some machines. As in those cases, adding processors adds cache and memory bandwidth which reduces overhead. Perhaps that is the case here.

Care must be used when comparing different machines. For example, the serial fraction on the Cray is 3 times larger than on the Sequent. Is the Cray really that inefficient? The answer is no. Since almost all the parallel work can be vectorized, the Cray spends relatively less time in the parallel part of the code than does the Sequent which has no vector unit. Since the parallelizable part speeds up more than the serial part which

Table I. Summary of Linpack report Table 2

Computer	p	Time(sec)	s	e	f
Cray Y-MP/8	1	2.17	—	—	—
Cray Y-MP/8	2	1.11	1.95	0.975	0.024
Cray Y-MP/8	3	0.754	2.88	0.960	0.021
Cray Y-MP/8	4	0.577	3.76	0.940	0.021
Cray Y-MP/8	8	0.312	6.96	0.870	0.021
IBM 3090-180S VF	1	7.27	—	—	—
IBM 3090-200S VF	2	3.64	2.00	1.000	0.002
IBM 3090-280S VF	2	3.65	1.99	0.995	0.004
IBM 3090-300S VF	3	2.46	2.96	0.987	0.007
IBM 3090-400S VF	4	1.89	3.85	0.963	0.013
IBM 3090-500S VF	5	1.52	4.78	0.956	0.011
IBM 3090-600S VF	6	1.29	5.64	0.940	0.012
Alliant FX/40	1	66.1	—	—	—
Alliant FX/40	2	34.8	1.90	0.950	0.053
Alliant FX/40	3	24.9	2.65	0.883	0.066
Alliant FX/40	4	20.5	3.22	0.805	0.080
Alliant FX/80	1	57.7	—	—	—
Alliant FX/80	2	29.8	1.94	0.970	0.032
Alliant FX/80	3	20.7	2.79	0.930	0.038
Alliant FX/80	4	16.2	3.56	0.890	0.041
Alliant FX/80	5	13.6	4.24	0.848	0.045
Alliant FX/80	6	11.8	4.89	0.815	0.046
Alliant FX/80	7	10.6	5.44	0.777	0.048
Alliant FX/80	8	9.64	5.99	0.749	0.048
Sequent	1	11111	—	—	—
Sequent	30	445	25.0	0.833	0.007
Convex C-210	1	15	—	—	—
Convex C-220	2	7.98	1.88	0.940	0.064
Convex C-240	4	4.03	3.72	0.930	0.025

Note: p=#processors, s=speed-up, e=efficiency, f=serial fraction

has less vector content, the fraction of the time spent in serial code is increased.

The Linpack benchmark report measures the performance of a computational kernel running on machines with no more than 30 processors. The results in Table II are taken from the work of the winners of the Gordon Bell Award [5]. Three applications are shown with maximum speed-ups of 639, 519, and 351 and efficiencies ranging from 0.9965 to 0.3430. We know this is good work since they won the award, but how good a job did they do? The serial fraction ranges from 0.00051 to 0.0019 indicating that they did a very good job, indeed.

The serial fraction reveals an interesting point. On all three problems, there is a significant reduction in the serial fraction in going from 4 to 16 processors (from 16 to 64 for the wave motion problem). As with the Convex results, these numbers indicate something akin to "superlinear" speed-up. Perhaps the 4-processor run sends longer messages than does the 16-processor run, and these longer messages are too long for the system to handle efficiently. At any rate, the serial fraction has pointed up an inconsistency that needs further study.

SCALED SPEED-UP

All the analysis presented so far refers to problems of fixed size. Gustafson [4] has argued that this is not how parallel processors are used. He argues that users will increase their problem size to keep the elapsed time of the run more or less constant. As the problem size grows, we should find the fraction of the time spent executing serial code decreases, leading us to predict a decrease in the measured serial fraction.

If we assume that the serial time and overhead are independent of problem size, neither of which is fully justified, [3]

$$T(p, k) = T_s + \frac{kT_p}{p}, \qquad (6)$$

where $T(p, k)$ is the time to run the program on p processors for a problem needing k times more arithmetic. Here k is the scaling factor and $k = 1$ when $p = 1$. Flatt [3] points out that the scaling factor k must count arithmetic, not some more convenient measure such as memory size.

Our definition of speed-up must now account for the additional arithmetic that must be done to solve our

Table II. Summary of Bell Award winning performance

p	s	e	f
	Wave Motion		
4	3.986	0.9965	0.0012
16	15.86	0.9913	0.00097
64	62.01	0.9689	0.00051
256	226.2	0.8836	0.00052
1024	639.0	0.6240	0.00059
	Fluid Dynamics		
4	3.959	0.9898	0.0035
16	15.47	0.9669	0.0023
64	58.53	0.9145	0.0015
256	201.6	0.7875	0.0011
1024	519.1	0.5069	0.00095
	Beam Stress		
4	3.954	0.9885	0.0039
16	15.46	0.9663	0.0023
64	57.46	0.8978	0.0018
256	177.5	0.6934	0.0017
1024	351.2	0.3430	0.0019

Note: p=#processors, s=speed-up, e=efficiency, f=serial fraction

larger problem. We can use

$$s_k = \frac{kT(1, 1)}{T(p, k)}. \tag{7}$$

The scaled efficiency is then $e_k = s_k/p$, and the scaled serial fraction becomes

$$f_k = \frac{1/s_k - 1/p}{1 - 1/p}. \tag{8}$$

By our previous definitions we see that $f = kf_k$ which under ideal circumstances would remain constant as p increases.

The scaled results are shown in Table III. Although these runs take constant time as the problem size grows, the larger problems were run with shorter time steps. A better scaling would be one in which the time integration is continued to a specific value.

If the run time were still held constant, the problems would scale more slowly than linearly in p. In these examples, the Courant condition limits the step size which means that the correct scaling would be $k = \sqrt{p}$. The scaling chosen for the problems of Table III is $k = p$.

As predicted [3], the efficiency decreases, barely, as p increases even for scaled problems. The scaled serial fraction, on the other hand, decreases smoothly. This fact tells us that the decreasing efficiency is not caused by a growing serial fraction. Instead it tells us the problem size is not growing fast enough to completely counter the loss of efficiency as more processors are added.

The variation in f_k as the problem size grows makes it difficult to interpret. If we allow for the fact that ideally the increase in the problem size does not affect the serial work, we again have a metric that should remain constant as the problem size grows. Table III shows how kf_k varies with problem size. We see that this quantity grows slowly for the wave motion problem, but that there is virtually no increase for the fluid dynamics problem. These results indicate that the serial work increases for the wave motion problem as the problem size grows but not for the fluid dynamics problem. The irregular behavior of kf_k for the beam stress problem warrants further study.

SUMMARY

We have shown that the measured serial fraction, f, provides information not revealed by other commonly used metrics. The metric, properly defined, may also be useful if the problem size is allowed to increase with the number of processors.

What makes the experimentally determined serial fraction such a good diagnostic tool of potential per-

Table III. Bell Award scaled problems. $k=p$

p	s_k	e_k	f_k	kf_k
		Wave Motion		
4	3.998	0.9995	0.00013	0.00053
16	15.95	0.9969	0.00020	0.0032
64	63.61	0.9939	0.000097	0.0062
256	254.1	0.9926	0.000029	0.0074
1024	1014	0.9902	0.0000098	0.010
		Fluid Dynamics		
4	3.992	0.9980	0.00067	0.0027
16	15.96	0.9975	0.00015	0.0024
64	63.82	0.9972	0.000046	0.0029
256	255.2	0.9969	0.000013	0.0033
1024	1020	0.9961	0.0000033	0.0034
		Beam Stress		
4	4.001	1.000	0.0	0.0
16	16.00	1.000	0.000021	0.00034
64	63.96	0.9994	0.000015	0.00098
256	255.8	0.9992	0.0000038	0.00096
1024	1023	0.9990	0.0000012	0.0013

Note: p=#processors, s_k=scaled speed-up, e_k=scaled efficiency, f_k=scaled serial fraction

formance problems? While elapsed time, speed-up, and efficiency vary as the number of processors increases, the serial fraction would remain constant in an ideal system. Small variations from perfect behavior are much easier to detect from something that should be constant than from something that varies. Since kf_k for scaled problems shares this property, it, too, is a useful tool.

Ignoring the fact that p takes on only integer values makes it easy to show that

$$\frac{d}{dp}\frac{1}{e} = f \tag{9}$$

if we ignore the overhead and assume that the serial fraction is independent of p. Thus, we see that the serial fraction is a measure of the rate of change of the efficiency. Even in the ideal case, $1/e$ increases linearly as the number of processors increases. Any deviation of $1/e$ from linearity is a sign of lost parallelism. The fraction serial is a particularly convenient measure of this deviation from linearity.

The case of problem sizes that grow as the number of processors is increased is only slightly more complicated. In this case

$$\frac{d}{dp}\frac{1}{e_k} = f_k + (p-1)\frac{df_k}{dp} = \frac{f}{k}\left(1 - \frac{p-1}{k}\frac{dk}{dp}\right). \tag{10}$$

If $k = 1$, *i.e.*, there is no scaling, equation (10) reduces to equation (9). If $k = p$, then $1/e$ has a slope of f/p^2 which is a very slow loss of efficiency. Other scalings lie between these two curves.

It is easy to read too much into the numerical values. When the efficiency is near unity, $1/s$ is close to $1/p$ which leads to a loss of precision when subtracting in the numerator of equation (5). If care is not taken, variations may appear that are mere round-off noise. All entries in the tables were computed from

$$f = 1 - \frac{1 - 1/s}{1 - 1/p}. \tag{11}$$

Since both s and p are considerably greater than unity and neither $1/s$ nor $1/p$ is near the precision of the floating point arithmetic, there is only one place where precision can be lost. Rounding the result to the significance of the reported times guarantees that the results are accurate. Similarly,

$$f_k = 1 - \frac{1 - 1/s_k}{1 - 1/p} \tag{12}$$

is used for the scaled serial fraction.

Although our numerical examples come from rather simple cases, one of us (Alan Karp) has successfully used this metric to find a performance bug in one of his applications. Noting an irregularity in the behavior of f led him to examine the way the IBM Parallel Fortran prototype compiler was splitting up the work in the loops. Due to an oversight, the compiler truncated the result of dividing the number of loop iterations by the number of processors. This error meant that one processor had to do one extra pass to finish the work in the loop. For some values of p this remainder was small; for others, it was large. The solution was to increase his problem size from 350, which gives perfect load balancing on his six-processor IBM 3090-600S only for 2 and 5 processors, to 360 which always balances perfectly. (This error was reported to the IBM Parallel Fortran compiler group.)

REFERENCES

1. Amdahl, G.M. Validity of the single processor approach to achieving large scale computer capabilities. In *Proceedings of AFIPS Spring Joint Computer Conference, 30*. Atlantic City, NJ, 1967.
2. Dongarra, J.J. Performance of various computers using standard linear equation software. Report CS-89-85, Computer Science Department, Univ. Tennessee, Knoxville, October 12, 1989.
3. Flatt, H.P., and Kennedy, K. Performance of parallel processors. *Parallel Comput. 12*, (Oct. 1989), 1–20.
4. Gustafson, J.L. Reevaluating Amdahl's Law. *Commun. ACM 31*, 5 (May 1988), 532–533.
5. Gustafson, J.L., and Montry, G.R., and Brenner, R.E. Development of parallel methods for a 1024-processor hypercube. *SIAM Sci. Stat. Comp. 9*, (July 1988), 609–638.

CR Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)—*parallel processors*; C.4 [**Computer Systems Organization**]: Performance of Systems—*measurement techniques*

General Terms: Measurement, Performance

Additional Key Words and Phrases: Parallel performance

ABOUT THE AUTHORS:

ALAN KARP is a staff member at IBM's Palo Alto Scientific Center. He has worked on problems of radiative transfer in moving stellar matter and in planetary atmospheres, hydrodynamics problems in pulsating stars and in enhanced oil recovery, and numerical methods for parallel processors. He is currently studying the interface between programmers and parallel processors with special attention to debugging parallel algorithms.

HORACE P. FLATT is manager of IBM's Palo Alto Scientific Center. He received a Ph.D. in mathematics from Rice University in 1958, subsequently becoming manager of the applied mathematics group of Atomics International, Inc. He joined IBM in 1961 and has primarily worked in management assignments in applied research in computer systems and applications. Authors' Present Address: IBM Scientific Center, 1530 Page Mill Road, Palo Alto, CA 94304.