ORSA Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Commentary—The "Tar Baby" of Computing: Performance Analysis

John L. Gustafson,

To cite this article:
John L. Gustafson, (1993) Commentary—The "Tar Baby" of Computing: Performance Analysis. ORSA Journal on Computing 5(1):19-21. https://doi.org/10.1287/ijoc.5.1.19

ORSA Journal on Computing
Vol. 5, No. 1, Winter 1993

0899-1499/93/0501-0019 $01.25

COMMENTARY

The "Tar Baby" of Computing: Performance Analysis

JOHN L. GUSTAFSON / *Ames Laboratory,*

Ames, IA 50011-3020;
Email: gus@tantalus.scl.ameslab.gov

It looks so easy. To find the speed of a computer, just take a job and time it. To compare two speeds, or try to establish speed in some type of universal unit, well.... Reporting performance is the "tar baby" of computing, and many authors and researchers have attacked this issue only to find themselves stuck in a mess of shaky logic and shakier assumptions. The more scientific one tries to be, the more difficult the problem is. Barr and Hickman have aimed their efforts at this issue, and have generated some thought-provoking ideas.

1. Polling for Definitions

The article by Barr and Hickman surveys the attempts of the past, and polls a community of 23 researchers to see how they would present performance data from hypothetical experiments. Although it is questionable whether opinion polls can ever settle matters of a scientific nature, it is instructive to see how much confusion exists over definitions, methods, and terminology in performance comparisons. The poll results show almost no consensus about performance reporting, but some of those polled wisely rebelled against distilling complex results down to elementary speedup numbers.

To use an analogy, suppose one asks for a precise measure of the intelligence of a human being. Scores obtained from standardized multiple-choice I.Q. tests are notoriously misleading, variable with test conditions, and poor correlators with the complicated multidimensional notion we call "intelligence." One could take a poll of psychologists to find out which person they consider more intelligent based on various test results, but the results of the poll wouldn't determine anything concrete. It seems doubtful that an opinion survey will ever settle the difficult questions of how to report computer performance.

2. Asking the Right Questions

The hard part about performance analysis is *to ask the right questions*. The obvious question to ask, "How much time does it take to run the application I'm interested in?" succumbs to both vagueness and, well, the possibility of being the wrong question. Some of these issues are discussed by Barr and Hickman, but not all.

Barr and Hickman ask, "What is *time*?" Is it the time perceived by the user, or is it merely the time spent in processing? Should "time" include the time spent in processing plus input and output time plus the time to load the program? Should it include the time waiting in line to use the computer? Should it include the time to compile or even to *develop* the program, amortized over the number of times the program is run?

Perhaps they should also ask a far more difficult question: "What is *work*?" If computer speed is work divided by time, we need a grasp of both the numerator and the denominator. Is the application carved in stone or does it vary with the computer which runs it? Does one adjust the quality of the answer according to the computing power available? Should one change the algorithm to match the architecture, or make the algorithm identical so that the comparison is a controlled experiment?

Neither work nor time is well-defined on computers. This is part of what makes performance analysis so difficult. The marketing departments of computer companies must deal with these issues every day, as they advertise their computers with SPECmarks, MFLOPS, MIPS, Transactions per Second, or whatever. Computational scientists advertise their results in terms of speedup, efficiency, or some concept of operational complexity divided by some concept of execution time. With poorly-defined rules, both scientists and marketing executives usually succumb to the human tendency to choose data that make their results look good. Consumers of such data eventually find that they have been misled, and come to distrust the whole field of performance evaluation. *We must not stop* trying to enforce objectivity in this field. Too much rides on understanding the performance of computers to just shrug it off as a hopeless endeavor.

A glimpse of the answer appears near the end of Barr and Hickman's article, but unnecessarily restricted to heuristic codes. They write: "Of primary interest in reporting on computational testing of heuristic-based codes are descriptive measures of the tradeoff between time and solution quality. Does the code find a 'good' solution quickly? Given enough time, does the code find a high quality solution?" *This is the crux of performance analysis, no matter what the application.* There is always a tradeoff between time and quality, whether the application is computing the flow over an airplane wing or simulating the global economy. Higher quality means more work. The *quality of the answer* for a given execution time is what we should compare. Looking at answer quality solves the problem of arithmetic precision in scientific benchmarking, where some computers use 32-bit floating point arithmetic or a crude type of 64-bit arithmetic to trade quality for speed. It also solves the issue of which algorithm or program to use in the comparisons. For any performance reporting, find and clearly state the quality measure. *If an application lacks a*

quality measure, then it is unsuitable for use as a performance metric.

3. Some Missing Background

Barr and Hickman miss some classic works in performance evaluation in their references. Credit is due to Hockney and Jesshope[4] for most of the vocabulary of speedup. They gave us the $n_{1/2}$ (vector length at which half the maximum speed is obtained) and R_∞ (asymptotic speed) terms for understanding vector startup and peak speed, and analogous terms for parallel computers in the late 1970s. I believe they are the source of the definition of speedup as serial time divided by parallel time, and of efficiency as speedup divided by the number of processors.

Academic precedent is also owed to Vipin Kumar at the University of Minnesota for the concept of "scaleup." Several years ago, Kumar talked about "isoefficiency," and has written many papers on the amount that a problem must be scaled to obtain a given speedup or efficiency. A recent summary of Kumar's seminal work appears in [5].

The term "embarrassingly parallel" was introduced as a joke by C. Moler at a supercomputing workshop in 1986. He originally intended the term to mean an application such as Mandelbrot fractals, Monte Carlo simulations, scene generation by ray tracing, or any other application in which there need be *no communication* between processors except at the beginning and end of the problem. When an application lets each processor be an island, one should be embarrassed to present the speedup as much of an achievement. Unfortunately, the term has become misused to refer to any application that seems to run fast and scale well on massively parallel systems. The wave equation, fluid dynamics, and structural analysis programs cited use intense interprocessor communication throughout, but were tuned with several techniques that allow their speedups to be over 1,000 on a hypercube with 1,024 processors. It is a "Catch-22" to dismiss successful parallelization as evidence of a trivial application. The term "artificial intelligence" is comparable in the reverse sense: whenever a problem in artificial intelligence is solved (playing a good game of chess, symbolic algebra, etc.), that problem is dismissed as being trivial and not proof of real intelligence. If we define artificial intelligence implicitly as anything humans can do that computers can't, we will never achieve "real" artificial intelligence. If we brush aside efficient parallelism as the sign of a trivial application, we will never achieve efficient parallelism on "real" applications.

It was probably with the goal of "absolute speedup" that Gordon Bell announced his challenge to obtain the best parallel speedup on real applications. He introduced his award by electronic mail in late 1986. Bell was interested in the absolute performance of parallel computers compared to conventional vector mainframes, but the rules indicated a goal of relative speedup. In response to his challenge, the work done at Sandia on the 1,024-node nCUBE[2] gave a table of 2-D data that contains all the cases discussed by Barr and Hickman: absolute speedup, relative speedup, scaled speedup, and fixed time speedup. Instead of presenting some distillation of the information, [2] presented all of the information and some suggested interpretations. Although [2] is usually cited to show that parallel processing can be very successful for real applications, it also dealt effectively with most of the issues of reporting parallel performance discussed by Barr and Hickman, including algorithmic changes.

Superlinear speedup is a phenomenon that gets rediscovered every few years. An excellent survey of the sources of superlinear speedup is given by D. Helmbold and C. McDowell, in [3]. Many causes exist beyond the ones pointed out by Barr and Hickman.

4. The Fallacy of Efficiency

The existence of the superlinear speedup phenomenon shows the fallacy in the classical definition of "efficiency" as speedup divided by the number of processors. With superlinear speedup, efficiency becomes greater than unity! Why does efficiency, so defined, *matter*? Here, the underlying assumptions are that processors are 100% of the cost of a computer, and that linear speedup is the ultimate goal. It is not the least bit difficult to imagine situations where a system running at a parallel efficiency of 30% is actually faster (in absolute terms) and less expensive than a system running at 60% efficiency.

A common fallacy in performance reporting, unfortunately not discussed by Barr and Hickman, is that conventional serial processors use their hardware with perfect efficiency. Modern serial computers have many functional units, caches, input and output processors, and memory structures that are idle during most clock cycles. On a parallel computer, the idleness is far more obvious than on a serial computer; yet, a parallel computer often has more of its circuitry at work than the serial computer running a serial algorithm. Why is communication not considered *work*? Doesn't a serial computer communicate between its components? Why is communication considered *overhead* while operations are considered *real work*? Both are essential to getting the job done.

Barr and Hickman do not mention one of the problems with conventional "speedup" (reduction in execution time): It *penalizes faster absolute speed*. Suppose one uses an Intel iPSC/1 hypercube to solve a problem on $1, 2, 4, \ldots, 128$ processors, and measures the speedup. That parallel processor, introduced in the mid-1980s, used i80286 processors with i80287 arithmetic coprocessors, and typically got about 40 KFLOPS per processor. The more recent Intel Gamma Touchstone, also with up to 128 processors, uses i860 processors that typically run at 5 to 8 MFLOPS—over 100 times faster. But the speedup one sees will be much, much less! The serial component is exaggerated, and communication costs also stick out when the computing speed is increased. This type of experiment was done by Barton and Withers, in [1].

Before anyone shouts, "That's Amdahl's law!" let me point out that the penalty for higher speed shows the fallacy of Amdahl's concept of speedup: Amdahl (and many other researchers) assume the task size is constant as the computing power increases, resulting in exaggeration of task overhead. If the problem being tested is scaled up

so it takes about the same amount of time on both systems, then the distortion effects almost vanish. If conventional speedup and efficiency measures guide us to lower absolute speed, then they should either be modified or discarded altogether. Credit for pointing out this fundamental unfairness in speedup as a goal is due to X.-H. Sun, in [6].

Although the Barr and Hickman article mentions many facets of parallel performance reporting, there are still many of a basic nature that are not addressed. The preceding examples illustrate just a few.

5. Comparing Different Algorithms

Barr and Hickman present the question of which algorithm to compare when different machines yield optimal speed with different algorithms. The question can be resolved as follows: Put all the algorithms into a single program (at least conceptually). Precede the main program with a self-test that shows which architecture the program is running on, including the number of processors. Then have the program pick the fastest strategy, based on experience or theoretical understanding of why certain methods are fastest. If one calls this composite program THE ALGORITHM, then one recovers the right to say that the program runs well on both architectures, while preserving the right to make performance comparisons.

6. Parallel Reporting Guidelines

Perhaps the best contribution made by Barr and Hickman is their "Parallel Reporting Guidelines" to allow better exchange of information among researchers. Their advice is good, scientific sense, but we all need to be reminded of the scientific method now and then: Report everything, both successes and failures. Explain *everything* that might have affected the results. Use statistical analysis when dealing with stochastic quantities, and sensitivity analysis when dealing with sensitive quantities. The only recommendation I question is "C. Perform final, reported testing on a dedicated or lightly loaded system." Why? Wouldn't it be better to test the computer in its normally loaded state, and state that load in the performance report? It would be misleading to measure a large IBM mainframe as a single-user, single-job computer when it is rarely used that way. In a way, lightly loading a system contradicts "A. Focus on the real time and cost required to solve difficult problems." A well-considered experimental design must make every effort to measure the computers in the manner in which they are actually used.

7. Final Comment

The article by Barr and Hickman performs a much-needed service: it illustrates the difficulty of reporting computational experiments, and makes us all aware of how much disagreement there is over the issues and even over the quantities being measured. The current disagreement is a symptom of a field in its adolescence: growing, changing, healthy, but completely unsettled. I hope their "Guidelines" are posted prominently on the wall next to everyone who publishes parallel computing results.

References

1. M. BARTON and G. WITHERS, 1989. Computing Performance as a Function of the Speed, Quantity, and Cost of the Processors, *Proceedings of Supercomputing '89*, 759–764.
2. J. GUSTAFSON, G. MONTRY and R. BENNER, 1988. Development of Parallel Methods for a 1024-Processor Hypercube, *SIAM Journal on Scientific and Statistical Computing 9:4*, 609–638.
3. D. HELMBOLD and C. MCDOWELL, 1989. Modeling Speedup Greater than *n*, *1989 International Conference on Parallel Processing Proceedings 3, 219–225.*
4. R. HOCKNEY and C. JESSHOPE, 1981. *Parallel Computing*, Adam Hilger, Philadelphia.
5. V. KUMAR and A. GUPTA, 1991. Analyzing the Scalability of Parallel Algorithms and Architectures: A Survey, *Proceedings of the 1991 International Conference on Supercomputing*, 1–10.
6. X.-H. SUN and J. GUSTAFSON, 1991. Toward a Better Parallel Performance Metric, *Parallel Computing 17*, 1093–1109.