The Association for Computing Machinery
1515 Broadway
New York, N.Y. 10036

ACM ISBN: 0-89791-678-6

Additional copies may be ordered prepaid from:

ACM Order Department
P.O. Box 12114
Church Street Station
New York, N.Y. 10257

Phone: 1-800-342-6626
(U.S.A. and Canada)
1-212-626-0500
(All other countries)
Fax: 1-212-944-1318
E-mail: acmpubs@acm.org

ACM Order Number:208940

Printed in the U.S.A.

The project for Suggesting Computer Science Agenda(s) for
High-Performance Computing is sponsored by the University
of Maryland Institute for Advanced Computer Studies (UMIACS),
and the National Science Foundation Center in Discrete Mathematics
and Theoretical Computer Science (DIMACS).

Developing a Computer Science Agenda for High-Performance Computing

Editor: Uzi Vishkin
University of Maryland

The Hidden Cost of Low Bandwidth Communication

Guy E. Blelloch Bruce M. Maggs Gary L. Miller

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Every year manufacturers of massively parallel computers release new machines with ever more impressive benchmark performance. Unfortunately, these machines are not widely used in practice. To date, they have been used efficiently mostly to solve problems whose communication structure is well understood. Furthermore, the software for solving these problems has been highly engineered, and both data and machine specific. Progress has been made recently on solving more irregular problems and machine-independent programming languages such as HPF have made software more portable, but the machines remain difficult to program. Our position is that much of this difficulty stems from a lack of appreciation of the impact of low performance interconnect on software development.

To understand why the cost of low performance interconnect is underestimated, it helps to look at the standards by which parallel computers have been measured until now. Five years ago, a typical measure of the power of a massively parallel computer was the peak rate at which it could perform floating-point operations, i.e., how many gigaflops (floating point operations per second) it could perform. Manufacturers produced machines that achieved very high floating point performance, but this measure was found to be nonpredictive because the machines could rarely achieve it. More recently, the performance of these machines has been judged against a set of standard benchmarks, which include the LINPACK and NAS [1] benchmarks. As shown in Figure 1, these benchmarks, especially the NAS, have exposed a large difference between peak and achievable performance. One of the conclusions that can be drawn from these benchmarks is that machines with high communication bandwidth perform well across the board, whereas peak floating-point performance is relevant only on embarrassingly parallel problems.

The benchmarks, however, still do not reveal the full cost of inadequate comunications bandwidth. Absent from the performance statistics is the cost of software development. In particular the statistics fail to capture

1. the time to write the code,
2. the time to write the compiler that translates the code,
3. the time to port the code to different distributions of data, and
4. the time to port to different machines.

These hidden costs can be substantial. First, although the NAS benchmarks are relatively simple, manufacturers of parallel machines typically spend multiple man years of software development in order to draw high performance from their machines. Second, the time reported for the development of code for specific benchmarks is often an underestimate of the true time, since benchmark-specific optimizations are often hidden in the compiler. Third, although the benchmarks are meant to measure the performance of parallel computers on basic tasks, the code for specific benchmarks is often highly tuned to the particular distribution of data specified by that benchmark. For example, most of the manufacturers' programs for solving the NAS integer sort benchmark exploit the fact that the low order bits of the keys are randomly distributed. One exception is the CRAY

Developing a CS Agenda for High-Performance Computing
© 1994 ACM 0-89791-678-6..$3.50

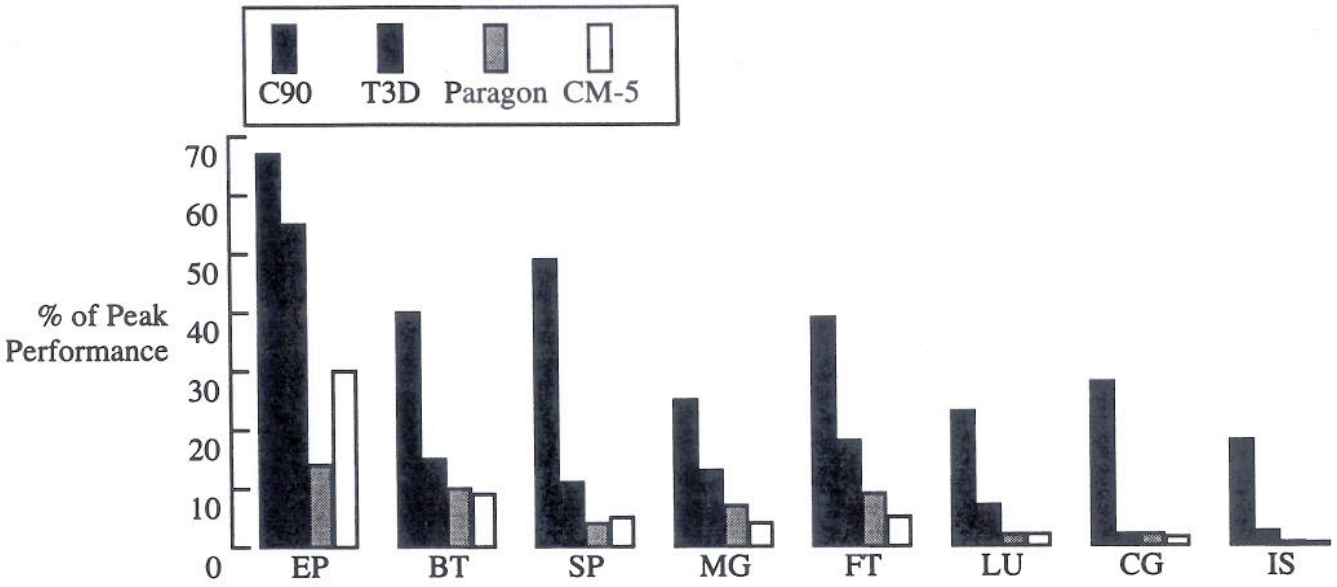

Figure 1: Percent of peak floating-point performance for the 8 NAS benchmarks on 4 parallel machines. The benchmarks names are Embarrassingly Parallel (EP), Multigrid (MG), Conjugate Gradient (CG), 3-D FFT PDE (FT), Integer Sort (IS), LU Simulated CFD Application (LU), SP Simulated CFD Application (SP), and BT Simulated CFD Application (BT). The benchmarks are ordered from the one that requires the least communication on the left to the one that requires the most on the right. The machines are ordered from the one that has the highest communication bandwidth on the left to the one that has the lowest bandwidth on the right. The numbers were calculated using the equation benchmark-flops/(time×peak-machine-flops). The benchmark-flops are the number of floating-point operations required by the serial version. In the case of the integer sort benchmark (IS), benchmark-flops is the number of integer operations. The times are taken from the official published numbers as of November 1993 and are all for the largest configuration of each machine for which numbers are reported (128 processors for the CM-5, Paragon and SP1, and 16 processors for the C90).

Y-MP C90, which is also one of the highest bandwidth machines. In fact, in order to be competitive on this benchmark, one of the companies replaced its general-purpose sorting routine with a special-purpose routine. As a consequence, these codes cannot be used as efficient general-purpose sorting routines. On the conjugate gradient benchmark, whose main inner loop is a sparse-matrix vector product, many of the manufacturers use an algorithm that works well for the specific data set but does not work for many other classes of sparse matrices. Finally, the algorithms used to achieve top performance vary substantially from machine to machine.

There are a variety of reasons for the high cost of developing benchmark software for a new generation of computers. For example, programmers of these machines are faced with using new programming languages and immature compilers and debuggers, and they have to learn to think about solving problems in parallel. However, the primary complication faced by these programmers is locality. In order to achieve acceptable benchmark performance on most of these machines, the programmer and the compiler writer must take into account a large difference in cost between accessing local memory and accessing remote memory. This complication not only increases the time that is spent developing benchmark codes, but also leads to programs that are both machine and data specific.

Two examples of ways in which local memory accesses differ in cost from remote memory accesses are in the time required for the access to be performed, and the frequency with which they can be performed (assuming that it is possible to pipeline multiple memory accesses). Following the LogP model [2], the time to perform a memory access is called the *latency*. Roughly speaking, the latency measures the time for a memory access request to leave the chip of the processor making the request, traverse a communication network, visit a memory chip, and return. In most communication networks, latency increases rapidly as the network becomes overloaded with traffic. Thus, in addition to latency, there is a maximum rate at which messages can be injected into a network. The inverse of this rate, which is called the *gap*, is the average time needed between consecutive message transmissions by any one processor in order to ensure that the network does not become overloaded. Another cost included in the LogP model is the *overhead*, which is the time that a processor must spend preparing a message or receiving a message while performing no other useful work.

Of the three costs, latency, overhead, and gap, latency appears to be the easiest to cope with. There are a variety of automatic techniques for hiding latency with minimal hardware support providing that the amount of excess parallelism, called *slackness* [4], is sufficiently large. These techniques include executing multiple independent threads (with rapid context switching) [3], emulating a collection of virtual processors, or performing vector-accesses (gather and scatter). At presen most parallel computers suffer from significant overhead, but future parallel computers are likely to include special hardware support, such as communication co-processors, to eliminate overhead. The most difficult problem is the gap. Although some algorithmic techniques have been developed to overcome the gap, i.e., minimize total communication, they tend to be problem and data specific. Thus, the gap problem leads to increased software development costs.

Furthermore, the gap is widely viewed as reflecting the inherent difference in speed between processors and networks. This belief is based in part on a confusion of gap and latency. Although the speed of light provides an inherent lower bound on the latency of any network, the gap, which measures bandwidth, is not limited by the speed of light. What actually limits bandwidth is the amount of money that a manufacturer is willing to spend on the construction of a communication network, which can be varied, for example, when deciding on the number of network pins per processor, the number of router nodes per processor, the number of engineers allocated to network design, and whether any modifications should be made to the processor chip. As Figure 2 shows the gaps in commerical parallel computers vary tremendously from machine to machine.

If software costs were weighted more heavily in deciding how much to spend on a communication network, manufacturers would likely choose to spend more. Unfortunately, the current quoted measures for the machines (either peak performance or benchmark results) do not adequately include these software costs. As a consequence, it is unlikely that a company can afford to take them

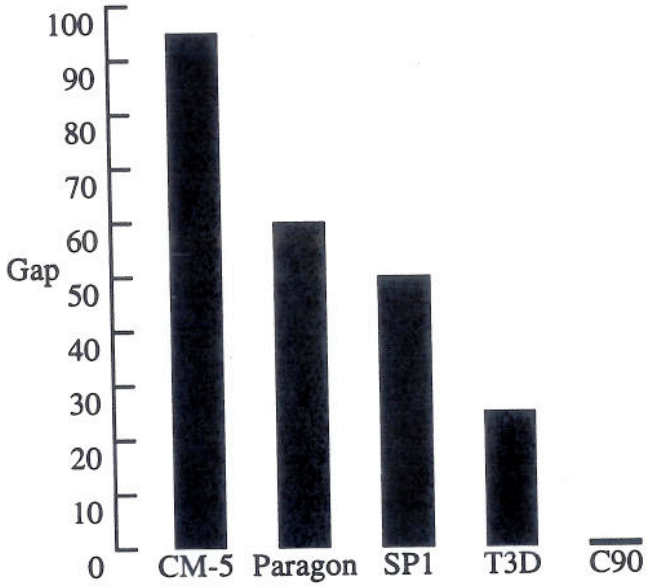

Figure 2: The gap for various parallel machines. This is a rough measure of the ratio of computation to communication power. It is measured as the peak floating-point performance (double-precision) of the full machine divided by the bisection bandwidth (in terms of 64-bit words). The numbers are for the largest configurations for which data was available and is approximate.

into account. In addition, the true magnitude of these software costs are nearly always underestimated because the measures tend to be qualitative.

We suggest a new standard for the manufacturers of parallel computers: the teraflop-terabyte standard. The goal is to produce a computer with a network capable of delivering messages across any bisection at a rate of one terabyte per second on the average, while at the same time performing floating-point operations at a peak rate of one teraflop. To illustrate this benchmark, consider a massively parallel computer composed of 4000 processors, each capable of 250 Mflops. To achieve a terabyte, each processor must be capable of sending and receiving data at the rate of 250 Mbytes per second. A machine that achieves the teraflop-terabyte standard must by definition have a reasonably small gap. In our 4000-processor example, the gap is roughly 8, because each floating-point operation involves 8-byte words. We believe that if the gap of a parallel machine is this small, then the programmer will be no more concerned with locality than is the programmer of a sequential machine, and that minimizing the locality concern will have a dramatic effect on software development costs.

References

[1] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon. NAS parallel benchmark results 10-93. Technical Report RNR-94-006, NASA Ames Research Center, March 1994.

[2] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1993.

[3] B. J. Smith. Architecture and applications of the HEP multiprocessor computer system. In *Real-Time Signal Processing IV*, pages 241–248, August 1981.

[4] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.