6 CHALLENGES OF FUTURE HIGH-END COMPUTING

David H. Bailey

NERSC,
Lawrence Berkeley Laboratory, U.S.A.

dhb@nersc.gov

Abstract: With the recent achievement of one teraflop/s (10^{12}flop/s) on the ASCI Red system at Sandia National Laboratory in the U.S., many have asked what lies ahead for high-end scientific computing. The next major milestone is one petaflop/s (10^{15} flop/s). Systems capable of this level of performance may be available by 2010, assuming key technologies continue to advance at currently projected rates.

This paper gives an overview of some of the challenges that need to be addressed to achieve this goal. One key issue is the question of whether or not the algorithms and applications anticipated for these systems possess the enormous levels of concurrency that will be required, and whether or not they possess the requisite data locality. In any event, new algorithms and implementation techniques may be required to effectively utilize these future systems. New approaches may also be required to program applications, analyze performance and manage systems of this scale.

Keywords: petaflops, teraflops, high-end computers, performance.

6.1 INTRODUCTION

In December 1996, a sustained rate of one teraflop/s (also written as 10^{12} floating-point operations per second) was achieved on the Linpack benchmark, using the ASCI Red system at Sandia National Laboratory in New Mexico, U.S.A. While no one has yet demonstrated a sustained performance rate exceeding one teraflop/s on a real scientific or engineering application, it is expected that this also will be achieved soon.

The next major milestone is a performance rate of one petaflop/s (also written as 10^{15} floating-point operations per second). It should be emphasized that we could just as well use the term “peta-ops”, since it appears that large scientific systems will be required to perform intensive integer and logical computation in addition to floating-point operations, and completely non-floating-point applications are likely to be important as well. In addition to prodigiously high computational performance, such systems must of necessity feature very large main memories, between ten terabytes (10^{13} bytes) and one petabyte (10^{15} bytes) depending on application, as well as commensurate I/O bandwidth and huge mass storage facilities. The current consensus of scientists who have performed initial studies in this field is that petaflops systems may be feasible by the year 2010, assuming that certain key technologies continue to progress at current rates (Sterling and Foster, 1996).

If one were to construct a petaflops system today, even with mostly low-cost personal computer components (ignoring for a moment the daunting difficulties of communication and software for such a system), it would cost some 50 billion U.S. dollars and would consume some 1,000 megawatts of electric power.

This paper addresses the hardware and software issues in building and effectively using a petaflops computer.

6.2 APPLICATIONS FOR PETAFLOPS SYSTEMS

The need for such enormous computing capability is often questioned, but such doubts can be dismissed by a moment's reflection on the history of computing. It is well known that Thomas J. Watson Jr., a president of IBM, once ventured that there was a worldwide market of only about six computers. Even the legendary Seymour Cray designed his Cray-1 system on the premise that there were only about 100 potential customers. In 1980, after the Cray-1 had already achieved significant success, an internal IBM study concluded that there was only a limited market for supercomputers, and as a result IBM delayed its entry into the market. In stark contrast to these short-sighted projections, some private homes now have Watson's predicted six systems, and the computational power of each of these personal computers is on a par with, if not greater than, that of the Cray-1. Given the fact that suburban homes have found a need for Cray-class computing, it is not hard to imagine that the requirements of large-scale scientific computing will continue to increase unabated into the future.

Some of the compelling applications anticipated for petaflops computers include the following (Stevens, 1995):

- Nuclear weapons stewardship.
- Cryptography and digital signal processing.
- Satellite data processing.
- Climate and environmental modeling.
- 3-D protein molecule reconstruction.
- Real-time medical imaging.
- Severe storm forecasting.
- Design of advanced aircraft.
- DNA sequence matching.
- Molecular nano-technology.

- Large-scale economic modeling.
- Intelligent planetary spacecraft.

Further, if the history of computing is any guide, a number of exotic new applications will be enabled by petaflops computing technology. These applications may have no clear antecedent in today's scientific computing, and in fact may be only dimly envisioned at the present time.

6.3 SEMICONDUCTOR TECHNOLOGY PROJECTIONS

Some of the challenges that lie ahead to achieving one petaflop/s performance can be seen by examining projections from the latest edition (1997) of the Semiconductor Technology Roadmap (SIA, 1997), as shown in Table 6.1. It can be seen from the table that Moore's Law, namely the 30-year-old phenomenon of relentless increases in transistors per chip, is expected to continue unabated for at least another ten years. Thus the capacity of memory chips is expected to increase as shown. However, the clock rate of both processors and memory devices is not expected to increase at anywhere near the same rate—ten years from now we can only expect clock rates that are double or triple today's state-of-the-art levels. What this means is continued improvements in performance will almost entirely come from a very aggressive utilization of highly parallel designs. It might be mentioned that some of the RISC processors now on the drawing boards will feature advanced features, such as multiple functional units, speculative execution, *etc*. In most cases these features can be seen as merely alternate forms of parallelism.

6.4 LITTLE'S LAW

Before proceedings further, let us review what is known as Little's Law of queuing theory (Bailey, 1997). It asserts, under very general assumptions, that the average number of waiting customers in a queuing system is equal to the average arrival rate times the average wait time per customer. This principle is a simple consequence of Fubini's theorem (the fact that a 2-D integral can be evaluated as an iterated integral in either order). To see this, let $f(t) =$ the cumulative number of arrived customers, and $g(t) =$ the cumulative number of departed customers. Assume $f(0) = g(0) = 0$,

Table 6.1 Semiconductor Technology Roadmap projections.

Characteristic	1999	2001	2003	2006	2009
Feature size (micron)	0.18	0.15	0.13	0.10	0.07
DRAM size (Mbit)	256	1,024	1,024	4,096	16,384
RISC processor (MHz)	1,200	1,400	1,600	2,000	2,500
Transistors (millions)	21	39	77	203	521
Cost per transistor (micro-cents)	1,735	1,000	580	255	100

and $f(T) = g(T) = N$. Consider the region between $f(t)$ and $g(t)$. By Fubini's theorem, $Q\,T = D\,N$, where Q is the average length of queue, and D is the average delay per customer. In other words, $Q = (N/T)$D. This is Little's Law.

What does this law have to do with high performance computing? Consider a single processor system with memory. Assume that each operation of a computation requires one word from local main memory. Assume also that the pipeline between main memory and the processor is fully utilized. Then, by Little's Law, the number of words in transit between CPU and memory (*i.e.* the length of memory pipe in a vector architecture, size of cache lines in a cache architecture, *etc*.) is equal to the memory latency times bandwidth. This observation generalizes immediately to multiprocessor systems: concurrency is equal to memory latency times bandwidth, where "concurrency" here is aggregate system concurrency, and "bandwidth" is aggregate system memory bandwidth. This form of Little's Law was first noted by Burton Smith of Tera.

6.5 THE CONCURRENCY CHALLENGE

Several designs have been proposed for a petaflops computer system (Sterling and Foster, 1996). For our purposes here we will mention two such designs. One of these is an all-commodity technology system. It consists of 100,000 nodes, each of which contains a 10 Gflop/s processor. We will assume here that these processors can perform four 64-bit floating-point operations per clock period, and that the clock rate is 2.5 GHz. The interconnection network is constructed from commodity network components that are expected to be available in the 2008 time frame. Another is the "hybrid technology multi-threaded" (HTMT) design, which is now being pursued by a team of researchers headed by Thomas Sterling of CalTech. It features 10,000 nodes, each of which contains a 100 Gflop/s processor, running at 100 GHz. These processors employ a novel technology known as "rapid single flux quantum" (RSFQ) logic, which operates in a cryogenic environment. The system includes multiple levels of memory hierarchy, and an interconnection network based on advanced fiber optic technology (Sterling and Foster, 1996).

Now let us try to calculate, for these two system designs, the overall system concurrency. In other words, calculate the the parallelism assumed in the hardware which a programmer must fully utilize to have a chance of achieving somewhere near the theoretical peak performance of the system. Let us assume here that both systems employ DRAM chips with a latency of 100 nanoseconds for read and write operations. We will also assume that the scientific calculation running on each system is very highly tuned, requires almost no inter-processor communication, but requires one word fetched from memory for each floating-point operation performed. These are obviously rather ideal assumptions.

For the commodity design, we will assume that each processor employs a cache memory system capable of fetching four 64-bit words, one for each floating-point pipe, from main memory each clock period, so that the main memory bandwidth is $4 \times 8 \times 2.5 \times 10^9$ bytes/s $=$ 40 gigabyte/s. Observe that at any given time, 250 words are in the pipeline between main memory per pipe (since the latency to main memory is 250 clock periods). Thus the overall system concurrency is 100,000 (nodes) $\times$ 4 (pipes/node) $\times$ 250 (words in memory pipe) = 10^8.

For the HTMT design, with the ultra-fast clock (100 GHz), the latency to DRAM main memory is now 10,000 clock periods. Thus we will assume that the multi-threaded processor design is capable of maintaining 10,000 outstanding memory fetches simultaneously, so as to enable the processor to sustain somewhere near peak performance. Then a similar calculation as above gives the concurrency as 10,000 (nodes) $\times$ 10,000 (words in memory pipe/node) = 10^8.

Note that these two concurrency figures are the same. In fact, either figure could have been calculated without any consideration of the details of the system design, simply by applying Little's Law. If we assume, as above, that one 64-bit word of data is required per flops performed, then the aggregate main memory bandwidth, measured in words/s, is equal to the performance in flops/s. Thus for a one petaflop/s system, Little's Law says that the system concurrency is merely latency (100 ns = 10^{-7} seconds) $\times$ bandwidth (10^{15} word/s), which is 10^8.

Needless to say, a system concurrency of 10^8 presents a daunting challenge for future high-performance computing. What this means is that virtually every step of a scientific calculation must possess and exhibit 10^8-way parallelism, or else performance is certain to be poor. Consider, for example, a calculation performed on the commodity system mentioned above, where 90% of the operations performed in this calculation can efficiently utilize 100,000 nodes (*i.e.* exhibits concurrency exceeding 10^8), but that a residual 10% can efficiently only utilize 1,000 nodes (*i.e.* exhibits concurrency not greater than 10^6). Then by Amdahl's Law, the overall performance of the application is limited to $10^{15}\ /\ (0.9/105 + 0.1/103)\ =\ 9.2 \times 10^{12}$ flop/s, which is less than one percent of the system's presumed peak performance.

As emphasized above, these analyses were done using highly idealized assumptions. But for computations that require more main memory traffic, or which involve a significant amount of inter-processor communication, the concurrency requirement will be even higher. It is thus clear that a key research question for the field of high performance computing is whether the applications anticipated for future high-end systems can be formulated to exhibit these enormous levels of concurrency. If not, then alternative algorithms and implementation techniques need to be explored.

6.6 THE LATENCY CHALLENGE

It is clear from the discussion so far that latency will be a key issue in future high performance computing. Indeed, even today latency is important in achieving high performance. Consider the latency data in Table 6.2. Note that the local DRAM latency on the SGI Origin is 62 clock periods. On some other RISC systems today the main memory latency exceeds 100 clock periods, and this figure is widely expected to increase further in the years ahead. But these latency figures pale when compared to some of the other latencies in the table.

Recall that in the discussion above on Little's Law, it was assumed that most operations accessed data in main memory. But if algorithms and implementation techniques can be formulated so that most operations are performed on data at a lower level of the memory hierarchy, namely cache or even CPU registers (where latency is lower),

Table 6.2 Latency data.

System	Latency	
	Seconds	Clock Ticks
SGI Origin, local DRAM	320 ns	62
SGI Origin, remote DRAM	1 μs	200
IBM SP2, remote node	40 μs	3,000
HTMT system, local DRAM	50 ns	5,000
HTMT system, remote memory	200 ns	20,000
SGI cluster, remote memory	3 ms	300,000

then there is some hope that Little's Law can be "beaten"—in other words, the level of concurrency that an application must exhibit to achieve high performance can be reduced.

These considerations emphasize the importance of performing more studies to quantify the inherent data locality of various algorithms and applications. Data locality needs to be quantified at several levels, both at low levels, corresponding to registers and cache, and at higher levels, corresponding to inter-processor communication. In general, one can ask if there are "hierarchical" variants of known algorithms, *i.e.* algorithms whose data locality patterns are well matched to those of future high-performance computer systems.

A closely related concept is that of "latency tolerant" algorithms; algorithms whose data access characteristics are relatively insensitive to latency, and thus are appropriate for systems where extra-low latency is not available. One reason this is of interest is the emerging fiber-optic data communication technology, which promises greatly increased bandwidth in future computer systems. This technology raises the intriguing question of whether on future systems it will be preferable to substitute latency tolerant, but bandwidth intensive algorithms, for existing schemes that typically do not require high long-distance bandwidth, but do rely on low latency.

For example, some readers may be familiar with what is often called the "four-step" FFT algorithm for computing the 1-D discrete Fourier transform (DFT). It may be stated as followed. Let n be the size of the data vector, and assume that n can be factored as $n = n_1 \times n_2$. Then consider the input n-long (complex) data vector to be a matrix of size $n_1 \times n_2$, where the first dimension varies most rapidly as in the Fortran language. Then perform these steps:

1. Perform n_1 FFTs of size n_2 on each row of the matrix.

2. Multiply the entry in location (j, k) in the resulting matrix by the constant $e^{(-2\pi ijk)/n}$ (this assumes the indices j and k start with zero).

3. Transpose the resulting matrix.

4. Perform n_2 FFTs of size n_1 on each row of the resulting matrix.

The resulting matrix of size $n_2 \times n_1$, when considered as a single vector of length n, is then the required 1-D DFT (Bailey, 1990). In an implementation on a distributed memory system, for example, it may additionally be necessary to transpose the array before performing these four steps, and, depending on the application, at the end as well. But it is interesting to note that:

- the computational steps are each performed without communication, *i.e.* "embarrassingly parallel";
- the transpose step(s), which are "complete exchange" operations on a distributed memory system, or "block exchange" operations on a cache system, require large global bandwidth, but can be done exclusively with fairly large, contiguous block transfers, which are latency tolerant; and
- the 1-D FFTs required in steps 1 and 4 themselves may be performed, in a recursive, hierarchical fashion, using this algorithm, to as many levels as appropriate.

In short, this algorithm is latency tolerant and hierarchical, so that it is expected to perform well on a variety of future high performance architectures.

A key characteristic of this algorithm is that it relies on array transposes for communication. This same approach, using array transpose operations instead of the more common block decomposition, nearest-neighbor communication schemes, may be effective on future systems for a variety of other applications as well, including many 3-D physics simulations.

6.7 NUMERICAL SCALABILITY

An issue that is related to, but independent from, the issue of parallel scalability is numerical scalability. An algorithm or application is said to be numerically scalable if the number of operations required does not increase too rapidly as the problem is scaled up in size.

It is an unfortunate fact that most of the solvers used in today's large 3-D physical simulations are not numerically scalable. Let n be the linear dimension of the grid in one dimension, so that the other dimensions are linearly proportional to n. Then as n increased, the number of grid points increases as n^3. But for most of the linear solvers used in today's computations, the number of iterations required must also be increased, linearly or even quadratically with n, because the increased grid size increases the condition number of the underlying linear system. Thus the operation count increases as n^4, or even faster in some cases.

This phenomenon is often assumed as an unavoidable fact of life in computational science, but this is not true. Research currently being conducted on domain decomposition and multi-grid methods indicate that many 3-D physical simulations can be performed using techniques that break the link between the number of iterations and the condition number of the resulting linear systems. Thus these methods may yield new computational approaches that dramatically reduce the computational cost of future large-scale physical simulations (Barth, 1997).

6.8 SYSTEM PERFORMANCE MODELING

Developing new algorithms for future computer systems naturally raises the question of how meaningful research can be done years before such systems are available. This suggests a need for effective tools and techniques to model the operation of a future system, thus permitting one to have confidence in the efficiency of an algorithm on such a system.

There are several approaches to perform studies of this sort. One approach is to employ a full-scale computer system emulation program. However, emulators of this scope typically run several orders of magnitude slower than the real systems they emulate. Nonetheless, such simulations may be necessary to obtain high levels of confidence in the performance of an algorithm on a particular design. Along this line, it is interesting to note that accurate simulations of future parallel systems are likely to require today's state-of-the-art highly parallel systems to obtain results in reasonable run times.

A more modest, but still rather effective, approach is to carefully instrument an existing implementation of an algorithm or application (such as by inserting counters), so as to completely account for all run time in a succinct formula. One example of this approach is (Yarrow and der Wijngaart, 1997). These researchers found that for the LU benchmark from the NAS Parallel Benchmark, the total run time T per iteration is given by:

$$T \ = \ 485FN^3 2^{-2K} + 320BN^2 2^{-K} + 4L +$$

$$(1 + \frac{2(2^K - 1)}{(N-2)})(2(N-2)(279FN^2 2^{-2K} + 80BN^{2-K} + L)) + 953F(N-2)N^2 2^{-2K})$$

where L = node-to-node latency (assumed not to degrade with large K), B = node-to-node bandwidth (assumed not to degrade with large K), F = floating point rate, N = grid size, and P = 2^{2K} = number of processors.

This formula permitted the authors to determine that the Class C size of the LU benchmark should run with reasonably good efficiently on systems with up to 16,384 processors, assuming the computer system's latency and bandwidth scales appropriately with system size.

6.9 HARDWARE ISSUES

So far we have focused on algorithm and application issues of future high-end systems. Of course there are other issues, which we can only briefly mention here. Among the hardware issues are the following:

1. Should high-end systems employ commodity device technology or advanced technology?

2. How can the huge projected power consumption and heat dissipation requirements of future systems be reduced?

3. Should the processors employ a conventional RISC design, or a more sophisticated multi-threaded design?

4. Should the system be of a distributed memory, or a distributed shared memory architecture?
5. How many levels of memory hierarchy are appropriate?
6. How will cache coherence be handled?
7. What design will best manage latency and hierarchical memories?
8. How much main memory should be included?

With regards to item 2, one advantage of the HTMT design is that the RSFQ processors feature a remarkably low electric power requirement. The HTMT system researchers now estimate that the entire processor complex for a one petaflop/s system would consume only 100 watts. The required super-cooling environment will obviously increase this figure, but it still would be orders of magnitude less than that of conventional RISC processors.

With regards to item 8, some researchers have assumed the “3/4”rule, namely that for typical 3-D physical simulations, memory increases as n^3, while computation increases as n^4. Thus the memory required for future high-end system will scale as the 3/4 power of the performance. Since systems today typically feature roughly one byte of memory per flop/s of computing power, an application of the 3/4 rule implies that petaflops systems will require only about 30 terabytes of main memory.

However, as mentioned above, advances in domain decomposition and multi-grid methods may overturn the rationale underlying the 3/4 rule. If so, then more memory would be required, possibly as much as one petabyte for a one petaflop/s system (on the other hand, larger problems could be performed). Also, an emerging class of “data intensive” problems appear to require, in many instances, larger memories than projected by the 3/4 rule. Finally, real petaflops systems, when they are available, will doubtless be used at least in part for development and multi-user batch processing, especially during daytime hours, even if during they are used off-hours for large, grand-challenge jobs. In an environment where multiple, more modest-sized jobs are being run, a memory-to-flop/s ratio more typical of day’s high-end computing will be required. In any event, it is clear that more scaling studies are needed to settle this important issue.

6.10 THE LANGUAGE ISSUE

One important question that needs further study is the issue of programming language. Today, many programmers of high-end system utilize message-passing language systems, such as MPI or PVM, since they tend to obtain the best run-time performance. But message passing languages are difficult to learn, use and debug. They are not a natural model for any notable body of applications, and they appear to be somewhat inappropriate for distributed shared memory (DSM) systems. Further, their subroutine-call format, which unavoidably involves a software layer, looms as an impediment to performance for future computer systems.

Other programmers have utilized array-parallel languages such as HPF and HPC. These language systems are generally easier to use, but their run-time performance lags significantly behind that of similar codes written using MPI. Further, the single-instruction, multiple data (SIMD) model assumed in array-parallel languages is inappropriate for a significant body of emerging applications, which feature numerous asynchronous tasks. The languages Java, SISAL, Linda and others each has its advocates, but none has yet proved its superiority for a large class of scientific applications on advanced architectures.

What will programmers of future systems need? One can dream of the following features:

1. Designed from the beginning with a highly parallel, hierarchical memory system in mind.
2. Includes high-level features for application scientists.
3. Includes low-level features for performance programmers.
4. Handles both data and task parallelism, and both synchronous and asynchronous tasks.
5. Scalable for systems with up to 1,000,000 processors.
6. Appropriate for parallel clusters of distributed shared memory nodes.
7. Permits both automatic and explicit data communication.
8. Permits the memory hierarchy to be explicitly controlled by performance programmers.

It is not known whether such features can be realized in a usable language for high-performance computing. Clearly more research is needed here.

6.11 SOFTWARE ISSUES

There are of course many software issues, including:

1. How can one manage tens or hundreds of thousands of processors, running possibly thousands of separate user jobs?
2. How can hardware and software faults be detected and rectified?
3. How can run-time performance phenomena be monitored?
4. How should the mass storage system be organized?
5. How can real-time visualization be supported?

It is clear that meeting these challenges will require significant advances in software technology. With regards to item 3, for example, the line plots often used today to graphically display the operation of say 16 processors, showing communication between them as time advances, will be utterly inappropriate for a system with 100,000 processors. Completely new approaches for representing performance information, and for automatically processing such information, will likely be required.

6.12 CONCLUSION

For many years the field of high performance computing has sustained itself on a fundamental faith in highly parallel technology, with patience that the flaws in the current crop of systems will be rectified in the next generation, all the while depending on the boundless charity of national government funding agencies. But the result of this informal approach is not hard to observe: numerous high performance computer firms have gone out of business, others have merged, and funding from several federal agencies (at least in the U.S.) has been cut.

What might help, in this author's view, is for researchers in the field to do more rigorous, quantitative studies, replacing the often informal studies that have been made to date. For example, vendors often ask scientists at government laboratories questions such as how much bandwidth is needed at various levels of the memory hierarchy, and how much latency can be tolerated by our applications. On such occasions the author has had little to offer, other than some vague comparisons with some existing systems. Clearly such studies would be both of near-term and long-term value to the field.

Along this line, perhaps the time has come to attempt some quantitative studies in arenas other than raw performance. For example, can the productivity of programmers using various programming systems be quantified? What about the usefulness of various performance tools – can this be quantified and measured? At the least, studies of this sort could help establish the need for continuing work in the field. Let the analyses begin!

References

Bailey, D. (1990). FFTs in external or hierarchical memory. *Journal of Supercomputing*, 4(1):23–35.

Bailey, D. (1997). Little's law and high performance computing. Available from the author.

Barth, T. (1997). A brief accounting of some well known results in domain decomposition. Available from the author.

SIA (1997). The national technology roadmap for semiconductors. Semiconductor Industry Association.

Sterling, T. and Foster, I. (1996). Petaflops systems workshops. Technical Report CACR-133, California Institute of Technology. See also: Sterling, T., Messina, P. and Smith, P. (1995), *Enabling Technology for Petaflops Computers*, MIT Press, Cambridge, MA.

Stevens, R. (1995). The 1995 petaflops summer study workshop. This report and some related material are available from http://www.mcs.anl.gov/petaflops.

Yarrow, M. and der Wijngaart, R. V. (1997). Communication improvement for the LU NAS parallel benchmark: A model for efficient parallel relaxation schemes. Technical report, NAS Systems Division, NASA Ames Research Center.