HPC computers: Status and outlook

J.J. Dongarra
University of Tennessee
and
Oak Ridge National Laboratory
and
University of Manchester
dongarra@eecs.utk.edu

1 Historical Overview

Looking back on the last four decades, high performance computing (HPC) has been characterized by rapid change in vendors, architectures, technologies, algorithms, software, and system usage. Despite all these changes, performance has evolved steadily, where performance is measured by the number of flops per second, a flop being an elementary floating-point operation (addition, subtraction, multiplication, or division). Often cited in this context is Moore's Law, which states that the number of transistors on integrated circuits doubles approximately every two years. Figure 1 plots the peak performance of various computers of the last six decades, all supercomputers of their time, and demonstrates how well Moore's law holds for performance for nearly the entire lifespan of modern computing.

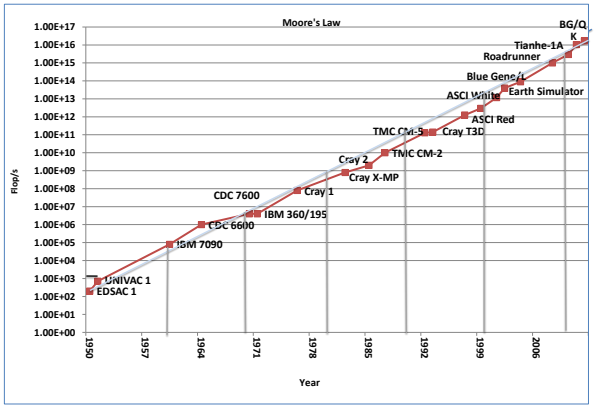

Figure 1: Peak performance of the fastest computer systems for the last six decades.

The initial success in the 1970s of vector computers, which could carry out operations on whole vectors at at time, was driven by raw performance. The introduction of this type of computer system started the modern supercomputing era. In the 1980s the availability of standard development environments and application software packages became more important. Next to performance, these criteria determined the success of multiprocessor vector systems, especially with industrial customers.

Massively parallel processing (MPP) computers, which share the work among a large number of processors, became successful in the early 1990s due to their better price/performance ratios, enabled by increased performance of "off the shelf" microprocessors. In the lower and medium market segments, the MPPs were replaced by microprocessor-based symmetric multiprocessing (SMP) systems (systems in which identical processors share the same memory) in the middle of the 1990s. The success of microprocessor-based SMPs, even for the very high-end systems, was the basis for the emergence of cluster concepts in the early 2000s. During the first half of the decade clusters of PCs and workstations became the prevalent architecture for many application areas. However, the Japanese Earth Simulator vector system (2002) demonstrated that many scientific applications could benefit greatly from a different computer architecture and created renewed interest within the scientific HPC community in new architectures and new programming paradigms.

The IBM Roadrunner system at Los Alamos National Laboratory, which employs a hybrid design built from commodity parts, broke the petaflops (10^{15} floating-point operations per second) threshold in June 2008. The next major target is exascale computing (10^{18} floating-point operations per second), a thousandfold increase over petascale, which is not expected to be achieved before 2018.

2 Challenges

Science priorities lead to scientific models, and models are implemented in the form of algorithms. Algorithm selection is based on various criteria, such as accuracy, verification, convergence, performance, parallelism, and scalability. Models and associated algorithms are not selected in isolation but must be evaluated in the context of the existing computer hardware environment. Algorithms that perform well on one type of computer hardware may become obsolete on newer hardware, so selections must be made carefully and may change over time. Moving forward to exascale will put heavier demands on algorithms in at least two areas: the need for increasing amounts of data locality in order to perform computations efficiently, and the need to obtain much higher factors of fine-grained parallelism as high-end systems support increasing numbers of compute threads. As a consequence, parallel algorithms must adapt to this environment, and new algorithms and implementations must be developed to exploit the computational capabilities of the new hardware. The transition from current sub-petascale and petascale computing to exascale computing will be

at least as disruptive as the transition from vector to parallel computing in the 1990's.

We now describe some of the particular challenges ahead in the use of high performance computers.

2.1 New Algorithms for Multicore Architectures

Multicore processors, in which a single chip contains two or more independent processing units called cores, are now ubiquitous on the desktop through to HPC systems. Scalable multicore systems bring a growing cost of communication relative to computation. Within a node (a single multicore processor) data transfer between cores is relatively inexpensive, but across nodes the cost of data transfer is becoming very large. This trend is addressed by new approaches such as communication-avoiding algorithms (see Section 2.4), algorithms that support simultaneous computation and communication, and algorithms that vectorize well and have a large volume of functional parallelism.

2.2 Adaptive Response to Load Imbalance

Adaptive multiscale algorithms are an important part of many applications because they apply computational power precisely where it is needed. However, they introduce dynamically changing computation that results in load imbalances from a static distribution of tasks. As we move towards systems with billions of processors, even naturally load-balanced algorithms on homogeneous hardware will present many of the same daunting problems with adaptive load balancing that are observed in today's adaptive codes. For example, software-based recovery mechanisms for fault-tolerance or energy-management will create substantial load-imbalances as tasks are delayed by rollback to a previous state or correction of detected errors. Scheduling based on a directed acyclic graphs (DAGs) also requires new approaches to optimize resource utilization without compromising spatial locality. These challenges require development and deployment of sophisticated software approaches to rebalance computation dynamically in response to changing workloads and conditions of the operating environment.

2.3 Multiple Precision Algorithms and Software

One instance of the increasingly adaptive nature of libraries is the capability to recognize and exploit the presence of mixed precision arithmetic. Motivation comes from the fact that, on modern architectures, 32-bit (single precision) floating-point operations can execute at least twice as fast as 64-bit (double precision) operations. The performance of algorithms for solving linear systems or computing eigenvalues or singular values can be significantly enhanced by applying a given method in single precision then using a few steps of iterative refinement in double precision to elevate the accuracy of the result from single

to double precision. This technique can be applied not only to conventional processors but also to other technologies such as graphics processing units (GPUs), and so can more effectively utilize heterogeneous hardware. The use of mixed precision exploits not only the greater speed of single precision arithmetic but also the reduce storage and memory traffic of single versus double precision arrays.

2.4 Communication Avoiding Algorithms

Algorithmic complexity is usually expressed in terms of the number of operations performed rather than the quantity of data movement within memory. However, in modern systems memory movement is increasingly expensive compared with the cost of computation. It is therefore necessary to develop algorithms that reduce communication to a minimum while not unduly increasing the amount of computation. A general approach is to derive bandwidth and latency lower bounds for various dense and sparse linear algebra algorithms on parallel and sequential machines, e.g., by extending the well-known lower bounds for the usual $O(n^3)$ matrix multiplication algorithm, and then to seek new algorithms that (nearly) attain these lower bounds. The study of communication-avoiding algorithms is in its infancy, but it is already leading to new algorithmic ideas and approaches.

2.5 Auto-tuning

Numerical libraries need to have the ability to adapt to the possibly heterogeneous environment in which they have to operate in order to achieve good performance, energy efficiency, load balancing, and so on. The objective is to provide a consistent library interface that remains the same for users independent of scale and processor heterogeneity, but which achieves good performance and efficiency by binding to different underlying code, depending on the configuration. In addition, the auto-tuning has to be extended to frameworks that go beyond library limitations, and are able to optimize data layout (such as blocking strategies for sparse matrix kernels), stencil auto-tuners (since stencil kernels, which update array elements according to a fixed pattern, are diverse and not amenable to library calls) and even tuning of the optimization strategy for multigrid solvers (optimizing the transition between the multigrid coarsening cycle and course grid solver to minimize run time). Adding heuristic search techniques and combining them with traditional compiler techniques will enhance the ability to address generic problems.

2.6 Fault Tolerance and Robustness for Large-Scale Systems

Modern PCs may run for weeks without rebooting and most data servers are expected to run for years. However, because of their scale and complexity, today's supercomputers run for only a few days before a reboot is needed. The major

challenge in fault tolerance is that faults in extreme scale systems, with their millions of processors, will be continuous rather than exceptional events. This requires a major shift from today's software infrastructure. On today's supercomputers every failure kills the application running on the affected resources. These applications have to be restarted from the beginning or from their last checkpoint. The checkpoint/restart technique will not scale to highly parallel systems because a new fault will occur before the application can be restarted, causing the application to become stuck in a state of constant restarts. New fault tolerant paradigms need to be developed and integrated into both the system software and user applications.

2.7 Building Energy Efficiency into Algorithm Foundations

Energy consumption is becoming a major issue in HPC, with energy costs for the some of the largest machines already exceeding a million dollars per year. Power and energy consumption must now be added to the traditional goals of algorithm design, namely correctness and performance. The emerging metric of merit is performance per watt. Energy reduction depends on software as well as hardware., so it is essential to build power and energy awareness, control and efficiency into the foundations of numerical libraries.

2.8 Sensitivity Analysis

As the high fidelity solution of models becomes possible, the next challenge is to study the sensitivity of the model to parameter variability and uncertainty and to seek an optimal solution over a range of parameter values. The most basic form, the forward method for either local or global sensitivity analysis, simultaneously runs many instances of the model or its linearization, leading to an embarrassingly parallel execution model. Such high-throughput computing tasks are well suited to using spare cycles on pools of PCs, for example running at night or weekends.

2.9 Numerical Pitfalls

Problems that warrant the use of the fastest computers are necessarily among the largest problems ever to be solved, according to any appropriate measure of problem dimension. Various mathematical or numerical difficulties can potentially arise as dimensions grow ever larger, including slower convergence of an iterative method that has performed well for smaller problems, computed results having lower accuracy due to an increased number of rounding errors, and overflow of intermediate results. A good example of what can go wrong concerns the use of random number generators to construct linear systems $Ax = b$ to be solved by Gaussian elimination with partial pivoting for benchmarking purposes. The obvious approach is to fill the columns of the matrix A, one by one, with the output from a pseudorandom number generator. A few years

ago, after a computation of this form lasting 20 hours, the computed result was found to be incorrect. The cause was eventually identified as a singular matrix A: the number of matrix elements exceeded the period of the random number generator, with the result that columns repeated and the matrix was singular. By itself, singularity should not affect the computation, since rounding errors usually ensure that the matrix is numerically nonsingular. However, the presence of exactly repeated columns eventually leads to zero pivots, which cause algorithm failure. The moral of the story is that a code that has worked perfectly up to a certain problem size can fail in subtle ways for larger problems.

One desirable numerical property of extreme-scale computing is bit-wise reproducibility of results for any fixed processor count. But current computing frameworks and libraries do not guarantee reproducibility. This is usually caused by a parallel reduction operation. While the corresponding operation is mathematically associative, associativity may not hold in floating point arithmetic. For example, the natural way to evaluate the sum $a + b + c + d$ is from left to right, but alternatives are $(a + b) + (c + d)$ and $(a + c) + (b + d)$, which are trivial examples of a parallel reduction operation, and these three expressions will usually produce different results in floating point arithmetic. In general, one cannot make assumptions about the order in which reduction operations are carried out in parallel, so the values computed in floating point arithmetic may depend on the number of threads of execution. This makes it much harder to debug programs. At extreme scale it may be possible to construct faster algorithms if the order of evaluation is not pre-specified, for example through the use of dynamic task scheduling. Thus, there may trade-offs between speed and reproducibility. Furthermore, it may be possible to more cheaply ensure a bound on the variability between different runs than to guarantee strict reproducibility, for example by using extra precision in selected parts of an algorithm. Many users may prefer non-reproducible results produced very quickly. along with a bound on the variability.

3 Outlook

The move to extreme-scale computing will require collaboration between hardware architects, systems software experts, designers of programming models, and implementers of the science applications that provide the rationale for these systems. The various issues discussed in this article will need to be considered from a whole system perspective, and the different tools will need to interoperate. As new ideas and approaches are identified and pursued, some will fail. As with past experience, there may be breakthroughs in hardware technologies that result in different micro and macro architectures becoming feasible and desirable, and these will require rethinking of algorithms and system software.

Further Reading

References

[1] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Anal. Appl.*, 32(3):866–901, 2011.

[2] J. Dongarra, P. Beckman, et al. International exascale software project roadmap. *Int. J. High Performance Computing Applications*, 25(1):3–60, 2011.

[3] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. Van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1998.

[4] J. J. Dongarra and J. Langou. The problem with the Linpack benchmark 1.0 matrix generator. *Int. J. High Performance Computing Applications*, 23(1):5–13, 2009.

[5] J. J. Dongarra and A. J. van der Steen. High-performance computing systems: Status and outlook. *Acta Numerica*, 21:379–474, 2012.