

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/282704681>

Computing beyond Moore's law.

Chapter · August 2015

CITATIONS

0

READS

1,454

1 author:



Bruno Carpentieri

Free University of Bozen-Bolzano

218 PUBLICATIONS 1,927 CITATIONS

SEE PROFILE

Computing beyond Moore's law

BY BRUNO CARPENTIERI

In my research as an applied mathematician, I develop numerical linear algebra methods for solving applied computational science problems. In this article, I outline how recent developments in high-performance computing are radically changing the way we do numerics.

All of us, whether we are supercomputing experts, big data analysts, or just use some computing technology occasionally for surfing the web, social networking, listening to music, and storing photos in the cloud, are using the same basic technology that was developed during the second world war for building nuclear weapons. The origins of computing are in the Manhattan project that designed bombs from 1942 to 1946 at the Los Alamos laboratory, directed by Julius Robert Oppenheimer, and the work done in England by Alan Mathison Turing on crypto analysis for breaking German ciphers, enabling the Allies to defeat the Nazis in crucial encounters.

In sixty years of computing we went from the bomb to the cloud, using electronic digital computers built on semiconductor technology and integrated circuits described in 1945 by the mathematician and physicist John von Neumann, with some technological innovation. For an incredibly long time we could get exponential performance growth from von Neumann architectures thanks to Moore's law. This is the reason for this remarkable longevity. In 1975, Gordon Moore, co-founder of the Intel Corporation, predicted a doubling in the number of transistors in a dense integrated circuit approximately every two years. His prediction has proved to be accurate to date. Com-

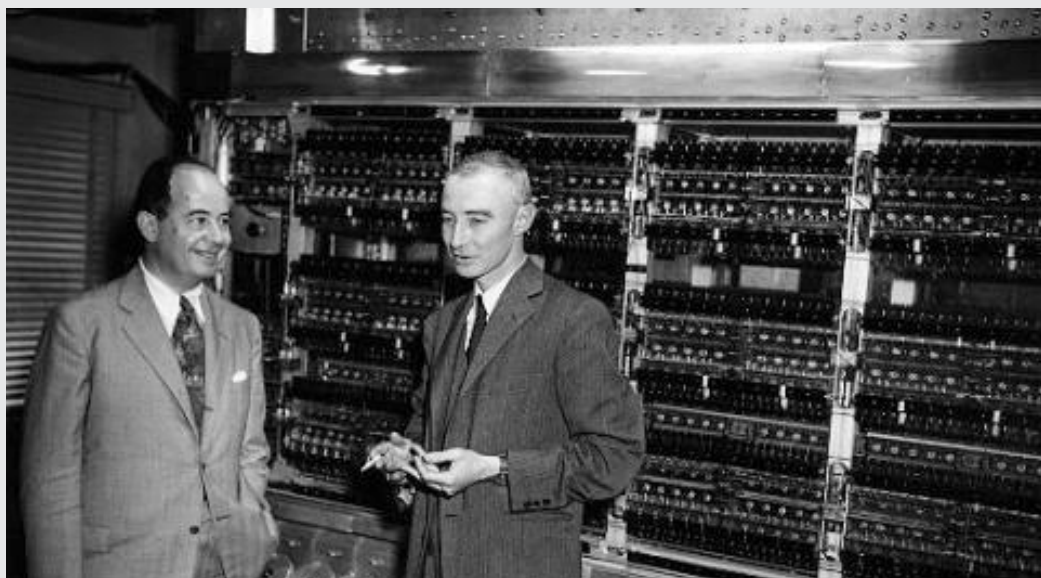


FIGURE 1 Julius Robert Oppenheimer and John von Neumann in front of MANIAC, the first digital computer, at the Institute for Advanced Study, Princeton, 1952.

puters became faster and faster over the years, and we never had to rethink the original von Neumann model.

In 2008, we entered the petascale era. The IBM Roadrunner was the first supercomputer in the world to hit petascale speed, a million billion floating point operations per second. Today petaflops computing is a fully established technology, counting 50 machines in the top 500 supercomputing list <http://www.top500.org/>. Meanwhile, high-performance computing (HPC) has become a thriving business, with the United States and China engaged in a compelling ride to build ever-larger and ever-more-powerful systems, Europe establishing a Partnership for Advanced Computing in Europe (PRACE) to enable high impact scientific discovery and engineering research and development in the field of HPC, and other countries such as Japan and new emerging markets approaching it fast.

Why do we need this level of computing power? The benefits reach virtually every sector of science, from energy to life science, to health, the environment, industrial manufacturing, information technology, and even entertainment. Petascale computing may open up unprecedented opportunities for research in computational science, promising discoveries at scale which can provide tangible benefits for science and society, enhance national security, and leverage economic competitiveness.

“Petascale computing may open up unprecedented opportunities for research in computational science.”

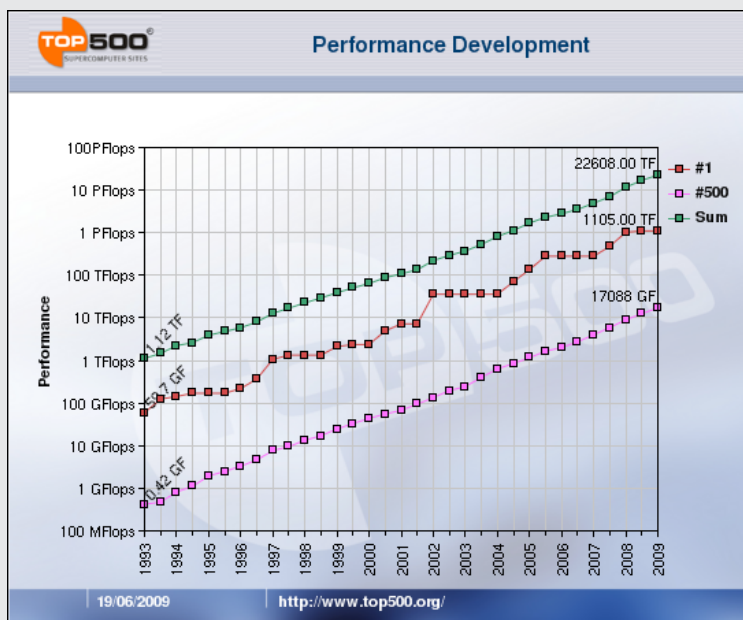
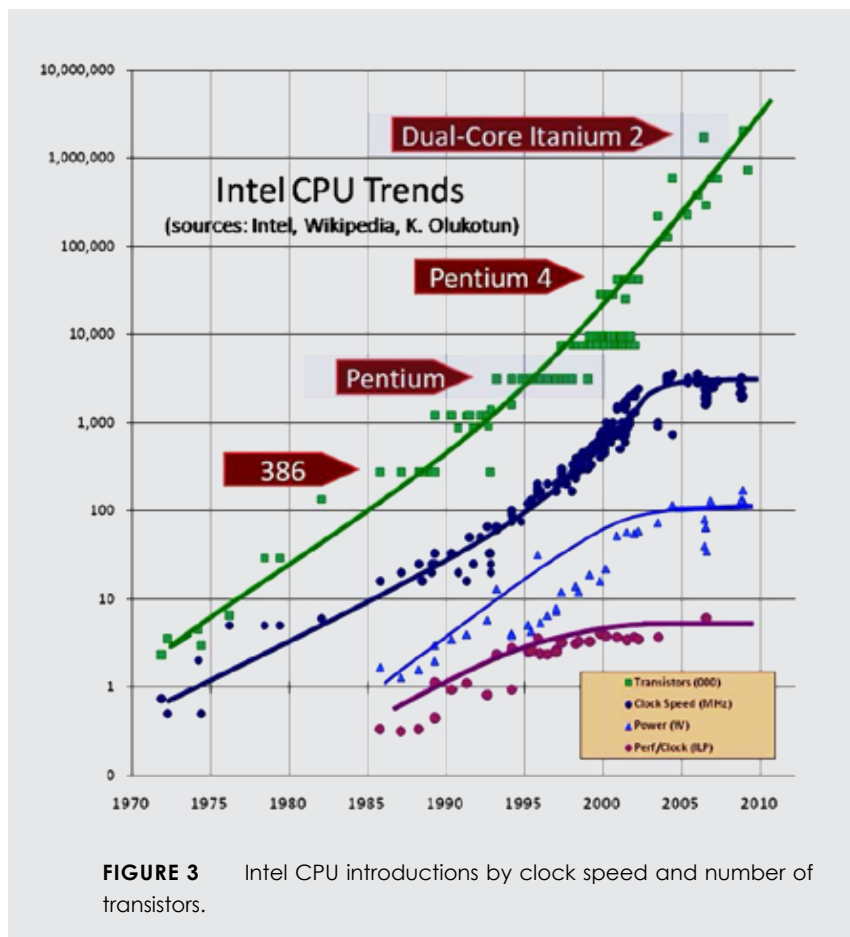


FIGURE 2 Development of supercomputer performance according to the top-500 list <http://www.top500.org/>.

The next big thing in supercomputing is to develop useful exascale-class machines that are hundreds of times faster than the current top-class supercomputers. The cost of building an exascale-class system by, say, 2020 is estimated at about US\$ 200 million for the actual machine, with the cost of operating such a system put at US\$ 400 million per year as this technology would lead to

systems with 100 MW or more power requirements. Due to the problems of power consumption and the stagnation of government funding with a consequent lack of investment and competition, several indicators predict that we will soon hit the end of Moore's law. We may need better technologies and, probably for the first time, to restructure the von Neumann architecture substantially; nowadays, two new technologies in particular, quantum computing and neuromorphic



computing, are given due consideration. Light will not get any faster, and clearly, exponential growth cannot continue forever before we reach the limits of physics [1].

Figure 3 graphs the history of Intel chip (CPU) introductions by clock speed and number of transistors. We see that the number of transistors has continued to rise over the years. However, around the beginning of 2003, a sharp flattening appears in the previous trend toward ever-faster CPU clock speeds. It has become harder and harder to produce faster processors due to too much heat, too high power consumption, and current leakage problems. On Intel chips, we reached 2GHz in August 2001. According to CPU trends before 2003, we should have seen the first 10GHz chips in early 2005. Today, the clock speed on the fastest Intel Core (i7-4790K) is around 4GHz. The clock race is

over, at least for now. And that puts us at a fundamental turning point in software development.

The free performance lunch is over

For around 30 years we have left computer architecture alone, and the same with written software. Computer vendors achieved performance gains by increasing the clock speed (doing the same work faster), optimizing execution flow (doing more work per CPU cycle), and increasing the size of the on-chip cache memory (data closer to the processor is much faster to access than the slow main memory). Speed-ups in any of these areas accelerated both non-parallel and parallel codes almost automatically, by simply recompiling the application on the new

machine to take advantage of the latest generation CPUs. That was a nice technological world to live in. Unfortunately, that world has already disappeared.

The new performance drivers are termed hyperthreading, multicore, and cache. Only one of these, cache, is the same as in the past. Hyperthreading enables us to run more instructions in parallel inside a single CPU and may offer a 5%-15% performance boost, sometimes more. Multicore means running two or more actual processors on one chip. The Intel Xeon Phi coprocessor architecture can have 60+ cores on a single slab of silicon; this means that a single-threaded application can exploit only 1/60 of Intel Xeon Phi's potential throughput. On-die cache sizes will continue to grow in the near future, and all application software can benefit vastly from it because accessing the main memory is typically 10 to 50 times slower than re-

trieving the same data from the cache: cache is speed on modern machines.

The free performance lunch is over. Petascale supercomputing can run hundreds of thousands of simultaneous processes, regardless of processor technology. If we want our application to benefit from the continued performance growth, it will need to exploit a high degree of concurrency. After object-oriented programming, concurrency may be considered the next major revolution in software development. It is not a new concept in computer science; the daunting levels of concurrency required on today's petascale systems is radically new.

Nothing on the horizon indicates that writing performant software for the next generation of parallel machines will get easier. Concurrent programming may have a steep learning curve, systems can be heterogeneous architectures with different parallel programming models that require a significant level of understanding of the hardware to optimize performance. Today's programming languages do not yet offer a higher-level programming model for concurrency. Additionally, not all problems are inherently parallelizable. It takes one woman nine months to produce a baby, but nine women cannot produce one baby in one month. However, if the goal is to produce many children, the human baby problem is an ideally parallelizable problem. We may need to redefine the goals when we parallelize software. The good news is that for many classes of applications the extra effort is worthwhile in terms of performance gains. But what does this matter to mathematics?

Petascale algorithms and applications

Computational science, the scientific investigation of physical processes through modelling and simulation on computers, which may be considered the third pil-

lar of science complementing theory and experimenting, is built upon the problems of continuous mathematics involving real or complex variables. Numerical analysis is the field of mathematics that designs and analyzes algorithms for solving such problems. Much computation in numerical analysis depends on linear algebra problems, such as solving systems of linear equations and least squares problems, computing eigenvalues and singular values of matrices. Take an integral or a partial differential equation, make it discrete, work with vectors and matrices. These problems arise so often in engineering and scientific practice that they occupied the minds of many mathematicians of the last century, from von Neumann to Turing. Without efficient numerical methods, computational science and engineering as practiced today would slow down sharply and maybe quickly halt.

Real and complex numbers cannot be represented exactly on computers. Approximation of quantities (numbers, functions, etc.) and rounding errors are an important part of the business of numerical analysis. To understand Gaussian elimination for solving linear systems, it is necessary to know floating point arithmetic, propagation of rounding errors,

“If the goal is to produce many children, the human baby problem is an ideally parallelizable problem.”

and the concepts of stability, conditioning, and backward error analysis. For example, Gaussian elimination with partial pivoting has very intricate stability properties. For some special $n \times n$ matrices, the algorithm is explosively unstable and useless as it amplifies rounding errors by a factor of order 2^n . However, for all practical problems it is backwardly stable; it finds the true solution to a nearby problem. In fifty years of matrix computation, no problems that excite such an explosive instability are known to have arisen under natural circumstances [2]. John von Neumann's paper on the complicated stability properties of Gaussian elimination in finite precision arithmetic is considered the first modern paper in numerical analysis.

Linear systems are somehow exceptional, because most other problems in numerical analysis cannot be solved exactly by a finite algorithm, even working in exact arithmetic. Abel and Galois showed that no finite algorithm can compute eigenvalues of matrices having a size bigger than 4×4 . The same conclusion extends to problems that include a nonlinear term or a derivative, e.g. root finding, differential and integral equations, quadrature, optimization problems. The aim is to find a fast and convergent algorithm. Hence, numerical analysis is concerned with rounding errors and also with all the different kinds of approximations errors such as iteration, truncation, and discretization errors. All of this work is applied successfully every day to solving thousands of challenging real-world problems.

Some years ago the discussion would have stopped at this point. However, the algorithms of numerical analysis are implemented on computers, and the characteristic of the machine plays an important role. The recent development in high-performance computing is radically changing the way we do numerics as applied mathematicians. The gaps between fast and slow algorithms are rapidly growing. Nowadays, there is often evidence that an infinite algorithm is better than a finite algorithm, even if the latter exists for a given problem. Infinite iterative methods for linear systems are more and more appealing to use than finite direct methods as computers become faster, matrices grow larger and very sparse, because we are solving three-dimensional problems, and the cubic computational cost of conventional Gaussian elimination is no longer affordable. The faster the computer, the more important the speed of algorithms. Multigrid methods and Krylov subspace methods with preconditioning are based on the idea of infinite iteration. They are considered two of the most important developments in numerical computation in the last twenty years. The class of Krylov subspace methods in particular is listed as one of the top ten algorithms of the 20th century because of

their great influence on the development and practice of science and engineering in the 20th century. Another famous example arises in linear programming. Linear programming problems are mathematically finite. For decades people solved them by the simplex method, which is a finite algorithm, until in 1984, Karmarkar showed that iterative algorithms are sometimes better.

We will be solving much bigger problems. Methods that do more operations per grid node, cell or element naturally will be preferred to more traditional and established techniques (e.g. higher-order discretization schemes, spectral element methods, discontinuous Galerkin discretization against conventional finite element discretization schemes). Novel classes of highly parallel numerical methods will need to be found. Structured data (e.g. regular Cartesian grids) are already coming back, as they may achieve better load balance than unstructured grids on computers

with hundreds of thousands processors. Only algorithms with linear or almost linear complexity will be of interest, as their computational cost for the solution increases linearly

“Trends indicate that opportunities in this field will continue to grow.”

with respect to the number of unknowns, which is expected to grow very large in the future. The numerical discretization of partial differential equations typically leads to large and sparse linear systems, whose solution may take up to 90-95% of the whole simulation time. Very few sparse matrix solvers scale efficiently on a very large number of processors. The development of new techniques in this area of research, alongside their efficient parallelization, will receive a growing interest in the coming years.

The use of more powerful (but also more complex) computing facilities should help in the search for additional speed, but this will also mean there will be even more factors that need to be considered when attempting to identify the optimal solution approach in the future. This emphasizes the continuing need for close cooperation between experimental physicists,

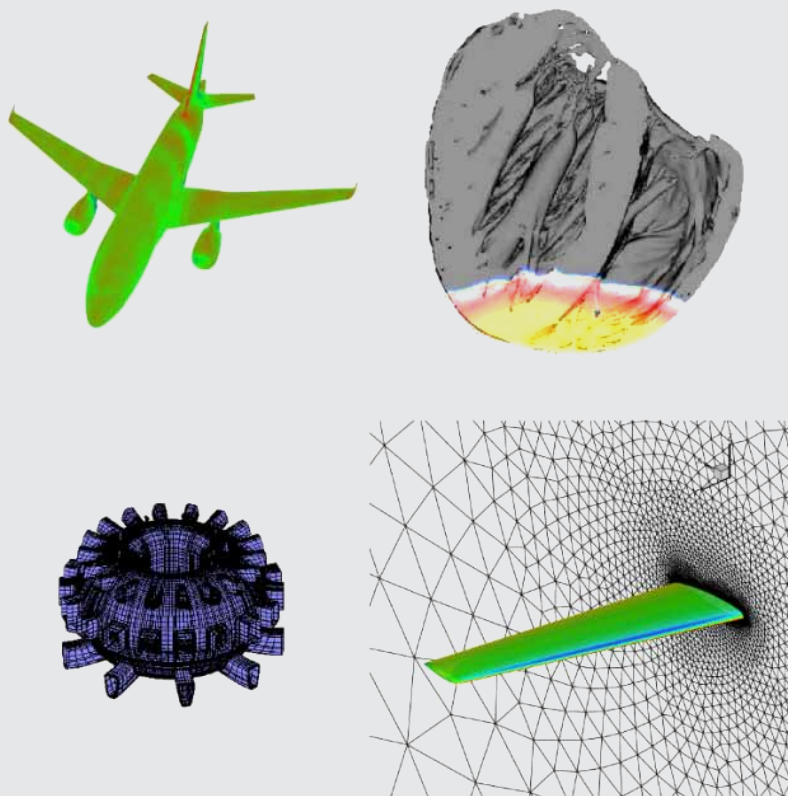


FIGURE 4 Examples of computational science problems from the author's research in electromagnetics, electrophysiology, fusion research, and fluid dynamics applications.

mathematicians, and computer scientists since it is very unlikely that any one person will be able to acquire all of the necessary skills. The rising skills gap in computational science has implications for both scientists, and also for the company that needs these skills, which sometimes by necessity have to hire candidates from outside the specific application domain. Examples come from finance, oil, pharma, and aerospace research, to name but a few [3, 4]. There are jobs for students in computational science and engineering who have a compelling interest in high performance computing and numerical analysis. Trends indicate that opportunities in this field will continue to grow •

References

- [1] H. Simon. Why we need exascale and why we won't get there by 2020. In AASTCS 2: Exascale Radio Astronomy Meeting Monterey, California, 2014.
- [2] L.N. Trefethen and David Bau, III. Numerical Linear Algebra. SIAM Book, Philadelphia, 1997.
- [3] R. Bordas, B. Carpentieri, G. Fotia, F. Maggio, R. Nobes, J. Pitt-Francis, and J. Southern. Simulation of cardiac electrophysiology on next-generation high-performance computers. *Physiol. Trans. A*, 367(1895):1951-1969, 2009.
- [4] B. Carpentieri, I.S. Duff, L. Giraud, and G. Sylvand. Combining fast multipole techniques and an approximate inverse preconditioner for large electromagnetism calculations. *SIAM J. Scientific Computing*, 27(3):774-792, 2005.