

Supercomputing applications to the numerical modeling of industrial and applied mathematics problems

Juan A. Acebrón

Departament d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili
Av. Països Catalans, 26 – 43007 Tarragona, Spain

`juan.acebron@urv.cat`

and

Renato Spigler

Dipartimento di Matematica, Università “Roma Tre”,
Largo S. L. Murialdo, 1 – 00146 Rome, Italy

`spigler@mat.uniroma3.it`

July 7, 2006

Abstract

Present and future *supercomputers* offer many opportunities and advantages to attack complex and demanding industrial and applied mathematical problems, but provide also new challenges. In the Peta-Flops regime, these concern both, the way to exploit the increasingly available power and the need of designing algorithms which are *scalable* and *fault-tolerant* at the same time. An example of a probabilistic domain decomposition method, which is indeed scalable and naturally fault-tolerant, is presented. Grid computing should also be mentioned as an increasingly popular way to perform massively distributed computing: it represents a way to exploit computing power, aside the existing supercomputers. Beyond classical supercomputers there is the prospective *quantum computer*, in view of which it is advisable to start now a search for suitable algorithms for entire classes of problems.

Keywords: supercomputers, supercomputing, parallel computing, Monte Carlo methods, scalability, fault-tolerance

1 Introduction: Supercomputers and supercomputing

Due to the ever more challenging problems put forth by the needs of *scientific computing*, computing power increases continuously, making it possible to cope with more complex applications. These problems include both, purely scientific advances and very practical industrial needs.

By definition, the fastest computers available are called *supercomputers*. Since the first time that the term supercomputer has been used, it refers to those machines whose performances are placed at the frontier of the technology of the moment. Some people call supercomputers the machines that are one generation ahead the “normal” computers in use but one generation behind the needs of solving the challenging problems that they want to solve. Historically, in the 1960s and in the 1970s, only the Cray’s were called supercomputers by everybody, until a number of other companies entered the arena where manufacturing computers more and more powerful was the mission. The Cray’s were essentially *vector* machines, equipped with a small number of powerful processors, however, while supercomputers now include (and tend to be only) those with a large number of processors. Indeed, since many years the trend has been that of designing computer architectures characterized by a very large number of possibly low power processors.

The most powerful machine till early in 2004 was the japanese Earth Simulator, a large-scale “parallel vector” supercomputer, in the line of vector computers similar to the Cray’s, and equipped with 5,120 CPUs, organized in 640 8-ways nodes, and capable of 8 GFlops (GigaFlops) per CPU, hence 41 TFlops (TeraFlops) in total. It was based on groups of few powerful vector systems.

In June 2004, IBM’s Blue Gene/L prototype came into operation, and scored in TOP500 Supercomputer List (which is based on a Linpack benchmark) at the 4th and then 8th position. In September 2004, it overtook the NEC’s Earth Simulator, reaching a speed of 36.1 TFlops (against the 35.83 TFlops of the Earth Simulator). On March 2005, the Blue Gene/L reached 134.5 TFlops, still using only half of the future configuration of 65,536 computer nodes. Another version of the Blue Gene, the Blue Gene/P, planned to be ready in 2006, is expected to break the barrier of 1 PFlops (PetaFlops), and the Blue Gene/Q to reach the speed of 3 PFlops.

Typically, such machines are special-purpose computers, being primarily conceived for some specific applications. For instance, the Earth Simulator was designed for environmental sciences, mostly meteorology, while the Blue

Gene family has been conceived for life science, and more precisely for describing protein folding and for genoma coding.

The point here is that most biological functions involve proteins and the protein's chemical composition is determined by a given sequence of aminoacids. These are linked to each other in a chain, and the protein folds into a complex three-dimensional shape. It is conjectured that the particular shape of a given protein essentially determines its function. It is remarkable that misfolding of certain proteins seems to be responsible for some serious diseases, such as cystic fibrosis, mad cow, Parkinson, and Alzheimer. Arbitrary strings of aminoacids, in general, do not fold into well-defined 3D structures. It is believed that natural evolution has selected out the proteins used in biological processes for their ability to fold in a reproducible way into particular 3D structures, and this occurs in a relatively short time. An understanding of the underlying mechanism can be obtained by appropriate large-scale biomolecular simulations [8].

The Earth Simulator achieved in the first few months the performance of 35.86 TFlops (about 90 % of the peak performance of 40.96 TFlops) in the Linpack benchmark test. Such performances suggested that the Earth Simulator would let science, industry, and human thinking itself to advance. Consider that having a computer capable, for instance, to forecast the weather of tomorrow in two hours is not the same that having this same task accomplished in, say, more than 24 hours! A true advance of science can be expected by the *quantitative* improvements of computers, e.g., in terms of their speed. The Earth Simulator had indeed a strong impact in computer science, demonstrating the feasibility of very large-scale parallel vector architectures, approaching the frontiers of the PFlops computing [16].

The most powerful supercomputer in Europe is, to date, the IBM's MareNostrum at the Barcelona Supercomputing Center (BSC). According to the TOP500 list, in November 2004 it became the fourth most powerful computer in the world, is operative since April 2005, is equipped with 4,800 processors, achieves 27.91 TFlops, and uses Linux operative system.

Correspondingly to the term supercomputer, *supercomputing* denotes the scientific computing than can be performed on such machines. It is well known that advances in hardware technology have always been ahead the desirable progress in software. Therefore, it is clear that designing new powerful algorithms, capable to fully exploit the best available machines, is of paramount importance. Unfortunately, many problems arise from the intimate connection between hardware and software, so that a software designer cannot ignore the computer architecture and other features of the machine

on which he or she intends to run his or her codes.

Supercomputers belong to a variety of types. Their architectures and other features require special care in designing algorithms to efficiently exploit them. Since one way to increase computing power is to increase the number of processors at work, and thus resort to *parallel computing*, the algorithms capable of running efficiently on supercomputers should possess a high degree of parallelism. In general, it is a challenge to transform a truly sequential algorithm into a parallel scheme. On the other hand, it is very rare that a complex algorithm lacks completely of any sequential part. Amdhal's law is still used to provide the reduced efficiency due to the such sequential parts.

However, some problems lead naturally to parallel algorithms. This is the case of the Monte Carlo algorithms, for instance, because they are based on taking averages over a large number of outcomes, each one similar to the others. Each of such results can be obtained on a separate processor, and only at the end taking the averages requires using all results at the same time. It is also clear that Monte Carlo simulations fit well the need of *scalability*, and even the issue of *fault-tolerance*. Indeed, the latter property seems to be rather natural for such algorithms (see §3 below).

Of course, here above we were referring to *classical* computing. Beyond, there is the prospective *quantum computer* (QC), still being developed in the physicists' labs. In 1981, Richard Feynman suggested to truly "emulate" (rather than "simulate", using his words) natural phenomena on a machine. Hence, he proposed to build some device that could imitate the Nature at its most basic level, the quantum mechanical level. In the following years, a number of physicists started thinking seriously how to build such a machine. Quantum computing and quantum information are hot research topics nowadays in Physics, but the subject is highly interdisciplinary, relating Physics to Computer Science, Communication Engineering and Cryptography, possibly Number Theory and Optics.

Apart from the practical realization of a QC – a nontrivial issue that will be clarified in a few or several years to come – it is convenient to start today to investigate which are the possible algorithms that might efficiently run on a QC. Some algorithms can already be devised. Few already famous algorithms, such as those bearing the names of Shor [15], Deutsch and Grover, were found in the recent years. Unfortunately, the existing prototypes of QCs are equipped with only about ten qubits (quantum bits, the quantum mechanical analogue of the classical bit), too few to be practically useful. Progress in increasing such a number, as well as to realize *which* physical solution would be best, is under way. To date, it seems that systems based

on ion traps (rather than, e.g., on NMR, the Nuclear Magnetic Resonance) are the most promising [12].

Why and what for a QC? Steady progress in miniaturizing electronic devices, starting from the centimetric scale of vacuum tubes, has reached in few years the scale of microns. We are already pretty close to the atomic scale, where quantum effects cannot be ignored any longer. A more subtle reason, put forth by M. Rasetti, which will make computer science and quantum mechanics closer, rests on the conjecture that the human brain itself acts as a QC. This means that our brain would work somehow like a computer, but it follows the laws of quantum mechanics. In fact, there is no classical explanation for the fast thinking of a chess champion, capable to beat or at least to compete with a classical computer. The latter must use powerful resources (in terms of speed and memory), while the champion's neurons work at a relatively slow time-scale.

Sometimes it seems that to theoretical computer scientists only algorithms capable of breaking the barrier of NP-complete problems matter. Perhaps, all optimization problems, typically NP-complete, could take advantage from having a QC. Since a QC would allow for an exponential advantage upon classical computers, if appropriately exploited, these scientists might not show a special interest if only a factor would be gained, still remaining within the realm of exponential complexity. In applied and industrial mathematics, however, some benefit could be obtained even in such a case. Therefore, it would be definitely worth finding algorithms that could be implemented on a prospective QC. Given that QCs, as well as most of classical supercomputers in operation today, are special-purpose computers, we should look, rather, for *problems* that could be solved on a QC, i.e., problems whose solution could be implemented on a QC, in a more or less natural way.

It was immediately recognized by a number of researchers that a QC solves naturally problems governed by unitary evolution operators. This is a necessary but not sufficient condition for having an exponential gain over classical machines. In fact, it was proved in [18] that when the amount of "entanglement" scales at most as $\log n$, n being the number of qubits involved in the computation, it is always possible to find an alternative classical algorithm which solves efficiently the same problem. The entanglement is an essential ingredient in every purely quantum computation, and refers to a quantum phenomenon consisting of a long-range correlation of any two or more objects. This is present even when such objects are arbitrarily far. Quantum Mechanics ultimately means, in the Schrödinger formulation, describing natural phenomena through the time evolution of some complex-

valued wave function. This function obeys the Schrödinger equation, and can be represented by the evolution of a unitary operator. Therefore, on the one hand we should find the most efficient way to compute numerically unitary operators. In the context of quantum computation, based on the quantum logic gates paradigm, this step involves designing optimal circuits, made with quantum gates. Optimization of these circuits might possibly be realized resorting to genetic or other evolutionary algorithms. On the other hand, we should look for problems that can be formulated in terms of unitary evolutions. There are a few of such problems: the nonlinear Schrödinger equation, used for instance in nonlinear Optics, in particular the Bose-Einstein Condensate equation, and other nonlinear partial differential equations of Mathematical Physics. An example is the class of integrable systems that can be described in terms of the so-called “Lax pair”, e.g., the Korteweg-de Vries equation, the Toda lattice, the Wigner equation. Also, quantum lattice equations, i.e., cellular automata schemes which mimic in a suitable discrete form the continuum physics equations, as it was done for the gas lattice (gas Boltzmann) equations, can be described in terms of unitary evolutions. In closing this section, we recall that some new mathematical methods to find optimal quantum circuits, interpreting this problem in geometric terms, has been proposed very recently by M.A. Nielsen et al. [13].

Going back to classical supercomputing, one of the most popular ways to exploit parallelism is given by several kinds of *problem decomposition* strategies [10]. We refer to one of the most studied field of applied and industrial mathematics, namely that of problems governed by partial differential equations (PDEs), that have to be solved numerically, see §2 below. Decomposing a given problem for a prescribed PDE may consist in

- *operator decomposition*, that is splitting the differential operators, such as, for instance, implementing an Alternating Direction Implicit (ADI) method;
- *function-space decomposition*, such as Fourier or other spectral methods (in particular, transforming the original problem, formulated in the “time domain” into another in the “frequency domain”, and parallelizing with respect to frequency);
- *domain decomposition*, where the prescribed space domain, where the solution to the given PDE is sought, is divided into a number of subdomains;

- *time-domain decomposition*, similar to the previously described space decomposition, but concerning subdividing the space-time domain with respect to time, see [11, 6], e.g.

All these methods have the purpose to split somehow the given mathematical problem into possibly independent parts, and hence to assign the solution of each part of the entire problem to one processor, which will work independently of all the others. Needless to say that, in general, such a task is not straightforward at all, and only for few problems it is clear how to accomplish it. In general, this issue is nontrivial, and a lot of ingenuity is needed to do the best.

What about scalability? Due to intercommunications among processors, one can hardly hope to attain the ideal speed-up, that is reducing to $1/p$ the computational time required using p processors in parallel, compared to the time employed by a single processor. And there is something worse: In fact, the intercommunication load may increase with p in such a way that a saturation is attained. This means that employing more processors would not increase the overall performance. This has been observed in practice (even though the situation may be not so bad as it seemed initially), using overlapping (Schwarz) type domain decomposition [9].

2 Numerical modeling

The role of mathematical modeling studying scientific and industrial problems nowadays cannot be overestimated. From experimental observations and related experience, we believe that certain sets of equations model rather satisfactorily a bunch of physical (and other) phenomena. Think for instance of the Navier-Stokes or the Euler equations, governing the behavior of ordinary fluids, the Boltzmann and other kinetic equations for rarefied gases or semiconductors, the Maxwell equations for electromagnetism, the Schrödinger equation for quantum mechanics. These are PDEs, sometimes nonlinear, and in addition the problems involving them are formulated over complex-shaped domains. Obtaining a solution in an explicit analytic form is usually out of question. The information hidden in there must be extracted resorting to numerical methods. The original problems are highly nontrivial, and also their numerical solution often represents a formidable task, requiring cutting-edge computational resources. There is no doubt that this approach, that call for computational performances high with respect to those available until recently, will advance entire branches of Science, even qualitatively.

Concerning industrial activities, it is widely recognized that innovation is one of the keys to favor development, and innovation requires flexibility. In turn, flexibility requires abstraction. Mathematics, the queen of logical thinking, is definitely recognized as the human activity most prone to abstraction. Mathematics does not only provide the basics of the rational thinking and methodologies, but also represents an effective and powerful tool for designing, testing, and building models and prototypes of a variety of industrial products and processes. Hence, it is Mathematics the key, the know-how capable of producing welfare in the modern Society.

In the past, modeling meant conducting experiments on physical models and environments. This task may be expensive, time-consuming, not very flexible, and in some case hazardous or just impossible. Think of the case of designing an entire aircraft. The wind gallery was and still is extensively used, but it is impossible to simulate there the conditions of hypersonic flows, it is very expensive to build a large number of physical models of an aircraft, and changing even one parameter may involve spending a lot of money and time. Not to mention medical imaging (which avoids painful and expensive dissections of living humans), simulating failures of a nuclear reactor, analyzing the aftereffects of a nuclear war, etc. Numerical modeling is instead much cheaper, flexible, ... and definitely safer!

Of course, some peculiar problems arise, since numerical modeling requires developing suitable mathematical models of the physical phenomena or industrial processes under investigation, and then designing efficient algorithms to solve the model equations numerically. The latter part cannot disregard the architectures of the supercomputer that will be used, with all related programming issues.

One of the most important and challenging areas is still that of Computational Fluid-Dynamics. Some of the relevant problems here are, e.g., those of naval hydrodynamics (connected with ship design), and of aerodynamics (relevant to aircraft design). From the mathematical standpoint, every problem concerning mechanics of sailing and flying can be reduced to the solution of the Navier-Stokes equations in the domain external to the space part occupied by the ship or the aircraft. Suitable boundary conditions should be imposed on the solid-fluid interface. Only when aircrafts, such as hypersonic airplanes or shuttles, fly in the very rarefied part of the atmosphere, the Boltzmann equation has to be used, rather. Swimming of fishes and microorganisms and flying of birds and insects fit also the same picture above, and again it is matter of solving the Navier-Stokes equations in the region exterior to the organism [4].

It is a remarkable demonstration of the power of numerical simulations,

the key-role played in designing the swiss sailing boat with which the team Alinghi conquered the America Cup in New Zealand in 2003 [14]. Among the other things, the fluid-wall interactions among solid vessel, water, and air, as well as optimization of winglets and other parts of the boat, have been successfully accomplished. While this was merely a sport competition, its being extreme spurred advances in naval engineering. Numerical simulations could replace building physically a number of prototypes and testing them. The savings in terms of money and time should be considered decisive.

In closing this section, let me make a general remark. Developing efficient numerical methods is and will always be an essential part of every successful accomplishment achieved in computing. One might think that brute force computational power would allow to break any barrier. An elementary example is offered by the numerical integration of ordinary differential equations. Why using methods such as Runge-Kutta or other more elaborated methods instead of the simpler Euler method with a smaller time-step? Reducing more and more the time-step size, hence the truncation error, results in increasing the rounding errors, which cumulate, making results useless when adopting too small time-steps. Resorting to more elaborated schemes allows, instead, to use larger time-steps. Therefore, advances in obtaining better algorithms may be even more important than exploiting progress in hardware. It is also remarkable that improvements in computer architectures have been more relevant than technological advances to increase performances.

3 An example of scalable and fault-tolerant method: Monte Carlo methods

Monte Carlo methods essentially consist of generating suitable sequences of “random numbers” on the computer, and evaluating a certain quantity a large number of times, say $N \gg 1$ (N is called the sample size), making use of such sequences, but each time facing a very similar problem. Then, the average over the various outcomes is taken. This approach can be followed to evaluate high-dimensional integrals, but also to solve certain PDEs through probabilistic methods. Usually, convergence when the sample size increases is very slow, the error being of the order of $\mathcal{O}(N^{-1/2})$, and statistical in nature. However, the Monte Carlo approach is trivially parallelizable, since each of the N problems above can be runned independently of the others. Stated in such terms, clearly, the method appears to be scalable as well. Moreover, it is also naturally fault-tolerant. In fact, assuming that

a fraction, n , of the N processors being used fails, meaningful results will be obtained from the remaining $N - n$ processors, and it will suffice just to ignore what could have come from the n processors which have failed. The final error will be slightly worse, of the order of $\mathcal{O}((N - n)^{-1/2})$. Note that

$$(N - n)^{-1/2} \approx N^{-1/2} \left(1 + \frac{1}{2} \frac{n}{N} \right), \quad \text{for } n \ll N,$$

that is $1.005 N^{-1/2}$ for a small fraction $n/N = 1/100$ (1%) of failing processors.

Another way to perform Monte Carlo-type simulations is offered by the possibility of using the so-called sequences of “quasi-random numbers” as opposed to the practical realization of the ideal random numbers, more precisely termed “pseudorandom” numbers. The former are, rather, sequences of deterministic numbers, uniformly distributed. They are however correlated, hence one should make a careful use of them, see [3]. Using quasi-random numbers, the error made in simulations is, upon taking averages, deterministic in nature and of order of $\mathcal{O}(N^{-1} \log^{d^* - 1})$, d^* denoting a certain “effective dimension”. In certain problems it is possible to reduce the geometric (or physical) dimension to the much lower effective dimension. Parallelization with quasi-random numbers is still an open issue, but if this would be possible, the failure of $n \ll N$ processors would imply the slightly larger error

$$\begin{aligned} (N - n)^{-1} \log(N - n) &= N^{-1} \left(1 - \frac{n}{N} \right)^{-1} \left[\log N + \log \left(1 - \frac{n}{N} \right) \right] \\ &\approx (1 + \alpha) N^{-1} [\log N - \alpha]. \end{aligned}$$

Here we assumed $d^* = 2$ and set $\alpha := n/N$. With $\alpha = 0.01$, and considering that N will be at least of the order of the thousands, hence $\log N \gg \alpha$ (e. g., $\log N = 3$ against $\alpha = 0.01$), we obtain

$$(N - n)^{-1} \log(N - n) \approx (1 + \alpha) N^{-1} \log N = 1.01 N^{-1} \log N.$$

In [1, 2], a probabilistic method was designed to solve boundary-value problems for elliptic PDEs, see §4. We stress that, as a rule, the failure of even a single processor would let the machines stop and the job abort, unless a suitable strategy is undertaken. Besides, even if one could ignore the work accomplished by the processors which have failed up to the moment that this happened, meaningless or wrong results would have been produced.

This observation may let appreciate the intrinsic peculiarity of Monte Carlo methods in handling processors failures.

On the other hand, with machines already having thousands or tens of thousands of processors (and soon hundreds of thousands or even millions of processors), working in parallel the probability of failures of a few processors or of the network is very high [7]. It can be estimated that it may occur every few minutes, if not more frequently. Some strategy to cope with such failures, concerning both hardware and software, lead to restart strategies or, more keenly, to design FT-MPI, the Fault-Tolerant version of MPI [5].

4 An algorithm based on Monte Carlo methods

In [1, 2], a probabilistic domain decomposition (PDD) method has been introduced for solving Dirichlet problems for linear elliptic PDEs on a two-dimensional domain. This approach is based on the possibility of representing probabilistically the solutions to such problems, which has the form of a suitable average. The latter requires computing a large number, say N , of paths of an underlying stochastic process, which can be obtained solving certain stochastic differential equations, whose coefficients are related to those of the PDE. The average above is taken over all such paths, evaluated at the first exit (or hitting) time, when the paths started at the point where the solution is being seeked cross the boundary of the domain. Results are poorly accurate, unless an enormous number of realizations of the stochastic process, N , is used (see §3). An appreciable advantage is obtained using quasi-random (instead of pseudorandom) number sequences, since accuracy improves, for a given sample size, or, alternatively, convergence is faster for a prescribed accuracy. Using quasi-random sequences is a delicate task, however, and some special care has to be used, in particular when they are used to numerically solve stochastic differential equations [3] as is needed here.

In [1, 2], only very few values of the solution were generated via Monte Carlo methods, and then used as pivotal nodes to obtain continuous approximations of interfacial values of the solution, see Fig. 1. The boundary-value problems for PDEs being global in nature, it is impossible to compute the solution at even a single point internal to the domain before solving the entire problem, by any deterministic method, while this is possible, somehow, by probabilistic methods.

After that approximate interfacial values have been obtained, the problem can be fully decoupled, and no communications among the processors

(whatever their number might be) affect the algorithm.

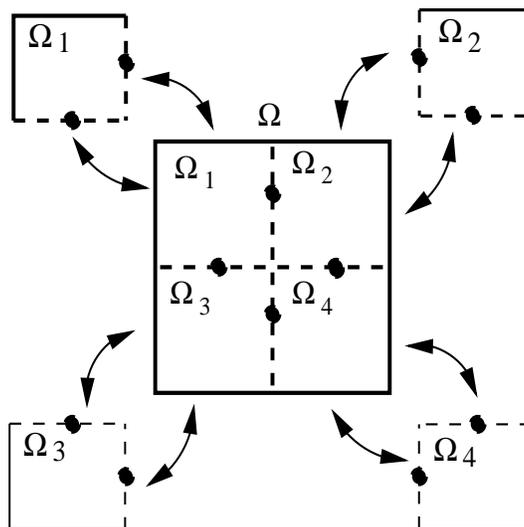


Figure 1: Sketchy diagram illustrating the numerical method, splitting the initial domain Ω into four subdomains, $\Omega_1, \Omega_2, \Omega_3, \Omega_4$.

In Fig. 2 (a), we show the contour-plot solution of the Dirichlet problem given by

$$\begin{aligned} & \frac{y^2 + 1}{2} u_{xx} + \frac{x^2 + 1}{2} u_{yy} + x u_x + y^2 u_y - (x^3 + y^2) u = \\ & P \cos(2x + y) + Q \sin(2x + y) \quad \text{in } \Omega = (0, 1) \times (0, 1), \\ & P = 1 + x(4 + x + 2x^2) + 2xy + x(4 + x)y^2 + y^3, \\ & Q = -\frac{1}{2}[-2 + x^4 + 2x^5 + 2x^3 y + y(5 - 4y + 6y^2) + x^2(1 + y + 6y^2)] \end{aligned} \quad (1)$$

with the boundary data

$$u(x, y)|_{\partial\Omega} = [(x^2 + y) \sin(2x + y)]_{\partial\Omega}. \quad (2)$$

The solution to such problem is $u(x, y) = (x^2 + y) \sin(2x + y)$.

The parameters used in simulations can be found in [1, 2]. A comparison of CPU times is given in Table 1.

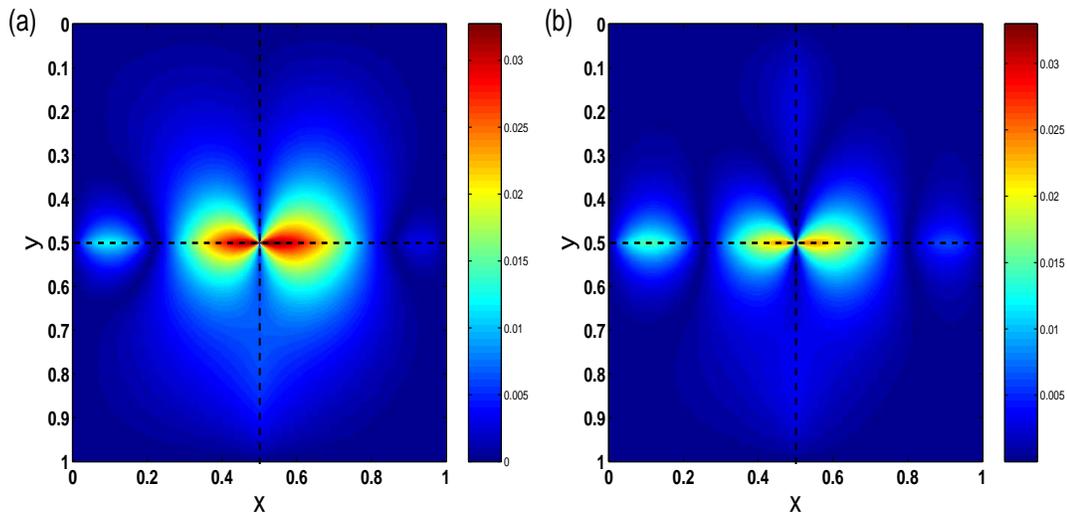


Figure 2: Pointwise numerical error in: (a) the PDD algorithm, and (b) the quasi-PDD algorithm.

Table 1: Overall CPU time in seconds for example D

<i>Processors</i>	<i>PFD</i>	<i>PDD_{Total}</i>	<i>PDD_{Monte Carlo}</i>	<i>PDD_{FD}</i>
4	9200.107	2087.947	3.492	2084.015
9	4098.381	489.684	3.872	485.484
16	2638.937	175.168	3.365	171.508

In Fig. 2 (b), the same is done but using quasi-random number sequences. The improvement is evident. In Table 1, the performance of our algorithm is shown, and a comparison is made with results obtained from “parallel finite differences” (PFD).

Note that, apart from balance considerations, one can use as many processors as domains, and the speed-up has been shown to be of order \sqrt{p} [1]. The ideal speed-up cannot be attained since computing the pivotal values by Monte Carlo is costly, even for a small number of pivotal nodes (even one node implies a nonnegligible cost!). However, the algorithm is scalable as $p \rightarrow \infty$ (no saturation enters the algorithm), and fault tolerant (§3).

Finally, observe that the probabilistically-induced domain decomposition method developed in [1, 2] presents two sources of parallelism. One rests on the intrinsic nature of the Monte Carlo methods, as described in §3, and even a single processor can be used per each realization (there may be thousands

of realizations or more, times the number, usually small, e.g., 4 or 5, of internal pivotal nodes). The other comes from the domain decomposition strategy, by which every subdomain can be assigned to a separate processor.

The fault-tolerant peculiarity of the Monte Carlo part of this method has already been illustrated in §3. Moreover, the domain decomposition part is also well-behaved: Once that full decoupling is attained by the probabilistic method, the solution can be computed by means of whatever (deterministic) numerical method, independently on each subdomain. Hence, a separate processor can be used on each subdomain. In case of failures of one or a few processors being used correspondingly to one or more subdomains, the solution at the points of such subdomains could be fully ignored: This will not affect the solution obtained at the same time in all subdomains where no failures occurred, and, in particular, all computed results are correct.

5 Grid computing

The idea to exploit a broad ensemble of even small computers, such as PCs somehow connected, goes back to 1959, but only in the mid-1990s it became clear that analyzing huge sets of data was indeed a major problem for every single whatever powerful computer. The software designer David Gedye and some collaborators proposed to consider the possibility to intrigue a number of people from all over the world to let them contribute to the search for extraterrestrial intelligence [17, pp. 810-813]. This enterprise requires processing sets of data having sizes of order of Tbytes or even Pbytes. Consequently, the time required by a single though powerful single supercomputer is by far exceeded. Underlying it is needed a suitable software to allow for these wide-range connections. In this way, the project for Search of Extraterrestrial Intelligence (SETI), SETI@home, was born.

The results were astonishing: A computer power of hundreds of thousands and even of few millions of PCs, located all over in the world, has been (and is being) used. Many thousands of small ordinary computers, which would otherwise sit idle, can be linked in huge networks, a grid of computers, the link being provided by the Internet. In April 2005, for instance, the so-called World Community Grid (WCG), established by the IBM International Foundation, signed up 100,000 computers. Indeed, WCG adopted the same technique as SETI@home and Climate-Prediction.net, i.e., to install a screen saver to analyze radio signal data (scanning e.m. radiation) from deep space, or to model climate changes, respectively. WCG is an open grid, and may run five or six projects at the same time [17, p. 773]. Other grids

do exist, for instance to study by the same “screen saver technique” used by SETI, gravitational waves or protein folding, since few years, some signing up more than 3 millions of linked computers, and some hopes to arrive to 10 millions. Myles Allen and David Stainforth were able to establish a SETI-like link among 100,000 people from 150 Countries, resulting into a computer system powerful twice the Earth Simulator!

Perhaps more important, this experiment spurred other people to establish similar networks, and thus Folding@home, Einstein@home, and LHC@home were started. In fact, the current status of many fields of Science today is characterized by being rich of collected data but poor in analyzing them. This is due to the size and the complexity of such collected data. Grid computing as hinted at above provides a *distributed computing* environment capable of processing hundreds of Tbytes or even Pbytes of data, a task that would require hundreds of thousands of years on a single whatever powerful computer. Example of large amounts of data can be found already, for instance, in the NASA’s Earth Observing System (about 1 Pbytes per year), and the CERN Large Hadron Collider, expected to be completed in Geneva by 2007, will involve several Pbytes of data per year out from each of its five detectors.

Some people consider the SETI-like approach and the grid computing as being two different things. Both are definitely examples of distributed computing and both exploit the Internet, but what is properly called “grid computing” calls for a set of standardized interfaces and protocols to allow the researchers to work across the web [17, p. 809]. Grid computing involve many computers to work in some interconnected way, while the SETI-like approach just exploits idle machines.

And, if connecting a sufficiently large number of small computers may represent an alternative to single powerful supercomputer what could we get if we could link in a grid system the 500 top (super)computer in the TOP500 list?

References

- [1] Acebrón, J.A., Busico, M.P., Lanucara, P., and Spigler, R., Domain decomposition solution of elliptic boundary-value problems, *SIAM J. Sci. Comput.* **27**, No. 2 (2005), 440-457.
- [2] Acebrón, J.A., Busico, M.P., Lanucara, P., and Spigler, R., Probabilistically induced domain decomposition methods for elliptic boundary-value problems, *J. Comput. Phys.*, **210**, No. 2 (2005), 421-438.

- [3] Acebrón, J.A. and Spigler, R., Fast simulations of stochastic differential systems, *J. Comput. Phys.* **208** (2005), 106-115.
- [4] Childress, S., Mechanics of Swimming and Flying, Cambridge Studies in Mathematical Biology, 2, Cambridge University Press, Cambridge-New York, 1981.
- [5] Fagg, G., Bukovsky, A., and Dongarra, J., Harness and Fault Tolerant MPI, *Parallel Computing* **27** (2001), 1479-1495.
- [6] Farhat, C., and Chandesris, M., Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid-structure applications, *Int. J. Numer. Meth. Engrg.* **58** (2003), 1397-1434.
- [7] Geist, A. and Engelmann, C., Super-scalable algorithms for computing on 100,000 processors, Lecture Notes in Computer Science, Springer, Vol. 3514, April 2005, pp. 313-321.
- [8] <http://www.research.ibm.com/bluegene/>
- [9] Keyes, D.E., How scalable is domain decomposition in practice?, in: 11th Int. Conf. on Domain Decomposition Methods (London 1998), <http://www.ddm.org>.
- [10] Keyes, D.E., Domain decomposition in the mainstream of computational science, in: 14th Int. Conf. on Domain Decomposition Methods (Morelos, México, 2002), <http://www.ddm.org>.
- [11] Lions, J.L., Maday, Y., and Turinici, G., A parareal in time discretization of PDE's, *C.R. Acad. Sci. Paris* **332** (2001), 661-668.
- [12] Nielsen, M.A., and Chuang, I.L., Quantum Computation and Information, Cambridge University Press, Cambridge, 2000.
- [13] Nielsen, M.A., Dowling, M.R., Gu, M., and Doherty, A.C., Quantum Computation as Geometry, *Science* **311**, 24 February 2006, pp. 1133-1135.
- [14] Parolini, N. and Quarteroni, A., Mathematical models and numerical simulations for the America's Cup, *Comput. Methods Appl. Mech. Engrg.* **194** (2005), no. 9-11, 1001-1026.

- [15] Shor, P.W., Polynomial-time algorithms for prime factorization and discrete logarithms on a Quantum Computer, *SIAM Journal on Computing* **26** (1997), 1484-1509.
- [16] Shimasaki, M and Zima, H.P., The Earth Simulator, Guest Editorial, *Parallel Comput.* **30**, Issue 12, December 2004, pp. 1277-1278.
- [17] Special issue on Distributed Computing, *Science* **308**, 6 May 2005.
- [18] Vidal, G., Efficient classical simulation of slightly entangled quantum computations, *Phys. Rev. Lett.* **91**,147902 (2003).