

A fully scalable algorithm suited for petascale computing and beyond

Juan A. Acebrón · Ángel Rodríguez-Rozas · Renato Spigler

the date of receipt and acceptance should be inserted later

Abstract Nowadays supercomputers have already entered in the petascale computing era and peak rate performance is dramatically increasing year after year. However, most of current algorithms are not capable of exploiting fully such a technology due to the well-known parallel programming related issues, such as synchronization, communication and fault tolerance. The aim of this paper is to present a probabilistic domain decomposition algorithm based on generating suitable *random trees* for solving nonlinear parabolic partial differential equations. These are of paramount importance since many important scientific and engineering problems are modeled by such type of differential equations. We stress that such algorithm is perfectly suited for both current and future high performance supercomputers, showing a remarkable performance and arbitrary scalability.

While classical algorithms based on a deterministic *domain decomposition* exhibits strong limitations when increasing the size of the problem and the number of processors involved, probabilistic methods rather allow us to exploit efficiently massively parallel architectures, being the problem fully decoupled. Large-scale simulations runned on a high performance supercomputer confirm such properties.

Keywords Domain decomposition · Monte Carlo methods · Petascale computing · Scalable algorithms · Fault tolerant algorithms

1 Introduction

Designing efficient parallel numerical algorithms for large-scale simulations becomes crucial for solving realistic problems arising from Science and Engineering. Most of them are based on *domain decomposition* (DD) techniques [12], since it has been shown to be specially suited for parallel computers. Nevertheless, classical domain decomposition methods usually suffer from process intercommunication and synchronization, and consequently the scalability of the algorithm turns out to be seriously degraded. Moreover, in order to reduce the effects of the two previous issues the algorithms to be implemented may lead to an unnecessary excess of computation. From the parallelism point of view, probabilistic methods based on Monte Carlo techniques offer a promising alternative to overcome these problems.

The possibility of using parallel computers to solve efficiently certain partial differential equations (PDEs) based on its probabilistic representation was recently explored. In fact, in [1,2], an algorithm for numerically solving linear two-dimensional boundary-value problems for elliptic PDEs, exploiting the probabilistic representation of solutions, were investigated for the first time. This consists in a hybrid algorithm, which combines the classical domain decomposition method [15], and the probabilistic method, and was called “probabilistic domain decomposition method” (PDD for short). The probabilistic method was used merely to obtain only very few values of the solution at some points internal to the domain, and then interpolating on such points,

J. A. Acebrón and A. Rodríguez-Rozas
Center for Mathematics and its Applications, Department of Mathematics, Instituto Superior Técnico, Av. Rovisco Pais 1049-001 Lisboa, Portugal,
E-mail: juan.acebron@ist.utl.pt (Juan A. Acebrón)
E-mail: angel.rodriguez@ist.utl.pt (Ángel Rodríguez-Rozas)

R. Spigler
Dipartimento di Matematica, Università “Roma Tre”, Largo S.L. Murialdo 1, 00146 Rome, Italy,
E-mail: spigler@mat.uniroma3.it (Renato Spigler)

thus obtaining continuous approximations of the sought solution on suitable interfaces. Known the solution at the interfaces, a full decoupling in arbitrarily many independent subdomains can be accomplished, and a classical local solver, arbitrarily chosen, used. This fact represents a definitely more advantageous circumstance, compared to what happens in any other existing deterministic domain decomposition method.

However, in contrast with the linear PDEs, a probabilistic representation for nonlinear problems only exists in very particular cases. In [3], we extended the PDD method to treat nonlinear parabolic one-dimensional problems, while in [4] it was conveniently generalized to deal with arbitrary space dimensions.

In this paper the performance of the PDD algorithm is analyzed and new nonlinear problems are solved, being the scalability properties of the algorithm analyzed. Since the local solver for the independent subproblems can be arbitrarily chosen, a direct method based on LAPACK for solving the ensuing linear algebra problem has been used. Other alternatives based on iterative methods are worth to be investigated, and will be done in a future work.

In the following, we review briefly the mathematical fundamentals underlying the probabilistic representation of the solution of nonlinear parabolic PDEs.

Let $u(\mathbf{x}, t)$ be a bounded function satisfying the Cauchy problem for a linear parabolic partial differential equation,

$$\frac{\partial u}{\partial t} = Lu - c(\mathbf{x}, t)u, \quad u(\mathbf{x}, 0) = f(\mathbf{x}), \quad (1)$$

where $\mathbf{x} \in \mathbb{R}^n$, L is a linear elliptic operator, say $L = a_i(\mathbf{x}, t)\partial_{x_i x_i}/2 + b_i(\mathbf{x}, t)\partial_{x_i}$, with continuous bounded coefficients, $c(\mathbf{x}, t) \geq 0$ and continuous bounded, continuous initial condition, f . The probabilistic representation of the solution u to Eq. (1) through the Feynman-Kac formula, is given by

$$u(\mathbf{x}, t) = E \left[f(\beta(t)) e^{-\int_0^t c(\beta(s), t-s) ds} \right] \quad (2)$$

see [10,11], e.g., where $\beta(\cdot)$ is the stochastic process starting at $(\mathbf{x}, 0)$, associated to the operator, L , and the expected values are taken with respect to the corresponding measure. When L is the Laplace operator, $\beta(\cdot)$ reduces to the standard n -dimensional Brownian motion, and the measure reduces to the Gaussian measure. In general, the stochastic process $\beta(\cdot)$ is the solution of a stochastic differential equation (SDE) of the Ito type, related to the elliptic operator in (1), i.e.,

$$d\beta = b(\beta, t) dt + \sigma(\beta, t) d\mathbf{W}(t). \quad (3)$$

Here $\mathbf{W}(t)$ represents the n -dimensional standard Brownian motion (also called Wiener process); see [11,8],

e.g., for generalities, and [13] for related numerical treatments. As is known, the solution to (3) is a stochastic process, $\beta(t, \omega)$, where ω , usually not indicated explicitly in probability theory, denotes the sample paths, which ranges on an underlying abstract probability space. The drift, b , and the diffusion, σ , in (3), are related to the coefficients of the elliptic operator in (1) by $\sigma^2 = a$, with $a > 0$.

Here the expected value is replaced by an arithmetic mean, since in practice we deal with a finite sample size, N . A possible strategy to evaluate the solution requires generating a random exponential time, S , obeying the density probability distribution $P(S) = c \exp(-cS)$ for every random paths. Then, depending on whether $S < t$ or not, the given path $\beta_j(t)$, $j = 1, 2, \dots, N$, contribute or not to the solution. The solution is then computed simply as

$$u(\mathbf{x}, t) = E[f(\beta(t))] \approx \frac{1}{N} \sum_{j=1}^N f(\beta_j(t)). \quad (4)$$

The paper is organized as follows. In Section 2 a probabilistic representation for nonlinear parabolic problems is presented. In Section 3 the PDD algorithm is described, showing their different components, and stressing how they can be parallelized in practice. Here we also discuss the important feature of being a natural fault-tolerant algorithm. In Section 4 a test example of a nonlinear two-dimensional PDE is runned to illustrate the PDD method, and a deep analysis of the computational cost and memory consumption is also undertaken. Finally, some conclusions are given.

2 Probabilistic representation of nonlinear PDEs

Let consider the following initial value problem for a nonlinear two-dimensional parabolic PDE,

$$\begin{aligned} \frac{\partial u}{\partial t} &= Lu - cu + \sum_{i=2}^m \alpha_i u^i, \\ \text{in } \Omega &= [-L, L]^n, \quad t > 0, \quad u(\mathbf{x}, 0) = f(\mathbf{x}), \\ u(\mathbf{x}, t)|_{\partial\Omega} &= g(\mathbf{x}, t), \end{aligned} \quad (5)$$

where $L \in \mathbb{R}$, $\alpha_i \in [0, 1]$, $\sum_{i=2}^m \alpha_i = 1$, being $f(\mathbf{x})$ and $g(\mathbf{x}, t)$, the initial condition and boundary data, respectively.

For the purpose of illustration, let consider the case in which $m = 3$, $\alpha_2 = \alpha_3 = 0.5$. Applying the Duhamel principle [9] to Eq. (5), we obtain

$$\begin{aligned} u(\mathbf{x}, t) &= e^{-ct} \int_{\Omega} d\mathbf{y} f(\mathbf{y}) p(\mathbf{x}, t, \mathbf{y}, 0) \\ &+ \int_0^t \int_{\Omega} ds d\mathbf{y} e^{-cs} \end{aligned}$$

$$\times \frac{1}{2} [u^2(\mathbf{y}, t-s) + u^3(\mathbf{y}, t-s)] p(\mathbf{x}, s, \mathbf{y}, 0), \quad (6)$$

where $p(\mathbf{x}, t, \mathbf{y}, \tau)$ is the associated Green's function, which satisfies the equation

$$\begin{aligned} \frac{\partial p}{\partial t} &= Lp, \quad \mathbf{x} \in \Omega, \quad t > \tau \\ p(\mathbf{x}, \tau, \mathbf{y}, \tau) &= \delta(\mathbf{x} - \mathbf{y}). \end{aligned} \quad (7)$$

The solution can be obtained by means of the Feynman-Kac formula in (2), and yields

$$p(\mathbf{x}, t, \mathbf{y}, \tau) = E[\delta(\beta(t) - \mathbf{y})], \quad (8)$$

where $\beta(t)$ is the solution to the stochastic differential equation in (3), with initial condition $\beta(\tau) = \mathbf{x}$. Eq. (6) yields

$$\begin{aligned} u(\mathbf{x}, t) &= E[f(\beta(t)) \mathbf{1}_{[S > t]}] \\ &+ \frac{1}{c} E[u^2(\beta(t-S), t-S) \mathbf{1}_{[S < t]} \mathbf{1}_{[l_r=0]}] \\ &+ \frac{1}{c} E[u^3(\beta(t-S), t-S) \mathbf{1}_{[S < t]} \mathbf{1}_{[l_r=1]}], \end{aligned} \quad (9)$$

where the time S is a random time, picked up from the exponential density distribution $p(S) = c \exp(-cS)$; $\mathbf{1}_{[cond]}$ is the indicator (or characteristic) function, being 1 or 0 depending whether *cond* is satisfied or not. In this case, l_r is a random variable with values 0 and 1, each one with probability $\frac{1}{2}$. The integral equation in Eq. (6) can be recursively solved, replacing the last term on the right-hand side with the solution $u(\mathbf{x}, t)$, obtaining the following expansion in terms of multiple exponential random times, S_i ,

$$\begin{aligned} u(\mathbf{x}, t) &= E[f(\beta(t)) \mathbf{1}_{[S_0 > t]}] + \frac{1}{c} E[f(\beta(t-S_0)) \\ &\times \mathbf{1}_{[S_1 > t-S_0]} f(\beta(t-S_0)) \mathbf{1}_{[S_2 > t-S_0]} \mathbf{1}_{[S_0 < t]} \mathbf{1}_{[l_r=0]}] \\ &+ \frac{1}{c} E[f(\beta(t-S_0)) \mathbf{1}_{[S_1 > t-S_0]} f(\beta(t-S_0)) \mathbf{1}_{[S_2 > t-S_0]} \\ &\times f(\beta(t-S_0)) \mathbf{1}_{[S_3 > t-S_0]} \mathbf{1}_{[S_0 < t]} \mathbf{1}_{[l_r=1]}] + \dots \end{aligned} \quad (10)$$

Note that in Eq. (10) each term contributes, in part, to the full solution.

Using such a branching stochastic process representation, Eq. (10) can be reformulated as

$$u(\mathbf{x}, t) = E\left[\prod_{i=1}^{k_t(\omega)} f(\mathbf{x}_i(t, \omega))\right], \quad (11)$$

where $\mathbf{x}_i(t, \omega) \in \mathbb{R}^n$ is the position of the i th stochastic process surviving at time t . In case of a boundary value problem, with the boundary data $u(\mathbf{x}, t)|_{\mathbf{x} \in \partial\Omega} = g(\mathbf{x}, t)$, a similar representation holds, i.e.,

$$\begin{aligned} u(\mathbf{x}, t) &= E\left[\prod_{i=1}^{k_t(\omega)} \left\{ f(\mathbf{x}_i(t, \omega)) \mathbf{1}_{[t < \tau_{\partial\Omega_i}]} \right. \right. \\ &\left. \left. + g(\beta_i(\tau_{\partial\Omega_i}), t - \tau_{\partial\Omega_i}) \mathbf{1}_{[t > \tau_{\partial\Omega_i}]} \right\} \right], \end{aligned} \quad (12)$$

where $\tau_{\partial\Omega_i}$ is the first exit time of the stochastic process $\beta_i(\cdot)$.

Note that, in the probabilistic representation of the nonlinear PDEs, the set of all branches of a given ‘‘tree’’ play the role of the single path (or realization) of the stochastic processes used in the linear case, see (2). An average is taken over all trees with a given ‘‘root’’, which will be the space point \mathbf{x} .

For more mathematical details, we refer to the reader to [3] and [4]. In practice, the probabilistic representation in (11) works as follows: A stochastic process initially located in $(\mathbf{x}, 0)$ (root of the tree) is generated and evolves in time. Simultaneously, a random time S_1 , picked up from the exponential probability density $p(S) = c \exp(-cS)$, is generated. If the random time S_1 is less than t , a number of branches corresponding to the power of the nonlinearity are created. This procedure is repeated successively until holds that $\sum_{i=1}^n S_i > t$. Whenever one of these possible branches reaches the final time, t , the initial value function, f , is evaluated at the position where the stochastic process associated to every branch was located. The solution is finally reconstructed multiplying all contributions coming from each branch. In practice, to achieve a reasonable small error in (11, 12), a large number of random trees should be generated. More details on the numerical errors related with the probabilistic representation are found in [3] and [4]. For the purpose of illustration, in Fig. 1 a sketchy picture is shown. This corresponds to the case of a tree with two branches, one of them reaching the final time t , while the other hitting the boundary.

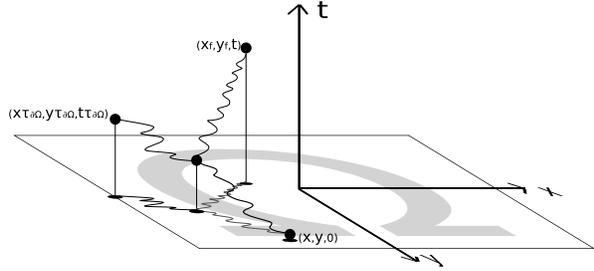


Fig. 1 A picture illustrating a typical random tree in 2D.

3 The PDD method

We now briefly describe the three parts of the algorithm:

Probabilistic part. This is the first step to be carried out, and consists in computing the solution of the PDE at a few suitable points by Monte Carlo, which is

essentially a mesh-free method. In Fig. 2 a sketchy diagram is plotted, illustrating how the algorithm works in practice for a two-dimensional case. Here the solution is obtained probabilistically at a few points pertaining to some “interfaces” conveniently chosen inside the space-time domain $D := \Omega \times [0, T]$, with $\Omega \subset \mathbb{R}^n$. Such interfaces divide the domain into p subdomains, Ω_i , from $i = 1, \dots, p$, being assigned to different processors, $p_i, i = 1, \dots, p$. The following parallelization strategies can be adopted:

- *Parallelizing by splitting in independent sets of points.* Since the number of points where the solution is computed is larger than the number of processors p , computing such a solution can be assigned as a task to different processors. This can be seen as a coarse-grain parallelization and is the more convenient strategy.
- *Parallelizing for each point.* Since the number of random trees turns out to be often much larger than p for a given point, the task to generate only a few of them can be accomplished by different processors. This corresponds to a mid-grain parallelization.
- *Parallelizing for a given realization.* The task to compute only a few branches can be assigned to different processors. This can be seen as a fine-grain parallelization.

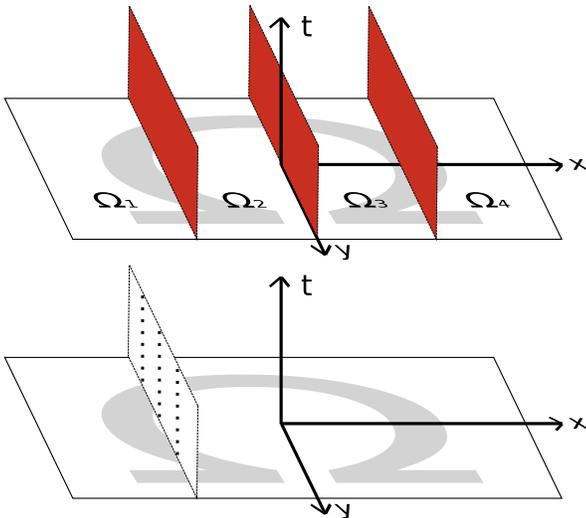


Fig. 2 A sketchy diagram illustrating the main steps of the algorithm in 2D: The figure on the left shows how the domain decomposition is done in practice. The figure on the right shows the points where the solution is computed probabilistically; these are used afterward as nodal points for interpolation.

Interpolation. Once the solution has been computed at few points on each interface, a second step consists in interpolating on such points, being used as nodal points,

thus obtaining continuous approximations of interfacial values of the solution. For that purpose, a tensor product interpolation based on cubic spline [5] was used. The computational cost of the two dimensional case is negligible compared with the time spent in the other parts of the algorithm. The nodal points were uniformly distributed on each plane, and a not-a-knot condition was imposed.

Local solver. The third and final step consists in computing the solution inside each subdomain, this task being assigned to different processors. This can be accomplished resorting to local solvers, which may use classical numerical schemes, such as finite differences or finite elements methods. LAPACK has been chosen for this purpose.

Fault tolerance-related issues

We stress that the algorithm is naturally fault-tolerant, being so in any of the parts it is composed. Here, we describe the peculiarities of such features:

- *Probabilistic part.* If a given processor p_i fails when computing the solution probabilistically at a given point, either any other processor may continue the pending task or, in case of no free processors at that time, this task may be postponed to be executed later.
- *Interpolation.* Similarly as in the previous case, when a processor fails, either the subdomain corresponding to such a processor may be reassigned to a neighbor processor or the task may be postponed to be executed afterward. Note that the failure merely affects to this processor.
- *Local solver.* Let suppose a failure occurs at a certain time $t = T_f$, when solving locally the PDE by a classical method. Then, the solution obtained at T_f may be used as a new initial data, solving the problem starting now from T_f rather than $t = 0$.

4 Results and performance

We now proceed with a performance analysis and a numerical example for a two-dimensional problem aiming to illustrate the behavior of the PDD algorithm. All simulations were carried out on the MareNostrum Supercomputer located at the Barcelona Supercomputing Center, using up to 1,024 processors.

4.1 Computational cost, memory consumption

Here we focus on both computational cost and memory consumption to estimate the performance of the

PDD method. Recall that once the solution has been computed on the interfaces, the full domain can be split into p subdomains, and assigned to different processors. In the subdomains in our decomposition, we used the Crank-Nicolson (implicit) finite difference method, and LAPACK for solve the associated linear algebraic system. In fact, the resulting matrix A associated to this problem to be solved is banded, with a bandwidth BW , say.

In the following, we keep fixed the space discretization of the two-dimensional finite difference computational mesh, choosing an equal number of computational nodes in both dimensions, that is $N_x = N_y = N$. Note that the bandwidth of the matrix A is N , being $A \in \mathbb{R}^{N^2 \times N^2}$. The memory consumption can be estimated as a function of the number of processors, p . When the matrix is not diagonally dominant, the LU factorization may be numerically unstable, and a partial pivoting with elimination might be necessary. In practice, this may require an additional storage to prevent a fill-in. Furthermore, reordering is often used to increase parallelism in the factorization. In [14,7], the local memory needed per processor, M_p , has been estimated, and shown to decrease proportionally to p , more precisely as

$$M_p = \frac{N^2}{p} 3BW. \quad (13)$$

Since the subdomains are now fully independent of each other, the bandwidth corresponds to the local matrix on each subdomain (assumed to be all identical, for simplicity), hence N/p . Therefore,

$$M_p = 3 \frac{N^3}{p^2}. \quad (14)$$

The local solver for the PDD method, being based on LAPACK for solving banded linear system, consists of the LU factorization followed by a forward/backward substitution. The computational cost is known to be of order of $N^2 BW^2$ for the LU factorization, and $N^2 BW$ for the forward/backward substitution, N^2 being the size of the square matrix and $BW = N/p$. Hence, with the local solver LAPACK, the computational cost of the local solver of the PDD can be readily estimated,

$$\begin{aligned} C &\approx \alpha_{opt}(4BW + 1) BW \frac{N^2}{p} + 3 \frac{N^3}{p^2} + \frac{N^2}{p} \\ &= \frac{N^2}{p} \left(4\alpha_{opt} \frac{N^2}{p^2} + (\alpha_{opt} + 3) \frac{N}{p} + 1 \right) \end{aligned} \quad (15)$$

see [14,7]. The last two terms are related to the computational time spent to compute the matrix coefficients and the right-hand side, respectively. Here the parameter $\alpha_{opt} < 1$ was introduced to account for the computational advantages gained when handling an optimized

LAPACK library, after tuning conveniently the BLAS library for the specific hardware platform used.

In view of such results, it is worth pointing out that the memory consumption for the PDD algorithm is proportional to p^{-2} . In practice, this allows the PDD algorithm to exploit at best the available computational resources, being capable to handle problems whose size can be much higher than other methods can afford. As for the computational cost, when $N \gg p$, the computational cost of the PDD method decreases as p^{-3} for large p .

Note that the probabilistic part of the algorithm (the Monte Carlo computations) turns out to be independent of the number of processors, p . The reason should be found in the way the PDD algorithm is implemented in practice on a parallel computer. We assigned the task of computing each set of interfacial values to a different processor. When the number of processors is sufficiently large, assigning these independent tasks to different processors can be done in a one-to-one mapping, leaving only one processor idle in such a mapping (in fact, the number of interfaces turns out to be equal to the number of processors minus one). Moreover, the size of the local linear algebraic problem corresponding to each subdomain is reduced considerably, making the computational time spent by the local solver negligible compared to the time spent by the Monte Carlo and the interpolation parts of the algorithm. Therefore, the overall computational time of the algorithm tends to remain constant when the number of processors is sufficiently large.

4.2 Numerical results

In this subsection we consider the following test problem.

Example A. The problem is given by

$$\begin{aligned} \frac{\partial u}{\partial t} &= (1 + \exp(-x^2)) \frac{\partial^2 u}{\partial x^2} + (1 + y^2 \sin^2(y)) \frac{\partial^2 u}{\partial y^2} \\ &\quad + (2 + \sin(x)e^y) \frac{\partial u}{\partial x} + (2 + x^2 \cos(y)) \frac{\partial u}{\partial y} \\ &\quad - u + \frac{1}{2}u^2 + \frac{1}{2}u^3, \end{aligned}$$

$$\text{in } \Omega = [-L, L]^2, \quad 0 < t < T, \quad u(x, y, t)|_{\partial\Omega} = 0, \quad (16)$$

with initial-condition $u(x, y, 0) = \cos^2\left(\frac{\pi x}{2L}\right) \cos^2\left(\frac{\pi y}{2L}\right)$. No analytical solution is known for this problem, hence the numerical error was controlled comparing the solution obtained with the PDD method with that given by a finite difference method, with a very fine space-time mesh.

Table 1 Example A: T_{TOTAL} denotes the computational time spent in seconds by PDD. T_{MC} , and T_{INT} correspond to the time spent by the Monte Carlo and the interpolation part, respectively; $Memory$ denotes the total memory consumption.

<i>Procs.</i>	T_{MC}	T_{INT}	<i>Memory</i>	T_{TOTAL}
128	454"	<1"	0.68 GBs	12983"
256	465"	<1"	0.17 GBs	3451"
512	468"	<1"	0.04 GBs	1184"
1024	464"	<1"	0.01 GBs	598"

In Fig. 3, the pointwise numerical error made when solving the problem of Example A by PDD is shown. Here the value of the parameters were kept fixed to $\Delta x = \Delta y = 10^{-2}$, $\Delta t = 10^{-3}$ and $L = 1$.

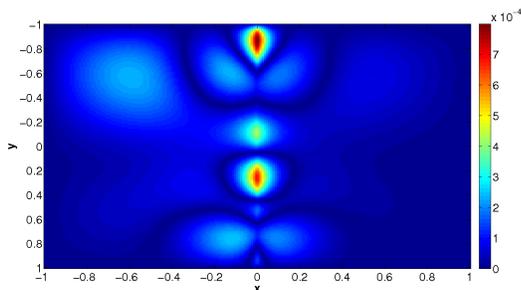


Fig. 3 Pointwise numerical error for the solution of problem "Example A", at $t = 0.5$, using two processors. Note that there is only one interface located at $x = 0$.

In Table 1 results from the Example A are shown, using up to 1,024 processors. Now, the parameters chosen for the local solver were $\Delta x = \Delta y = 2.5 \times 10^{-4}$, $T = 0.5$, $L = 1$ and $\Delta t = 10^{-3}$. Note that the algorithm scales with the number of processors according to a factor of approximately equal to 4 initially with $p = 128$, when doubling the number of processors. This is so when the number of processors is not too large ($p = 128$), being the computational workload per processor sufficiently large for this case.

5 Conclusions and future work

The PDD algorithm for solving nonlinear parabolic partial differential equations shows a remarkable scalability, being clearly suited for petascale supercomputers. Such algorithm allows to decouple the full problem into several independent subproblems, the computational cost being dramatically alleviated. Many different sources of parallelism for this algorithm can be exploited efficiently, since intercommunication overhead among processors is almost negligible. Moreover, the algorithm turns out to be naturally fault-tolerant. Concerning the

computational cost of the algorithm, it is worth to observe that the PDD algorithm scales quadratically or even cubically, provided that the workload per processor is sufficiently large, while the classical schemes scales linearly with the number of processors,

A numerical example has been given, showing the excellent scalability properties of the PDD algorithm in large-scale simulations, using up to 1,024 processors on a high performance supercomputer. Summarizing, the proposed algorithm appears to be a promising alternative to the traditional parallel schemes to solve nonlinear partial differential equations. Such an algorithm is characterized by desirable features, such as low intercommunication overhead, and fault-tolerance, allowing to exploit at best massively parallel supercomputers as well as Grid computing. Finally, it is worth to observe that in principle any nonlinear partial differential equations whose solution can be represented probabilistically is amenable to be solved by the PDD algorithm. Among them we have very important equations, such as the Maxwell-Vlasov [16], and the Navier-Stokes equations [17], governing Plasma Physics, and Fluid dynamics problems, respectively.

Acknowledgments

This work was supported by the Portuguese FCT. The authors thankfully acknowledge computer resources, technical expertise, and assistance provided by the Barcelona Supercomputing Center - Centro Nacional de Supercomputaci3n.

References

1. Acebr3n, 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. Acebr3n, 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. Acebr3n, J.A., Rodr3guez-Rozas, A., and Spigler, R., *Efficient parallel solution of nonlinear parabolic partial differential equations by a probabilistic domain decomposition*, (2009), submitted.
4. Acebr3n, J.A., Rodr3guez-Rozas, A., and Spigler, R., *Domain decomposition solution of nonlinear two-dimensional parabolic problems by random trees*, J. Comput. Phys., **228**, No. 15 (2009), 5574-5591.
5. Antia, H.M., *Numerical methods for scientists and engineers*, Tata McGraw-Hill, New Delhi, 1995.
6. Arbenz, P., Cleary, A., Dongarra, J., and Hegland, M., *A comparison of parallel solvers for diagonally dominant and general narrow-banded linear systems*, Parallel and Distributed Computing Practices, **2**, No. 4 (1999), 385-400.

-
7. Arbenz, P., Cleary, A., Dongarra, J., and Hegland, M., *A comparison of parallel solvers for diagonally dominant and general narrow-banded linear systems II*, EuroPar '99 Parallel Processing, Springer, Berlin, (1999), pp. 1078-1087.
 8. Arnold, L.: *Stochastic Differential Equations: Theory and Applications*. Wiley, New York (1974)
 9. DuChateau, P. and Zachmann, D.: *Applied Partial Differential Equations*. Dover Publications (2002).
 10. Freidlin, M.: *Functional Integration and Partial Differential Equations*. Annals of Mathematics Studies no. 109, Princeton Univ. Press, Princeton (1985)
 11. Karatzas, I., and Shreve, S.E.: *Brownian Motion and Stochastic Calculus*. 2nd ed., Springer, Berlin (1991)
 12. Keyes, D. E., *Domain Decomposition Methods in the Mainstream of Computational Science*, Proceedings of the 14th International Conference on Domain Decomposition Methods, UNAM Press, Mexico City, (2003), pp. 79-93.
 13. Kloeden, P.E., and Platen, E.: *Numerical Solution of Stochastic Differential Equations*. Springer, Berlin (1992)
 14. Petersen, W., and Arbenz, P.: *Introduction to parallel computing. A practical guide with examples in C*. Oxford Univ. Press, (2004).
 15. Quarteroni, A., and Valli, A., *Domain Decomposition Methods for Partial Differential Equations*, Oxford Science Publications, Clarendon Press, Oxford, 1999.
 16. Poisson-Vlasov in a strong magnetic field: A stochastic solution approach R. Vilela Mendes arXiv:0904.2214 (April 2009)
 17. Waymire, E.: Probability and incompressible Navier-Stokes equations: An overview of some recent developments. *Prob. Surveys*, **2** 1-32 (2005)