Efficient parallel solution of nonlinear parabolic partial differential equations by a probabilistic domain decomposition

Juan A. Acebrón [a] , Ángel Rodríguez-Rozas [a] , and Renato Spigler [b]

[a] *Center for Mathematics and its Applications, Department of Mathematics, Instituto Superior Técnico Av. Rovisco Pais 1049-001 Lisboa, Portugal*

[b] *Dipartimento di Matematica, Università "Roma Tre", Largo S.L. Murialdo 1, 00146 Rome, Italy*

Abstract

Initial- and initial-boundary value problems for nonlinear one-dimensional parabolic partial differential equations are solved numerically by a probabilistic *domain decomposition* method. This is based on a probabilistic representation of solutions by means of *branching* stochastic processes. Only few values of the solution inside the space-time domain are generated by a Monte Carlo method, and an interpolation is then made so to approximate suitable interfacial values of the solution inside the domain. In this way, a fully decoupled set of sub-problems is obtained. This method allows for an efficient massively parallel implementation, is scalable and fault tolerant. Numerical examples, including some for the KPP equation and beyond are given to show the performance of the algorithm.

Key words: Monte Carlo methods, domain decomposition, branching stochastic processes, nonlinear parabolic problems, parallel computing, fault-tolerant algorithms
PACS: 65C05, 65C30, 65M55, 65N55

Email addresses: juan.acebron@ist.utl.pt (Juan A. Acebrón), angel.rodriguez@ist.utl.pt (Ángel Rodríguez-Rozas), spigler@mat.uniroma3.it (Renato Spigler).

1 Introduction

Probabilistic representations of solutions of certain partial differential equations (PDEs) have become popular nowadays, since in some cases they provide powerful analytical tools to establish existence, uniqueness, and various properties of solutions. On the contrary, among the numerical methods successfully developed over the years to solve PDEs, the probabilistic methods, based on Monte Carlo techniques, have never been the most used, due to their intrinsically rather poor efficiency.

However, probabilistic methods for solving deterministic problems has been proved to be very useful, indeed competitive, in evaluating high-dimensional integrals, since in deterministic quadrature rules the number of function evaluations grows exponentially as the dimension increases, while it is independent of the dimension in case of the Monte Carlo methods [21].

On the other hand, probabilistic methods offer important computational advantages even solving PDEs, since the algorithms and the corresponding codes are especially suited for *parallel computing* [28]. This is due to the fact that the solution is computed through an expected value over a given finite sample whose elements are independent from each other. This is of paramount importance since it allows to develop parallel codes with extremely low communication overhead among the various processors, and affects positively crucial properties such as *scalability* and *fault tolerance*. The latter should be intended as valid when even a single processor falls out during the computation and is known that it does.

When the computational domain is split into a number of subdomains, and correspondingly a number of independent processors are used to realize a domain decomposition solution of the full problem, the tightly-coupled nature of the classical deterministic numerical schemes requires a frequent exchange of data among the processors, thus degrading appreciably the overall performance. We stress that scalability and fault tolerance are critical issues in view of the high performance supercomputers of present and future generation, equipped with hundred of thousands of processors, in order to fully exploit the available computational resources [3,14]. In addition, an important practical implication of the probabilistic representation of solutions rests on the possibility of obtaining the solution even at a single point inside the domain. The deterministic methods, instead, require solving the entire problem, owing to the global nature of the boundary-value problem for PDEs.

In the last few years, several methods have been proposed to solve numerically *nonlinear* PDE problems probabilistically, see [27,34,35,37], e.g. However, the possibility of exploiting such methods efficiently on parallel computers (even

for linear problems), seems to have been overlooked, and several attempts have been made only recently for this purpose for some linear problems. In [1,2], in fact, a hybrid algorithm, based on Monte Carlo and classical domain decomposition methods, has been proposed for solving linear elliptic boundary value problems. In short, the idea consists of generating only few interfacial values along given, possibly artificial interfaces inside the domain, then obtaining approximate values upon interpolation on such interfaces; see also [29]. Such values are used as boundary data to split the original problem into a number of fully decoupled sub-problems. The method was called "probabilistic domain decomposition" method (PDD, for short), since it combines the two main ingredients, a probabilistic approach for evaluating interfacial values, and a classical domain decomposition strategy [11,31]. Such an algorithm was applied successfully to solve two-dimensional elliptic problems, showing excellent scalability properties when runned on high performance supercomputers [4].

Concerning nonlinear PDEs, we should recall the seminal work by H.P. McKean [26], where a probabilistic representation of the solution to the nonlinear parabolic equation known as Kolmogorov-Petrovskii-Piskunov equation (KPP, for short) was found. Deep contributions by M. Freidlin [16] to the generalized KPP equation should also be acknowledged. Other important achievements to obtain probabilistic representation of solutions to PDEs was accomplished in [38] and in [17].

A few authors have contributed, over the years, to the solution of certain nonlinear PDE problems by probabilistic methods. Notably, A.S. Sherman and C.S. Peskin [35] devised the later called "gradient random walk" method to solve one-dimensional reaction diffusion equations, e.g. the KPP equation, subject to purely initial values. The idea is based on solving, rather, the PDE satisfied by the gradient of the seeked solution, with some advantages. An extension to the two-dimensional case was instead presented in [36], where several previous results, such as those by Chorin, e.g., [13], and by others are recalled. The method of Sherman and Peskin enjoys the remarkable properties of being grid free, automatically self-adaptive (providing high resolution near sharp fronts), and stable independently of the time step size, but, admittedly computationally expensive [36].

The KPP equation and the Sherman-Peskin method have been studied, mostly from a theoretical point of view, by B. Chauvin and A. Rouault in several papers, who proved the convergence of that method in [12]. H. Régnier and D. Talay in [33,34] established the rate of convergence of Sherman-Peskin method, extending its applicability to some more general nonlinear one-dimensional convection-reaction-diffusion equations.

Finally, in [32] J.M. Ramirez was motivated to acquire a physical intuitive

picture for the branching stochastic model introduced in the probabilistic representation derived in [25]. Here, a linear diffusion equation as well as the Burgers equation were solved numerically by means of convenient branching random walks, and later improved using a Picard iteration scheme.

In all aforementioned papers, however, no issue concerning efficiency of the numerical approach was ever addressed. Considering that all probabilistic methods are essentially based on some Monte Carlo or Monte Carlo-like approach, and consequently characterized by a poor performance, this point should be considered of primary importance. Related issues are (as recalled above) massive parallelism, scalability, and fault tolerance of the corresponding algorithms.

The purpose of this paper is to generalize our previous approach, based on the PDD method and earlier developed for linear elliptic equations, to solve initial- as well as initial-boundary value problems for linear and especially nonlinear parabolic differential equations. The linear case is considered here merely to illustrate in the simplest way the PDD algorithm that we propose. Rather unexpectedly, however, it turned out that even in such case important computational advantages can been observed with respect to some existing more traditional parallel schemes. In order to assess the computational feasibility of our algorithm, we have compared our results with those obtained using competitive (freely available) parallel numerical codes, which are widely used by the high-performance scientific community.

The plan of the paper is the following. In Section 2, we first consider, for clarity, linear problems. Some necessary mathematical generalities are provided and the algorithm is described. In Section 3, we consider nonlinear problems,and the complexity of the probabilistic part as well as various sources of numerical errors, which affect the PDD method are discussed. Numerical examples, among which some concerning the KPP equation, are given in this section, where the efficiency of the PDD algorithm is illustrated. In a short final section, we summarize the high points of the paper.

2 Linear problems

In this section we consider only linear problems, in order to better illustrate our method. We collect some preliminary results on the probabilistic treatment in § § 2.1, and present numerical examples in § § 2.2.

Diffusion processes appear frequently in Physics and Engineering, in modeling a broad variety of phenomena. Inspired by R. Feynman's work on path integrals in quantum theory, M. Kac realized that a similar formulation could be applied to the solution of the heat equation, and other related diffusive (parabolic) linear partial differential equations. This representation is the Feynman-Kac formula.

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

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

where $x \in \mathbf{R}$, L is a linear elliptic operator, say $L = a(x,t)\partial_{xx}/2 + b(x,t)\partial_x$, with continuous bounded coefficients, $c(x,t) \geq 0$ 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

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

see [16,22], e.g., where $\beta(\cdot)$ is the stochastic process starting at $(x,0)$, 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 one-dimensional Brownian motion, and the measure reduces to the Gaussian measure. In general, the (one-dimensional) 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.,

$$d\beta = b(\beta,t)\,dt + \sigma(\beta,t)\,dW(t). \tag{3}$$

Here $W(t)$ represents the one-dimensional standard Brownian motion (also called Wiener process); see [22,6], e.g., for generalities, and [23] 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 $a > 0$.

The representation in Eq. (2) can be generalized to deal with problems on bounded domains, say $\Omega \subset \mathbf{R}$, where some boundary data $u(x,t)|_{x\in\partial\Omega} =$

$g(x,t)$ of the Dirichlet type is prescribed. Thus, the following representation holds for the solution, being now continuous and bounded on $\overline{\Omega} \times (0,T]$,

$$u(x,t) = E\left[f(\beta(t))\, e^{-\int_0^t c(\beta(s),t-s)ds}\, \mathbf{1}_{[\tau_{\partial\Omega}>t]}\right]$$
$$+E\left[g(\beta(\tau_{\partial\Omega}), t-\tau_{\partial\Omega})\, e^{-\int_0^{\tau_{\partial\Omega}} c(\beta(s),t-s)ds}\, \mathbf{1}_{[\tau_{\partial\Omega}<t]}\right]. \tag{4}$$

Here $\tau_{\partial\Omega}$ denotes the first exit (or hitting) time of the path $\beta(\cdot)$, started at $(x,0)$, when $\partial\Omega$ is crossed, and $\mathbf{1}_{[\tau_{\partial\Omega}>t]}$ is the indicator (or characteristic) function, being 1 or 0 depending whether $\tau_{\partial\Omega}$ is or is not greater than t.

The final goal is to compute the solution at a few single points, inside the space-time domain. Computing the solution at a high number of points so to cover a full computational domain is possible but exceedingly expensive, even though this approach could be pursued when the available processors are extremely numerous.

In case of linear parabolic equations, it is required generating a given number of random paths, obeying the stochastic differential equation in (3), and track them until they touch the boundary for the first time, or reach a prescribed final time t. The latter corresponds to the case of an initial value problem, while the former to a Dirichlet boundary value problem. Then, the solution to the equation at a given point (x,t) can be obtained by the Feymann-Kac formula in (4) or (2), depending on whether boundary conditions are prescribed or not. Here the expected value is replaced by an arithmetic mean, since in practice we deal with a finite sample size, N. An alternative strategy to evaluate the solution was proposed in [32], which 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 $\beta_j(t)$, $j = 1, 2, \ldots, N$, contribute or not to the solution. The solution is then computed simply as

$$u(x,t) = E[f(\beta(t)] = \frac{1}{N}\sum_{j=1}^{N} f(\beta_j(t)). \tag{5}$$

Consider, for the purpose of illustration, the Dirichlet boundary value problem for the one-dimensional heat equation, in presence of a constant sink term, $c > 0$, that is

$$\begin{aligned} &\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} - cu, \quad a < x < b,\, t > 0 \\ &u(x,0) = f(x). \\ &u(a,t) = f(a) = 0,\, u(b,t) = f(b) = 0 \end{aligned} \tag{6}$$

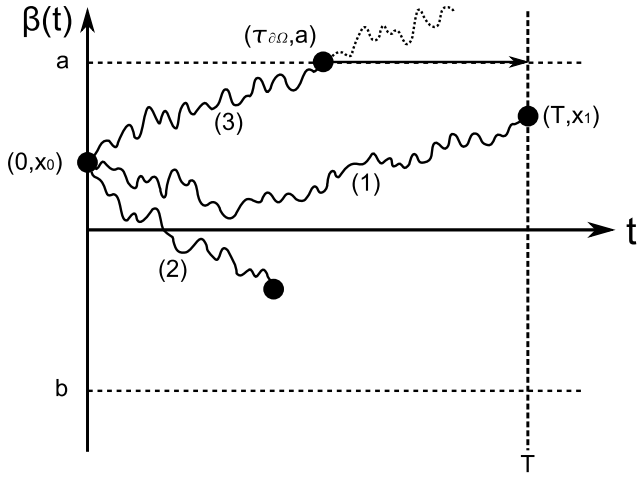

Fig. 1. The three possible scenarios for a random path for the one-dimensional heat equation with a constant sink term.

In Fig. 1, we sketch the three possible scenarios the random paths $\beta_j(t)$ can undergo. In fact, it may happen that a given random path which is governed by a randomly chosen exponential time, S, either reaches the boundary of the space-time domain or not. If it does, it may either hit the parabolic boundary or the remaining part of the boundary, i.e., $[a, b] \times \{t\}$. When the generated value of S is less than t, the path above is ignored in the computation of the solution at time t.

Generally speaking, it is important to evaluate accurately the first time that the path crosses the parabolic boundary, hence some care should be paid to such a task, [19,20]; see also [9]. This was done in [1,2], for the case of linear elliptic two-dimensional BV problems, and has also been done in this paper, for parabolic problems.

2.2 The numerical method

In practice, the algorithm can be decomposed in three steps. In order to illustrate how it works, we plotted in Fig. 2 a sketchy diagram where such steps are shown. In short, the first step concerns computing the solution at a few points along chosen (possibly artificial) interfaces, by means of a probabilistic, Monte Carlo-type method, see Fig. 2(b). Once the solution is known at such points, the second step consists in interpolating on such "nodes", obtaining interfacial values of the sought solution, as is shown in Fig. 2(c). Finally, in Fig. 2(d) the third step is devoted merely to compute the solution inside each subdomain, assigning the numerical solution on each subdomain to separate

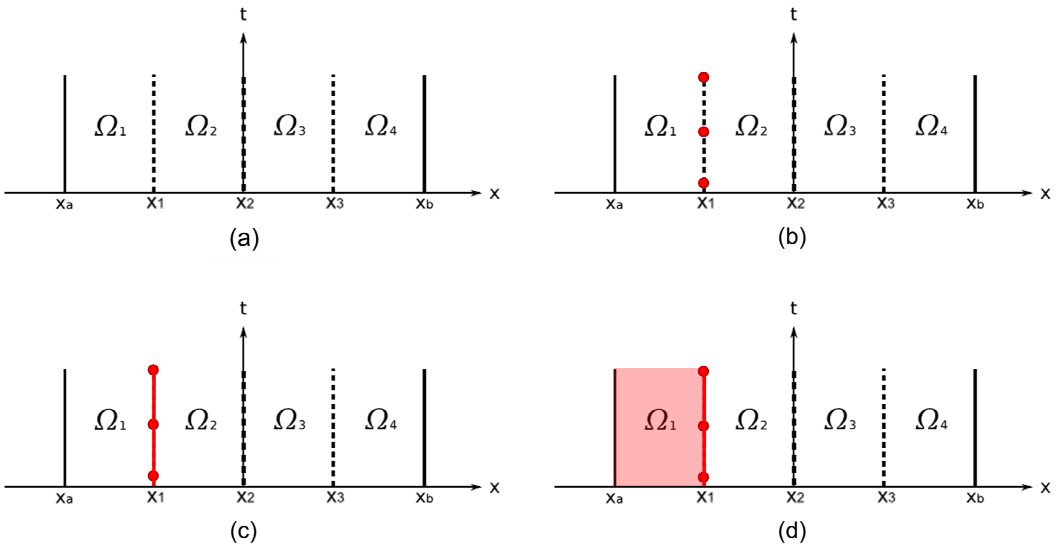

Fig. 2. A sketchy diagram illustrating the main steps of the algorithm.

processors, resorting to classical numerical methods, such as finite differences or finite elements methods, as local solvers.

2.3 Interpolation in time

Consider a PDE parabolic problem on a rectangular domain in space-time, $D := [a, b] \times (0, T]$, and divide its closure, $\overline{D} := [a, b] \times [0, T]$ in P "slices", $D_i := [x_{i-1}, x_i] \times [0, T]$, with $i = 1, 2, \ldots, P$, $x_0 = a$, $x_P = b$. Assume that we have obtained, by the Monte Carlo procedure above, some values of the solution at the points (x_i, t_j), for some values of j, say $j = 1, 2, \ldots, J$. We then interpolate on each interval $x = x_i$ the sought solution, $u(x_i, t)$, on the nodes t_j. This can be done, e.g., by Chebyshev polynomials.

2.4 Local solver

Once we have obtained the interfacial approximations of $u(x_i, t)$, say $\tilde{u}(x_i, t)$, for each fixed i, $i = 1, 2, \ldots, P$, and $0 \leq t \leq T$, we face the possibility of solving the given parabolic problems on each subdomain, D_i independently of each other, since now the previously missing boundary values (on the interfaces $x = x_i$, interval to D) are available. This allows for a *parallel* solution, assigning the task of solving each problem to a separate processor. Local solvers of any kind, even different on each subdomain, can be used.

We stress that in practice there are three sources of parallelization, namely (1) the Monte Carlo generation of internal node functional values (even each single

tree can be runned on independent processors), (2) the interpolation part (the interpolation on each interface can be accomplished independently), and (3) the domain decomposition solution (that can be assigned to independent local solver). Moreover, each of such three stages enjoyed a natural fault tolerant property: (1) if a number of processors fail in the Monte Carlo simulations, it will be enough to ignore the result from them using the remaining trees. Hence, at a price of a small additional errors, the algorithm will still provide meaningful results. (2) Failure of processors computing interpolated values of the solution on some interfaces may only imply to neglect, temporarily, the solution on those subdomains having such interfaces as part of their boundary. (3) Failure of processors responsible for the numerical solution on some sub-domains can also be temporarily neglected, while the solution computed by the local solvers on the remaining sub-domains will be computed correctly. Note that on the interfaces and on the sub-domains where the processors failed, the solution can be computed restarting again the algorithm.

2.5 Numerical examples

In this subsection, we present some numerical examples to illustrate the probabilistically induced domain decomposition algorithm developed in the previous subsection. All simulations below were runned on the massively parallel supercomputer MareNostrum, exploiting up to $1,024$ processors from a total of $10,240$ processors. Such a supercomputer is capable to deliver a peak performance of 94.21 Teraflops, and its hybrid architecture consists of 2,560 nodes, each with IBM PowerPC 970MP processors and 8 Gb of shared memory, linked through a Myrinet intercommunication network. The operating system is Linux, and the compilers used are based on IBM XL Compilers.

In order to assess the performance of our method, a comparison was made solving the problem by classical numerical schemes. The space-time domain as well as the subdomains in our decomposition being simple, we used a Crank-Nicolson finite difference method. On the subdomains we used LAPACK, while the full domain solution was computed by SCALAPACK, which is suited for the parallel solution of banded linear systems.

Example 1. An IV problem for the heat equation with potential.

Consider the problem

$$u_t = u_{xx} - u, \quad \text{on } \mathbf{R} \times (0,T], \quad u(x,0) = \frac{e^{-\frac{x^2}{0.4}}}{\sqrt{0.4\pi}}, \ x \in \mathbf{R},$$

whose analytical solution, that we use to estimate the overall numerical error,

is

$$u(x,t) = e^{-t}\,\frac{e^{-\frac{x^2}{4(t+0.1)}}}{\sqrt{4\pi(t+0.1)}}.$$

The necessarily bounded computational domain was chosen spatially large, i.e., $-5\times10^5 \le x \le 5\times10^5$, and the space and time step sizes $\Delta x = 10^{-2}$, $\Delta t = 10^{-3}$. Recall that, implementing the PDD method, the space-time domain is decomposed in slices as shown in Fig. 2.

Note that our PDD method also allows to avoid approximating the unbounded domain of the original problem by a bounded domain, thus creating artificial boundaries on which suitable BCs should be imposed. In fact, the solution on any given artificial boundary could be computed by means of the PDD, and this can be done, in principle, accurately.

We compared the results obtained by the PDD with those computed through SCALAPACK on the whole domain, correspondingly to the same error (which can be precisely estimated having the analytical solution). In the PDD algorithm, the numerical errors come from the probabilistic part, the interpolation, and the local solvers. The first dominates, and the Monte Carlo evaluation of the nodal values is characterized by a statistical error of order of $O(N^{-1/2})$, N being the sample size, cf. [1]. Hence, we used typically $N = 10^6$ realizations. In Figure 3, the pointwise numerical error made solving numerically the case above by the PDD method is shown. Since the computational domain is rather large, for the purpose of illustration only a slice of the full domain, $[-10, 10] \times (0, T]$, is depicted. Note that the maximum value of the error is of order of 10^{-3}, which corresponds to the statistical error, being $N = 10^6$ the sample size. In the following examples, the pointwise numerical error has also been computed, and it turns out to be always of the same order.

In Fig. 4 a comparison is shown between the computational time achieved with the PDD method and with SCALAPACK, both measured in units of t_0, using up to $1,024$ processors. Here t_0 is the computational time achieved by the faster method running with 128 processors, which turns out to be 574 seconds spent by the PDD algorithm. Clearly, not only the PDD outperforms the other, but more, the the SCALAPACK based method seems to become closer to a saturation.

Example 2. A BV problem for the heat equation with potential. Consider the problem

$$u_t = L^2\,u_{xx} - u, \quad u(x,0) = \sin\left(\frac{\pi x}{L}\right), \quad u(0,t) = u(L,t) = 0,$$

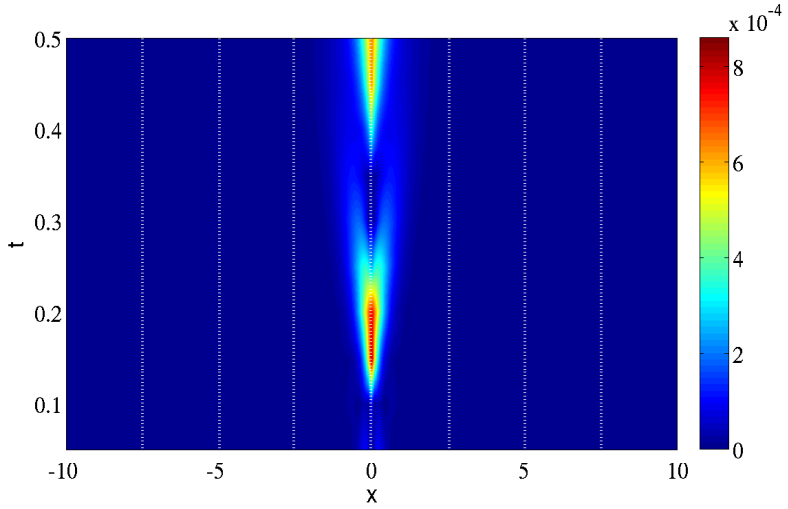

Fig. 3. Example 1: Pointwise numerical error made by the PDD algorithm. The white dotted lines denotes the interfaces separating the different subdomains in which the full domain was partitioned. Parameters are $\Delta x = 10^{-2}$, $\Delta t = 10^{-3}$, $N = 10^6$.

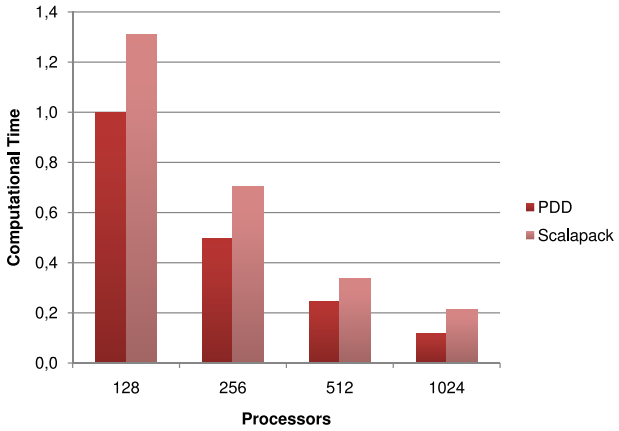

Fig. 4. Example 1: Comparison of the computational times (being $t_0 = 574$ seconds) for both methods, PDD and SCALAPACK, for different number of processors.

whose solution is

$$u(x,t) = e^{-(1+\pi^2)t} \sin\left(\frac{\pi x}{L}\right).$$

In Fig. 5, the computational times for both methods, the PDD and SCALAPACK, are shown. Here, and in the following examples for BV problems, the parameter L is kept fixed to 10, and Δx and Δt, to 10^{-8} and 10^{-3}, respectively. Note again that the measured computational times with the PDD method are significantly shorter than those obtained with SCALAPACK, and this for any number of processors.

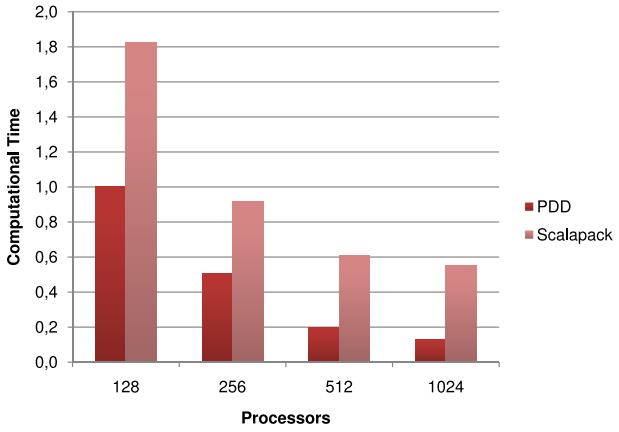

Fig. 5. Example 2: Comparison of the computational times (measured in units of $t_0 = 1,328$ seconds) for both methods, PDD and SCALAPACK, for different number of processors.

3 Nonlinear problems

In the following, we consider nonlinear PDE problems, focusing first in some probabilistic preliminary representations, and then the numerical method based on them.

3.1 Probabilistic representations

Probabilistic representations do exist also for *semilinear* parabolic equations. Indeed, in [26] H.P. McKean derived the representation formula

$$u(x,t) = E[\prod_{i=1}^{k_t(\omega)} f(x_i(\omega,t))] \tag{7}$$

for the KPP equation

$$u_t = u_{xx} + u(u-1), \quad -\infty < x < +\infty, \ t > 0, \tag{8}$$

subject to the initial value $u(x,0) = f(x)$, for $-\infty < x < +\infty$; see also [16]. It is understood that in (7) the point $x_i(\omega,t)$ is the position of the ith stochastic process surviving at time t, ω denoting the chance variable. The quantity $k_t(\omega)$ is the random number of descendants at time t. A similar representation can be derived for the solution to more general nonlinear parabolic equation problems like

$$\frac{\partial u}{\partial t} = Lu - cu + \sum_{i=2}^{m} \alpha_i u^i, \tag{9}$$

where $m \geq 2$ is an integer, $\alpha_i \geq 0$, $\sum_{i=2}^{m} \alpha_i = 1$, and c is a positive constant. Such a representation is based on generating *branching* diffusion processes, associated with the elliptic operator in Eq. (1), and governed by an exponential random time, S, with probability density $p(S) = c\,\exp(-cS)$. In order to illustrate how the method can be applied and works in practice, consider first the Cauchy problem for Eq. (9), with the initial value $u(x,0) = f(x)$, and a purely quadratic nonlinearity, $\alpha_2 = 1$. Applying the Duhamel principle [15] to Eq. (9), we obtain

$$\begin{aligned} u(x,t) &= e^{-ct} \int_{\Omega} dy\, f(y)\, p(x,t,y,0) \\ &+ \int_0^t \int_{\Omega} ds\, dy\, e^{-cs}\, u^2(y,t-s)\, p(x,s,y,0), \end{aligned} \tag{10}$$

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

$$\begin{aligned} &\frac{\partial p}{\partial t} = Lp, \quad x \in \Omega, \quad t > \tau \\ &p(x,\tau,y,\tau) = \delta(x-y). \end{aligned} \tag{11}$$

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

$$p(x,t,y,\tau) = E[\delta(\beta(t) - y)], \tag{12}$$

where $\beta(t)$ is the solution to the stochastic differential equation in (3), with initial condition $\beta(\tau) = x$. Eq. (10) yields

$$u(x,t) = E[f(\beta(t))\, \mathbf{1}_{[S>t]}] + \frac{1}{c} E[u^2(\beta(t-S), t-S)\, \mathbf{1}_{[S<t]}], \tag{13}$$

where the time S is a random time, picked up from the exponential density distribution $p(S) = c\exp(-c\,S)$. The integral equation in Eq. (10) can be recursively solved, replacing the last term on the right-hand side with the solution $u(x,t)$, obtaining the following expansion in terms of multiple exponential random times, S_i,

$$\begin{aligned} u(x,t) &= E[f(\beta(t))\, \mathbf{1}_{[S_0>t]}] \\ &+ \frac{1}{c} E[f(\beta(t-S_0))\, \mathbf{1}_{[S_1>t-S_0]} f(\beta(t-S_0))\, \mathbf{1}_{[S_2>t-S_0]}\, \mathbf{1}_{[S_0<t]} \\ &+ \frac{1}{c^2} E[f(\beta(t-S_0))\, \mathbf{1}_{[S_1>t-S_0]} f(\beta(t-S_0-S_2))\, \mathbf{1}_{[S_3>t-S_0-S_2]} \end{aligned}$$

$$\times f(\beta(t-S_0-S_2))\mathbf{1}_{[S_4>t-S_0-S_2]}\mathbf{1}_{[S_2<t-S_0]}\,\mathbf{1}_{[S_0<t]}] + \cdots \tag{14}$$

Note that in Eq. (14) each term contributes, in part, to the full solution.

A clear picture can be obtained, in terms of branching stochastic processes as follows: We first generate a sufficiently high number of random exponentially distributed times, S_i, such that their sum is less than or equal to the final time, t. For every random time, we split the given stochastic process-solution of (3) in as many branches as those corresponding to m, the degree of nonlinearity. They depart from the point where the previous stochastic process was at time S_i, and continue along independent trajectories until the next occurrence, S_j ($j > i$), takes place. Whenever one of the possible branches reaches the final time, t, the initial value, f, is evaluated at the position where the stochastic process was located. Finally, the solution is reconstructed multiplying all contributions coming from each branch. For the purpose of illustration, in the second picture of Fig. 6 a sketchy configuration is plotted, correspondingly to three branches at the final time, t. Note that such a configuration represents graphically what appears in the last term of Eq. (14). Using such a branching stochastic process representation, Eq. (14) can be reformulated as

$$u(x,t) = E[\prod_{i=1}^{k_t(\omega)} f(x_i(t,\omega))], \tag{15}$$

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

$$u(x,t) = E\left[\prod_{i=1}^{k_t(\omega)} \left\{f(x_i(t,\omega))\mathbf{1}_{[t<\tau_{\partial\Omega_i}]} + g(\beta_i(\tau_{\partial\Omega_i}), t-\tau_{\partial\Omega_i})\mathbf{1}_{[t>\tau_{\partial\Omega_i}]}\right\}\right], \tag{16}$$

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

Note that, in the probabilistic representation of the solution to 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), and (4). An average is taken over all trees with a given "root", which will be the space point x.

It is useful at this point to recall some terminology pertaining to trees, usually directed trees. A tree is a connected graph, i.e., a set of nodes linked by *edges*, with only one starting node, called *root*, a number of final nodes, called *leaves*, while the other (internal) nodes are called *vertices*, and we call *branch* every set of edges joining the root to a given leaf.

The strategy we followed can be better explained through an example. Consider, as a simple case, the initial-boundary value problem for the classical KPP equation,

$$\begin{aligned} &\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} - cu + u^2, \quad a < x < b,\, t > 0 \\ &u(x,0) = f(x) \\ &u(a,t) = f(a) = 0,\, u(b,t) = f(b) = 0. \end{aligned} \tag{17}$$

Here the picture is more involved compared to the case of linear equations, since the various realizations of suitable stochastic processes are replaced by realizations of suitable random trees. Proceeding initially as in the linear case, when the randomly generated time, S, turns out to be less than t, a bifurcation in two new paths occurs, and both will evolve according to the same rule. On each of such paths we consider a new randomly generated time (according to the same probability distribution as before), at which again pairs of random paths will start. This procedure generates a tree, which ends only when all the bifurcating paths hits the boundary of the space-time domain. There are again, as in the linear case, two possible ways to hit the boundary, i.e., hitting the parabolic boundary, or the set $[a,b] \times \{t\}$. Note that, others than in the linear case, both paths exiting from a given bifurcation point may affect the solution in a different way, and in fact they contribute to the first or second term in (4). A careful boundary treatment is required also in the present case.

3.2 Analysis of the probabilistic part of the PDD method for nonlinear PDEs

In this subsection we estimate the computational complexity in terms of the computational time required to compute the probabilistic part of the algorithm, when solving a nonlinear PDE as that in (9), at a single point (x,t). More precisely, we generate N branching stochastic processes initially located at x, and follow them until they reach a prescribed final time, t.

The branching stochastic process associated to the nonlinear term u^m, requires creating m branches every time a "splitting event" occurs, according to an exponentially distributed time. The latter occurs according to the linear term, $-cu$. Note that the number of branches, k, is related to the number of splitting events, N_e, by $k = (m-1)N_e + 1$.

The computational time spent to generate any given branch, is a function of the final time, t, as well as of the time step, Δt, chosen to solve numerically the associated stochastic differential equation in (3). In addition, we should take into account the random times Δt_s responsible for branching. Recall that these are picked up from the exponential distribution $c\exp(-c\Delta t_s)$. It

is thus necessary that the time-step discretization, used to solve (3), also captures the instants when the random exponential times occur. In practice, the actual time step is chosen according to the minimum value between Δt and Δt_s. Note that Δt_s is chosen randomly, and the probability of being less than Δt is $1 - \exp(-c\Delta t)$. When this occurs, the actual time step used for the numerical solution of (3) should be Δt_s. Averaging over all branching stochastic processes, we obtain the most probable time step to be used, which is given by

$$\overline{\Delta t_s} = \int_0^{\Delta t} ds\, s\, c\, e^{-cs} + \Delta t \int_{\Delta t}^{\infty} ds\, c\, e^{-cs} = \frac{1 - e^{-c\Delta t}}{c}. \tag{18}$$

The computational time can be measured, typically, in terms of the number of iterations in time, required to fully generate a branching stochastic process with k branches up to the final time, t. Defining t_c as the time spent per iteration, such computational time can be estimated as $kt_c\, t/\overline{\Delta t_s}$. In case of N branching processes, the average computational time, t_b (b standing for "branching"), turns out to be

$$t_b = N \sum_{k=1}^{\infty} kt_c \frac{t}{\overline{\Delta t_s}} P(k), \tag{19}$$

where $P(k)$ is the probability of finding a branching stochastic process with k branches.

Such a probability can be evaluated by first enumerating and then summing up the various probabilities, $p_i(k)$, of having k branches as a final configuration, that is

$$P(k) = \sum_{i=1}^{N_D} p_i(k). \tag{20}$$

Here N_D denotes the total number of possible diagrams characterized by k branches, and $p_i(k)$ the probability of obtaining each of them. In Fig. 6 we show, for the purpose of illustration, some diagrams for $k = 2, 3, 4$. In particular, for $k = 2$, and 3 the total number of possibilities of obtaining 2, and 3 branches is shown. It turns out that each diagram contributes equally to the probability function (20). Therefore, such a probability can be obtained simply counting the number of possible diagrams, N_D, with k branches, and then multiplying by the probability of having one of them, that is $P(k) = N_D p_1(k)$. For convenience, we consider the special diagram shown in Fig. 7. The probability of obtaining such a diagram as a final configuration is given by

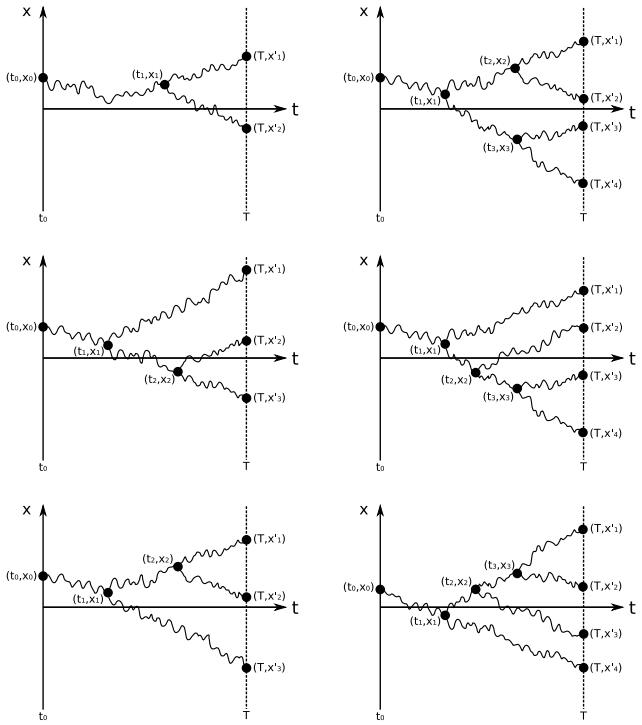

Fig. 6. Configuration diagram for the case of 2, 3, and 4 branches.

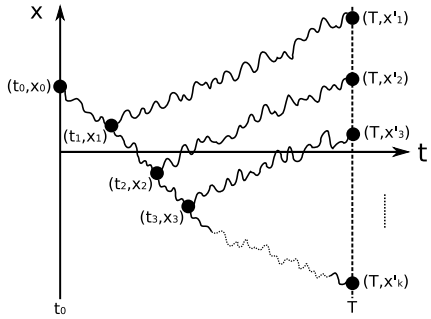

Fig. 7. Configuration diagram with k branches.

$$p_1(k) = c^{N_e} \int_0^t d\,t_1 \int_{t_1}^t d\,t_2 \int_{t_2}^t d\,t_3 \cdots \int_{t_{n-1}}^t d\,t_n\, e^{-ct_1}\, e^{-c(t_2-t_1)}\, e^{-c(t_3-t_2)} \cdots e^{-c(t_n-t_{n-1})}$$
$$\times e^{-cm(t-t_n)}\, e^{-c(m-1)(t-t_{n-1})} \cdots e^{-c(m-1)(t-t_2)}\, e^{-c(m-1)(t-t_1)}. \quad (21)$$

Changing variables by $t_i' = t_i - t$, where $i = 1, \ldots, N_e$, it follows

$$p_1(k) = c^{N_e}\, e^{-ct} \int_{-t}^0 d\,t_1' \int_{t_1'}^0 d\,t_2' \ldots \int_{t_{n-1}'}^0 d\,t_n' e^{c(m-1)t_1'}\, e^{c(m-1)t_2'} \ldots\, e^{c(m-1)t_n'}, \quad (22)$$

which can be readily integrated, yielding

$$p_1(k) = \frac{1}{N_e!(m-1)^{N_e}}\, e^{-ct}\left[1 - e^{-c(m-1)t}\right]^{N_e}. \tag{23}$$

Concerning the number of possible diagrams, N_D, this should be a function of the number of splitting events, N_e. In fact, it obeys the recursive relation

$$N_D(N_e) = N_D(N_e - 1)\,\nu(N_e - 1), \tag{24}$$

where $\nu(N_e - 1)$ is the number of branches corresponding to $N_e - 1$ splitting events. Repeating further (24), we obtain

$$N_D(N_e) = \prod_{i=0}^{N_e-1} [i(m-1)+1] = (k-m+1)!^{(m-1)}, \tag{25}$$

where $(k-m+1)!^{(m-1)}$ denotes the multifactorial $m-1$ of $(k-m+1)$, i.e., $n!^{(k)} := 1$ if $0 \le n < k$, and $n!^{(k)} := n(n-k)!^{(k)}$ if $n \ge k$.

Summarizing, the probability of finding a brownian motion with k branches is given by

$$P(k) = (k-m+1)!^{(m-1)} \frac{1}{N_e!(m-1)^{N_e}}\, e^{-ct}\left[1 - e^{-c(m-1)t}\right]^{N_e}, \tag{26}$$

where $N_e = (k-1)/(m-1)$.

The validity of this analytical result can be confirmed by some numerical simulations, consisting of generating N branching brownian motion, governed by exponential random times, and counting the number of branches obtained. A comparison between the probability $P(k)$ as function of k, obtained both numerically and theoretically, is plotted in Fig. 8. This has been done for the case $m = 2$ in Fig. 8(a), and $m = 3$ in Fig. 8(b), and for two different values of the final time, t. The perfect agreement validates the formula (26).

Once the probability function, $P(k)$ is known, the computational time t_b spent during the probabilistic part of the algorithm, can be evaluated. From (19), we have

$$t_b \le N t_c \frac{t}{\overline{\Delta t_s}} \sum_{k=1}^{\infty} kP(k), \tag{27}$$

which can be promptly evaluated when $m = 2$. In fact, in this case

$$P(k) = e^{-ct}\left(1 - e^{-ct}\right)^{k-1}, \tag{28}$$

and

$$\sum_{k=1}^{\infty} kP(k) = e^{-ct} \sum_{k=1}^{\infty} k(1-e^{-ct})^{k-1} = \frac{e^{-ct}}{(1-e^{-ct})^2}, \tag{29}$$

and hence, for $t \to +\infty$,

$$t_b = O\left(N t_c \frac{t}{\Delta t_s} e^{ct}\right). \tag{30}$$

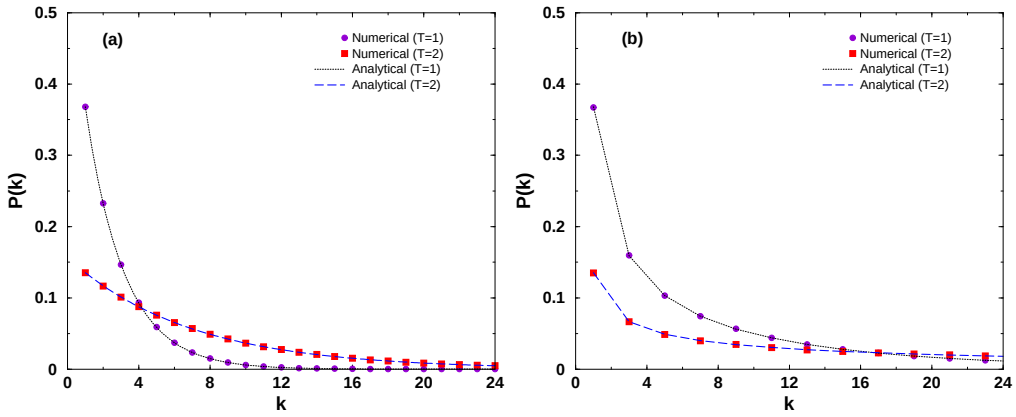

Fig. 8. Comparison between the probability function $P(k)$ obtained analytically and numerically simulating N branching brownian motion processes. This has been done for (a) $m = 2$, and (b) $m = 3$, for two different values of the final time; $t = 1$, and $t = 2$. Parameters are $N = 10^6$, $c = 1$.

Similarly, we have, for $m = 3$,

$$P(k) = \frac{(k-2)!!}{\left(\frac{k-1}{2}\right)! 2^{(k-1)/2}} e^{-ct} \left[1 - e^{-2ct}\right]^{(k-1)/2}, \tag{31}$$

which can be simplified being

$$\frac{(k-2)!!}{\left(\frac{k-1}{2}\right)! \, 2^{(k-1)/2}} = \frac{(k-1)!}{[\left(\frac{k-1}{2}\right)!]^2 2^{k-1}},$$

and using the Stirling approximation for $k \to \infty$. We obtain

$$\frac{(k-1)!}{[\left(\frac{k-1}{2}\right)!]^2 2^{k-1}} \sim \sqrt{\frac{2}{\pi}} (k-1)^{-1/2}.$$

The Stirling approximation, however, yields meaningful results only for k large. Hence, we split the series in (29) into two parts. On the right-hand side, the

first part, for k from 1 to some fixed $\overline{k}$, will give a small contribution for t large, indeed of order of $O(e^{-ct})$. The remaining part, being a sum over values of $k \geq \overline{k}$, with $\overline{k}$ sufficiently large, can be estimated by a term of the order of

$$\sum_{k=\overline{k}}^{\infty} kP(k) \sim \sqrt{\frac{2}{\pi}}\, e^{-ct} \sum_{k=\overline{k}}^{\infty} k\,(k-1)^{-1/2}\,\alpha^{(k-1)/2}.$$

where we set $\alpha := 1 - e^{-2ct}$. An estimate of t_b can then be obtained by a term of order of

$$\sum_{r=0}^{\infty} r^{1/2}\,\alpha^r,$$

hence of order of

$$\int_0^{+\infty} x^{1/2}\,\alpha^x\,dx = \Gamma(3/2)\,|\log\alpha|^{-3/2}.$$

We have finally, for large t,

$$t_b = O\left(Nt_c\frac{t}{\overline{\Delta t_s}}e^{-ct}|\log(1-e^{-2ct})|^{-3/2}\right) = O\left(Nt_c\frac{t}{\overline{\Delta t_s}}e^{2ct}\right) \tag{32}$$

In the general case, of $m \in \mathbf{N}$, $m > 3$, an estimate can be obtained as follows. The multifactorial term in (25) is first written as

$$N_D(N_e) = \exp\left\{\sum_{i=0}^{N_e-1} \log\left[(m-1)i+1\right]\right\},$$

and the series here can be estimated by a term of the order of

$$\int_0^{N_e-1} \log\left[(m-1)x+1\right]\,dx = \frac{1}{m-1}\,\left[u\log u - u\right]_1^U, \quad U := (m-1)(N_e-1)+1,$$

and hence, for large k (or N_e),

$$N_D(N_e) = O\left(U^{U/(m-1)}e^{-N_e}\right), \quad U \sim (m-1)N_e.$$

Then, using the Stirling approximation, $kP(k)$ will be of order of

$$k\,U^{U/(m-1)}\,\frac{e^{-N_e}\,\alpha^{N_e}\,e^{-ct}}{N_e!(m-1)^{N_e}} \sim k\,\left[(m-1)N_e\right]^{N_e}\,\frac{e^{-N_e}\,\alpha^{N_e}\,e^{-ct}}{N_e^{1/2}N_e^{N_e}\left(\frac{m-1}{e}\right)^{N_e}},$$

where now we set $\alpha := 1 - e^{-c(m-1)t}$. Simplifying the previous expression, $kP(k)$, being $N_e \sim k/(m-1)$ turns out to be of order of $k^{1/2}(\alpha^{1/(m-1)})^k$, and hence, proceeding as before, of order of $|\log\left(1-e^{-c(m-1)t}\right)|^{-3/2}$. We conclude that now

$$t_b = O\left(Nt_c\,\frac{t}{\overline{\Delta t_s}}\,e^{\frac{3m-5}{2}ct}\right). \tag{33}$$

In Fig. 9, a comparison between the analytical results obtained for $m = 2$ and $m = 3$ in (32) and the measured computational time is shown as function of the final time, t in log scale for the y-axis. The latter was fitted, for convenience, to a linear function to compare it with the theoretical estimate. Note the good agreement with the theoretical results.

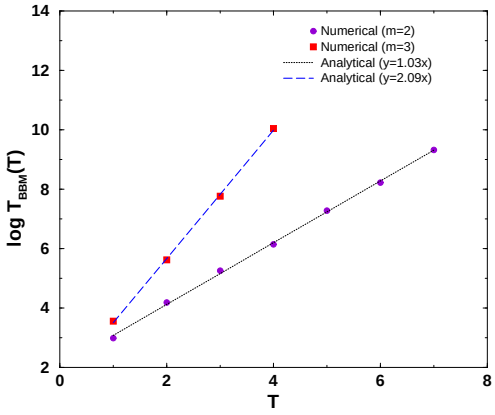

Fig. 9. Comparison between the computational time spent in solving the KPP equation at a single point $x = 0$, and the estimated computational time obtained theoretically in (30), and (32), respectively. This has been done for $m = 2$, and $m = 3$. Other parameters are $N = 10^6$ and $c = 1$

From the estimates in (32), we may expect a dramatic growth in the computational time, when t is large. Numerical simulations showed, however, that in practice the contribution to the solutions coming from trees with a large number of branches is negligible. Hence, the trees can be "pruned" in a suitable way, without losing much accuracy. This issue can be substantiated, but will not be addressed in this paper, see [5].

3.3 Numerical examples

Example 3. An IV problem problem for the KPP equation. Consider the problem

$$u_t = D\,u_{xx} - u + u^2, \quad u(x,0) = 1 - \frac{1}{\left(1 + \exp\frac{x}{\sqrt{6\,D}}\right)^2} \tag{34}$$

on the line. This problem is known [8] to possess the traveling wave solution

$$u(x,t) = 1 - \frac{1}{\left(1 + \exp\frac{\frac{x}{\sqrt{D}} - \frac{5}{\sqrt{6}}t}{\sqrt{6}}\right)^2}. \tag{35}$$

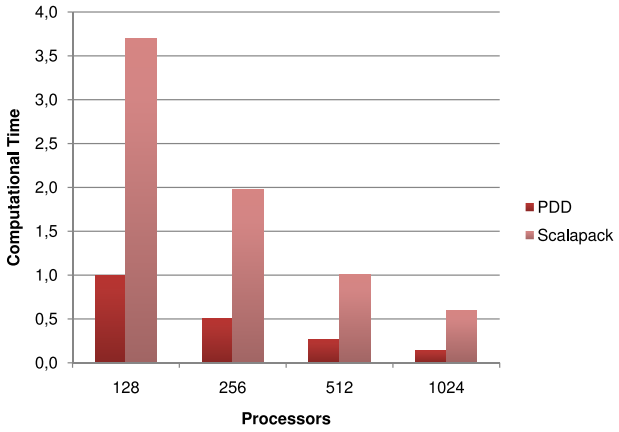

Fig. 10. Example 3: Comparison of the computational times (being $t_0 = 1,548$ seconds) for both methods, PDD and SCALAPACK, for a different number of processors. Here $D = 1$, and other parameters as in Example 1.

The results obtained in this example (see Fig. 10), are even more impressive than those of Example 1, which fact is intriguing because the present case refers to a nonlinear problem. The PDD method is now more involved, since it requires generating realizations of random *trees* rather than of realizations of stochastic processes.

However, it turns out that when we evaluate numerically functionals of branching process, the fluctuations around the mean are often non-gaussian (see [17] , e.g.). Hence, a large deviation analysis is required to assess the reliability of the numerical results. It was shown in [17] by the authors that, typically, a sample size of order 10^6 was enough to guarantee reliability, in their rather complex problem. We found such a size sufficient also in our case, where we were able to check the actual numerical error, comparing with the analytical solution.

It is noteworthy that, in the present example, as well as in many other nonlinear PDE initial value problems with constant coefficients, the probabilistic part of the algorithm can be sped up. In this case, the representation (15) becomes

$$u(x,t) = E\left[\prod_{i=1}^{k_t(\omega)} f(x + x_i'(t,\omega))\right], \tag{36}$$

where $x_i'(t,\omega)$ is the position of the ith stochastic branch reaching the final time, t, and belonging to the tree with root at $x = 0$. This means that only a single family of random trees, all starting from the same root, have to been considered, and then we can construct the solution by simply translating arguments. This procedure increases enormously the efficiency of the algorithm.

Example 4. A BV problem problem for the KPP equation.

Consider the same equation and initial value in (34), but for $-L \leq x \leq L$, with BCs obtained merely setting $x = \pm L$ in (35). In this and in the following example, the previous strategy (of only computing one family of random trees), cannot be implemented. The overall performance of the algorithm, however, is still by far superior to that achieved using SCALAPACK. In Fig. 11 a comparison is made between the computational times for both, PDD and SCALAPACK.

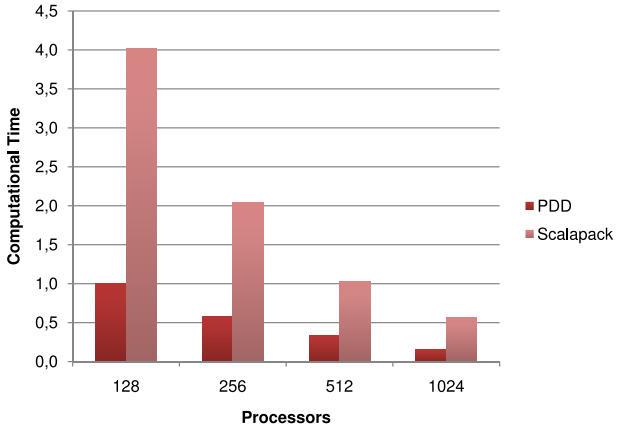

Fig. 11. Example 4: Comparison of the computational times (measured in units of $t_0 = 3,179$ seconds) for both methods, PDD and SCALAPACK, for different number of processors. Here $D = 1$, and other parameters as in Example 2.

To illustrate how the Monte-Carlo part of the PDD algorithm behaves for arbitrarily small diffusion coefficients, D, the error made solving numerically the BV problem for the KPP equation in Example 4, at a single point, is plotted in Fig. 12, for several values of D. Note that the numerical error remains almost constant when D increases. It is well known from the literature that the Monte Carlo methods can be even useful for solving degenerate diffusion equations, see [24], e.g. In contrast, the deterministic methods for solving parabolic problems are strongly affected by the value of the diffusion coefficient, the numerical error being likely larger for smaller values of D. Indeed, this is clearly shown in Fig. 13. Here the maximum numerical error made in solving Example 4 with $L = 10$ and $t = 0.5$ is plotted for several different values of D. Therefore, in order to keep the numerical error about constant and reasonably small, the number of mesh-points should increase accordingly, hence increasing the overall computational time. Such a behavior contrasts with that observed using the Monte Carlo method, where the computational time remains almost constant. This is because no further changes in the discretization parameter concerning sample size and time discretization are required, being the numerical error about constant for smaller values of the diffusion coefficients. Since the PDD method combines both, a Monte Carlo method for solving the PDE on the interfaces, and a deterministic local solver for each subdomain, the overall computational time of the algorithm should increase whenever the diffusion coefficient decreases.

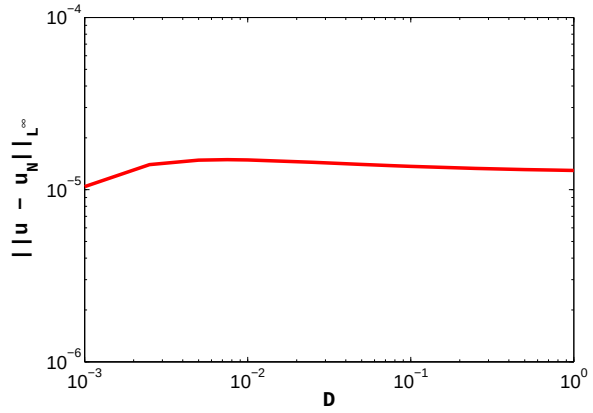

Fig. 12. Numerical error made in computing probabilistically the solution $u(x,t)$ of the BV problem in Example 4 for different values of the diffusion coefficient, D. Here $x = 0.1$, $t = 0.5$, $L = 10$; the other parameters are as in Example 2.

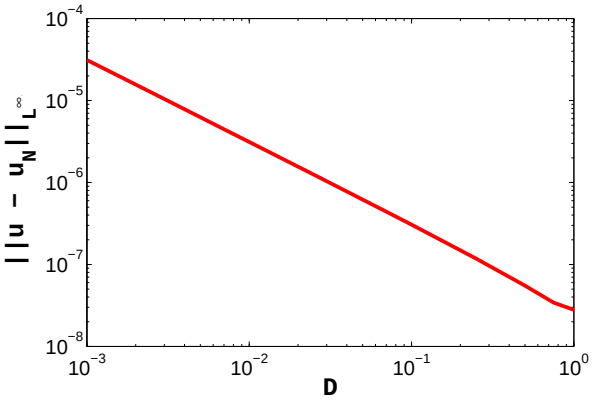

Fig. 13. Maximum numerical error made in solving with LAPACK the BV problem in Example 4 for different values of the diffusion coefficient, D. Here $t = 0.5$, $L = 10$; the other parameters are as in Example 2.

The next examples concern a nonlinear parabolic PDE more general than the KPP.

Example 5. An IV problem and a Dirichlet BV problem for a nonlinear parabolic equation with variable coefficients.

Consider the problem

$$u_t = (x^2+1)u_{xx} + [2+\sin(x)]u_x - u + \frac{1}{2}u^2 + \frac{1}{2}u^3,$$

$$u(x,0) = 1 \text{ for } 0 \le x < 1, \quad \text{and } u(x,0) \equiv 0 \text{ elsewhere on the line,}$$

as well for the Dirichlet BV problem for the same equation and initial value, but taken for $-L \le x \le L$, and BCs

$$u(-L,t) = u(L,t) = 0.$$

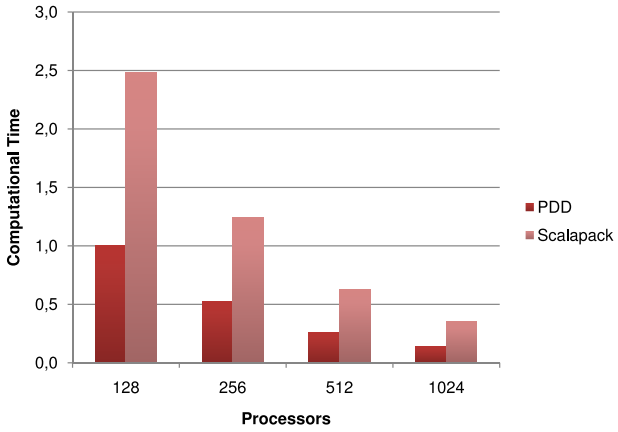

Fig. 14. Example 5 (IV problem): Comparison of the computational times (measured in units of $t_0 = 1,442$ seconds) for both methods, PDD and SCALAPACK, for different numbers of processors.

The analytical solutions to such problems is not known. A comparison was made with the numerical solution obtained by finite difference implicit methods, runned with very fine space-time grids.

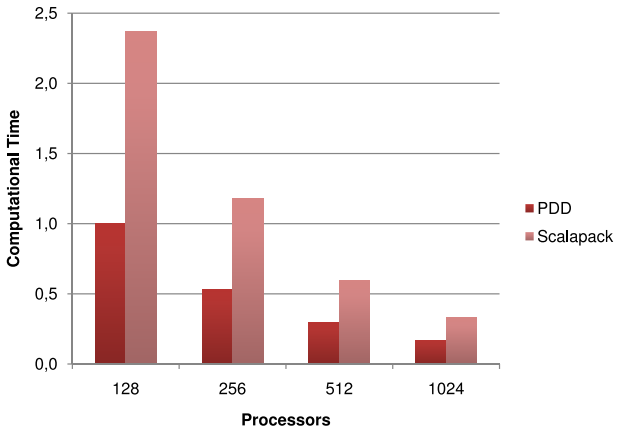

Fig. 15. Example 5 (BV problem): Comparison of the computational times (measured in units of $t_0 = 2,927$ seconds) for both methods, PDD and SCALAPACK, for different numbers of processors.

In Fig. 14, and Fig. 15, computational times corresponding to the IV and BV problems, respectively, are plotted. As in the previous examples, the PDD method clearly wins over SCALAPACK.

4 Summary

An efficient probabilistic method for solving numerically *nonlinear* one-dimensional parabolic problems, has been developed. This has been done for both, initial value and initial-boundary problems (with Dirichlet data). The technique is

based on computing averages on suitably generated branching stochastic processes, which generalize the stochastic processes used in the linear problems. Others than the traditional deterministic methods, the probabilistic representations allow for computing the solution at single points, internal to the domain, without solving the full problem. This remarkable feature has been exploited to accomplish a *domain decomposition* of the space-time domain, in such a way that the numerical solution can be assigned to an arbitrary number of independent processors, on massively parallel supercomputers, without any intercommunication overhead. In fact, the algorithm turns out to be fully *scalable* and naturally *fault tolerant*. Large-scale simulations on up to $1,024$ processors have been carried out to confirm the high performance of the algorithm in practice. A comparison has been done with other efficient deterministic parallel algorithms.

Acknowledgments

This work was supported, in part, by the Italy-Spain "Integrated Actions", the Portuguese FCT, and the Italian GNFM-INdAM. The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the Barcelona Supercomputing Center - Centro Nacional the Supercomputación.

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, 440-457 (2005)

[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, 421-438 (2005)

[3] Acebrón, J.A., and Spigler, R.: Supercomputing applications to the numerical modeling of industrial and applied mathematics problems. J. Supercomputing, **40**, 67-80 (2007)

[4] Acebrón, J.A., and Spigler, R.: A fully scalable parallel algorithm for solving elliptic partial differential equations. Lect. Notes in Comput. Sci. **4641** 727-736, (2007)

[5] Acebrón, J.A., Rodríguez-Rozas, A., and Spigler, R.: Domain decomposition solution of nonlinear two-dimensional parabolic problems by random trees. In preparation (2009)

[6] Arnold, L.: Stochastic Differential Equations: Theory and Applications. Wiley, New York (1974)

[7] Blömker, D., Romito M., and Tribe, R.: A probabilistic representation for the solutions to some non-linear PDEs using pruned branching trees. Ann. Inst. H. Poincaré Probab. Statist. **43**, 175-192 (2007)

[8] Brazhnik, P.K., and Tyson, J.J.: On travelling wave solutions of Fishers equation in two spatial dimensions. SIAM J. Appl. Math. **60** 371-391 (1999)

[9] Buchmann, F.M.: Simulation of stopped diffusions, J. Comput. Phys. **202** 446-462 (2005)

[10] Caflisch, R. E.: Monte Carlo and quasi-Monte Carlo methods. Acta Numerica (1998), 1-49 [Cambridge University Press, Cambridge, 1998].

[11] Chan, Tony F., and Mathew, Tarek P.: Domain decomposition algorithms. Acta Numerica (1994), 61-143 [Cambridge University Press, Cambridge, 1994].

[12] Chauvin, B and Rouault, A.: A stochastic simulation for solving scalar reaction-diffusion equations, Adv. Appl. Prob. **22** 88-100 (1990)

[13] Chorin, A.J.: Numerical study of slightly viscous flow. J. Fluid Mech., **57** 785-796 (1973).

[14] Dongarra, J., Fox, G., Kennedy, K., Torczon, L., Gropp, W., Foster, I., and White, A.: The Sourcebook of Parallel Computing, Morgan Kaufmann (2002)

[15] DuChateau, P. and Zachmann, D.: Applied Partial Differential Equations. Dover Publications (2002).

[16] Freidlin, M.: Functional Integration and Partial Differential Equations. Annals of Mathematics Studies no. 109, Princeton Univ. Press, Princeton (1985)

[17] Floriani, E., Lima, R., and Vilela Mendes, R.: Poisson-Vlasov: Stochastic representation and numerical codes. Eur. Phys. Journal D, **46**, 295–302 (2008)

[18] Geist, G.A.: Progress towards Petascale VirtualMachines. In Euro PVM/MPI 2003, J. Dongarra, D. Laforenza, S. Orlando (Eds.), Lecture Notes in Computer Science, pp. 10-14, Springer, Berlin (2003)

[19] Jansons, K.M., and Lythe, G.D.: Efficient numerical solution of stochastic differential equations using exponential timestepping. J. Statist. Phys., **100**, Nos.5/6 pp. 1097–1109 (2000)

[20] Jansons, K.M., and Lythe, G.D.: Exponential timestepping with boundary test for stochastic differential equations. SIAM J. Sci. Comput., **24** 1809–1822 (2003)

[21] Kalos, M.H., and Withlock, P.A.: Monte Carlo Methods, Vol. I: Basics. Wiley, New York (1986)

[22] Karatzas, I., and Shreve, S.E.: Brownian Motion and Stochastic Calculus. 2nd ed., Springer, Berlin (1991)

[23] Kloeden, P.E., and Platen, E.: Numerical Solution of Stochastic Differential Equations. Springer, Berlin (1992)

[24] Lapeyre, B., Pardoux, E., and Sentis, R.: Introduction to Monte-Carlo methods for transport and diffusion equations. Oxford Univ. Press, (2003).

[25] Le Jan, Y., and Sznitman, A.S.: Stochastic cascades and 3-dimensional Navier-Stokes equations. Prob. Theory and Relat. Fields., **109** 343-336 (1997)

[26] McKean, H.P.: Application of brownian motion to the equation of Kolmogorov-Petrovskii-Piskunov. Comm. on Pure and Appl. Math., **28** 323-331 (1975)

[27] Milstein, G.N., and Tretyakov, M.V.: Stochastic Numerics for Mathematical Physics. Springer (2004)

[28] Petersen, W., and Arbenz, P.: Introduction to parallel computing. A practical guide with examples in C. Oxford Univ. Press, (2004).

[29] Peirano, E., and Talay, D.: Domain decomposition by stochastic methods, Domain Decomposition Methods in Science and Engineering, Natl. Auton. Univ. Mex., México, 2003, pp. 131-147 (electronic) [Proc.s of the 14th Int. Conf. on Domain Decomposition Methods, Ed.s I. Herrera, D. E. Keyes, O. B. Widlund, and R. Yates, Cocoyoc, México, 2002].

[30] Peskin, C.S.:A random-walk interpretation of the incompressible Navier-Stokes equation. Commum. Pure. Appl. Math. **38** 845-852 (1985).

[31] Quarteroni, A., and Valli, A.: Domain Decomposition Methods for Partial Differential Equations. Oxford Science Publications, Clarendon Press, Oxford (1999)

[32] Ramirez, J.M.: Multiplicative cascades applied to PDEs (two numerical examples). J. Comput. Phys., **214** 122-136 (2006)

[33] Regnier, H., and Talay, D.: Vitesse de convergence d'une méthode particulaire stochastique avec branchements: C.R. Acad. Sci. Paris Sér. I Math. **332** 933-938 (2001).

[34] Regnier, H., and Talay, D.: Convergence rate of the Sherman and Peskin branching stochastic particle method. Proc. Royal Soc. Lond. Ser. A Math. Phys. Eng. Sci. **460** 199-220 (2004).

[35] Sherman, A.S., and Peskin C.S.: A Monte Carlo method for scalar reaction diffusion equations. SIAM J. Sci. Stat. Comput., **7** 1360-1372 (1986).

[36] Sherman, A. and Mascagni, M.: A gradient random walk method fr two-dimensional reaction-diffusion equations. SIAM J. Sci. Comput., **15** 1280-1293 (1994).

[37] Talay, D.: Probabilistic Numerical Methods for Partial Differential Equations: Elements of Analysis. In: D. Talay and L. Tubaro (Eds.), Probabilistic Models for Nonlinear Partial Differential Equations, Lecture Notes in Mathematics **1627**, 48-196 (1996).

[38] Waymire, E.: Probability and incompressible Navier-Stokes equations: An overview of some recent developments. Prob. Surveys, **2** 1-32 (2005)