Probabilistically-induced domain decomposition methods for elliptic boundary-value problems

Juan A. Acebrón[a], Maria Pia Busico[b], Piero Lanucara[b], and Renato Spigler[c]

[a] *Departamento de Automática, Escuela Politécnica, Universidad de Alcalá, Crta. Madrid-Barcelona km. 31.600, 28871 Alcalá de Henares, Spain*

[b] *CASPUR, Via dei Tizii 6 b, 00185 Rome*

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

Abstract

Monte Carlo as well as *quasi-Monte Carlo* methods are used to generate only few interfacial values in two-dimensional domains where boundary-value elliptic problems are formulated. This allows for a *domain decomposition* of the domain. A continuous approximation of the solution is obtained interpolating on such interfaces, and then used as boundary data to split the original problem into *fully decoupled* subproblems. The numerical treatment can then be continued, implementing any deterministic algorithm on each subdomain. Both, Monte Carlo (or quasi-Monte Carlo) simulations and the domain decomposition strategy allow for exploiting *parallel* architectures. *Scalability* and natural *fault tolerance* are peculiarities of the present algorithm. Examples concern Helmholtz and Poisson equations, whose probabilistic treatment presents additional complications with respect to the case of homogeneous elliptic problems without any potential term and source.

Key words: Monte Carlo methods, quasi-Monte Carlo methods, domain decomposition, parallel computing, fault-tolerant algorithms
PACS: 65C05, 65C30, 65N55

Email addresses: `juan.acebron@uah.es` (Juan A. Acebrón), `MariaPia.Busico@caspur.it` (Maria Pia Busico), `lanucara@caspur.it` (Piero Lanucara), `spigler@mat.uniroma3.it` (Renato Spigler).

1 Introduction

It is well known that the solution to boundary-value problems for certain linear partial differential equations admits a probabilistic representation, and that this can be taken, in principle, as a basis for computation. Consider the elliptic boundary value problem

$$Lu - c(x)u = f(x) \quad \text{in } \Omega, \qquad u|_{\partial\Omega} = g, \tag{1}$$

where $\Omega \subset \mathbf{R}^d$, and L denotes a linear elliptic operator, say $L = a_{ij}(x)\partial_i\partial_j + b_i(x)\partial_i$ (using the summation convention), with continuous bounded coefficients, $c(x) \geq 0$ and bounded continuous, continuous boundary data g, source term f in $L^2(\Omega)$, and $\partial\Omega$ Lipschitz continuous. The probabilistic representation of the solution is given by

$$u(x) = E_x^L \left[g(\beta(\tau_{\partial\Omega}))e^{-\int_0^{\tau_{\partial\Omega}} c(\beta(s))\,ds} - \int\limits_0^{\tau_{\partial\Omega}} f(\beta(t))\, e^{-\int_0^t c(\beta(s))\,ds}\, dt \right], \tag{2}$$

see [1,2], e.g., where $\tau_{\partial\Omega}$ is the first exit (or hitting) time of the path $\beta(\cdot)$ started at x when $\partial\Omega$ is crossed. $\beta(\cdot)$ is the stochastic process 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 d-dimensional Brownian motion, and the measure reduces to the Gaussian measure. In general, the process $\beta(\cdot)$ is the solution to a stochastic differential equation (SDE) of the Ito type related to the elliptic partial differential equation in (1), namely

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

Here $W(t)$ represents d-dimensional standard Brownian motion (also called Wiener process); see [2,3], e.g., for generalities, and [4–6] for the related numerical treatment. As is known, the solution to (3) is a stochastic process, $\beta(t,\omega)$, where ω, which usually is not indicated explicitly in probability theory, denotes the “chance variable”, ranging on a suitable abstract probability space. The drift, b, and the diffusion, σ, in (3), are related to the coefficients of the elliptic operator in (1) by $b = (b_i)^T$, and $\sigma\sigma^T = a$, with $\sigma = (\sigma_{ij})$, $a = (a_{ij})$.

We shall confine ourselves to 2-dimensional problems, hence we shall write (x, y) instead of the vector variable x in all examples in Section 4 below.

The Monte Carlo approach, based on the numerical computation of the solution to problem (1) through the representation formula in (2), is considered very inefficient, at least in low dimension. It can be viewed as the last resource to be exploited, for instance when the boundary of the domain has a complicated geometry, which fact rules out the adoption of any other deterministic algorithm [7]. In fact, one of the advantages of Monte Carlo methods is that they do not require any structured grid.

In the companion paper [8], we have proposed a domain decomposition approach for the numerical treatment of rather general linear elliptic boundary-value problems. The idea was to compute *only few interfacial* values inside the domain, Ω, from formula (2), and interpolate on the points where the values above have been obtained. These nodes are viewed as located on suitable interfaces inside the domain, irrespective of the fact that such interfaces are physical or not. Then, a continuous approximation of the trace of solutions can be constructed, and this will be used as boundary data for the subdomains. Full decoupling into as many subdomains as we wish can be realized in this way. The key idea of using a probabilistic representation of solutions to elliptic problems *only* to accomplish a preliminary domain decomposition, was first proposed in [9], and later in [10].

In [8], however, only homogeneous elliptic equations without potential terms and sources were considered in the numerical examples. In fact, the representation formula in (2) holds even in the more general case of equation (1). In this paper, we cope with the new difficulties which may arise from the presence of potential terms, like $c(x)u$, as well as of source terms, like $f(x)$. While the basic machinery developed here is the same as there, such new terms require estimating first exit times besides computing first exit points, as well as using the entire paths due to the quadrature.

This method can be called a "probabilistic domain decomposition"(PDD) method. We stress that it can fully exploit *parallel architectures*. In fact, (*i*) it implements a domain decomposition algorithm; (*ii*) *every* realization (or path) of the stochastic process starting at every point can be simulated independently (if we generate N sample paths at m points, we can use up to mN independent processors). Such a degree of parallelization is compatible with the use of possibly different and even geographically distant processors (grid computing, heterogeneous computing) and/or clusters of them. This algorithm is also naturally *fault tolerant*, a property whose demand is becoming increasingly important, in view of machines working in the petaflop regime, equipped with hundreds of thousands or millions of processors [11].

A remarkable improvement of the the performance of the classical Monte Carlo method [12], which is based on the so-called *pseudorandom* numbers (which mimic the ideal random numbers), can be achieved using, instead, sequences of *quasi-random* numbers [13–16]. The corresponding strategy is called quasi-Monte Carlo, and when using such sequences in our approach, the method will be called a “quasi-probabilistic domain decomposition” (quasi-PDD) method. Such sequences are actually deterministic and their elements are uniformly distributed. Unfortunately, they are subject to a certain degree of correlation. Sequences of quasi-random numbers have been applied in the past with some success to the numerical evaluation of high-dimensional integrals [17], in particular in problems of financial mathematics [18]. Other applications have been made to the generation of quasi-random paths of stochastic processes in [19,20], to the Boltzmann equation [21], and to a simple system of diffusion equations in $\mathbf{R}^d$ [22]. A failure in applying them to the solution of stochastic differential equations has been pointed out in [23], but it has been shown in [24] that a careful implementation allows for a successful use of them. Indeed, a kind of scrambling of the quasi-random numbers at each time step, namely a suitable reordering strategy, has been proved to be effective.

In Section 2, some generalities are discussed, while in Section 3 various sources of numerical error which affect the PDD and the quasi-PDD methods are pointed out. Numerical examples are shown in Section 4, where the efficiency of the PDD and of quasi-PDD algorithms is illustrated. In the final section we summarize the high points of the paper.

2 Deterministic versus probabilistic domain decomposition

Solving a boundary-value problem, for instance a Dirichlet problem, for a given elliptic partial differential equation on a given domain, Ω, by domain decomposition, consists of dividing Ω into a number of subdomains and computing then the solution on each of such subdomains on separate processors, see [25,26], e.g. However, the boundary-value problems above are global in nature, so that the solution on the interfaces internal to Ω, that one would like to use as boundary data for the sub-problems, cannot be computed in advance, before solving the full problem.

Approximations of interfacial values are constructed imposing continuity of solutions and of certain derivatives across the interfaces. This requires solving linear algebraic subsystems by iterative processes, typically characterized by high condition numbers [26], especially in methods without overlap (iterative substructuring methods).

It is claimed that using parallel computers with a few hundred processors and with $10^6 - 10^7$ variables, the global cost of the method is dominated by that spent by local solvers. The future generation of parallel machines however are designed with hundreds of thousands or even millions of processors, in which case intercommunication across the numerous subdomains might dramatically affect the overall performance. Full decoupling of the original problem into an arbitrary number of subdomains is therefore highly desirable, and this can be achieved by a probabilistic domain decomposition method.

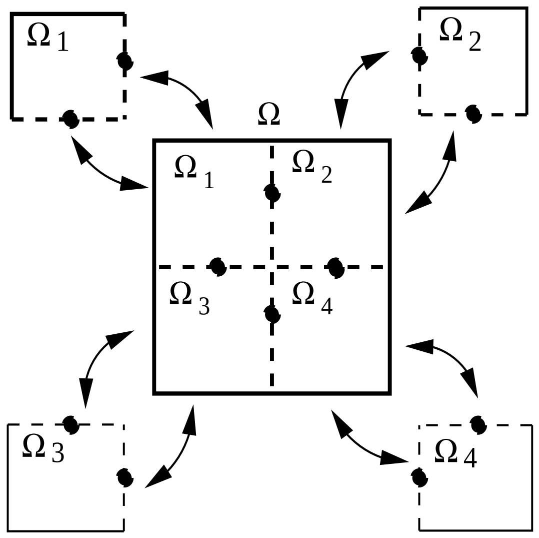

Fig. 1. Sketchy diagram illustrating the numerical method, splitting the initial domain Ω into four subdomains, $\Omega_1, \Omega_2, \Omega_3, \Omega_4$.

By means of a *probabilistic* representation of solutions, numerical approximations can be generated at every point inside Ω, *without* solving the full problem. The basic idea is to compute only a few values of the solution on certain chosen interfaces, and then interpolate to get continuous approximations. These can be used as boundary values to decouple the problem into sub-problems, see Fig. 1. Each of such sub-problems can then be solved independently on a separate processor. Clearly, neither communication among the processors, nor iteration across the interfaces is needed.

It should be noticed that even though a rather poor approximation can be expected on the interfaces due to the Monte Carlo method, numerical errors inside each subdomains become smaller due to the discrete maximum principle.

In closing this section, we stress that the PDD algorithm is characterized by a high degree of parallelism, because it combines the main advantage of the domain decomposition strategy with the inherent parallelism of the Monte Carlo method. The PDD algorithm can be shown to be characterized by a speed-up $S_p = T_1/T_p \approx (T_1/2kT_{MC})\sqrt{p}$, as $p \to \infty$. Here T_1 is the time spent for solving sequentially problem (1) on the entire domain, T_p is the time required to solve it in parallel with p processors,

k is the number of points on each interfaces, and T_{MC} is the time required for computing by the Monte Carlo simulation a single interfacial value. This result has been established in [8], where numerical examples simpler than here were presented, but also holds in the present case. In addition, the PDD algorithm is scalable, and also enjoys the intriguing feature of being naturally "fault tolerant" [11]. In fact, clearly, a failure of a small percentage of processors in the course of the Monte Carlo computation of the nodes would not force the entire code to stop, but only a modest additional error would be produced. On the other hand, if some processors fail when the local solvers are being runned to compute the solution on the various subdomains, the output will be incomplete but correct. The only missing results will be those corresponding to the processors which have failed.

3 The PDD algorithm and the various sources of numerical error

The PDD algorithm that has been developed in [8] as well as in this paper, is a hybrid numerical method in that, it consists of both, probabilistic and deterministic parts. Consequently it is affected by several kinds of errors, some of which are statistical in nature. The core of the algorithm is given by numerical approximation of a few internal values to be thought of as pivots for interpolation. Interpolation and subsequent solution in the subdomains by deterministic solvers are standard ingredients, and we shall comment later on such issues. Concentrating on the numerical evaluation of the nodal values, we observe that a number of numerical errors can be singled out. First of all, expected values to be computed on the basis of the representation formula in Eq. (2) have to be replaced by arithmetic averages. This yields the usually dominating error made in Monte Carlo simulations, which is of a statistical nature and of order $N^{-1/2}$, N being the sample size. Using quasi-random number sequences, instead of pseudorandom sequences, this error is deterministic and of order $N^{-1} \log^{d^*-1} N$, d^* denoting the "effective" space dimension. Concerning the dependence of such an error on dimension, it is important to emphasize that the effective space dimension, d^*, can be reduced dramatically in practice [24]. A straightforward implementation requires that as many independent quasi-random numbers as the number of steps needed to exit the boundary, times the geometric dimension, is used, and this is d^*. Clearly, a large value of d^* appreciably increases the error, unless N is taken very large, destroying the advantage of using quasi-random numbers instead of pseudorandom numbers. However, a more favorable alternative does exist. In this paper, as well as, e.g., in [8,20,22,24], the value d^* can be replaced by the geometric dimension, d. In fact, only d independent sequences of quasi-random numbers, one for each coordinate, can be used, provided that a reordering strategy is adopted to destroy the inherent correlations.

Moreover, approximating paths of the stochastic process, $\beta(t)$, has to be obtained solving numerically the SDE in Eq. (3). The truncation error involved here depends on the method we choose, for instance the Euler scheme, Taylor-based schemes, or the exponential timestepping scheme. Another source of errors that might dominate the overall error occurs in the evaluation of first exit points and times. In fact, the first exit time, $\tau_{\partial\Omega}$, as well as the related first exit point, $\beta(\tau_{\partial\Omega})$, appears explicitly in Eq. (2). It is worth noting that, other than in the case of homogeneous equations without potential and sources considered in [8], computing first exit times (in addition to first exit points) is now required. Furthermore, numerical quadrature now enters Eq. (2) again owing to the nonzero coefficients $c(x)$ and $f(x)$ in Eq. (1). Such novel aspects encountered in the present paper needs special care, and we may expect that they produce additional errors. In Fig. 2, the dependence of the numerical error on N is given in a logarithmic scale. Such a plot provides evidence of the possibility of controlling the dimensionaly issue.

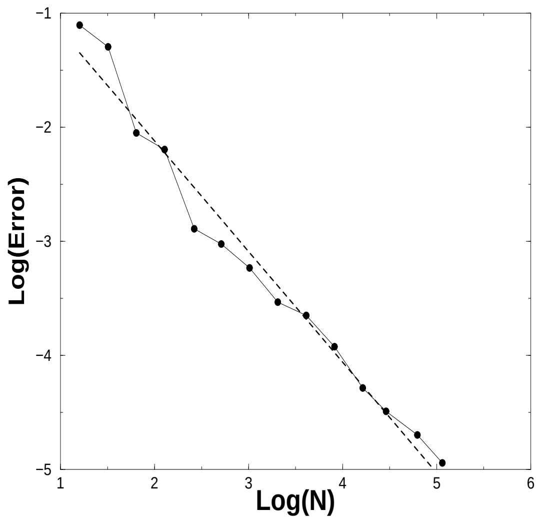

Fig. 2. Numerical error made solving Example A for several values of N at the point $(0.3, 0.5)$. The marked points denote the solution obtained by the quasi-Monte Carlo method, while the dashed line is obtained fitting the data by a linear regression method. The time step used is $\Delta t = 10^{-4}$.

3.1 The numerical errors in computing nodal values

The essential part of our method consists of obtaining numerical approximations of the solution to Eq. (1). In the numerical evaluation of the representation formula (2), set, for short, $\tau = \tau_{\partial\Omega}$, and

$$u(x) = v(x) + w(x), \tag{4}$$

$$\overline{f} = -f, \tag{5}$$

$$\psi(\beta(\cdot), t) = e^{-\int_0^t c(\beta(s))\, ds}. \tag{6}$$

Thus (2) becomes

$$u(x) = v(x) + w(x) = E_x^L[g(\beta(\tau))\,\psi(\beta(\cdot),\tau)] + E_x^L\left[\int_0^\tau \overline{f}(\beta(t))\,\psi(\beta(\cdot),t)\,dt\right]. \tag{7}$$

Moreover, we assume that $c(\cdot) \geq 0$ and that $c(\cdot)$, $g(\cdot)$, and $f(\cdot)$ are bounded and Lipschitz continuous, with Lipschitz constants L_c, L_g, and L_f, respectively. Note that $L_{\overline{f}} = L_f$.

Consider first the numerical approximation of $v(x)$. We want to approximate

$$E_x^L[g(\beta(\tau))\,\psi(\beta(\cdot),\tau)],$$

but, in practice, we shall compute instead

$$\frac{1}{N}\sum_{j=1}^{N} g(x_j(t_j))e^{-I[x_j(\cdot);t_j]},$$

where $x_j(\cdot)$ is an approximation to the jth path $\beta_j(\cdot)$, t_j an approximation to τ^j, and e^{-I} an approximation of the term ψ by means of a numerical quadrature formula. In fact, in practice we can only simulate a finite number, say N, of realizations of the paths $\beta(t) \equiv \beta(t,\omega)$, ω denoting the "chance variable", customarily omitted in all formulae. While the "label" ω runs on some abstract probability space, taking infinitely many values, we can only generate N paths, $\beta_j(t)$, $j = 1, 2, \ldots, N$. The same can be said concerning the first exit times, $\tau \equiv \tau(\omega)$, which we shall denote by τ^j, $j = 1, 2, \ldots, N$. Besides, both, the paths $\beta_j(t)$ and τ^j will be approximated numerically, and $x_j(t)$ and t_j, will denote, respectively, their approximations. The $x_j(t)$ are obtained integrating numerically the underlying SDEs, hence the discrepancy between $\beta_j(t)$ and $x_j(t)$ is the truncation error.

In order to estimate the overall error, say ε_N, made computing $v(x)$, we can write

$$\varepsilon_N = \varepsilon_N^{(1)} + \varepsilon_N^{(2)} + \varepsilon_N^{(3)} + \varepsilon_N^{(4)}, \tag{8}$$

where

1. $$\varepsilon_N^{(1)} = E_x^L[g(\beta(\tau))\,\psi(\beta(\cdot),\tau)] - \frac{1}{N}\sum_{j=1}^{N} g(\beta_j(\tau^j))\,\psi(\beta_j(\cdot),\tau^j); \tag{9}$$

2. $$|\varepsilon_N^{(2)}| \leq \frac{1}{N}\sum_{j=1}^{N} |g(\beta_j(\tau^j)) - g(x_j(\tau^j))|\, e^{-\int_0^{\tau^j} c(\beta_j(s))\,ds} + \frac{1}{N}\sum_{j=1}^{N} |g(x_j(\tau^j))| \left| e^{-\int_0^{\tau^j} c(\beta_j(s))\,ds} - e^{-\int_0^{\tau^j} c(x_j(s))}\, ds \right|$$

$$\leq \frac{1}{N} L_g \sum_j |\beta_j(\tau^j) - x_j(\tau^j)| + \max|g|\, L_c \frac{1}{N} \sum_j \int_0^{\tau^j} |\beta_j(s) - x_j(s)|\, ds$$

$$\equiv \frac{1}{N} L_g \sum_j |\varepsilon^j_{trunc.}(\tau^j)| + \max|g|\, L_c \frac{1}{N} \sum_j \int_0^{\tau^j} |\varepsilon^j_{trunc.}(s)|\, ds$$

$$\leq [L_g + L_c \max|g| \cdot \langle \tau^j \rangle] \cdot \varepsilon_{trunc.}, \tag{10}$$

where we set $\langle \tau^j \rangle = N^{-1} \sum_j \tau^j$, the average value of the exit times;

3.
$$|\varepsilon^{(3)}_N| \leq \frac{1}{N} \sum_{j=1}^N |g(x_j(\tau^j)) - g(x_j(t_j))|\, e^{-\int_0^{\tau^j} c(x_j(s))\, ds}$$

$$+ \frac{1}{N} \sum_{j=1}^N |g(x_j(t_j))| \left| e^{-\int_0^{\tau^j} c(x_j(s))\, ds} - e^{-\int_0^{t_j} c(x_j(s))\, ds} \right|$$

$$\leq L_g \frac{1}{N} \sum_{j=1}^N L_{x_j} |\tau^j - t_j| + \max|g| \cdot \max|c|\, \frac{1}{N} \sum_{j=1}^N |\tau^j - t_j|$$

$$\leq [L_g \max_j L_{x_j} + \max|g| \cdot \max|c|] \cdot \max_j |\tau^j - t_j|; \tag{11}$$

4.
$$|\varepsilon^{(4)}_N| \leq \frac{1}{N} \sum_{j=1}^N |g(x_j(t_j))| \left| e^{-\int_0^{t_j} c(x_j(s))\, ds} - e^{-I[c(x_j(\cdot)); t_j]} \right|$$

$$\leq \max|g|\, \frac{1}{N} \sum_{j=1}^N \left| \int_0^{t_j} c(x_j(s))\, ds - I[c(x_j(\cdot)); t_j] \right|$$

$$\leq \max|g| \cdot \varepsilon'_{quadr.}, \tag{12}$$

where $\varepsilon'_{quadr.}$ denotes the maximum error with respect to j, made in the numerical quadrature of $c(x_j(s))$.

Note that $\varepsilon^{(1)}_N = O(N^{-1/2})$ when classical Monte Carlo simulations are conducted, while it is $O(N^{-1} \log^{d^*-1} N)$ when quasi-Monte Carlo methods are implemented. It should be observed that no attempt of approximating continuous paths of the underlying stochastic processes is made using quasi-random numbers. In fact, the representation formula (2) can be interpreted, rather, as taking an average of a certain random *variable*. Conceptually, there is no difference with respect to the case when high-dimensional integrals are computed upon generation of quasi-random sequences, see [19], e.g. On the other hand, some authors did use quasi-random sequences to obtain approximate solutions to partial differential equations simulating quasi-random paths of continuous stochastic processes, as it was done here, see for instance [19,22,27,28], to quote just a few. In [27], it was shown that simulating quasi-random walks can be advantageous even to solve certain *nonlinear* quations. A serious problem can be found, instead, in the potentially high value of the effective dimension, d^*. In fact, in the present simulations, a (relatively) large number of discrete time-steps to be used to exit the boundary implies a correpondingly high dimension. On the one hand, this increases the exponent of the logarithmic factor in the numerical error, and, on the other hand, it produces unwanted correlations. A scrambling strategy, such as reordering, has been proven, however, to be effective in destroying such correlations. This has been done succesfully in the literature by a few authors, see [15,22,24], e.g. In [8,24], it was shown that a careful use of only two strings of quasi-random numbers suffices, hence $d^* = 2$ could be used there. No question like this arises obviously using pseudorandom numbers.

The second error, $\varepsilon_N^{(2)}$, is due to the truncation error made approximating numerically the path $\beta_j(t)$ by $x_j(t)$. It will be, e.g., of order of $O(\Delta t)$ when using a Euler scheme in the weak sense, see [5]. In Eq. (10), $\langle \tau^j \rangle$ is an approximation of the mean exit time, $E_x[\tau]$, whose value depends on the stochastic process and on the domain. For instance, for the Brownian motion exiting from a ball of radius r, $E_x[\tau]$ is known to be of order r^2 [2]. The third term, $\varepsilon_N^{(3)}$, is due, clearly, to the error made approximating τ^j with t_j. This error, which has been proven to be of order $\sqrt{\Delta t}$ for the Euler scheme, might dominate, [29,30]. Hence, a special care should be paid in the implementation to reduce it to order Δt.

The fourth term, $\varepsilon_N^{(4)}$, finally, is a new kind of error, which did not appear in the previous paper [8]. In fact, it comes from the numerical evaluation of the integrals in formula (2). Note that accomplishing the numerical quadrature at the very beginning, one should consider the quadrature of the function $c(\beta_j(t))$. In this case, even for c smooth, the approximation would be of order $\sqrt{\Delta t}$, due to the fact that the realizations $\beta_j(t)$ are merely Hölder continuous (with exponent 1/2). In the present approach, we face instead the numerical quadrature of $c(x_j(t))$, which is Lipschitz continuous, $x_j(t)$ being piecewise linear. Therefore, an accuracy of order Δt can be achieved.

Concerning the evaluation of $w(x)$, let write

$$E_x^L\left[\int_0^\tau \overline{f}(\beta(t))\,\psi(\beta(\cdot),t)\,dt\right] - \frac{1}{N}\sum_{j=1}^N J[\overline{f}(x_j(\cdot)), c(x_j(\cdot)); t_j]$$

$$= \eta_N^{(1)} + \eta_N^{(2)} + \eta_N^{(3)} + \eta_N^{(4)}, \tag{13}$$

where

1. $$\eta_N^{(1)} = E_x^L \left[\int_0^{\tau} \overline{f}(\beta(t))\, \psi(\beta(\cdot), t)\, dt \right] - \frac{1}{N} \sum_{j=1}^{N} \int_0^{\tau^j} \overline{f}(\beta_j(t))\, \psi(\beta_j(\cdot), t)\, dt, \tag{14}$$

   which is of order of $N^{-1/2}$ whenever classical Monte Carlo simulations are conducted, and of order of $N^{-1} \log^{d^*-1} N$ using quasi-Monte Carlo methods;

2. $$\begin{aligned} |\eta_N^{(2)}| &\le \frac{1}{N} \sum_{j=1}^{N} \int_0^{\tau^j} \left| \overline{f}(\beta_j(t))\, \psi(\beta_j(\cdot), t) - \overline{f}(x_j(t))\, \psi(x_j(\cdot), t) \right| dt \\ &\le \frac{1}{N} \sum_{j=1}^{N} \int_0^{\tau^j} |\overline{f}(\beta_j(t)) - \overline{f}(x_j(t))| \cdot |\psi(\beta_j(\cdot), t)|\, dt \\ &+ \frac{1}{N} \sum_{j=1}^{N} \int_0^{\tau^j} |\overline{f}(x_j(t))| \cdot |\psi(\beta_j(\cdot), t) - \psi(x_j(\cdot), t)|\, dt \\ &\le L_f\, \frac{1}{N} \sum_{j=1}^{N} \int_0^{\tau^j} |\beta_j(t) - x_j(t)|\, dt \\ &+ \max |f|\, \frac{1}{N} \sum_{j=1}^{N} \int_0^{\tau^j} dt \int_0^{t} |\beta_j(s) - x_j(s)|\, ds \\ &\le [L_f + \max |f|] \langle \tau^j \rangle \cdot \max_{j,s} |\beta_j(s) - x_j(s)|, \end{aligned} \tag{15}$$

   which is of the order of the truncation error, say $\varepsilon_{trunc.}$; note that above $0 \le \psi(\cdot,\cdot) \le 1$ for $c(\cdot) \ge 0$, and that $L_{\overline{f}} = L_f$ and f may replace $\overline{f}$ everywhere;

3. $$\begin{aligned} |\eta_N^{(3)}| &\le \frac{1}{N} \sum_{j=1}^{N} \left| \int_{t_j}^{\tau^j} |f(x_j(t))|\, e^{-\int_0^t c(x_j(s))\, ds}\, dt \right| \\ &\le \max |f| \cdot \max_j |\tau^j - t_j|, \end{aligned} \tag{16}$$

   which is of the order of ε_τ, the error made approximating the first exit times;

4. $$\begin{aligned} |\eta_N^{(4)}| &\le \frac{1}{N} \sum_{j=1}^{N} \int_0^{t_j} |f(x_j(t))| \cdot \left| e^{-\int_0^t c(x_j(s))\, ds} - e^{-I[c(x_j(\cdot));t]} \right| dt \\ &+ \frac{1}{N} \sum_{j=1}^{N} \left| \int_0^{t_j} \overline{f}(x_j(t))\, e^{-I[c(x_j(\cdot)),t]}\, dt - J[\overline{f}(x_j(\cdot)), c(x_j(\cdot)); t_j] \right| \\ &\le \max |f| \cdot \max_j |t_j| \cdot \varepsilon'_{quadr.} + \varepsilon''_{quadr.}, \end{aligned} \tag{17}$$

   where $\varepsilon''_{quadr.}$ denotes the maximum error with respect to j made in the numerical quadrature of $\overline{f}(x_j(t))\, e^{-J[\overline{f}(x_j(\cdot)),c(x_j(\cdot));t_j]}$.

Here the same comments hold regarding the order of each type of error. The importance of all such errors will be illustrated within the numerical examples presented in section 4.

Other Monte Carlo-based numerical methods to solve elliptic boundary-value problems, which do not imply solving SDEs, thus avoiding all kinds of errors described above, exist. For instance, the so-called "walking on the spheres" approach [34,35] belongs to this category. This method, however, which essentially requires evaluating Green's functions seems to be difficult to apply at least when a variable diffusion coefficient enters the elliptic equation. Moreover, other sources of numerical error beset this type of algorithms, e.g. the approximation of the boundary by the so-called ε-strip, and the fact that constant diffusions allow for using merely random walks instead of continuous stochastic processes.

3.2 Interpolation on the internal nodes

The next step of the algorithm is interpolation on the nodes computed in the previous subsection. Such nodes can be thought of as laying on certain interfaces internal to the domain, Ω. Chebyshev interpolation has been chosen in view of its global quasi-optimality [31]. Moreover, interpolating a given C^k function by the nth degree Chebyshev polynomial of the first kind, the error is of order of n^{-k}. In addition, the errors affecting the nodal values themselves should be taken into account, because such values have been obtained by the Monte Carlo or quasi-Monte Carlo simulations. Due to the stability properties of the Chebyshev interpolation, such an error turns out to be under control. In particular the Lebesgue constant involved grows only logarithmically with n. In our algorithm, we only need, usually, $n = 2$ or 3 nodes on each interface. The numerical error made in the interpolation part of the algorithm will be plotted in one example in section 4 below.

3.3 Local solvers

The final step of the algorithm consists of solving a number of independent subproblems, one in each subdomain. In fact, the previous procedure allows for fully decoupling, and thus any deterministic algorithm can be implemented to solve such subproblems. Since the focus of the algorithm is not on the local solvers, we used the simplest method, i.e., finite differences (FDs). Jacobi iterations have then been conducted to solve the ensuing linear algebraic systems. The termination criterion was chosen according to the specific model example and to the number of subdomains.

4 Numerical examples

In this section, we present a few numerical examples to illustrate the probabilistically-induced domain decomposition algorithm developed in this paper. While the inherently high degree of parallelism allows for implementing an MPI code, we used the standard parallelization library called OpenMP, which is designed for shared memory parallel architectures. The numerical tests have been conducted using a 16 processor parallel machine, the IBM Power 3, with a peak performance of 24 GFLOPS. A comparison is made in all examples below with a parallel version of a standard finite difference solver (PFD).

As a general remark, note that both the Monte Carlo simulation based on pseudorandom numbers in the PDD method, and the solution based upon domain decomposition, allows for massively parallel computation. Unfortunately, one cannot exploit independently these two ingredients, the Monte Carlo simulations and the domain decomposition strategy, both well suited to parallel computing, to further increase the overall degree of parallization. In fact, an interpolation process must take place after the Monte Carlo generation of the pivotal values, before the computation on separate subdomains can start. In the quasi-PDD method, however, the first part of the algorithm, mentioned above, does not allow for a full parallel implementation. This is due to the fact that correlations among the quasi-random numbers within each sequence do exist, which destroy one of the key properties required for truly random number sequences. The way out from such a drawback is to scramble the quasi-random sequences at each time step, which can be realized by means of a suitable reordering strategy. Reordering consists in relabeling all realizations according to their radial distances from the starting point, at each time step. While this mechanism has been shown to be effective in many instances [8,20,22,24], it contrasts with the possibility of computing in parallel all realizations.

In the following examples, the global error reduction observed passing from PDD to quasi-PDD is depicted in contour plots, and the efficiency of PDD is compared with that achieved with PFD.

Example A. A contour plot showing the pointwise numerical error made solving the elliptic boundary value problems

$$u_{xx} + u_{yy} - 5u = 0 \qquad \text{in } \Omega = (0,1) \times (0,1), \tag{18}$$

with the boundary condition

$$u(x,y)|_{\partial\Omega} = \left(e^{2x+y}\right)_{\partial\Omega}, \tag{19}$$

by PDD and quasi-PDD appears in [8]. We include it also here, however, for the purpose of illustration, see Fig. 3.

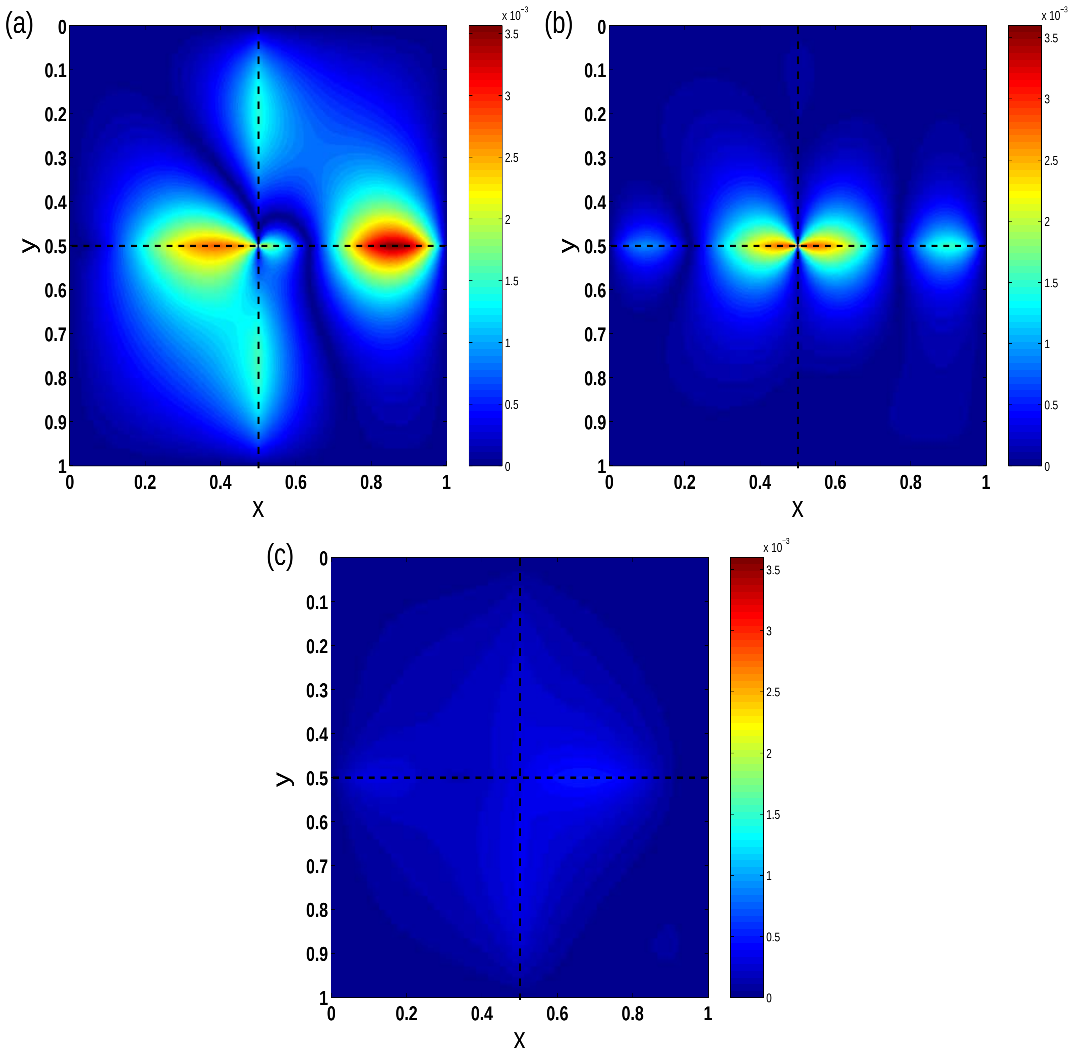

Fig. 3. Example A. Pointwise numerical error in: (a) the PDD algorithm, (b) the quasi-PDD algorithm with two nodal points on each interface, evaluated by quasi-Monte Carlo, and (c) the quasi-PDD algorithm with three nodal points on each interface. Parameters are $N = 10^4$, $\Delta x = \Delta y = 2 \times 10^{-3}$, $\lambda = 10^3$.

Here we also show the effect of a suitable boundary treatment, aimed at approximating accurately the first exit times, see Fig. 4. The convergence of the numerical method as a function of the number of realizations, N, is illustrated in Fig. 2. The numerical error has been computed at the points $(x, 0.5)$, for several values of equally spaced abscissae. In addition to using the Euler method with a constant time step, we solved the underlying SDEs by an exponential timestepping method [32,33]. The latter method is based on generating random time steps, picked up from an exponential distribution characterized by a parameter λ, with $\langle \Delta t \rangle = 1/\lambda$. A major advantage of such a procedure rests in the availability of an explicit analytic formula for the hitting probability. Knowing this, allows for an accurate computation of the first exit times. Note that this would be impossible for general problems if constant time step schemes are adopted. In Fig. 4, the numerical errors at points $(x, 0.5)$, obtained using the Euler method, the exponential timestepping method, and the exponential timestepping method along with a suitable boundary check, are shown. The stochastic process associated to Eq. (18) is actually a 2D standard Brownian motion, for which the Euler scheme yields the exact solution for every Δt. Therefore, the truncation error related to the Euler method applied to the corresponding SDEs vanishes. In addition, adopting quasi-random number sequences with $N = 10^4$ realizations, the error inherent to the Monte Carlo simulations is of order 10^{-4}, and thus the overall dominating error is due to the error made approximating exit times. Note that in Fig. 4, the results obtained using exponential timestepping are (slightly) worse than those obtained by the Euler method because the latter is the exact solution of the discrete problem while the former is not. When, however, a boundary check is made by virtue of explicit knowledge of the hitting probability, the corresponding results are by far better. Here, the value $\lambda = 10^3$ has been used.

In Fig. 2, the numerical error made solving Example A at the point $(0.3, 0.5)$ by quasi-Monte Carlo is shown as function of the number of realizations, N. The numerical method used to solve the associated SDE is the Euler method with a rather small time step, $\Delta t = 10^{-4}$, in order to keep negligeable the truncation error as well as the the error made in approximating exit times. Here the curve obtained fitting the data is also plotted, to display the dependence on the number of realizations. The fitted curve, obtained by a linear regression method, turns out to be $y = -0.9698x - 0.1808$.

This example can also to exploited to illustrate the effect of the interpolation process on the (absolute value of the) numerical error. In Fig. 5, such an error is plotted versus x, keeping $y = 0.5$ fixed, using two, three, and four nodal points to interpolate by Chebyshev polynomials. The solution at both endpoints is known from the boundary conditions. For this problem, the maximum interpolation error made using three internal nodes reduces to about one third of that obtained with two nodes. Using four nodes, it drops to the same order of that on the quasi-Monte Carlo computed nodal values themselves. Increasing further the number of nodes would be therefore useless.

Example B. Consider the 2D elliptic equation

$$u_{xx} + u_{yy} = 2 \qquad \text{in } \Omega = (0, 1) \times (0, 1), \tag{20}$$

subject to the boundary data

$$u(x, y)|_{\partial\Omega} = g(x, y), \tag{21}$$

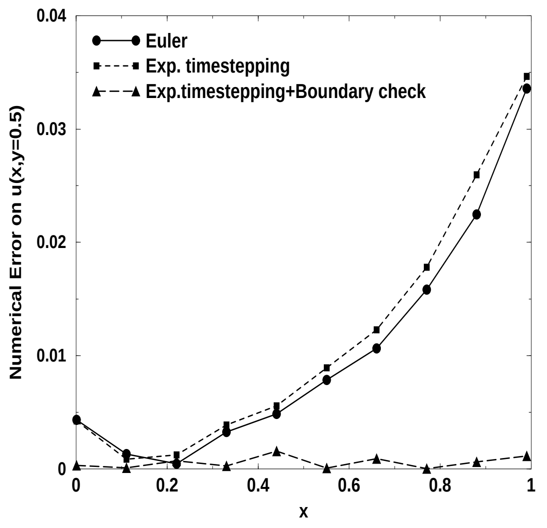

Fig. 4. Numerical error made solving Example A. The quasi-Monte Carlo method was used at the points $(x, 0.5)$. Parameters are $N = 10^4$, $\Delta t = 10^{-2}$, and $\lambda = 10^3$.

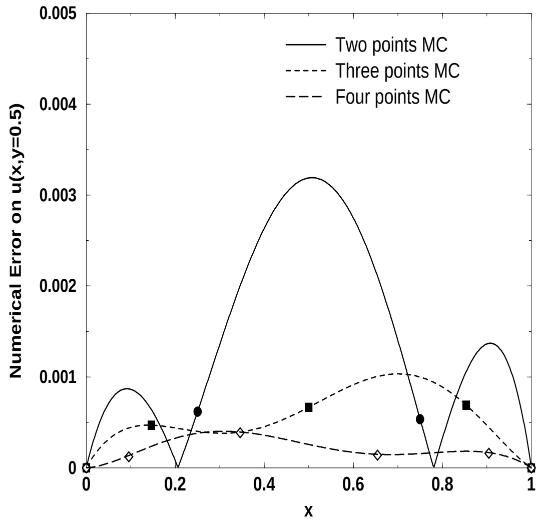

Fig. 5. Numerical error made solving the elliptic problem in Example A by quasi-Monte Carlo, computing two, three and four nodal points (marked by circle, square, and diamond symbols, respectively), and then interpolating on such points. The parameters here are $N = 10^4$, $\lambda = 10^3$.

where $g(x, y) = (3x^2 + x\,y - 2y^2)_{\partial\Omega}$. The analytical solution of this problem is $u(x, y) = 3x^2 + x\,y - 2y^2$ in $\overline{\Omega}$.

In Fig. 6, a contour plot of the pointwise numerical error made using PDD and the quasi-PDD algorithms is shown. Note that the maximum error on the entire domain Ω is achieved on the interfaces, and more precisely on the interpolation nodes. This agrees with the observation made in section 2 about the maximum principle. It should be remarked that the quasi-PDD algorithm outperforms the PDD algorithm. As parameters here we used $N = 10^4$ realizations, $\Delta x = \Delta y = 2 \times 10^{-3}$ grid size, $\lambda = 10^3$ timestepping parameter (and

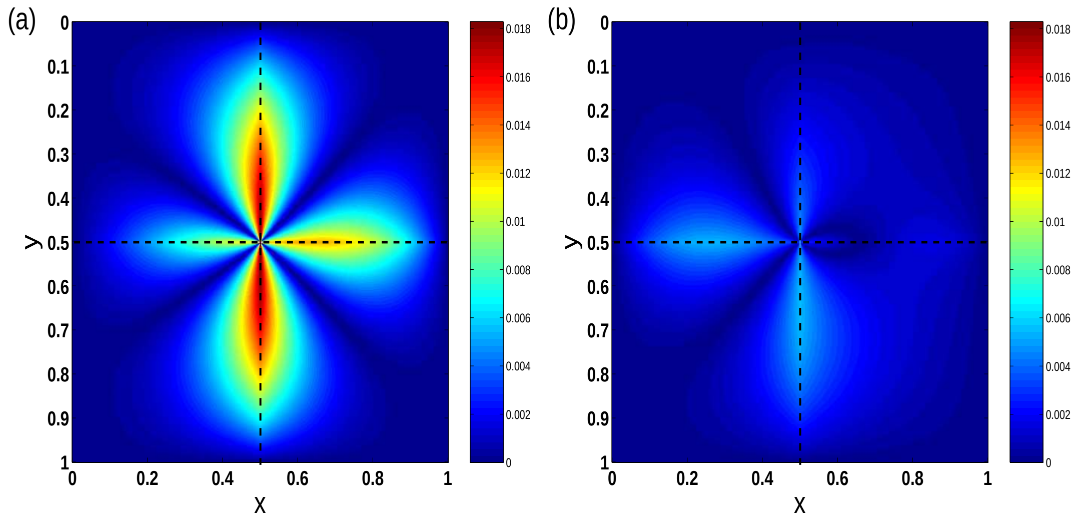

Fig. 6. Example B. Pointwise numerical error in: (a) the PDD algorithm, and (b) the quasi-PDD algorithm. Parameters are $N = 10^4$, $\Delta x = \Delta y = 2 \times 10^{-3}$, $\lambda = 10^3$.

thus an average time-step $\langle \Delta t \rangle = 10^{-3}$) to integrate the SDEs in Eq. (3).

Numerical experiments show that two nodes (that is four nodal points including the end points) suffice on each of the two interfaces. The local solver is based on Jacobi iteration, where the termination criterion has been chosen experimentally equal to 10^{-7}.

Table 1
Overall CPU time in seconds for example B

$Processors$	PFD	PDD_{Total}	$PDD_{Monte\,Carlo}$	PDD_{FD}
4	435.273	104.039	3.602	100.097
9	215.420	28.996	4.004	24.722
16	204.365	11.776	3.160	8.362

The second column (PFD) in Table 1 shows the total computational time (in seconds) spent by the parallel finite difference algorithm using $p = 4$, 9, and 16 processors, which corresponds to 4, 9, and 16 subdomains. The corresponding time spent by the PDD algorithm is shown in the third column. In the last two columns, this quantity is split into two parts, i.e, that required by the Monte Carlo simulation, and that needed by the local solvers. The two methods have been compared for about the same maximum error, 10^{-3}. In both algorithms the CPU time decreases as p increases, and this trend is more dramatic in the PDD algorithm. Moreover, the CPU time decreases for each given number of processors, passing from PFD to PDD, and this behavior is more pronounced, when the number of processors is higher.

Example C. Consider the elliptic equation with constant diffusion and a variable source,

$$\Delta u = (5x^2 + 5y - 4x - 2)\, e^{-(x+2y)} \qquad \text{in } \Omega = (0,1) \times (0,1), \tag{22}$$

with the boundary data

$$u(x,y)|_{\partial\Omega} = \left[(x^2 + y)\, e^{-(x+2y)}\right]_{\partial\Omega}, \tag{23}$$

the solution being given by $u(x,y) = (x^2 + y)\, e^{-(x+2y)}$.

In Fig. 7 similar results to those of Example B are shown. The parameters used here are the same as there.

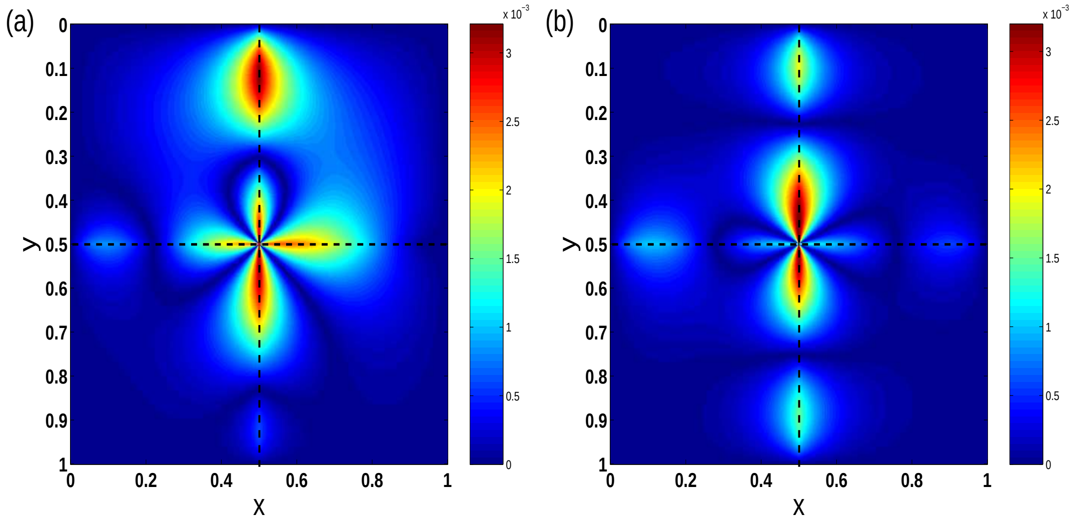

Fig. 7. Example C. Pointwise numerical error in: (a) the PDD algorithm, and (b) the quasi-PDD algorithm. Parameter values are the same as in Example A.

Table 2
Overall CPU time in seconds for example C

$Processors$	PFD	PDD_{Total}	$PDD_{Monte\,Carlo}$	PDD_{FD}
4	4654.939	1061.623	3.586	1057.493
9	2106.996	252.618	3.900	248.331
16	1368.333	93.689	3.104	90.246

We only comment the results of Table 2. Even though the problem is now more complex that the previous one, which is reflected by the longer CPU time appearing in Table 2, the PDD method still wins over the PFD. Again, the quasi-PDD algorithm outperforms the PDD algorithm, see Fig. 7

Example D. This example illustrates the performance of the numerical method in solving a general elliptic equation with variable diffusion, variable drift, variable potential, and variable source term,

$$\frac{y^2+1}{2}u_{xx}+\frac{x^2+1}{2}u_{yy}+x\,u_x+y^2\,u_y-(x^3+y^2)u = \\ P\cos(2x+y)+Q\sin(2x+y) \qquad \text{in } \Omega=(0,1)\times(0,1), \quad (24)$$

$$P = 1+x(4+x+2x^2)+2x\,y+x(4+x)y^2+y^3, \\ Q=-\frac{1}{2}[-2+x^4+2x^5+2x^3\,y+y(5-4y+6y^2)+x^2(1+y+6y^2)] \quad (25)$$

with the boundary data

$$u(x,y)|_{\partial\Omega}=\Big[(x^2+y)\sin(2x+y)\Big]_{\partial\Omega}, \qquad (26)$$

the solution being $u(x,y)=(x^2+y)\sin(2x+y)$.

Figure 8 is analog to the previous ones, and the parameter values used are the same as in Example B. Also in this case, we show in Fig. 8 contour plots for the pointwise numerical errors. General comments can be made as in the two previous example and, again, the same parameters have been used. As for the CPU times in Table 8, note that all values are greater than in Example C, due to the higher complexity of the present case.

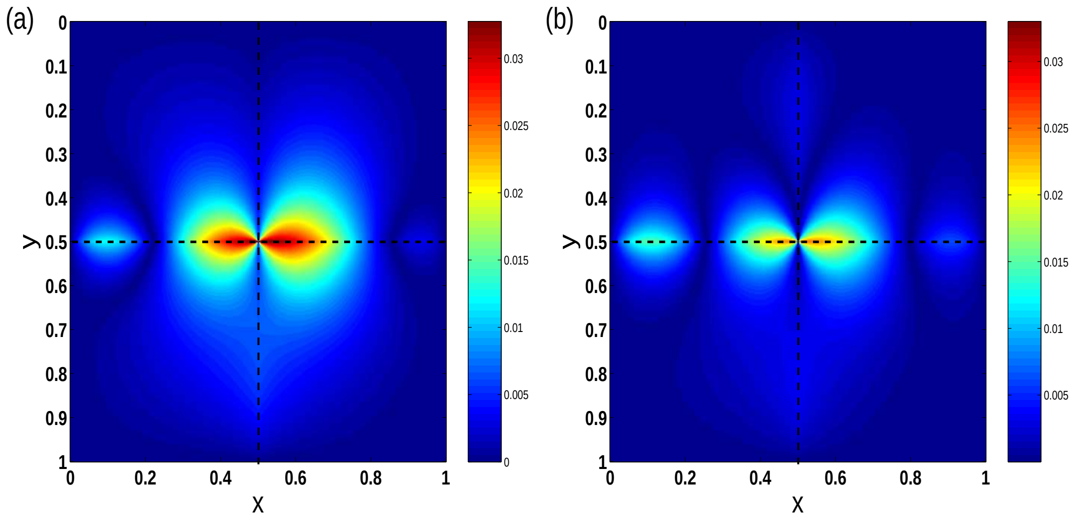

Fig. 8. Example D. Pointwise numerical error in: (a) the PDD algorithm, and (b) the quasi-PDD algorithm. Parameters are the same as in Example B.

Example E. In this example, the performance of the numerical method is tested in solving a general elliptic equation of the Poisson type, having an analytically unkown solution. Consider the Dirichlet problem

Table 3
Overall CPU time in seconds for example D

$Processors$	PFD	PDD_{Total}	$PDD_{Monte\,Carlo}$	PDD_{FD}
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

$$u_{xx} + u_{yy} = e^{\sin(x^2+y)} \qquad \text{in } \Omega = (0,1) \times (0,1), \tag{27}$$

with the boundary data

$$u(x,y)|_{\partial\Omega} = 0. \tag{28}$$

To quantify the numerical error, an accurate numerical solution was obtained solving problem (27)-(28) by a multigrid method, instead of using an analytical form of the solution, as in all the previous examples. Such analytical expression seems not to be available. The solution has been computed discretizing and solving the second-order difference approximation by multigrid iterations, for a grid of size 769 × 769. To compare it with the solution obtained by the PDD method, the solution above has been regrided to 500×500 using a cubic interpolation scheme. Both, the NAG routine d03edf and the free software *MUDPACK* [36], have been used.

As in the previous examples, in Fig. 9 the contour plots are shown for the pointwise numerical error. In Fig. 9 (a) and (b), only two nodes on each interface have been used in the PDD as well as in the quasi-PDD method, respectively. Figure 9 (c) shows the same quantity, obtained with three nodes on each interface (that is, using six nodal points in total). Note that increasing the number of nodal points yields an overall reduction of the numerical error on the whole domain.

Table 4
Overall CPU time in seconds for example E

$Processors$	PFD	PDD_{Total}	$PDD_{Monte\,Carlo}$	PDD_{FD}
4	3381.015	628.936	5.443	623.178
9	2184.664	167.103	13.638	153.208
16	2879.030	83.374	24.213	58.915

In Table 4, the CPU times required to solve the problem by the PFD and PDD method, are shown as a function of the number of processors. As in the previous examples, the PDD method wins over the PFD scheme, and the advantage in the CPU time is now even more striking.

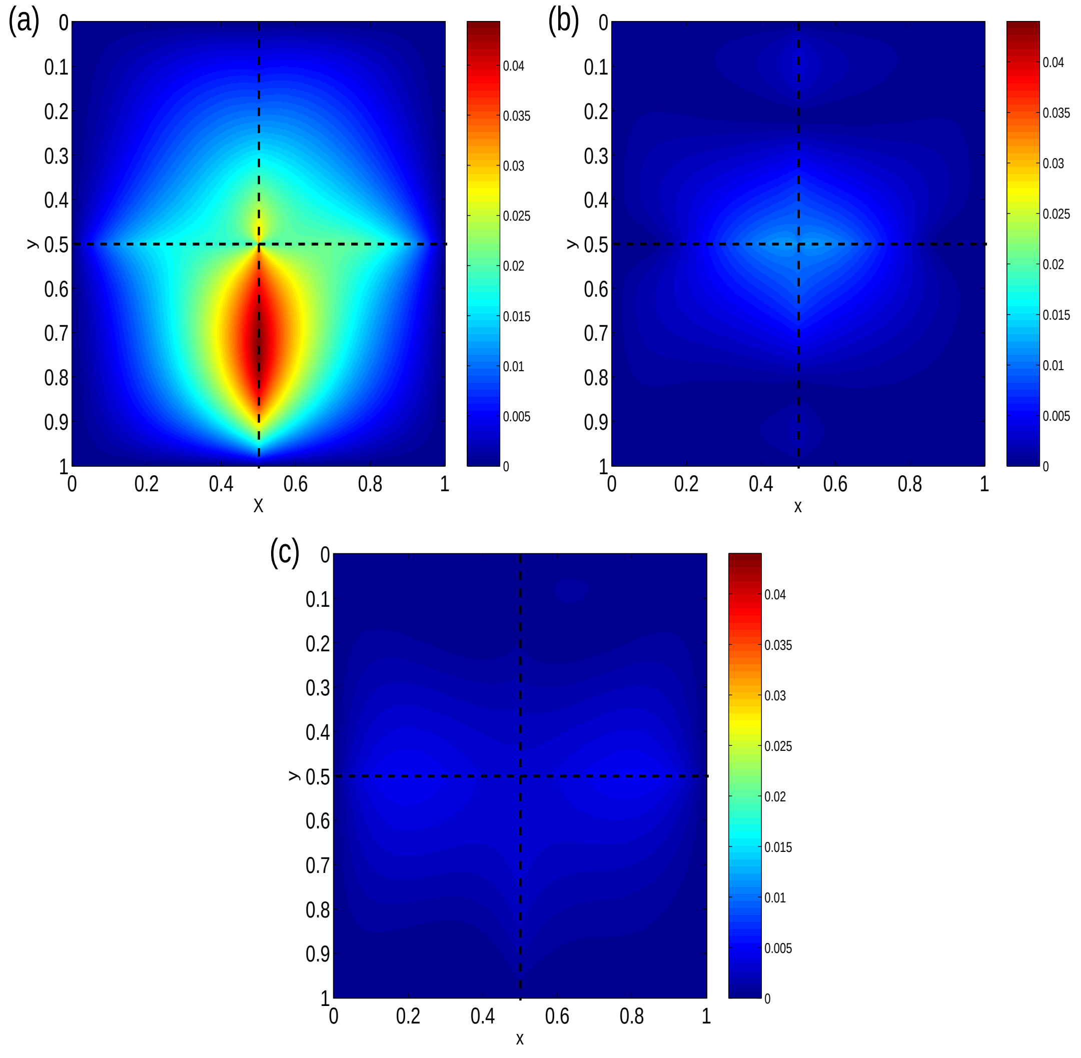

Fig. 9. Example E. Pointwise numerical error made with: (a) the PDD algorithm, (b) the quasi-PDD algorithm with two nodal points on each interface, evaluated by quasi-Monte Carlo, and (c) the quasi-PDD algorithm with three nodal points on each interface. Parameters are the same as in Example B.

5 Summary

Monte Carlo methods for solving Dirichlet problems for general linear elliptic equations have been rarely considered so far, mostly due to their poor performance. In this paper we have shown that such a performance can be dramatically improved by a variety of new techniques. One of these consists of accomplishing a domain decomposition based on computing by Monte Carlo only few interfacial values. Thus, the degree of parallelism which characterizes the algorithm is increased. Another winning strategy comes from adopting sequences of quasi-random numbers (instead of pseudorandom numbers), where special care has to be paid, such as "reordering" at each time step. Finally, a suitable boundary treatment has been shown to be essential, since the numerical error might otherwise dominate. An excellent way to accomplish the latter task turns out to be adopting an exponential timestepping method. Examples given here concerned boundary value problems for elliptic equations with variable coefficients, including a potential term and sources, and thus for instance Helmholtz and Poisson equations. The algorithm developed here is also characterized by *scalability* as the number of processors increases, and by being naturally *fault tolerant*.

References

[1] Freidlin, M., *Functional Integration and Partial Differential Equations*, Annals of Mathematics Studies no. 109, Princeton Univ. Press, Princeton, 1985.

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

[3] Arnold, L., *Stochastic Differential Equations: Theory and Applications*, Wiley, New York, 1974.

[4] Higham, D.J., *An algorithmic introduction to numerical simulation of stochastic differential equations*, SIAM Rev., **43**, No.3 (2001), pp. 525–546.

[5] Kloeden, P.E., and Platen, E., *Numerical Solution of Stochastic Differential Equations*, Springer, Berlin, 1992.

[6] Platen, E., *An introduction to numerical methods for stochastic differential equations*, Acta Numerica (1999), 197-246 [Cambridge University Press, Cambridge, 1999]

[7] Mascagni, M., *A tale of two architectures: parallel Wiener integral methods for elliptic boundary value problems*, Astfalk, Greg (ed.), Applications on advanced architecture computers. Philadelphia, PA: SIAM Software-Enviroments-Tools, pp. 27–33 (1996); SIAM News, **23**, No.4 (1990), pp. 8–12.

[8] Acebrón, J.A., Busico, M.P., Lanucara, P., and Spigler, R., *Domain decomposition solution of elliptic boundary-value problems via Monte Carlo and quasi-Monte Carlo methods*, SIAM J. Sci. Comput., (2005), in press.

[9] Spigler, R., *A probabilistic approach to the solution of PDE problems via domain decomposition methods*, "ICIAM 1991" (The Second International Conference on Industrial and Applied Mathematics), Washington, DC, July 8-12, 1991, Contributed Presentation, Session CP5, p. 12, Book of Abstracts.

[10] Talay, D., *Probabilistic Numerical Methods for Partial Differential Equations: Elements of Analysis*, Lecture Notes in Math., **1627**, Springer, Berlin, 1996, pp. 148–196.

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

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

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

[14] Morokoff, W.J., and Caflisch, R.E., *Quasi-random sequences and their discrepancies*, SIAM J. Sci. Statist. Comput., **15** (1994), pp. 1251–1279.

[15] Moskowitz, B., and Caflisch, R.E., *Smoothness and dimension reduction in quasi-Monte Carlo methods*, J. Math. Comput. Modeling, **23** (1996), pp. 37–54.

[16] Niederreiter, H., *Random Number Generation and Quasi-Monte Carlo Methods*, SIAM, Philadelphia, PA, 1992.

[17] Morokoff, W.J., and Caflisch, R.E., *Quasi-Monte Carlo integration*, J. Comput. Phys., **122** (1995), pp. 218–230.

[18] Caflisch, R.E., Morokoff, W., and Owen, A.B., *Valuation of mortgage-backed securities using Brownian bridges to reduce effective dimension*, J. Comput. Finance, **1** (1997), pp. 27–46.

[19] Morokoff, W.J., *Generating quasi-random paths for stochastic processes*, SIAM Rev., **40** (1998), pp. 765–788.

[20] Morokoff, W.J., and Caflisch, R.E., *A quasi-Monte Carlo approach to particle simulation of the heat equation*, SIAM J. Numer. Anal., **30** (1993), pp. 1558–1573.

[21] Lécot, C., *A quasi-Monte Carlo method for the Boltzmann equation*, Math. Comp., **56** (1991), pp. 621–644.

[22] Lécot, C., and El Khettabi, F., *Quasi-Monte Carlo simulation of diffusion*, J. Complexity, **15** (1999), pp. 342–359.

[23] Hofmann, N., and Mathé, P., *On quasi-Monte Carlo simulation of stochastic differential equations*, Math. Comp., **66** (1997), pp. 573–589.

[24] Acebrón, J. A., and Spigler, R., *Fast simulations of stochastic dynamical systems*, J. Comput. Phys., (2005), in press.

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

[26] Quarteroni, A., and Valli, A., *Domain Decomposition Methods for Partial Differential Equations*, Oxford Science Publications, Clarendon Press, Oxford, 1999.

[27] Ogawa, S., and Lécot, C., *A quasi-random walk method for one-dimensional reaction-diffusion equations*, Math. Comput. Simul. **62** (2003), pp. 487–494.

[28] Coulibaly, I., and Lécot, C., *Simulation of diffusion using quasi-random walk methods*, Math. Comput. Simul. **47** (1998), pp. 153–163.

[29] Gobet, E., *Weak approximation of killed diffusion using Euler schemes*, Stochastic Process. Appl., **87** (2000), pp. 167–197.

[30] Buchmann, F.M., *Computing exit times with the Euler scheme*, ETH Research Report No. 2003-02, March 2003; *Simulation of stopped diffusions*, J. Comput. Phys. **202** (2005), pp. 446–462.

[31] Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P., *Numerical Recipes*, Cambridge University Press, Cambridge, 1996.

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

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

[34] Mascagni, M., Karaivanova, A., and Li, Y., *A quasi-Monte Carlo method for elliptic boundary value problems*, Monte Carlo Meth. Appl., **7** (2001) pp. 283–294.

[35] Sabelfeld, K., *Monte Carlo Methods in Boundary Value Problems*, Springer, Berlin, 1991.

[36] http://www.scd.ucar.edu/css/software/mudpack/