A new probabilistic approach to the domain decomposition method

Juan A. Acebrón[1] and Renato Spigler[2]

[1] Departamento de Automática, Escuela Politécnica, Universidad de Alcalá, Crta. Madrid–Barcelona, Km. 31.600, 28871 Alcalá de Henares, Madrid, Spain
juan.acebron@uah.es

[2] Dipartimento di Matematica, Università di "Roma Tre,", Largo S. Leonardo Murialdo 1, 00146 Roma, Italy
spigler@mat.uniroma3.it

A hybrid numerical scheme based on a probabilistic method along with a classical domain decomposition is proposed for solving numerically linear elliptic boundary-value problems. Full decoupling can be accomplished by computing few values of the solution inside the domain by Monte Carlo or quasi-Monte Carlo techniques, and interpolating on the nodal points where the solution has been obtained previously. Thus, this method appears to be fault-tolerant as well as suited for time decomposition. Some examples are shown to illustrate performance and scalability.

1 Introduction

The domain decomposition method is nowadays considered one of the most natural ways to exploit parallel architectures in solving boundary-value problems for partial differential equations (PDEs). The main idea consists of decoupling the original problem into several sub-problems, and was proposed originally in the seminal work of H. A. Schwarz in 1870. More precisely, the given domain is divided into a number of subdomains, and then assign the task of the numerical solution on such separate subdomains to different processors. However, the computation cannot run independently for each subdomain, because they are coupled together through an internal interface, where the solution is unknown. Therefore, for every computational time step, processors have to exchange data along these interfaces, slowing down the overall performance.

In fact, due to the global character of the PDE, the solution cannot be obtained at a single point inside the domain prior to solving the full problem. Consequently, certain iterations are required across the chosen (or prescribed) interfaces, in order to determine approximate values of the sought solution

inside the original domain. There exists two approaches for a domain decomposition depending on whether domains are overlapping or not overlapping, see [DA94, QUA99, TOS05], e.g. Being the domains coupled, some additional numerical work is needed, and therefore, it is doubtful whether full scalability can be attained as the number of the subdomains (hence, of the processsors) increases unboundedly.

In order to overcome such a drawback, a new method has been recently proposed [AN05, AN05b]. The core of the method is based on combining a certain probabilistic method suited for solving elliptic and parabolic partial differential equations along with a classical domain decomposition method, namely a *probabilistic domain decomposition* method (PDD). This approach allows to obtain the solution in some points, internal to the domain, without solving first the entire problem. In fact, this can be done by means of the probabilistic representation of the solution. The basic idea is to compute only 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.

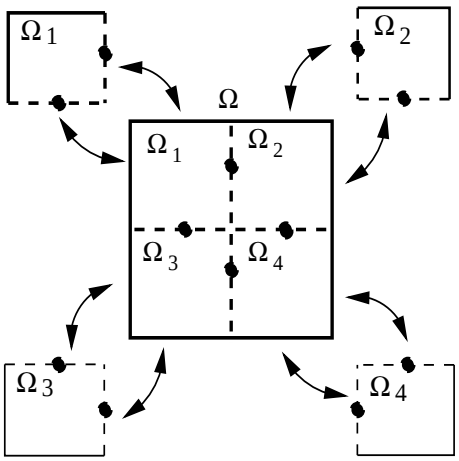

Fig. 1. Sketchy diagram illustrating the numerical method, splitting the initial domain Ω into four subdomains, $\Omega_1, \Omega_2, \Omega_3, \Omega_4$.

Solving numerically boundary-value problems for linear elliptic and parabolic partial differential equations by the probabilistic representation of their solutions, essentially by a Monte Carlo method, is known since long time, see [CH98, KAL86]. It is based on the well-known Feymann-Kac formula, which is a representation extremely powerful, and inherently parallel since it al-

lows for obtaining the solution in single points inside the domain. However, it can be hardly used because requires evaluating accurately the first exit point along with the first exit time from the domain, as well as solving numerically a boundary value problem for a stochastic differential equation. Both issues, however, can be managed reasonably resorting to several powerful numerical and asymptotic techniques, extracted from probability theory, number theory and statistical physics, see [AN05c, JS00, JS03].

2 Generalities

To the purpose of illustration, let confine ourselves to the case of the Dirichlet problem for a linear elliptic equation,

$$Lu - c(\mathbf{x})u = f(\mathbf{x}), \quad \mathbf{x} \in \Omega \subset \mathbf{R}^2, \qquad u|_{\partial\Omega} = g, \tag{1}$$

where $L := \sum_{i,j=1}^{2} a_{ij}(\mathbf{x})\partial_i\partial_j + \sum_{i=1}^{2} b_i(\mathbf{x})\partial_i$ is a linear elliptic operator with smooth coefficients, $c(\mathbf{x}) \geq 0$, boundary $\partial\Omega$ of the domain Ω also smooth, as well as the boundary data, g, and the source term, f. The probabilistic representation is given by,

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

see [FRE85, KAR91], e.g., where $\beta(t)$ is the (vector-valued) stochastic process associated to the elliptic operator L, which solves the system of (Ito type) stochastic differential equations (SDEs)

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

where $W(t)$ represents the 2-dimensional standard brownian motion (also called Wiener process), and $\tau_{\partial\Omega}$ is the first passage (or hitting) time of the path $\beta(t)$ started at the point $\mathbf{x}$ to $\partial\Omega$. As usual, the dependence of β on the chance variable, running on the underlying probability space, is not displayed. When the operator L is the Laplace operator, Δ, the stochastic process $\beta(t)$ reduces to the standard 2-dimensional brownian motion. The drift vector, $\mathbf{b}$ in (3) is the same appearing in the operator L, that is $\mathbf{b}(\mathbf{x}) = (b_1, b_2)^T$, while the diffusion matrix, σ, is related to the coefficients a_{ij} by the relation $\sigma\sigma^T = a \equiv (a_{i,j})_{i,j=1,2}$.

The representation formula in (2) is used to obtain few values of the solution at some points inside the domain Ω. The expected value is approximated by an arithmetic mean (which is known to provide the best estimator) over N realizations of the process β, at the price of a (statistical) error of order of $N^{-1/2}$. The main catch of using a Monte Carlo method rests on this fact, which entails a rather poor accuracy, unless N is taken extremely large. An

alternative, however, do exist, and consists of resorting to sequences of quasi-random numbers [NIE92], which have been used succesfully in mathematical finance, and recently applied in solving stochastic differential equations with high efficiency [AN05c]. Using quasi-random numbers allows for speeding up the calculations with respect to the classical Monte Carlo method based on pseudorandom numbers, since now the statistical error becomes of order of N^{-1}. Such a method is called quasi- Monte Carlo.

Since evaluating the solution via a probabilistic method may require a large number of realizations (large sample size) in order to reduce the associated statistical error, we compute the solution in a few points along the interfaces between subdomains. Such points will be used as nodal points to evaluate interpolating the solution at the interfaces.

Apart from the statistical error, there are however other sources of numerical error which affect the evaluation of $u(\mathbf{x})$ by means of (2), besides that due to the finite sample size mentioned above. These are due to: (*i*) the truncation error made in the numerical solution of the SDEs in (3); (*ii*) the uncertainty of estimating first exit times; (*iii*) the numerical quadrature errors in (2). Estimating precisely the first exit times and the first exit points (which are also needed for problems with $c(\mathbf{x})$ and $f(\mathbf{x})$ not vanishing for all $\mathbf{x}$) has been often overlooked in the existing literature. An efficient way to locate accurately the first exit time is based on the use of an exponential timestepping, see [JS00, JS03]. All these sources of error have been analyzed in [AN05, AN05b].

At the present time, machines working in the petaflops regime, and endowed with hundred of thousands or even millions of processors are planned for the near future, and taking full advantage from massive parallel computing would be highly desirable. With such machines, the issue of *scalability* remains open, at least in some cases. As it was pointed out in [KEY98], Schwarz-type DD methods are not truly scalable, at least in the theoretical sense, since their parallel efficiency in solving elliptic problems is subject to degradation as the number of processors, p, goes to infinity, indeed when p starts being over the thousands. It seems however that things go better, in practice, for a number of reasons, described in [KEY98].

In addition, the possibility of failure of even few processors is very likely to occur frequently [GEI03]. Therefore, algorithms which are scalable and *fault-tolerant* at the same time would be extremely important if not mandatory.

The method proposed here seems to be free of the aforementioned drawbacks. In fact, decoupling is complete, and it was shown in [AN05, AN05b] that scalability is attained with respect to an arbitrary number of subdomains or processors, and the algorithm is naturally fault-tolerant. The latter property rests on two ingredients, one due to the intrinsic parallelizability of the Monte Carlo methods, and to the full decoupling that can be realized.

3 Numerical examples

Below we show some examples to illustrate the numerical method here proposed. It is worth to stress that, even though the "pivotal" values generated by Monte Carlo or by quasi-Monte Carlo are poorly accurate (unless an extremely large sample, N, of realizations is used), and the Chebyshev interpolation adds some additional error, the numerical error inside each subdomain is dominated by the boundary errors, due to the maximum principle, and decays rapidly going inside.

Note that a comparison with a true deterministic DD method has not yet been done, being shown only a comparison with "parallel finite differences". However, we do not expect that our algorithm migh outperform necessarily any given deterministic DD method, but, rather, that our approach might win over others regarding full scalability and fault-tolerance.

All codes have been implemented using OpenMP, which is a standard parallelization library, designed for shared memory computer architectures. We simply used a 16 processor IBM Power 3 machine, working at 375 MHz clock, and with a peak performance of 24 GFLOPS.

Example 1. Let consider the Dirichlet problem [AN05b]

$$\frac{y^2+1}{2}u_{xx}+\frac{x^2+1}{2}u_{yy}+x\,u_x+y^2\,u_y-(x^3+y^2)u= P\cos(2x+y)+Q\sin(2x+y) \qquad \text{in } \Omega=(0,1)\times(0,1), \quad (4)$$

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

with the boundary data

$$u(x,y)|_{\partial\Omega}=\left[(x^2+y)\sin(2x+y)\right]_{\partial\Omega}, \quad (6)$$

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

Table 1. CPU time in seconds for example 1

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

In Fig. 2*a* and 2*b*, the pointwise numerical error is shown, made correspondingly to the PDD with pseudorandom sequence of numbers, and quasi-random, respectively. Here only two nodes on each interface have been used.

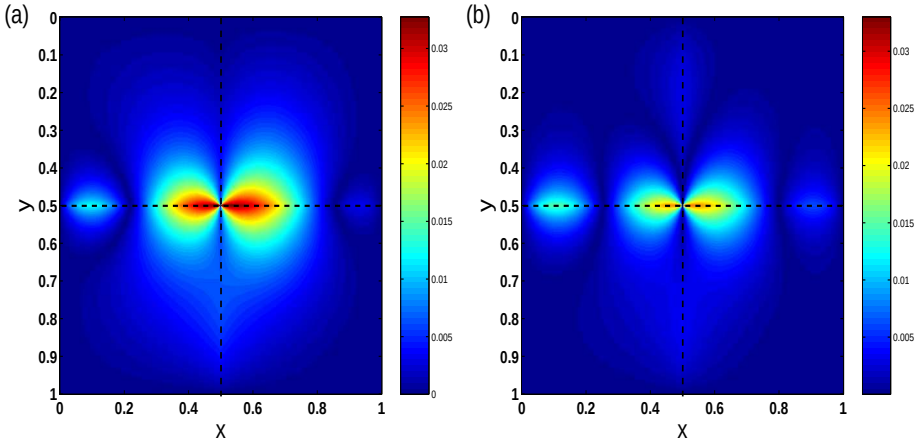

Fig. 2. Example 2. Pointwise numerical error in: (a) the PDD algorithm, and (b) the quasi-PDD algorithm. Parameters are $N = 10^4$, $\Delta x = \Delta y = 2 \times 10^{-3}$, $\Delta t = 10^{-3}$.

It should be remarked that the quasi-PDD algorithm outperforms the PDD algorithm. 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, such a 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 2. Consider the so-called Stommel model, which is a two dimensional model for ocean circulation, and is given by

$$u_{xx} + u_{yy} + \beta\, u_x = -\alpha \sin(\pi y/2) \qquad \text{in } \Omega = (0,1) \times (0,1), \tag{7}$$

with the boundary data $u(x,y)|_{\partial\Omega} = 0$, and $\alpha = 10$, $\beta = 1$. In this example an analytical solution is unknown, and to quantify the numerical error, an accurate numerical solution obtained solving the elliptic equation by a multigrid method has been used instead. Similarly to the previuous example, it is shown in Fig. 3 the contour plots for the pointwise numerical errors.

4 Conlusions

A hybrid method based on a probabilistic approach along with a classical domain decomposition for the numerical solution of elliptic partial differen-

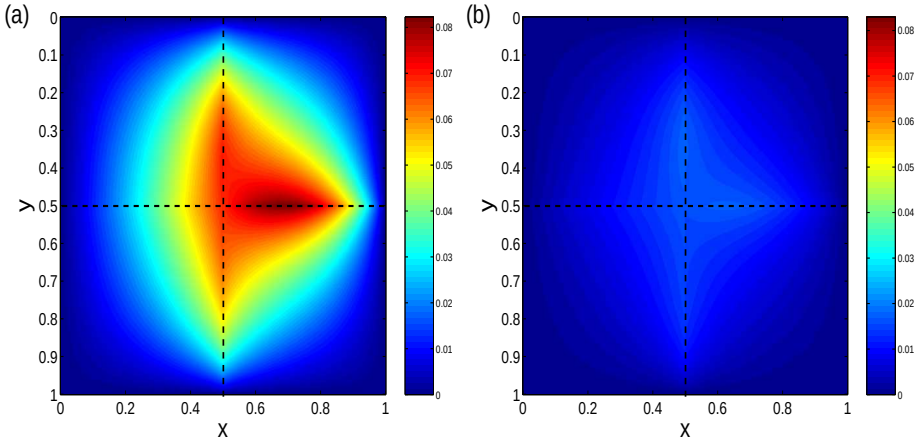

Fig. 3. Example 2. Pointwise numerical error in: (a) the PDD algorithm, and (b) the quasi-PDD algorithm. Parameters are as in Fig. 2

tial equation, has been described. The associated stochastic differential equation is solved by Monte Carlo simulation only at very few points of a given interface internal to the domain. Then, the solution at the interface is constructed interpolating by using such points as nodal points. Consequently, a full splitting into several subdomains, to be handled by separate processors acting simultaneously, can be accomplished. Since it has been shown in the literature that the ensuing error may dominate, an essential ingredient of the algorithm consists of a suitable boundary treatment. Moreover, it has been shown that efficieny can be increased adopting sequences of "quasi-random numbers" (instead of the more customary pseudorandom numbers).

It is worthwhile noticing that a comparison with true deterministic domain decomposition algorithms has not been made yet. This has been done only with parallel finite differences. Such an algorithm allows for an automatic distribution of the computational load among the prescribed subdomains. We do not expect that our code might be necessarily competitive with the existing deterministic codes, but, rather, that it might compete as for its scalability and fault-tolerance properties. In fact, such approach allows for a complete decoupling among processors, without degrading the overall performance due to strong interprocessors communication. Therefore, it appears to be well suited for grid computing and nowadays supercomputers with hundreds of thousands of processors or more.

The availability of such a large number of processors in supercomputers, and to put every available processor to work, suggests to think about developing new strategies of parallelization, which exploit now time [FT03]. The method proposed here can be generalized to account for parabolic partial differential equations, where the time evolution of the solution should be now

taken into account. In fact, the probabilistic method can be used as well now to evaluate the solution of a parabolic partial differential equation in any given point and time inside the spatio-temporal domain.

References

[AN05] Acebrón, J.A., Busico, M.P., Lanucara, P., Spigler, R.: Domain decomposition solution of elliptic boundary-value problems via Monte Carlo and quasi-Monte Carlo methods. SIAM J. Sci. Comput., in print (2005).

[AN05b] Acebrón, J.A., Busico, M.P., Lanucara, P., Spigler, R.: Probabilistically-induced domain decomposition methods for elliptic boundary-value problems. Submitted (2005).

[AN05c] Acebrón, J.A., Spigler, R.: Fast simulations of stochastic dynamical systems. J. Comput. Phys., in print (2005).

[CH98] Caflisch, R.E.: Monte Carlo and quasi Monte Carlo methods. Acta Numerica, 1–49 (1998)

[DA94] Dryja, M., Widlund, O.B.: Domain decomposition algorithms with small overlap. SIAM J. Sci. Comput., **15**, 604–620 (1994)

[FT03] Farhat, C., Chandesris, M.: Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid-structure applications. Int. J. Numer. Meth. Engng., **58**, 1397–1434 (2003)

[FRE85] Freidlin, M.: Functional integration and partial differential equations. In: Annals of Mathematics Studies no. 109. Princeton Univ. Press (1985)

[GEI03] Geist, G.A.: Progress towards Petascale Virtual Machines. In: Dongarra, J. (ed) Lecture Notes in Computer Science. Springer, Berlin (2003).

[JS00] Jansons, K.M., Lythe, G.D.: Efficient numerical solution of stochastic differential equations using exponential timestepping. J. Statist. Phys., **100**, 1097–1109 (2000)

[JS03] Jansons, K.M., Lythe, G.D.: Exponential timestepping with boundary test for stochastic differential equations. SIAM J. Sci. Comput., **24**, 1809–1822 (2003)

[KAL86] Kalos, M.H., Withlock, P.A.: Monte Carlo methods, Vol. I: Basics. Wiley, New York (1986)

[KAR91] Karatzas, I., Shreve, S.E.: Brownian motion and stochastic calculus, 2nd ed. Springer, Berlin Heidelberg New York (1991)

[KEY98] Keyes, D.E.: How scalable is domain decomposition in practice?. In: Eleventh International Conference on Domain Decomposition Methods (London, 1998). DDM.org, 286-297 (electronic). Augsburg (1999)

[NIE92] Niederreiter, H.: Random number generation and quasi Monte-Carlo methods. SIAM (1992).

[QUA99] Quarteroni, A., Valli, A.: Domain decomnposition methods for partial differential equations. Oxford Science Publications, Clarendon Press (1999)

[TOS05] Toselli, A., Widlund, O.: Domain Decomposition Methods - Algorithms and Theory. Springer Series in Computational Mathematics, Vol. 34 (2005).