Scalability and performance analysis of a probabilistic domain decomposition method

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

[1] Departament d'Enginyeria Informàtica i Matemàtiques,
Universitat Rovira i Virgili, Av. Països Catalans, 26
43007 Tarragona, Spain,
`juan.acebron@urv.cat`
[2] Dipartimento di Matematica, Università "Roma Tre", 1, Largo S.L. Murialdo,
00146 Rome, Italy
`spigler@mat.uniroma3.it`

Abstract. In this paper, we analyze the scalability and performance of a probabilistic domain decomposition strategy for solving linear elliptic boundary-value problems. Such a strategy consists of a hybrid numerical scheme based on a probabilistic method along with a domain decomposition, and full decoupling can be accomplished. It is shown that such a method performs well for an arbitrarily large number of processors, while the classical deterministic approach is strongly affected by intercommunications. Therefore, the overall performance degrades dramatically for rather large number of processors. Furthermore, we find that the probabilistic method is scalable as the number of subdomains, i.e., the number of processors involved, increases. This fact is clearly illustrated by an example.

1 Introduction

Domain decomposition (DD) is considered one of the most natural ways to decouple boundary-value (BV) problems for partial differential equations (PDEs) into sub-problems, in order to exploit parallel architectures, thus allowing for high-performance scientific computing of large-scale problems. The idea, which in its essence goes back to the 1870 work of H. A. Schwarz, is to split the given domain into a number of subdomains, and then assign the task of the numerical solution on such separate subdomains to as many separate processors. The main drawback, however, is represented by the fact that the solution is needed on some interfaces internal to the domain, while solving BV problems for PDEs has a global character. That is, the solution cannot be obtained even 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. Both, overlapping domains and not overlapping domains have been considered in the literature, see [13, 14], e.g. In any case, some additional numerical work is required to accomplish this task, and it is doubtful whether scalability can be attained as the number of the subdomains (hence, of the processsors) increases unboundedly [10]. Recently, a method has been developed which avoids the intercommunication problems inherent to any traditional DD approach [1, 2, 4]. The method based on a probabilistically induced domain decomposition (PDD), and has been proven to be rather successful in homogeneous parallel architectures. In Section 2, some generalities about the method are discussed. In Section 3, a numerical example is shown, where the performance in a MPI environment was tested. In the final section, we summarize the high points of the paper.

2 The probabilistic method

The core of the probabilistic method is based on combining a probabilistic representation of solutions to elliptic or parabolic PDEs with a classical domain decomposition method (DD). This algorithm can be referred to as a "*probabilistic domain decomposition*" (for short, PDD) method. This approach allows to obtain the solution at some points, internal to the domain, without first solving the entire boundary-value 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 by Monte Carlo simulations on certain chosen interfaces, and then interpolate to obtain continuous approximations of it on such interfaces. The latter can then be used as boundary values to decouple the problem into subproblems, see Fig. 1. Each such subproblem can then be solved *independently* on a separate processor. Clearly, neither communication among the processors nor iteration across the interfaces are needed. Moreover, the PDD method does not even require balancing. In fact, after decomposing the domain into a number of subdomains, each problem to be solved on them will be totally independent of the others. Hence, each problem can be solved by a single host. Even though some hosts may end the computation much later than others, the results obtained from the faster hosts are correct, and can be immediately used, when necessary.

The implementation of the PDD algorithm can be accomplished through the following steps:

1. Compute only *few* interfacial values by Monte Carlo simulations.
2. Interpolate on the corresponding nodes to obtain boundary values for the subdomains.
3. Compute the solution to the original problem in each subdomain by standard methods (e.g., finite differences or finite elements).

Therefore, the key idea is to generate only very few values of the sought solution, u, by a probabilistic method, the Monte Carlo method, see [6], e.g. In some sense, this approach allows to obtain the solution in some points, internal to the domain, without solving first the entire problem. This can be done by means of the probabilistic representation of the solution,

$$u(\mathbf{x}) = E^L_{\mathbf{x}} \left[g(\beta(\tau_{\partial\Omega})) e^{-\int_0^{\tau_{\partial\Omega}} c(\beta(s))\,ds} - \int_0^{\tau_{\partial\Omega}} f(\beta(t))\, e^{-\int_0^t c(\beta(s))\,ds}\, dt \right], \quad (1)$$

see [7], 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)

$$d\beta = \mathbf{b}(\mathbf{x})dt + \sigma(\mathbf{x})dW(t), \tag{2}$$

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 $\partial\Omega$.

The Monte Carlo approach is trivially parallelizable, since each of the N problems above can be runned independently of the others. Stated in such terms, clearly, the method appears to be scalable as well. Even though the probabilistic algorithm we derived and considered here is scalable, naturally fault tolerant [8], and well suited to grid and heterogeneous computing [5], it suffers for some weakness, due to the inherently poor accuracy of all Monte Carlo methods. A considerable improvement, however, can be achieved using sequences of "quasi-random numbers" [6, 12] instead of sequences of pseudorandom numbers. The pseudorandom numbers are those numbers obtained in practice, when we try to generate truly random numbers, hence are approximately characterized by a statistical distribution. The quasi-random numbers, instead, are deterministic uniformly distributed numbers. Using the latter allows to obtain an error (now deterministic) of order of $\mathcal{O}(N^{-1}\log^{d^*-1}N)$, d^* representing a certain "effective" space dimension. It was shown in [3] that the underlying system of SDEs can indeed be solved numerically in a very efficient way.

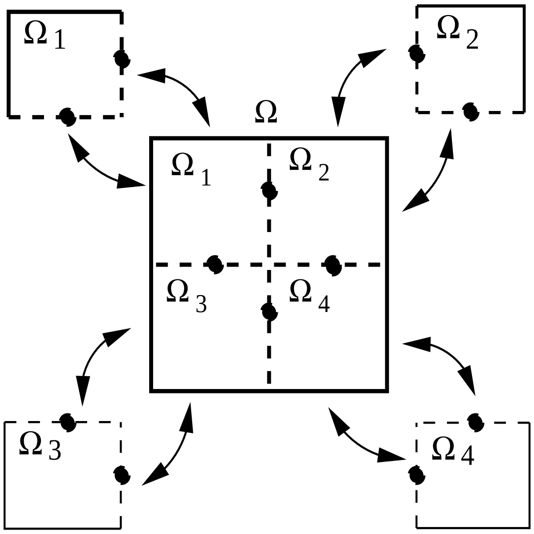

Fig. 1. Sketchy diagram illustrating the numerical method, splitting the initial domain Ω into four subdomains, $\Omega_1, \Omega_2, \Omega_3, \Omega_4$.

3 Numerical example

Here we present a numerical example, aiming at comparing the performance and scalability achieved by a classical deterministic domain decomposition method and the PDD method. For the purpose of illustration, we analyzed a prototype partial differential equation on a rather elementary domain, indeed, the unit square. The partial differential equation chosen is the Laplace equation.

A code in an MPI environment has been implemented and runned on the MareNostrum supercomputer located at the Barcelona Supercomputing Center (BSC).

Below, in order to compare the relative performance of the PDD method against some deterministic method, we solved the same equation in both ways. The deterministic algorithm is extracted from the numerical package pARMS, having chosen the overlapping Schwarz method with a FGMRES iterative method preconditioned with ILUT as local solver, see [15]. To split the given linear algebraic system corresponding to the full discretized problem into a number of subproblems, and solve them independently, in parallel, it is necessary to accomplish a mesh partitioning. This should be done balancing the overall computational load, minimizing, at the same time, the intercommunication occurring among the various processors. We used ParMETIS [11] as a partitioner, configuring it according to the characteristics of the particular mesh we adopted.

To make the comparison meaningful, we discretized the local problems within the PDD algorithm by finite differences, and solved the ensuing linear algebraic system by the same FGMRES iterative solver preconditioned with ILUT.

Consider the following Dirichlet problem,

$$u_{xx} + u_{yy} = 0 \qquad \text{in } \Omega := (0,1) \times (0,1), \tag{3}$$

$$u(x,y)|_{\partial\Omega} = g(x,y), \tag{4}$$

where $g(x,y) := \left[\left(x^2 - y^2\right)\right]_{\partial\Omega}$, for the Laplace equation in two dimensions. The solution of such a problem is explicitly known, and is $u(x,y) = (x^2 - y^2)$ in $\overline{\Omega} = [0,1] \times [0,1]$. In order to analyze scalability of both domain decomposition methods, the initial domain was scaled conveniently as function of the number of processors, p involved. This has been done in order to keep constant the computational load per processor, being the space discretization fixed to $\Delta x = \Delta y = 1.25 \times 10^{-3}$. Here two cases have been studied so far. Firstly, the domain was scaled along the x dimension proportionally to the nu mber of processors, p (in the following, it will be termed as Case A), and secondly, both dimensions x, and y were simultaneously rescaled (Case B). That is, the rescaled domain for case A becomes $[0,p]x[0,1]$, while for case B is $[0,\sqrt{p}]x[0,\sqrt{p}]$.

For both cases, the parallel run time of the PDD method can be estimated. In the following, we focus only on estimating the time spent by the Monte Carlo part of the algorithm. This is mainly because once a continuous approximation of the interfaces is obtained, the problem is fully decoupled. Hence,the corresponding time spent by the local solver in each subdomain (being of equal size) on separate processors is independent of the number of processors.

In [1], it was shown that the time T_{MC} required for computing a single interfacial value of the solution by Monte Carlo or quasi-Monte Carlo is given by

$$T_{MC} \sim N^{1+\beta/\alpha}\, d\, f(d). \tag{5}$$

Here α is the convergence order of the numerical scheme used to solve the underlying SDE, and β is equal to $1/2$ or 1, depending on whether pseudorandom or quasi-random number sequences is chosen. N denotes the number of realizations, while d corresponds to the problem dimension. In particular, for this example, the function $f(d)$ turns out to be constant, when only one dimension is scaled, and proportional to p, when both are scaled. This is related on how the mean first exit time depends on dimension, see [1].

For the PDD method, let us consider k nodal points on each interface. Note that the error due to interpolation, depends on the length of the interfaces. Therefore, to keep bounded such an error, the number of nodal point should be increased accordingly to the number of processors involved. This should be done only for the case B, because the length of the interfaces remains constant for case A. In fact, for case B the effective number of nodal points required to keep bounded the interpolation error should increase as $p^{1/2}$. The global domain is partitioned into p subdomains. These are nonoverlapping squares $(0,1)\times(0,1)$. For case A, a $1D$ partitioning is applied, and the number of interfaces is $p-1$. On the other hand, for case B, the number of interfaces is $2(\sqrt{p}-1)$ Therefore, for case A the overall CPU time spent by Monte Carlo would be of order $T_{MC}(p-1)$, while for case B, $2T_{MC}(\sqrt{p}-1)$ Thus, for case A the parallel run time for this part of the algorithm is as follows:

$$T_p = T_{MC}k\frac{p-1}{p} \sim T_{MC}k \quad \text{as } p\to\infty, \tag{6}$$

while for case B yields,

$$T_p = 2kT'_{MC}p\sqrt{p}\frac{\sqrt{p}-1}{p} = 2kT'_{MC}(p-\sqrt{p}), \tag{7}$$

being $T'_{MC} = T_{MC}/f(d)$. In Fig. 2(a), and 2(b), the parallel run time results for case A and case B, respectively, are depicted. Note that the Monte Carlo part remains constant for case A, while for case B increases linearly as function of the number of processors. This agrees qualitatively with the theoretical estimation obtained above.

In Fig. 3(a), and 3(b) the pointwise numerical error made in the PDD and DD methods are shown in a contourplot. Parameters were chosen conveniently in order to attain a comparable error for both methods. For the PDD method, k was kept fixed to 2. Note that the maximum error made in each subdomain is indeed attained on the corresponding boundary. The exponential timestepping [9] used to solve the underlying SDEs (2) was characterized by $\lambda = \langle \Delta t\rangle^{-1}$, Δt being the random exponentially distributed time step used in solving the SDEs, and the bracket denoting its average. Note that maximum error is of order 10^{-2}, which corresponds mainly to the statistical error obtained from the Monte

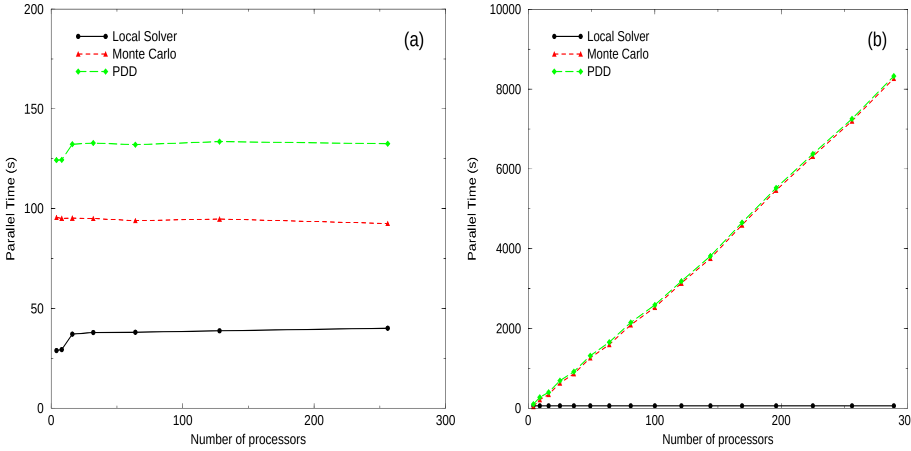

Fig. 2. Parallel run time of the PDD method versus the number of processors, where the time spent by Monte Carlo and the local solver is also detailed:(a) Case A, and (b) Case B.

Carlo method at the nodal points. Increasing the accuracy can be attained by increasing the sample size, or resorting to sequences of "quasi-random" numbers keeping fixed the sample size, see [2].

Fig. 4(a), and 4(b) shows the results corresponding to the deterministic domain decomposition method for cases A, and B, respectively. Here the overall parallel run time has been decomposed in two parts, corresponding to the time spent by the partitioner, and the iteration part. Note that both of them increases when the number of processor grows as expected. Moreover, the iteration part increases unbounded for larger number of processors due to the heavy intercommunications, degrading dramatically the performance of the algorithm.

4 Conclusions

The analysis of the performance of a probabilistic method, to accomplish domain decomposition for the numerical solution of linear elliptic boundary-value problems in two dimensions, has been conducted. The solution is generated by Monte Carlo simulations to solve the associated stochastic differential equations only at very few points inside the domain. A Chebyshev interpolation using such points as nodes is then constructed, and a full splitting into several subdomains, to be handled by separate processors acting concurrently, is made.

A comparison with a deterministic DD algorithm for different partitions of the scaled domain has been made here for the first time. This has been done in an MPI environment. Working in an MPI environment also allows to test the effect of processor intercommunications which beset all deterministic DD algorithms.

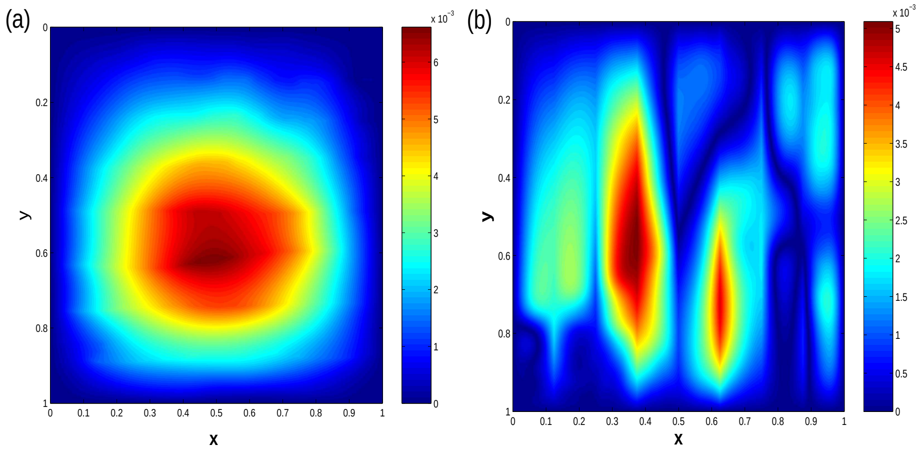

Fig. 3. Pointwise numerical error in: (a) the DD algorithm, and (b) the PDD algorithm.

Besides the competitive results observed in the numerical example, the PDD method is expected to be competitive concerning scalability and fault-tolerance. These are indeed key issues if one intends to run codes on machines working with hundreds of thousands of processors or more.

Acknowledgements.

J.A.A. acknowledges support from the Ministerio de Ciencia y Tecnología (MEC) through the Ramón y Cajal programme. The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the Barcelona Supercomputing Center-Centro Nacional de Supercomputación.

References

1. 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., **27**, 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**, 421–438 (2005).
3. Acebrón, J.A., and Spigler, R., “Fast simulations of stochastic dynamical systems”, J. Comput. Phys., **208**, 106–115 (2005).
4. Acebrón, J.A., and Spigler, R., “Supercomputing applications to the numerical modeling of industrial and applied mathematics problems”, J. Supercomputing, **40**, 67–80 (2007).

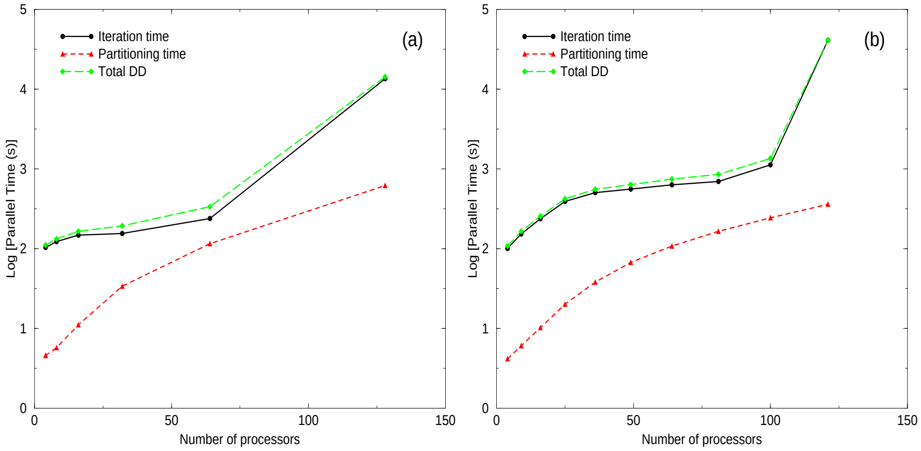

Fig. 4. Parallel run time of the DD method versus the number of processors, where the time spent by the partitioner and the iteration procedure is also detailed:(a) Case A, and (b) Case B.

5. Acebrón, J.A., Durán, R., Rico, R,, and Spigler, R., “A new domain decomposition approach suited for grid computing”, Lecture Notes in Computer Science, in press (2007)
6. Caflisch, R.E., “Monte Carlo and quasi Monte Carlo methods”, Acta Numerica. Cambridge University Press, 1–49. (1998).
7. Freidlin, M.: Functional integration and partial differential equations. Annals of Mathematics Studies no. 109, Princeton Univ. Press (1985).
8. Geist, G.A., “Progress towards Petascale Virtual Machines”, Lecture Notes in Computer Science, **2840**, 10–14 (2003).
9. Jansons, K.M., and Lythe, G.D., “ Exponential timestepping with boundary test for stochastic differential equations”, SIAM J. Sci. Comput., **24** 1809–1822 (2003).
10. Keyes, D.E., “ How scalable is domain decomposition in practice?” in the *Eleventh International Conference on Domain Decomposition Methods* (London, 1998), 286-297 (electronic), DDM.org, Augsburg, (1999).
11. Karypis, G., and Kumar, V., “ParMETIS–parallel graph partitioning and field–reducing matrix ordering”, http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview
12. Niederreiter, H.: Random number generation and quasi Monte-Carlo methods. SIAM (1992).
13. Quarteroni, A., and Valli, A.: Domain decomposition methods for partial differential equations. Oxford Science Publications, Clarendon Press (1999).
14. Toselli, A., and Widlund, O.: Domain Decomposition Methods - Algorithms and Theory. Springer Series in Computational Mathematics, Vol. 34 (2005).
15. Li, Z., Saad, Y., and Sosonkina, M., “pARMS: a parallel version of the algebraic recursive multilevel solver”, Numerical Linear Algebra with Applications, **10**, 485–509 (2003).