

A new domain decomposition approach suited for grid computing

Juan A. Acebrón¹, Raúl Durán², Rafael Rico², and
Renato Spigler³

¹ Departament d'Enginyeria Informàtica i Matemàtiques,
Universitat Rovira i Virgili, Av. Països Catalans, 26
43007 Tarragona, Spain,
`juan.acebron@urv.cat`

² Departamento de Automática, Escuela Politécnica,
Universidad de Alcalá de Henares, Crta. Madrid-Barcelona, Km 31.600,
28871 Alcalá de Henares, Madrid, Spain,
`raul.duran@uah.es, rafael.rico@uah.es`

³ Dipartimento di Matematica, Università "Roma Tre", 1, Largo S.L. Murialdo,
00146 Rome, Italy
`spigler@mat.uniroma3.it`

Abstract. In this paper, we describe a new kind of domain decomposition strategy for solving linear elliptic boundary-value problems. It outperforms the traditional ones in complex and heterogeneous networks like those for grid computing. 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. While the deterministic approach is strongly affected by intercommunication among the hosts, the probabilistic method is scalable as the number of sub-domains, i.e., the number of processors involved, increases. This fact is clearly illustrated by an example, even operating in a grid environment.

1 Introduction

It is well-known that a number of applications can benefit from operating in a grid environment [4, 5, 18]. A grid system could be exploited for solving very large problems, which are otherwise intractable even with the existing supercomputers, but latency will be a crucial issue. All experiments show this. The most successful case of grids is that of handling huge amounts of data, as in the celebrated example of the SETI project (SETI@home), and other similar to it. In these applications latency does not play any negative role. Due to the high degree of system heterogeneity and high latency, however, only few applications seem to be computationally advantageous. Indeed, parallel scientific computing to handle problems based on the solution of partial differential equations (PDEs) is currently at a crossroad. Among the several numerical methods proposed in the literature for solving boundary-value problems for PDEs, domain decomposition methods seem to be particularly well-suited for parallel architectures.

The main idea consists in decoupling the original problem into several subproblems. More precisely, the given domain is divided into a number of subdomains, and the task of the numerical solution on such separate subdomains is then assigned to different processors or computational sites. However, the computation cannot run independently for each subdomain, because the latter are coupled to each other through internal interfaces where the solution is still unknown. Therefore, the processors have to exchange data along these interfaces at each computational time step, resulting in a degradation of the overall performance. In fact, due to the global character of the PDE boundary-value problem, the solution cannot be obtained even at a single point inside the domain prior to solving the entire problem. Consequently, certain iterations are required across the chosen (or prescribed) interfaces in order to determine approximate values of the solution sought inside the original domain. There exist two approaches for a domain decomposition, depending on whether the domains overlap or do not overlap; see [16, 17], e.g. Given the domain, the subproblems are coupled, hence some additional numerical work is needed, and therefore it is doubtful whether full scalability might be attained as the number of the subdomains increases unboundedly.

Moreover, implementing such a method on a parallel architecture involves partitioning the given mesh into the corresponding subdomains, balancing the computational load as well as minimizing the communication overhead at the same time. For this purpose, traditionally, graph partitioning strategies have been used [10, 11]. Such strategies rest on considering the problem's domain as a graph made of weighted vertices and communication edges. Then, the balancing problem can be viewed as a problem of partitioning a graph by balancing the computational load among subdomains while minimizing the communication edge-cut. However, most of the traditional partitioners are no longer suitable for the emerging grid strategies [12]. In fact, these algorithms only consider balancing computational load and reducing communication overhead, while in practice the data movement cost may be a crucial issue in view of the high communication latency on a grid.

In order to overcome this drawback, several strategies have been already proposed. In [14, 15], an unbalanced domain decomposition method was considered. The idea there consists of hiding communication latency by performing useful computations, and assigning a smaller workload to processors responsible for sending messages outside the host. This approach goes in the right direction, but other ways are currently in progress. For instance, two graph partitioners, suited for grid computing, i.e., capable to take into account the peculiarities of grid systems, with respect to both, CPUs and network heterogeneity, are Mini-Max [8] and PaGrid [9]. These algorithms try to minimize the overall CPU time instead of balancing computational load, minimizing intercommunication at the same time. In a homogeneous environment, these two actions are equivalent, but in the heterogeneous case, things are very different. A parallel version of METIS, ParMETIS [10, 11], is only capable to handle CPUs heterogeneity.

In this paper, we propose a method recently developed by some of the authors and others [1, 2]. It 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 grid 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 approach 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.

Fault tolerance of algorithms is an essential issue in grid environments, since a typical characteristic of a grid is to allow dynamically aggregation of resources what results in a strong modification of the structure of the system itself. Hence, all classical methods, which require continuous communications among the various processors involved, will be seriously affected and the overall performance in general degraded. Even a single processor exiting the system at some point, will in general abort the running process, since all the remaining processors must wait for the results expected from such processor. The probabilistic method, instead, is unaffected by this kind of events, since all subproblems are fully decoupled and the unaffected processors may continue processing their task.

3 Numerical examples

Here we present some numerical examples, aiming at comparing the performance achieved by a classical deterministic domain decomposition method and the PDD method, in a grid environment. To this purpose, the Globus Toolkit’s services and the Globus-based MPI library (MPICH-G2) [7, 19] have been used. We built

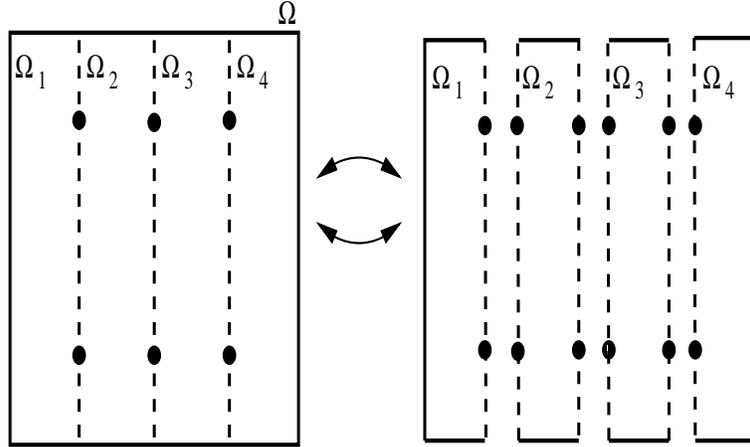


Fig. 1. Sketchy diagram illustrating the numerical method, splitting the initial domain Ω into four subdomains, $\Omega_1, \Omega_2, \Omega_3, \Omega_4$.

the highly heterogeneous computing system sketched in Fig.2, made with 16 PCs AMD Duron processors (below referred to as of type Cluster A), 1.3 GHz, linked to each other through a Fast Ethernet connection, plus 2 other PCs, one being an AMD Athlon XP 2400+ (type B), the other an AMD-K6, 500 MHz (type C). These two PCs are also linked through a Fast network, but they are linked to the previous group of 16 PCs through a 10 Ethernet connection (which is much slower than the other, as is well known). We chose the example of the Dirichlet problem

$$u_{xx} + (6x^2 + 1)u_{yy} = 0 \quad \text{in } \Omega = (0, 1) \times (0, 1) \quad (1)$$

with the boundary data

$$u(x, y)|_{\partial\Omega} = [(x^4 + x^2 - y^2)/2]_{\partial\Omega} \quad (2)$$

the solution being $u(x, y) = (x^4 + x^2 - y^2)/2$.

This problem has been solved discretizing it by finite differences with size $\Delta x = \Delta y = 1/N$, $N = 1200$.

In Fig. 3(a) and 3(b), the pointwise numerical error is shown, made correspondingly to the DD and the PDD methods. For the former, only two nodes on which interpolation was made have been used on each interface. Parameters were chosen conveniently in order to attain a comparable error for both methods.

The deterministic algorithm we adopted is extracted from the numerical package pARMS [13], where the overlapping Schwarz method with a FGMRES iterative method has been chosen [6, 13]. This was preconditioned with ILUT as local solver. 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

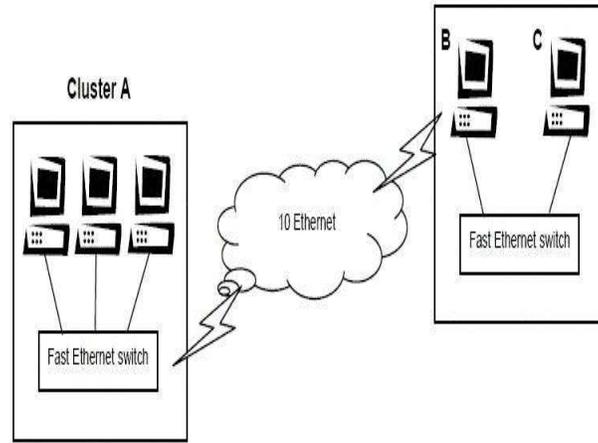


Fig. 2. Diagram showing the Grid environment in which the algorithms have been tested.

be done balancing the overall computational load (according to the processors' heterogeneity), minimizing, at the same time, the intercommunication occurring among the various processors. As already pointed out, operating in a grid environment, this task is by far more challenging than in a homogeneous hosts setting.

At this point, we used ParMETIS as a partitioner, configuring it according to the characteristics of the particular grid we adopted. For instance, it is possible to assign the computational load according to the CPU performance of each available processor. With this, the CPU heterogeneity is taken into account. As for the intercommunication problem, we observe that minimizing communications overhead may not correspond to minimizing graph's edge-cuts, since one should weight somehow the importance of every specific edge connecting pairs of nodes of the mesh. This importance depends on the specific problem to be solved, i.e., on the physics embodied in the problem, besides the geometry of the domain and the discretization underlying the chosen algorithm (such as finite differences, finite elements, etc.). Clearly, different partitions of the given domain lead to different linear algebraic problems on each subdomain. In particular, using iterative methods, the local problems will be characterized by matrices with different condition numbers.

On each subdomain, the time required for the computations is proportional to the size of the problem, i.e., to the number of nodes or the square of the matrix dimension, times the condition number of such a matrix. The latter determines the number of iterations required to achieve a prescribed accuracy in the iterative local solution. Recall that the time spent for computations on each subdomain (i.e., due to a given processor) is that spent per iteration times the number of iterations, and the former is proportional to the number of nodes. Of course, using direct (instead of iterative) methods, the condition numbers of the local

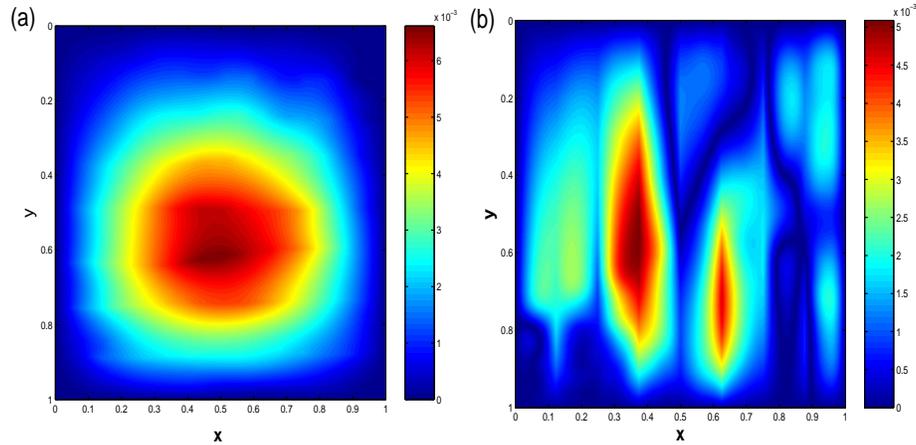


Fig. 3. Pointwise numerical error in: (a) the DD algorithm, and (b) the PDD algorithm.

matrices, A_i in $A_i x_i = b_i$, $i = 1, \dots, p$ (p being the number of subdomains) are only important in the amplification of errors in the entries of A_i and b_i . Direct methods, however, are used only when the local problems are small, typically with A_i of dimension not larger than a few hundreds.

The number of nodes in each subdomain could be chosen according to the specific performance of each CPU (the more powerful the CPU is, more nodes are assigned to it). The condition number of A_i is a much more delicate issue, since it also depends on the physics, i.e., in our problems, on the coefficients of the PDE. In the specific example above, in view of the asymmetry in x and y , the particular partition of the domain is very relevant, requiring very different iteration numbers. In Fig. 4(a), numerical experiments pertaining to the example below, show a partition which takes into account only the geometry of the problem. In Fig. 4(b), instead, a different partition has been obtained which takes into account the physics as well, having introduced such dependence in the weights of the edges. These weights increase with x , in the y direction. Therefore, in the configuration underlying Fig. 4(b), the total number of iterations required is smaller than in case of Fig. 4(a) (we needed 220 versus 294 iterations). From this example it appears clearly that minimizing the edge-cuts as in the configuration in Fig. 4(a) does not minimize the overall computational time, since the iteration number in such a case is larger than that needed in Fig. 4(b), see Table 1.

In the PDD approach, we discretized the local problems by finite differences, and solved the corresponding linear algebraic systems by the same iterative method. Here, the partitioning has been done as shown in Fig. 1, which corresponds, somehow, to the *optimal* one found above for the DD.

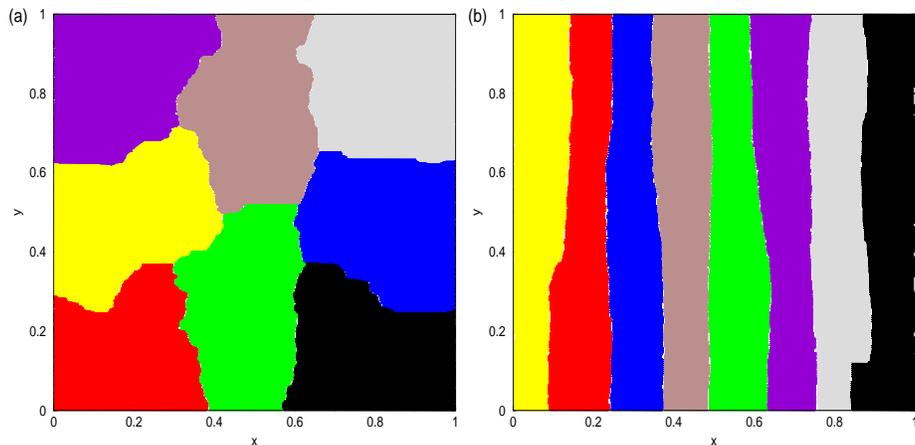


Fig. 4. Partitions of the given domain into 8 subdomains, and assigned to 8 processors for two different configurations: (a) Only geometry was taken into account, (b) Both geometry and physics was considered.

Table 1. CPU time in seconds for the heterogeneous system

<i>Processors</i>	<i>PDD(unbalanced)</i>	<i>PDD(balanced)</i>	<i>DD(A)</i>	<i>DD(B)</i>
8	88.67	43.18	372.72	282.73
16	46.36	25.67	300.86	221.30
18	32.52	18.44	270.21	302.75

As in the deterministic DD, CPU heterogeneity can be taken into account, hence all subdomains may have different area (and thus are characterized by different computational load). Moreover, in this case, due to the full decoupling among the various subdomains, the iteration times as well as the iteration numbers pertaining to every subdomain are easily computable. This is clearly shown in Fig. 5, where 16 processors have been used: all but the processors labelled by $p = 2$ and $p = 3$ correspond to type B and C, respectively. Note that the type C processor is the slowest one, hence the time needed is longer, and on the other hand to obtain the entire solution one has to wait for it to finish. However, other

Table 2. CPU time in seconds for the homogeneous subsystem (Cluster type A)

<i>Processors</i>	<i>PDD</i>	<i>DD(A)</i>	<i>DD(B)</i>
8	38.78	169.71	132.35
16	20.82	242.01	197.49

than in the deterministic DD, the PDD allows to use the solution computed in the various subdomains as soon as the corresponding processors complete their task. In addition, being now the subdomains fully uncoupled from each other, this allow us to minimize easily the overall CPU time, assigning a higher computational load to the faster processors, and reducing that to the slower ones. In Fig. 5, the iteration times and the number of iterations required by the local solver (labelled by p), have been displayed in case of 16 processors. A comparison is made balancing the computational load, according to the heterogeneity of the system, and without any balancing.

Table 1 shows the performance of both methods, the PDD, and DD. For the latter, two different partitions, labelled by A and B (see Fig. 4), have been used. The results for the PDD algorithm were obtained, balancing or not balancing the computational load. The two methods have been compared correspondingly to about the same maximum error, 10^{-3} (see Fig. 3). Note that the PDD algorithm scales well, as well as the DD in configuration A. However, the performance of the DD algorithm seems to be much better in configuration B, for a low number of processors, degrading when more processors are included in the system. In any case, the PDD always outperforms the DD method, the results being even more striking when the number of processors is higher. This fact is clearly due to the high intercommunication overload inherent to the DD algorithm.

In Table 2, the CPU times required to solve the problem in the subsystem Cluster of type A by the PDD and by the DD method, are shown for two different number of processors. Note that DD does not scale for both partitioning configurations, in contrast to the results shown in Table 1 for the whole heterogeneous system. The advantage in terms of CPU time achieved with the PDD method in comparison to the DD scheme, has now been reduced, even though is still important.

4 Observations and conclusions

In view of the previous results, grid computing can be considered as an emerging alternative system, that could compete with the most powerful supercomputers [3], in both, data parallelization *and* scientific computing. In the latter case, however, the development of suitable algorithms is still in its infancy. The probabilistic domain decomposition proposed in this paper is a simple but promising method, going in this direction. Monte Carlo methods, which are one of the ingredients of our approach, are known to be “embarrassingly parallel”, but more, they seem to be fully scalable, fault tolerant, and well suited to heterogeneous computing, in particular to that special case represented by grid computing.

Acknowledgements

This work was completed during a visit of J.A.A. in Rome, supported by the Short-Term Mobility program of the Italian CNR for the year 2006. J.A.A. also acknowledges support from the Spanish Ramón y Cajal Program. R.D., and

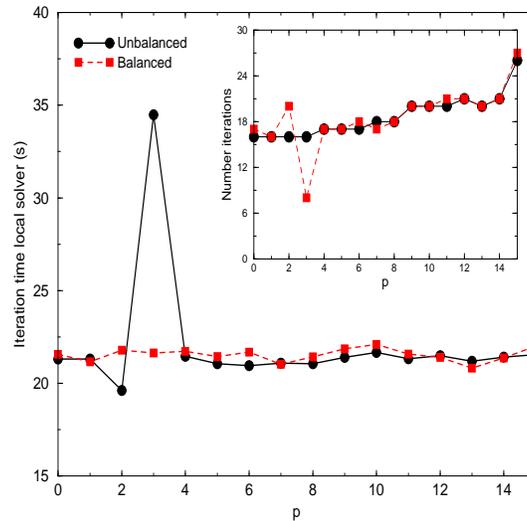


Fig. 5. Iteration times and number for each subdomain in the PDD algorithm, being computed with 16 processors.

R.R were funded through grant UAH PI2005/072 and CAM-UAH2005/042. We thank J. Cardoso, and E. Moratilla for their invaluable technical support and in developing the Grid computer system environment in which our algorithms were tested.

References

1. 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.* **27** (2005) 440–457
2. Acebrón, J.A., Busico, M.P., Lanucara, P., pigler, R.: Probabilistically induced domain decomposition methods for elliptic boundary-value problems. *J. Comput. Phys.* **210** (2005) 421–438
3. Acebrón, J.A., Spigler, R.: Supercomputing applications to the numerical modeling of industrial and applied mathematics problems. *J. Supercomputing* (2007), in press.
4. Boghosian, B.M., Coveney, P.V.: Scientific applications of Grid computing. *IEEE CS Press* **7** (2005) 10–13
5. Dong, S., Karniadakis, G.E., Karonis, N.T.: Cross-site computations on the Tera-Grid. *IEEE CS Press* **7** (2005) 14–23
6. Dongarra, J.J., Duff, I.S., Sorensen, D.C., van der Vorst, H.A.: *Numerical Linear Algebra on High-Performance Computers*. SIAM, Philadelphia (1998).
7. Foster, I.: *Globus Toolkit Version 4: Software for Service-Oriented Systems*. *Lecture Notes in Computer Science* **3779** (2005) 2–13
8. Harvey, D.J., Das, S.K., Biswas, R.: Design and performance of a heterogeneous grid partitioner. *Algorithmica* **45** (2006) 509–530

9. Huang, S., Aubanel, E., Virendrakumar, C.B.: PaGrid: A mesh partitioner for computational grids. *J. Grid Computing* **4** (2006) 71–88
10. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* **20** (1998) 359–392
11. Karypis, G., Kumar, V.: ParMETIS—parallel graph partitioning and field-reducing matrix ordering.
<http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview>
12. Li, Y., Lan, Z.: A survey of load balancing in grid computing. *Lecture Notes in Computer Science* **3314** (2004) 280–285
13. Li, Z., Saad, Y., Sosenkina, M.: pARMS: a parallel version of the algebraic recursive multilevel solver. *Numerical Linear Algebra with Applications* **10** (2003) 485–509
14. Otero, B., Cela, J.M., Badia, R.M., Labarta, J.: A domain decomposition strategy for grid environments. *Lecture Notes in Computer Science* **3241** (2004) 353–361
15. Otero, B., Cela, J.M., Badia, R.M., Labarta, J.: Performance analysis of domain decomposition applications using unbalanced strategies in grid environments. *Lecture Notes in Computer Science* **3795** (2005) 1031–1042
16. Toselli, A., Widlund, O.: *Domain Decomposition Methods - Algorithms and Theory*. Springer Series in Computational Mathematics, Vol. 34 (2005).
17. Quarteroni, A., Valli, A.: *Domain decomposition methods for partial differential equations*. Oxford Science Publications, Clarendon Press (1999)
18. *Distributed Computing*, Science, Special Issue, **308** (2005) 809–821
19. <http://www.globus.org>, (2006).