Strategies of Domain Decomposition to Partition Mesh-Based Applications onto Computational Grids

Beatriz Otero and Marisa Gil

Abstract— In this paper, we evaluate strategies of domain decomposition in Grid environment to solve mesh-based applications. We compare the balanced distribution strategy with unbalanced distribution strategies. While the former is a common strategy in homogenous computing environment (e.g. parallel computers), it presents some problems due to communication latency in Grid environments. Unbalanced decomposition strategies consist of assigning less workload to processors responsible for sending updates outside the host. The results obtained in Grid environments show that unbalanced distributions strategies improve the expected execution time of mesh-based applications by up to 53%. However, this is not true when the number of processors devoted to communication exceeds the number of processors devoted to calculation in the host. To solve this problem we propose a new unbalanced distribution strategy that improves the expected execution time up to 43%. We analyze the influence of the communication patterns on execution times using the Dimemas simulator.

Index Terms— Domain decomposition methods, load balancing algorithms, parallelism and concurrency, simulation.

1 Introduction

DOMAIN decomposition strategy is used for efficient parallel execution of mesh-based applications. These applications are widely used in various disciplines such as engineering, structural mechanics and fluid dynamics and require high computational capabilities [1]. Computational Grids are emerging as a new infrastructure for high performance computing. Clusters of workstations of multiple institutions can be used to efficiently solve PDEs in parallel where the problem size and number of processors are chosen to maintain sufficient coarse-grained parallelism. A workstation can be a computer or a group of computers and hereafter we will refer to both as *host*.

Our focus is simulations that make finite element analysis to solve the problems that arise from the discretization of PDEs on meshes. The general algorithmic structure of the explicit simulations is composed of two nested loops. The first, the outer loop, performs the discretization of PDE with reference to the simulation time. The second, the inner loop, applies this discretization onto all finite elements of the mesh. This inner loop performs a matrix-vector product. This numerical operation represents between 80% and 90% of the total iteration time, so an efficient parallelization of this calculation might significantly improve the total simulation time. To this end, we use domain decomposition techniques, where matrix and vector are decomposed properly in sub-domains of data that are mapped in one processor. Each sub-domain has interrelations with each of the others in the boundary elements.

- *B. Otero is with the Department d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, 08034, Spain.*
- *M. Gil is with the Department d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, 08034, Spain.*

In order to obtain optimal performance of mesh-based applications in Grid environments a suitable partitioning method should take into consideration several features, such as the characteristics of the processors, the quantity of traffic in the network, the latency and the bandwidth between processors both inside the host and between hosts. Most partitioners do not have this capacity; therefore they do not produce good results when the network and processors in the Grid have a heterogeneous nature.

In this paper, we evaluate mesh-based applications in Grid environments using a domain decomposition technique. The main objective of this study is to improve the execution time of mesh-based applications in Grid environments by overlapping remote communications and useful computation. To achieve this, we propose new strategies of domain decomposition for partitioning mesh-based applications in computational Grids where the workload can vary depending on the characteristics of the processor and of the network. The successful deployment of parallel mesh-based applications in a grid environment must involve efficient mesh partitioning. We use the Dimemas tool to simulate the behavior of the distributed applications in Grid environments [2].

This work is organized as follows. Section 2 briefly discusses the work related with this study. Section 3 describes the applications and the algorithmic structure of explicit simulations. Section 4 defines the Grid topologies considered and describes the tool used to simulate the Grid environment. Section 5 deals with the mesh-based applications studied and the workload assignment patterns. Section 6 shows the results obtained in the environments specified for the three different data distribution patterns. Section 7 presents the new unbalanced distribution that solves the problems of the unbalanced distribution proposed before. Finally, the conclusions of the

work are presented in Section 8.

2 Related Work

We can distinguish between two distinct types of related work, one based on the partitioner method and other based on the load balancing. As mentioned above, the success of parallel mesh-based applications in Grid environments depends on efficient mesh partitioning. Several works have already proposed partitioners for computational Grids. PART, JOSTLE, SCOTCH, MinEX, PaGrid and METIS are some examples of these.

PART [3] uses intensive algorithms and simulation annealing, so requires a parallel implementation to obtain good performance. JOSTLE [4] produces data partitions without taking into account the communication cost for each processor. SCOTCH [5] has the same limitation of JOSTLE because it generates partitions for homogeneous interprocessor communication cost. MinEX [6] makes partitions without taking into account the application granularity. PaGrid [7] uses some techniques already applied by other partitioners but adds a stage for load balancing the execution time. PaGrid produces comparable partitions to JOSTLE and attempts some improvement by minimizing the estimated execution time. Finally, METIS [8] is based on a multilevel recursive bisection algorithm.

All of these approaches consider estimated execution time rather than communication cost to measure the performance of a mesh partitioner. However, minimizing the communication between hosts is fundamental in computational Grids to reduce the execution time.

As regards workload, there are some works dealing with the relationship between architecture and domain decomposition algorithms [9]. There are several studies on latency, bandwidth and optimum workload to take full advantage of the available resources [10, 11]. There are also analyses of the behavior of MPI applications in Grid environments [12, 13]. In all of these cases, the same workload for all the processors is considered.

Li et al. [14] provide a survey of the existing solutions in load balancing as well as new efforts to deal with it in the face of the new challenges in Grid computing. In this work they describe and classify different schemes of load balancing for grid computing, but there is no solution which would be fully adaptive to the characteristics of the Grid.

In previous works we suggested two unbalanced distribution strategies, called *singleB-domain* and *multipleB-domain,* to execute mesh-based applications in Grid environments [15, 16, 17, 18]. All of these use unbalanced data distribution and they take into account the execution platform and the processor characteristics. Both strategies minimize the communication between the processors and reduce the expected execution time by up to 53% when compared with a balanced distribution strategy. In this paper we present the details of the two unbalanced distributions proposed above. We describe the characteristics of the applications executed and the schemes to explicit simulations, and we propose a new unbalanced distribution, called *multipleCB-domain* distribution, which combines the two previous unbalanced distributions and allows a more efficient processor utilization.

3 Applications and Simulations

In this section we describe the mesh-based applications features and the general algorithmic structure of the simulation schemes.

3.1 Mesh-based Applications

Finite element methods have been fundamental techniques in the solution of problems in engineering modeled by PDEs. These methods include three basic steps:

1. Step 1: The physical problem is written in variational form (also called weighted residual form).
2. Step 2: The problem's domain is discretized by complex shapes called elements. This is called meshing.
3. Step 3: The variational form is discretized using quadrature rules leading to a system of equations. The solution of this system represents a discrete approximation of the solution of the original continuum problem.

Applications that involve a meshing procedure are referred to as mesh-based applications (step 2). Mesh-based applications are naturally suited for parallel or distributed systems because these applications require large amounts of processing time. Furthermore, mesh-based applications can be partitioned to execute concurrently on heterogeneous computers in a Grid. Implementing the finite element method in parallel involves partitioning the nodes global domain into *nprocs* processors. Our example applications use explicit finite element analysis for problems involving sheet stamping and car crashes [19]. We describe each of these below.

Sheet stamping problems. Performance prediction of sheet stamping dies during the die-design process. As well as market pressure for faster delivery of dies and cost reduction, large car manufacturers increasingly tend to offload design responsibility onto their suppliers. Typically, dies are designed by highly experienced people who know what sort of die is needed to produce a part of a given shape. On the basis of the design, fine-tuning is performed by actually using dies to produce parts, observing the result and manually milling the die until the sheet comes out as specified by the contractor. In complex cases, it is very difficult to produce a good die design by intuition. In addition to the associated costs, failure to meet a delivery date damages a company's image and has a negative impact on future business.

Numerical simulations could provide the quantitative information needed to minimize modifications during the manufacturing process [19]. Simulations with serial codes take as long as 40 to 60 processor hours and usually require high end workstations. Parallel stamping simulations enable die manufacturing companies to alter their die design procedures: instead of an iterative approach, they can run more analyses before the first die prototypes are made. This reduces overall die design and manufacturing time, which is vital for the industry.

This problem has high computational requirements. The size of the problem is very large because of the high complexity of the design model. In this case, Grid computing is necessary because a complex model is required to solve the problem and currently supplier companies do not have enough computational power available to perform such simulations. They are forced to use the services of remote computers that belong to other companies.

Car crash problems. The car body design and the passenger's safety must be considered. Before performing a real crash test, hundreds of crash worthiness simulations are computed and analyzed [20]. Car crash simulations are required to predict the effects on new advanced materials of various collisions, such as two cars colliding. As in the above case the car crash problem has high computational requirements and the platform Grid is a good option to realize the simulations at less cost.

3.2 Simulations of Mesh-based Applications

When doing structural analysis of mesh-based applications such as car crash and sheet stamping we use the displacement equation [21]. This equation determines the numerical solution for our applications. The discretization of the displacement equation using the finite element method has the following mathematical equation:

$$[M]\{\mathring{u}\}+[K]\{u\} = \{F^a\} \quad (1)$$

where:

[M] is the mass matrix
[K] is the tensor matrix
$\{\mathring{u}\}$ is the acceleration vector
{u} is the displacement vector and
$\{F^a\}$ is the force vector.

To obtain the numerical solution of (1), we use the central difference method and numerical integration in time, and so obtain the follow equation:

$$\{u_{n+1}\} = [A]\{u_n\} \quad (2)$$

Equation (2) defines the explicit method to determine the numerical solution of our applications. Details of the characteristics of matrix [A] and the $\{u_n\}$ vector can be seen in [21].

3.3 Explicit Simulations

In the previous subsection, we described the general algorithmic structure to determine the numerical solution for our simulations. The simulations have two different schemes, called with or without matrix assembling, depending on whether or not the matrix inside the inner loops is gathering.

The scheme without matrix assembling uses a calculation algorithm element by element, thus it is not necessary to gather a global equation system. The algorithm structure of this scheme is the following:

```
for (all step time of simulation)
 for (all finite elements)
   Obtain global vector associate to finite element ({u_n^e})
   Solve system:  {u_{n+1}^e} = [A^e]{u_n^e}
   Scatter  {u_{n+1}^e} to global vector ({u_{n+1}})

 Calculate residual vector
 Update boundary conditions
 Calculate next step of time
```

As we can see in the above algorithm, the scheme without matrix assembling makes one matrix-vector product operation per element, using a part of the global vector associate to that element ($\{u_n^e\}$). The result obtained ($\{u_{n+1}^e\}$) is scattering to global vector ($\{u_{n+1}\}$). After this, the residual vector is calculated and the boundary conditions are updated if necessary. This scheme has the advantage that the global matrix [A] does not need to be formed in the inner loop. This leads to considerable savings in memory and allows to solve large problems in relatively small memory PC's.

In contrast, the second scheme needs an initial gathering of the global matrix. After this, a matrix-vector product operation is needed as in the first scheme. The algorithm for this scheme is the following:

Our work follows the without matrix assembling scheme to produce the explicit simulations as this saves both memory and time.

```
for (all step time of simulation)
  for (all finite elements)
    Gather  [A^e]: [A] = [A] + [A^e]

  Solve system:  {u_{n+1}} = [A]{u_n}
  Calculate residual vector
  Update boundary conditions
  Calculate next step of time
```

4 DIMEMAS AND GRID ENVIRONMENT

We use a performance simulator called Dimemas. Dimemas is a tool developed by CEPBA[1] for simulating parallel environments [2, 12, 13]. This tool is fed with a trace file and a configuration file. In order to obtain the trace file, the parallel application is executed using an instrumented version of MPI [22]. It is important to note that this execution can be done on any kind of computer. The configuration file contains details of the simulated architecture, such as number of the nodes, latency and bandwidth between nodes. Dimemas generates an output file that contains the execution times of the simulated application for the parameters specified in the configuration file. Furthermore, it is possible to obtain a graphical representation of the parallel execution. Figure 1 shows the sequence of steps to obtain the output file.

The Dimemas simulator considers a simple model for point to point communications. This model breaks down the communication time into five components:

1. Latency time is a fixed time to start the communication.

[1] European Center for Parallelism of Barcelona

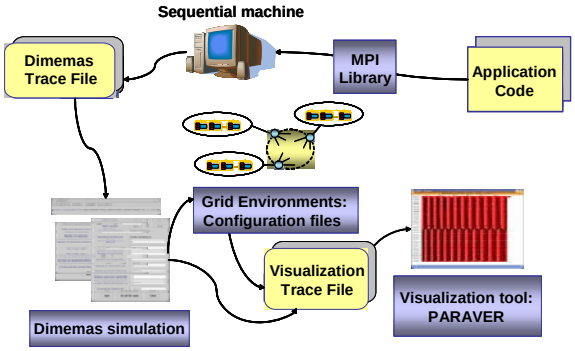

Fig. 1. The Dimemas tool.

2. Resource contention time is dependent on the global load in the local host [23].
3. The transfer time is dependent on the message size. We model this time with a bandwidth parameter.
4. The WAN contention time is dependent on the global traffic in the WAN [24].
5. The flight time is the time spent in the transmission of the message, during which no CPU time is used [23]. It depends on the distance between hosts. We consider hosts distributed at equal distances, since our environment is homogeneous.

We consider an ideal environment one where resource contention time is negligible: there are an infinite number of buses for the interconnection network and as many links as the number of different remote communications the host has with others hosts. For the WAN contention time, we use a linear model to estimate the traffic in the external network. We have considered the traffic function with 1% influence from internal traffic and 99% influence from external traffic. Thus, we model the communications with just three parameters: latency, bandwidth and flight time. These parameters are set according to what is commonly found in present networks. We have studied different works to determine these parameters [24, 25]. Table 1 shows the values of these parameters for the internal and external host communications. The internal column defines the latency and bandwidth between processors inside a host. The external column defines the latency and bandwidth values between hosts. The communications inside a host are fast (latency 25 µs, bandwidth 100 Mbps), and the communications between hosts are slow (latency of 10 ms and 100 ms, bandwidth of 64 Kbps, 300 Kbps and 2 Mbps, flight time of 1ms and 100 ms).

We model a Grid environment using a set of hosts. Each host is a network of Symmetric Multiprocessors (SMP). The Grid environment is formed by a set of connected hosts. Each host has a direct full-duplex connection with any other host. We do this because we think that some of the most interesting Grids for the scientist involve nodes that are themselves high-performance parallel machines or clusters. We consider different topologies in this study: two, four and eight hosts.

5 Data Distribution

This work is based on the use of distributed applications that solve sparse linear systems using iterative methods. These systems arise from the discretization of partial differential equations, especially when explicit methods are used. These algorithms are parallelized using domain decomposition for the data distribution. Each particular domain has a parallel process associated to it.

A matrix-vector product operation is carried out in each iteration of the iterative method. The matrix-vector product is performed using a domain decomposition algorithm, as a set of independent computations and a final set of communications. The communications in this context are associated to the domain boundaries. Each process exchanges the boundary values with all its neighbors so that, each process has as many communication exchanges as neighbor domains [26, 27]. For each communication exchange, the size of the message is the length of the boundary between the two domains involved. We use METIS to perform the domain decomposition for the initial mesh [28, 29, 30].

Balanced distribution pattern. This is the usual strategy for domain decomposition algorithms. It generates as many domains as processors in the Grid. The computational load is perfectly balanced between domains. This balanced strategy is suitable in homogeneous parallel computing, where all communications have the same cost. Figure 2 shows an example of a finite element mesh with 256 degrees of freedom (dofs) with the boundary nodes for each balanced partition. We consider a Grid with 4 hosts and 8 processors per host. We solve an initial decomposition in four balanced domains. Figure 3 shows the balanced domain decomposition.

TABLE 1
Latency, Bandwidth and Flight Time Values

Parameters	Internal	External
Latency	25 µs	10 ms and 100 ms
Bandwidth	100 Mbps	64 Kbps, 300 Kbps and 2Mbps
Flight time	-	1 ms and 100 ms

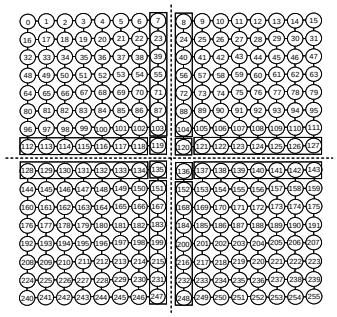

Fig. 2. Boundary nodes per host

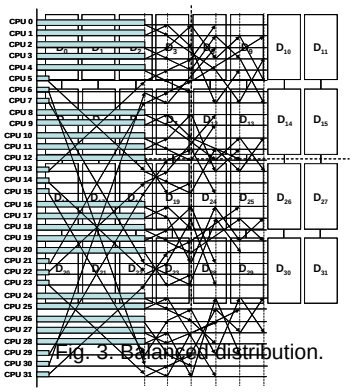

Fig. 3. Balanced distribution.

Fig. 6.c.Communication diagram for a computational iteration of the *MultipleB-domain* distribution.

Unbalanced distribution pattern. Our proposal is to build some domains with a negligible computational load. Those domains are devoted only to manage the slow communications. In order to do this, we divide the domain decomposition in two phases. First, balanced domain decomposition is done between the number of hosts. This guarantees that the computational load is balanced between hosts. Second, unbalanced domain decomposition is done inside a host. This second decomposition involves splitting the boundary nodes of the host sub-graph. We create as many special domains as remote communications. Note that these domains contain only boundary nodes, so they have negligible computational load. We call these special domains *B-domains* (Boundary domains). The remainder host sub-graph is decomposed in *nproc-b* domains, where *nproc* is the number of processors in the host and *b* stands for the number of *B-domains*. We call these domains *C-domains* (Computational domains). As a first approximation we assign one CPU to each domain. The CPUs assigned to *B-domains* remain inactive most of the time. We use this policy to obtain the worst case for our decomposition algorithm. This inefficiency could be solved assigning all the *B-domains* in a host to the same CPU. We consider two unbalanced decomposition of the mesh shows in figure 2. First, we create a sub-domain with the layer of boundary nodes for each initial domain (*singleB-domain*), which contains seven computational domains (Fig. 4). Second, we create some domains (*multipleB-domain*) for each initial partition using the layer of boundary nodes. Then, the remainder mesh is decomposed in five computational domains (Fig. 5).

We must remark that the communication pattern of the

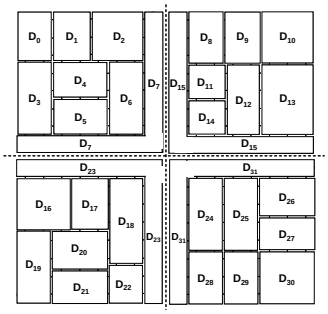

Fig. 4.*SingleB-domain* distribution.

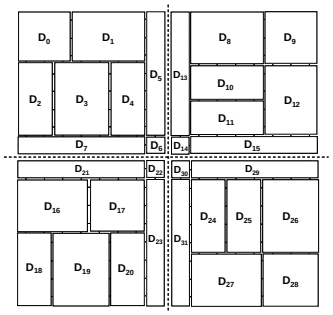

Fig. 5.*MultipleB-domain* distribution.

balanced and the unbalanced domain decomposition may be different, since the number of neighbors of each domain may also be different. Figure 6 illustrates the communication pattern of the balanced/unbalanced distributions for this example. The arrows in the diagram represent processors interchanging data. The beginning of the arrow identifies the sender. The end of the arrow identifies the receiver. Short arrows represent local communications inside a host, whereas long arrows represent remote communications between hosts. In Fig. 6.a, all the processors are busy and the remote communications are done at the end of each iteration. The bars in the diagram represent the computational time to each processor. In Figs. 6.b and 6.c, the remote communication takes place overlapped with the computation. In figure 6.b, the remote communication is overlapped only with the first remote computation. In figure 6.c, all remote communications in the same host are overlapped.

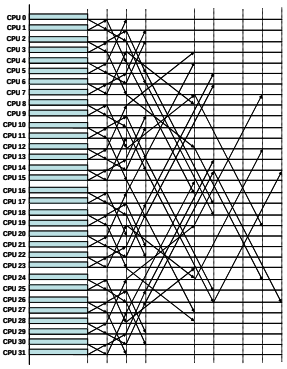

Fig. 6.a. Communication diagram for a computational iterations of the balanced distribution.

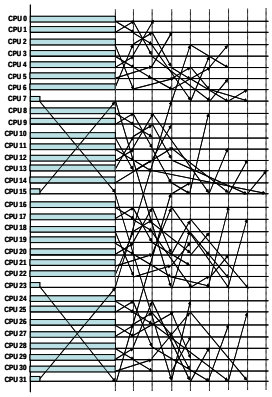

Fig. 6.b.Communication diagram for a computational iteration of the *SingleB-domain* distribution.

6 RESULTS

In this section we show the results obtained using Dimemas. We simulate a 128 processors machine using the following Grid environment. The number of hosts is 2, 4 or 8; the number of CPUs/host is 4, 8, 16, 32 or 64; thus, we have from 8 to 128 total CPUs. The simulations were done considering lineal network traffic models. We consider three significant parameters to analyze the execution time behavior: the communication latency between hosts, the bandwidth in the external network and the flight time.

As data set, we consider a finite element mesh with 1,000,000 dofs. This size is usual for car crash or sheet stamping models. We consider two kinds of meshes, which define most of the typical cases. The first one, called stick mesh, can be completely decomposed in strips, so there are, at most, two neighbors per domain. The second one, called box mesh, cannot be decomposed in strips, so the number of neighbors per domain could be greater than two. The size of the stick mesh is 10^4x10x10 nodes. The size of the box mesh is 10^2x10^2x10^2 nodes.

Figures 7.a, 7.b, 8.a and 8.b show the time reduction percentages compared with the balanced distribution for each Grid configuration in stick mesh as a function of the bandwidth. The unbalanced decomposition reduces the execution time expected for the balanced distribution in most cases.

For a Grid with 2 hosts and 4 processors per host, the predicted execution time of the balanced distribution is better than other distributions because the number of remote communications is two. In this case, the *multipleB-domain* unbalanced distribution has only one or two processors per host computation.

The results are similar when we consider that the external latency is equal to 100 ms (figs. 9.a, 9.b, 10.a and 10.b). Therefore, the value of this parameter has not significant impact on the results for this topology. In the other cases, the benefit of the unbalanced distributions ranges from 1% to 53% of time reduction. The execution time reduction increases until 82% for other topologies and configurations. For 4 and 8 hosts, the *singleB-domain* unbalanced distribution has similar behavior than the balanced distribution, since the remote communications cannot be overlapped and they have to be done sequentially. In this case, the topologies having few processors per computation are not appropriate. The unbalanced distribution reduces the execution time up to 32%.

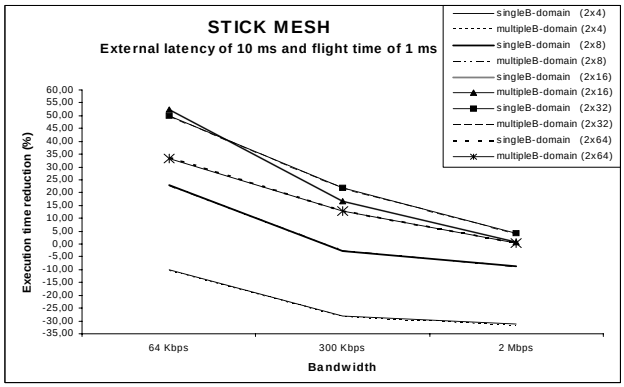

Fig. 7.a. Execution time reduction for the stick mesh with external latency of 10 ms and flight time of 1 ms (2 hosts).

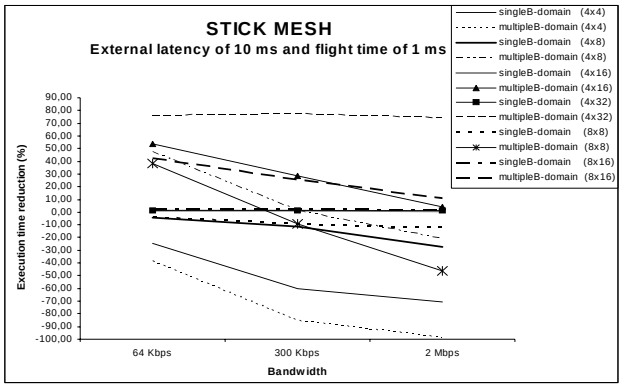

Fig. 7.b. Execution time reduction for the stick mesh with external latency of 10 ms and flight time of 1 ms (4 and 8 hosts).

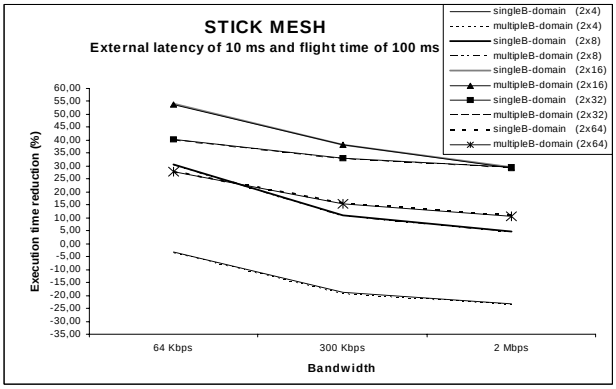

Fig. 8.a. Execution time reduction for the stick mesh with external latency of 10 ms and flight time of 100 ms (2 hosts).

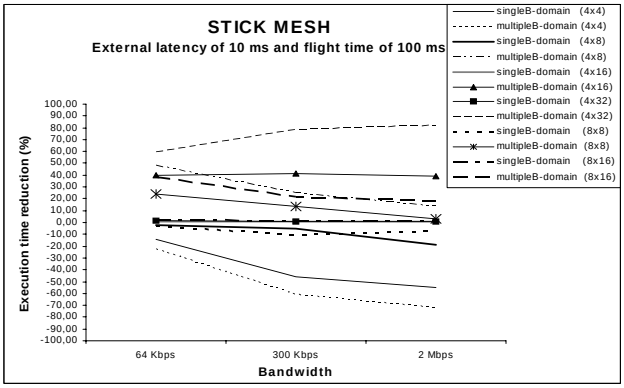

Fig. 8.b. Execution time reduction for the stick mesh with external latency of 10 ms and flight time of 100 ms (4 and 8 hosts).

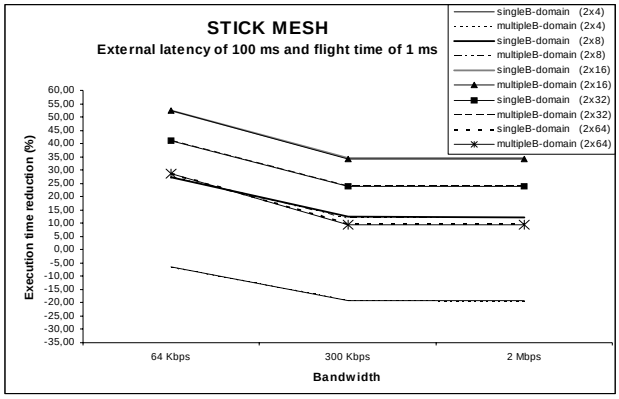

Fig. 9.a. Execution time reduction for the stick mesh with external latency of 100 ms and flight time of 1 ms (2 hosts).

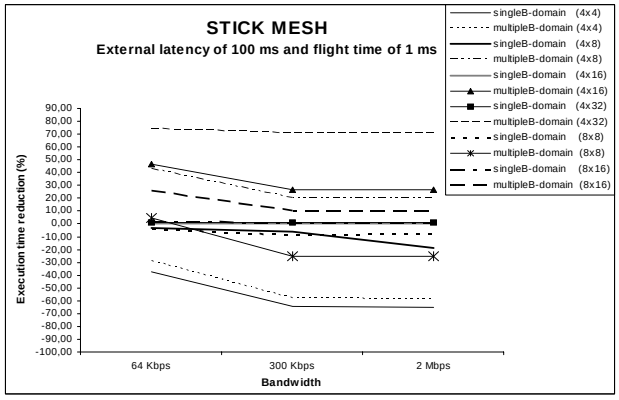

Fig. 9.b. Execution time reduction for the stick mesh with external latency of 100 ms and flight time of 1 ms (4 and 8 hosts).

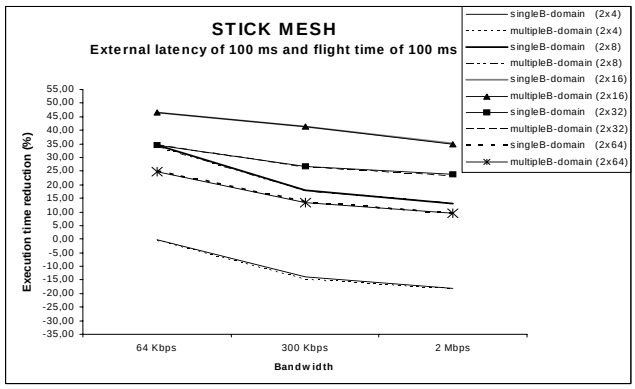

Fig. 10.a. Execution time reduction for the stick mesh with external latency and flight time of 100 ms (2 hosts).

Figures 11.a, 11.b, 12.a and 12.b show the reduction of the expected execution time obtained for each Grid configuration varying the flight time, the external latency and the bandwidth in a box mesh. For the 2 hosts configuration in a box mesh, the behavior for *singleB-domain* and *multipleB-domain* unbalanced distribution is similar, since the number of remote communications is the same. Variations of the flight time and the external latency improve the results up to 85%.

Figures 11.b and 12.b shows the reduction on the expected execution time obtained for 4 and 8 hosts. The in-

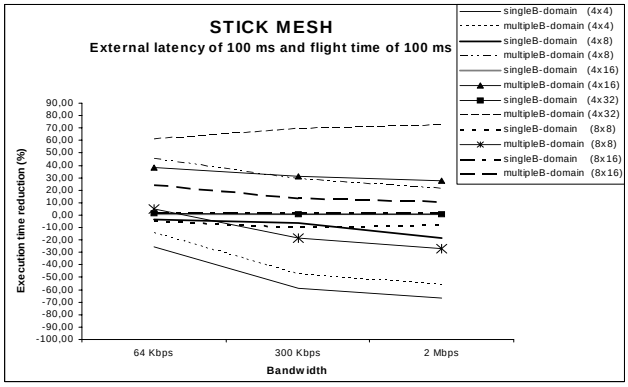

Fig. 10.b. Execution time reduction for the stick mesh with external latency and flight time of 100 ms (4 and 8 hosts).

fluence of the external latency on the application performance in a box mesh increases the percentage of reduction of the execution time up to 4%. We suppose that the distance between hosts is the same. However, if we consider hosts distributed at different distances, we obtain similar benefits for the different distributions. Moreover, if the calculation capacity of each processor in a host is different, the initial data partition will be done consider it. Then the data in each processor will not be the same but

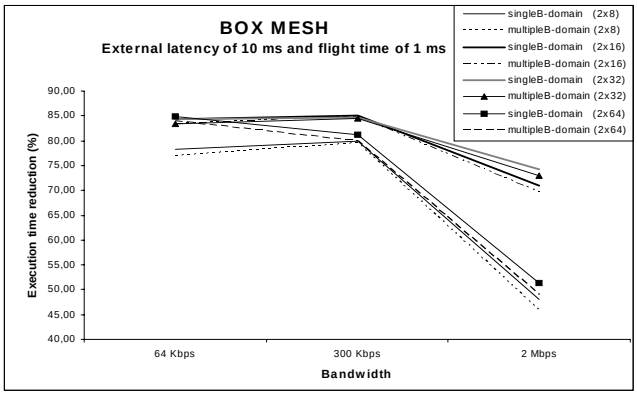

Fig. 11.a. Execution time reduction for the box mesh with external latency of 10 ms and flight time of 1 ms (2 hosts).

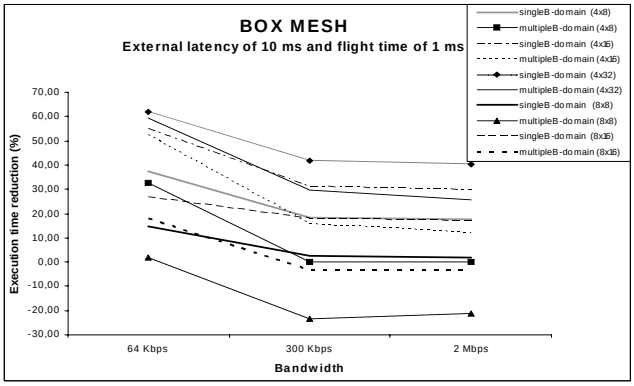

Fig. 11.b. Execution time reduction for the box mesh with external latency of 10 ms and flight time of 1 ms (4 and 8 hosts).

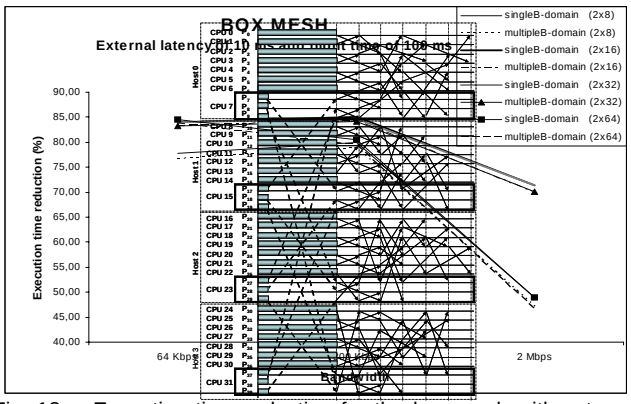

Fig. 12.a. Execution time reduction for the box mesh with external latency of 10 ms and flight time of 100 ms (2 hosts).

Fig. 14. Communication diagram for a computational iteration the *MultipleCB-domain* distribution.

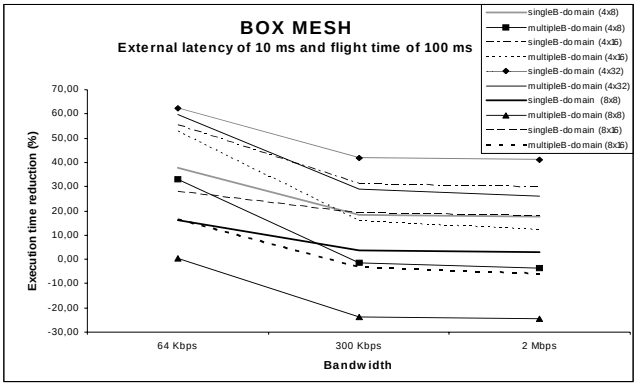

Fig. 12.b. Execution time reduction for the box mesh with external latency of 10 ms and flight time of 100 ms (4 and 8 hosts).

the computational load will be balanced between processors.

The number of remote and local communications varies depending on the partition and the dimensions of the data meshes. Table 2 shows the maximum number of communications for a computational iteration. The number of remote communications is higher for a box mesh than for a stick mesh. Thus, the box mesh suffers from higher overhead.

We propose the use of unbalanced distribution patterns to reduce the number of remote communications required. Our approach shows to be very effective, especially for box meshes. We observe that the *multipleB-domain* with unbalanced distribution is not sensitive to the latency increase until the latency is larger than the computational time. However, the execution time for the balanced distribution increases with the latency.

7 *MULTIPLECB-DOMAIN* DISTRIBUTION

The *multipleB-domain* unbalanced distribution creates as many special domains per host as external communications. Then, the scalability of the unbalanced distribution will be moderated, because a processor is devoted just to manage communications for every special domain. The optimum domain decomposition is problem dependent, but a simple model can be built to approximate the optimum.

TABLE 2
MAXIMUM NUMBER OF COMMUNICATIONS FOR A COMPUTATIONAL ITERATION

	STICK MESH					
	Balanced		*singleB-domain*		*multipleB-domain*	
Host x CPUs	Remote / Local Communication		Remote / Local Communication		Remote / Local Communication	
2x4	1	1	1	1	1	1
2x8	1	1	1	1	1	1
2x16	1	1	1	1	1	1
2x32	1	1	1	1	1	1
2x64	1	1	1	1	1	1
4x4	1	1	2	2	1	3
4x8	1	1	2	2	1	3
4x16	1	1	2	2	1	3
4x32	1	1	2	2	1	3
8x8	1	1	2	2	1	3
8x16	1	1	2	2	1	3
	BOX MESH					
2x4	2	3	1	3	1	3
2x8	4	5	1	6	1	6
2x16	5	8	1	7	1	8
2x32	6	7	1	15	1	14
2x64	7	8	1	25	1	24
4x8	7	5	3	6	4	6
4x16	10	9	3	11	4	9
4x32	9	8	3	22	4	14
8x8	13	5	6	7	13	7
8x16	13	4	6	13	13	11

In addition, to reduce the number of processors performing remote communications in the *multipleB-domain* we propose to assign all *B-domains* in a host to a single CPU, which concurrently will manage all the communications. We will call this unbalanced distribution *multipleCB-domain*. Figures 13 and 14 illustrate the domain decomposition and communication pattern of the *multipleCB-domain* distribution for the example described in section 5.

The main difference between the *multipleB-domain* and *multipleCB-domain* is the amount of domains per host because in the second case all communications are assigned to the same CPU inside a host. In Figure 5, the *multipleB-domain* distribution has 8 data domains per host, now *multipleCB-domain* distribution has 10 data domains per

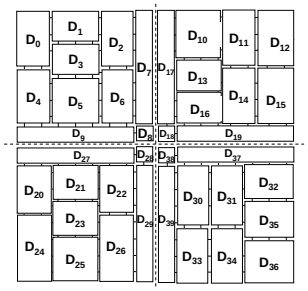

Fig. 13. *MultipleCB-domain* distribution.

host, for three of them will be assigned to one CPU (special domains) into the host and the remainder of data domains will be assigned to the rest of the CPUs. Now we have solely a CPU that manages remote communications and a larger number of CPUs performing computation. This kind of distribution allows us to minimize the number of idle CPUs in a host devoted only to remote communications.

For a Grid with 2 hosts, the predicted execution time is the same that to the *multipleB-domain* because the number of remote communications is only one. However, when considering 4 or 8 hosts, *multipleCB-domain* domain makes a reduction in execution time percentage up to 43% compared to balanced distribution, while *multipleB-domain* distribution makes a reduction percentage up to 53%. In general, *multipleCB-domain* distribution is 10% worse than *multipleB-domain* distribution, mainly due to the problems in managing concurrency remote communications in the simulator.

It is also important to look at the MPI implementation [31]. The ability to overlap communications and computation depends on this implementation. A multithread MPI implementation could overlap communication and computation, but problems with context switching between threads and interferences between processes could appear.

In a single thread MPI implementation we can use non-blocking `send/receive` with a `wait_all` routine. However, we have observed some problems with this approach. The problems are associated with the internal order in no blocking MPI routines for sending and receiving actions. In our experiments, this could be solved programming explicitly the proper order of the communications. But the problem remains for a general case. We conclude that it is very important to have no blocking MPI primitives that actually exploit the full duplex channel capability. As a future work, we will consider other MPI implementations that optimize the collective operations [32, 33].

8 Conclusions

In this paper, we present an unbalanced domain decomposition strategy for solving problems that arise from discretization of partial differential equations on meshes. Applying the unbalanced distribution in different platforms is simple, because the data partition is easy to obtain. We compare the results obtained with the classical balanced strategy used. We show that the unbalanced distribution pattern improves the execution time of domain decomposition applications in Grid environments. We considered two kinds of meshes, which define the most typical cases. We show that the expected execution time can be reduced up to 53%.

The unbalanced distribution pattern reduces the number of remote communications required per host compared with the balanced distribution, especially for box meshes. However, the unbalanced distribution can be inappropriate if the total number of processors is less than the total number of remote communications. The optimal case is when the number of processors making calculation in a host is twice the number of processors managing remote communications. Otherwise, if the number of processors making calculations is small, then the unbalanced distribution will be less efficient than the balanced distribution. In this case, we propose the use of the *multipleCB-domain* distribution. In this distribution all remote communications in a host are concurrently managed by the same CPU. This distribution has around a 10% worse execution time than others unbalanced distributions.

In general, to obtain a good performance in the strategies presented in this paper the number of processors per host needs to be equal or higher than 8. In other case the number of processors performing computation is not enough to overlap remote communications.

Acknowledgments

This work was supported by the Ministry of Science and Technology of Spain under contract TIN2007-60625, the HiPEAC European Network of Excellence and Barcelona Supercomputing Center (BSC).

References

[1] G. Allen et al. "Classifying and enabling grid applications. Concurrency and Computation", *Practice and Experience*, vol.0, pp. 1-13, 2000. (Journal citation)

[2] Dimemas, Internet, http://www.cepba.upc.es/dimemas/. 2000.

[3] J. Chen and V. E. Taylor. "Mesh partitioning for efficient use of distributed systems", *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 1, pp.67-79, 2002. (IEEE Transactions)

[4] C. Walshaw and M. Cross. "Multilevel mesh partitioning for heterogeneous communications networks", *Future Generation Computer Systems*, vol. 17, no. 5, pp. 601-623, 2001. (Journal citation)

[5] F. Pelligrini and J. Roman. "A software package for static mapping by dual recursive bipartitioning of process and architecture graphs", *Proc. of the High Performance Computing and Networking*, pp. 493-498, 1996. (Conference proceedings)

[6] S. K. Das, D. J. Harvey and R. Biswas. "MinEX: a latency-tolerant dynamic partitioner for grid computing applications", *Future Generation Computer Systems*, vol. 18, no. 4, pp. 477-489, 2002. (Journal citation)

[7] S. Huang, E. Aubanel and V. Bhavsar. "Mesh partitioners for computational grids: a comparison", *Computational Science and Its Applications*, LNCS 2269, pp. 60-68, 2003. (Journal or magazine citation)

[8] S. Kumar, S. Das and R. Biswas. "Graph partitioning for parallel applications in heterogeneous grid environments", *Proc. Sixteenth International Parallel and Distributed Processing Symposium*, 2002, doi.ieeecomputersociety.org/10.1109/IPDPS.2002.1015564. (Conference proceedings)

[9] W. D. Gropp and D. E. Keyes. "Complexity of Parallel Implementation of Domain Decomposition Techniques for Elliptic Partial Differential Equations", *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 2, pp. 312-326, 1988. (Journal citation)

[10] D. K. Kaushik, D. E. Keyes and B. F. Smith. "On the Interaction of Architecture and Algorithm in the Domain-based Parallelization of an Unstructured Grid Incompressible Flow Code", *Proc. Tenth International Conference on Domain Decomposition Methods*, pp. 311-319, 1997. (Conference proceedings)

[11] W. Gropp et al. "Latency, Bandwidth, and Concurrent Issue Limitations in High-Performance CFD", *Proc. First Mit Conference on Computational Fluid and Solid Mechanics*, pp. 830-841, 2001. (Conference proceedings)

[12] R. M. Badia et al. "Dimemas: Predicting MPI Applications Behavior in Grid Environments", *Proc. of the Workshop on Grid Applications and Programming Tools GGF8*, 2003. (Conference proceedings)

[13] R. M. Badia et al. "Performance Prediction in a Grid Environment", *Proc, First European across Grid Conference*, 2003. (Conference proccedings)

[14] Y. Li and Z. Lan. "A Survey of Load Balancing in GridComputing. Computational and Information Science", *Proc. First International Symposium (CIS'04)*, pp. 280-285, 2004. (Conference proceedings).

[15] B. Otero et al. "A Domain Decomposition Strategy for GRID Environments", *Proc. Eleventh European PVM/MPI Users' Group Meeting*, pp. 353-361, 2004. (Conference proceedings)

[16] B. Otero and J. M. Cela. "A workload distribution pattern for grid environments", *Proc. the 2007 International Conference on Grid Computing and Applications*, pp. 56-62, 2007. (Conference proceedings)

[17] B. Otero et al. "Performance Analysis of Domain Decomposition", *Proc. Fouth International Conference Grid and Cooperative Computing*, pp. 1031-1042, 2005. (Conference proceedings)

[18] B. Otero et al. "Data Distribution Strategies for Domain Decomposition Applications in Grid Environments", *Proc. Sixth International Conference on Algorithms and Architecture for Parallel Processing*, pp. 214-224, 2005. (Conference proceedings)

[19] W. Sosnowski. "Flow Approach-Finite Element Model for Stamping Processes versus Experiment", *Computer Assisted Mechanics and Engineering Sciences*, vol. 1, pp. 49-75, 1994. (Journal citation)

[20] N. Frisch et al. "Visualization and Pre-processing of Independent Finite Element Meshes for Car Crash Simulations", *The Visual Computer*, vol. 18, no. 4, pp. 236-249, 2002. (Journal citation)

[21] Z. H. Zhong. *Finite Element Procedures for Contact-Impact Problems*. Oxford University Press, pp.1-372, 1993. (Book style)

[22] Paraver. http://www.cepba.upc.es/dimemas. 2002.

[23] R. M. Badía et al. "DAMIEN: Distributed Applications and Middleware for Industrial Use of European Networks". D5.3/CEPBA. IST-2000-25406, unpublished. (Unpublished manuscript)

[24] R. M. Badía et al. "DAMIEN: Distributed Applications and Middleware for Industrial Use of European Networks". D5.2/CEPBA. IST-2000-25406, unpublished. (Unpublished manuscript)

[25] B. Otero and J. M. Cela. "Latencia y ancho de banda para simular ambientes Grid", Technical Report TR-UPC-DAC-2004-33, UPC. España, 2004. (Technical report with report number)
http://www.ac.upc.es/recerca/reports/DAC/2004/index,ca.html.

[26] D. E. Keyes. "Domain Decomposition Methods in the Mainstream of Computational Science", *Proc. Fourteenth International Conference on Domain Decomposition Methods*, pp. 79-93, 2003. (Conference proceedings)

[27] X. C. Cai. "Some Domain Decomposition Algorithms for Nonselfadjoint Elliptic and Parabolic Partial Differential Equations", Technical Report TR- 461, Courant Institute, NY, 1989. (Technical report with report number)

[28] K. George and K. Vipin K. "Parallel multilevel k-way partitioning scheme for irregular graphs", *SIAM Rev.*, vol 41, no. 2, pp. 278-300, 1999. (Journal citation)

[29] K. George and K. Vipin. "A fast and high quality multilevel scheme for partitioning irregular graphs", *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359-392, 1998. (Journal citation)

[30] Metis, Internet, http://glaros.dtc.umn.edu/gkhome/views/metis. 2011.

[31] Message Passing Interface Forum, MPI-2: Extensions to the MPI, 2003. http://scc.ustc.edu.cn/zlsc/cxyy/200910/W020100308601028317962.pdf (2011)

[32] N. Karonis, B. Toonen and I. Foster. "Mpich-g2: A Grid-enabled Implementation of the Message Passing Interface", *Journal of Parallel and Distributed Computing*, vol. 63, no. 5, pp. 551-563, 2003. (Journal citation)

[33] I. Foster and N. T. Karonis. "A Grid-enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems", *Proc. of the ACM/IEEE Supercomputing*, 1998. (Conference proceedings)

B. Otero received her M.Sc. and her first Ph.D. degrees in Computer Science at University of Central of Venezuela in 1999 and 2006, respectively. After that, she received her second Ph.D. in Computer Architecture and Technology in 2007 at Polytechnic University of Catalonia (UPC). Currently, she is an Assistant Professor at the Computer Architecture Department at UPC. Her research interests include parallel programming, load balancing, cluster computing, and autonomic communications. She is member of HiPEAC Network of Excellence.

M. Gil is an Associate Professor at the Universitat Politècnica de Catalunya (UPC). She received her Ph.D. in computer science from the UPC in 1994. Her research is primarily concerned with the design and implementation of system software for parallel computing, to improve resource management. Her work focus mainly in the area of OS, middleware and runtime multicore architectures support. She is member of HiPEAC Network of Excellence and in the SARC European project.