Mesh Partitioning for Parallel Computational Fluid Dynamics Applications on a Grid

Youssef Mesri* — **Hugues Digonnet**** — **Hervé Guillard***

* *Smash project, Inria-Sophia Antipolis, BP 93, 06902 Sophia-Antipolis Cedex, France {Youssef.Mesri,Herve.Guillard }@sophia.inria.fr*

** *CEMEF, Ecole des mines de Paris, BP 207, 06904 Sophia-Antipolis Cedex, France {Hugues.Digonnet@ensmp.fr}*

ABSTRACT. *The problem of partitioning unstructured meshes on a homogeneous architecture is largely studied. However, existing partitioning schemes fail when the target architecture introduces heterogeneity in resource characteristics. With the advent of heterogeneous architecture as the Grid, it becomes imperative to study the partitioning problem taking into account the heterogeneous platforms. In this work, we present a new mesh partitioning scheme, that takes into account the heterogeneity of CPU and networks. Our load balancing mesh partition strategy improves the performance of parallel applications running in a heterogeneous environment. The use of these techniques are applied to some model problems in CFD. Experimental results confirm that these techniques can improve the performance of applications on a computational Grid.*

RÉSUMÉ. *La parallélisation des méthodes de volumes ou d'éléments finis repose sur des techniques de décomposition de domaines qui imposent un partitionnement de maillage préalable aux calculs. Ce problème a été largement étudié pour des architectures homogènes où tous les CPU sont identiques et reliés par un réseau rapide. Avec l'émergence d'architectures hétérogènes de type grilles de calcul constituées d' une aglomération de clusters et de processeurs géographiquement distribués et reliés par des réseaux hétérogènes, il est devenu prímordial de re-éxaminer ce problème en prenant en compte les caractéristiques de ces plate-formes en terme de CPU et de réseau. Dans ce travail, nous présentons un nouveau schéma de partitionnement de maillages, prenant en compte l' hétérogéneité des CPU et du réseau. Cette stratégie d'équilibrage de partitions améliore les performances des applications tournant dans un environement de type grille.*

KEYWORDS: *Grid computing, mesh partitioning, Distributed computing, Load balancing, performance study.*

MOTS-CLÉS : *Maillages non-structurés, Méthodes éléments finis, volumes finis, Partitionnement de maillages, Calcul sur Grille, Calcul distribué, équilibrage de charge*

1. Introduction

Computer simulations are becoming increasingly important as the only means for the design and interpretation of many different process. The scope and accuracy of these simulations are severely limited by available computational power, even using to-day's most powerful supercomputers. We can break through these limits by simultaneously harsening multiple networked supercomputers running single massively parallel simulation to carry out more complex and high-fidelity simulations. This is the basic idea that, since the mid 1990s, made developed the so-called computational grid. Computational grid concept provides the means for coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The Grid concept extends older concepts of distributed computing such as the cluster-computing, but in contrast to older systems the Grid allows resources to be allocated to computing needs on an ad hoc basis.
In the last years there was a wide-spread acceptance of Grid computing that bring to growth of several projects. One of them is the MecaGRID[1] project between INRIA Sophia Antipolis, CEMEF(Centre de mise en forme des materiaux de l'Ecole des Mines de Paris-Sophia Antipolis), and the IUSTI(Institut Universitaire des systemes Thermiques et Indistriels) in Marseille. The aim of the project is to build a computational grid devoted to fluid dynamics applications, using a set of clusters interconnected by a wide area network. Mesh applications are a class of problems that requires such high-end computational power since performance must be scalable into hundreds of processors considering the current technology ([BAR 99]),([DJO 00]).
The successful deployment of compute-intensive applications in a grid environment such as the Grid of the MECAGRID project, involves efficient partitioning on a truly heterogeneous distributed architecture which makes no assumptions on the computing resources. As more remote resources are added, the heterogeneity of the computing platform also grows.
In this paper, we demonstrate how diversity in the architecture characteristics affects the efficiency of mesh partitioning.
In the literature, they are several algorithms for solving the graph partitioning problem for homogeneous architecture models. Partitioning schemes such as Metis ([KAR 98]), Chaco ([HEN 94]), and Jostle([WAL 98]) employ multilevel strategies but fail to address the limitations imposed by heterogeneity in the underling system. These partitioning scheme assume the processing speed of the computing resources to be uniform and the communication network connecting the resources to be of equal capacity.
We propose a heterogeneous partitioner, called MeshMigration that takes into account the architecture characteristics. MeshMigration generates a high quality partition and provides a load balance on each processor of the heterogeneous architecture. We have tested MeshMigration with realistic meshes and results shown that our approach provides an efficient partitioning on a grid platform and minimize the application execution time.
This paper is organized as follows: In section 2, we describe the workload model, fol-

1. http://www-sop.inria.fr/smash/

lowed by a model of distributed heterogeneous architectures in section 3. In section 4, we formally define the partitioning problem. Section 5 discusses Meshmigration, the proposed local heterogeneous partitioner and its complexity analysis. In section 6, we experimentally study the performance of MeshMigration for realistic workloads. Finally in section 7, we will describe preliminary experimental results for a finite element code executed on the Grid.

2. Workload model

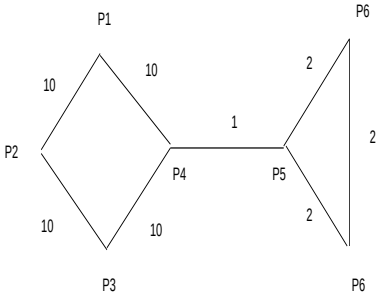
Computational fluid dynamics (CFD) applications usually operate on a huge set of application data associated to unstructured meshes. For this reason CFD problems represent a significant part of high performance supercomputing applications. Finite element and finite volume methods use unstructured meshes. However, depending on the characteristics of the discretization method, the work flow graph representing the application can be either the nodal graph (mesh nodes), dual graph(mesh element), the combination of both, or some special purpose presentation. In this work based on the previous work of ([BAS 00]), we consider the combined graph for modeling the FE and FV application.
We represent the application as a weighted undirected graph $W = (V(W), E(W))$, which we will call the *workload graph* . Each vertex v has a computational weight $\omega(v)$, which reflects the amount of the computation to be done at v. An edge between vertices u and v, denoted $\{u, v\}$, has a computational weight $\omega(\{u, v\})$, which reflects the data dependency between them.

3. Heterogeneous architecture model

Partitioning applications onto heterogeneous architecture such as a Grid environment requires a special model architecture that reflects both heterogeneous resource characteristics and also non-homogeneous communication network between these different resources.

The machine architecture can be represented as a weighted undirected graph $A = (V(A), E(A))$, which we will call the architecture graph. It consists in a set of vertices $V(A) = \{p_1, p_2, ..., p_n\}$, denoting processors, and a set of edges $E(A) = \{\{p_i, p_j\} | p_i, p_j \in P\}$, representing communication links between processors. Each processor p has a processing weight s_p, modeling its processing power per unit of computation. Each link has a link weight v_{pq}, that denotes the communication bandwidth per unit of communication between processors p and q.
In this work, we will assume that the machine architecture can be represented by a complete graph, that is we will assume that given any two processors p and q, there always exist a path connecting them even if they are not directly connected by a physical link. In this case, the weight of the edge $\{p, q\}$ will be evaluated as the minimum of the link weights on the shortest path connecting p and q. The Matrix (1) gives for

instance the communication weights matrix corresponding to the adjacent architecture graph.



Architecture graph

$$C = \begin{pmatrix} \times & 10 & 10 & 10 & 1 & 1 & 1 \\ 10 & \times & 10 & 10 & 1 & 1 & 1 \\ 10 & 10 & \times & 10 & 1 & 1 & 1 \\ 10 & 10 & 10 & \times & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & \times & 2 & 2 \\ 1 & 1 & 1 & 1 & 2 & \times & 2 \\ 1 & 1 & 1 & 1 & 2 & 2 & \times \end{pmatrix} \quad (1)$$

Communication Matrix

4. Partitioning Problem

We consider a workload graph $W(V(W), E(W))$ which represents the application, and a architecture graph $A(V(A), E(A))$ which represents the Grid. On a computational grid, the machine architecture is heterogeneous both for network and processors. So we consider the characteristics of architecture graph to define the partitions. A mapping of a workload graph onto a architecture graph can be formally described by:

$$m \ : \ V(W) \longrightarrow V(S) \quad (2)$$

where $m(v) = p$, if the vertex v of W is assigned to processor p of A.
In order to evaluate the quality of a mapping, we define two cost models: one for estimating the computational cost and the other one for the communication cost evaluation.

4.1. *Computational cost*

For each mapping of the workload graph onto the architecture graph we can estimate the computational cost as follows: If a vertex v is assigned to a processor p, the computational cost is given by $t_p^v = \omega(v)/s_p$, that is the ratio of the computational weight of v per the processing weight of p. Computational cost estimates the time units required by p to process v.

4.2. *Communication cost*

The communication cost is introduced when we have a data communication transfer between two different nodes in the target graph. Suppose $\{u, v\} \in E(W)$ and $u \in V(W)$ is assigned to processor p and $v \in V(W)$ is assigned to processor q. The data is transfered from the local memory of p to the local memory of q via message passing. In this case, the communication cost is given by $c_{pq}^{u,v} = \omega(\{u, v\})/v_{pq}$, that is the ratio of the communication weight of edge $\{u, v\}$ per the link weight between p and q. The communication cost represents the time units required for data transfer between the vertices u and v.

4.3. *Cost function*

Let $m : V(W) \longrightarrow V(A)$ be a mapping of $W(V)$ onto $V(A)$, the weight of subgraph assigned to a processor p in $V(A)$, is just the sum of the weights of the vertices in the subgraph: $C(p, m) = \sum_{v \in V(A), m(v)=p} \omega(v)$. For all p in $V(A)$, the computational time is given by $t_p = \frac{C(p,m)}{s_p}$, where $C(p, m)$ is defined above and s_p is the processing weight.
Let $\{p.q\} \in E(A)$, we define the communication cost associated to the processors p and q as:

$$\mathcal{C}(\{p, q\}, m) = \sum_{\substack{m(u)=p \\ m(v)=q \\ \{u,v\} \in E(W)}} c_{pq}^{u,v}$$

The total communication time associated to processor p is defined by:
$\mathcal{C}(p, m) = \sum_{i \in V(A) \neq p} \mathcal{C}(\{p, q\}, m)$. To evaluate the quality of the mapping, we define a cost function as follows: $\Phi(W, A, m) := T+C$ where $T = {}^t\big(t_1, \ldots, t_{card(V(A))}\big)$ and $C = {}^t\big(\mathcal{C}(1, m), \ldots, \mathcal{C}(card(V(A)), m)\big)$. The definition of the graph partitioning problem is to find a partition (mapping m) which minimizes the cost function $\Phi(W, A, m)$. Clearly, the problem is extensible to the classical graph partitioning and task assignment problem, and it is well known that this problem is NP-complete. In the next section, we describe the iterative algorithm chosen to minimize this cost function and find the efficient partitioning.

5. Heterogeneous local method of parallel repartitioning

MeshMigration is a graph/mesh repartitioning scheme developed for heterogeneous architectures such as the Grid. We employ a local method of parallel repartitioning developed during the DRAMA project ([BAS 00]).
The principal steps of this strategy are the followings:
- Form disjoint pair of processors that will present an important gain for the cost function (see section 5.2).

- Optimization the mapping on each pair formed: This optimization is performed by transferring vertices (elements and nodes) from one processor to another by using the notion of strip migration (see section 5.1).

- The two previous steps are iterated as long as we are able to globally optimize the partition.

5.1. *Strip migration*

Let (p, q) be two processors and let
$\mathcal{I}_{p.q} = \{\{u, v\} \in E(W)/m(u) = p \quad \text{and} \quad m(v) = q\}$ be the interface between the processors p and q. For any w such that $m(w) = p$ (resp. q) the topological distance of w from the interface is defined as the shortest path between w and a vertex of the interface in the mesh nodal graph (if w is a node of the mesh) or in the mesh dual graph (if w is an element). We then define, a strip as the set of nodes and elements that have the same topological distance from the interface. The optimization of the partition is then performed by transferring strips from processor p to processor q in order of increasing topological distance and as long as this transfer improves the cost function.

5.2. *Formation of processor pairs*

The goal of this algorithm is to perform a parallel and automatic clutch of processors. It provides the maximum pairs of processors which consent to optimize the partitions.
If we consider a pair of processors (p, q), the cost function of initial partition between p and q is given by: $F_0|_{pq} = max(t_p, t_q) + \mathcal{C}(\{p, q\}, m)$ where $\mathcal{C}(\{p, q\}, m) = \mathcal{C}(\{q, p\}, m)$. To improve the initial partition, we evaluate the cost function strip per strip as long as we find the best strip associated to the minimum of the cost function on p, denoted F^p_{min} and on q, F^q_{min}.
Then, we define a Friendship function between the processors p and q which is given by the maximal potential gain:

$$\textit{Friendship}(p, q) = max\Big(F_0|_{pq} - F^p_{\text{min}}, F_0|_{pq} - F^q_{\text{min}}\Big) \tag{3}$$

The first pair of processors formed is given by:
$\textit{Friendship}(p, q) = max(\textit{Friendship}(i, j))$, for all i and j in $V(A)$.
The migration between p and q is determined as follows:
if $(F_0|_{pq} - F^p_{\text{min}}) > (F_0|_{pq} - F^q_{\text{min}})$ (resp. $(F_0|_{pq} - F^p_{\text{min}}) < (F_0|_{pq} - F^q_{\text{min}})$), the elements and nodes having a distance lower than the best strip distance associate to F^p_{min} (resp. F^q_{min}) are migrated from p to q (resp. from q to p).

5.3. *Partitioning strategy*

In order to provide an efficient partitioning in a grid environment, we introduce a hierarchical view-point in this algorithm. We define two architecture levels in the target graph: level one is the set of processors, and the second is the set of clusters. Then, the partitioning is carried out in two stages:
1- We consider a grid composed by several clusters. We decompose the mesh in order to assign one partition to each cluster taking into account the bandwidth of communications layers between these various clusters. The decomposition takes place via the relationship defined in the previous paragraph. We denote N numbers of clusters and $G = \{C_0, ..., C_{N-1}\}$ the set of clusters. Formally:
Let p and q two processors in the architecture graph, $p \in C_i$ and $q \in C_j$:

$$\begin{cases} \text{if} \quad C_i \neq C_j \\ \qquad \textit{Friendship}(p,q) = max\Big(F_0|_{pq} - F^p_{\text{min}}, F_0|_{pq} - F^q_{\text{min}}\Big) \\ \text{else} \quad \textit{Friendship}(p,q) = 0 \end{cases} \tag{4}$$

2- We denote $\Pi = \{\pi_0, ..., \pi_{N-1}\}$, the set of the sub-domains mapped on G. For every partition π_i assigned to the cluster $C_i = \{p^i_0, ..., p^i_{I-1}\}$, where I the number of processors of C_i, we re-partitioned π_i on the set of processors of C_i:
For every p and q two processors on the architecture graph, $p \in C_i$ and $q \in C_j$:

$$\begin{cases} \text{if} \quad C_i = C_j \\ \qquad \textit{Friendship}(p,q) = max\Big(F_0|_{pq} - F^p_{\text{min}}, F_0|_{pq} - F^q_{\text{min}}\Big) \\ \text{else} \quad \textit{Friendship}(p,q) = 0 \end{cases} \tag{5}$$

The strategy adopted reduces the complexity of the algorithm.

6. Performance results

In this section, we present practical results of the proposed heterogeneous partitioning method on a grid environment. For our experiments, we consider a jet in cross-flow(JICF) mesh often used to simulate fluid dynamics phenomena (e.g. injections for cooling systems, jets for V-STOL aircraft, exhaust of vehicles). This mesh(3D) consists of 400 thousand nodes and 3.8 millions elements(tetrahedral). In table 1, we show a set of results obtained from a set of architecture graph. We use two clusters, pf machines(processor speed 1GHz, bandwidth intra-cluster 100Mb/s) and nina machines (processor speed 2GHz, bandwidth intra-cluster 1Gb/s), the bandwidth between two clusters is 100Mb/s. The number of processors varies from 2 to 32 processors. The parameters used in this table are described as follows:
PT: the partitioning time is the total time (read input file + partitioning + write output

files) needed by the MeshMigration partitioner.
Φ: cost function
$Max(Min)$ elements: the maximum(minimum) number of elements for all partitions.
$\lambda = \frac{max(t_p)}{min(t_p)}$: the load imbalance, where $t_p = \frac{C(p,m)}{s_p}$. To simplify the presentation of the results, we consider $C(p, m) =$ nb_Elp, where nb_Elp is the number of elements assigned to processor p. We have a balanced load when $\lambda \simeq 1$.

Partitioner	Cost functions	1pf-1nina	2pf-2nina	4pf-4nina	8pf-8nina	16pf-16nina
Homog.	Φ	1.37232e+06	699951	352804	266230	893602
	PT(seconds)	589.562	639.762	856.102	967.167	1155.21
	Max elements	1166697	585120	293254	148427	75106
	Min(t_p)	486123.75	243800	122189.16	61844.58	31294.16
	Min elements	1166194	581132	288675	143497	75066
	Max(t_p)	1166194	581132	288675	143497	75066
	λ	2.39	2.38	2.36	2.32	2.39
Heterog.	Φ	805537	421440	217025	137211	59278.7
	PT(seconds)	339.615	390.888	465.138	739.666	758.149
	Max elements	1648787	847293	427327	221036	114330
	Max(t_p)	686994.58	353038.5	178052.91	92098.33	47637.5
	Min elements	684104	328452	166263	78630	40720
	Min(t_p)	684104	328452	166263	78630	40720
	λ	1.004	1.07	1.07	1.17	1.17

Table 1. *Comparison between Homogeneous and Heterogeneous partitioning, for the JICF test case partitioned in 2 to 32 processors*

From the table 1, we can extract the following remarks:
- About cost function (Φ):
The homogeneous partition does not optimize the cost function, because the homogeneous partitioner does not takes into account the machine architecture. Otherwise, while increasing the number of processors, the number of interfaces between partitions increases, furthermore, the produced communication penalize the homogeneous approach too much. However, the heterogeneous approach significantly decreases the cost function that estimates the execution time. The figure[figure 1], clearly shows the variation of the cost function in the two homogeneous and heterogeneous cases according to the number of processors.
- About the time partitioning (PT) and load balancing:
The strip migration method used in the heterogeneous approach allows to accelerate the partitioning time as it is shown in the table 1. It is also possible to see the efficiency of our approach on the level of the load balancing, the parameter λ is nearly equal to one for the heterogeneous partitions, but not balanced for the homogeneous partitions.

After the comparison of the heterogeneous and homogeneous methods, we study now,

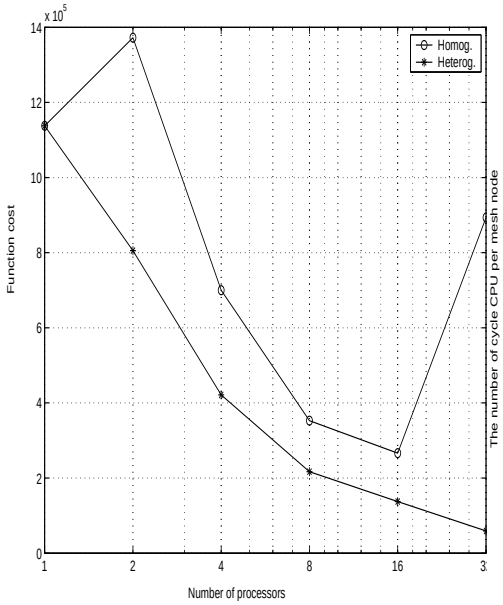


figure 1: Evaluation of the cost function

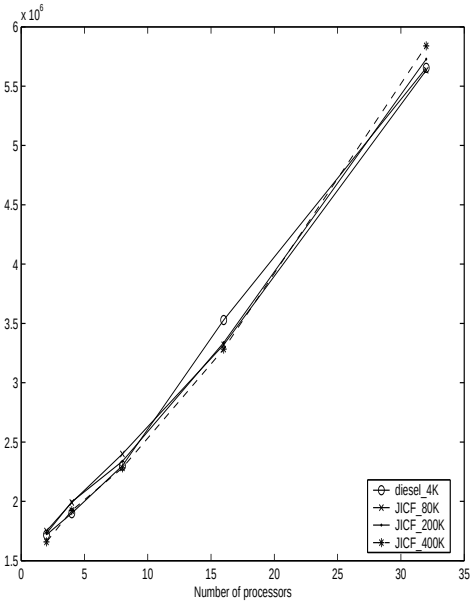


figure 2: Four different mesh sizes

the scalability of our approach. We present test meshes with different size: a diesel_4K mesh with 4 thousand nodes, a JICF_80K mesh with 80 thousand nodes, a JICF_200K mesh with 200 thousand nodes and a JICF_400K mesh with 400 thousand nodes. We compute the partitioning time (PT) needed to partitioning different meshes on a architecture when the number of processors varies from 2 to 32 processors, then we compute the ratio of the number of CPU cycles per the number of nodes for each mesh.

In figure [2], we show the curves that represent the number of CPU cycles needed to process one node according to the number of processors. It is seen that all curves are linear and almost identical, that means that, independently of the type and size of mesh, there is a linear behavior of the partitioner: the partitioning cost is linear according to the number of partitions (number of processors) and also linear according to the number of mesh nodes. This result, allows to predict the necessary time for partitioning any other type of mesh and for any number of partitions.

In figure [3.1],[3.2] and [3.3], we present the JICF_400K mesh partitioned on 16 partitions, 8 partitions assigned to nina cluster and 8 partitions assigned to pf cluster. The used architecture graph is composed of 8 nina machines and 8 pf machines.

In this figure, we observe that interfaces between the two sites is reduced at the intersection between the pipe and the rectangle, there is therefore a strong minimization of the communications on a weak link (in relation to the intra-sites networks) moreover,

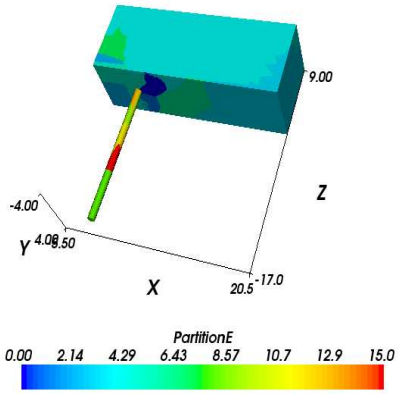


figure 3.1: Heterogeneous partitioning, for the JICF test case partitioned in 16 processors(8 nina, 8 pf)

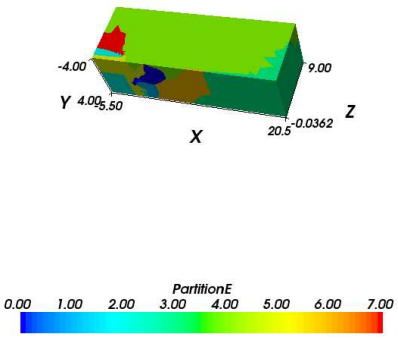


figure 3.2: 8 Partitions in nina cluster

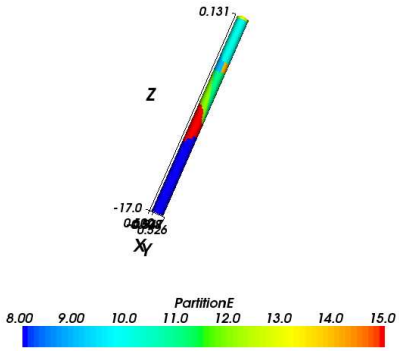


figure 3.3: 8 Partitions in pf cluster

the load is balanced on each processor.

7. Experimentation of a finite element code on the grid

In order to validate this approach, we run a simple finite element code. This code solves the Stokes equations in a domain Ω :

$$\begin{cases} \nabla.(2\eta\epsilon(v)) - \nabla p &= 0 \\ \nabla.v &= 0 \end{cases} \tag{6}$$

with the boundary conditions :

$$\left\{ \begin{array}{lcll} p & = & p_0 & \text{on} \quad \Gamma_{in} \\ p & = & 0 & \text{on} \quad \Gamma_{out} \\ v & = & 0 & \text{on} \quad \Gamma - \Gamma_{in} - \Gamma_{out} \end{array} \right. \tag{7}$$

These equations are solved using the mixed-element P1+/P1 (linear interpolation over the elements for the velocity and the pressure with a bubble interpolation for the velocity). The bubble unknown is condensed and lead to a global linear system with only 4 unknowns per node of the mesh representing the domain Ω. The mesh is partitioned using the partitioner previously detailed, this partition allows us to distribute the linear system over the processors. The resolution is then done in parallel with the PETSc library using a conjugate residual resolution with a incomplete LU preconditioner.

The table 2 compares the results obtained on the JICF(400K) test case when using 32 processors (16 PF and 16 NINA) with homogeneous and optimized partitions.

PF-NINA	results
Nb iter	927
Resolution (s)	339.97
Assembling (s)	9.40651
Solver (s)	349.37
Total (s)	349.42

Homogeneous partition

PF-NINA	results
Nb iter	850
Resolution (s)	174.22
Assembling (s)	6.61217
Solver (s)	180.83
Total (s)	180.86

Optimized partition

Table 2. *Influence of an optimized partition against an homogeneous one*

This shows that we are able to reduce the calculation time of about 50% by considering the characteristics of the grid.

8. Conclusion

In this paper, we have presented a new graph/mesh partitioner, called MeshMigration, for partitioning workloads graph onto heterogeneous architecture graph. This mesh partitioner algorithm execute in parallel and we have shown that his execution time is linear with respect to the number of processors and the size of the mesh. We have also shown that optimized load balancing strategies improves the performance of the applications executed on a heterogeneous environment.

Acknowledgements

This work has been performed in the framework of the ACI-GRID 2002 MecaGrid of the French Ministry of Research.

9. References

[BAR 99] BARNARD S., BISWAS R., SAINI S., VAN DER WIJNGAART R., YARROW M., ZECHTER L., FOSTER I., LARSSON O., "Large-scale Distributed Computational Fluid Dynamics on the information Power Grid using Globus", 7th symp. on the Frontiers of Massively Parallel Computation, Annapolis, MD, Feb 1999, p. 60-67.

[BAS 00] BASERMANN A., MAETEN B., ROOSE D., FINBERG J., LONSDALE G., "Dynamic load balancing of finite element applications with the DRAMA library", vol. 25(2), 2000, p. 83-98.

[DJO 00] DJOMETRI M., BISWAS R., VAN DER WIJNGAART R., YARROW M., "Parallel and Distributed Computational Fluid Dynamics: Experimental Results and Challenges", vol. 1970, Bangalore, India, Dec 2000, p. 183-193.

[HEN 94] HENDRICKSON B., LELAND R., "The Chaco User's Guide, version 2.0", Technical report sand94-2692, 1994, Sandia National Laboratories.

[KAR 98] KARYPIS G., KUMAR V., "MeTiS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0", report , 1998, University of Minnesota, Dept. of Computer Science and Engineering.

[WAL 98] WALSHAW C., "Parallel Jostle Userguide,and V. Kumar", report , 1998, University of Greenwich, London, UK.

Computational Fluid Dynamics (CFD) is a discipline that has always been in the vanguard of the exploitation of emerging and developing technologies. Advances in both algorithms and computers have rapidly been absorbed by the CFD community in its quest for more accurate simulations and reductions in the time to solution.Â Three-dimensional simulation on a parallel computer of supersonic coflowing jets (O. Louedin, J. Ryan). Navier-Stokes algorithm development within the FAME mesh environment (S.H. Onslow et al.). Partitioning and parallel development of an unstructured, adaptive flow solver on the NEC-SX4 (H. van der Ven, J.J.W. van der Vegt). Distributed Computing. Parallel Computation: Models and Methods review ed by Daniel B. Szyld, Tem ple Universit y Parallel Comput at ion: This is a well-written book suitable for classroom weather. The fact that nowadays we can run such M odels and M et hods use at the senior, or beginning-graduate level, in programs in a fraction of that time is due in part By Selim G. Akl computer science or computer engineering. to the use of parallel computers and to the 608 pages Parallel computers are increasingly being numerical methods designed for.Â The bibliography is fairly They include shared-memory machines, els: for the PRAM in chapters 3â€“6, for complete, with over 650 entries. which the author calls PRAMs (parallel linear arrays in Chapter 7, for meshes in Finally, I want to mention a nice dis... Computational Fluid Dynamics (CFD) is a method to solve problems related to uids numerically. There are numerous studies applying a variety of CFD solvers to solve dierent uid problems. Usually these problems require the CFD results to be generated quickly as well as precisely.Â Am-ritkar et al. [10] pointed out that OpenMP can improve data locality on a shared memory platform compared to MPI in a uid-material application. However, Krpic et al. [11] showed that OpenMP performs worse when running large scale matrix multiplication even on shared-memory computer system when compared to MPI. KEYWORDS: Grid computing, mesh partitioning, Distributed computing, Load balancing, performance study. MOTS-CLÃ‰S : Maillages non-structurÃ©s, MÃ©thodes Ã©lÃ©ments finis, volumes finis, Partitionnement de maillages, Calcul sur Grille, Calcul distribuÃ©, Ã©quilibrage de charge.Â The aim of the project is to build a computational grid devoted to fluid dynamics applications, using a set of clusters interconnected by a wide area network. Mesh applications are a class of problems that requires such high-end computational power since performance must be scalable into hundreds of processors considering the current technology ([BAR 99]),([DJO 00]).Â On a computational grid, the machine architecture is heterogeneous both for network and processors.