Efficient Grid Resource Selection for a CEM Application

Eddy Caron, Cristian Klein, Christian Pérez

RenPar'19 / SympA'13 / CFSE'7
Toulouse, France, du 9 au 11 septembre 2009

Efficient Grid Resource Selection for a CEM Application*

Eddy CARON, Cristian KLEIN, Christian PÉREZ

LIP, École Normale Supérieur de Lyon,
46 Allée d'Italie
69364 Lyon, France
{Eddy.Caron,Cristian.Klein}@ens-lyon.fr, Christian.Perez@inria.fr

Résumé
Computational Electromagnetics (CEM) is a domain which provides numerical solutions to compute antenna performance, electromagnetic compatibility, radar cross section and electromagnetic wave propagation. The ever-increasing need for more precision and larger meshes raises the natural question whether it is worth porting CEM algorithms to computer grids. Due to the nature of the computations, CEM algorithms are not trivially parallelisable, as data dependency inside the mesh implies communication. The goal of this paper is to answer the question: given a set of resources, what is the subset of resources one should chose among, to minimise the time it takes to solve a CEM problem. After presenting a model of the application execution time, several algorithms for selecting resources are described. The limits of the theoretical model is then compared against experimental results, obtained from the Grid'5000 platform.

Mots-clés : CEM, Grid, Mapping, Grid'5000

1. Introduction

Partial Differential Equations (PDE) are involved in the solution to many problems, like car crash test simulation or dynamics fluid. However, solving the PDEs by variables elimination is difficult for all but the simple cases. For solving real-world problems, one may use the *Finite Element Method* (FEM), which consists in representing the space as a mesh. The resulting algorithms are non-trivially parallelisable, having irregular computations. As the size of the meshes increases, the question whether such applications could be speed up by deploying them in grid environments is naturally raised. The DiscoGrid project [1] aims to efficiently use hierarchical platforms, by proposing a message-passing API [3] with hierarchy-specific functions, which would allow an application to take better data-placement decisions. For example, the project focuses on Computational ElectroMagnetics (CEM) applications, which uses the FEM to solve electromagnetic problems.
The MAXDG1 software is used for the numerical resolution of the three-dimensional Maxwell equations in the time domain, for heterogeneous media and using tetrahedral unstructured meshes. Initially parallelized using MPI, it has already been ported to use the DiscoGrid API. Running the application consists in two steps. First, the mesh is being partitioned based on a resource description file, having the ability to assign bigger meshes to resources with more computation power. This is achieved using the *partIt* tool [4]. Second, the created submeshes are used to launch the application on the chosen resources.
Given a hierarchical topology, `partIt` is able to efficiently partition the mesh to minimise communication between resources, however, partitioning is always done for all resources, without taking into account that communication time might actually slow down the application. This article studies how to improve on this, and, given a set of resources and a mesh, how to choose the subset of resources which would minimise the execution time of the application.
The remaining of the paper is organised as follows: Section 2 introduces the application and resource model used to estimate the execution time; Section 3 presents and compares several algorithms to select

* This work was supported by the French National Agency for Research project DISC (ANR-05-CIGC-005).

resources; Section 4 displays experimental results and explains differences between the theoretical values and those obtained in practice; finally, Section 6 concludes the paper and opens up future perspectives.

2. Estimating Total Execution Time

Resource Modeling
In this study, resources consist in hosts which are grouped inside clusters. Let n_C be the number of clusters, each cluster i having $n_H^{(i)}$ hosts. Each host in cluster i has network access of bandwidth $B_H^{(i)}$, the intra-cluster latency being $l_{LAN}^{(i)}$. Hosts are homogeneous within the same cluster. Cluster i has an uplink bandwidth of $B_C^{(i)}$, while the *maximum* inter-cluster latency is l_{WAN}. Hosts are considered to be single-core, as the goal is to model grid-level behaviour.

Application Modeling
MAXDG1 is an iterative application, each iteration being composed of several stages of computation, separated by communication phases (UPDATE or ALLREDUCE). While ALLREDUCEs imply a blocking operation, where all nodes have to wait, the UPDATEs can be done either in either blocking or non-blocking mode. ALLREDUCEs operate on small amounts of data (maximum 7 double-precision values), while UPDATEs may operate on either small or large data sizes, depending on how the partitioning has been done. Each iterations has $n_U = 2$ UPDATEs of the same size and $n_{AR} = 4$ ALLREDUCEs.

2.1. Computation

When running the application sequentially on various hardware, we observed that some phases of the computation run faster on processors with more cache, while others favor CPU frequency. These computation phases are separated by communication, which force faster hosts to wait after the slower ones. The time spend by the whole grid in a computation phase is the maximum of the per-cluster computation times. Therefore, we give each phase a distinct speed coefficient $\alpha_j^{(i)}$, representing the number of seconds required to process one tetrahedra in computation phase j on a host of cluster i. Tests on Grid'5000 hardware show that the proportionality is respected within an error of 20%, because all objects on which iterations are done (vertices, faces, etc.) are proportional to the number of tetrahedra. The time spent in one computation phase becomes: $t_j^{(i)} = \alpha_j^{(i)} \cdot n_{tl}^{(i)}$, where $n_{tl}^{(i)}$ is the number of local tetrahedra assigned to a host on cluster i. Some of these computation phases can be overlapped with UPDATEs.

2.2. Communication

The basic formula used to estimate communication time is $t = l + s/B$, where t is the elapsed time for completing the communication, l is the latency of the network between the sender and the receiver, s is the size of the message and B is the available bandwidth.

ALLREDUCE In order to model the ALLREDUCE operation, we first have to choose an algorithm which implements it. We chose to reduce the number of WAN traversals, as this is by far the largest source of latency. Therefore, we used a similar approach to the one described in [6], which works by first doing a local reduce on a head node of each cluster, doing an all-to-all exchange between head nodes of the locally reduced values, then doing a cluster-wide broadcast. Given the fact that the size of the ALLREDUCE messages is small (less than 100 bytes) and that intra-cluster latency is small compared to inter-cluster latency[2] ALLREDUCE time becomes: $t_{AR}^{(i)} = l_{WAN}$.

UPDATE Unfortunately, the number of connections (artificial faces) between neighbor meshes is unpredictable and depends both on the mesh and how it has been partitioned. While the number of artificial faces is computed by the partitioning tool, this is not an option when choosing resources, as the operation may take a lot of time.

In order to find an upper bound for the time of UPDATE, we chose to start from the hypothesis that, since the number of tetrahedra is proportional to the volume of the mesh, while the number of artificial faces should be proportional to the surface of the mesh, the function should somehow represent the ratio

[2] For $n_H^{(i)} = 1024$ and $l_{LAN}^{(i)} = 44$ μs the intra-cluster latency is 0.88 ms, which is small compared to WAN latencies, which are of the order of tenth of ms. For clusters with a very large number of hosts, whether intra-cluster latency can really be ignored should be revisited, but is outside the scope of this paper.

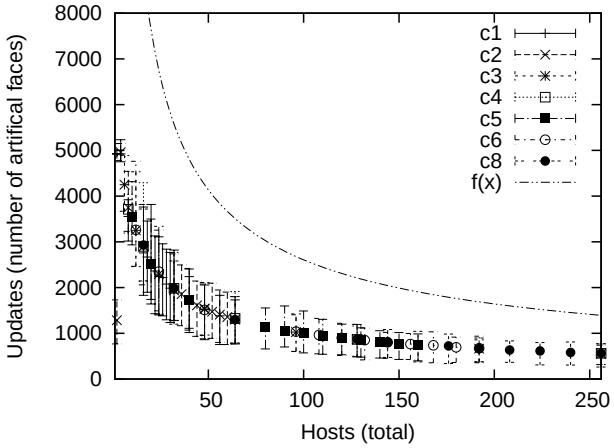


Figure 1: Number of inter-host artifical faces for a mesh with 588000 tetrahedra.

between surface and volume: $n_U = \beta \cdot n_{tl}^{2/3}$. Two separate values for β are taken, one for inter-host UPDATEs (β_H) and one for inter-cluster UPDATEs (β_C).
For the meshes we had at hand, $\beta_H = 5$ did a good job for inter-host UPDATEs, always being greater than the actual number of exchanges (Figure 1). The number of artificial faces between submeshes belonging to different clusters showed a similar trend, where in the above formula, we input the number of cluster-local tetrahedra and use $\beta_C = 1$.
As we cannot guarantee that this value will work well for all meshes and all partitions, we will use a self correcting approach. After choosing resources we will run the partitioning tool to see the actual number of updates. Should they affect our estimations significantly, a new value for β_H and β_C is computed and the process is restarted.
Assuming the application exchanges s_U bytes for each artificial face, the UPDATE time is:

$$t_U^{(i)} = \underbrace{l_{WAN} + \frac{s_U}{min\left(B_C^{(j)}, n_H^{(j)} \cdot B_H^{(j)}\right)} \cdot \beta_C \cdot \left(\frac{n_t}{n_C}\right)^{2/3}}_{\text{Inter-cluster exchanges}} + \underbrace{\frac{s_U}{B_H^{(i)}} \cdot \beta_H \cdot \left(\frac{n_t}{n_{tH}}\right)^{2/3}}_{\text{Intra-cluster exchanges}} \quad (1)$$

where n_{tH} is the total number of hosts.
In practice, we observed that certain phenomena hinder using the whole host bandwidth for UPDATE operations. First, the default TCP behaviour on many operating systems (as per RFC 2861) consists in doing a slow start after a certain period of communication inactivity. Due to the fact that computations between the UPDATEs might take some time, this means that TCP will never reach its optimal window, for which the bandwidth is maximal. On Linux, this issue can be solved by setting `net.ipv4.tcp_slow_start_after_idle` to `0`, which will force the TCP protocol to use the last window, even after a long period of inactivity.
Second, bidirectional TCP transfers cannot reach the maximum full-duplex network capacity, not even on the LAN. For example, running an `iperf` dual test returns less than 1Gbps for both directions when using Gigabit Ethernet cards. This is caused by the two transfers congesting each others ACK path. During our estimation we will assume that the effective available bandwidth is only 50% of the one inputted by the user. A more elaborate study regarding *transfer completion time* in high-speed networks is presented in [5].

2.3. Totals

We compute the total iteration time as the sum of the maximum per-cluster time of each computation or communication phase. For synchronous communication the UPDATE time has to be added to the duration of an iteration, while for asynchronous communication it has to be maxed with the computation which can be overlapped.

2.3.1. Testing

Our application exchanges 6 double-precision floating-point values for each artificial face ($s_U = 48$bytes). We did a limited number of experiments on the Grid'5000 platform, in order to validate our model. We ran our experiments on the Toulouse `pastel` and Sophia `sol` cluster, both having Dual-Core AMD Opteron(tm) Processor 2218, running at 2.6GHz with 1MB of RAM. For this processor we measured $\alpha \approx 1.033 \cdot 10^{-5}$ (seconds / tetrahedron) and $\alpha_U \approx 2.198 \cdot 10^{-6}$ (seconds / tetrahedron). The inter-cluster latency was $l_{WAN} = 8.1$ ms, while the bandwidth of each host was $B_H = 1$ Gbps. The results of these tests showed a good predictability, the error being less then 5%.

3. Selecting Resources

This section deals with the selection of grid resources for the MAXDG1 application. First, the factors that have to be taken into account are enumerated, next several algorithms are described. Finally, the algorithms are compared, from a performance and precision point of view, both in the average and worst cases.

This paper assumes that the intra-cluster latencies are negligible compared to inter-cluster latencies. Since the number of artificial faces does not change significantly when partitioning the mesh with different weights, as would be required when adding hosts to a cluster, when choosing resources, either all hosts from a cluster belong to our chosen resource set, or none of them.

MAXDG1 uses about 600 bytes for each tetrahedron. This means that a system with 1GB of RAM available for data will be able to store submeshes with at least $1,750,000$ local tetrahedra. Since this is quite a lot, compared to the computation time required to solve such a problem (we estimate at least 11s per iteration), this study considers that memory is not an impediment when choosing resources, however it should not be too difficult to adapt the algorithms to take this into account.

3.1. Algorithms

Due to space constraints we will only list here the driving principles of the algorithms, without detailing how they work nor presenting their pseudo-code.

3.1.1. Exhaustive

The first algorithm we devised is an exhaustive search inside the solution space, allowing us to find the optimal resource subset, limited only by the quality of our estimations. As expected, its time complexity is exponential in the number of clusters (n), being $O(2^n)$. It can be used for small cases, to compare the precision of the other algorithms. On a Intel Core 2 Duo @ 2GHz, exhaustive search takes about 78 seconds for 18 clusters, to compare to the 25 clusters of Grid'5000.

3.1.2. Random

We have also considered a totally random algorithm, which blindly choses clusters. Because generating a random resource set is a very cheap operation, the random algorithm takes virtually no time compared to other algorithm. Therefore, in order to more sensibly compare algorithms as a product between precision and performance, we will generate a thousand random solutions and choose the best out of it.

3.1.3. Greedy

The naive greedy approach consists in starting with an empty resource set (whose total iteration time is infinity) and in each step adding a cluster which would minimise the total iteration time. The algorithm stops when no matter what cluster we add, the total iteration time is increasing.

It is very easy to fool the naive approach. The first chosen cluster will always be the one with the largest computation power. If that cluster is connected to a high-latency network, chances are that it does not belong to the best resource set. Therefore, we chose to improve on the naive greedy approach, by starting from a resource set initially filled with each cluster in turn. The complexity becomes $O(n^3)$.

Although results are greatly improved, the greedy approach is still far from returning the optimal solution. Often we have clusters grouped together in "low-latency islands". For example, we might have two clusters in Hawaii and two in Australia. Suppose we have the two Australian clusters in our current resource set. Depending on the size of the mesh, computation power and network latency, it might happen that adding only one Hawaiian cluster does not improve performance, however adding both does. In such a

Algorithm	Fail (‰)	Error (%)			Time (ms)		
		min	avg	max	min	avg	max
exhaustive	0	0	0	0	689	706	1012
rnd	1000	24	601	2410	0	0	0
rnd1k	782	0	22	363	176	185	211
greedy	2	0	ϵ	4	12	37	92
grouping	0	0	0	0	28	59	132
sa	630	0	19	209	25	57	100
sa.greedy	1	0	ϵ	4	42	96	211
sa.grouping	0	0	0	0	59	118	219

Table 1: Comparison of Algorithms in heterogeneous cases. ϵ means the error is less than 1%.

case, the greedy approach would not return an optimal solution.

3.1.4. Greedy with Cluster Grouping

In order to improve on the greedy algorithm, we wanted to somehow enable the greedy algorithm to intelligently add multiple clusters at a time. Since clusters that are in the same city or same country are more likely to have smaller latency between them, we chose to group them and allow the algorithm to add a whole group at a time. While it is somewhat challenging, one can still find corner cases in which the algorithm does not return the optimal solution.

3.1.5. Simulated Annealing

Since in essence we have to find the global minimum of a certain function, which also has local minima, we decided to experiment with simulated annealing [2], due to its popularity and ease of implementation. We used this algorithm to build a solution either from an empty resource set, or improve the solution found by the two greedy approached described above.

3.2. Testing

We tested our algorithms off-line, using a test case generator. The network consists in multiple levels: world, country, city and cluster, organised in a tree-like structure, each branch has a randomly chosen latency. We devised several test-cases and presented their results in separate tables. For each algorithms, we show the number of times it failed to find the best solution, i.e. another algorithm outperformed it (*Fail*), the deviation from the best solution (Error = $(t_{\text{found}} - t_{\text{best}})/t_{\text{best}}$); ϵ meaning the error is less than 1%) and the time it took to find the solution (*Time*).

3.2.1. Heterogeneous Test Case

The goal of this first test is to compare the results of the algorithms against the known optimal value given by the exhaustive algorithm. The total number of clusters is fixed and relatively small, to make exhaustive testing feasible and to have an idea of how much time each algorithms requires.

Network heterogeneity is obtained using the following configuration: there are two countries with world latencies in $[50, 100]$ ms and three cities with country latencies in $[10, 50]$ ms. Each city had two clusters with latencies in $[1, 5]$ ms. This guarantees that the latencies in the network will respect the triangle rule.

To simulate computation power heterogeneity, we took the performance values of a real clusters and multiplied them with a uniform random value in $[0.8, 1.2]$ ms. Each cluster was given between 16 and 64 hosts. 1000 distinct test cases have been generated. The results are presented in Table 1 and show that intelligently (even if not optimally) choosing resources pays off. Greedy and grouping perform well in such a heterogeneous case, having small errors. Simulated annealing by itself is not a very good solution. When using it over greedy, it does not offer improvements which would justify the extra time.

3.2.2. Very Heterogeneous Case

In this test run, we wanted to cover an as diverse as possible problem space to discover possible corner cases in the proposed algorithms. We kept the same network latency structure, but each test case had between 1 and 4 countries, 1 and 10 cities per country, 1 and 3 clusters per city and between 8 and

Algorithm	Fail (‰)	Error (%)			Time (ms)	
		min	avg	max	min	max
rnd	926.6	0	278	1107	0	13
rnd1k	828.1	0	184	973	81	3247
greedy	4.9	0	ϵ	14	0	13660
grouping	1.6	0	ϵ	4	1	16259
sa	776.8	0	50	874	32	499
sa.greedy	3.8	0	ϵ	14	33	13927
sa.grouping	0.5	0	ϵ	1	34	16782

Table 2: Comparison of Algorithms in very heterogeneous cases. ϵ means the error is less than 1%

Algorithm	Fail (‰)	Error (%)			Time (ms)		
		min	avg	max	min	avg	max
exhaustive	0	0	0	0	39	43	79
rnd	688	0	96	491	0	0	2
rnd1k	15	0	0	17	161	171	666
greedy	127	0	2	36	19	27	101
grouping	2	0	ϵ	ϵ	29	40	171
sa	335	0	5	84	39	61	183
sa.greedy	0	0	0	0	64	90	181
sa.grouping	0	0	0	0	72	103	181

Table 3: Comparison of Algorithms in a Close to Homogeneous Case. ϵ means the error is less than 1%

128 hosts per cluster. Due to the size of these tests it was too lengthy to run the exhaustive algorithm and considered that an algorithm failed to find the best solution, if another one outperformed it. 10000 distinct test cases have been generated. The results are presented in Table 2.
We observe that the greedy algorithm performs quite well, but grouping improves results. When analysing the test for which greedy failed, as previously, we observed that by taking one cluster at a time, it may hang in local-minima. The grouping algorithm jumps over such local-minima, but may take too many clusters at a time and thus miss the global-minimum. Adding simulated annealing helps both algorithms. Interestingly, the simulated annealing phase after the greedy was sometimes "luckier" than one after grouping, which is why the simulated annealing with grouping also has failures.

3.2.3. Close to Homogeneous Case

From the above test, it would seems that the greedy algorithm is quite a good one, however it does not behave well when resources are in a certain configuration, as described in Section 3.1.3. In order to explore how easy it is to find such a configuration, we devised a test in which resources are closer to homogeneous. We kept the same network latency configuration, with only one country and two cities, each city having four clusters. Each cluster has the same number of hosts which can be 64 or 128 and all hosts have the same processing power.
1000 distinct test cases have been generated. The results are presented in Table 3. We observe that the greedy approach behaves poorly, being surpassed precision-wise by the random approach. Hence, a good algorithm is grouping which is further improved with an extra simulated annealing phase.

4. Experimental Results

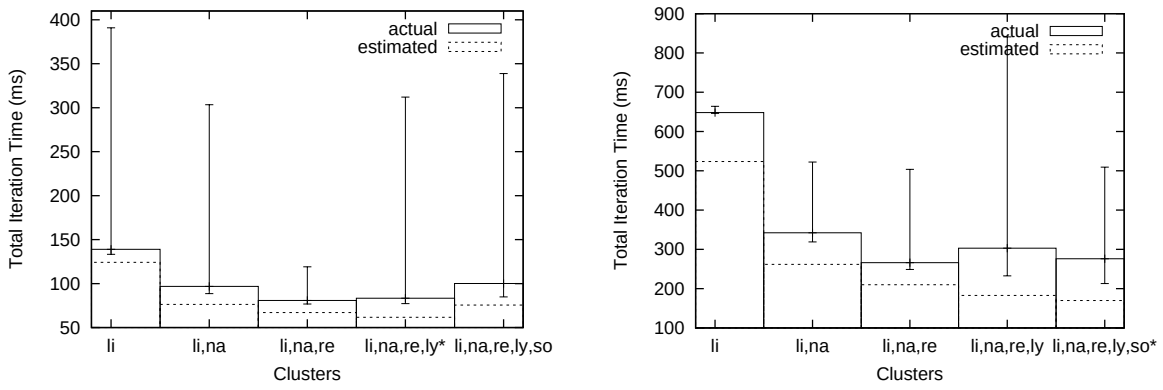
4.1. Objectives

We devised an experiment having two goals. First, we wanted to check whether our theoretical model holds in the real world, with what error and what are the causes of these errors. Second, we wanted to know whether the expected optimal solution, found by our algorithms, conforms to reality.

Site	Cluster	nH	α [1]
Lille	chinqchint	32	6.75e-06
Lyon	capricorne	48	1.46e-05
Nancy	griffon	48	8.20e-06
Rennes	paraquad	32	8.94e-06
Sophia	azur	48	1.47e-05

[2] This is the microbenchmark value measured in total seconds per tetrahedron for a mesh of 19800 tetrahedra.

Table 4: Grid'5000 configuration used for our experiments.



(a) Mesh with 588000 tetrahedra

(b) Mesh with 2480674 tetrahedra

* Optimal solution as found by our resource selection utility.

Figure 2: Comparison between estimated and measure time for different Grid'5000 configurations.

4.2. Description

The experiments were done on the Grid'5000 platform, using the resource presented in Table 4. In order for the latencies to be of the order of those found in grid environments, we avoided using two clusters from the same site. We used two meshes, one having 588000 tetrahedra and the other having 2480674 tetrahedra. Our CEM resource selection utility was used to choose the expected optimal configuration for each of the two meshes. All deterministic algorithms (i.e exhaustive, greedy and grouping) found the same solutions.

Test runs have been done for the expected optimal configuration, and for other configurations obtained by adding unselected clusters (fastest first) or removing clusters (slowest first) from the expected optimal configuration. This should generate enough test cases both to validate the theoretical model and to see real-world behaviour around the expected optimal configuration.

The results for both meshes are presented in Figure 2. The X-axes describes the different configurations which were used in our experiments, the configuration marked with a star being the one found by our resource selection utility. For each configuration we plotted the expected per-iteration time together with the measured per-iteration times. Since we observed large variations from one iteration to another, we also plotted the minimum and the maximum per-iteration time.

4.3. Analysis

We observe that the value of the fastest iteration closely follows our estimated values. The biggest error comes from estimating the computation time, which, due to the way the mesh is split and due to different processor cache sizes, does not behave perfectly linearly, as in our model. This also causes unoptimal mesh splitting, therefore adding a new cluster might not induce the expected speedup and, by increasing waiting time in the other clusters, may even slow down the application.

We also observed that the average per-iteration time deviates quite a bit from the minimum, mostly due

	On Orsay	On Grid			
n_T	Iter (ms)	Iter (ms)	Comm (ms)	n_C	Selected Clusters
588,245	17	17	0	3	Or (3)
2,480,674	71	53	27	13	Li (4), Na (2), Or (3), Re (4)
5,135,732	148	78	29	15	Li (4), Ly (2), Na (2), Or (3), Re (4)
100,000,000	2879	765	114	25	all
1,000,000,000	28796	7256	751	25	all

Table 5: Expected optimal configuration for Grid'5000 for different mesh sizes.

to a few iterations are up to four time slower than the minimum. Whether these deviations are normal or were caused by specific experimental conditions, and whether they can be estimated or bounded in any way, needs to be further investigated.

For both meshes, we observe that our application missed the optimal solution, but the extra time is less than 4%. For the bigger mesh, disturbances caused a slowdown of the whole application when adding the Lyon cluster. However, per-iteration time decreased when adding the Sophia cluster.

5. Numerical Application

We gave the whole Grid'5000 infrastructure as input to our resource selection utility having a total of 25 clusters. The results are presented in Table 5, which shows the estimated per-iteration time (Iter) and communication time (Comm) when running the application on the fastest site (Orsay) and on the grid. The bigger the size of the mesh, the bigger the benefit of using a grid. If the mesh is medium sized only a portion of Grid'5000 is used.

6. Conclusion

The need for solving the CEM problem for ever increasing meshes leads to the idea of running these applications in grid environments. This article has presented a model of how CEM applications and MAXDG1 in particular behave in grid environments and presented several algorithms for choosing grid resource so that execution time is minimised. Experimental results look promising and have validated our theoretical assumptions.

Future perspectives include extending our study to multi-core machines. This would require extending the `partIt` tool to support three hierarchical levels and added a module to DHICO (the DiscoGrid API implementation we used) to handle cores. Studying other CEM- or FEM-like application should be the next step, with the goal of determining whether application modeling can be automated.

Bibliographie

1. DiscoGrid project. Technical report, INRIA, 2007. Available online: http://www-sop.inria.fr/nachos/team_members/Stephane.Lanteri/DiscoGrid/.
2. P. Brucker. *Scheduling Algorithms*. Springer, 5th edition, March 2007.
3. DiscoGrid Project. Work-package WP1: Specification of an API for hierarchical communications. Technical report, INRIA, 2007. Available online: http://www-sop.inria.fr/nachos/team_members/Stephane.Lanteri/DiscoGrid/docs/SR-1.1.pdf.
4. DiscoGrid Project. Work-package WP3: Multi-level partitioning tool. Technical report, INRIA, 2008. Available online: http://www-sop.inria.fr/nachos/team_members/Stephane.Lanteri/DiscoGrid/docs/TR-3.1.pdf.
5. R. Guillier, S. Soudan, and P. Primet. TCP variants and transfer time predictability in very high speed networks. In *Infocom 2007 High Speed Networks Workshop*, May 2007.
6. T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang. MAGPIE: MPI's collective communication operations for clustered wide area systems. In *ACM SIGPLAN Notices*, pages 131–140, 1999.