

Balancing the Computational Effort over the Subdomains for a Parallel Multifrontal Solver

Christophe Denis and Jean-Paul Boufflet

Department of Computing Engineering
Compiègne University of Technology, Compiègne, France
e-mail: {Christophe.Denis, Jean-Paul.Boufflet}@utc.fr

Piotr Breitkopf*, Alain Rassinoux and Michel Vayssade

Department of Mechanical Engineering
Compiègne University of Technology, Compiègne, France
e-mail: {Piotr.Breitkopf, Alain.Rassinoux, Michel.Vayssade}@utc.fr

Key words: finite element method, multifrontal solver, load balancing

Abstract

We work on direct methods to solve large sparse linear systems issued from finite element modeling in mechanical engineering. Our pragmatic approach aims to improve the performance of a parallel frontal solver code which uses a finite element by finite element assembly strategy over each subdomain. We first use standard partitioning methods to split problems into subdomains having a balanced amount of data. Then, by applying our heuristics, the partition is modified in order to have a balanced number of operations over the subdomains. This heuristics uses an estimator of the number of operations required to eliminate internal equations of each subdomain.

The number of operations required to solve in parallel a sparse linear system depends on the mesh partitioning (i.e. the domain is split into subdomains), the domain reordering, the subdomains reorderings and the type of solver used. To obtain the partition we can use for instance the softwares *METIS* [11], *SCOTCH* [13] and *CHACO* [10]. These methods are based on graph partitioning techniques. Other methods are proposed by the community of computational mechanics [12] [9]. These methods aim to reduce the communications between processors and to balance the subdomain amount of data.

We present in this paper a heuristics that modifies an initial partition in order to balance the computational effort over the subdomains. We first use an estimator that evaluates the number of operations required to treat each subdomain. We present the estimator that has been modeled and tested in previous works [4] [7] [14]. Then, we transfer finite elements between subdomains to correct the partition. We use *METIS* and our implementation of [12] to build the initial partitions. We use a set of meshes issued from finite elements models as our benchmark. We measure the computing time on each subdomain, the resolution time of the interface problem and the global resolution time before and after the use of our heuristics. According to these preliminary results obtained with our parallel multifrontal solver, it seems to be interesting to balance a partition by taking into account the number of operations over the subdomains rather than the amount of data.

In section 2, we present the problem formulation and our experimental protocol. In section 3, we define the algorithm of the heuristics. Experimental results are shown in section 3. Finally, we conclude and give some future directions.

1 Problem Formulation and Experimental Protocol

The academic software *SIC* (Système Interactif de Conception) has been developed by the community of computational mechanics at UTC since 1988 [15]. This framework integrates various finite element models. The resolution methods involve:

- sequential, parallel, sparse, direct and iterative solvers;
- subdomain decomposition and reordering;
- sparse matrices reordering.

The multifrontal solver of *SIC* uses BLAS level 1 and PVM, MPI or SHMEM communication toolboxes. Let \mathcal{P} be a partition of s subdomains denoted SD_j ($1 \leq j \leq s$). The s subdomains SD_j distributed among s processors and form the input data for the solver. Each subdomain SD_j contains a list of finite elements and a list of interface nodes. The resolution process is divided into two main steps:

1. condensing each subdomain in parallel;
2. solving the interface problem.

During these two steps we use a straightforward implementation of the frontal solver [8]. Each processor j applies a frontal solver that interleaves the assembly of the finite elements of the subdomain SD_j and the elimination of the internal unknowns. This first step, called the subdomain condensations, gives only residual matrices containing coefficients of the interface unknowns only. These interface unknowns can not be eliminated at this stage because the corresponding equations involve the data of another

subdomain. This residual matrix can be considered as a finite superelement. In step 2 we apply the frontal solver to the finite superelements corresponding to subdomains. This strategy is well suited to the finite element interpretation of the frontal solver working on an element by element basis [5] [6]. This approach involves simultaneous matrix assembly and elimination and should not be confused with the interpretation of a frontal solver [1] [3] working on the global system $A \cdot x = b$.

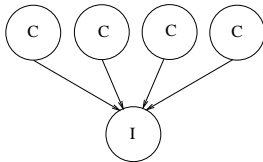


Figure 1: out-tree of tasks for a partition \mathcal{P} of 4 subdomains

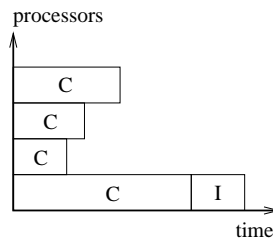


Figure 2: scheduling on 4 processors of the original partition

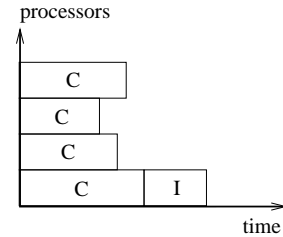


Figure 3: scheduling of the balanced partition: maximum condensation time decreases, interface augments

The overall process is represented in figure 1 where C is a condensation task and I the solving of the interface problem. Our [14] estimator of the number of operations required to condense a subdomain SD_j is:

$$Q(SD_j, V_{num}^j) = \alpha_1 \cdot \sum_{k \in incorporation} f_k^{(j)} + \alpha_2 \cdot \sum_{k \in elimination} (f_k^{(j)})^2 \quad (1)$$

where V_{num}^j is the local reordering vector representing the assembly order of finite elements during the condensation of SD_j . The index k indicates the incorporation or the elimination of an unknown in the frontal matrix $\mathcal{F}^{(j)}$ of size $f_k^{(j)}$. The number of operations required to incorporate a new unknown in the frontal matrix is $O(f_k^{(j)})$ whereas the number of operations required to eliminate a unknown is $O((f_k^{(j)})^2)$ [14].

The purpose of the work presented in the present paper is to build partitions having a balanced number of operations rather than a balanced amount of data. We use a two steps approach. The partitions (Ma) are initialized with an algorithm inspired from the Malone work [12]. A global reordering vector is first computed on the finite element graph using [2]. Then the partition is built by splitting the vector into s equal parts. An alternative initialization for the partitions (Me) is *METIS* 2.0, whos routines can be directly called in the software *SIC*. In the second step, we use our heuristics to improve the partitions in term of equal estimation of the number of operations.

The experimental protocol is as follows:

- the initial partitions of the meshes are obtained with Me and Ma ;
- we apply our heuristics based on the estimator $Q(SD_j, V_{num}^j)$ to obtain the partitions MeC and MaC (C for corrected);
- we compare the execution times for Me , Ma , MeC and MaC .

2 Algorithm to Balance the Computational Load

Table 1: Notations used in the algorithm

s :	number of subdomains.
V_{num}^j :	local reordering vector of finite elements for the subdomain SD_j with $j \in \{1, \dots, s\}$.
V_{dec} :	partition vector of size m (the domain has m finite elements). $V_{dec}[e] = j$ means that finite element $e \in SD_j$.
$V_{num}^j best$:	local reordering vector for each SD_j for the best solution encountered.
$V_{dec} best$:	partition vector for the best solution encountered.
$Q(j, V_{num}^j)$:	estimation of the number of operations required to condense SD_j with $j \in \{1, \dots, s\}$.
m_j :	number of finite elements $\in SD_j$.
m_t :	number of finite elements to transfer from one subdomain to another one.
Nb_iter_glob :	number of iterations done in the global balancing loop .
$Limite_glob$:	maximum number of iterations of the global balancing loop.
Nb_iter_loc :	number of iterations done in the local balancing loop.
$Limite_loc$:	maximum number of iterations of the local balancing loop.
$Estim_glob$:	estimation of the global balancing quality.
$Estim_glob_best$:	estimation of the global balancing quality for the best solution encountered.
$Estim_loc$:	estimation of the local balancing quality.
$Epsilon_glob$:	upper bound for the global balancing.
$Epsilon_loc$:	upper bound for the local balancing.

We introduce in table 1 some notations. The algorithm (cf algo. 1) is divided into two main steps: the initialization step and the balancing step. The initialization step (lines 1 to 5) loads the initial partition vector V_{dec} and the local reordering vectors V_{num}^j . The balancing step (lines 6 to 28) balances the computational load over two subdomains. If we try to balance two subdomains without a common boundary we increase the number of interface nodes, so we only consider two subdomains with a common boundary. The global balancing loop stops either when a number of iterations ($Nb_iter_glob = Limite_glob$) or when the global quality criterion ($Estim_glob \leq Epsilon_glob$) is reached. The best global solution minimizes $max(Q(SD_i, V_{num}^i))$ for $i \in \{1, \dots, s\}$. The local balancing (lines 9 to 22) is the main function of this algorithm. Let SD_i and SD_j be two subdomains with $Q(SD_j, V_{num}^j) > Q(SD_i, V_{num}^i)$. The number of operations required to condense SD_i is lower than the number of operations required to condense SD_j . In order to balance the computational effort over these two subdomains, the algorithm transfers m_t finite elements from SD_j to SD_i . The transfer of $\frac{Q(SD_j, V_{num}^j) - Q(SD_i, V_{num}^i)}{2}$ operations from SD_j to SD_i would balance these two subdomains. Known the mean number of operations by finite element ($\frac{Q(SD_j, V_{num}^j)}{m_j}$), the number m_t of finite elements to transfer is estimated by :

$$m_t \leftarrow \frac{Q(SD_j, V_{num}^j) - Q(SD_i, V_{num}^i)}{2} \cdot \frac{m_j}{Q(SD_j, V_{num}^j)} \quad (2)$$

Algorithm 1: Algorithm to Correct the Computational Load

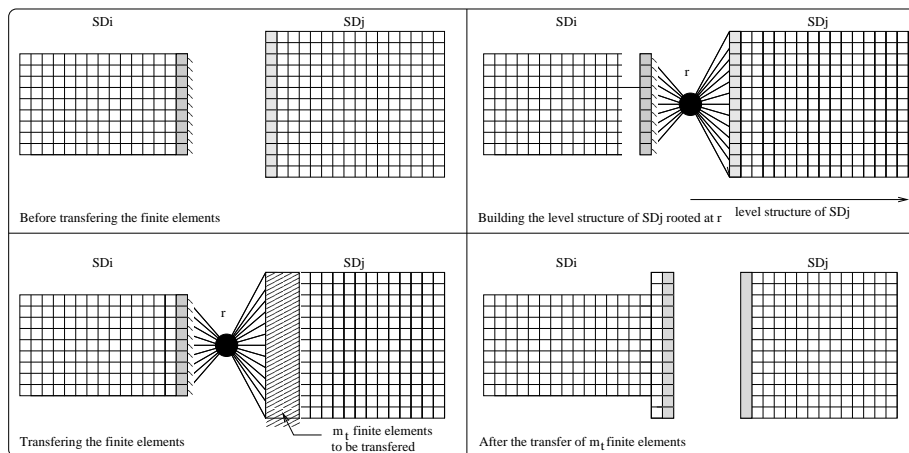
```

1: Load the partition vector  $V_{dec}$  {Initialization}
2: Load the local reordering vectors  $V_{num}^i$  pour  $i \in \{1, \dots, s\}$ 
3:  $Nb\_iter\_glob \leftarrow 0$ 
4:  $Estim\_glob\_best \leftarrow \infty$ 
5:  $j \leftarrow 0$ 
6: repeat
7:    $Nb\_iter\_glob \leftarrow Nb\_iter\_glob + 1$ 
8:    $j \leftarrow modulo(j + 1, s - 1) + 1$  {Begin of the global balancing loop}
9:   {Local balancing between the subdomains  $SD_i$  (neighbour of the subdomain  $SD_j$ ) and the sub-
      domain  $SD_j$ }
10:  for all subdomains  $SD_i$  sharing a boundary with  $SD_j$  do
11:    {Begin of the local balancing between the subdomain  $SD_j$  and the subdomain  $SD_i$ }
12:     $Nb\_iter\_loc \leftarrow 0$ 
13:    repeat
14:       $Nb\_iter\_loc \leftarrow Nb\_iter\_loc + 1$ 
15:      if  $Q(SD_j, V_{num}^j) > Q(SD_i, V_{num}^i)$  then
16:         $m_t \leftarrow \frac{Q(SD_j, V_{num}^j) - Q(SD_i, V_{num}^i)}{2} \cdot \frac{m_j}{Q(SD_j, V_{num}^j)}$  {Transfer of  $m_t$  finite elements
          from  $SD_j$  to  $SD_i$ }
17:         $V_{dec} \leftarrow \mathbf{Transfer}(SD_j, SD_i, m_t)$ 
18:      else
19:         $m_t \leftarrow \frac{Q(SD_i, V_{num}^i) - Q(SD_j, V_{num}^j)}{2} \cdot \frac{m_i}{Q(SD_i, V_{num}^i)}$  {Transfer of  $m_t$  finite elements
          from  $SD_i$  to  $SD_j$ }
20:         $V_{dec} \leftarrow \mathbf{Transfer}(SD_i, SD_j, m_t)$ 
21:      end if
22:       $V_{num} \leftarrow \mathbf{Update\_V\_num}(V_{num}, V_{dec})$ 
23:       $Estim\_loc \leftarrow \mathbf{Compute\_Estim\_loc}(Q(SD_i, V_{num}^i), Q(SD_j, V_{num}^j))$ 
24:      until ( $Nb\_iter\_loc \geq Limite\_loc$ ) or ( $Estim\_loc < Epsilon\_loc$ )
25:    end for
26:     $Estim\_glob \leftarrow \mathbf{Compute\_Estim\_glob}$ 
27:    if  $Estim\_glob < Estim\_glob\_best$  then
28:       $(V_{dec}^{best}, V_{num}^{best}) \leftarrow \mathbf{Save}(V_{dec}, V_{num})$  {Save the best solution}
29:       $Estim\_glob\_best \leftarrow Estim\_glob$ 
30:    end if
31:  until ( $Nb\_iter\_glob \geq Limite\_glob$ ) or ( $Estim\_glob < Epsilon\_glob$ )
32:  return  $V_{num}^{best}$  and  $V_{dec}^{best}$ 

```

The local balancing loop stops either when the number of iterations ($Nb_iter_loc = Limite_loc$) or when the local quality criterion ($Estim_loc \leq Epsilon_loc$) is reached.

The primitive **Transfer** chooses finite elements near the common boundary in order to limit the growth of the interface. Let us consider the two subdomains SD_i and SD_j of the figure 4. On the top left corner there are the two subdomains before the transfer. The grey parts of these two subdomains represent their boundary finite elements. Let us suppose that m_t finite elements have to be transferred from SD_j to


 Figure 4: Example of transfer of m_t finite elements

SD_i . A level structure is computed on the subdomain SD_j in order to determine the finite elements to be transferred. We create a virtual root r connected to the boundary finite elements of each subdomain (cf. top right corner of figure 4). We compute the level structure from r to the finite elements of SD_j . Finite elements are then transferred level by level (starting with the first level) from the subdomain SD_j to the subdomain SD_i until m_t have been transferred. The two subdomains after the transfer are shown on the bottom right corner of the figure 4. That permits us to minimize the growth of the boundary. This algorithm works on a *SOLARIS* workstation during some seconds for the small meshes and during about ten seconds for the mesh *MISSILE4* having 332 988 unknowns. In the examples of section 4 *Limite_loc* and *Limite_glob* are set to 10. We arbitrarily set $\alpha_1 = \alpha_2 = 1$.

3 Results

The parallel computing has been done on a PC-cluster composed of 10 processors PENTIUM III 666 Mhz (384 Mo) under LINUX Red Hat 7.1 with a switch Cisco Catalyst 3524XL and a Ethernet network 100 Mhz. The solver we used is the multifrontal solver of the software *SIC*.

Table 2: Description of the meshes used

Mesh name	Number of finite elements	Number of nodes	Number of ddl
FUSEEN	36 570	8 374	3
MISSILE2	6 802	3 421	6
MISSILE3	13 615	27 152	6
MISSILE4	27 804	55 498	6
SUSPEN.D1	18 171	4 839	3

The table 2 gives the features of the meshes used in our tests. The number of unknowns varies from 14 517 to 332 988. In all tables, the indication *OOC* (out-of-core) means that the memory is insufficient to treat a subdomain in core. All times are indicated in seconds. The first columns of tables 3 to 5 indicate the mesh names. The columns called “Meth.” of these tables indicate the type of partition (*Me*, *MeC*,

Ma or MaC). The columns called “ T_I ” (resp. “ T_G ”) give the resolution time of the interface problem (resp. the global resolution time). The columns “ T_{SD_i} ” of these tables gives the condensation times of each subdomain SD_i for mesh partitions with $s = 2, 4$ and 8 subdomains. For example, at line 4 and column 5 of table 3, the condensation time of the subdomain SD_3 for the mesh $MISSILE2$ partitioned into 4 subdomains using $METIS$ is 38.5 s.

Table 3: CPU times for mesh partitions (Me, MeC, Ma and MaC) into 2 subdomains

Mesh name	Meth.	T_{SD_1}	T_{SD_2}	T_I	T_G	Meth.	T_{SD_1}	T_{SD_2}	T_I	T_G
FUSEEN	Me	241.9	265.1	0.6	268.9	Ma	144.6	869.6	6.3	883.7
FUSEEN	MeC	242	257.1	0.5	263.1	MaC	258.9	238.2	5.5	273.4
MISSILE2	Me	41.1	101.7	0.1	103.3	Ma	10.9	41.4	0.1	42.5
MISSILE2	MeC	58.9	50.1	0.1	61.4	MaC	13.4	20.2	0.1	21.2
MISSILE3	Me	797	OOO	OOO	OOO	Ma	239.7	619.2	0.4	625.7
MISSILE3	MeC	872.5	1302.5	1.1	1521.8	MaC	327	321.7	0.6	346.1
MISSILE4	Me	OOO	OOO	OOO	OOO	Ma	OOO	OOO	OOO	OOO
MISSILE4	MeC	OOO	OOO	OOO	OOO	MaC	OOO	OOO	OOO	OOO
SUSPEN_D1	Me	OOO	OOO	OOO	OOO	Ma	127.1	264.2	17.4	285.5
SUSPEN_D1	MeC	OOO	OOO	OOO	OOO	MaC	169.2	65.2	0.8	172.3

Table 4: CPU times for mesh partitions (Me, MeC, Ma and MaC) into 4 subdomains

Mesh name	Meth.	T_{SD_1}	T_{SD_2}	T_{SD_3}	T_{SD_4}	T_I	T_G
FUSEEN	Me	74.6	84.2	88.9	101.1	10.5	114.3
FUSEEN	MeC	74.6	84.2	88.9	101.1	10.5	114.3
MISSILE2	Me	14.3	35.2	38.5	23.1	1.6	41.1
MISSILE2	MeC	24.4	24.4	20.7	19.9	3.5	29.3
MISSILE3	Me	622.6	426.8	732.7	288.8	14.4	761.5
MISSILE3	MeC	295.6	405.5	403.8	344.3	20.3	437
MISSILE4	Me	OOO	OOO	OOO	OOO	OOO	OOO
MISSILE4	MeC	OOO	OOO	OOO	OOO	OOO	OOO
SUSPEN_D1	Me	58.9	72.5	72.7	65.1	1.2	77
SUSPEN_D1	MeC	59.3	66.2	65.7	61.7	5.4	75.4
FUSEEN	Ma	45.4	264.4	334.6	251.6	96	444.3
FUSEEN	MaC	103.1	82.6	95.2	115.9	103.5	242.2
MISSILE2	Ma	9.4	41.3	27.9	24.7	1.3	44.4
MISSILE2	MaC	12.1	28.4	27.9	24.7	1.4	32.2
MISSILE3	Ma	121.4	450.2	332.4	256.3	10.6	469.2
MISSILE3	MaC	187.6	244.8	296.4	254.8	17.3	318.9
MISSILE4	Ma	613.1	OOO	OOO	1389.9	OOO	OOO
MISSILE4	MaC	1 200.6	1 038.3	1 444.2	1 389.9	31.8	1 603.7
SUSPEN_D1	Ma	21.1	31.2	88.5	22.7	68.9	161.6
SUSPEN_D1	MaC	46.2	18.7	30.5	22.7	68.4	119.1

The figure 5 (resp. 6) shows the condensation times for each subdomain for the mesh $MISSILE2$ (resp. $SUSPEN_D1$) partitioned into 4 subdomains with 4 partitions (Me , MeC , Ma et MaC). We can remark that the maximum condensation times are achieved with Me or Ma . The partitions MeC (resp. MaC) obtained by correcting the initial partition Me (resp. Ma) decrease the maximum condensation

Table 5: CPU times for mesh partitions (Me, MeC, Ma and MaC) into 8 subdomains

Mesh name	Meth.	T_{SD_1}	T_{SD_2}	T_{SD_3}	T_{SD_4}	T_{SD_5}	T_{SD_6}	T_{SD_7}	T_{SD_8}	T_I	T_G
FUSEEN	Me	38	37.2	25.5	38.2	21	44.3	43.5	57.2	65.4	140
FUSEEN	MeC	38	37.2	25.5	38.2	21	44.3	43.5	57.2	65.4	140
MISSILE2	Me	11	11.2	7.5	2.6	11.8	11.8	8.6	8	17.8	32
MISSILE2	MeC	11.4	9.8	7.4	2.6	11.7	11.4	8.6	8	19.4	34.6
MISSILE3	Me	239.5	169.1	122.1	213	173.8	209.3	120.8	67.3	182.3	444.3
MISSILE3	MeC	167	171.7	159.6	191.8	185.4	169	126.3	115.6	241.3	448.4
MISSILE4	Me	1 114.8	721.3	273.2	513.9	OOO	669.4	580.3	641.3	OOO	OOO
MISSILE4	MeC	684.4	832.5	436.6	569.2	570	698.4	712.7	659.2	515.5	1475
SUSPEN_D1	Me	13	15.1	17.4	12.8	15.2	20.6	11.1	16	3.9	31.6
SUSPEN_D1	MeC	13.7	12.6	17.2	15.1	13.6	20.3	12.1	16	6	32.4
FUSEEN	Ma	7.9	61.4	74.1	73.5	48.5	65.5	65.3	71.6	926.6	1041.8
FUSEEN	MaC	36.5	48.5	53.8	64.6	30.5	41	61.3	71.6	974.6	1091.5
MISSILE2	Ma	4.1	13.8	13.5	12.5	10.1	9.6	7.2	6	16.1	34.1
MISSILE2	MaC	4.8	9.1	10	12.5	10.1	9.6	7.2	6	18	32.7
MISSILE3	Ma	60	218.4	214.8	174.3	155	152.7	123.1	94.7	119.2	345.7
MISSILE3	MaC	90.9	140.4	155.2	174.3	155	152.7	123.1	94.7	123.6	303.1
MISSILE4	Ma	323.3	962.4	981	860	643.7	781.4	557.3	471.4	397.6	1 324.7
MISSILE4	MaC	546.2	614.2	763.7	860	643.7	781.4	557.3	471.4	475.8	1 382.9
SUSPEN_D1	Ma	4	7.9	9.3	3.8	7.4	16.1	10.8	9.7	302.3	323.9
SUSPEN_D1	MaC	9	4.2	12.9	2.3	5	16.4	10.9	9.7	300.2	335.7

times by better distributing the computational load. For example, the maximum condensation times for the mesh *MISSILE2* partitioned into 2 subdomains (*Ma*) is $T_{SD_2} = 41.4s$. For *MaC* the time T_{SD_2} is reduced to $20.2s$. Consequently, the global resolution time decreases from $42.5s$ to $21.2s$. There is a gain of 50%.

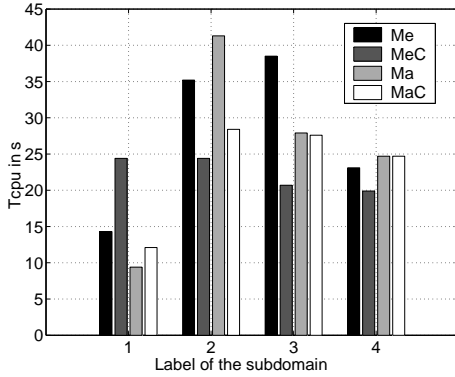


Figure 5: CPU condensation times of the mesh *MISSILE2* partitioned into 4 subdomains using the four methods

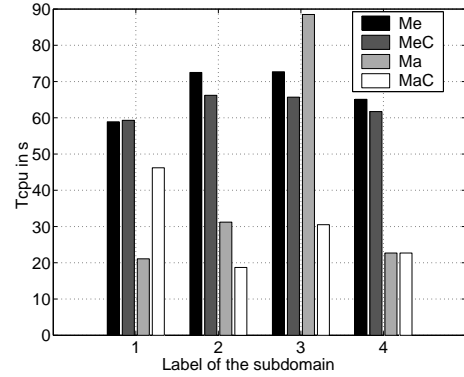


Figure 6: CPU condensation times of the mesh *SUSPEN_D1* partitioned into 4 subdomains using the four methods

For the mesh *MISSILE3* (table 5, column 11, lines 6 and 7) the gain obtained by balancing the subdomains is lost because the resolution time T_I of the interface problem increases from $182.3 s$ to $241.3 s$

s. The proposed method better balances the condensation times but increases the size of the interface. For a mesh partition into 2 or 4 subdomains, the gain obtained on the subdomain condensations is high enough to compensate the growth of the interface. But for a partition of 8 subdomains, the subdomains are initially better balanced and the resolution time of the interface problem becomes dominant.

For small number of subdomains, our algorithm permits to treat in core subdomains which are initially out-of-core. For the example *MISSILE3* (table 3, column 4, lines 6 and 7), we remark that it is impossible to treat in core the subdomain SD_2 of the initial partition. We also observe this phenomenon for the mesh *MISSILE4*: the subdomain SD_5 can be treated in core only after balancing (cf. table 5, lines 8 and 9, column 7).

By splitting into large number of subdomains, the condensation times are smoothed but the resolution time of the interface problem increases. Then, there exists a number of subdomains which gives the minimum global resolution time for a given partition and for the implementation of the solver. Our heuristics attempts to decrease the number of subdomains required to reach this minimum global resolution time.

We can remark the same phenomena for the partitions built with the method *Ma*, but this method builds partitions which have more interface nodes than those obtained by *Me*.

4 Conclusion

The basic implementation of our parallel multifrontal solver permits us to propose an estimator for use with our heuristics. Our algorithm improves mesh partitions issued from various partitioning tools by balancing the number of operations over the subdomains. The preliminary results have to be confirmed on other problems and by using different partitioning schemes. The modification of the boundary involved by the heuristics generally increases the number of the interface nodes. Consequently, the resolution time of the interface problem can increase. Further work is needed in order to limit the growth of the interface.

References

- [1] P. R. Amestoy, I. S. Duff, J. Y. L'Excellent, J. Koster, *A fully asynchronous multifrontal solver using distributed dynamic scheduling* SIAM Journal on Matrix Analysis and Applications, 23, (2001), 15–41.
- [2] I. S. Duff, J. K. Reid, J. A. Scott, *The Use of Profile Reduction Algorithms with a Frontal Code*, Int. J. for Num. Meth. in Eng., 28 (1989), 2555–2568.
- [3] T. A Davis, *User's guide to unsymmetric-pattern multifrontal package (UMFPACK, version 1.1)*, Tech. Report TR-95-004, CISE Dept., Univ. of Fl., Gainesville, FL, (1995).
- [4] S. Negre S., J. P Boufflet, *Partitionnement de maillage et équilibrage de charges pour un solveur parallèle multifrontal*, Actes de RenPar'10, Rencontre Francophone du Parallélisme des Architectures et des Systèmes, (1998), 187–190.
- [5] Y. Escaig, G. Touzot, *Application des méthodes de décomposition de domaines au calcul parallèle*, Revue Européenne des Eléments Finis, 6, 5, (1997), 589–609.
- [6] J. F Lassignardie, *Application du calcul distribué en mécanique des solides et des structures*, Phd Thesis, INSA de Rouen, (2001).
- [7] J. P Boufflet, P. Breitkopf, S. Negre, A. Rassinieux, M. Vayssade, *Improving domain decomposition for multifrontal resolution of mechanical problems: finite element reordering and load balancing issues*, Proceeding on CD-ROM of ECCOMAS 2000, European Congress on Computational Methods in Applied Sciences and Engineering, (2000).
- [8] B. M Irons, *A Frontal Solution Program for Finite Element Analysis*, Int. J. for Num. Meth. in Eng., 2, (1970), 5–32.
- [9] C. Farhat, M. Lesoinne, *Automatic Partitioning of Unstructured Meshes for the Parallel Solution of Problems in Computational Mechanics*, Int. J. for Num. Meth. in Eng., 36, (1993), 745–764.
- [10] B. Hendrickson, R. Leland, *The CHACO user's guide, version 2.0. Tech. Report*, Sandia National Laboratories, (1995).
- [11] G. Karypis, V. Kumar, *METIS a software package for partitioning unstructured graph, partitioning meshes, and computing fill-reducing orderings of sparses matrices*, Tech. Report. University of Minnesota, Dept. of Comp. Sci., (1998).
- [12] J. G Malone, *Automated Mesh Decomposition and Concurrent Finite Element Analysis For Hypercube Multiprocessors Computers*, Comp. Meth. Appl. Mech. Eng., 70, (1988), 27–58.
- [13] F. Pellegrini, *SCOTCH 3.1 User's guide*, Rapport du LaBRI, URA CNRS 1304, Université Bordeaux I, (1997).
- [14] S. Negre, J. P Boufflet, *Une heuristique pour l'équilibrage de charge d'un solveur parallèle multifrontal*, Rapport du laboratoire HeuDiaSys, UMR 6599, Université de Technologie de Compiègne, (2001).
- [15] P. Breitkopf, G. Touzot, *Architecture des Logiciels et Langages de Modélisation*, Revue Européenne des Eléments Finis, 1, 3, (1992), 333–368.