A parallel algorithm for direct solution of large sparse linear systems, well suitable to domain decomposition methods

Ibrahima Gueye* — Xavier Juvigny* — Frédéric Feyel*
François-Xavier Roux* — Georges Cailletaud**

** ONERA - Centre de Châtillon*
29, avenue de la Division Leclerc F-92322 Châtillon cedex
firstname.lastname@onera.fr

*** Centre des Matériaux P. M. FOURT, Mines ParisTech - UMR CNRS 7633*
BP 87, F-91003 Evry cedex
georges.cailletaud@ensmp.fr

ABSTRACT. *The goal of this paper is to develop a parallel algorithm for the direct solution of large sparse linear systems and integrate it into domain decomposition methods. The computational effort for these linear systems, often encountered in numerical simulation of structural mechanics problems by finite element codes, is very significant in terms of run-time and memory requirements.In this paper, a two-level parallelism is exploited. The exploitation of the lower level of parallelism is based on the development of a parallel direct solver with a nested dissection algorithm and to introduce it into the FETI methods. This direct solver has the advantage of handling zero-energy modes in floating structures automatically and properly. The upper level of parallelism is a coarse-grain parallelism between substructures of FETI. Some numerical tests are carried out to evaluate the performance of the direct solver.*

RÉSUMÉ. *Le but de ce papier est de mettre au point un algorithme parallèle pour la résolution directe de grands systèmes linéaires creux et l'intégrer dans les méthodes de décomposition de domaine. Ces systèmes linéaires, souvent rencontrés lors de la simulation numérique de problèmes de mécanique des structures par des codes de calcul par éléments finis, sont résolus avec des coûts très importants en temps de calcul et en espace mémoire. Dans ce papier, un parallélisme à deux niveaux a été exploité. L'exploitation du niveau inférieur de parallélisme a d'abord consisté à réaliser un solveur direct parallèle basé sur une technique de dissection emboîtée et à l'intégrer ensuite dans la méthode FETI. Ce solveur direct a l'avantage de traiter automatiquement et proprement les modes à énergie nulle dans des structures flottantes. Le niveau supérieur est un parallélisme efficace à gros grain entre les sous-structures de FETI. Des tests ont été effectués pour évaluer les performances du solveur direct.*

KEYWORDS: *two-level parallelism, parallel algorithm, nested dissection, zero-energy modes.*

MOTS-CLÉS : *parallélisme à deux niveaux, dissection emboîtée, modes à énergie nulle.*

DOI:10.3166/EJCM.18.589-605

1. Introduction

The direct resolution of large sparse linear systems has for long played a significant role in numerical simulation of many scientific computing problems, including problems in computational mechanics of structures. In structural mechanics, the use of finite element codes often leads to very large sparse linear systems. Solving these linear systems is really very expensive in terms of computational time and memory requirements.

The parallelization of these codes becomes a mandatory technique. It is able to reduce the computational cost and thus allows to simulate in structural mechanics large-scale models with increasingly complex constitutive material behaviour and refined geometries (geometry of components, level of detail of microstructure models). The domain decomposition methods are a natural way to parallelize these codes. One of the most commonly used method is FETI method[1] (Farhat *et al.*, 1994). It is a non-overlapping domain decomposition method based on a "dual approach" which consists to introduce continuity condition at the interfaces between subdomains. The FETI method has the advantage of being robust and well suited to the problems studied in structural mechanics. However, studies carried out on large numerical tests (tens of millions of unknowns) showed that the effectiveness of this method is decreasing beyond a number of subdomains (a few hundred). If we want to split large-scale models in a reasonable number of subdomains, we will be faced with very large local systems. Moreover, with the evolution of microprocessor technology in terms of power, we are witnessing the birth of new massively multi-core architectures. The essential interest and strength of multi-core solutions is to enable the simultaneous execution of a task by core. Some parallel algorithms and software can take full advantage of these massively multi-core systems.

It is in this context that this study was presented. It consists to implement a parallel algorithm for the direct solution of large sparse linear systems and integrate it into domain decomposition methods. A two-level parallelism is exploited. The exploitation of the lower level of parallelism is based, as first step, on a development of a parallel direct solver. We propose to use the technique of nested dissection for finding an elimination ordering in solving sparse systems (George, 1973), (George *et al.*, 1981), (Charrier *et al.*, 1989). This technique is well suited for parallel computing because it enables to reorder the linear system matrix and divide the factorization in as many steps as levels of dissection algorithm. As a second step, we introduce the parallel direct solver into FETI methods (Farhat *et al.*, 2000), (Gosselet *et al.*, 2006). This direct solver, based on multi-threading parallelism, allows us to do a parallel local resolution in FETI and has the advantage of handling zero-energy modes in floating substructures automatically and properly. The highest level is a coarse-grain parallelism between substructures of FETI. The iterative method used in solving FETI interface problem is adapted to the direct solver for improving the parallel efficiency at this level. We evaluate the performance (CPU and elapsed execution time required to perform the

1. Finite Element Tearing and Interconnecting.

factorization and, forward and backward substitution phasis) of the direct solver and compare it to a very effective linear solver: DSCPack solver[2] (Raghavan, 2001). To this end, large computational mechanics problems are performed with the Zebulon finite element analysis code, which is a solver for nonlinear mechanical problems developed jointly by ONERA[3], ENSMP and NW Numerics (Seattle, USA). This code has been parallelized with the FETI methods. In addition, the code uses in sequential a direct solver to factorize the linear systems.

The rest of the paper is structured as follows. In the next section we describe the various steps in the development of the parallel sparse direct solver. Therefore, we first present in details the proposed nested dissection algorithm. Next, we show how to use this technique for numerical factorization and solution phasis. In the case of presence of floating substructures, we describe the computation of the zero-energy modes. In Section 3, we evaluate the performance of the parallel direct solver and compare it to other direct solvers in Zebulon FEA code. We conclude in Section 4 with a few teachings for future steps after this study.

2. Parallel sparse direct solver

In this section, we develop a parallel sparse direct solver using a direct method based on an LU factorization of the sparse linear system matrix. The matrix can be symmetric or unsymmetric. In the case of a factorization of singular linear systems, a strategy is used to handle automatically and properly zero-energy modes. The ability of a solver to detect singularities is essential in domain decomposition methods for the operator itself or to build automatically optimal preconditioners. This solver consists of three major steps to resolve large sparse linear systems by direct method:

– The analysis step which computes a reordering and symbolic factorization of the matrix. The reordering techniques are used to minimize fill-in entries of the matrix during the numerical factorization and to exhibit as many independent calculations as possible. In this direct solver, a nested dissection reordering method is used. The choice to use essentially nested dissection method provided from the fact that this technique allows at first, more parallelism in the factorization than those using minimum degree techniques (Tinney *et al.*, 1967), but more importantly, it often produces better results. This analysis step produces an ordering matrix and an assembly tree (or supernodal elimination tree). This assembly tree is then used to carry out the subsequent steps.

– The numerical step determines the lower and upper triangular factors of the matrix according to the assembly tree that was previously produced.

– The third step is the solution phasis. Here, the numerical solution of the system is obtained by solving the lower and upper triangular systems resulting from factorization (forward elimination and backward substitution).

---

2. Domain Separator Codes Package.
3. French Aerospace Lab.

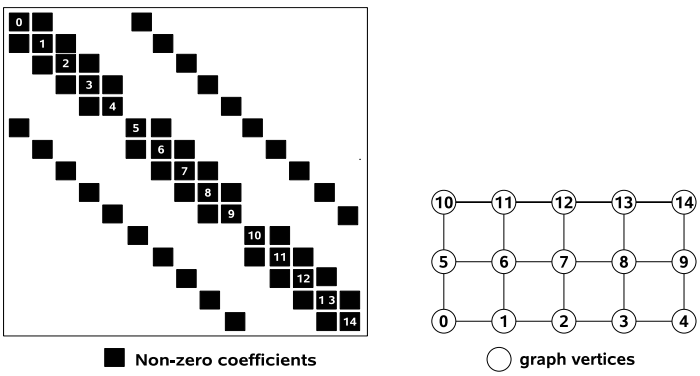
2.1. *Nested dissection*

The nested dissection technique is one of the most attractive methods for finding an elimination ordering in solving sparse linear systems. This technique orders, with the help of separators, the graph vertices (or mesh nodes) associated to the sparse system to be factorized. A graph $G(V, E)$ of a matrix consists of a set of vertices V and a set of edges E connecting the vertices. Each vertex of the graph represents an unknown of the sparse system. And a separator is a small set of vertices whose removal divides the rest of the graph or subgraph into two disjoint components. Different levels of separators are used to order the vertices. Lower level nodes are ordered before upper level ones. Note that here by a "level" we mean a set of separators in the same level of dissection.

2.1.1. *Principle*

The nested dissection approach is based on a recursive bisection of the graph of the matrix M to be factorized. A first bisection is performed by selecting a set of vertices forming a separator. This separator is then removed from the original graph, and this generates to it a partition into two unconnected subgraphs G_1 and G_2. The separator is chosen so that its size is as small as possible and that the sizes of the two subgraphs delimited are equivalent. Each of the two subgraphs is then bisected recursively following the same principle, until subgraphs are successively generated size sufficiently small.

To illustrate this nested dissection method, we consider a sparse linear system $Mx = b$, where the structure of M is presented in Figure 1(a). The graph (or mesh nodes) representing the matrix M is shown in Figure 1(b). We assume that we have a recursive sub-structuring of the mesh nodes (unknowns) into four subdomains represented in Figure 2.



(a) Structure of matrix M (b) Graph or mesh nodes

Figure 1. *Example of sparse system*

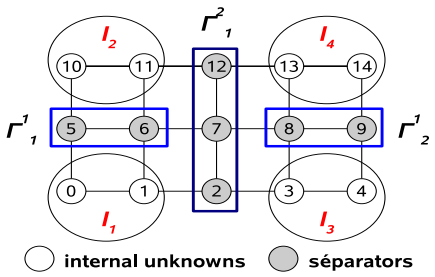


Figure 2. *Recursive sub-structuring by bisection*

Our nested dissection approach consists in ordering the unknowns of the sparse linear system in blocks. We start with the inner subdomains nodes and then we finish with the separators of the graph. The diagonal blocks $M_{I_iI_i}$ located at level 0 correspond to the subdomains I_1, I_2, I_3 and I_4 of the sub-structuring. The diagonal blocks $M_{\Gamma_j^l\Gamma_j^l}$, located at the other levels l ($l > 0$) correspond to the separators Γ_1^1, Γ_2^1 and Γ_1^2. The extra-diagonal blocks $M_{I_i\Gamma_j^l}$ and $M_{\Gamma_j^lI_i}$ correspond to the connections between the unknowns in subdomain I_i and those belonging to the separator Γ_j^l, and the extra-diagonal blocks $M_{\Gamma_i^l\Gamma_j^m}$ ($l \neq m$) correspond to the connections between the unknowns in separator Γ_i^l and the those in separator Γ_j^m (see Figure 3).

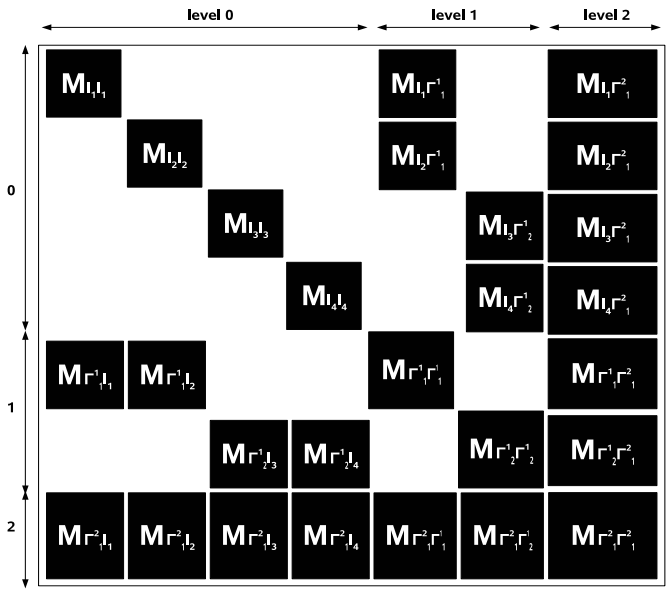


Figure 3. *Reordered Matrix M*

The nested dissection uses a "divide and conquer" approach and thus, it makes it this algorithm easily to parallelize. It generates balanced supernodal elimination trees whose supernodes are sets of unknowns (see Figure 4). These trees reflect the dependency of unknowns during elimination and therefore makes easier the execution of the parallel numerical factorization of the reordered matrix M.

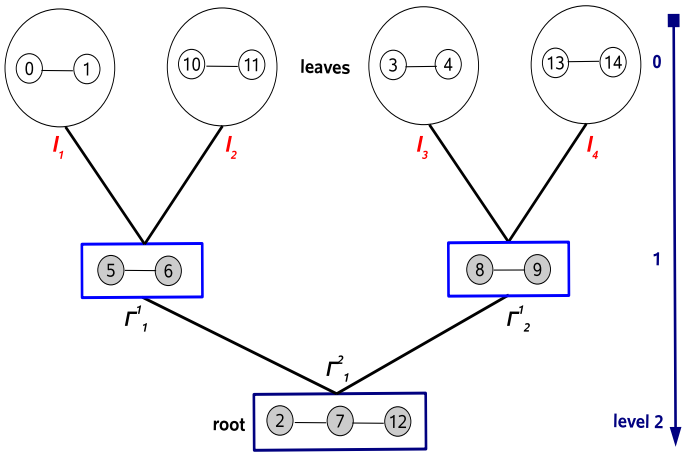


Figure 4. *Supernodal elimination tree*

2.1.2. *Construction of assembly trees*

An assembly tree is a supernodal elimination tree where, on each node, we classify the unknowns into a set of internal unknowns and a set of interface unknowns.

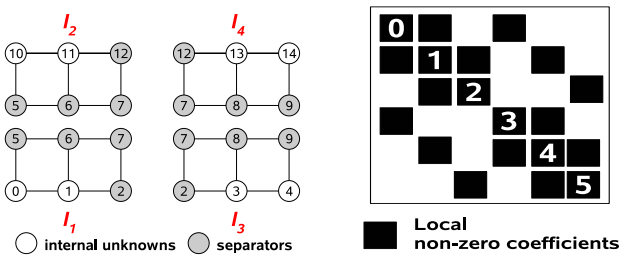
– The internal unknowns are the ones that do not belong to the separator of a parent node in the tree.

– The interface unknowns are exactly the ones that will become separator unknowns on a parent node.

An important aspect of the assembly trees is that they only define a partial ordering for the factorization. These trees are used to carry out the numerical and the solution phasis of the sparse linear systems. We have two approaches to build assembly trees.

In the first approach, we first partition the global finite element mesh into a set of sub-meshes, and then dispatch unknowns (degrees of freedom) into the sub-meshes according to their element ownership. So we get a decomposition of the initial mesh into subdomains (sets of unknowns). The local matrix associated with the subdomain I_i will not have the same structure as the global matrix. An example of decomposition is shown in Figure 5(a) and Figure 5(b). In this approach, the mesh partitioning is managed by the METIS partitioner (Karypis *et al.*, 1995), which does not support all the history on the partition. We use a genetic algorithm (Bui *et al.*, 1996) to build directly an assembly tree that contains all the history on the partition. This genetic

algorithm starts with a population constituted by the subdomains I_i of the decomposition.



(a) Subdomains (b) A local matrix structure

Figure 5. *Decomposition into four subdomains*

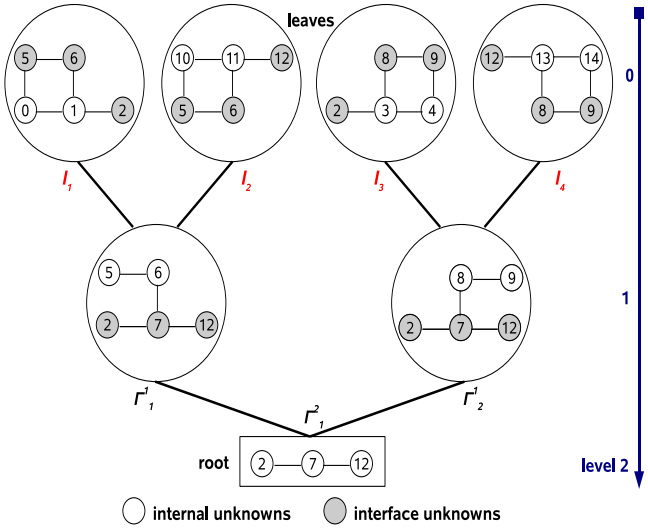


Figure 6. *Assembly tree*

The second approach is based on a recursive bisection of the original graph using a modified METIS partitioner which does support all the history on the partition. This recursive bisection generates a supernodal elimination tree. We use the supernodal elimination tree to construct a assembly tree. For each node I_i (resp. Γ_j^l) in the assembly tree, its internal unknowns are those unknowns on the subdomain I_i (resp. separator Γ_j^l) defined in the supernodal elimination tree (Figure 4). The interface unknowns are the ones that will become separator unknowns on a parent node. To find the interface unknowns of a node I_i (resp. Γ_j^l) of the assembly tree, we search all the dependencies between his internal unknowns and those of his ancestors (nodes on the higher levels). For a given internal unknown n_i in I_i, if there is a dependency with

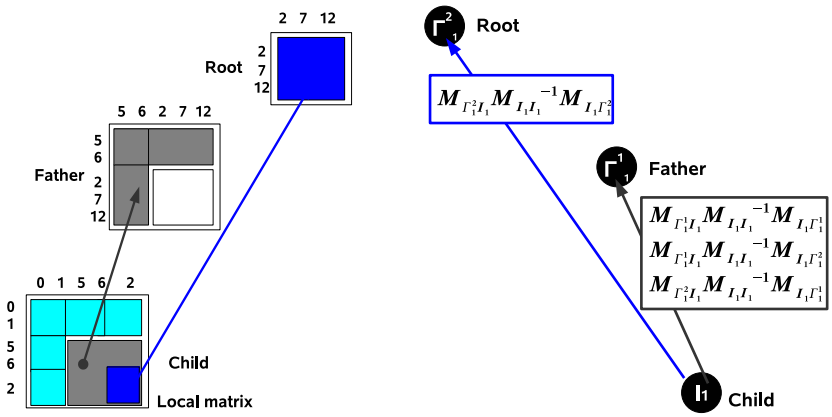
a internal unknown n_j of a ancestor, then n_j is an interface unknown of I_i (resp. Γ_j^l). At the end of the research, we get an assembly tree as is illustrated in Figure 6. Our choice is focused on this second approach because in it we already have all the history on the recursive sub-structuring of the graph (supernodal elimination tree). It allows a faster construction of an assembly tree. In this second approach, the separators are smaller than those obtained in the first approach. This produces a better result at the numerical factorization and the solution phasis.

2.2. *Numerical phasis*

An important step for implementing the direct solver is the numerical phasis. It consists to determine implicitly the lower and upper triangular factors of the sparse matrix system M. We first proceed to the static condensation on the separator unknowns, and then go on to the numerical factorization of M.

2.2.1. *Static condensation*

The static condensation computes a local system for each node in the assembly tree. The first step consists in eliminating the internal unknowns in each subdomain I_i of the tree. The elimination of the internal unknowns is equivalent to computing the local Schur complements or contributions $M_{\Gamma_j^l I_i} {M_{I_i I_i}}^{-1} M_{I_i \Gamma_j^l}$ and $M_{\Gamma_j^l I_i} {M_{I_i I_i}}^{-1} M_{I_i \Gamma_k^m}$ in each subdomain I_i. The diagonal blocks $S_{\Gamma_j^l \Gamma_j^l}$ and the extra-diagonal blocks $S_{\Gamma_j^l \Gamma_k^m}$ and $S_{\Gamma_k^m \Gamma_j^l}$ in the higher levels will then be created by assembling the lower level Schur complements (l and $m > 0$).



(a) Assembly process (b) Contributions computing

Figure 7. *Assembly and computational process: illustration for* I_1

In Figure 7(a) and Figure 7(b), this first step is illustrated for subdomain I_1. Then, we show how the process to assembly and compute the local Schur complements is performed.

The second step consists in eliminating, level by level, the internal unknowns in each node Γ_j^l in the higher levels l of the tree ($l > 0$). The elimination starts at the level 1 and, at each level, the process is the same as in the first step. We first perform an Gaussian elimination on the diagonal blocks $S_{\Gamma_j^l \Gamma_j^l}$ created in the first step and then, we compute the contribution blocks $S_{\Gamma_k^m \Gamma_j^l} {S_{\Gamma_j^l \Gamma_j^l}}^{-1} S_{\Gamma_j^l \Gamma_k^m}$ and $S_{\Gamma_k^m \Gamma_j^l} {S_{\Gamma_j^l \Gamma_j^l}}^{-1} S_{\Gamma_j^l \Gamma_p^n}$ in each node Γ_j^l. The diagonal blocks $S_{\Gamma_k^m \Gamma_k^m}$ and the extra-diagonal blocks $S_{\Gamma_k^m \Gamma_p^n}$ and $S_{\Gamma_p^n \Gamma_k^m}$ in the following levels are updated (m and $n > l$).

A standard assembling algorithm applied to the blocks $S_{\Gamma_k^m \Gamma_p^n}$ previously created gives the global Schur complement S. The assembly algorithm is done step by step. Theses blocks are fully populated because each contribution block is a dense matrix. However, the global Schur complement is sparse and is never assembled explicitly. In order to compute efficiently each local Schur complement, we can use the BLAS level 3 routines (Dongarra *et al.*, 1990) based on block computations.

$$S = \begin{pmatrix} S_{\Gamma_1^1 \Gamma_1^1} & 0 & S_{\Gamma_1^1 \Gamma_1^2} \\ 0 & S_{\Gamma_2^1 \Gamma_2^1} & S_{\Gamma_2^1 \Gamma_1^2} \\ S_{\Gamma_1^2 \Gamma_1^1} & S_{\Gamma_1^2 \Gamma_2^1} & S_{\Gamma_1^2 \Gamma_1^2} \end{pmatrix} \qquad [1]$$

For the example given in Section 2.1, the global Schur complement of M is given by equation [1]. For the first step of the algorithm, we have at level 1:

$$\begin{cases} S_{\Gamma_1^1 \Gamma_1^1} = M_{\Gamma_1^1 \Gamma_1^1} - \sum_{i=1}^{2} M_{\Gamma_1^1 I_i} {M_{I_i I_i}}^{-1} M_{I_i \Gamma_1^1} \\ S_{\Gamma_1^1 \Gamma_1^2} = M_{\Gamma_1^1 \Gamma_1^2} - \sum_{i=1}^{2} M_{\Gamma_1^1 I_i} {M_{I_i I_i}}^{-1} M_{I_i \Gamma_1^2} \\ S_{\Gamma_1^2 \Gamma_1^1} = M_{\Gamma_1^2 \Gamma_1^1} - \sum_{i=1}^{2} M_{\Gamma_1^2 I_i} {M_{I_i I_i}}^{-1} M_{I_i \Gamma_1^1} \end{cases}$$

and

$$\begin{cases} S_{\Gamma_2^1 \Gamma_2^1} = M_{\Gamma_2^1 \Gamma_2^1} - \sum_{i=3}^{4} M_{\Gamma_2^1 I_i} {M_{I_i I_i}}^{-1} M_{I_i \Gamma_2^1} \\ S_{\Gamma_2^1 \Gamma_1^2} = M_{\Gamma_2^1 \Gamma_1^2} - \sum_{i=3}^{4} M_{\Gamma_2^1 I_i} {M_{I_i I_i}}^{-1} M_{I_i \Gamma_1^2} \\ S_{\Gamma_1^2 \Gamma_2^1} = M_{\Gamma_1^2 \Gamma_2^1} - \sum_{i=3}^{4} M_{\Gamma_1^2 I_i} {M_{I_i I_i}}^{-1} M_{I_i \Gamma_2^1} \end{cases}$$

At the root (level 2), we have:

$$S_{\Gamma_1^2\Gamma_1^2} = M_{\Gamma_1^2\Gamma_1^2} - \sum_{i=1}^{4} M_{\Gamma_1^2 I_i} {M_{I_i I_i}}^{-1} M_{I_i\Gamma_1^2}$$

The next step of the algorithm corresponds to the elimination of internal unknowns in each node Γ_j^1 located at level 1 of the assembly tree. We repeat the same process to assemble the global Schur complement at level 2 of the tree (root). Then, we have:

$$S_{\Gamma_1^2\Gamma_1^2} = S_{\Gamma_1^2\Gamma_1^2} - \sum_{j=1}^{2} S_{\Gamma_1^2\Gamma_j^1} {S_{\Gamma_j^1\Gamma_j^1}}^{-1} S_{\Gamma_j^1\Gamma_1^2}$$

2.2.2. *Numerical factorization*

The numerical factorization is performed by computing implicitly the lower triangular factor L and the upper triangular factor U of the sparse matrix system M using an assembly tree. We first perform a sparse LU factorization of $M_{I_iI_i}$ for each substructure I_i in the form $L_{I_iI_i}U_{I_iI_i}$. Next, we perform a dense LU factorization of $S_{\Gamma_j^l\Gamma_j^l}$ for each separator Γ_j^l in the form $L_{\Gamma_j^l\Gamma_j^l}U_{\Gamma_j^l\Gamma_j^l}$. Dimensions of the diagonal blocks $M_{I_iI_i}$ and $S_{\Gamma_j^l\Gamma_j^l}$ are often small. A skyline or frontal solver is used to factorize them beacause these solvers are more efficient. During the factorization, the elimination of internal unknowns in each node can be done, as soon as those in children have been treated.

For the example of Section 2.1, we can write the triangular matrix L and U in the following forms. The structure of these triangular matrix is similar to that of M described in Figure 3.

$$L = \begin{pmatrix}
M_{I_1I_1} & & & & & & \\
0 & M_{I_2I_2} & & & & & \\
0 & 0 & M_{I_3I_3} & & & & \\
0 & 0 & 0 & M_{I_4I_4} & & & \\
M_{\Gamma_1^1I_1} & M_{\Gamma_1^1I_2} & 0 & 0 & S_{\Gamma_1^1\Gamma_1^1} & & \\
0 & 0 & M_{\Gamma_2^1I_3} & M_{\Gamma_2^1I_4} & 0 & S_{\Gamma_2^1\Gamma_2^1} & \\
M_{\Gamma_1^2I_1} & M_{\Gamma_1^2I_2} & M_{\Gamma_1^2I_3} & M_{\Gamma_1^2I_4} & S_{\Gamma_1^2\Gamma_1^1} & S_{\Gamma_1^2\Gamma_2^1} & S_{\Gamma_1^2\Gamma_1^2}
\end{pmatrix}$$

and

$$U = \begin{pmatrix}
I & 0 & 0 & 0 & M_{I_1I_1}^{-1}M_{I_1\Gamma_1^1} & 0 & M_{I_1I_1}^{-1}M_{I_1\Gamma_1^2} \\
 & I & 0 & 0 & M_{I_2I_2}^{-1}M_{I_2\Gamma_1^1} & 0 & M_{I_2I_2}^{-1}M_{I_2\Gamma_1^2} \\
 & & I & 0 & 0 & M_{I_3I_3}^{-1}M_{I_3\Gamma_2^1} & M_{I_3I_3}^{-1}M_{I_3\Gamma_1^2} \\
 & & & I & 0 & M_{I_4I_4}^{-1}M_{I_4\Gamma_1^2} & M_{I_4I_4}^{-1}M_{I_4\Gamma_1^2} \\
 & & & & I & 0 & S_{\Gamma_1^1\Gamma_1^1}^{-1}S_{\Gamma_1^1\Gamma_1^2} \\
 & & & & & I & S_{\Gamma_2^1\Gamma_2^1}^{-1}S_{\Gamma_2^1\Gamma_1^2} \\
 & & & & & & I
\end{pmatrix}$$

2.3. *Resolution of triangular systems*

The solution of the system $Mx = b$ is determined by solving the triangular systems $Ly = b$ and $Ux = y$. Here L and U are obtained from the numerical factorization of the sparse matrix M. For the example in Section 2.1, the solution of these triangular systems is guided by an assembly tree and carried out in three phasis:

– a forward elimination phasis, which solves the local systems $M_{I_iI_i}y_{I_i} = (L_{I_iI_i}U_{I_iI_i})y_{I_i} = b_{I_i}$ ($i = 1$, 2, 3 and 4) and updates the terms $b_{\Gamma_j^l}$ of the right-hand side b:

$$\begin{pmatrix} b_{\Gamma_1^1} \\ b_{\Gamma_2^1} \\ b_{\Gamma_1^2} \end{pmatrix} = \begin{pmatrix} b_{\Gamma_1^1} \\ b_{\Gamma_2^1} \\ b_{\Gamma_1^2} \end{pmatrix} - \begin{pmatrix} M_{\Gamma_1^1 I_1}y_{I_1} + M_{\Gamma_1^1 I_2}y_{I_2} \\ M_{\Gamma_2^1 I_3}y_{I_3} + M_{\Gamma_2^1 I_4}y_{I_4} \\ \sum_{i=1}^{4} M_{\Gamma_1^2 I_i}y_{I_i} \end{pmatrix} \quad [2]$$

– a solution phasis of the condensed problem on interface, where $S_{\Gamma_j^l\Gamma_j^l} = L_{\Gamma_j^l\Gamma_j^l}U_{\Gamma_j^l\Gamma_j^l}$. This phasis is carried out level by level:

$$\begin{pmatrix} S_{\Gamma_1^1\Gamma_1^1} & 0 & 0 \\ 0 & S_{\Gamma_2^1\Gamma_2^1} & 0 \\ S_{\Gamma_1^2\Gamma_1^1} & S_{\Gamma_1^2\Gamma_2^1} & S_{\Gamma_1^2\Gamma_1^2} \end{pmatrix} \begin{pmatrix} I & 0 & S^{-1}_{\Gamma_1^1\Gamma_1^1}S_{\Gamma_1^1\Gamma_1^2} \\ 0 & I & S^{-1}_{\Gamma_2^1\Gamma_2^1}S_{\Gamma_2^1\Gamma_1^2} \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} x_{\Gamma_1^1} \\ x_{\Gamma_2^1} \\ x_{\Gamma_1^2} \end{pmatrix} = \begin{pmatrix} b_{\Gamma_1^1} \\ b_{\Gamma_2^1} \\ b_{\Gamma_1^2} \end{pmatrix} \quad [3]$$

– a backward substitution phasis, which computes the local solutions:

$$\begin{pmatrix} x_{I_1} \\ x_{I_2} \\ x_{I_3} \\ x_{I_4} \end{pmatrix} = \begin{pmatrix} y_{I_1} \\ y_{I_2} \\ y_{I_3} \\ y_{I_4} \end{pmatrix} - \begin{pmatrix} M^{-1}_{I_1I_1}M_{I_1\Gamma_1^1}x_{\Gamma_1^1} + M^{-1}_{I_1I_1}M_{I_1\Gamma_1^2}x_{\Gamma_1^2} \\ M^{-1}_{I_2I_2}M_{I_2\Gamma_1^1}x_{\Gamma_1^1} + M^{-1}_{I_2I_2}M_{I_2\Gamma_1^2}x_{\Gamma_1^2} \\ M^{-1}_{I_3I_3}M_{I_3\Gamma_2^1}x_{\Gamma_2^1} + M^{-1}_{I_3I_3}M_{I_3\Gamma_1^2}x_{\Gamma_1^2} \\ M^{-1}_{I_4I_4}M_{I_4\Gamma_2^1}x_{\Gamma_2^1} + M^{-1}_{I_4I_4}M_{I_4\Gamma_1^2}x_{\Gamma_1^2} \end{pmatrix} \quad [4]$$

2.4. *Taking into account singular linear systems*

There are many sequential or parallel direct solvers, but no or few can detect automatically and properly zero-energy modes for singular linear systems. Our goal is to implement a direct solver, which handles automatically and properly singularities in case of presence of zero-energy modes in the floating substructures. These movements can have a geometrical or physical origin, or appear when splitting the structure into substructures. The approach for computing the zero-energy modes consists in detecting zero pivots during the numerical factorization of M. The computation starts at the leaves of the assembly tree and progresses up to the root. First, we check if near zero pivots appear when factorizing each local matrix $M_{I_iI_i}$ at level 0. If we found a near zero pivot at the k^{th} row and column, then we block the degree of freedom k. The blocking is to set to zero the k^{th} row of $M_{I_iI_i}$ (resp. $M_{I_i\Gamma_j^l}$) and the k^{th} column of

$M_{I_iI_i}$ (resp. $M_{\Gamma_j^l I_i}$). The diagonal entrie $M_{I_iI_i}(k,k)$ is set to one. Next, we proceed with the higher levels ($l > 0$) following the same process. Then, the diagonal block $S_{\Gamma_j^l\Gamma_j^l}$ of node Γ_j^l is factorized. If a near zero pivot is found at its p^{th} row and column, the unknown p is blocked. The p^{th} row of $S_{\Gamma_j^l\Gamma_j^l}$ (resp. $S_{\Gamma_j^l\Gamma_i^m}$) and the p^{th} column of $S_{\Gamma_j^l\Gamma_j^l}$ (resp. $S_{\Gamma_i^m\Gamma_j^l}$) are set to zero. The diagonal entrie $S_{\Gamma_j^l\Gamma_j^l}(p,p)$ is set to one.

At the end of the factorization, we obtain a list $SingVals$ containing $nsing$ near zero pivots, which are candidates to be zero-energy modes of the global matrix M. Indeed, some zero pivots encountered when we factorize the local matrix $M_{I_iI_i}$ could be undesirable effects caused by the recursive sub-structuring of the graph. To compute the zero-energy modes, we first condense the global system on the singular unknowns of $SingVals$ to obtain a small Schur complement S_s. Then, we perform an Gaussan elimination with full pivoting on S_s and check if zero pivots are found. If we don't find zero pivots, then the matrix M is non-singular. If we find a number e of zero pivots ($0 < e < nsing$), then these pivots are the actual zero-energy modes and we unblock the $nsing - e$ degrees of freedom which were blocked during the factorization of M. A basis of M null-space is built using its e corresponding rows and columns.

Finally, we find the general solution of the sparse linear system $Mx = b$ in the form $x = M^+b + N\alpha$, where $\alpha \in \mathbb{R}^e$ is a vector of e arbitrary entries and the vector M^+b is a particular solution of the linear system.

2.5. *Direct solver parallelization*

The "divide and conquer" strategy of the nested dissection technique leads to a high-level parallelism. The parallelization of the direct solver is done in a natural way. Given that there are no dependencies on internal unknowns belonging to different nodes located at the same level on the tree, we treat these nodes at the same time on separate sets of cores. The approach selected for parallelizing the solver consists to use multi-threading technology based on multi-tasks, which works on shared memory nodes. POSIX threads and a OpenMP threading software (Intel MKL) are used to implement multi-threads in the static condensation phasis, the numerical factorization and the solution of triangular systems. We also use a mixed programming model for taking advantage of the benefits of both models POSIX threads and OpenMP in the numerical factorization.

3. Performance evaluation

The performance of a direct solver can be evaluated in different ways. Here, we use the CPU and Elapsed time required to perform analysis, numerical and solution phasis with the implemented direct solver. We compare its sequential performance with other direct solvers in Zebulon finite element analysis code and we analyse its performance in Zebulon FEA parallel code. For the evaluation, we simulate 3-D linear elasticity problems. The following sequential tests were performed on a bi-processors Intel Quad-Core Xeon X5460 64-bit machine, with 32 GB of memory and 3.16 GHz

frequency. the parallel tests were performed on a Linux cluster with 51 machines (bi-processors AMD Opteron 64-bit with a memory of 2 to 16 GB).

3.1. *Sequential performance analysis*

For analysing our Dissection solver performance, we first compare its the CPU execution times with other ones. Next, we evaluate the CPU times required to perform forward elimination and backward substitution for solving linear systems with multiple right-hand sides.

Table 1 shows the CPU execution times for some direct solvers available in Zebulon FEA code. DSCPack is a direct solver based on an approach similar to the principle of nested dissection but, it doesn't take into account singular linear systems. On the other side, Sparse Direct and Frontal are two other direct solvers that take into account singularities of these systems.

Table 1. *CPU execution time (sec.)*

Solvers \ Problems size	107811	206763	397953
DSCPack	45.41	154.2	557.8
Dissection	67.17	207.9	776.2
Sparse Direct	871.7	3292	+10440
Frontal	2185	9972	***

On the results in Table 1, we observe that the performance of DSCPack are slightly better than our Dissection solver. But we should not be ashamed of this fact since DSCPack solver can't handle the zero-energy modes in the floating substructures. And that's where the solver that we have implemented becomes more profitable for FETI methods than other existing solvers in Zebulon FEA code.

Dissection solver can be used for solving linear systems with multiple right-hand sides. In Table 2, we show the performance of forward and backward substitution phasis for a problem with 206763 degrees of freedom. The number of right hand sides ($Nrhs$) ranges from 1 to 2000.

Table 2. *Forward and backward substitution time (sec.)*

Solvers \ Nrhs	1	50	100	150	2000
DSCPack	1.246	62.32	124.6	186.9	249.3
Dissection	2.627	58.47	114.9	172.7	231.1

The analysis of theses results allows to see that we have good performance with the new Dissection solver in the forward and backward substitution phasis. This could be of great interest for improving for solving the FETI interface problem, where several forward and backward substitutions are performed successively to satisfy the continuity of the solution across the interfaces between subdomains.

3.2. *Parallel performance analysis*

We analyze the performance obtained for solving a large-scale linear elasticity problem with 397953 degrees of freedom. In Figure 8, we present the elapsed execution time for the parallel versions implemented. These parallel versions are:

– a multi-threads version with POSIX threads;

– a version using an OpenMP threading software;

– a mixed version with POSIX threads and OpenMP.

For this problem, we achieve a optimum gain factor of about 2.5 in execution time when the number of threads is equal to 4.

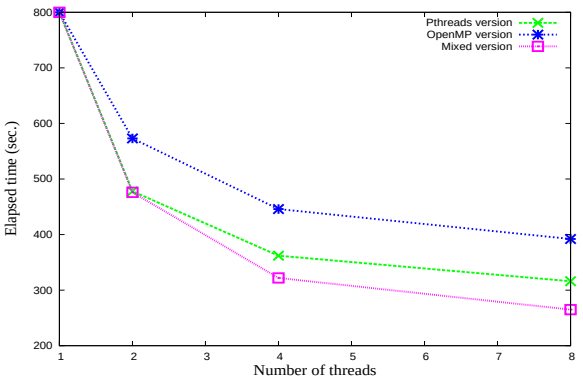


Figure 8. *Elapsed execution time*

Dissection solver has been choosen as local solver in FETI methods implemented in Zebulon FEA parallel code. Some parallel tests were performed. Our tests are linear elasticity problems. The global problem size grows linearly with the number of subdomains (see Figure 9).

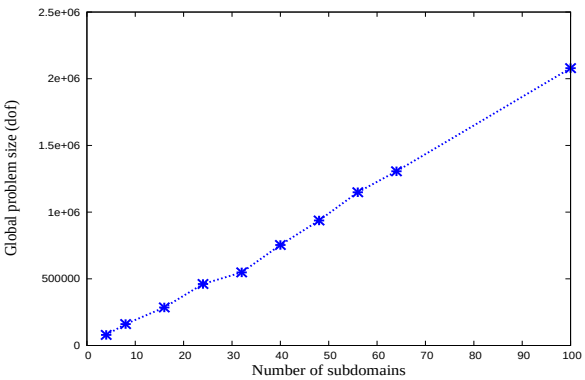


Figure 9. *Global problems: size*

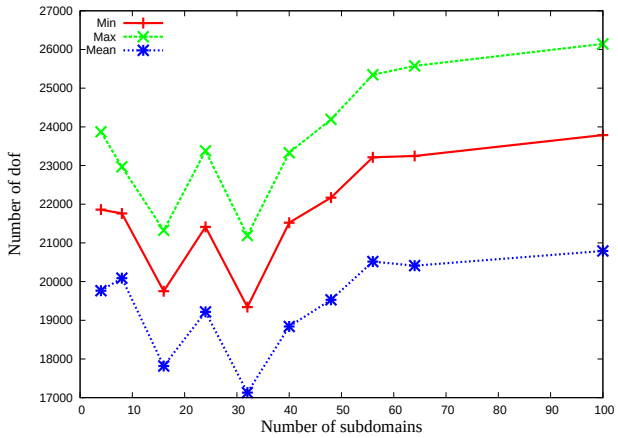


Figure 10. *Size per subdomain*

The results are presented in Figure 11 and Figure 12. An analysis of these results shows that it's easy to get high performance by solving much larger problems. When the number of subdomains grows the communication time increases too, thus lowering FETI performance. In this example, it means that the subdomain sizes (around 20000 dof, see Figure 10) are too low. The execution time (min or max) required to obtain local solutions is small compared to the communication time. The result is clear: we must use bigger subdomains and a parallel version of the Dissection solver for getting high performance. All these tests are done without multi-threading.

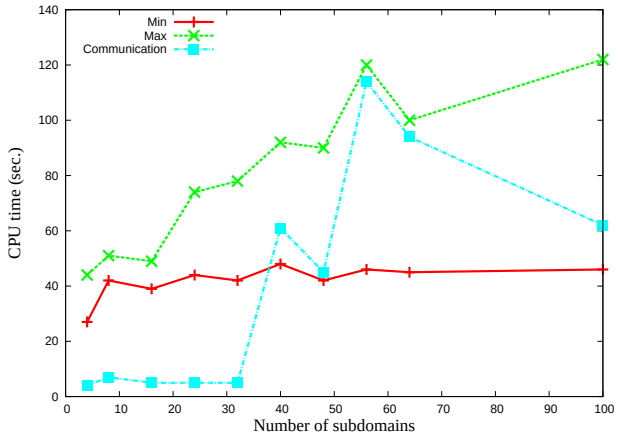


Figure 11. *Solver execution time and communication*

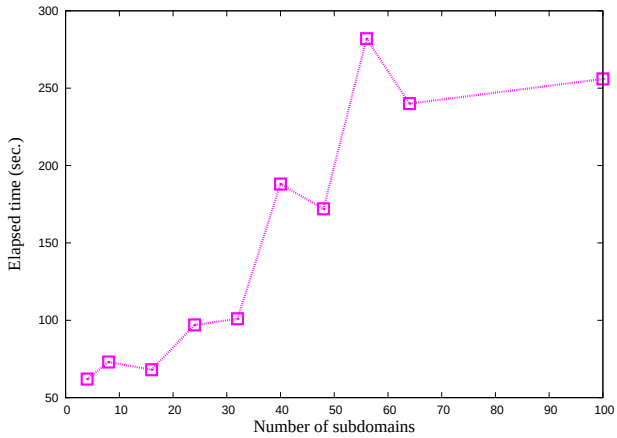


Figure 12. *Total execution time*

4. Conclusion

In this paper, we have implemented a parallel direct solver based on a multi-threading technology and using a nested dissection reordering method, which leads to a high-level parallelism. This solver has been integrated into FETI methods and it handles automatically and properly zero-energy modes in floating substructures. Large-scale mechanical problems have been simulated with Zebulon finite element analysis parallel code and we have got good numerical performance results with the help of our direct solver.

In this paper, we limited the study to the exploitation of the lower level of parallelism. Our future work aims to improve the iterative method used in solving FETI interface problem using the direct solver we developed for achieving maximum performance in ZebuLon FEA parallel code.

5. References

Bui T. N., Moon B.-R., " Genetic algorithm and graph partitioning", *IEEE Transactions on Computers*, vol. 45, n° 7, p. 841-855, 1996.

Charrier P., Roman J., " Algorithmique et calculs de complexité pour un solveur de type dissections emboîtées", *Numerische Mathematik*, vol. 55, p. 463-476, 1989.

Dongarra J. J., Duff I. S., Croz J. D., Hammarling S., " A set of Level 3 Basic Linear Algebra Subprograms", *ACM Transactions on Mathematical Software*, vol. 16, p. 1-17, 1990.

Farhat C., Pierson K., Lesoinne M., " The second generation FETI methods and their application to the parallel solution of large-scale linear and geometrically non-linear structural analysis problems", *Computer Methods in Applied Mechanics and Engineering*, vol. 184, n° 2-4, p. 333-374, 2000.

Farhat C., Roux F.-X., " Implicit parallel processing in structural mechanics", *Computational Mechanics Advances*, vol. 2, n° 1, p. 1-124, 1994.

George A., " Nested dissection of a regular finite element mesh", *SIAM Journal on Numerical Analysis*, vol. 10, n° 2, p. 345-363, 1973.

George A., Liu J. W.-H., *Computer solution of large sparse positive definite systems*, Prentice Hall, 1981.

Gosselet P., REY C., " Non-overlapping domain decomposition methods in structural mechanics", *Archives of Computational Methods in Engineering*, vol. 13, n° 4, p. 515-572, 2006.

Karypis G., Kumar V., " Metis: Unstructured graph partitioning and sparse matrix ordering system", *http://www-users.cs.umn.edu/ karypis/metis*, 1995.

Karypis G., Kumar V., " A fast and high quality multilevel scheme for partitioning irregular graph", *SIAM Journal on Scientific Computing*, vol. 20, n° 1, p. 359-392, 1999.

Liu J. W.-H., " The role of elimination trees in sparse factorization", *SIAM Journal of Matrix Analysis and Applications*, vol. 11, n° 1, p. 132-172, 1990.

Raghavan P., " DSCPACK home page", *http://www.cse.psu.edu/ raghavan/Dscpack*, 2001.

Tinney W. F., Walker J. W., " Direct solutions of sparse network equations by optimally ordered triangular factorization", *Proceedings of the IEEE*, vol. 55, n° 11, p. 1801-1809, 1967.

Received: 17 June 2008
Accepted: 18 June 2009