Computer Assisted Mechanics and Engineering Sciences, 3: 1–8, 1996.

Optimization of direct domain decomposition methods

Yves Escaig and Gilbert Touzot
INSA de Rouen, Place Emile Blondel, BP 8, 76131 Mont Saint Aignan, France

(Received July 21, 1995)

In this paper, the problem of parallelizing the finite element method for distributed memory computers using domain decomposition methods is addressed. We focus on direct domain decomposition methods because they are robust and well adapted to multi-level decompositions. Two important problems concerning these methods are studied: the condensation of subdomains and the resolution of the interface problem. Finally, results are presented which show that, in sequential implementation, direct domain decomposition methods are more efficient than standard LDL^t-skyline solvers, and compare favorably with state-of-the-art LDL^t-sparse solvers.

1. INTRODUCTION

In this paper, we focus on the solving of large scale problems of structural mechanics, using the finite element method. One of the major challenges for this method is its efficient implementation on parallel computers. The most common approach to parallelization is the use of domain decomposition methods. However, any advocated parallel processing method has to be competitive with the best sequential algorithm. It is the purpose of this paper to study the domain decomposition method and compare its performance with state-of-the-art sequential methods.

Basically, domain decomposition methods solve the initial problem independently on each subdomain, and add constraints to fit the local solutions at the interfaces between the subdomains.

These constraints can be:

- equality of the forces, which leads to the well-known Schur complement method
- equality of the displacements, which leads to the FETI (Finite Element Tearing and Interconnecting) method
- hybrid combinations of the two previous techniques

Detailed references can be found in [8]. For the Schur complement method, the decomposition algorithm can be written in the case of two subdomains as follows:

1. Renumber equations so that the internal degrees of freedom (d.o.f.) of each subdomain appear first

$$[k](u) = (f) \Longleftrightarrow \begin{bmatrix} k_{11} & 0 & k_{13} \\ 0 & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$

2. Eliminate all internal d.o.f.

$$\begin{bmatrix} k_{11} & 0 & k_{13} \\ 0 & k_{22} & k_{23} \\ 0 & 0 & \overline{k_{33}} \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \overline{f_3} \end{pmatrix} \tag{1}$$

where

$$\overline{k_{33}} = k_{33} - k_{31}k_{11}^{-1}k_{13} - k_{32}k_{22}^{-1}k_{23} \tag{2}$$

$$\overline{f_3} = f_3 - k_{31}k_{11}^{-1}f_1 - k_{32}k_{22}^{-1}f_2 \tag{3}$$

3. Solve the interface problem

$$\left[k_{33} - k_{31}k_{11}^{-1}k_{13} - k_{32}k_{22}^{-1}k_{23}\right](u_3) = \left(f_3 - k_{31}k_{11}^{-1}f_1 - k_{32}k_{22}^{-1}f_2\right) \tag{4}$$

4. back-substitute the interface solution on the internal d.o.f.

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} k_{11}^{-1}f_1 + k_{11}^{-1}k_{13}u_3 \\ k_{22}^{-1}f_2 + k_{22}^{-1}k_{23}u_3 \\ u_3 \end{pmatrix}$$

It is possible to satisfy these constraints iteratively, i.e. solving the linear system (4) using iterative methods [12, 14]. This strategy has the advantage that it does not require the construction of the matrix $\overline{k_{33}}$, which saves computational time and reduces memory usage. However, each iteration at the interface level requires the resolution on each subdomain of the linear system (2) corresponding to the internal d.o.f. This is best achieved if these linear systems are factorized. But even in this case, the number of iterations at the interface level must be kept to a minimum. Therefore, many researches have been allured to develop very efficient preconditioners (see proceedings of the conferences on domain decomposition methods, for example [10]). But the performances of these preconditioners are strongly problem dependent (linear, nonlinear, 3D, shells, ...).

It is also possible to solve the linear system (4) using a direct method. This strategy is robust, well adapted to multi-level (or hierarchical) decomposition. It can be used for ill-conditioned problems and for solving of several problems corresponding to different right hand sides (especially useful for multiple load cases or parametric design).

In this case, domain decomposition can be seen as a block LU factorization. But domain decomposition methods (direct or iterative) offer additional advantages:

- possibility of local modifications of one subdomain without recalculating the whole problem;
- re-use of subdomains, possibly with geometrical transformations;
- independent definitions of subdomains, provided that a method for dealing with non-conforming interfaces is available;
- possibility of limitating mesh refinements to a subset of subdomains when using auto-adaptive methods.

Because of the vital importance of direct solvers for industrial applications, we chose to concentrate on direct methods for solving the interface problem. This choice implies two major difficulties. First, it is necessary to build the stiffness matrix of the interface problem, which requires inversion of the subdomains stiffness matrices and not only factoring them (see Eq. (4)). One way to overcome this difficulty is to use a frontal elimination method inside each subdomain (see the next section). The second difficulty involves solving of the interface problem itself, which means storing and factorizing the matrix $\overline{k_{33}}$. This matrix is often considered to be full, which implies a significant increase of memory usage and solution time when the number of subdomains grows. Fortunately, it will be shown in Section 3 that this matrix is sparse, and that its sparsity increases with the number of subdomains.

In the first part of the article, an efficient algorithm will be presented for building the interface problem matrix. Then the solving of the interface problem will be studied, and comparisons with standard skyline or sparse solvers will be given.

2. CONDENSATION METHOD

To create the interface problem matrix, Eq. (4), it is necessary to condense the stiffness matrix of each subdomain, that is to calculate $k_{bi}k_{ii}^{-1}k_{ib}$ (where indices i refer to internal d.o.f. and indices b refer to subdomain boundary d.o.f.). The explicit calculation of $k_{ii}^{-1}k_{ib}$ requires $2 \times nbdof$ triangular system resolutions, where $nbdof$ is the number of subdomain boundary d.o.f. This step can be interpreted in the following way: each internal d.o.f. brings its contribution to the stiffness of each boundary d.o.f., so that the behavior of the condensed boundary is equivalent to the behavior of the entire subdomain. This step is very time consuming because the calculations are not local, i.e. restricted not only to the neighbour d.o.f. It would be much more efficient if this process could be executed step by step, so that only the internal d.o.f. connected to the boundary d.o.f. updated the boundary stiffness matrix. This requires that the internal d.o.f. appear at the bottom of the internal stiffness matrix k_{ii}, so that they are modified by the elimination of all other internal d.o.f. This constraint is unfortunately incompatible with the bandwidth reduction of the matrix k_{ii}.

To overcome this difficulty, a frontal method [11] can be used. The frontal method has the advantage of allowing very flexible strategies concerning the sequence of elimination of equations. When applying the frontal method to subdomains condensation, it is necessary to assemble the boundary d.o.f. in the frontal matrix, and to retain them until all internal d.o.f. have been eliminated. At the end of the frontal elimination process, the frontal matrix is exactly the condensed matrix $k_{bi}k_{ii}^{-1}k_{ib}$, Eq. (2).

In order to study the performance of this condensation method, two sets of tests were run. In the first set, the frontal method is compared to the classical LDL^t-skyline method for the non-decomposed problem. This test should assume that the choice of the frontal method is not penalizing. When implementing the frontal algorithm, some care must be taken. During the assembly–elimination process, equations which are to be eliminated are located somewhere in the middle of the frontal matrix. Usually, elimination is performed both for the lines above the pivot line and for the lines below it. We found it more efficient to move the pivot lines to the beginning of the frontal matrix before proceeding with the elimination. The number of operations performed is the same, but it is possible to avoid some tests and to use optimized dense linear algebra routines. It is obviously very difficult to state in the general case that one factorization algorithm is better than another one, because it is problem dependent. Therefore, we only give here results for 2D and 3D test cases (Fig. 1), for growing number of d.o.f., which show that both methods (frontal and skyline) are equivalent. Other tests, not reported here, lead to the same conclusion.

All numerical tests reported in this paper have been conducted using the finite element software SIC [15, 1, 3], and were run on a SGI Challenge L R4400 150 MHz sequential system.

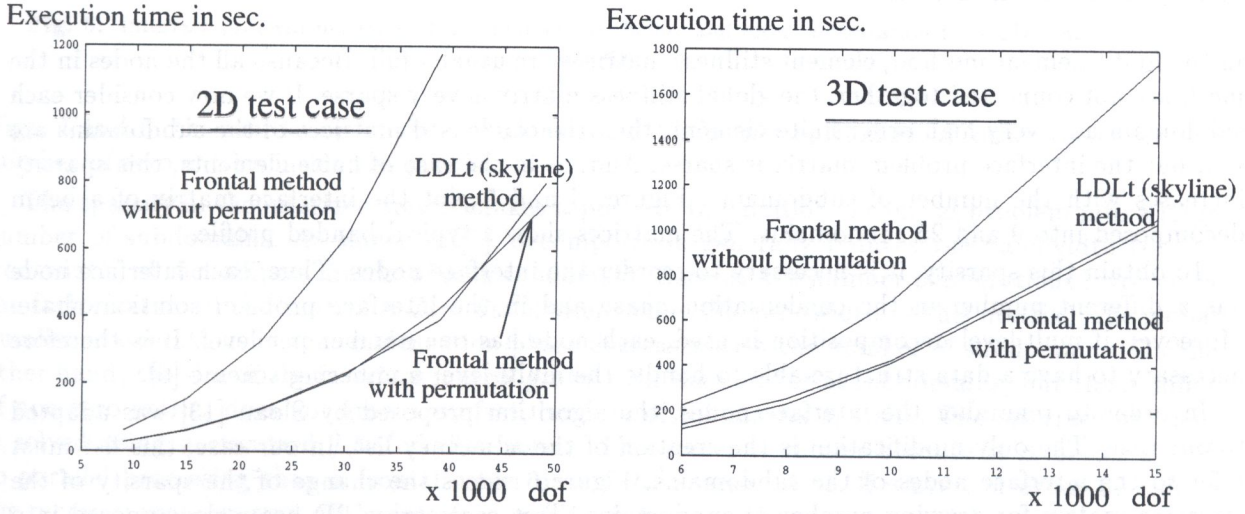

Fig. 1. Comparison of the frontal method and an LDL^t-skyline method for growing number of d.o.f.

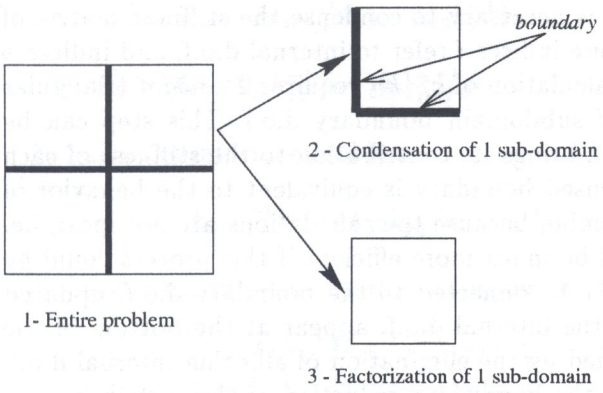

Fig. 2. Test case for the comparison of condensation versus factorization

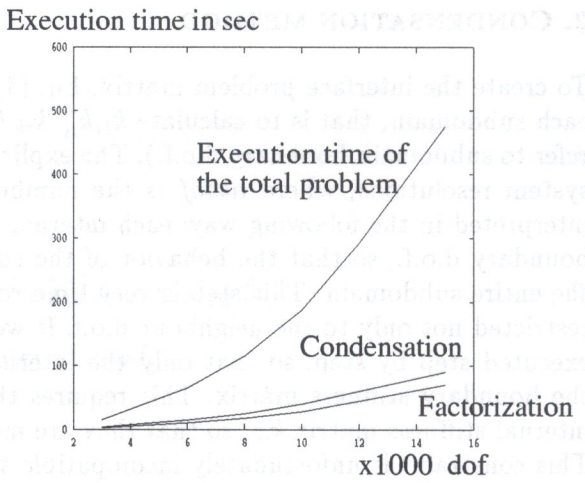

Fig. 3. Execution time comparison between condensation and factorization

The second set of tests aims at quantifying the overhead of condensation with respect to factorization, that is the overhead of calculating $k_{bi}k_{ii}^{-1}k_{ib}$ compared to only factorizing k_{ii}. As in the previous set of tests, execution time depends on the problem itself, and on the number of boundary d.o.f. of the subdomains which are condensed. In order to make the comparison possible, we consider a 2D beam (resp. 3D cube) to be decomposed into 4 (resp. 8) subdomains. Then, we compare the condensation time of one subdomain with the factorization time of a global problem made of one subdomain (see Fig. 2).

The results obtained (see Fig. 3) show that condensation time is not much greater than factorization time (about 25%). This result is achieved by renumbering the elements for the frontal method, so that boundary d.o.f. are grouped at the end of the assembly-elimination process.

In this section, we showed that the condensation algorithm which has been implemented is competitive with the standard factorization method on a sequential machine. But, as shown for example in [5, 7], the decomposition into subdomains usually reduces the bandwidth (or frontwidth) of the stiffness matrix of the subdomains. Therefore, one can expect an overall execution time improvement. However, this depends on the time spent in the solution of the interface problem.

3. Interface problem

In the finite element method, element stiffness matrices are usually full. Because all the nodes in the mesh are not connected together, the global stiffness matrix is very sparse. If we now consider each subdomain as a very high order finite element, then the condensed matrices of the subdomains are full, but the interface problem matrix is sparse. And, as in the case of finite elements, this sparsity increases with the number of subdomains. Figures 4 and 5 plot the interface matrix of a beam decomposed into 9 and 25 subdomains. The matrices show a typical banded profile.

To obtain this sparsity, it is necessary to reorder the interface nodes. Thus, each interface node has a different number in the condensation phase and in the interface problem solution phase. Moreover, if multilevel decomposition is used, each node has one number per level. It is therefore necessary to have a data structure able to handle the multi-level numbering scheme [6].

In order to renumber the interface nodes, the algorithm proposed by Sloan [13] was adapted to our case. The only modification is the creation of the adjacency list. In our case, this list must refer to the interface nodes of the subdomains. Figure 6 shows the change of the sparsity of the interface matrix for growing number of subdomains. Test cases are a 2D beam decomposed into regular square subdomains and 3D cube decomposed into sub-cubes. As it can be noticed from

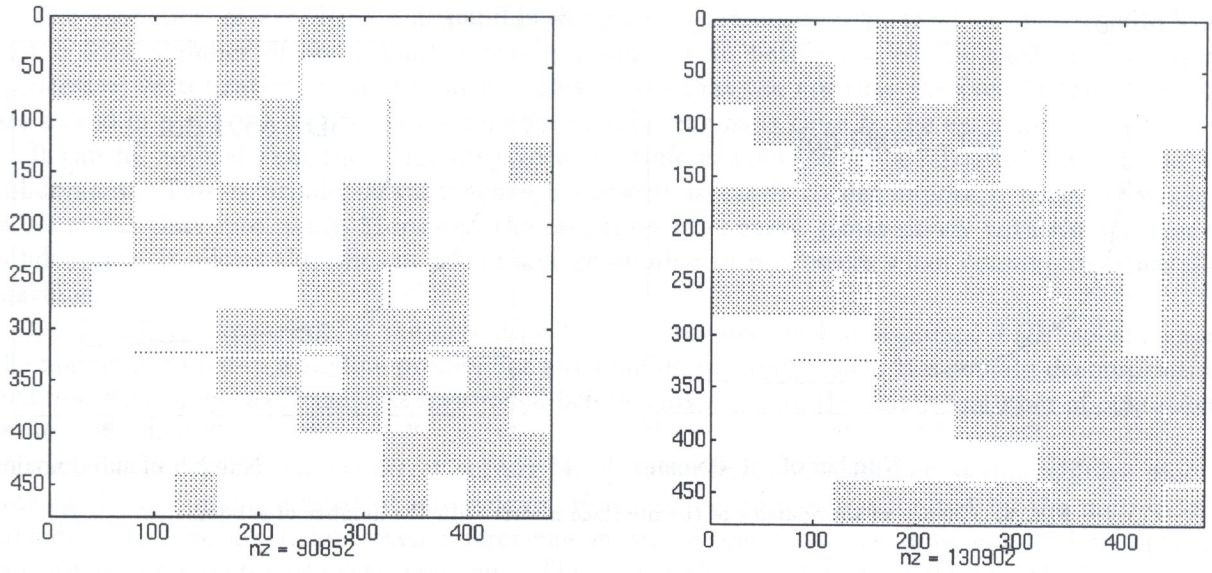

Fig. 4. Interface problem matrix with 9 subdomains, before and after factorization for a 2D problem with 5642 d.o.f.

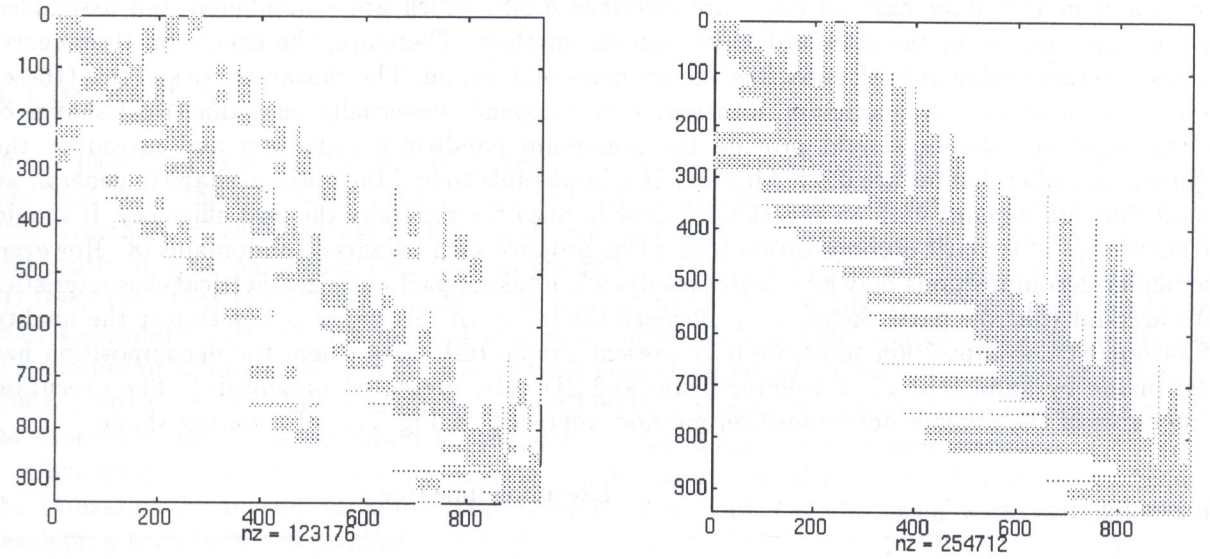

Fig. 5. Interface problem matrix with 25 subdomains, before and after factorization for a 2D problem with 5642 d.o.f.

Figs. 4 and 5, the reordering of the equations could be further optimized, in order to reduce the number of zero entries inside the profile.

The sparsity of the interface matrix makes it possible to partition the initial problem into a large number of subdomains. As shown in [7], decomposition generally reduces the bandwidth (or the frontwidth) of the stiffness matrix of each subdomain. Because the number of operations involved in the factorization of this matrix is $O(n \cdot bw^2)$(where bw is the bandwidth of this matrix), the global number of operations in the condensation phase decreases with the number of subdomains. On the other hand, the size of the interface matrix grows with the number of subdomains, but the number of non-zero entries does not grow very fast, because of the matrix sparsity. The interface problem is solved using a standard LDL^t-skyline solver. This solver is chosen because it is better suited to parallel processing than a frontal solver or a LDL^t-sparse solver. However, in order to reduce the memory requirements of the interface matrix, it is planned to use a parallel sparse solver. The results presented in this paper were obtained using a sequential LDL^t-skyline solver. We now study

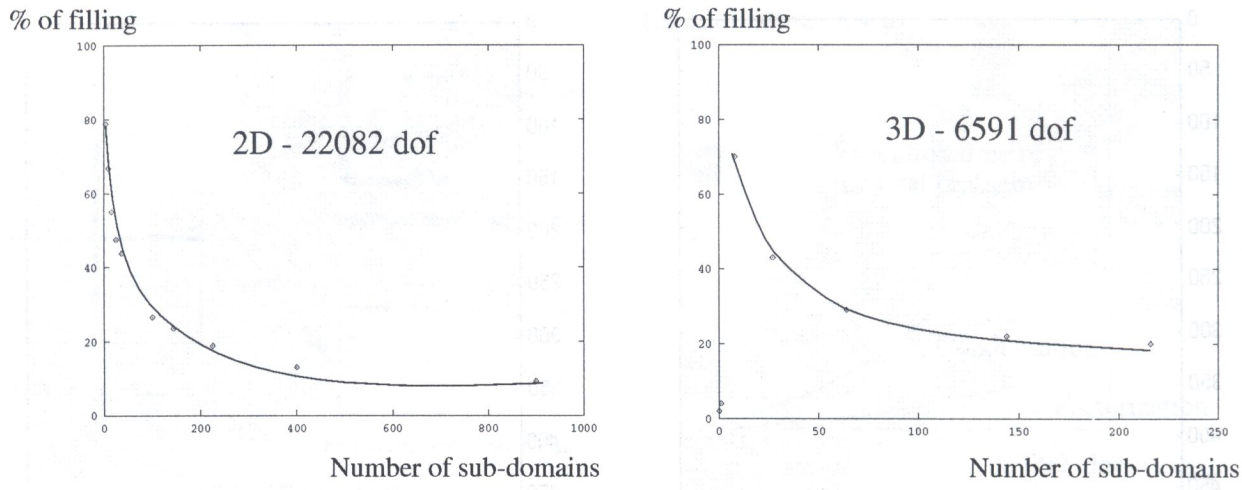

Fig. 6. Change of the sparsity of the interface matrix with the number of subdomains

the shape of the execution time curve when the number of subdomains grows. There are two limit cases for the number of subdomains: (a) only one subdomain, and (b) as many subdomains as finite elements. In these two cases, the execution time is the same: in one case, there are no interface d.o.f., and in the other case all d.o.f. are interface d.o.f., which are renumbered and assembled in the same way as in the standard finite element method. Therefore, the execution time curve, having the same value at both ends, has one (or more) extremum. The characteristics of this (these) extremum (minimum or maximum, location, value) depends essentially on geometrical shapes of subdomains: the shapes should provide the maximum bandwidth reduction and introduce the minimum number of interface d.o.f. Although it is impossible to find the optimal shape in general, we found that subdomains with an aspect ratio close to one offer generally the best efficiency. It should be noted that automatic decomposition tools often produce such balanced subdomains [8]. However, particular decompositions may give better results when taking particular geometrical characteristics into account (narrow parts, holes, ...). Because the intent of this paper is not to test the quality of automatic decomposition tools, we only present simple test cases where the decomposition has been made by hand (the 2D cantilever beam and 3D cube presented previously). The execution time curve of the domain decomposition method, reported on Fig. 7, is of a convex shape.

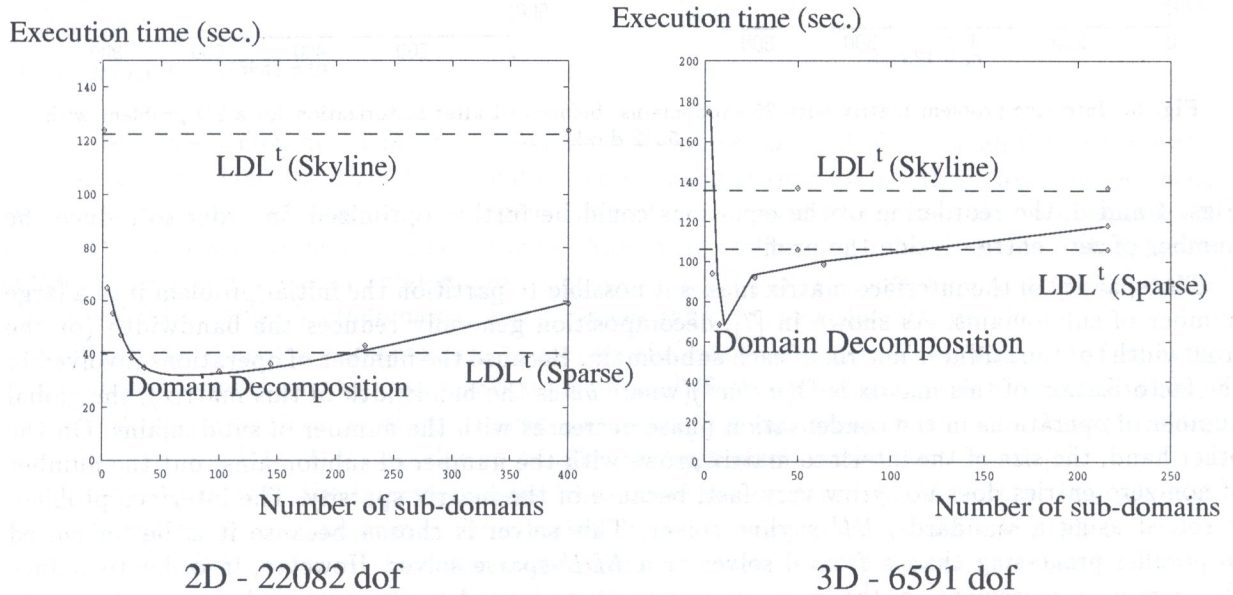

Fig. 7. Change of the execution time with the number of subdomains

The reduction of execution time, for the domain decomposition method, is due to reduction of the total number of operations performed, and not to better use of the hardware resources (processor, vectorization, memory, cache). This means that the matrices involved in the algorithm are stored in a sparser way when the number of subdomains is close to the extremum.

It can be noticed that the minimum execution time is achieved for a relatively low number of subdomains. This is usually advantageous because it is easier to decompose a mesh into a few subdomains than into many. Moreover, the execution time curve grows slowly with the number of subdomains. It allows the efficient use of a large number of processors when running on a parallel machine.

On Fig. 7, a comparison of the execution time is provided with a standard LDL^t solver where the matrix is stored in a skyline profile. For any number of subdomains, the domain decomposition method is much more efficient. As mentioned before, this is due to the smaller number of operations to be performed.

Since several years, sparse direct solvers [4, 9] have often outperformed skyline solvers due to storing the minimum number of zero entries. However, performance is limited by the indirections introduced in the algorithm when referencing matrix elements, which does not lead to optimal performance on vector or cache machines. This is not the case for the frontal method where the frontal matrices are dense. Therefore, despite the greater number of operations performed, execution time for the domain decomposition method can be smaller than for sparse solvers (see Fig. 7). Comparisons are made with the sparse solver implemented in the software SIC [2].

4. Conclusion

In the paper, an enhanced version of a direct domain decomposition method has been presented. The matrix of the interface problem is assembled as a skyline matrix, and factorized using the direct method. Skyline storage is here preferred to frontal storage because it leads to an easier and more efficient parallelization. Results showed, for two particular but representative 2D and 3D test cases, that this strategy leads to an efficient method. The method always performs better than a standard LDL^t-skyline method and often better than a LDL^t-sparse solver, depending on the number of subdomains. Results also showed that the method allows the partition of the initial domain into a large number of subdomains (until 900 in our examples) without increasing drastically the execution time. It is important for the efficient use of massively parallel machines.

Obviously, the results have to be validated on actual industrial problems. However, this implies the utilization of an efficient automatic decomposition tool. Collaboration with research teams developing such tools are planned.

It is also necessary to compare the direct method with iterative domain decomposition methods in order to establish the domain of validity of each method.

One of the major advantages of domain decomposition methods is the parallel implementation. To obtain good results, it is necessary to condense the subdomains in parallel and to factorize the interface problem matrix also in parallel. Such an implementation has been made. Results will be presented in a forthcoming paper.

The use of this direct decomposition method for problems with local nonlinearities is currently under investigation. Only subdomains with nonlinear elements need to be recondensed at each step of a Newton-Raphson algorithm, which saves a large amount of computation time.

References

[1] S. Aunay. *Architecture de logiciels de modélisation et traitements distribués.* PhD thesis, Université de Compiègne, 1990.

[2] C. Baranger. *Méthodes de résolution parallèles de grands systèmes appliquées au calcul de structures par éléments finis.* PhD thesis, Université de Compiègne, 1995.

[3] P. Breitkopf, G. Touzot. Architecture des logiciels et langages de modélisation. *Revue Européenne des Eléments Finis*, **1**(3): 333–368, 1992.

[4] I. Duff, A. Erisman, J. Reid. *Direct Methods for Sparse Matrices.* Oxford University Press, Oxford, England, 1987.

[5] I. S. Duff, J. A. Scott. The use of multiple fronts in gaussian elimination. RAL Report RAL-94-040, Rutherford Appleton Laboratory, 1994.

[6] Y. Escaig, M. Vayssade, G. Touzot. Automatisation de la décomposition de domaines dans un logiciel de calcul par éléments finis. In *Actes du Colloque National en calcul des Structures*, volume 2, pages 847–873. Hermes, 1993.

[7] Y. Escaig, M. Vayssade, G. Touzot. Une méthode de décomposition de domaines multifrontale multiniveaux. *Revue Européenne des Eléments Finis*, **3**(3): 311–337, 1994.

[8] C. Farhat, F.X. Roux. Implicit parallel processing in structural mechanics. Monograph, Computational Mechanics Advances, 1994.

[9] A. George, J. Liu. *Computer solution of large sparse positive definite systems.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.

[10] R. Glowinski, G. Golub, G. Meurant, J. Periaux, eds. *First international symposium on domain decompisition methods for partial differential equations*, Philadelphia, 1988. SIAM.

[11] B.M. Irons. A frontal solution program for finite element analysis. *Int. Jour. Num. Meth. Eng.*, **2**: 5–32, 1970.

[12] F-X. Roux. *Méthode de décomposition de domaine à l'aide de multiplicateurs de Lagrange et application à la résolution en parallèle des équations de l'élasticité linéaire.* PhD thesis, Université de Paris 6, 1989.

[13] S. Sloan. A fortran program for profile and wavefront reduction. *Int. J. Numer. Meth. Engrg.*, **28**: 2651–2679, 1989.

[14] P. Le Tallec, Y-H. De Roeck, M. Vidrascu. Domain decomposition methods for large linearly elliptic three dimensional problems. Rapport de recherche 1182, INRIA, 1990.

[15] G. Touzot, M. Vayssade. Un système de modélisation interactive. Rapport final Contrat MRT 86.E.0265.01, Université de Technologie de Compiègne, 1989.