# A scalable parallel assembly for irregular meshes based on a block distribution for a parallel block direct solver ⋆

David Goudin[1] and Jean Roman[1]

LaBRI, Université Bordeaux I - ENSERB, CNRS UMR 5800,
351, cours de la Libération, 33405 Talence Cedex, France
{goudin, roman}@labri.u-bordeaux.fr

**Abstract.** This paper describes a distribution of elements for irregular finite element meshes as well as the associated parallel assembly algorithm, in the context of parallel solving of the resulting sparse linear system using a direct block solver. These algorithms are integrated in the software processing chain EMILIO being developped at LaBRI for structural mechanics applications. Some illustrative numerical experiments on IBM SP2 validate this study.

## 1 Introduction

The OSSAU code of the CEA/CESTA is an industrial vectorized structural mechanics code, non linear in time and in two or three dimensions. It is a computational finite element code which solves plasticity (or thermo-plasticity, possibly coupled with large displacements) problems; it therefore computes the response of a structure to various physical constraints (external forces, pressure, ...).

The response of the structure is characterized by the node displacements of the finite element discretization. The total load applied to the structure is divided into a certain number of increments that are successively applied; each increment solves the non linear problem by a Newton-Raphson technique that leads to the iterative process $K(U_i) \cdot \Delta U_i = R(U_i)$ where $\Delta U_i$ represents the unknown incremental displacement vector, $K(U_i)$ the consistent stiffness matrix and $R(U_i)$ the right-hand-side vector. The positive definite symmetric matrix $K$ and the vector $R$ are computed by assembling elementary matrices $Ke(U_i)$ and elementary residual vectors $Re(U_i)$ associated with the finite elements $e$ of the mesh respectively:

$$K(U_i) = \sum_{e \in \Omega} Ke(U_i),\ R(U_i) = \sum_{e \in \Omega} Re(U_i) \ . \tag{1}$$

⋆ This work is supported by the French *Commissariat à l'Énergie Atomique* CEA/CESTA under contract No. 7V1555AC, and by the GDR ARP (iHPerf group) of the CNRS.

When dealing with complex 3D objects whose meshes imply a very great number of finite elements, several difficult problems arise [5]. Since the matrix of these systems does not have good properties (mainly due to the use of the Serendipity family of finite elements), classical iterative methods do not behave well. Therefore, as we want to obtain a robust versatile industrial software, we must use direct linear out-of-core solvers that take into account the sparsity of the system. When the number of Degrees Of Freedom becomes too large (specially above 150,000 DOFs), the out-of-core capabilities are not yet sufficient and parallel direct sparse solvers must be used, for reasons of memory capabilities and acceptable solving time.

As described before, the assembly of matrix $K$ consists in computing the elementary matrix of each element of the mesh and then adding this elementary matrix to the stiffness matrix $K$. An elementary matrix describes the physical characteristics of the element, and the complexity of its calculation essentially depends on the constraints applied to the structure. In certain test cases that we must deal with (3D Cologne Challenge for example), the assembly cost in time of the stiffness matrix $K$ is equivalent to the one of the matrix factorization. Therefore, the parallelization of the assembly step is a crucial problem.

The parallel assembly is a step in the software processing chain EMILIO (figure 1) developped at LaBRI and which gives efficient solutions to the problem of the parallel resolution of large sparse linear systems by direct methods [2, and included references]. This chain is organized into two major parts: a sequential pre-processing with a *global strategy* (ordering of the finite element graph [7], block symbolic factorization [1], repartitionning and distribution of the blocks of the factored matrix for the parallel block solver, distribution of the elements for the parallel assembly) and the parallel phase (parallel assembly and parallel resolution of the system). To avoid costly redistributions, we propose in section 1 an original distribution scheme of the elements so that the induced parallel assembly (described in section 2) will use efficiently the distribution of the matrix blocks for the EMILIO $LDL^T$ block solver [3, 4]. This distribution of the matrix blocks can be performed according to a 1D or a 2D scheme. Moreover, the informations regarding the geometrical description of the mesh are duplicated on the processors; indeed, this amount of data is negligeable with respect to the storage needed by $K$.

Section 3 will provide some numerical experiments on an IBM SP2 for a large class of sparse matrices coding regular and irregular meshes, including performance results and analysis. We will conclude with remarks on the benefits of this study and our future works.

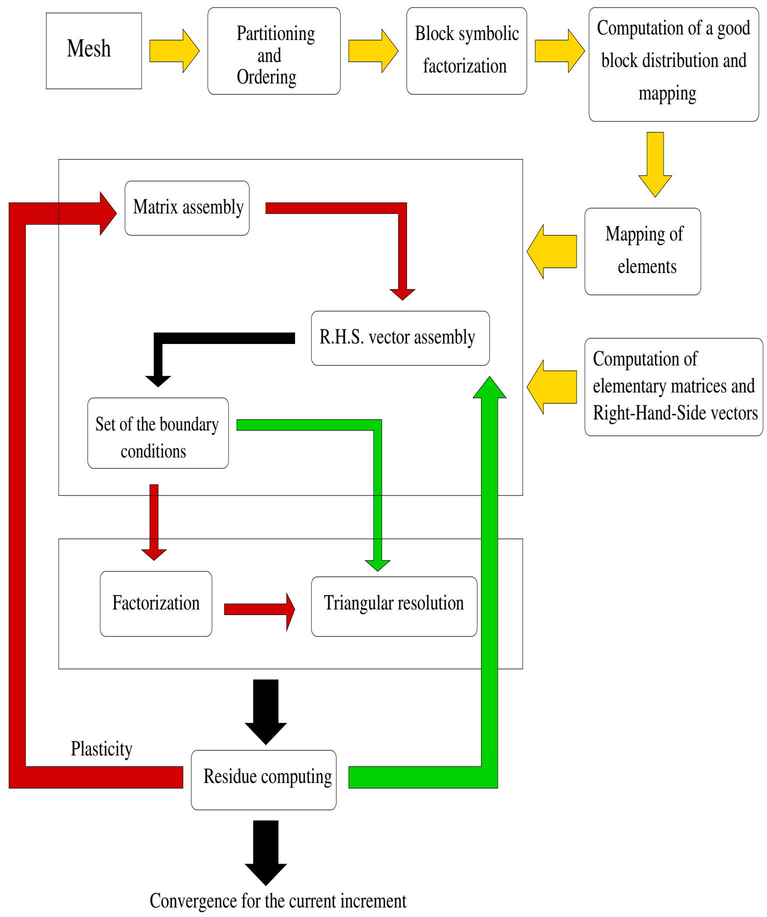


**Fig. 1.** Software processing chain EMILIO.

## 2 A Parallel Assembly induced by a Nested Dissection Strategy

### 2.1 Distribution of the mesh elements

The ordering step of our pre-processing computes a symmetric permutation of the matrix $K$ such that its factorization will exhibit as much concurrency as possible while incurring low fill-in. We use a tight coupling of the Nested Dissection and Approximate Minimum Degree algorithms. The *partition of the finite element graph* associated with $K$ into supernodes is achieved by merging the partition of separators computed by the Nested Dissection algorithm and the supernodes amalgamated for each subgraph ordered by Halo Approximate Minimum Degree (see [7, 6] for more details). This topological separation of the finite element graph induces a natural topological decomposition of the set of elements of the mesh into independent subdomains. The unknowns belonging to independent subgraphs that will be eliminated in parallel in the factorization process belong to elements located in independent subdomains of the mesh; the assembly of these elements can be performed in parallel too. Therefore, our distribution of elements will make use of both the mesh after the reordering, as well as the distribution of column blocks computed in the partitionning and mapping phase for the factored matrix. In this paper, we restrict our attention to a parallel assembly induced by a 1D column block distribution for the solver.

Our algorithm balances the total amount of computation in the sense that each processor must compute an equivalent number of elementary matrices. The communications corresponding to frontier elements are not taken into account for the balance, because they are supposed to be overlapped by the computations. The algorithm proceeds in two phases.

In a first phase, one builds for each element the list of the processors onto which the nodes (and so the associated columns) composing this element have been mapped by the block column distribution. If this list consists in only one processor $P$, the element is *totally local* to $P$, and *non totally local* otherwise. So, we can compute the minimum number $M$ of totally local elements for all of the processors, and we map $M$ totally local elements to each processor. Then, we store the remaining elements (the remaining totally local elements and the non totally local elements) in an array $T$ with the number of owner processors for each of them.

In the second phase, we first sort this array $T$ by decreasing order according to the number of owner processors; then, we determine on which processor each element must be mapped, favouring first the processor owning the highest number of nodes belonging to the element (locality criterion of external contributions), and second according to the number of elements already mapped. For the local elements of $T$ which were not mapped during the first phase, we try to optimize in addition the locality of the communications between the processor that computes the elementary matrix and the one that owns the column blocks to update.

### 2.2 Parallel assembly algorithm

The parallel assembly uses the distribution of the matrix column blocks and the distribution of the elements of the mesh. The latter not only defines the list of the elements whose elementary matrices will be computed on each processor but also some additional information for communication (the sizes of the messages to send, the number of the messages to receive...).

For each processor, we begin by processing the non totally local elements which have been mapped on it. Before computing the elementary matrix of a non local element, we see if one or several messages are not waiting for reception. In that case, we make a blocking receive (we agree to receive at most a number N of messages to schedule the computations and the reception activities of the processor), and we carry out, according to the contents of the messages, the addition of contributions in the blocks of the matrix; otherwise, we compute the elementary matrix of a non totally local element, we prepare the messages to send to the processors involved, and we process the addition of local contributions, if there are any.

When the processing of the "non totally local elements" is complete, we then process the totally local elements, that is to say we compute their elementary matrices, and then add the contributions in the local blocks. This step is fully parallel, without any communication. However, before calculating one such elementary matrix, we test if a message containing local update is still waiting, and if it is the case, it is processed first.

This strategy leads to a complete *overlap of the communications* induced by the treatment of non totally local elements by the computations due to the totally local elements. We use a similar strategy for the right-hand-side vector assembly.

## 3 Numerical Experiments and Analysis

The results we present in this section were obtained on an 192 node IBM SP2 at CINES (Centre Informatique National de l'Enseignement Supérieur, Montpellier, France), whose nodes are 120 MHz Power2SC thin nodes (480 MFlops peak performance) having 256 MBytes of physical memory each. Switch interrupts are enabled with default delay to perform non-blocking communication efficiently.

These experiments are performed on a collection of regular and irregular meshes : a 3D $39 \times 39 \times 39$ cube (1 DOF per node) and some irregular meshes from CEA/CESTA problems (3 DOFs per node).

Table 1 gives the characteristics of these test meshes, the dimension of the associated matrices $K$, and the time used by the algorithm to distribute the elements on a number of processors varying from 4 to 64. This sequential time on one processor grows linearly with the number of elements and very slowly (logarithmically) with the number of processors; therefore, our distribution algorithm is very little time consuming.

**Table 1.** Characteristics of the test meshes.

| Name | Elements | Nodes | Columns | Processors | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 4 | 8 | 16 | 32 | 64 |
| COL30 | 1260 | 6791 | 20373 | 0.014 | 0.022 | 0.025 | 0.032 | 0.046 |
| COL75 | 3150 | 16736 | 50208 | 0.037 | 0.040 | 0.052 | 0.074 | 0.112 |
| CUBE39 | 54872 | 59319 | 59319 | 0.281 | 0.332 | 0.491 | 0.914 | 1.553 |
| BILLES | 5184 | 5539 | 16617 | 0.035 | 0.044 | 0.066 | 0.103 | 0.152 |
| CEL3DM | 14400 | 10360 | 31080 | 0.107 | 0.142 | 0.216 | 0.249 | 0.401 |

**Table 2.** Assembly performance results (time in seconds) on the IBM SP2.

| Name | Processors | | | | | Elt time |
|---|---|---|---|---|---|---|
| | 4 | 8 | 16 | 32 | 64 | |
| COL30 | 3.50 | 1.93 | 0.98 | 0.56 | 0.32 | 0.01s |
| COL75 | 8.79 | 4.64 | 2.38 | 1.36 | 0.80 | - |
| CUBE39 | 139.52 | 69.71 | 35.15 | 17.89 | 9.02 | - |
| BILLES | 13.40 | 6.72 | 3.45 | 1.86 | 1.09 | - |
| CEL3DM | 37.32 | 18.72 | 9.55 | 4.92 | 2.72 | - |
| COL30 | 16.56 | 8.55 | 4.58 | 2.31 | 1.24 | 0.05 |
| COL75 | 41.13 | 20.72 | 10.67 | 5.88 | 3.01 | - |
| CUBE39 | 690.42 | 344.44 | 172.55 | 86.60 | 43.47 | - |
| BILLES | 65.30 | 32.72 | 16.45 | 8.40 | 4.46 | - |
| CEL3DM | 181.89 | 91.29 | 45.64 | 22.93 | 11.49 | - |
| COL30 | 34.42 | 16.77 | 8.43 | 4.28 | 2.23 | 0.1s |
| COL75 | 80.90 | 40.60 | 20.67 | 10.68 | 5.48 | - |
| CUBE39 | 1374.69 | 687.19 | 344.06 | 172.10 | 87.41 | - |
| BILLES | 130.34 | 65.22 | 32.91 | 16.56 | 8.32 | - |
| CEL3DM | 362.83 | 181.81 | 90.91 | 45.80 | 22.86 | - |

Table 2 presents the execution times of parallel assembly with 4 to 64 processors for three elementary matrix computation times : 0.01 s, 0.05 s and 0.1 s. These times come from profiled real applications (3D Cologne Challenge for example). On both regular (CUBE39) and irregular problems (test cases from CEA), the results achieved by our parallel algorithm are very scalable; moreover, varying elementary matrix computation time has no impact on this scalability. This confirms the quality of our element distribution and the fact that our parallel assembly implementation fully overlaps communications due to non totally local elements by the computation of the elementary matrices of totally local elements. Indeed, we can see on figure 2 that communications (leftmost part of the charts) are all better overlapped by local computations (rightmost part of the charts) and even more that the elementary matrix computation time is important.

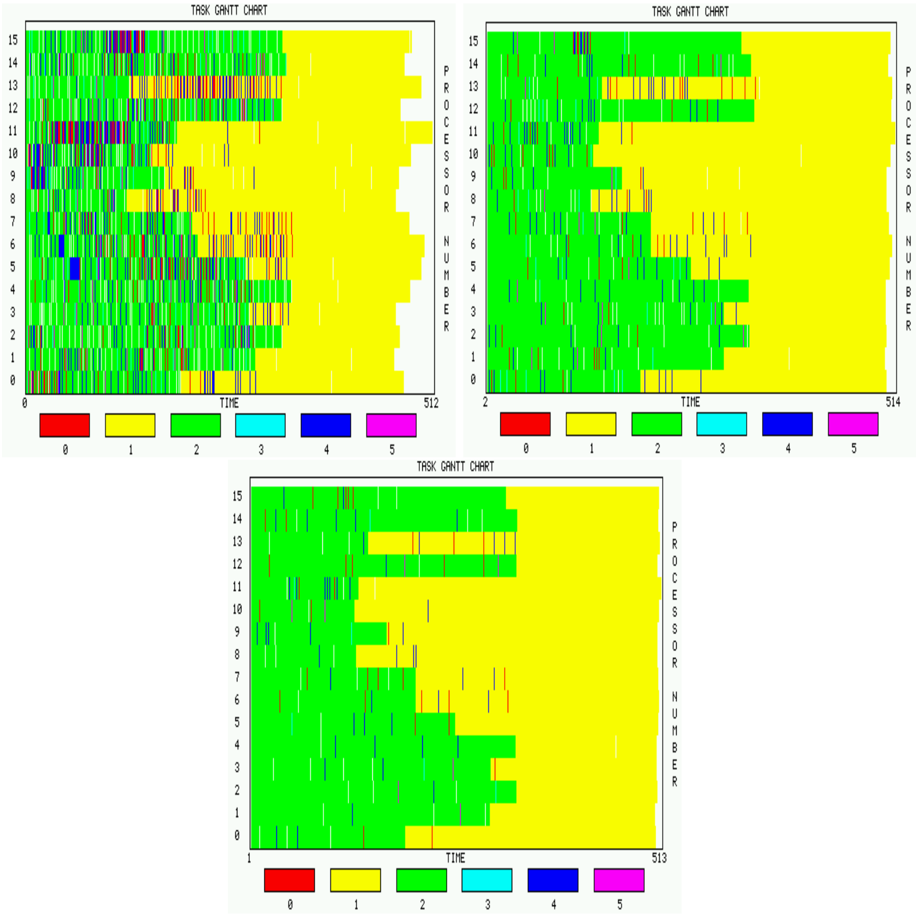


**Fig. 2.** Gantt charts for COLOGB75; times = 0.01 s, 0.05 s and 0.1 s.

## 4 Conclusion and Prospects

This paper presents promising results on the parallelization of the assembly of a matrix in a structural computational code. We can now assemble matrices with a great number of DOFs in a acceptable CPU time. This work regarding the parallel assembly is still in progress; we are currently developping a more general strategy corresponding to a 2D block distribution of the matrix and studying the problem of non-linear plasticity which requires us to redistribute some elements during the increments according to their computation cost.

## References

1. P. Charrier and J. Roman. Algorithmique et calculs de complexité pour un solveur de type dissections emboîtées. *Numerische Mathematik*, 55:463–476, 1989.
2. D. Goudin, P. Hénon, F. Pellegrini, P. Ramet, J. Roman, and J.-J. Pesqué. Parallel sparse linear algebra and application to structural mechanics. *accepted in Numerical Algorithms, Baltzer Science Publisher*, 2000, to appear.
3. P. Hénon, P. Ramet, and J. Roman. A Mapping and Scheduling Algorithm for Parallel Sparse Fan-In Numerical Factorization. In *Proceedings of EuroPAR'99*, number 1685 in Lecture Notes in Computer Science, pages 1059–1067. Springer Verlag, 1999.
4. P. Henon, P. Ramet, and J. Roman. PaStiX: A Parallel Sparse Direct Solver Based on a Static Scheduling for Mixed 1D/2D Block Distributions. In *Proccedings of Irregular'2000*, number 1800 in Lecture Notes in Computer Science, pages 519–525. Springer Verlag, May 2000.
5. P. Laborde, B. Toson, and J.-J. Pesqué. On the consistent tangent operator algorithm for thermo-pastic problems. *Comp. Methods Appl. Mech. Eng.*, 146:215–230, 1997.
6. F. Pellegrini and J. Roman. Sparse matrix ordering with SCOTCH. In *Proceedings of HPCN'97*, number 1225 in Lecture Notes in Computer Science, pages 370–378. Springer Verlag, April 1997.
7. F. Pellegrini, J. Roman, and P. Amestoy. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. In *Proceedings of Irregular'99*, number 1586 in Lecture Notes in Computer Science, pages 986–995. Springer Verlag, April 1999. *Extended paper appeared in Concurrency: Practice and Experience, 12:69-84, 2000.*