Parallelizing a hybrid finite element-boundary integral method for the analysis of scattering and radiation of electromagnetic waves ✩

R. Durán Díaz [a,*], R. Rico [a], L.E. García-Castillo [b], I. Gómez-Revuelto [c], J.A. Acebrón [d], I. Martínez-Fernández [b]

[a] Departamento de Automática, Universidad de Alcalá, Alcalá de Henares, Spain
[b] Departamento de Teoría de la Señal y Comunicaciones, Universidad Carlos III, Madrid, Spain
[c] Departamento de Ingeniería Audiovisual y Comunicaciones, Universidad Politécnica de Madrid, Madrid, Spain
[d] Center for Mathematics and its Applications (CEMAT), Instituto Superior Técnico, Lisbon, Portugal

ARTICLE INFO

ABSTRACT

This paper presents the practical experience of parallelizing a simulator of general scattering and radiation electromagnetic problems. The simulator stems from an existing sequential simulator in the frequency domain, which is based on a finite element analysis. After the analysis of a test case, two steps were carried out: first, a "hand-crafted" code parallelization of a convolution-type operation was developed within the kernel of the simulator. Second, the sequential HSL library, used in the existing simulator, was replaced by the parallel MUMPS (MUltifrontal Massively Parallel sparse direct Solver) library in order to solve the associated linear algebra problem in parallel. Such a library allows for the distribution of the factorized matrix and some of the computational load among the available processors. A test problem and three realistic (in terms of the number of unknowns) cases have been run using the parallelized version of the code, and the results are presented and discussed focusing on the memory usage and achieved speed-up.

1. Introduction

This paper presents the practical experience and the results obtained in the parallelization of the code corresponding to a novel hybrid finite element-boundary integral method that permits an efficient analysis and solution of general problems of radiation and scattering of electromagnetic waves. This method will be referred to as *finite element—iterative integral equation evaluation* (FE-IIEE) due to the nature of the algorithm, as it will become apparent later on.

The analysis of the radiation and scattering of electromagnetic waves is an important issue that finds applications in many electromagnetic engineering areas. Modern radiating structures present complex configurations with the presence of several conductive and permeable (possibly anisotropic) materials; also,

they may use exotic materials (bandgap, chirality, handedness and so on) that are designed to have special radiation or scattering properties. In many cases, the permeable and/or exotic materials are located in the proximity of the antenna, as in general antenna platforms or surfaces of aircraft, ships, or ground vehicles where the antennas are mounted. Specifically, the antennas may be conformal to those surfaces which, in general, cannot be accurately approximated by canonical curvature surfaces. The same considerations hold for the analysis of the scattering of electromagnetic waves by the type of structures mentioned above. Thus, a rigorous, reliable and powerful method of analysis is required.

In this context, finite element method (FEM [1]) is very flexible and capable of handling within the same code complex geometries, non-canonical surfaces, exotic permeable materials, anisotropy, and so on. However, FEM formulation does not incorporate the radiation condition. Thus, when dealing with open region problems, as in the case of scattering and radiation of electromagnetic waves, an artificial boundary must be used to truncate the problem domain (mesh) in order to keep the number of unknowns finite.

The hybridization of FEM with the use of the boundary integral (BI) representation of the exterior field endows the FEM analysis with a numerically exact radiation boundary condition at the mesh

* Corresponding author.
E-mail addresses: raul.duran@uah.es (R. Durán Díaz), rafael.rico@uah.es (R. Rico), luise@tsc.uc3m.es (L.E. García-Castillo), igomez@diac.upm.es (I. Gómez-Revuelto), juan.acebron@ist.utl.pt (J.A. Acebrón), ignafer@tsc.uc3m.es (I. Martínez-Fernández).

truncation boundary. This has led to several hybrid schemes FEM-BI (see, for example, [2–4], or, more recently, [5]). However, in contrast with standard FEM-BI approaches, FE-IIEE preserves the original sparse and banded structure of the FEM matrices, allowing the use of efficient FEM solvers. These features are achieved through the use of a non-standard multiplicative iterative Schwarz type of domain decomposition approach, as explained below. The main difference of FE-IIEE with respect to other approaches based on the domain decomposition paradigm [6] is that FE-IIEE only requires the evaluation of the boundary integral terms but no solution of the integro-differential system is performed. Additional advantages of the FE-IIEE decoupling approach are the reuse of codes for non-open region problems, easy hybridization with asymptotic (high frequency) techniques, [7–10], easier parallelization, and integration with adaptive FEM approaches.

In this work, we did not intend to develop a brand-new parallel FEM-BI simulator but rather to show the experience and results obtained along the process of parallelizing an already existing sequential FEM-BI simulator. To achieve this goal we identified bottlenecks from the point of view of both memory usage and computational load, targeting a modified code which is scalable in the range of a modest number of processors. This result overcomes the limitations of the original simulator, especially in terms of memory availability, thus allowing the analysis of larger problems.

2. FE-IIEE algorithm

Consider the following domain decomposition setup of the open region problem (see Fig. 1). The original infinite domain is divided into two overlapping domains: a FEM domain (Ω^{FEM}) bounded by the surface S and the infinite domain exterior to the auxiliary boundary S' (Ω^{EXT}). Thus, the overlapping region is limited by S' and S. For simplicity, the region exterior to S is assumed to be a homogeneous medium. Several FEM domains may exist. Also, infinite (electric or magnetic) ground planes may be taken into account analytically using the appropriate Green's function. The boundary S may be arbitrarily shaped but typically it is selected conformal to S'. The distance from S' to S is usually small, typically in the range of 0.05λ to 0.2λ, where λ is the free-space wavelength at the considered frequency. In this way, the FEM domain can be truncated very close to the sources of the problem thus reducing the number of unknowns of the problem.

By means of FEM, a sparse system of equations that models (using the double-curl vector wave equation) the electromagnetic field solution in the FEM domain is obtained:

$$\begin{bmatrix} K_{II} & K_{IS} \\ K_{SI} & K_{SS} \end{bmatrix} \begin{Bmatrix} \{g_I\} \\ \{g_S\} \end{Bmatrix} = \begin{Bmatrix} \{b_I\} \\ \{b_\Psi\} \end{Bmatrix} \tag{1}$$

Subscripts S and I refer to the degrees of freedom g on S and in the inner region, respectively. Vector b_I is null for scattering problems whereas it is a function of the inner sources for radiation problems. Vector b_Ψ (to be defined formally later in Eq. (3)) is a weighted integral of the value of the boundary condition on S. A local and absorbing type boundary condition is used on S.

In the present implementation, a Cauchy (Robin) boundary condition is used:

$$\hat{\mathbf{n}} \times \left(\bar{\bar{f}}_r^{-1} \nabla \times \mathbf{V} \right) + jk\hat{\mathbf{n}} \times \hat{\mathbf{n}} \times \mathbf{V} = \Psi \quad \text{at } \Gamma_S \tag{2}$$

where Γ_S denotes the surface S (making it clear that S is part of Ω^{FEM} boundary). The symbol $\hat{\mathbf{n}}$ stands for a unitary vector normal to Γ_S, j stands for the imaginary unit, $\mathbf{V}$ denotes either the vector electric or magnetic field, f_r is a material constant (see correspondences in Table 1), and Ψ is an iteratively computed function, subsequently defined. The symbol k is the wave number, i.e., ω/c where ω is the angular frequency and c is the speed of light. Last, Ψ, defined subsequently, is computed iteratively. The use of this type of boundary condition in this context was proposed in [11]. The absorbing character of the boundary condition on S yields a FEM solution free of interior resonances and improves the convergence of the method.

Vector b_Ψ can now be formally defined as

$$\{b_\Psi\}_i = \int_{\Gamma_S} \mathbf{N}_i \cdot \Psi \, d\Gamma \tag{3}$$

where $\mathbf{N}_i$ refers to the i-th finite element basis function of the mesh restricted to Γ_S. In this sense, the present implementation makes use of the isoparametric second-order curl-conforming basis functions for tetrahedra presented in [1,12].

Once the system of Eq. (1) has been obtained, the algorithm of FE-IIEE is as follows:

(1) An initial value of Ψ, denoted as $\Psi^{(0)}$, is assumed. Specifically, $\Psi^{(0)} = \mathbf{0}$ for radiation problems. $b_{\Psi^{(0)}}$ is computed using (3).
(2) FEM system (1) is solved. Fields on S' are calculated in order to compute the electric and magnetic current densities $\mathbf{J}_{\mathrm{eq}}$ and $\mathbf{M}_{\mathrm{eq}}$ of the equivalent exterior problem.
(3) The field, and its curl, over S radiated by $\mathbf{J}_{\mathrm{eq}}$ and $\mathbf{M}_{\mathrm{eq}}$ are calculated using the integral expressions:

$$\mathbf{V}^{\mathrm{IE}}(\mathbf{r} \in \Gamma_S) = \oiint_{S'} (\mathbf{L}_{\mathrm{eq}} \times \nabla G) \, dS' - jkh \oiint_{S'} \left(\mathbf{O}_{\mathrm{eq}} \left(G + \frac{1}{k^2} \nabla\nabla G \right) \right) dS' \tag{4}$$

$$\nabla \times \mathbf{V}^{\mathrm{IE}}(\mathbf{r} \in \Gamma_S) = jkh \oiint_{S'} (\mathbf{O}_{\mathrm{eq}} \times \nabla G) \, dS' - \oiint_{S'} (\mathbf{L}_{\mathrm{eq}} (k^2 G + \nabla\nabla G)) \, dS' \tag{5}$$

where h stands for the immittance of the homogeneous medium (see Table 1), and G denotes Green's function for a
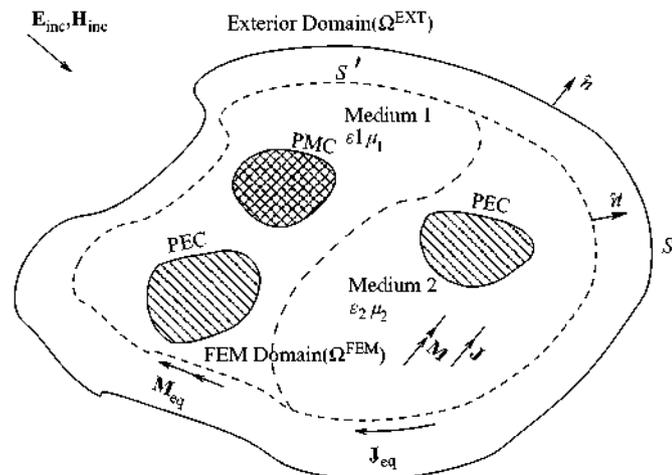

Fig. 1. Domain decomposition of the open region problem.

Table 1
Formulation magnitudes and material parameters.

	V	f_r	g_r	h	O	L
E formulation	E	μ_r	ε_r	η	J	M
H formulation	H	ε_r	μ_r	$\frac{1}{\eta}$	M	−J

ε_r and μ_r denote the electric permittivity and magnetic permeability, respectively, with respect to vacuum. Symbol η refers to medium impedance and **E**, **H** to electric and magnetic fields, respectively.

homogeneous medium,

$$G \equiv G(\mathbf{r}, \mathbf{r}') = \frac{e^{-jk(\mathbf{r}-\mathbf{r}')}}{4\pi|\mathbf{r}-\mathbf{r}'|}$$

which typically is the free space. It is worth noting that the methodology is also valid for non-homogeneous exterior regions (as those with infinite metallic planes, layered media, and so on) by using the corresponding integral expression representation of the field and Green's function of the exterior region.

Note the convolutional character of computational operation that (4), (5), represent. To compute the value of the field (and its curl) at each of the required points (integration points) on S (vector $\mathbf{r}$), all (integration) points on S' are used (vector $\mathbf{r}'$). This means that a double loop has to be implemented. The kernel depends on Green's function, which depends on $\mathbf{r}-\mathbf{r}'$. Thus, assuming a similar number of (integration) points on S and S', denoted as $N_S \approx N_{S'}$, the computational cost is $O(N_S^2)$.

(4) A new value of Ψ ($\Psi^{(i+1)}$ in general, i representing the iteration number) is computed by introducing the values of the fields $\mathbf{V}(\mathbf{r} \in \Gamma_S)$ and $\nabla \times \mathbf{V}(\mathbf{r} \in \Gamma_S)$ in (2).

(5) The error between $\Psi^{(i+1)}$ and $\Psi^{(i)}$ is calculated. More precisely, the error is measured using b_Ψ. If the relative error $\|b_{\Psi^{(i+1)}} - b_{\Psi^{(i)}}\| / \|b_{\Psi^{(i+1)}}\| < \delta$, δ being the error threshold, the algorithm stops. Otherwise, it continues from step 2 on, using the new value $\Psi^{(i+1)}$ at the right-hand side.

Fig. 2 shows the flow chart of FE-IIEE. It can be observed that the algorithm consists of the conventional blocks in a FE code, and two extra blocks. The first one calculates the initial boundary condition on S, $\Psi^{(0)}$, whereas the second upgrades it for the next iteration, $\Psi^{(i+1)}$, using the equivalent sources on S' and Green's function.

This method converges in a few iterations, typically, fewer than 10 for $\delta \approx 10^{-4}$. The good convergence of the method is based on the absorbing character of the boundary condition on S, Eq. (2), and in the overlapping of the FEM (interior) domain and the exterior domain (further insight can be found in [13,14]). In the present implementation, a direct solver is used to solve the FEM problem. The factorization of the FEM matrix is performed only once at the first iteration. The FEM solutions for the second and subsequent iterations are obtained by simple forward and backward substitutions. Thus, the numerical cost of the second and subsequent iterations (relative to the factorization phase) is very small. It is remarkable that even if an iterative solver would

have been used, the solution of the previous iteration cycle could have been used as an initial guess for the next iteration of the FEM solver.

3. Computational analysis of the simulation method

As explained in the Introduction, this paper shows the process of parallelizing an existing sequential simulator. For this reason, the first task we faced was to analyze the consumption of computational resources by the different existing building blocks inside the sequential code, in order to prioritize their parallelization. By computational resources we refer to two clearly different aspects:

- time in computational cycles; and
- memory consumption.

The first aspect refers to the total time needed for the application to compute the results, whereas the second aspect has an impact over the size limit of the problems to be solved. In the following, we present the methodology used to tackle the parallelization process.

3.1. Methodology

The computational analysis of the simulator was conducted over the existing sequential code, running a test problem. This test problem was not expected to yield precise quantitative results, but it should permit to identify the bottlenecks in terms of computational resources, as explained above. Moreover, the test problem was selected keeping in mind that the available computational resources were (at that time) rather limited. The test problem consisted in the scattering problem of a plane wave incident on a dielectric cube with losses ($\varepsilon_r = 2-2j$ and $\mu_r = 1$). The imaginary part of ε_r is used to model lossy electrical material, which is a common approach in electrical engineering; see, for instance, [15]. The size of the cube was 0.9λ; its surface is used as fictitious border S'; conformal to S' and separated 0.1λ apart is S, the FEM domain boundary. The number of mesh elements is 2684, and the number of unknowns is 18 288.

The available system was a cluster of eight nodes (blades in terminology of the manufacturer, Sun Microsystems) model *SunFire B1600*. The technical details of each node can be found in Table 2.
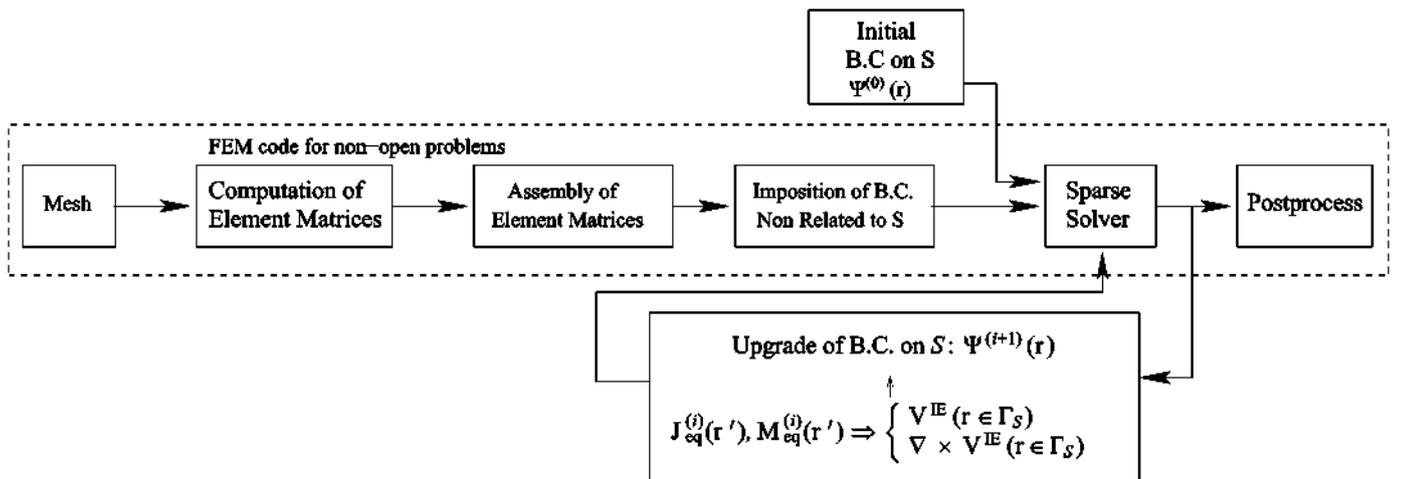
Fig. 2. Flow chart of FE-IIEE.

Table 2
Configuration of every cluster node.

Processor	Mobile AMD Athlon(tm) XP-M 1800+
Clock	1.5 GHz
Cache	256 kb
Main memory	1 Gb
Network interface	1 Gbps
Hard disk	30 Gb

3.2. Monitorization

The sequential source code has been conveniently probed by inserting time and memory-size controlling functions at critical points.

The probed code was sequentially executed for the test problem using only one cluster node of the *SunFire B1600*. As a result, it became clear that the program exhibits two separate computational phases:

- an initial phase, where the matrix is factorized;
- an iterative phase, where the code performs a number of iterations inside the kernel loop before achieving the required accuracy. For the particular test problem considered, obtaining a threshold relative error δ between subsequent iterations below 10^{-4} required a total of six iterations. This kernel loop is clearly observed in the flow chart of Fig. 2.

The results obtained with the previous execution are qualitatively depicted in Fig. 3. The bar on the left-hand side shows the relative amount of computation needed for each phase, corresponding to a part of the flow diagram on the right-hand side. This figure makes it clear that, as explained above, the main time consumers are the factorization (uppermost part of the left bar) and the iterative phase (middle part of the left bar, showing the aggregate time for all necessary iterations). Remark that these results are valid only for the test case under consideration, and cannot be extrapolated to larger problems. In particular, it would be wrong to deduce that the proportion of execution time spent in the factorization versus the iterative part keeps constant for problems of any size. The aim was just to obtain information about the computational behavior of the application.

In Fig. 2, the block named "Sparse Solver" includes the initial phase, where the factorization is carried out, and the iterative phase, where a $A \cdot X = B$ resolution is performed. Both are handled using a sparse solver, since A is a sparse matrix. The original sequential implementation used the HSL linear algebra library package (formerly known as *Harwell Subroutine Library*, see [16]) as building block for simulation.

As it is well known, HSL is a collection of Fortran packages for large scale scientific computation written and developed by the Numerical Analysis Group at the Rutherford Appleton Laboratory and by other experts and collaborators. Many HSL routines make use of BLAS [17] (Basic Linear Algebra Subprograms). The sequential code uses a Fortran solver: either ME62 or ME42 routines, depending on whether the matrix A is symmetric/hermitian or not.

In the present sequential implementation, the algorithm used for element ordering (crucial for the performance of the frontal solver) with HSL is a variant of Sloan's algorithm [18] that incorporates also spectral reordering (details are given in [19]). The algorithm is implemented in MC63 routine of HSL. Indirect element ordering with default MC63 parameters is used prior to calling the frontal solver factorization routines.
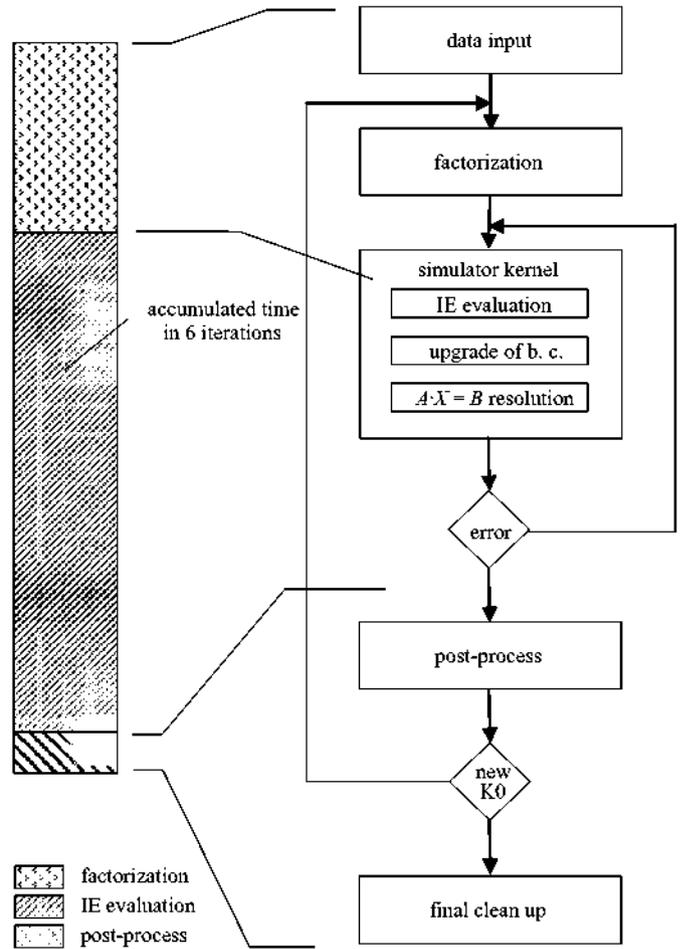

Fig. 3. Program flow and monitorization points.

In general, memory consumption for a multifrontal solver (for a full explanation on multifrontal solvers, see for example [20]), such as the ones used in this work, is linearly related to the maximum wavefront (see, for instance, [19]), which in turn is strongly dependent on the order in which the elements are assembled. For this reason, ordering algorithms have an enormous impact on the final memory consumption, thus making it very difficult to supply general rules regarding precise memory requirements.

To summarize, parallelizing the code is absolutely mandatory in view of the limitations of the sequential code. Such an effort will allow to exploit larger computational resources in terms of memory as well as speeding-up the simulations. Therefore, larger problems could be run in a reasonable time.

3.3. Analysis of the monitorization

Regarding time consumption (depicted qualitatively in Fig. 3), the monitorization showed that the largest time consumption takes place in:

- The factorization of the matrix (performed only once in the first iteration).
- The block named as "IE Evaluation" (inside the "simulator kernel", as shown in Fig. 3). In this block, the computation of the field and its curl on S, $\mathbf{V}^{IE}$, $\nabla \times \mathbf{V}^{IE}$, is performed. This block corresponds to the evaluation of the double surface integrals of

expressions (4), (5) (step 3 of the FE-IIEE algorithm shown above in Section 2).

The main conclusions of the analysis are the following. As stated above in Section 2, once the FEM matrix is factorized, the solution of the linear system $A \cdot X = B$ is obtained in a virtually negligible time within each iteration of the algorithm. This shows in practice that the computational load of this method, once the FEM matrix is factorized, is essentially in the convolution type operation corresponding to the evaluation of the integro-differential expressions (4), (5). With respect to the memory consumption, we observed that it remains constant during the full execution time. The initial memory size reserve stays unchanged until it is freed shortly before the conclusion.

3.4. Planned improvements

Based on the results of the analysis for the sequential execution, it was decided to attempt two sets of actions.

(1) Once identified the main time consumers, a first hand-crafted parallelization was accomplished, involving IE evaluation, since it is the main consumer of computer cycles.
(2) Concerning the initial factorization of the matrix and the block "$A \cdot X = B$ resolution", the goal was to replace the solver in the HSL library by a parallel sparse solver. We expected that the parallel solver could improve both the execution time (since we have several processes working in parallel) and the size of the solvable problems (since the parallel solver could make advantageous use of the memory present on the different processors).

The first item is described next in Section 4, while the second one is deeply analyzed in Section 5.

4. Hand-crafted code parallelization

The parallelization was carried out using the message passing interface (MPI, [21]), with the support of the MPICH2 library (specifically, version 1.0.8).

The hand-crafted code parallelization targeted essentially the heavy computation load given by the convolution-type operations involved in the IE evaluation of the exterior problem, Eqs. (4), (5). Observe that a double loop is required since the computation of the field (and the curl) at each point in S involves the integration over all points in S'.

Thus, the parallelization consisted in distributing the execution of the outer loop over the available processes and performing a final reduction at the end of the loops, so that all processes have the complete field and curl vectors. Since this process involves a very small amount of communication, we expected a quasi-linear speed-up on the number of used processes.

5. Replacement of the sequential solver

In the following, we focus on replacing the sequential sparse solver by an efficient parallel solver. For this purpose, we have selected MUMPS (see [22]), a well-known package for solving systems of linear equations.

The goal was the replacement of the HSL library, used by the sequential version of the simulator, by the MUMPS library inside both the factorization and "$A \cdot X = B$ resolution" blocks. We selected an appropriate MUMPS configuration for the test

problem: in this case, the matrix A is sparse and either symmetric or unsymmetric depending on the boundary conditions of the problems, which affects the setting of the corresponding configuration variables. We selected to set up MUMPS for the general unsymmetric case. We used complex double precision arithmetic and we actually let the master processor participate in the computations. We chose the most up-to-date version of MUMPS at the onset of the experiments, which turned out to be 4.8.3.

Basically, the idea was to replace the HSL subroutines by the corresponding (in a broad sense) ones from MUMPS. Obviously, there is no one-to-one correspondence among the subroutines in HSL and the subroutines in MUMPS, and the interfaces are also completely different. As explained in Section 3.2 the frontal solver used originally was HSL ME62. So the goal was to replace it by the corresponding frontal solver in MUMPS. Though the programming interfaces to MUMPS and HSL are very different, several data structures are the same, such as, for example, the structure that represents which variables belong to which element. So by cross-examination and comparing, we found a correspondence between the routines in HSL and the ones performing similar tasks in MUMPS. Eventually, the replacement, though not completely straightforward, was carried out successfully.

MUMPS requires three steps to solve the linear system and a (re-)ordering is mandatory during the analysis phase, also very important for the performance of the solver, as in the HSL case. However, MUMPS leaves to the user the choice of basically two external ordering packages. We considered two ordering packages: PORD (described in [23]) and METIS (described in [24]).

In general, the solution of a sparse system of linear equations $A \cdot X = B$ on a parallel computer gives rise to a graph partitioning problem. In particular, when parallel direct methods are used to solve a sparse system of equations (such as we do), a graph partitioning algorithm can be used to compute a fill-reducing ordering that leads to higher degrees of concurrency in the factorization phase (see [25,26]). In fact, METIS belongs to a class of algorithms based on multilevel graph partitioning with a moderate computational complexity, providing excellent graph partitions.

6. Trade-offs among the different parallelization options

In this section, we present the results obtained, once the hand-crafted parallelization, and the replacement of HSL sequential solver by MUMPS library (and corresponding interface) have been performed, using the test case as input. A number of executions of the test problem were performed, using from 1 to 8 cluster processes, with MUMPS with PORD (labeled MUMPS & PORD in the figures), and MUMPS with METIS (labeled MUMPS & METIS in the figures).

The main goal of the test problem was to validate the results obtained from the parallel version with those of the original sequential version. Also, it served as a debugging platform for preliminary tests of the scalability of the parallel version.

However, to really test the capabilities of the newly parallelized application, along with the capabilities of MUMPS, we set up three realistic examples endowed with an increasingly higher number of unknowns. The results obtained from these latter examples will be presented later on.

6.1. Trade-offs of the ordering packages

The results depicted in Fig. 4 refer to the test case mentioned in Section 3.1, and deserve several comments. The times shown in the figure refers to a complete run of the problems.

First of all, regarding wall time, it should be noticed that MUMPS equipped with METIS clearly outperforms MUMPS with PORD, especially when the number of involved processes is low.
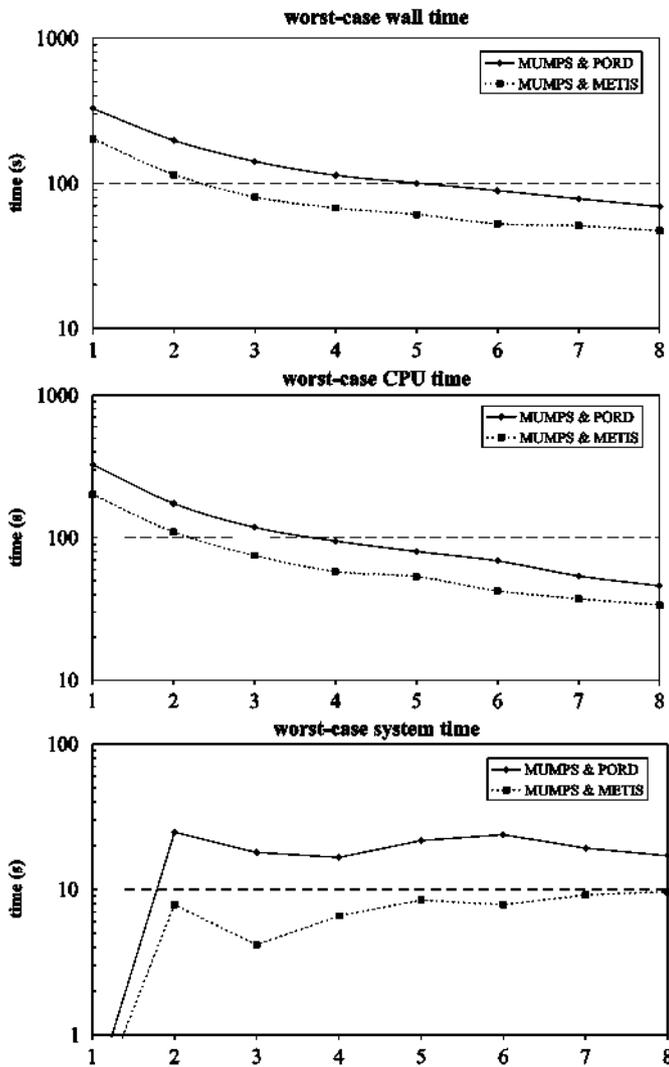
Fig. 4. MUMPS (with PORD and METIS element orderings) time comparisons as a function of the number of processes.
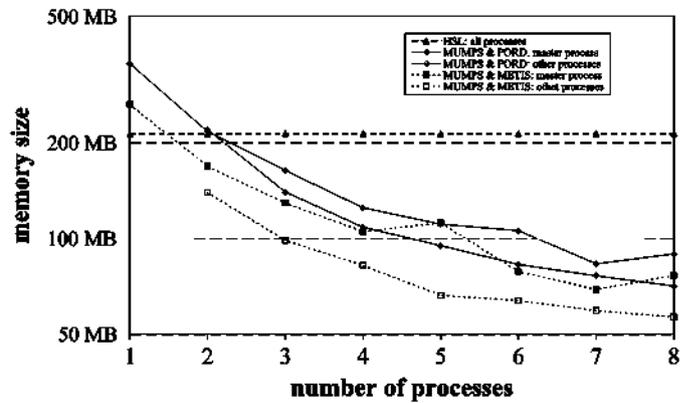

Fig. 5. HSL and MUMPS (with different element orderings) memory usage comparisons.
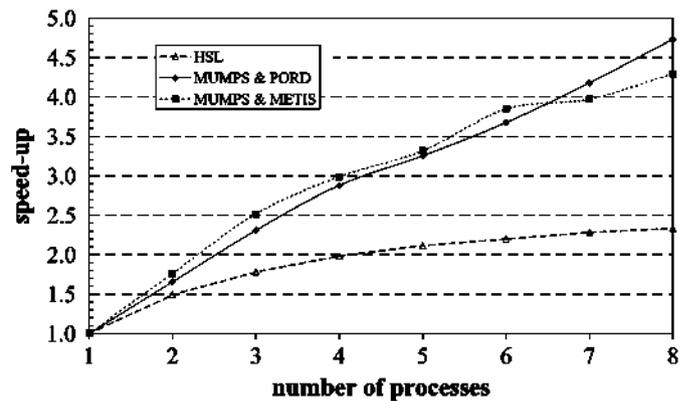
Fig. 6. HSL and MUMPS (with different element orderings) wall time speed-up.

Regarding CPU time, MUMPS using METIS outperforms MUMPS using PORD as well. However, when the number of process grows, the performances of MUMPS with both ordering schemes become closer.

The third picture shows somewhat surprising results. When using PORD, the system time spent is far from negligible and displays an irregular pattern as the number of involved processes grows. Some analysis on the system calls performed by the application revealed that these system calls are of the type *poll*, which seems to suggest communication issues among the processes. On its part, MUMPS with METIS shows that the system time spent is still remarkably high (though much lower than the corresponding when PORD is used) and stays roughly stable as the number of processes grows. MUMPS developing team is aware of this problem and it will be fixed in future versions. Remark that the scale in this picture is much smaller (actually, one-tenth) than in the previous ones, in order to make apparent the system time behavior.

6.2. Memory trade-offs

The results for memory consumption in the test case can be seen in Fig. 5, for HSL, MUMPS with PORD and with METIS.

Since HSL is a sequential library, its memory consumption remains constant for any number of processes. The behavior when using MUMPS depends slightly on the ordering package under test, displaying better results for the METIS case. From two processes onwards, MUMPS memory consumption has been lower than for the HSL case, and scales down acceptably.

6.3. Speed-up comparison

Fig. 6 shows the speed-up comparison among the different parallelization options, for the test case. The figure gives the speed-up for three different configurations, namely HSL, MUMPS using PORD, and MUMPS using METIS.

Since HSL is a sequential library, the HSL configuration achieves its speed-up only through the hand-crafted parallelization. However, the factorization and backward substitution phases are serial, thus limiting the maximum reachable speed-up, as the shape of the curve seems to suggest. For the test problem, the speed-up for eight processes is roughly 2.5.

The behavior of the speed-up for MUMPS, either with PORD or with METIS is roughly linear, since most part of the code has been parallelized. It reaches a value of approx. 4.5 for eight processes.

MUMPS with METIS achieves for most part of the graph a better speed-up than the one achieved by PORD. Though not shown in the graph, METIS also has the smallest execution time for any number of processes, which confirms the good features of METIS ordering package.

7. Realistic examples: setting and results

The previous sections dealt with the test case proposed in Section 3.1, supplying a detailed analysis in terms of computational load and memory consumption. In particular, the test case showed a very nice scalability, next to 100%. However, as already stated, the test case is too small so as to allow drawing definite conclusions regarding the ability of MUMPS to scale for larger problems since, at least in theory, frontal solvers are considered to display a low scalability. For these reasons, we conducted three examples using a more realistic number of unknowns.

7.1. Problem description

The problem considered for the examples is the scattering problem of a plane wave incident on a dielectric sphere with losses ($\varepsilon_r = 2.16-2j$ and $\mu_r = 1$). Conformal to the sphere surface, and separated 0.2 times the radius apart, is the fictitious border, S'. For each of the three examples, a growing number of mesh elements is considered (simulating an electrically larger sphere each time), in order to increase the computational load and memory needed to solve the problem. Table 3 shows details on each of the problems.

For each case, the number of unknowns is on the order of 7 to 8 times the number of mesh elements, which accounts for more than 2 millions of unknowns for the largest example.

7.2. Hardware and software settings

The examples were executed on a different, more powerful platform than the one used for the test case. This platform consists of a cluster of four nodes, each one of them endowed with the features shown in Table 4.

The cluster is interconnected using 1 Gbit Ethernet, and uses shared disks with enough space.

Regarding software, we used the following tools and versions:

- MUMPS 4.9.1,
- Intel MPI 3.1,
- Intel MKL 10.0.1.014,
- Intel Fortran 10.1.011.

Remark that the version of MUMPS used here is higher than the one used for the test cases (it was 4.8.3 at that time, as stated in Section 5). However, the differences between both versions are essentially limited to bug fixing and some minor enhancements.

Table 3
Example problems.

Label	DS1	DS2	DS3
# of elems.	94,939	181,685	313,028
$\simeq$ # of unknowns	750,000	1,400,000	2,500,000
Relative size	1.0	1.9	3.3

Table 4
Configuration of every cluster node.

Proc.	Intel Xeon	Intel Xeon	Intel Xeon	Intel Xeon
# of cores	8	8	8	8
Clock	2.33 GHz	1.86 GHz	2.66 GHz	2.66 GHz
Cache	4096 kb	4096 kb	6144 kb	6144 kb
Main mem.	32,889,776 kb	32,889,816 kb	32,889,812 kb	32,889,812 kb
Mem/core	8,222,444 kb	8,222,454 kb	8,222,453 kb	8,222,453 kb

MUMPS was configured so as to use "out-of-core factorization". In this mode, the factors are written to disk during the factorization phase, and are read once a solution is requested. This results in a reduction in memory requirements, while the time needed for the solution phase is only slightly increased.

7.3. Execution methodology

To obtain the desired results, each example was solved using an increasing number of processes compatible with the resources available in the cluster.

The examples DS1 and DS2 were executed using from 3 to 16 processes, distributed among the nodes in a round-robin fashion. However, example DS3 was of a big size. This forced to execute 4, 8, 12, and 16 processes so that they were distributed evenly over the nodes. Otherwise, the required memory would overflow the amount of memory available on one or more of the nodes.

Moreover, after several executions, it soon became clear that METIS was the only available ordering option for the three examples, DS1, DS2, and DS3. The option PORD was to be discarded because it required an amount of memory that totally overflowed the resources of our cluster. PORD was only effectual in our cluster when executing the example DS1, and was out of question for DS2 and DS3. For this reason, the results make use only of the METIS ordering option for all the examples. This seems to be an experimental confirmation of the strong advice given by MUMPS developers in favor of the use of METIS ordering package.

7.4. Speed-up results

Since the application displays a two-phase behavior, namely, the factorization phase and the iterative phase, we decided to show a graph of speed-up for each phase. Additionally, we give plots showing the speed-up for the complete execution and total execution time. This information is presented in Figs. 7–18, organized on a per-case basis.

In these figures, the speed-up has been computed considering the sequential time as the overall computational time measured when running with three processes. This is so because the size of the problem largely exceeds the available memory for running sequentially on a single core, thus making it necessary to create at least three processes in order to allocate the problem in memory. Data have been fitted by means of the least-square method, assuming a linear dependence on the number of processes. The value of the slope is shown on each figure. Some figures showing
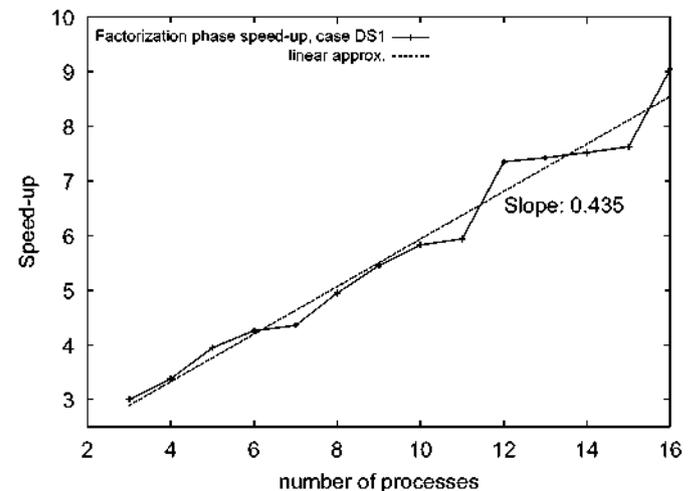

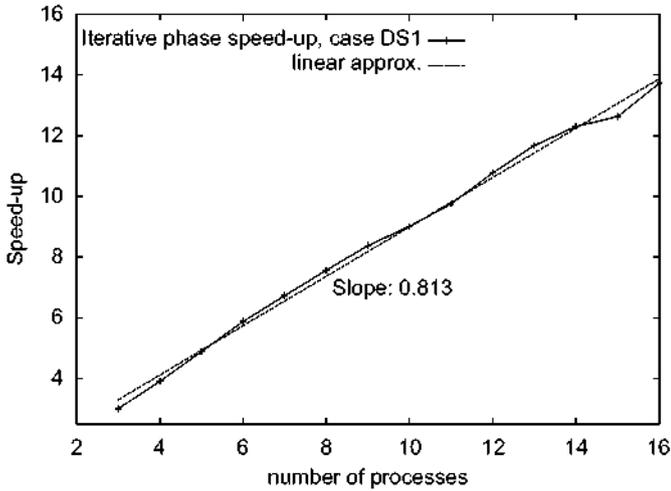Fig. 7. MUMPS speed-up for the case DS1 in the factorization phase.

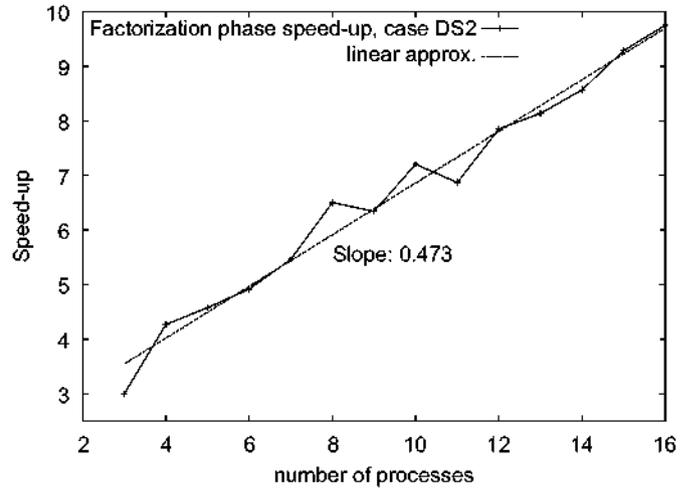**Fig. 8.** Speed-up for the case DS1 in the iterative phase.

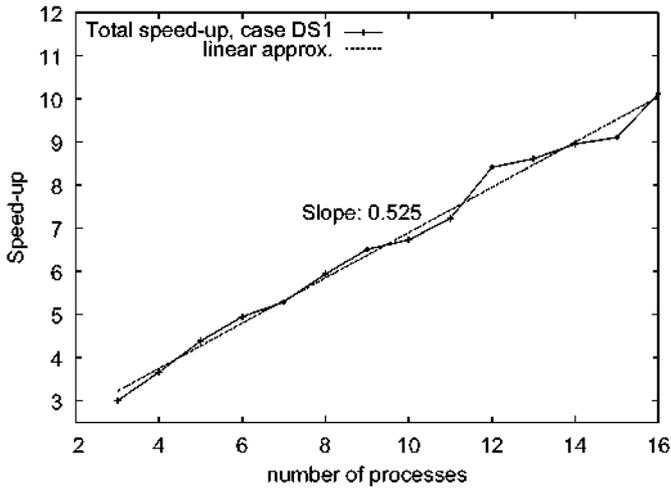**Fig. 11.** MUMPS speed-up for the case DS2 in the factorization phase.

Fig. 9. Total speed-up for the case DS1.
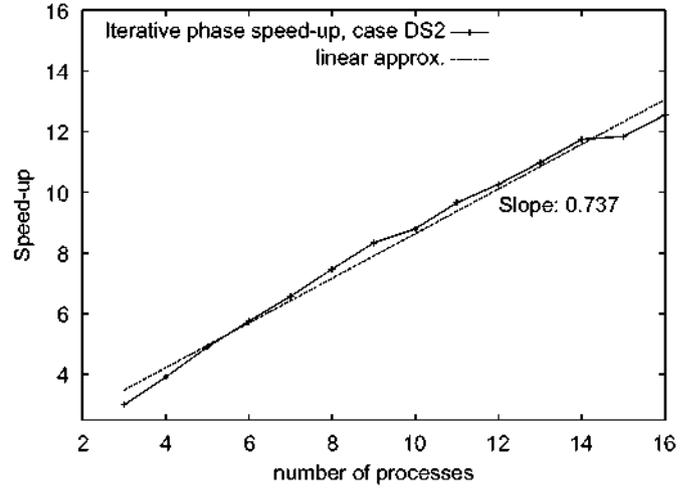
Fig. 12. Speed-up for the case DS2 in the iterative phase.
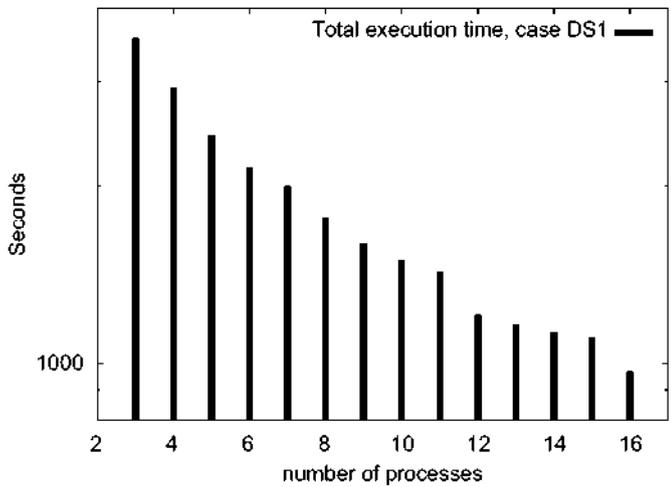
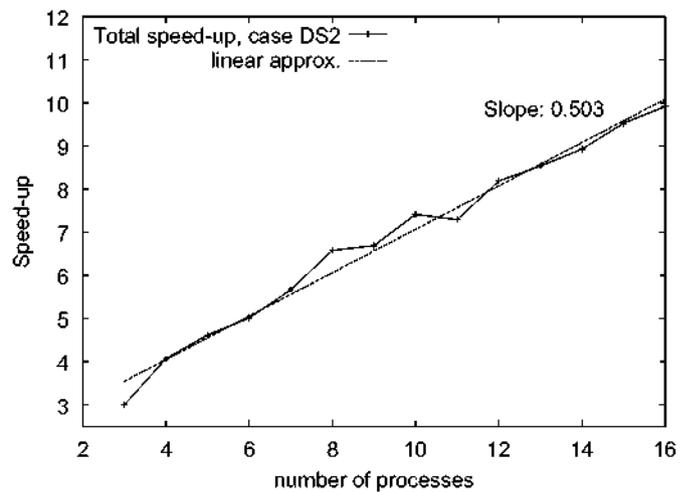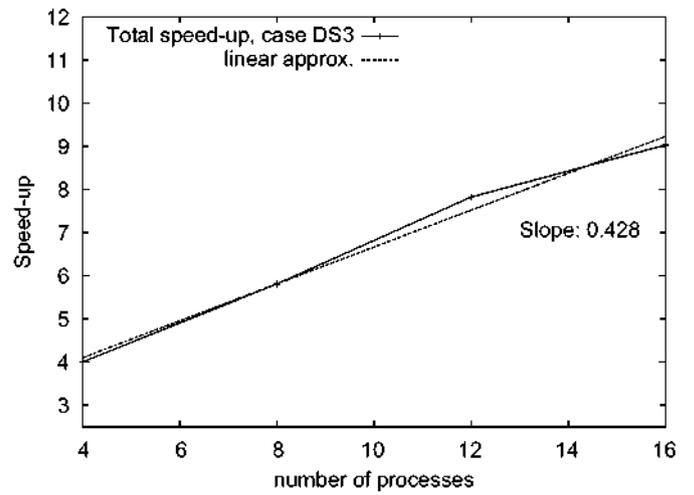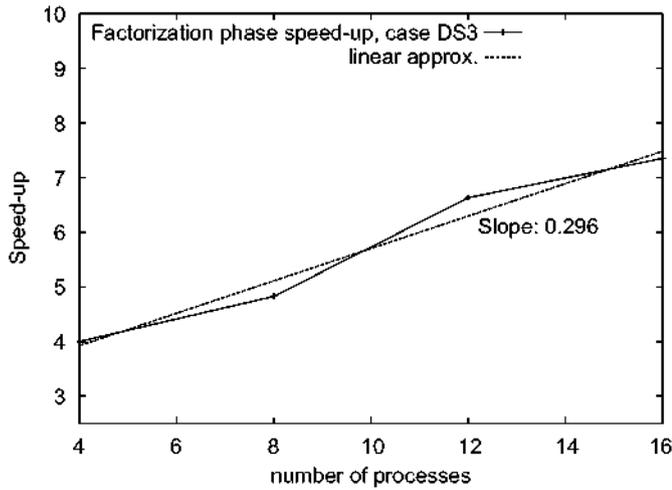**Fig. 10.** Total execution time for the case DS1.

Fig. 13. Total speed-up for the case DS2.

the total execution time (in log scale) as a function of the number of processes have been depicted for each case.

Figs. 19-21 present the relative amount of execution time devoted to the factorization phase (thick bars) and to the iterative phase (thin bars), for each example, and for the corresponding number of processes. Last, Fig. 22 puts together the bars corresponding to the three examples for 16 processes, so that a comparison among the three examples is possible at a glance.
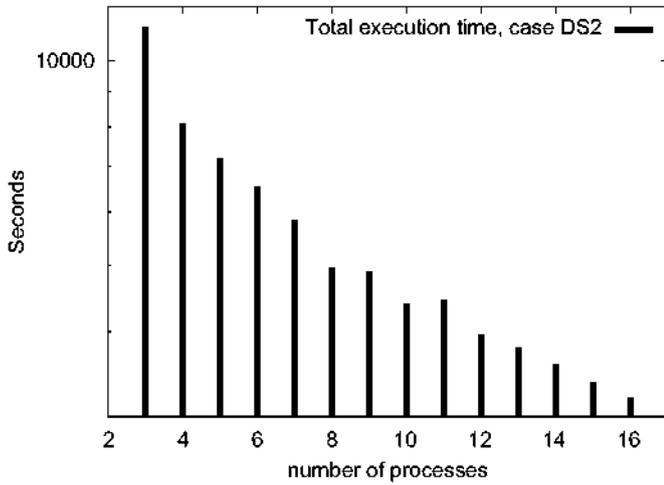
Fig. 14. Total execution time for the case DS2.

Fig. 17. Total speed-up for the case DS3.

Fig. 15. MUMPS speed-up for the case DS3 in the factorization phase.
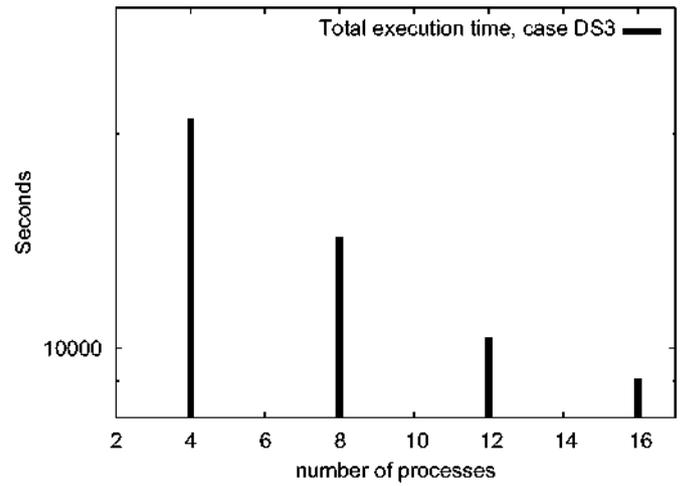

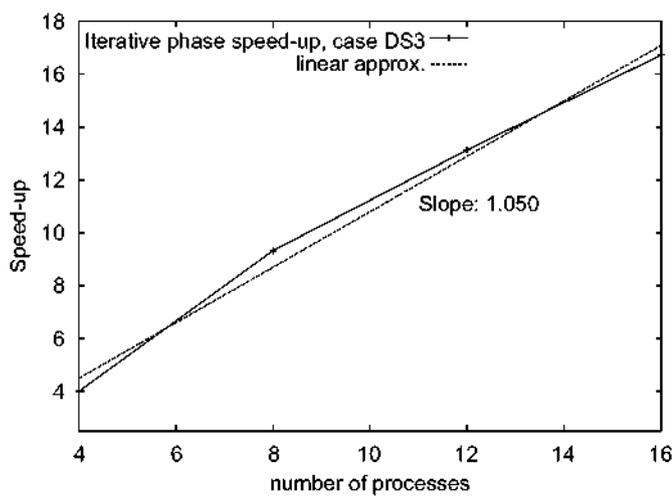Fig. 18. Total execution time for the case DS3.

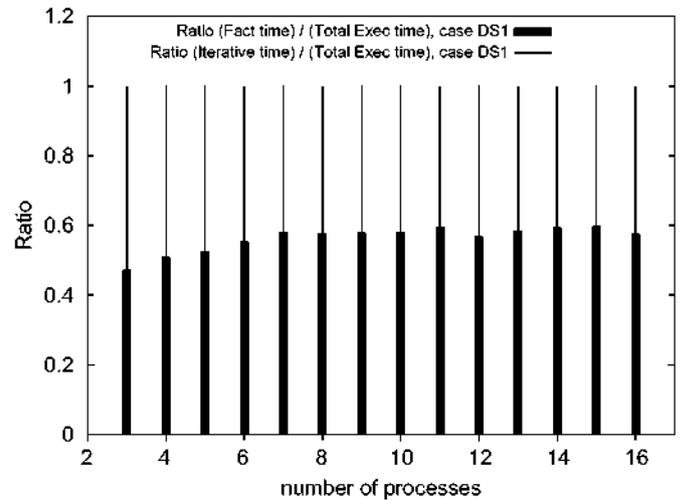Fig. 16. Speed-up for the case DS3 in the iterative phase.

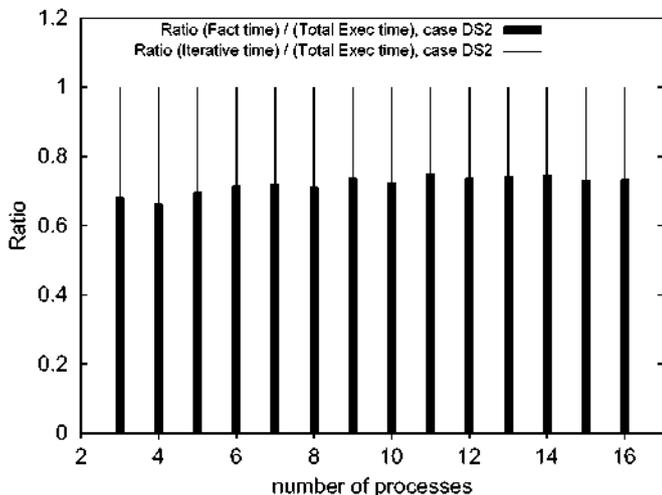Fig. 19. Ratio fact time vs. total exec. time, case DS1.

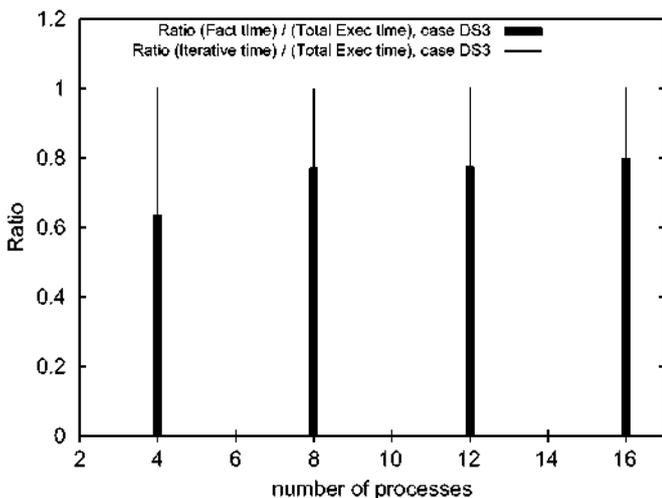**Fig. 20.** Ratio fact time vs. total exec. time, case DS2.

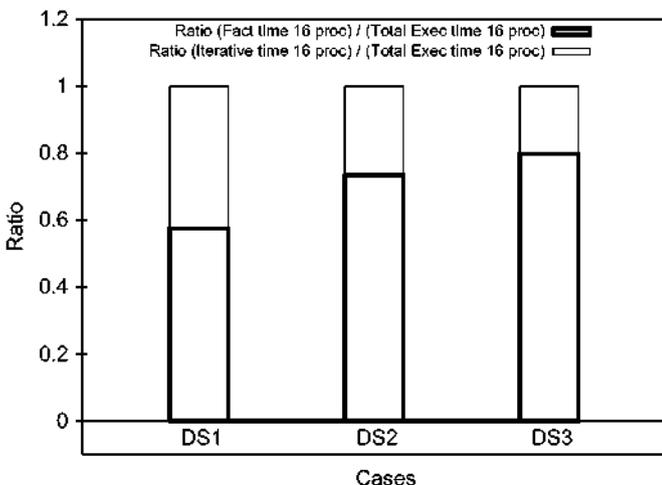**Fig. 21.** Ratio fact time vs. total exec. time, case DS3.

Fig. 22. Ratio fact time vs. total exec. time, all cases, 16 processes.

These graphs reveal a number of interesting facts, which we summarize hereafter.

- For each example, the factorization phase does not scale well, as the number of processes is increased (in other words, the slope is far from 1.0).
- Besides, also in the factorization phase, the larger the problem, the worse the scalability. This is particularly clear in Fig. 15, where the value of the slope is roughly equal to 0.3.
- The iterative phase scales much better, with values on the order of 0.8 (the value of 1.05 for DS3 might be due to a scanty number of data points).
- Also in the iterative phase, the scalability is virtually independent of the size of the problem.Figs. 9, 13, and 17
- show the speed-up for the complete execution, with slightly decreasing values as the problem size grows. The actual slopes are 0.525, 0.503, and 0.428 for DS1, DS2, and DS3, respectively.
- The behavior of the total speed-up is also apparent in Figs. 10, 14, and 18, where a regular decrease of execution time as the number of processes increases is observed. Fig. 22
- makes it apparent that, as the problem size grows, so does the fraction of execution time devoted to factorization. This means that the proportion of factorization time vs. iterative time does not stay fixed. Rather, factorization time becomes the most important contribution to the total amount of execution time as the problem size increases.

7.5. Memory usage

The most demanding resource limit is imposed by the amount of memory available in each node: the more processes are executed on the same node, the less memory is available for each process to run. So, in practice, the amount of available memory dictates the size limit of solvable problems on a given hardware platform.

The tests conducted over the examples DS1, DS2, and DS3 clearly showed that the most memory-greedy phase was the factorization. The iterative phase requires a much smaller amount of memory. For this reason, all memory-related results that we present are related to the factorization phase.

First of all, Figs. 23–25 show the average per-process memory consumption for the three examples. Remark that the vertical axis
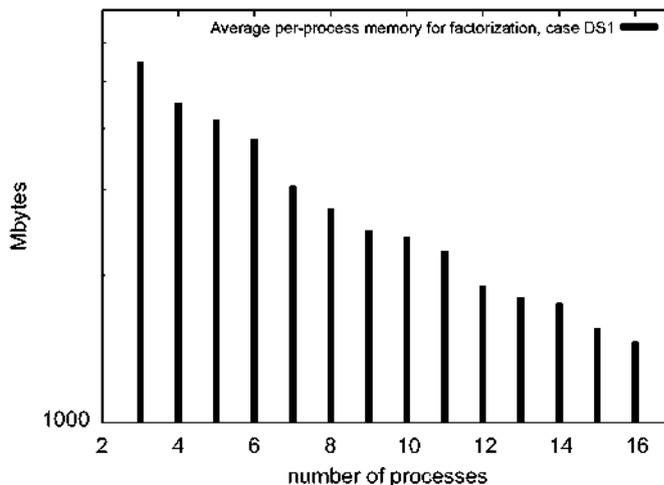

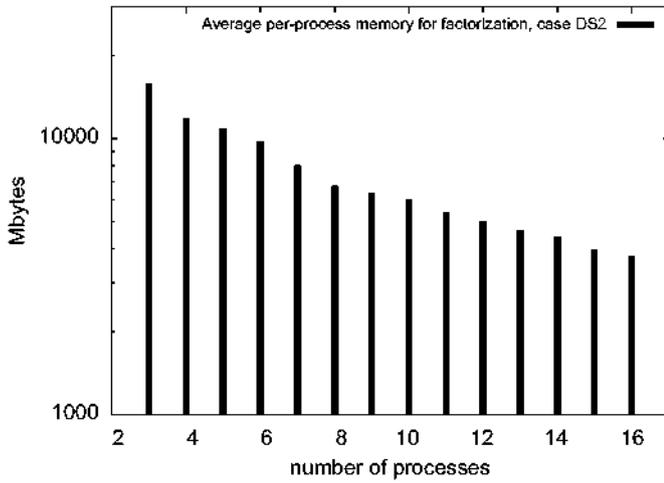**Fig. 23.** MUMPS average per-process memory consumption in factorization for the case DS1.

Fig. 24. MUMPS average per-process memory consumption in factorization for the case DS2.
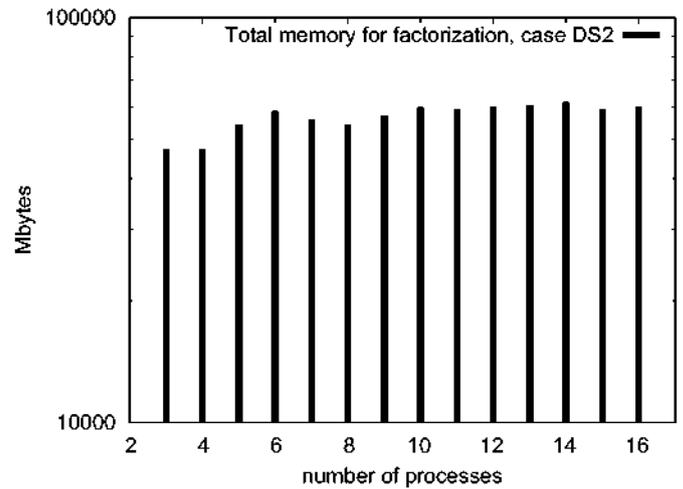

Fig. 25. MUMPS average per-process memory consumption in factorization for the case DS3.

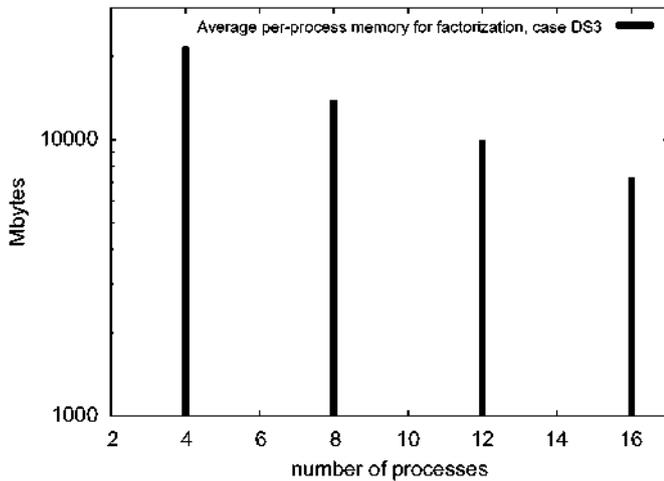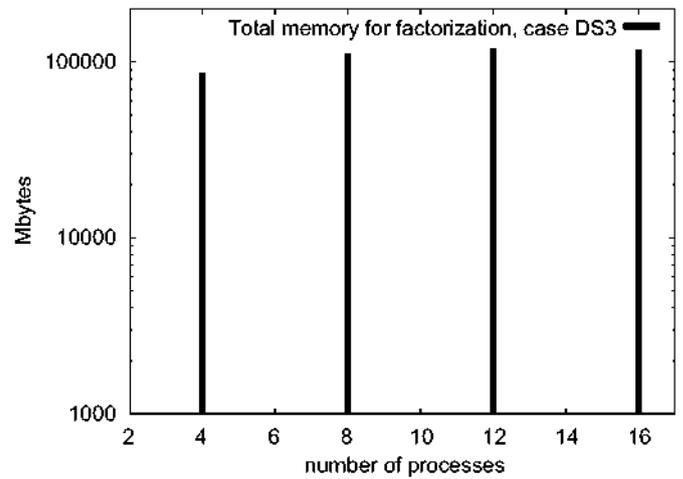**Fig. 26.** MUMPS total memory consumption in factorization for the case DS1.

Fig. 27. MUMPS total memory consumption in factorization for the case DS2.
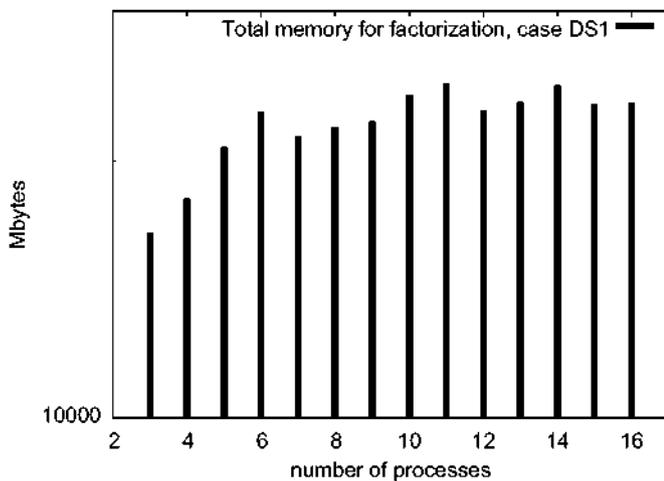

Fig. 28. MUMPS total memory consumption in factorization for the case DS3.

However, the problem of the limit in the total size of available memory remains. So, in Figs. 26–28, we plotted the product "average required per-process memory" times the "number of processes" for each example, to have an idea about the total memory required for the cluster to run a given example. We hoped that these figures showed a more or less constant amount of memory, irrespective of the number of processes. Indeed, the figures suggest a slightly rippled increase, but in general terms, it confirms the nice behavior of MUMPS regarding the scalability of the required memory as the number of processes grows.

The following table shows the averaged total memory usage for each example, along with the relative sizes of the examples, as presented in Table 3, and the relative amount of total memory used. The comparison of these two rows shows that total memory usage grows faster than the problem size by a factor of $\simeq 1.4$.

Label	DS1	DS2	DS3
Avg. total mem. usage (Mbytes)	22,139.2	56,773.2	108,405
Relative problem size	1.0	1.9	3.3
Relative tot. mem. usage	1.0	2.6	4.9
Rel. mem. size/rel. probl. size	1.0	1.4	1.5

is a log scale, and measured in Mbytes. It is remarkable that the required per-process memory scales fairly well as the number of processes increases.

Moreover, since the total amount of memory available in our cluster is roughly 130 Gbytes, it follows that DS3 is the one of the largest examples that can be run over that platform.

8. Conclusions

This paper has described the practical experience and the results obtained after the parallelization of the previously existing sequential code of a simulator that performs the analysis and resolution of general scattering and radiation problems in an efficient way by implementing FE-IIEE algorithm.

From the computational point of view, FE-IIEE algorithm consists of the iteration of a two-step process: the solution of a large sparse FEM system, and the update of the right-hand side of the system by means of the evaluation of convolution type integrals using Green's function of the exterior domain. The detailed computational analysis of the application revealed two main heavy tasks: one is the initial factorization of FEM matrix and the other is the mentioned convolution type operation with Green's function. Due to the nature of FE-IIEE, the processing on the FEM part and on the boundary integral part are decoupled, thus making their parallelization easy when compared with conventional FE-BI methods.

The parallelization of the two main tasks identified above was attempted. The parallelization of the integrals with Green's function was performed using the message-passing parallelization paradigm, assisted by MPICH2 for the test case and by INTEL MPI for the examples DS1, DS2, and DS3. Both MPICH2 and INTEL MPI are implementations of the MPI de facto standard. The parallelization of the code consisted essentially in distributing the execution of the loops to the different processes, and yielded very reasonable results. The parallelization of the factorization of the FEM matrix was attempted by replacing HSL library (originally used in the sequential version) by MUMPS library, which was designed to be run in a parallel environment (using MPI). MUMPS has proved to be very efficient for the factorization phase of our test problem, compared with HSL. The iterative phase, when the solution is obtained by forward and backward eliminations, was also performed in parallel using the cofactors within each processor.

The execution of a relatively small test case gave rise to some conclusions (see Figs. 4–6). Regarding execution time, MUMPS really was able to scale it down as the number of processes grows. The use of MUMPS along with the METIS ordering package displays always a better behavior than when used with the PORD ordering package. However, for both packages the system time consumption seems to be rather high, thus suggesting possible communication overheads inside the MUMPS library.

An important advantage of using MUMPS, independently of the ordering package used, is that memory consumption is distributed among all the processes. However, this distribution is not completely balanced. Actually, process 0 is in charge of assembling and storing the FEM matrix before factorization,[1] and for this reason, it consumes slightly more memory than the rest of the processes. Even taking this into account, MUMPS permits the successful solution of much bigger problems than if a pure sequential solver, such as HSL, were used, even on "small" machines.

Finally, three realistic examples were executed to test the scalability of the direct solver MUMPS when dealing with regular-sized problems. Some interesting conclusions were achieved.

First of all, some conclusions regarding *execution time*.

- MUMPS is able to scale the factorization phase to a certain extent, but this scalability clearly degrades as the problem size grows.
- The proportion of execution time spent on the factorization part vs. iteration using a fixed number of processes is not maintained as we increase the problem size: the factorization time weighs more and more as the problem size grows.

- The iterative phase scales reasonably well. This is related to our ability to parallelize the convolutional part of the algorithm, which accounts for most of the execution time. The solution of the system is achieved in virtually negligible time.

Secondly, some conclusions regarding *memory usage*.

- Per-process memory usage is well distributed over the nodes and scales down almost linearly as the number of processes grows.
- Total memory usage remains roughly constant for each problem size.
- Total memory usage grows faster than problem size by a factor of $\simeq 1.5$ for our examples.
- The size of solvable problems is limited by the amount of available memory on the platform. The execution time poses no problem in this regard.

Acknowledgments

References

[1] M. Salazar-Palma, T.K. Sarkar, L.E. García-Castillo, T. Roy, A.R. Djordjevic, Iterative and Self-adaptive Finite-elements in Electromagnetic Modeling, Artech House Publishers, Inc., Norwood, MA, 1998.

[2] B.H. McDonald, A. Wexler, Finite element solution of unbounded field problems, IEEE Transactions on Microwave Theory and Techniques 20 (12) (1972) 841–847.

[3] O.C. Zienkiewicz, D.W. Kelly, P. Bettess, The coupling of the finite element method and boundary solution procedures, International Journal for Numerical Methods in Engineering 11 (1977) 355–375.

[4] J.M. Jin, V.V. Liepa, Application of hybrid finite element method to electromagnetic scattering from coated cylinders, IEEE Transactions on Antennas and Propagation 36 (1) (1988) 50–54.

[5] M.M. Botha, J. Jin, On the variational formulation of hybrid finite element-boundary integral techniques for time-harmonic electromagnetic analysis in 3D, IEEE Transactions on Antennas and Propagation 52 (11) (2004) 3037–3047.

[6] B. Stupfel, M. Mognot, A domain decomposition method for the vector wave equation, IEEE Transactions on Antennas and Propagation 48 (5) (2000) 653–660.

[7] L.E. García-Castillo, I. Gómez-Revuelto, F. Sáez de Adana, M. Salazar-Palma, A finite element method for the analysis of radiation and scattering of electromagnetic waves on complex environments, Computer Methods in Applied Mechanics and Engineering 194 (2–5) (2005) 637–655.

[8] I. Gómez-Revuelto, L.E. García-Castillo, M. Salazar-Palma, T.K. Sarkar, Fully coupled hybrid method FEM/high-frequency technique for the analysis of radiation and scattering problems, Microwave and Optical Technology Letters 47 (2) (2005) 104–107.

[9] R. Fernández-Recio, L.E. García-Castillo, I. Gómez-Revuelto, M. Salazar-Palma, Fully coupled hybrid FEM-UTD method using NURBS for the analysis of radiation problems, IEEE Transactions on Antennas and Propagation 56 (3) (2008) 774–783.

[10] R. Fernández-Recio, L.E. García-Castillo, I. Gómez-Revuelto, M. Salazar-Palma, Fully coupled multi-hybrid FEM-PO/PTD-UTD method for the analysis of scattering and radiation problems, IEEE Transactions on Magnetics 43 (4) (2007) 1341–1344.

[11] J.-D. Benamou, B. Després, A domain decomposition method for the Helmholtz equation and related optimal control problems, Journal of Computational Physics 136 (1997) 68–82.

[12] L.E. García-Castillo, M. Salazar-Palma, Second-order Nédélec tetrahedral element for computational electromagnetics, International Journal of Numerical Modelling: Electronic Networks, Devices and Fields 13 (2–3) (2000) 261–287.

[13] S. Alfonzetti, G. Borzi, N. Salerno, Iteratively-improved Robin boundary conditions for the finite element solution of scattering problems in unbounded domains, International Journal for Numerical Methods in Engineering 42 (1998) 601–629.

---

[1] This occurs when the input matrix is supplied in elemental format, as it is our case.

[14] M.N. Vouvakis, K. Zhao, S.M. Seo, J.-F. Lee, A domain decomposition approach for non-conformal couplings between finite and boundary elements for unbounded electromagnetic problems in R^3, Journal of Computational Physics 225 (1) (2007) 975–994 , doi:10.1016/j.jcp.2007.01.014.

[15] C.A. Balanis, Advanced Engineering Electromagnetics, John Wiley & Sons, Inc., New York, NY, USA, 1989.

[16] HSL, A collection of fortran codes for large scale scientific computation ⟨http://www.numerical.rl.ac.uk/hsl⟩, 2002.

[17] Basic Linear Algebra Subprograms (BLAS) ⟨http://www.netlib.org/blas/⟩.

[18] S.W. Sloan, An algorithm for profile and wavefront reduction of sparse matrices, International Journal for Numerical Methods in Engineering 23 (1986) 1315–1324.

[19] J.A. Scott, On ordering elements for a frontal solver, Technical Report, 31, RAL, 1998.

[20] J.W.H. Liu, The multifrontal method for sparse matrix solution: theory and practice, SIAM Review 34 (1) (1992) 82–109.

[21] MPI: Message-Passing Interface, URL: ⟨http://www.netlib.org/mpi/⟩, 2007.

[22] MUMPS Solver ⟨http://www.enseeiht.fr/lima/apo/MUMPS/⟩.

[23] J. Schulze, Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods, BIT Numerical Mathematics 41 (4) (2001) 800–841.

[24] G. Karypis, V. Kumar, Analysis of multilevel graph partitioning, in: Supercomputing '95: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing (CDROM), ACM Press, New York, NY, USA, 1995, p. 29.

[25] A. George, J.W. Liu, Computer Solution of Large Sparse Positive Definite Systems, Prentice Hall Professional Technical Reference, Englewood Cliffs, NJ, USA, 1981.

[26] V. Kumar, A. Grama, A. Gupta, G. Karypis, Introduction to Parallel Computing: Design and Analysis of Algorithms, Benjamin-Cummings Publishing, Co., Inc., Redwood City, CA, USA, 1994.

[27] P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, Multifrontal parallel distributed symmetric and unsymmetric solvers, Computer Methods in Applied Mechanics and Engineering 184 (2–4) (2000) 501–520.

[28] 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 (1) (2001) 15–41.

[29] P.R. Amestoy, A. Guermouche, J.-Y. L'Excellent, S. Pralet, Hybrid scheduling for the parallel solution of linear systems, Parallel Computing 32 (2) (2006) 136–156.