Parallelising a simulator for the analysis of electromagnetic radiation using MUMPS library[*]

R. Rico López[†]
Dep. de Automática
Univ. de Alcalá
Alcalá de Henares, Spain
rafael.rico@uah.es

V. Escuder Cabañas
Dep. de Automática
Univ. de Alcalá
Alcalá de Henares, Spain
virginia.escuder@uah.es

R. Durán Díaz
Dep. de Automática
Univ. de Alcalá
Alcalá de Henares, Spain
raul.duran@uah.es

L.E. García-Castillo
Dep. de Teoría de la Señal y Comunicaciones
Univ. Carlos III
Madrid, Spain
luise@tsc.uc3m.es

I. Gómez-Revuelto
Dep. de Ingeniería Audiovisual y Comunicaciones
Univ. Politécnica de Madrid
Madrid, Spain
igomez@diac.upm.es

J.A. Acebrón
Center for Mathematics and its Applications
Lisbon, Portugal
juan.acebron@ist.utl.pt

ABSTRACT

The practical experience of parallelising a simulator of general scattering and radiation electromagnetic problems is presented. The simulator stems from an existing sequential simulator in the frequency domain and can be fruitfully used in applications such as the test of coverage of a wireless network, analysis of complex structures, and so on. After the analysis of a test case, two steps were carried out: firstly, a "hand-crafted" code parallelisation was developed within the kernel of the simulator. Secondly, the sequential library, used in the existing simulator, was replaced by the parallel MUMPS library in order to solve the associated linear algebra problem in parallel. For factorising the matrix with MUMPS, two different ordering methods have been considered.

Keywords

Parallel Computing, MPI, Sparse Direct Solvers, MUMPS library

1. INTRODUCTION

This paper presents the practical experience and the results obtained in the parallelisation of the code of a simulator that implements a novel hybrid Finite Element-Boundary Integral, known as *Finite Element - Iterative Integral Equation Evaluation* (FE-IIEE) method ([11], [12], [10]), based on the well-known Finite Element Method (FEM). This method permits an efficient analysis and solution of general problems of radiation and scattering of electromagnetic waves.

The analysis of the radiation and scattering of electromagnetic waves is an important issue that finds applications in many electromagnetic engineering areas. Currently, companies face the challenge of cutting costs in many areas and in this context, it is very interesting the use of simulators before the actual setting up of several kinds of resources. One of such resources is, for example, the growing demand of wireless networks in urban areas. In this and other settings, simulators can be advantageously used in order to reduce the time to availability and design cycles of products.

Modern radiating structures, which may exhibit complex configurations with the presence of different materials, call for the use of FEM (see, for example, [17]), which is very flexible and able to handle 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. For this reason, FEM is hybridised with the use of the Boundary Integral (BI) representation of the exterior field, thus endowing the FEM analysis with a numerically exact radiation boundary condition at the mesh truncation boundary. Several hybrid schemes FEM-BI have been proposed (see, for example, [16, 21, 13]). 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.

In this work we show the experience and results obtained along the process of parallelising an already existing sequential FE-IIEE 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.

[*]Jointly funded by Comunidad Autónoma de Madrid and Universidad de Alcalá under grant number CAM-UAH2005/042. Also, supported by the Ministerio de Educación y Ciencia, Spain, under project TEC2007-65214/TCM

[†]Corresponding author.

VALUETOOLS 2009, October 20-22, Pisa, Italy
Copyright © 2009 ICST 978-963-9799-70-7
DOI 10.4108/ICST.VALUETOOLS2009.7456

2. COMPUTATIONAL ANALYSIS OF THE SIMULATION METHOD

In this section we analyse the consumption of computational resources by the different building blocks inside existing the code in order to prioritise their parallelisation. 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.

2.1 Methodology

The computational analysis of the existing code of the simulator was carried out by running a test problem. The test problem was selected keeping in mind that the available computational resources were rather limited. The test problem consists in the scattering problem of a plane wave incident on a dielectric cube with losses. The number of mesh elements is 2,684, and the number of unknowns is 18,288.

The available system was a cluster of eight nodes (known as blades by the manufacturer, Sun Microsystems) model *SunFire B1600*. The technical details of each node can be found in Table 1.

Table 1: Configuration of every cluster node

Processor:	AMD Athlon XP-M 1800+
Clock:	1.5 Ghz
Cache:	256 KB
Main memory:	1 GB
Hard disk:	30 GB
Network interface:	1 Gbps

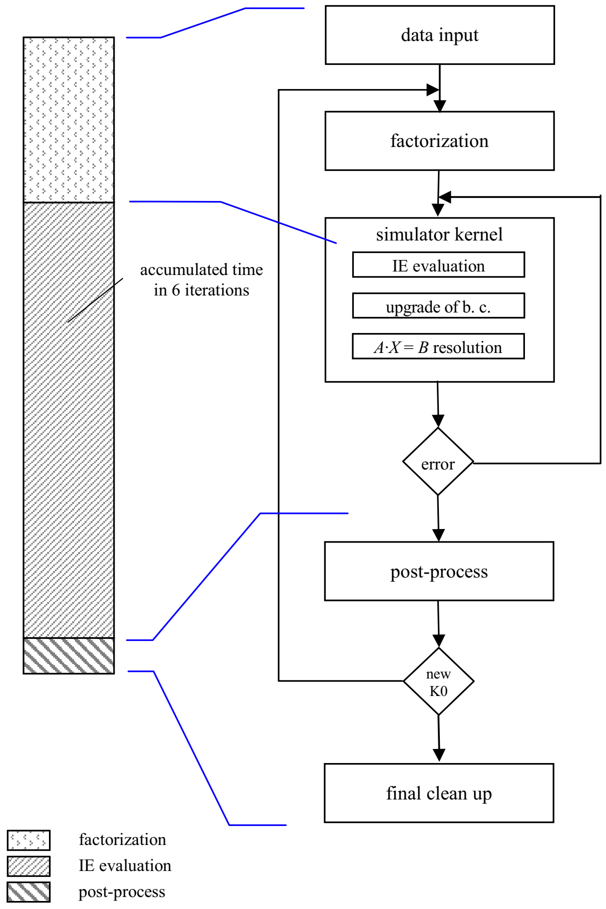

Figure 1: Program flow and monitorisation points

2.2 Monitorisation

The sequential source code was 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 factorised;
- an iterative phase, where the code performs a number of iterations inside the simulator kernel before achieving the required accuracy. For the test problem considered, obtaining a threshold relative error δ between subsequent iterations below 10^{-4} required a total of six iterations.

The previous scheme can be repeated for different frequencies. In our test case, only one frequency, denoted $K0$ in Fig. 1, was used.

The results obtained with the previous execution are depicted in Fig. 1, showing the points where the monitorisation was inserted. Only those blocks that have a relevant computational load are shown. In this Fig. it is apparent that the simulator kernel consists of the evaluation of a double integral, the upgrade of the boundary conditions and the solution of a linear sparse system, $A \cdot X = B$. Both the initial phase and the iterative phase 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 [2]) as building block for simulation.

HSL is a collection of Fortran packages for large scale scientific computation that makes extensive use of BLAS package (Basic Linear Algebra Subprograms, see [1]). The sequential code uses a Fortran solver: either ME62 or ME42 routines, depending on whether the matrix A is symmetric/hermitian or not. Optionally, those routines may work with direct-access files for the matrix factors so that large problems can be solved using a relatively small in-core memory. However, the storage in disk has a limited speed to access factors and the size of the files can render the situation unmanageable as the number of unknowns grows.

Thus, the parallelisation is motivated to overcome the limitations of the sequential code, so that larger problems can be run in a reasonable time frame and with limited memory resources. Specifically, the results relative to the present implementation shown in this paper have been obtained using ME62 Fortran solver (complex symmetric case) with in-core

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7456
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7456

storage.

The algorithm used for element ordering (crucial for the performance of the frontal solver) with HSL is a variant of Sloan's algorithm [20], implemented in MC63 routine of HSL. Indirect element ordering with default MC63 parameters is used prior to calling the frontal solver factorisation routines.

2.3 Analysis of the monitorisation

Regarding time consumption (depicted qualitatively in Fig. 1), the monitorisation showed that the largest time consumption takes place in:

- The factorisation of the matrix (performed only once in the first iteration).
- The block named as "IE Evaluation", inside the simulator kernel. This block corresponds to the evaluation of double surface integrals (the core of FE-IIEE algorithm).

The main conclusions of the analysis are the following. Once the FEM matrix is factorised, 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 practise that the computational load of this method, once the FEM matrix is factorised, is essentially in the operation corresponding to the evaluation of the integrals. 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.

Yet another conclusion is that the program flow is independent of the input data set since the two phases described above are always executed. Thus, if we analyse the performance gain due to parallelisation for a particular input data set, then it is to be expected that similar performance gains can also be obtained when using another (possibly larger) input data set.

2.4 Parallelisation approaches

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 parallelisation was accomplished, involving "IE evaluation" block, since it is the main consumer of computer cycles.
2. Concerning the initial factorisation of the matrix and the block "$A \cdot X = B$ resolution", the goal was to replace the solver in the HSL library by MUMPS library, a parallel sparse solver. We expected that MUMPS 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 new code includes the benefits coming from both approaches: the convolution operation has been parallelised and the solver has been changed. Both items are dealt with in detail in the next sections: The first item is described in section 3, and the second one is analysed in section 4.

3. HAND-CRAFTED PARALLELISATION

The parallelisation was carried out using the message passing interface (MPI, [5]), with the support of the MPICH2 library (specifically, version 1.0.8)

The hand-crafted code parallelisation targeted essentially the heavy computation load given by the convolution-type operations involved in the "IE evaluation" of the exterior problem. The fact is that a double loop is required since a double surface integral is involved, as already explained.

Thus, the parallelisation 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 data, needed to resume the computation.

This method involves a small amount of communication, basically a *reduction* in MPI jargon, performed at the end of the convolution. For this reason, the communication time is almost negligible.

4. REPLACEMENT OF THE SEQUENTIAL SOLVER

We focus now on replacing the sequential sparse solver by an efficient parallel solver. As previously stated, we have chosen MUMPS for this purpose.

4.1 The MUMPS library

MUMPS (see [3]) is a package for solving systems of linear equations of the form $AX = B$, where A is a square sparse matrix that can be either unsymmetric, symmetric positive definite, or general symmetric. MUMPS uses a multifrontal technique which is a direct method based on either the LU or the LDL^T factorisation of the matrix. MUMPS exploits both parallelism arising from sparsity in the matrix A and from dense factorisation kernels. MUMPS offers several built-in ordering algorithms, a tight interface to some external ordering packages such as METIS [14] and PORD [18], and the possibility for the user to input a given ordering. The parallel version of MUMPS requires MPI for message passing and makes use of BLAS, BLACS [4], and ScaLAPACK [6] libraries.

MUMPS distributes the work tasks among the processes, but an identified process (the master) is required to perform most of the analysis phase, to distribute (if the matrix is centralised) the incoming matrix to the other processes (slaves), and finally collecting the solution.

The system $AX = B$ is solved in three basic steps:

1. Analysis. The master performs an ordering based on the symmetrised pattern $A+A^T$, and carries out symbolic factorisation.
2. Factorisation. The original matrix is first distributed to processes that will participate in the numerical factorisation. The numerical factorisation on each frontal matrix is conducted by a *master processor* and one or more *slave processors*. Each processor allocates an array for the so-called contribution blocks and for the factors; the factors must be kept for the solution phase.
3. Solution. The right-hand side B is broadcast from the master to the other processes. These processors compute the solution X using the distributed factors computed during Step 2.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7456
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7456

The analysis phase is performed on the master process. This process is the one with rank 0 in the communicator provided to MUMPS. By setting a special variable, MUMPS allows the master to participate in computations during the factorisation and solve phases, just like any other process.

4.2 Plugging in the MUMPS library

Our goal was the replacement of the HSL library, used by the sequential version of the simulator, by the MUMPS library inside both the factorisation 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. Due to the nature of the test cases involved, we chose to set up MUMPS for the general unsymmetric case. We used complex double precision arithmetic and we actually let the master participate in the computations. We chose the most up-to-date version of MUMPS at the onset of this project, 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. 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. Some pieces of code received strong modifications, though the details are rather too technical to be dealt with here. To summarise, by cross-examination and comparing, we found a correspondence between the routines in HSL and the ones performing the similar tasks in MUMPS. We adapted the interface of the corresponding routines to the data structures inside the code. Eventually, the replacement, though not completely straightforward, was carried out successfully.

4.3 Changing the ordering package

As it was mentioned before, MUMPS requires three steps to solve the linear system and a (re-)ordering is mandatory during the analysis phase. However, as stated above, MUMPS leaves to the user the choice of basically two external ordering packages. Initially we considered PORD package, since it is the default option. We, then, recompiled MUMPS to include METIS ordering package. Actually, the latter is strongly recommended by MUMPS developers. The practical effects on the performance due to the selection of the package will become apparent later on.

5. TRADE-OFFS AMONG PARALLELISATION OPTIONS

In this section, we present the results obtained, once the hand-crafted parallelisation, and the replacement of HSL sequential solver by MUMPS library (and corresponding interface) have been performed.

To enable some comparisons, a number of executions of the test problem were performed, using from 1 to 8 cluster processes and an executable with the hand-crafted parallelisation inside the convolution loop and three configurations for the solver:

- with HSL (labelled HSL in the figures);
- with MUMPS with PORD (MUMPS & PORD in the figures);
- with MUMPS with METIS (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.

5.1 Trade-offs of the ordering packages

The results depicted in Fig. 2 refer to the test case mentioned in section 2.1, and deserve several comments. It is worth noting that 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. Regarding CPU time, MUMPS using METIS outperforms MUMPS using PORD as well. As the number of process grows, the performances of MUMPS with both ordering schemes become closer, though.

The third picture shows 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. 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 behaviour, so that in practise this time does not impact significantly on the overall performance. However, the origin of this irregularities is still to be ascertained.

5.2 Memory trade-offs

The results for memory consumption in the test case can be seen in Fig. 3, 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 behaviour 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.

In general, memory consumption for a multifrontal solver (for a full explanation on multifrontal solvers, see for example [15]), 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.

5.3 Speedup comparison

Fig. 4 shows the speedup comparison among the different parallelisation options, for the test case. The figure gives two results, CPU time and speedup, for three different configurations, namely HSL, MUMPS using PORD (labelled

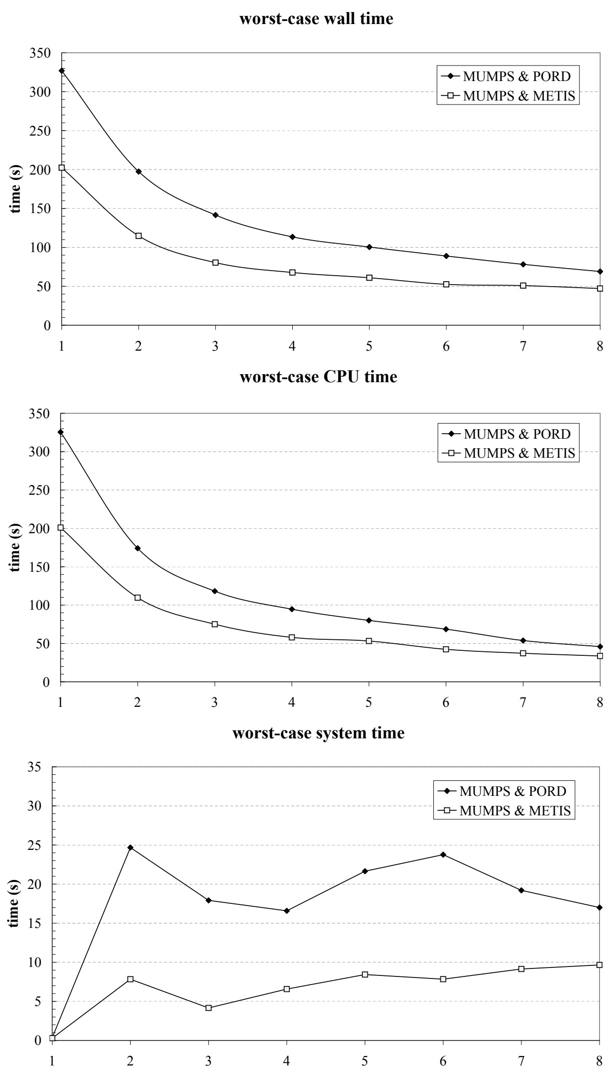

Figure 2: MUMPS (with PORD and METIS element orderings) time comparisons as a function of the number of processes

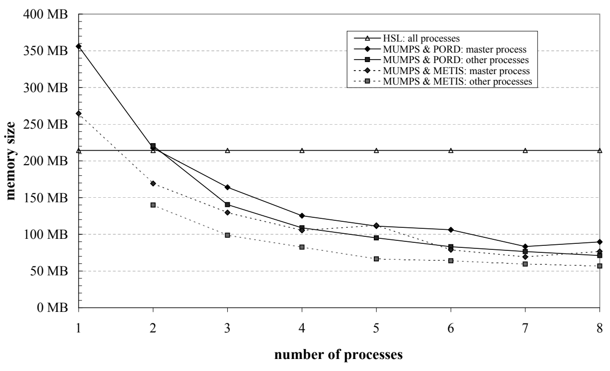

Figure 3: HSL & MUMPS (with different element orderings) memory usage comparisons

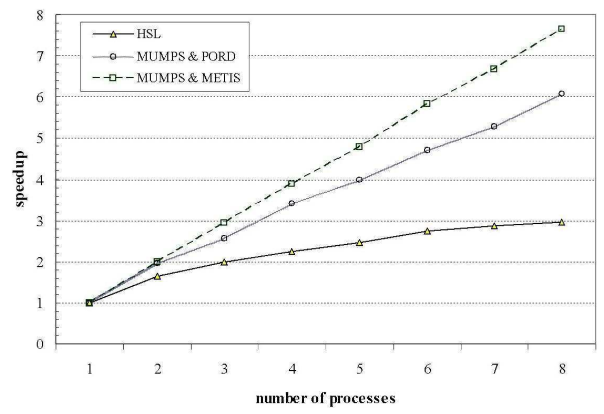

Figure 4: HSL & MUMPS(with different element orderings) CPU time and speedup

MUMPS in the figure), and MUMPS using METIS.

Since HSL is a sequential library, the HSL configuration achieves its speedup only through the hand-crafted parallelisation. However, the factorisation and backward substitution phases are serial, thus limiting the maximum reachable speedup, as the shape of the curve seems to suggest. For the test problem, the speedup for 8 processes is roughly 3.

Regarding MUMPS with PORD, it is remarkable that, for the test problem, MUMPS only outperforms HSL in terms of CPU time when the number of processes is greater than 3. This behaviour is due to the longer times used by the factorisation phase when the number of processes is low. As for the speedup curve, it displays a nearly linear behaviour, since most part of the code has been parallelised. It reaches a value of 6 for 8 processes.

MUMPS with METIS achieves the best behaviour even using only one process. The METIS ordering package is key to the quasi-linear speedup observed (nearly 8, for 8 processes) so it can be deemed fully scalable. This speedup in MUMPS using METIS is essentially due to a much improved factorisation process with respect to MUMPS using PORD.

6. CONCLUSIONS

This paper has described the practical experience and the results obtained after the parallelisation 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 a initial phase, where the matrix is factorised, and an iterative phase, where a system of double integrals is evaluated. Both tasks account for the main computational load of the simulator. The parallelisation of the integrals was performed using the message-passing parallelisation paradigm, assisted by MPICH2, an implementation of the MPI de facto standard, and consisted essentially in distributing the execution of the loops to the different processes, thus yielding very reasonable results. The parallelisation of the factorisation of the FEM matrix was attempted by replacing HSL library (originally used in the sequential version) by MUMPS library, designed to be run in a parallel environment (using

MPI). MUMPS has proved to be very efficient for the factorisation phase of our test problem, which runs faster than HSL when more than 3 processes are involved. Moreover, MUMPS was recompiled to take advantage of METIS, showing that the behaviour of the program notably improved in terms of wall time.

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 factorisation, and for this reason, it consumes slightly more memory than the rest of the processes. Even taking this into account, the use of a cluster such as the one used in this work (endowed with rather limited resources) permits the successful solution of much bigger problems than if a pure sequential solver, such as HSL, were used.

It is interesting to remark, as a lesson learnt, that the performance of our sequential simulator has been greatly improved with no too much effort by using an existing tool, such as MUMPS solver, which conveniently replaced the sequential solver with a parallel one, thus dramatically increasing the size of the problems that can be attacked and, as a by-product, improving the speed performance.

Moreover, our sequential simulator turned out to be amenable to a simple and performant parallelisation since a big part of the computational load was due to a convolution-type operation, which was very easy to distribute among the processes with virtually no communication cost added.

7. ACKNOWLEDGEMENTS

As per usage conditions, we kindly acknowledge the use of HSL 2002 ([2]) as a tool to achieve part of the results reported in this paper.

We also take the opportunity to congratulate and thank the MUMPS team for their great job done and kind support. For further details of MUMPS, see, for example, [7], [8], and [9].

8. REFERENCES

[1] Basic Linear Algebra Subprograms (BLAS). `http://www.netlib.org/blas/`.

[2] HSL (2002). A collection of fortran codes for large scale scientific computation. `http://www.numerical.rl.ac.uk/hsl`.

[3] MUMPS Solver. `http://www.enseeiht.fr/lima/apo/MUMPS/`.

[4] Basic Linear Algebra Communication Subprograms, 2007.

[5] MPI: Message-Passing Interface, 2007.

[6] Scalable Linear Algebra PACKage, 2007.

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

[8] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

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

[10] R. Fernández-Recio, L. E. García-Castillo, I. Gómez-Revuelto, and 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):774–783, Mar. 2008.

[11] L. E. García-Castillo, I. Gómez-Revuelto, F. Sáez de Adana, and 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*, 194/2-5:637–655, Feb. 2005.

[12] I. Gómez-Revuelto, L. E. García-Castillo, M. Salazar-Palma, and 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):104–107, Oct. 2005.

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

[14] G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, page 29, New York, NY, USA, 1995. ACM Press.

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

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

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

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

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

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

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

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7456
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7456