The High Performance Solution of Sparse Linear Systems and its Application to Large 3D Electromagnetic Problems

D. Goudin, A. Pujols, M. Sesques, M. Mandallena, J.J. Pesqué, B. Stupfel[1]

CEA/DAM/CESTA, BP 2, F-33114 Le Barp, France
david.goudin@cea.fr

Abstract. The numerical treatment of high frequency electromagnetic scattering in inhomogeneous media is very computationally intensive. For scattering, the electromagnetic field must be computed around and inside 3D complex bodies composed of inhomogeneous media. Because of this, accurate numerical methods must be used to solve Maxwell's equations in the frequency domain. In this paper, we consider the hybrid integral equation and the finite element techniques. For some high frequency applications, these numerical approaches lead to linear systems that are too large for current computer architecture. In order to solve these very large systems, typically tens of millions of Degrees Of Freedom (DOF), we have combined modern numerical methods with very efficient parallel algorithms.

Related topics: parallel computing, large scale simulation

1 Introduction

Our primary focus is on the simulation of electromagnetic (EM) phenomena around 3D stealthy bodies using the time harmonic formulation. This problem leads to numerically solving Maxwell's equations in penetrable bodies and unbounded domains. We are interested in predicting Radar Cross Section (RCS) of bodies. The computation of RCS for complex 3D bodies for high frequencies can lead to systems with millions of DOF. For these simulations, we need to combine efficient algorithms, multiple levels of parallelism, and high performance linear solvers. Our main goal is to study 3D bodies that may be composed of complex materials.

High accuracy is an important aspect for these computations, especially in achieving confidence for very low levels of RCS. From a numerical point of view, we have to overcome two major hurdles. First, the computational domain is unbounded. Thus, one procedure consists of truncating the domain and putting an Absorbing Boundary Condition (ABC) on the interface (Limiting Conditions of Berenger for example). But for our applications, this approach is not sufficiently accurate. We choose instead to take into account the exact far field radiation condition using a Boundary Integral Equation (BIE) on the interface, which can be very close to the target surface or even on the target surface itself. This BIE formulation leads to a dense linear system (complex non-hermitian but symmetric).

The second major difficulty is discretization accuracy, which correlates to very large linear systems. In fact, in the time-harmonic formulation, the size of the mesh, and thus the linear system, is a function of the wavelength. Typically, we need to put at least 10 discretization points per wavelength in each direction. With the domain sizes mentioned above, problems with several millions of DOF are common. So, in order to reach higher frequency levels, we need very efficient numerical methods, as well as high performance computing.

2 Surface discretization

Our initial electromagnetics simulation codes were based on numerical methods such as the BIE method and Partial Differential Equation (PDE). The PDE discretization uses the finite elements method and gives rise to the solution of linear systems. Concerning the BIE method, we choose a classical BIE approach with the Electric Field Integral Equation method (EFIE). It leads to a dense linear system with complex coefficients. This matrix is dense, non-hermitian but symmetric. Using this formulation, since the matrix does not have good properties, iterative methods do not perform well.

The main advantage of this BIE method is that only the surface has to be meshed when the body is perfectly conducting. When the body is composed of coating with high refractive index, discretization of the external surface is sufficient using a local reflection operator R (function of the surface impedance of the coating). For more complicated isotropic coating, we only have to mesh the interfaces of all of the homogeneous zones. These interfaces are typically between the homogeneous medium and the outer boundary of the body where the RCS is given. For this reason, the number of electric and magnetic current unknowns on the interface is reasonable. However, this approach leads to a dense matrix.

The BIE method is not suitable when we have many thin media layers. In fact, the number of DOF nearly equals to the one given by a volume finite element discretization, and moreover, the BIE matrix is dense. It is well known, for large-scaled problems, dense direct methods are infeasible because they require the storage of N^2 entries of the matrix, and $O(N^3)$ floating point operations to compute the factorization. Moreover, thin layers can cause numerical difficulties in the computation of matrix coefficients involving DOF that belong to elements that are too close.

Finally, non-isotropic coatings are also non-tractable for the BIE. Thus, in order to overcome all these complexities, we have developed a hybrid numerical method.

3 Hybrid numerical method: coupling volumic subdomains and boundary formulations

We have extended our surface discretization and BIE code to include an hybrid discretization approach. To describe this approach, consider the coupling interface between two subdomains, the interior of the penetrable body and the truncated exterior domain. The interface is the surface of the body. For the exterior domain we still use the BIE so that the radiation boundary condition can be satisfied. In the interior, we apply a Finite Element (FE) method to a PDE. The unknowns in the interior are the electric fields. This coupling method leads to a mesh that consists of tetrahedrons in the interior domain, and triangles on the coupling interface. The unknowns of this hybrid method are the electric fields in the interior and the electric and magnetic currents on the interface.

The key idea of this new approach is to partition the domain into concentric regions and to apply a Robin transmission condition between the interfaces. On the outermost boundary, the radiation boundary condition is handled with a new BIE formulation (Despres Integral Equations, EID). This formulation is used because of the algebraic structure of the generated linear system. In particular, this structure allows us to derive convergence theory for iterative methods.

To solve the complete linear system (the system with concentric subdomains and the outer boundary problem), rather than applying a direct linear solver method, we have adopted a direct/iterative domain decomposition approach. In this Domain Decomposition, an outermost iteration is combined with an inner loop involving direct methods for the volume subdomains problems. For the outer boundary condition, a separate iterative scheme is also used (fast multipole method).

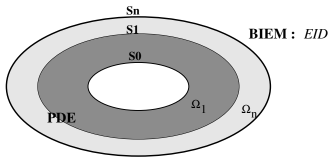

Fig. 1. A weak coupling of numerical methods.

3.1 Domain Decomposition

The volume domain is divided into m concentric subdomains Ω_k, starting from the interior to exterior. The radiation problem, the last domain Ω_m, is the subdomain where we use the EID formulation. We proceed in an iterative fashion solving this domain decomposition by looping over the subdomains from the interior to exterior.

The domain decomposition algorithm, which is a block Gauss Seidel cycle, is intrinsically sequential: the first subdomain Ω_1 is solved then the second one Ω_2, continuing until Ω_m where the integral equation is solved.

The boundary condition of each domain is taken at the interior interface at iteration (n+1) and at the exterior interface at iteration n. On the interior interface of the first domain, which is the boundary of the obstacle, we apply the perfectly conducting boundary condition (or Leontovitch condition). For the last subdomain, we apply the radiation boundary condition using the BIE.

Since obviously our Domain Decomposition Method (DDM) is intrinsically sequential, our parallelization effort will focus on the solution procedure for each subdomain solve. Each of these subdomain solves can consist in either a standard parallel Conjugate Gradient iteration or a parallel direct method. For the purpose of high performance computing, and overall for good convergence, we choose the parallel direct solver library EMILIO, developed in collaboration with the Bacchus team of INRIA Bordeaux Sud-Ouest (France).

4 Emilio : software library for solution of large sparse linear systems by direct methods

As explain in section 3, we choose to use parallel direct methods for their robustness and the predictability of their performances. Over the past few years, direct methods have made significant progress thanks to research studies on both the combinatorial analysis of Gaussian elimination process and on the design of parallel block solvers optimized for high-performance computers. It is now possible to solve real-life three-dimensional problems having in the order of several millions equations, in a very effective way with direct solvers. This is achievable by exploiting superscalar effects of modern processors and taking advantage of computer architectures based on networks of SMP nodes. All of the techniques described in this paper have been integrated in the PASTIX software, which uses the static mapping and sparse matrix ordering software package SCOTCH. To get more details about all the techniques described here, it is easy to find some papers in Scientific Journals or in proceedings.

The main characteristic of our solver library is that it relies on blockwise algorithms and on a load balancing and computational task scheduling that take into account the target architecture. We use these techniques for high performance sparse supernodal LDL^T or LL^T parallel factorization without pivoting for large sparse symmetric positive definite systems, and for sparse supernodal LU parallel factorization with static pivoting for non-symmetric matrices having a symmetric pattern.

The library developed at INRIA is called PASTIX while the industrial version of this software (EMILIO) includes in addition algorithms for parallel assembly in the context of the finite element method. Moreover, as in our EM problems we only deal with complex coefficients, EMILIO performs all of its computations on complex variables in the symmetric matrix.

In order to achieve efficient parallel sparse factorization, we consider the following pre-processing phases (see figure 2).

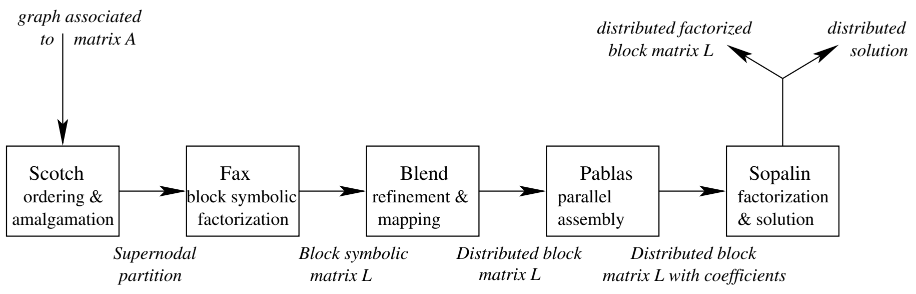
graph associated to matrix A
Scotch ordering & amalgamation
Supernodal partition
Fax block symbolic factorization
Block symbolic matrix L
Blend refinement & mapping
Distributed block matrix L
Pablas parallel assembly
Distributed block matrix L with coefficients
Sopalin factorization & solution
distributed factorized block matrix L
distributed solution

Fig. 2. Emilio/PaStiX library for solving large sparse linear systems.

– The *ordering* phase (performed by the Scotch software) computes a symmetric permutation of the initial matrix A such that factorization process 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 original graph 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.

– The *block symbolic factorization* phase (performed by the Fax software) determines the block data structure of the factorized matrix L associated with the partition resulting from the ordering phase. One can efficiently perform such a block symbolic factorization in quasi-linear space and time complexities. From this block structure, one can deduce the weighted task graph that captures all dependencies between blocks, as well as the supernodal elimination tree.

– The *block repartitioning and scheduling* phase (performed by the Blend software) refines the previous partition by splitting large supernodes in order to exploit concurrency within dense block computations, and maps the resulting blocks onto the processors of the target architecture. We presented in a mapping and scheduling algorithm based on a combination of 1D and 2D block distributions. This algorithm computes an efficient static scheduling of the block computations for the supernodal parallel solver that uses local aggregations of contributions to minimize the communication volume. This can be done by very precisely taking into account the computational costs of the BLAS3 primitives, the communication cost and the cost of local aggregations.

– The *parallel sparse factorization* phase (performed by the Sopalin software) is fully driven by the scheduling of the block computations produced by the previous phase.

The *parallel backward forward solves* phase uses the same scheme of computations and communications as Sopalin.

- The *parallel assembly* phase is a special phase, because it doesn't exist in the PaStiX software but only in Emilio. This phase is performed by the Pablas software. It concerns the matrix associated with finite elements discretization. Since, as said above, the global matrix is distributed (see in previous lines), the parallel assembly is guided by two distributions : the distribution of the blocks of the factorized matrix (computed by Blend) and the distribution of the elements of the mesh. This distribution assigns a roughly equivalent number of mesh elements to every processor (it uses criteria of locality and information from the mesh). The assembly is fully parallel and could be used in different domains of application and thus in different simulation codes based on finite element method.

Nevertheless, a major memory bottleneck in our parallel supernodal factorization scheme is caused by this local aggregation mechanism. The local aggregation mechanism is due to the fact that during the factorization of some local column-blocks, a processor has to update several times a block A_{ij} mapped on another processor (as illustrated on figure 3). In order to overcome this

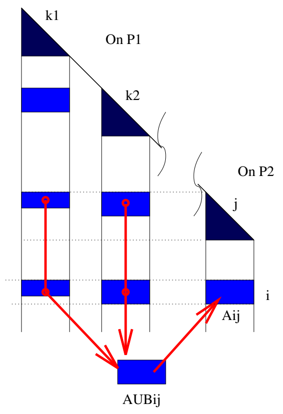

Fig. 3. Local aggregation of block updates. Column-block k_1 and k_2 are mapped on processor P_1, column-block j is mapped on processor P_2. Contributions from processor P_1 to the block A_{ij} of processor P_2 are locally summed in AUB_{ij}.

problem of aggregation, the new generation of supercomputers gives us a response. Because nowadays, the massively parallel high performance computers are generally designed as networks of SMP nodes. So, on those SMP-nodes architectures, to fully exploit shared memory advantages, a relevant approach is to use an hybrid MPI+threads implementation of our algorithm. As said previously, the local aggregation mechanism is inherent to the message passing model used in our parallel factorization algorithm. For very large matrices from 3D problems, the highest peak of memory consumption is all the more reduced of aggregated block contributions during the factorization amounts several times the local factorized matrix part on a processor. Thus, in an SMP context, the simplest way to lower the use of aggregate update blocks is to avoid message-passing communications between processors on a same SMP node. In an SMP node, a unique MPI process spawns a thread on every processor. These threads are then able to address the whole memory space of the SMP node. Each thread is therefore able to access the matrix part mapped onto its MPI process. By this way, though the computational tasks are distributed between the threads, any update of the matrix part of the MPI process is directly performed in the local matrix. The communication between MPI processes still uses the local aggregation scheme we described earlier. This implies that the amount of extra-memory required for the storage of the AUBs depends only on the number of MPI processes. Therefore, the additional memory needed to store the AUBs is all the more reduced that the SMP nodes are wider (in terms of number of processors).

5 Results

The parallel experiments were run on the TERA10 supercomputer of CEA/DAM with a network based on a Quadrics switch. All computations are performed in double precision and all time results are given in seconds. The blocking size parameter for BLAS3 computations is set to 60 and we use here a one dimensional distribution as default. NNZ represents the number of non zeroes in the matrix A, and OPC is the number of operations needed for the LDLt factorization.For each test, we use only one Gauss Seidel iteration, because we only wanted to test the performance of the new direct solver. Moreover, the global computation time for the all runs is mainly due to the factorization time of the volumic subdomains. In fact, the surfacic domains are quite small, so we can easily use Scalapack to compute the solution of the dense linear systems.

The table below shows results for a problem with N1=23 millions of volumic DOF and N2=3132 surface DOF. These results came from the MPI version of our 3D code and the MPI version of the solver PaStiX. NNZ(A)= 1.38 e+10, OPC=2.1 e+14. This is the biggest test case we could run with our full MPI code because the memory needed for this test case is about 60Gbytes per MPI task, that means for 64 MPI tasks we had to reserve 32 SMP nodes with 128 Gbytes per node (and 16 cores per SMP node). In fact we must reserve 512 cores, use only 64 of them, just because of the memory consumption...Concerning the preprocessing phase, because we use for the moment only a sequential version, it takes about 45 minutes and more than 4 Gigabytes of RAM.

Step	C.P.U. (sec)
Assembly	900
Factorization	8700
Solution	40

The table below shows computational time of the factorization (because this operation is the most C.P.U. time consumming for individual operation) for the same 3D object, but this time with the MPI+threads version of the solver PaStiX. We give the results with 16 threads per MPI task (our SMP nodes have 16 cores), and the time for preprocessing is the same than the MPI one.

Number of Tasks	C.P.U.	Max Memory (Gbytes)
(MPI/Threads)	(sec.)	per SMP node
512 (32/16)	1461	29

The table below shows computational time of the factorization for a 3D object with N1=45 millions of volumic DOF (only one subdomain in the volume), NNZ(A)=2.59e+10, OPC=1.04e+15. The time needed to compute the preprocessing is about one hour and more than 8 Gbytes of RAM.

Number of Tasks	C.P.U.	Max Memory (Gbytes)
(MPI/Threads)	(sec)	per SMP node
512 (32/16)	6126	49
1024 (64/16)	3054	32
2048 (128/16)	2018	35

The table below shows computational time of the factorization for a 3D object with N1=83 millions of volumic DOF (only 1 subdomain in the volume), NNZ(A)=5.97e+10, OPC=4.28e+15. Concerning this case, in the future we will need to use the PTScotch software instead of Scotch because for the moment, when we use Scotch it takes more than 2 hours and more than 16Gbytes of RAM to compute the ordering. The first tests with PTScotch are very encouraging and we are working on the integration of PTScotch in our industrial software.

Number of Tasks	C.P.U.	Max Memory (Gbytes)
(MPI/Threads)	(sec.)	per SMP node
768 (48/16)	27750	115

In the last three test cases, the memory overhead is drastically reduced thanks to the hybrid MPI+threads implementation. In addition, we observe a significant decreasing of the factorization time and a better global scalability of the parallel solver.

6 Conclusion and Prospects

This paper presents promising results on the parallel solution of large sparse linear systems by direct methods and so, on the parallel solution of RCS for

3D bodies. The arrival of massively parallel supercomputers with tens thousands of processors gives the possibility to solve larger linear systems (tens or hundreds millions of DOF). The work regarding all the stages combined in our new simulation code in electromagnetism is still in progress. Especially, thanks to the powerful SMP version of the solver PaStiX, we are currently developping a complete MPI+threads version of the code to bypass our problems with memory. It is important to note that we have validated with success all the physics contained, thanks to comparisons with other codes (in particular 2D codes) and measurements. Moreover, relying on the direct solver precision and robustness, a whole new method based on full multigrid is under development. Unlike classical iterative solvers, the multigrid setup phase uses a coarse solution given by direct solver as an entry point. Refining volumic subdomain and using previous coarse solution, allows us to compute a new linear system whose fine solution is near the interpolated coarse one. Simple iterative methods such as Jacobi can reduce the error, leading to an accurate fine solution at minimal cost. Moreover, the Jacobi method got two main advantages: it is intrinsically parallel (reusing direct solver unknowns distribution) and can be performed without assembling the whole system.