

# Parallel Distributed Numerical Simulations in Aeronautic Applications

G. Alléon<sup>\*</sup>   S. Champagneux<sup>†‡</sup>   G. Chevalier<sup>†</sup>   L. Giraud<sup>†</sup>   G. Sylvand<sup>\*</sup>

TR/CFD-PA/05/44

## Abstract

The numerical simulation plays a key role in industrial design because it enables to reduce the time and the cost to develop new products. Because of the international competition, it is important to have a complete chain of simulation tools to perform efficiently some virtual prototyping. In this paper, we describe two components of large aeronautic numerical simulation chains that are extremely consuming of computer resource. The first is involved in computational fluid dynamics for aerodynamic studies. The second is used to study the wave propagation phenomena and is involved in acoustics. Because those softwares are used to analyze large and complex case studies in a limited amount of time, they are implemented on parallel distributed computers. We describe the physical problems addressed by these codes, the main characteristics of their implementation. For the sake of re-usability and interoperability, these softwares are developed using object-oriented technologies. We illustrate their parallel performance on clusters of symmetric multiprocessors. Finally, we discuss some challenges for the future generations of parallel distributed numerical software that will have to enable the simulation of multi-physic phenomena in the context of virtual organizations also known as the extended enterprise.

## 1 Introduction

The development of a complete chain of numerical simulation tools to predict the behaviour of complex devices is crucial in an industrial framework. Nowadays the numerical simulation is fully integrated in the design processes of the aeronautics industry. During the last decades, robust models and numerical schemes have emerged and have been implemented in simulation codes to study the various physical phenomena involved in the design of aeronautic applications. The simulations tools enable the engineers to perform virtual prototyping of new products and to study their behaviours in various contexts before a first real prototype is built. Such a tool permits the reduction of the time to market and the cost to design a new product. A typical simulation platform is composed by three main components that are

1. the preprocessing facilities that include the CAD systems and the mesh generators,
2. the numerical simulation softwares, that implement the selected mathematical models and the associated numerical solution techniques,
3. the post-processing facilities, that enable the visualization of the results of the numerical simulations. This latter step is often performed using virtual reality capabilities.

---

<sup>\*</sup>EADS CRC, Centre de Toulouse, Centre 1, 4, Avenue Didier Daurat, 31700 Blagnac, France

<sup>†</sup>CERFACS, 42 Avenue G. Coriolis, 31057 Toulouse Cedex, France

<sup>‡</sup>Current address: Airbus, 316 route de Bayonne, 31060 Toulouse, France

Each of these main components can operate with each other. It is necessary to go back and forth between them when the characteristics of the studied devices change. This occurs for instance when numerical optimization is performed to maximize some governing parameters of the new product. Even though each of these three components intensively use computing resource, we focus in this paper on the second one that plays a central role. It involves various ingredients ranging from applied physics and applied mathematics to computer science. In this paper, we concentrate on two applications in aeronautics that intensively use large computing resources. These applications are the computational fluid dynamics and the computational acoustics. They both require the solution of partial differential equations (PDE) defined in large domains and discretized using fine meshes. Because those calculations are one step in a larger production process, they should be performed on high performance computers so that they do not slow-down the overall process. In the recent years, these high performance computers are mainly clusters of symmetric multiprocessors (SMP) that have demonstrated to be cost effective. The parallel approach in that context consists in splitting the complete mesh into sub-meshes so that each processor is in charge of one sub-mesh. The exchanges between the computing nodes are implemented using a message passing library that in general is MPI. Finally, for the sake of re-usability and interoperability, these simulations codes are developed using object-oriented technologies and languages like C++. However, some low level computationally intensive calculations are still coded in Fortran.

The paper is organized as follows. In Section 2 we describe the code for the aerodynamic applications and its parallel implementation. The existing code was initially developed and used on vector supercomputers mainly in serial processing. The present study is an analysis and adaptation of the code for SMP platforms with cache based memory architecture. Its parallel performance is studied on real life problems using cluster of SMP including HP-Compaq Alphaserver, IBM SP machine and the new Cray XD-1 computer. In Section 3, we present a code used to perform simulation in another important field of the aeronautics , namely the wave propagation studies. In that section we briefly present the features of the code and show numerical experiments on industrial geometries. The parallel performance of this code is illustrated on a cluster of SMPs developed by Fujitsu on top of bi-processor Opteron node. We conclude with some remarks on the future challenges for the parallel distributed numerical softwares that should enable the simulation of multi-physic phenomena in the context of virtual organizations. This latter framework is also known as the extended enterprise.

## 2 The aerodynamic simulations

We consider the numerical simulation of external compressible flows that range from the unbounded evolution of wave vortex generated by the aircrafts, to the flow around vehicles or objects like aircrafts (see Figure 1), helicopters, missiles, etc. We also address the study of the internal flows in complex industrial devices like piston engines, compressor, turbines, etc. The power increase of modern computers enables either the solution of larger problems, or the improvement of the accuracy (model, details) of present industrial problems, or the use of CFD for optimization process. Moreover the industry wants to integrate earlier in the design process the multi-disciplinary aspects such that fluid/structure, flight mechanics, acoustic, aerothermal, etc. This multi-disciplinary scale increases the complexity of the simulation codes.

The numerical simulation of these aerodynamic phenomena is nowadays considered as an elementary building block of the industrial conception process and should therefore be efficient, flexible and easy to integrate in the computational chain.

### 2.1 Physical Modelling

The aerodynamic problems, we aim at simulating, are described by the convection/diffusion model of Navier-Stokes that is valid in continuous regime. The Navier-Stokes equations, in their conser-

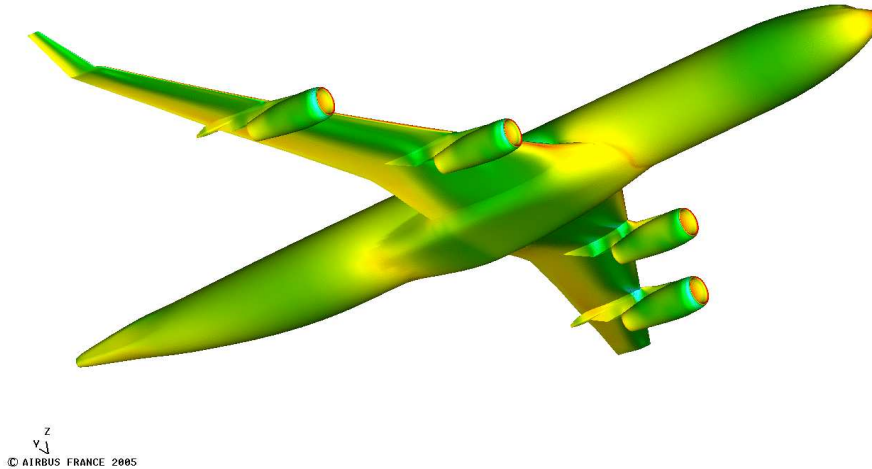


Figure 1: Full aircraft Navier-Stokes simulation

vative form, can be written

$$\frac{\partial W}{\partial t} + \text{div}(\mathcal{F}) = 0 \quad (1)$$

where  $W$  is the state vector  $(\rho, \rho u, \rho v, \rho w, \rho E)$  with  $\rho$  the density,  $(\rho u, \rho v, \rho w)$  the momentums and  $\rho E$  the internal energy.  $\mathcal{F}$  is the flux tensor and corresponds to the difference of the convective fluxes and the viscous fluxes. This time-dependent system of nonlinear PDEs enables us to guaranty the conservation of mass, momentum and energy. This system is closed via compartment laws. The fluid is Newtonian and complies with the Stokes assumptions for which the heat flux is given by the Fourier law. Finally this system of PDEs is defined within a finite space and time domain and boundaries conditions are required both in time and in space.

One of the major difficulties is to properly simulate turbulent flows that are involved in all industrial calculations. Various models can be considered to tackle this problem. The choice is mainly governed by the complexity of the application and by the time constraints to produce the results using affordable computing resources. Various possibilities can be considered. The Direct Numerical Simulation (DNS), that does not rely on any model, is extremely CPU consuming. The Reynolds Average Navier-Stokes (RANS) equations are another possibility where additional PDEs are added to the system to model the impact of the turbulence. Some intermediate techniques exist that are the Large Eddy Simulation (LES) or Detached Eddy Simulation (DES). Even though the computer performance has significantly increased in the last decades, the RANS approach is still the only affordable solution that can be usually considered in operational industrial codes.

The numerical techniques described in the next sections have been implemented in the package elsA [3, 10] using object oriented technology. Started in 1997 by ONERA, this package is currently jointly developed with CERFACS that joined the project in 2001 and with DLR that joined the collaboration in 2004. The numerical experiments of this paper related to CFD applications have been performed with this package.

## 2.2 Numerical modelling

### 2.2.1 Discretization schemes

In a finite volume framework, the problem (1) writes:

$$\int_{\Omega} \frac{\partial W}{\partial t} dV + \oint_{\partial\Omega} \mathcal{F} \cdot \nu ds = 0, \quad (2)$$

where  $\nu$  is the outer normal. When 3D spatial discretization is introduced, Equation (2) becomes:

$$\frac{d}{dt} (V_{i,j,k} W_{i,j,k}) + R_{i,j,k} = S_{i,j,k}, \quad (3)$$

where  $V_{i,j,k}$  is the cell volume and  $S_{i,j,k}$  the source term of the cell  $i, j, k$  in the mesh. Once written in terms of flux balance, the residual becomes

$$R_{i,j,k} = F_{i+1/2,j,k} - F_{i-1/2,j,k} + F_{i,j+1/2,k} - F_{i,j-1/2,k} + F_{i,j,k+1/2} - F_{i,j,k-1/2}. \quad (4)$$

In elsA there are several choices for the spatial discretization. However, in this study we have only used the Jameson scheme based on a centered approximation of the fluxes. For the sake of stability, a numerical dissipation is added:

$$F_{i+1/2} = \mathcal{F}_{i+1/2} \cdot s_{i+1/2} - d_{i+1/2}. \quad (5)$$

When looking for a steady solution, the system is solved using an implicit iterative scheme with pseudo-time stepping procedure, derived from the unsteady formulation. The simplest approximation of the time derivative (Backward Euler formulation) used in Equation (3) gives:

$$V \frac{W^{n+1} - W^n}{\Delta t} + \theta R(W^{n+1}) + (1 - \theta) R(W^n) = 0. \quad (6)$$

A linearization of the residual at time  $n + 1$  is written:

$$R^{n+1} = R^n + \frac{\partial R}{\partial W} \Delta W + \mathcal{O}(\Delta t^2),$$

that, combined with Equation (6) gives,

$$\left( \frac{V}{\Delta t} I + \theta \frac{\partial R}{\partial W} \right) \Delta W = -R(W^n). \quad (7)$$

The difficult part in Equation (7), is to get a good estimate for the Jacobian matrix  $\left( \frac{\partial R}{\partial W} \right)$ . If we apply the ‘‘First Order Steger and Warming Flux Vector Splitting’’ [25] method, we end up with a diagonal dominant linear system with numerical fluxes that writes

$$F_{i+1/2} = f^+(W_i) + f^-(W_{i+1}). \quad (8)$$

The linearization of the fluxes at time  $n$  reads:

$$F_{i+1/2}^{n+1} = F_{i+1/2}^n + A^+(W_i, s_{i+1/2}) \Delta W_i + A^-(W_{i+1}, s_{i+1/2}) \Delta W_{i+1},$$

where  $A$  is the Jacobian matrix of the fluxes, written as  $A^\pm = P \Lambda^\pm P^{-1}$  where  $\Lambda^\pm$  is a diagonal matrix composed of the positive (resp. negative) eigenvalues of  $A$ .

In elsA, there are several techniques to estimate the matrix  $A^\pm$ , but the scalar approximation is commonly used. It consists in the use of the spectral radius of  $A$ , denoted by  $r(A)$ , instead of

the matrix, and to make a direct estimation of  $A^\pm \Delta W$  to avoid the explicit building of the  $A^\pm$  matrix. That is,

$$A^\pm \Delta W \approx \frac{1}{2} (\Delta F \pm r(A) \Delta W)$$

where  $\Delta F$  is an approximation of the convective flux variation. This same type of linearization is used for the boundary conditions computation during the solution.

In elsA the linear system is solved using a LU-SSOR (for Lower/Upper Symmetric Successive Over Relaxation) method [28] based on a decomposition in lower, diagonal and upper triangular parts of the matrix:

$$(\mathcal{L} + \mathcal{D} + \mathcal{U}) \Delta W = -R.$$

The system is solved by  $p$  relaxation iterations, each of them consists of two steps:

$$\begin{aligned} (\mathcal{L} + \mathcal{D}) \Delta W^{p+1/2} &= -R - \mathcal{U} \Delta W^p, \\ (\mathcal{U} + \mathcal{D}) \Delta W^{p+1} &= -R - \mathcal{L} \Delta W^{p+1/2}. \end{aligned}$$

One should notice that one of the interest of the LU-SSOR method is that it allows a limited implicit coupling between the computational domains.

### 2.2.2 Numerical solution techniques

In the package we are interested in, the physical domain is discretized using a mesh composed by a set of structured blocks referred to as a structured multi-block mesh. The multi-block approach

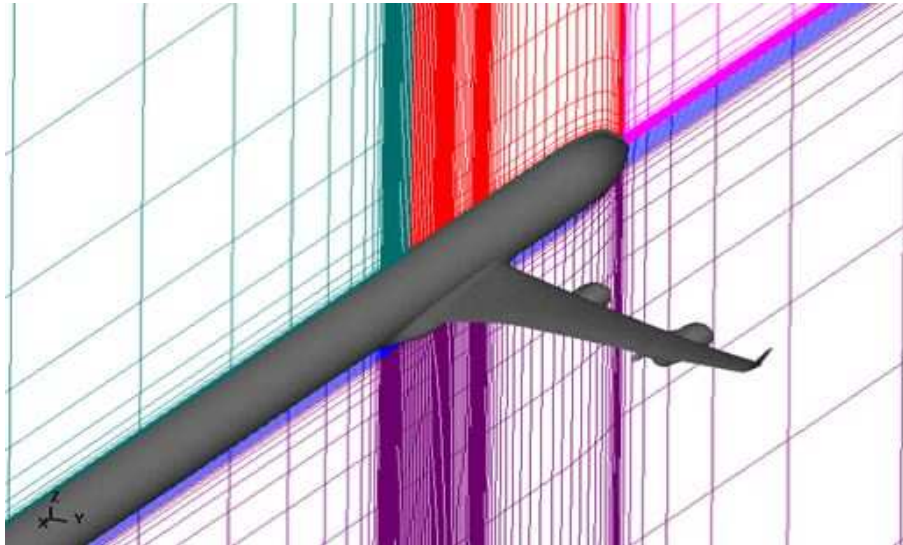


Figure 2: Multi-block mesh on a large aircraft

enables the padding of the complete domain with elementary sub-blocks. Each of these sub-blocks is topologically equivalent to a cube, which simplifies significantly the underlying data structure handled by the code. No indirect addressing is used in the inner-most loops of the code and the memory accesses are very regular. Consequently the code runs efficiently. Furthermore, this approach is well suited to capture the strong anisotropies that exist in the physical solutions that are computed. Such anisotropies can be observed in the boundary layers close to the walls, in the shear layers and in the shocks for examples. In Figure 2 we depict an example of a multi-block mesh around an aircraft.

The connectivity between the elementary blocks could be arbitrary in the volume (mesh overset techniques, chimera) or on a surface, with possibly no direct matching of the mesh cells across the surface. This strategy of meshing is rather simple and can be automated easily. Its flexibility and simplicity are real assets in the conception and optimization phase involved in the design process. It is also very convenient for some multi-physics calculations such that in aeroelasticity where it is important to adapt the mesh to the shape of an object that evolves in time.

The Navier-Stokes equations are discretized in space using a finite-volume technique, then boundary conditions are set on the multi-block meshes boundaries. The ODE problem is then iteratively solved in time (or pseudo time). The convection operator, stabilized by an artificial dissipation term [12], uses a special formulation in order to enable the resulting scheme to simulate either transonic flows with discontinuities (shock wave) or incompressible flows. Implicit schemes using geometric multigrid solvers are implemented to efficiently solve the problems associated with steady or unsteady flows. This nonlinear geometric multigrid technique enables ones to obtain simulation times that comply with computational fluid dynamic calculation in an operational framework.

The large scale problems that are currently solved within various Airbus business units can be addressed thanks to the parallel distributed capabilities of elsA to be described in the next sections.

## 2.3 The scalar parallel performance

In a high performance computing framework, a preliminary step before to go for parallel computing is a basic code tuning. This step aims at getting a good megaflops rate out of a single processor execution; the gain is basically multiplied by the number of processors in a parallel environment.

In this section, we illustrate various possible techniques experimented in the elsA code to take the best advantage of the computing power available within a node of a SMP. The elsA code has been designed and run for many years on high performance computers that were vector processors based platforms. With the advent of modern microprocessors the gap in performance between the vector and “scalar” processors has shrunk. Clusters of SMPs have spread in industry and the large calculations are currently performed indifferently on parallel vector computers or on clusters of SMPs. From a software design point of view this means that the code should be efficient on both platforms. For a maintenance viewpoint only one source code must be maintained with as little as possible differences between the final codes that will eventually be compiled on the target machine.

In the first part, we illustrate how the leading dimension of the 2-dimensional arrays handled by the code should be defined in order to avoid poor performance related to a bad usage of the memory hierarchy (various level of cache). We then investigate the possibility to use OpenMP to implement a fine grain parallelism. Finally, we report on parallel distributed experiments performed on large problems.

### 2.3.1 Single node performance

Because the elsA package was preliminary developed for vector machines, its basic data structure is designed for this type of target platforms. In particular, the dimension of the mesh is the leading dimension of all the arrays in memory. It is well known that such data structure is not well adapted for scalar processors. In this study, a major effort was focused on simple techniques to preserve the efficiency of the code on platforms based on SMP clusters. On these scalar computers reusing the data loaded in the high levels of the memory hierarchy (level 1, level 2 or level 3 cache or physical memory) is the main concern when trying to ensure high sustained megaflops rate of the computation. Based on the knowledge of the main features of the memory hierarchy (number of level of cache, size of the cache, cache management policy, TLB...) it is possible to get rid of some very poor performance observed for some sizes of the blocks.

This poor behaviour is illustrated in Figure 3 (a) where we report on the megaflops rate to perform the computation on a cube with a varying mesh size. It can be observed that this rate can drastically drop down for some grid sizes. A loss of performance up to a factor of 5 can often be observed. It can also be seen in Figure 3 (a) that the location of the peaks is periodic. They correspond to the situation where the limited associativity of either the cache or the Translation Lookaside Buffer (TLB) becomes the main bottleneck. These performance drops are localized

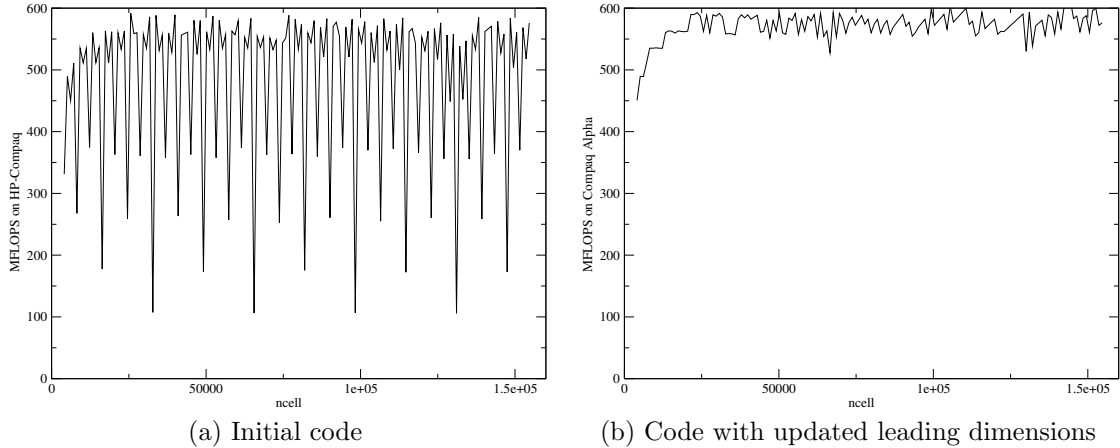


Figure 3: Megaflops rate of elsA on HP-Compaq varying the size of the cube

around data set size multiple of some power of two, depending on the memory architecture (size of the caches and associativity). But it is far from fortuitous, because elsA, like a number of structured CFD codes, uses a convergence acceleration technique based on multigrid. The algorithm imposes that the computational meshes have dimensions in multiple of 8 (for example). So the odds are strong that the dimensions of the arrays are right on the wrong dimensions for the memory throughput.

The first step of the optimization that attempts to remove these peaks consists in identifying what level of the memory hierarchy is responsible for that. The methodology consists in using a probing code that mimics the most often occurring memory access pattern. Most of the computation is performed in low level routines written in Fortran. These routines mainly access 2-dimensional arrays, where the leading dimensions are the number of cells in the mesh. The number of columns is the number of variables associated with each cell and depends on the type of simulation. This storage and memory access are known to be the most efficient one for vector computing. Using this probing technique we were able to identify that the bottleneck was the L1 cache associativity for the Alpha processor on the HP-Compaq. For the IBM machine the same associativity problem was related to his TLB entries. After this step, we have changed the size of the dynamically allocated arrays so that we avoid the memory contention. The change in the size of the leading dimension is computed by a simple function that depends on the number of memory arrays, their size, and the architecture/associativity of the cache or TLB. This idea is very similar to the one used on vector machines to avoid memory bank conflicts (due to the high order interleaving of the memory). This technique has the main advantage of only requiring a very minor change in the code. The gain introduced by these modifications are clearly shown in Figure 3(b). It can be seen that the megaflops rate is now nearly independent of the number of cells in the mesh.

### 2.3.2 The parallel shared memory (SMP) performance: OpenMP

Some of the numerical schemes implemented in elsA were selected for their ability to fully take advantage of the vectorization. Many vectorization directives are inserted to help the compilers and we wondered if we could simply replace them by some OpenMP directives. We first focused our attention on one of the main time consuming routine that implements a relaxation scheme [28] either to solve the Jacobian or to be used as the smoother in a nonlinear multigrid method [12]. The simple replacement of vectorization directives by OpenMP [18] ones gave very disappointing performance. In particular, rather poor speed-ups were observed on large meshes and speed-downs were even observed on the small meshes. This is far to be optimal in the framework of multigrid because going down in the grid hierarchy quickly leads to small grids. To get good speed-ups even on small grids we have implemented a 1D block partition of the grid. The relaxation scheme implements a pipelined technique between blocks, each block being handled by a thread. The speed-up performances of the multi-threaded code are reported in Figure 4. It can be seen that an

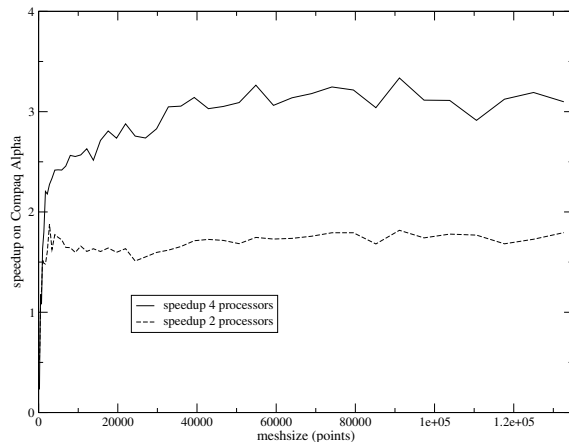


Figure 4: Speed-ups on a 4-processor HP-Compaq SMP node

asymptotic speed-up close to 1.7 is quickly observed when two threads are used. For 4 processors the asymptotic behavior is observed for slightly larger mesh sizes that are still satisfactory for a usage in a multigrid framework. Unfortunately, the resulting code differs quite deeply from the initial code for vector processors. Consequently, this fine grain - OpenMP - parallelism has not been selected for implementation in the official source tree of the code.

### 2.3.3 The parallel distributed performance

In a parallel distributed the strategy consists in splitting the set of blocks into subset of similar number of nodes (e.g. same computational weight) while trying to minimize communication between subsets. In order to do so, we first built a weighted graph  $\mathcal{G} = \{V, E\}$ . Each vertex of  $V$  corresponds to one block of the multi-block mesh, the value at this vertex is defined by its size measured by its number of cells. There is an edge  $E_{i,j}$  between  $v_i$  and  $v_j$  if the blocks  $i$  and  $j$  are connected in the multi-block mesh (i.e. share a face or an edge), the weight of this edge is defined by the number of cells at the interface between the two blocks. The graph  $\mathcal{G}$  is then split using the METIS [13] software which provides us with a well-balanced partition of the set of blocks with small interfaces between the blocks.

In Figure 5 we depict the observed speed-up on a complete calculation on a 3D industrial civil aircraft form an European company. For that calculation the number of cells is about 9 millions and the computing platform is a cluster of bi-processor Opteron connected through a Gigabit Ethernet network. Up to eight processors the speed-up is almost linear and slightly deteriorates

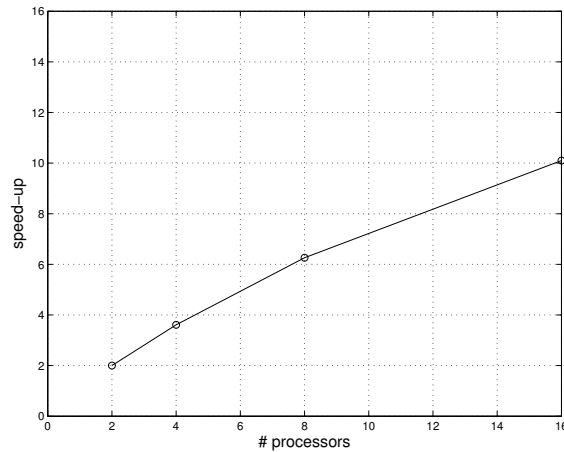


Figure 5: Speed-ups on a full aircraft calculation using 9 millions of cells

on 16 processors. In Figure 6 we display the parallel performance of the code on the ONERA M6 Wing test problem. These performances have been observed on a Cray XD1 computer. For the two problems and two different computing platforms, the performance of the code are comparable with slightly better speed-ups on the XD1 probably due to the efficiency of the interconnection network. We mention that on the typical computing platform used for the simulations involved in

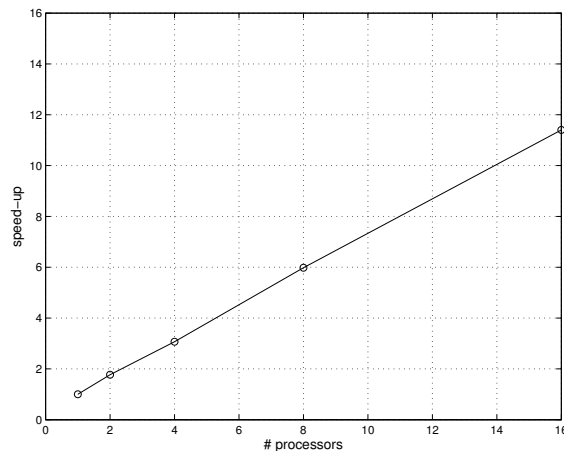


Figure 6: Speed-ups on the ONERA M6 Wing using 670 000 cells

a design procedure, the number of processors usually used does not exceed 16. Nevertheless the scalability of the code could look disappointing, but it should be noticed that the functionalities (CFD but also mesh deformation, multi-physics, etc..) of the code are now all parallel. Moreover, a rewriting of the communications and some changes in the algorithm are under way in order to reduce the number of communications, and gather them to improve the performance.

Overall the performances of the code in terms of CPU throughput and parallel capacities allows already an industrial use of the code on SMP architectures. The work in progress aims at improving the scalability for medium range parallelism (64-128 nodes) for unsteady applications. However, it is worth mentioning that robustness and inter-operability are key points. Consequently some parallel optimizations are discarded because they might corrupt them.

### 3 The wave propagation simulations

We are now interested in another aspect of the simulations in aeronautic applications, which is the acoustic wave propagation problem. We are interested in the simulation of the sound propagation around complex structure (such as planes) in order to predict the noise and/or vibrations induced by the shape of the objects and the positioning of the various sources (engines for instance). In this context, the fluid is supposed to be perfect, homogeneous and still. Although no results are reported in this paper, some recent developments enable the simulations with uniform flows.

#### 3.1 The governing equations

##### 3.1.1 Helmholtz equation

In this section, we introduce the integral equations used in acoustic in our study. We are interested in the solution of the Helmholtz equations in the frequency domain using an integral equation formulation. We consider the case of an object  $\Omega^-$  illuminated by an incident plane wave  $u_{inc}$  of pulsation  $\omega$  and wave number  $k = \omega/c$  where  $c$  is the wave celerity. The boundary  $\Gamma$  of the object has a rigid part  $\Gamma_r$  and a treated part  $\Gamma_t$  of impedance  $\eta$ . The outer normal is  $\nu$ .

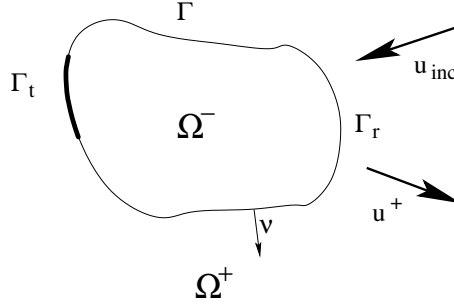


Figure 7: Acoustic problem

We want to compute the diffracted field  $u^+$  in the outer domain  $\Omega^+$ . It is solution of the problem  $(P^+)$ :

$$(P^+) \begin{cases} \Delta u^+ + k^2 u^+ = 0 & \text{in } \Omega^+, \\ \frac{\partial u^+}{\partial \nu} = -\frac{\partial u_{inc}}{\partial \nu} & \text{on } \Gamma_r, \\ \frac{\partial u^+}{\partial \nu} + i\frac{k}{\eta} u^+ = -\left( \frac{\partial u_{inc}}{\partial \nu} + i\frac{k}{\eta} u_{inc} \right) & \text{on } \Gamma_t, \\ \lim_{r \rightarrow +\infty} r \left( \frac{\partial u^+}{\partial r} - iku^+ \right) = 0. \end{cases}$$

##### 3.1.2 Integral formulation

We now denote  $\phi$  and  $p$  the jumps of the tangential parts of  $u$  and  $\partial u/\partial \nu$  on  $\Gamma$ . The knowledge of  $p$  and  $\phi$  entirely solves the problem, since the following representation theorem allows then the computation of the diffracted pressure  $u$  in any point  $x$  (see [17]):

$$u(x) = \int_{\Gamma} \left( G(x, y) p(y) - \frac{\partial G(x, y)}{\partial \nu_y} \phi(y) \right) dy, \quad (9)$$

where  $G(x, y)$  is the Green's function, solution of  $\Delta u + k^2 u = -\delta_0$ , that is

$$G(x, y) = \frac{e^{ik\|x-y\|}}{4\pi\|x-y\|}.$$

In the case of a rigid body, all the terms involving  $\Gamma_t$ ,  $\eta$ ,  $\lambda^t$  and  $\lambda$  disappear. We then obtain the variational formulation:

$$\begin{cases} \text{find } \phi \text{ such that } \forall \phi^t, \text{ we have} \\ -\frac{1}{ik} \int_{\Gamma \times \Gamma} \frac{\partial^2 G}{\partial \nu_x \partial \nu_y} \phi(y) \phi^t(x) dy dx = -\frac{1}{ik} \int_{\Gamma} \frac{\partial u_{inc}(x)}{\partial \nu} \phi^t(x) dx. \end{cases} \quad (10)$$

### 3.1.3 Discretization

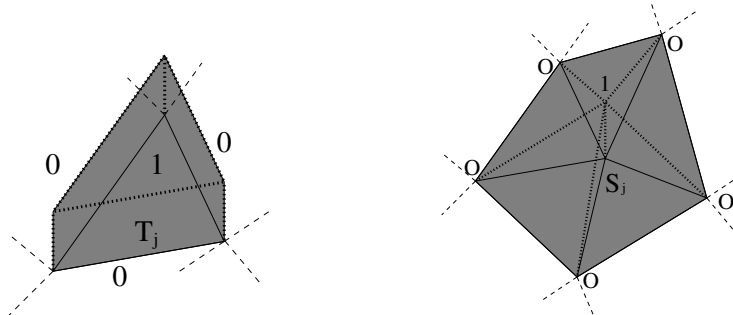


Figure 8: P0 (left) and P1 (right) basis functions

In order to discretize this system, we use a surfacic triangle mesh of the boundary  $\Gamma$ . The pressure jump  $\phi$  is discretized using P1 linear basis functions, the pressure normal derivative jump  $\lambda$  is discretized using P0 basis functions (see Figure 8). We end up with a complex, dense, symmetric linear system to solve.

## 3.2 The numerical solution schemes

### 3.2.1 The parallel Fast Multipole calculation

The integral approach used here to solve the Helmholtz equation has several advantages over a classic surfacic formulation. First, it requires to mesh only the surface of the objects, not the propagation media inside and around it. In an industrial context, where the mesh preparation is a very time consuming step, this is crucial. Second, the radiation condition at infinity is treated naturally by the formulation in an exact manner. At last, the surfacic mesh allows us to have a very accurate description of the object's shape, which is not the case with finite difference methods for instance.

The main drawback of integral formulation is that it leads to solve dense linear systems difficult to tackle with iterative solvers. Even though the spectral condition number is usually not very high, the eigenvalues distribution of these matrices is not favorable to fast convergence of unpreconditioned Krylov solvers [1]. Furthermore for large objects and/or large frequencies, the number of unknowns can easily reach several millions. In that case, the classic iterative and direct solvers are unable to solve our problem, because they become very expensive in CPU time and storage requirements.

The Fast Multipole Method (FMM) is a new way to compute fast but approximate matrix-vector products. In conjunction with any iterative solver, it is an efficient way to overcome these

limitations ([21, 23, 24]). It is fast in the sense that CPU time is  $\mathcal{O}(n \cdot \log(n))$  instead of  $\mathcal{O}(n^2)$  for standard matrix-vector products, and approximate in the sense that there is a relative error between the “old” and the “new” matrix-vector products  $\varepsilon \approx 10^{-3}$ . Nevertheless, this error is not a problem since it is usually below the target accuracy requested to the iterative solver or below the error introduced by the surfacic triangle approximation.

We give now an overview of the single level multipole algorithm. We first split the surface  $\Gamma$  into equally sized domains using for instance a cubic grid (as shown in Figure 9). The degrees of freedom are then dispatched between these cubic cells. Interaction of basis functions located in neighbouring cells (that is, cells that share at least one vertex) are treated classically, without any specific acceleration.

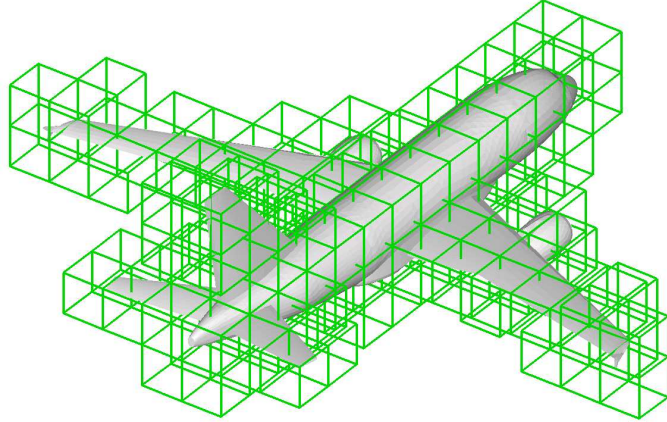


Figure 9: Use of a cubic grid to split the mesh

The calculation of the interactions between unknowns located in non-neighbouring cells are accelerated with the FMM. Basically, the idea is to compute the far field radiated by the unknowns located in each box, and to compute interaction between boxes using these far field data. By doing so for all the couples of non-neighbouring boxes, and by treating classically the interaction between neighbouring boxes, we can compute the entire matrix vector product. This algorithm is very technical, and has been extensively explained in the literature ([8, 9, 26]). The optimal complexity of the single level FMM is  $\mathcal{O}(n_{dof}^{3/2})$  (where  $n_{dof}$  is the number of degrees of freedom) against  $\mathcal{O}(n_{dof}^2)$  for a standard matrix vector product.

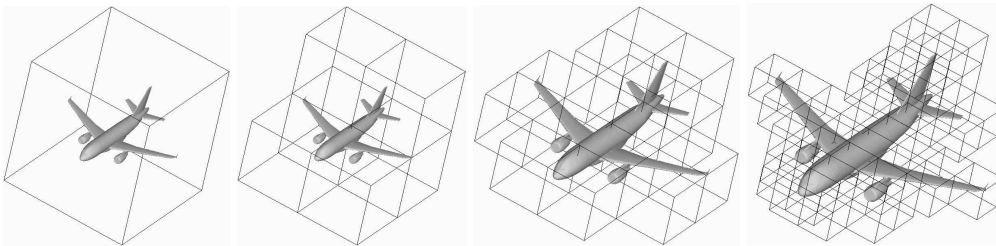


Figure 10: Subdivision of an aircraft through an octree

The multi-level fast multipole algorithm uses a recursive subdivision of the diffracting object as shown in Figure 10. The resulting tree is called an *octree*. The idea is to explore this tree from the root (largest box) to the leaves, and at each level to use the single level multipole method to treat interactions between non-neighbouring domains that have not yet been treated at an upper

level. At the finest level, we treat classically the interactions between neighbouring domains. The optimal complexity of the multi-level FMM is  $\mathcal{O}(n_{dof} \cdot \log(n_{dof}))$ . All the results given in this paper use the multi-level FMM.

In order to widely exploit the available parallel architectures, we have developed a parallel distributed memory implementation of the FMM algorithm based on the message passing paradigm and the MPI library. The distribution of the octree is made level by level, each of them is equally distributed among the processors. When a given cell  $\mathcal{C}$  is affected to a processor, all the computations required to build the functions for this cell are performed exclusively on this processor. Each phase of the calculation (initializations, ascents or descents between two levels, transfers at a given level, integrations) is preceded by a communication phase. Therefore a complete multipole calculation on a parallel machine consists of an alternance of computation steps and communication steps. This requires a good load balancing at each level of the octree.

### 3.2.2 The parallel iterative linear solvers

**The iterative solvers:** we focus now on the solution of the linear system

$$Ax = b$$

associated with the discretization of the wave propagation problem under consideration. As mentioned earlier, direct dense methods based on Gaussian elimination quickly becomes unpractical when the size of the problem increases. Iterative Krylov methods are a promising alternative in particular if we have fast matrix-vector multiplications and robust preconditioners. For the solution of large linear systems arising in wave propagation, we found that GMRES [19] was fairly efficient [6, 26].

Most of the linear algebra kernels involved in the algorithms of this section (sum of vectors, dot products calculation) are straightforward to implement in a parallel distributed memory environment. The only two kernels that require to pay attention are the matrix-vector product and the preconditioning. The matrix vector product is performed with the parallel FMM implementation described in Section 3.2.1. The preconditioner is described in the next section, its design was constrained by the objectives that its construction and its application must be easily parallelizable.

**The preconditioner:** the design of robust preconditioners for boundary integral equations can be challenging. Simple parallel preconditioners like the diagonal of  $A$ , diagonal blocks, or a band can be effective only when the coefficient matrix has some degree of diagonal dominance depending on the integral formulation [22]. Incomplete factorizations have been successfully used on nonsymmetric dense systems [20] and hybrid integral formulations [15], but on the EFIE the triangular factors computed by the factorization are often very ill-conditioned due to the indefiniteness of  $A$ . This makes the triangular solves highly unstable and the preconditioner ineffective.

Approximate inverse methods are generally less prone to instabilities on indefinite systems, and several preconditioners of this type have been proposed in electromagnetism (see for instance [1, 4, 5, 7, 27]) and their efficiency in acoustics has been shown in [11]. Owing to the rapid decay of the discrete Green's function, the location of the large entries in the inverse matrix exhibit some structure [1]. In addition, only a very small number of its entries have relatively large magnitude. This means that a very sparse matrix is likely to retain the most relevant contributions of the exact inverse. This remarkable property can be effectively exploited in the design of a robust approximate inverse.

The original idea of an approximate inverse preconditioner based on Frobenius-norm minimization is to compute the sparse approximate inverse as the matrix  $M$  which minimizes  $\|I - AM\|_F$  subject to certain sparsity constraints. The Frobenius norm is chosen since it allows the decoupling of the constrained minimization problems into  $n_{dof}$  independent linear least-squares problems, one for each column of  $M$ , when preconditioning from the right. Hence, there is considerable scope for parallelism in this approach.

For choosing the sparsity pattern, the idea is to keep  $M$  reasonably sparse while trying to capture the “large” entries of the inverse, which are expected to contribute the most to the quality of the preconditioner. On boundary integral equations the discrete Green’s function decays rapidly far from the diagonal, and the inverse of  $A$  may have a very similar structure to that of  $A$  [1]. The discrete Green’s function can be considered as a column of the exact inverse defined on the physical computational grid. In this case a good pattern for the preconditioner can be computed in advance using graph information from  $\tilde{A}$ , a sparse approximation of the coefficient matrix constructed by dropping all the entries lower than a prescribed global threshold [1, 5, 14]. When fast methods are used for the matrix-vector products, all the entries of  $A$  are not available and the pattern can be formed by exploiting the near-field part of the matrix that is explicitly computed and available in the FMM. In that context, relevant information for the construction of the pattern of  $M$  can be extracted from the octree.

In our implementation we use two different octrees, and thus two different partitionings, to assemble the preconditioner and to compute the matrix-vector product via the FMM. The size of the smallest boxes in the partitioning associated with the preconditioner is a user-defined parameter that can be tuned to control the number of nonzeros computed per column. The parallel implementation for building the preconditioner consists in assigning disjoint subsets of leaf boxes to different processors and performing the least-squares solutions, required by the Frobenius norm minimization, independently on each processor. At each step of the Krylov solver, applying the preconditioner is easily parallelizable as it reduces to a regular matrix-vector product. This preconditioner is referred to as SPAI in the rest of this paper.

### 3.3 Parallel performance

We show here an example of computation realized with a realistic object (a rigid wing portion carrying almost 6 millions of unknowns) and a simplified illumination (a plane wave at 3150 Hz). Basically, we would obtain the same computational performance no matter what the right hand side of the system is. These computations were performed on a cluster based on Opteron processors (at 2.4 GHz), SCSI disks and a Gigabit Ethernet interconnection.

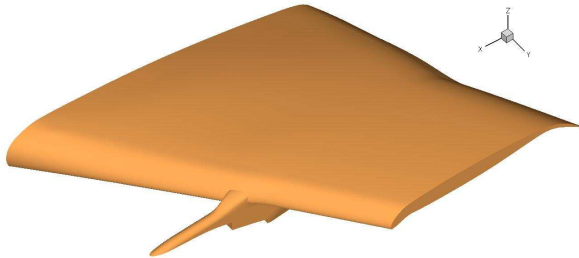


Figure 11: Mesh of a rigid wing portion

The solution of the initial system was obtained in 9 hours on 32 processors, using CFIE formulation plus SPAI preconditioner. The CFIE formulation is a variant of the one detailed above, which has the ability to converge much faster but works only on rigid bodies. The matrix vector products require 78 seconds each with the FMM, with only 10 seconds spent in the MPI communications. A final residual norm decrease of  $6 \cdot 10^{-3}$  was obtained after 100 iterations.

Then, using the surfacic potentials obtained by the solver, we ran a near field computation. That is, with a simple matrix-vector product (accelerated by the FMM) using Equation (9), we compute the diffracted and total pressure around the wing. This calculation was performed on a grid of  $700 \times 1400$  points in a vertical plane where  $x$  is varying from -35 m to +25 m and  $z$  ranging

from  $-15$  m to  $+15$  m ( $x$  being the axis of the plane). We notice that we are able to computed accurately the pressure in a domain of  $60\text{ m} \times 30\text{ m}$  that is  $560\lambda \times 280\lambda$ .

Both the solution of the linear system with full-GMRES and the near field computation would have been impossible to treat without the fast multipole acceleration. With the solution technique that was used a few years ago that was based on a dense  $LU$  factorization, this solution would have been affordable. The storage of the complex matrix would require 261 TBytes of RAM and its factorization would last more than 18 hours on the 32 768 processor IBM BlueGene computer. This elapsed time is estimated assuming that the factorization is run at the peak performance of the machine that “only” has 8 TBytes of RAM. That is, if the matrix might fit in memory, the numerical solution using previous numerical technology would require twice as much time on a computer having more than a thousand times more processors. We remind that the IBM BlueGene computer is currently the fastest computer installed in the world.

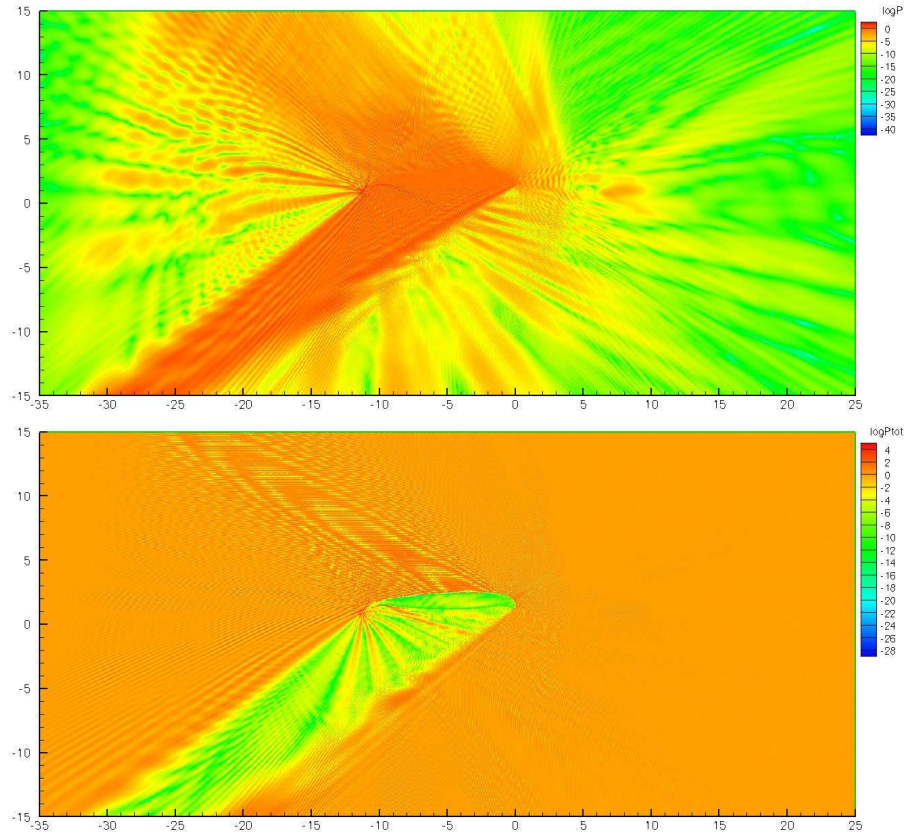


Figure 12: Diffracted and total pressure around the wing computed here on a  $700 \times 1400$  points grid using the near field FMM

### 3.4 Parallel Scalability

In this section, we will present the level of performance that can be obtained with this method for various numbers of processors. For that purpose, we use the case described in the previous paragraph. We obtain very different results whether we look at the solver’s performance or the near field computation performance. Indeed, the solver behaves nicely (see Table 1) since it is mainly “pure” computation. Table 1 shows the total time, the matrix assembly time, the SPAI

assembly time (in minutes) and the time for one matrix-vector product (in seconds).

Number of processors	Total time (minutes)	FMM assembly (minutes)	SPAI assembly (minutes)	FMM time (seconds)
8	1887	324	600	285
16	1035	142	275	154
32	522	61	134	78

Table 1: Parallel performance of the FMM solver

On the other hand, for the near field calculation a huge text result file is written by processor 0 at the end of the process. The writing of this large file is sequential which explains the poor parallel performance observed on the total time as it can be seen in Table 2. The matrix-vector time scales pretty well, but it represents only a small part of the entire process. Hence, this is usually enough to use 4 processors for this second part of the simulation.

Number of processors	Total time (minutes)	FMM time (seconds)
2	78	30
4	53	15
8	46	8
16	48	5

Table 2: Parallel performance of the FMM near field computation

A more thorough analysis of the FMM scalability was performed in [26] for the electromagnetic wave propagation problem (which is very close to the acoustic model used here). A parallel performance of up to 80 percent on 64 processors for the total solver time was obtained on high performance computers such as an IBM Power 3 and a SGI Origin 3800. On our cluster, the gigabit interconnection is too slow to allow such speed-up. Nevertheless, the parallel performance remains acceptable up to 32 processors.

## 4 Concluding remarks and prospectives

A new challenge for aeroacoustics is now to refine the underlying models. Up to now, the acoustics scattering simulations have been done assuming an acoustic source term based on a modal model. Even though this approach has been validated by numerous studies, it is quite restrictive in terms of configurations it can handle. To be more specific, at this stage axial geometries can be taken into account but more complex - especially non radial shapes - are off the road limiting the emergence of innovative solutions for this environmental problem. A more generic approach would be to handle both physics using complex 3D approaches. At the industrial level, this is usually a bit more complex than expressing the coupled system and solving it using the appropriate mathematical and numerical instruments. Usually, each of the involved physics are solved in structures that have few connections between them. As a matter of fact, this situation is then reflected in the underlying tools - very few of them are able to export intermediate data that could enable the design of weak or strong coupling methods.

EADS-CRC is now engaged in the European project SIMDAT which aims at experimenting the solution of aeroacoustics coupled problems in virtual organizations. At this stage, the test problem under consideration is the optimization of a wing design under multi-physics criteria. There are multiple engineering challenges, the top ranked being to be capable to set-up an adequate numerical

scheme leveraging the use of existing parallel distributed software. Other interesting challenges being to be able to operate virtual organizations under the collaborator/competitor model. The collaborating partners on this project may be competitors on some others. This framework requires to enforce strict access control and confinement of the data in a shared environment.

The sound generated by turbulence is obviously an important source of noise and raises many questions of fundamental and engineering interest. Recent advances in both computational fluid dynamics and computational acoustics, such as those presented previously, offer many tools to develop new techniques in computational aeroacoustics. A more complete overview on the different approaches can be found in [2]. The approach currently retained is an hybrid method based on computation of the aerodynamic fluctuations by solving the Navier-Stokes equations, and on the calculation of the radiated noise by acoustic analogy. The current acoustic analogy that have been selected is the so called Lighthill's analogy which is a third-order wave equation namely the Lilley's equation [16]. Using this analogy, the acoustic pressure generated by a turbulent flow is expressed as a function of the Lighthill tensor  $T_{ij} = \rho u_i u_j$ . This tensor is therefore calculated using an aerodynamics solver and the most significant sources are then injected in the acoustics solver.

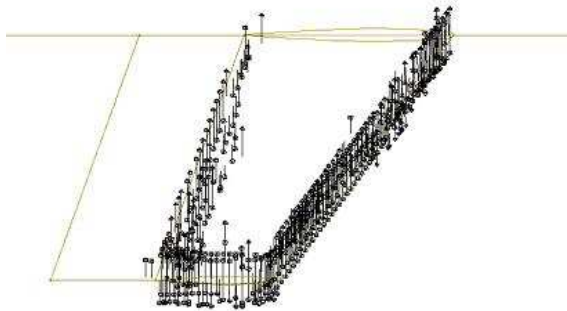


Figure 13: Significant Lighthill tensors for ONERA M6 wing test-case

The new generation of object oriented computational fluid dynamics and computational acoustics software has been proved to be effective on a large number of industrial problems. Moreover, their performances have been assessed on a large number of high performance computing platforms. On the acoustics side, the last ten years have shown a deep penetration of exact numerical methods in the engineering processes. As a consequence, the acoustic department is challenging these new numerical methods due to the size and performance of the new airplane generation. This is reinforced by the strict regulation for the control of acoustic pollution by aircrafts. Finally, it should be remarked that multi-scale physics and algorithms are probably very promising approaches for high-performance computing and numerical simulation in a near future. A challenge is therefore to be capable to deploy such new tools in the context of virtual organizations also known as the extended enterprise.

## References

- [1] G. Alléon, M. Benzi, and L. Giraud. Sparse approximate inverse preconditioning for dense linear systems arising in computational electromagnetics. *Numerical Algorithms*, 16:1–15, 1997.
- [2] C. Bailly and C. Bogey. Simulations numériques en aéroacoustique. In *Acoustique dans les écoulements*, Rocquecourt, France, January 12-15 2000. Ecoles CEA-EDF-INRIA.

- [3] L. Cambier and M. Gazaix. elsA: an efficient object-oriented solution to CFD complexity. In 40th AIAA Aerospace Science Meeting & Exhibit, Reno, number AIAA 2002-0108, pages 14–17, January 2002.
- [4] B. Carpentieri. *Sparse preconditioners for dense linear systems from electromagnetic applications*. PhD thesis, CERFACS, Toulouse, France, April 2002.
- [5] B. Carpentieri, I. S. Duff, and L. Giraud. Sparse pattern selection strategies for robust Frobenius-norm minimization preconditioners in electromagnetism. *Numerical Linear Algebra with Applications*, 7(7-8):667–685, 2000.
- [6] B. Carpentieri, I. S. Duff, L. Giraud, and G. Sylvand. Combining fast multipole techniques and an approximate inverse preconditioner for large parallel electromagnetism calculations. *SIAM J. Scientific Computing*, to appear.
- [7] K. Chen. An analysis of sparse approximate inverse preconditioners for boundary integral equations. *SIAM J. Matrix Analysis and Applications*, 22(3):1058–1078, 2001.
- [8] R. Coifman, V. Rokhlin, and S. Wandzura. The Fast Multipole Method for the Wave Equation: A Pedestrian Prescription. *IEEE Antennas and Propagation Magazine*, 35(3):7–12, 1993.
- [9] E. Darve. *Méthodes multipôles rapides : Résolution des équations de Maxwell par formulations intégrales*. PhD thesis, Université Paris 6, juin 1999.
- [10] M. Gazaix, A. Jolles, and M. Lazareff. The elsA object-oriented computational tool for industrial applications. In *23rd Congress of ICAS*, Toronto, Canada, September 8-13 2002.
- [11] L. Giraud, J. Langou, and G. Sylvand. On the parallel solution of large wave propagation problems. *Journal of Computational Acoustics*, to appear.
- [12] A. Jameson. Solution of the Euler equations for two dimensional transonic flow by a multigrid method. *Appl. Math. Comput.*, 13:327, 1983.
- [13] G. Karypis and V. Kumar. METIS, unstructured graph partitioning and sparse matrix ordering system. version 2.0. Technical report, University of Minnesota, Department of Computer Science, Minneapolis, MN 55455, August 1995.
- [14] L. Yu. Kolotilina. Explicit preconditioning of systems of linear algebraic equations with dense matrices. *J. Sov. Math.*, 43:2566–2573, 1988. English translation of a paper first published in *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo im. V.A. Steklova AN SSSR 154 (1986) 90-100*.
- [15] J. Lee, J. Zhang, and C.-C. Lu. Incomplete LU preconditioning for large scale dense complex linear systems from electromagnetic wave scattering problems. *J. Comp. Phys.*, 185:158–175, 2003.
- [16] G. M. Lilley. The radiated noise from isotropic turbulence. *Theor. Comput. Fluid. Dyn.*, 6:281, 1994.
- [17] J. C. Nédélec. Ondes acoustiques et électromagnétiques; équations intégrales. Technical report, Cours DEA, Ecole Polytechnique, Paris, 1996.
- [18] OpenMP Architecture Review Board. OpenMP Fortran Application Program Interface. Technical Report Version 2.0, 2000.
- [19] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7(3):856–869, 1986.

- [20] K. Sertel and J. L. Volakis. Incomplete LU preconditioner for FMM implementation. *Microwave and Optical Technology Letters*, 26(7):265–267, 2000.
- [21] X. Q. Sheng, J. M. Jin, J. Song W. C. Chew, and C. C. Lu. Solution of combined field integral equation using multilevel fast multipole algorithm for scattering by homogeneous bodies. *IEEE Ant. Propag.*, 46(11):1718–1726, November 1998.
- [22] J. Song, C-C Lu, and W. C. Chew. Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects. *IEEE Transactions on Antennas and Propagation*, 45(10):1488–1493, October 1997.
- [23] J. M. Song and W. C. Chew. The fast Illinois solver code : Requirements and scaling properties. *IEEE Computational Science and Engineering*, 5(3):19–23, July-September 1998.
- [24] J. M. Song, C. C. Lu, W. C. Chew, and S. W. Lee. Fast Illinois solver code. *IEEE Antennas and Propagation Magazine*, 40(3), June 1998.
- [25] J. L. Steger and R. F. Warming. Flux vector splitting of the inviscid gas-dynamic equation with applications to finite difference methods. *Journal of Computational Physics*, 40:263–293, 1981.
- [26] G. Sylvand. *La méthode multipôle rapide en électromagnétisme : performances, parallélisation, applications*. PhD thesis, Ecole Nationale des Ponts et Chaussées, Juin 2002.
- [27] S. A. Vavasis. Preconditioning for boundary integral equations. *SIAM J. Matrix Analysis and Applications*, 13:905–925, 1992.
- [28] S. Yoon and A. Jameson. A LU-SSOR scheme for the Euler and Navier-Stokes equations. In *AIAA 25th Aerospace Sciences Meeting*, number AIAA-87-0600, Reno, Nevada, USA, January 12-15 1987.