Solving the Helmholtz equation at high wave numbers on a parallel computer with a shared virtual memory.

Marie-Odile Bristeau [*] Jocelyne Erhel [†] Philippe Feat [‡]
Roland Glowinski [§] Jacques Periaux [¶]

[*]INRIA-Rocquencourt - 78153 Le Chesnay Cedex - FRANCE
[†]IRISA - Campus Universitaire de Beaulieu - F-35042 RENNES Cedex - FRANCE
[‡]IRISA - Campus Universitaire de Beaulieu - F-35042 RENNES Cedex - FRANCE
[§]Department of Mathematics,University of Houston - Houston, TEXAS 77204-3476 - USA
[¶]Dassault Aviation,78 Quai Marcel Dassault - 92214 Saint-Cloud - FRANCE

Running Head : wave equation on a KSR1

Author receiving the queries :
Jocelyne Erhel
IRISA
Campus de Beaulieu
35042 Rennes Cedex
FRANCE
tel +33 99 84 73 39
fax +33 99 38 38 32
e-mail erhel@irisa.fr

Abstract

In this paper, we address the implementation of finite element codes on a parallel computer providing a shared virtual memory. We solve the Helmholtz equation at high wave numbers by a method combining least squares formulation and finite element discretization. Numerical experiments concerning the scattering of planar waves by nonconvex obstacles show the efficiency of the new algorithm. Parallelism relies heavily on shared variables and parallel loops ; performances are enhanced by considering local versus global nodes and by renumbering the mesh. Experiments are implemented on a KSR computer with up to 20 processors.

1 Introduction

The advent of parallel computers allows to solve large and complex scientific problems. Whereas a shared memory architecture provides more flexibility and facility to use, a distributed memory organisation provides scalability thanks to a large number of processors.

A new paradigm is emerging where the parallel architecture is built upon physically distributed memories but logically shared variables. This solution is adopted for example in the KSR computer with a so-called "ALL-CACHE" facility in the hardware. It is also implemented in the KOAN system [Lahjomri et Priol, 1992] which is a software developed on the hypercube iPSC/2.

In this paper, we address the implementation of finite element methods using unstructured meshes on such architectures. The main drawback of unstructured meshes is the requirement for indirect addressing which has a strong impact on data locality. We deal with this problem for a very time-consuming operation of most finite element codes, namely the assembling of a matrix or a right-hand side. Elementary matrices are computed independently at each element of the mesh, using nodal variables and are accumulated at the nodes of the element. This assembling induces write conflicts between neighboring elements having a common node.

Finite element codes have been successfully implemented on vector computers using a coloring technique to define a partition of the elements into sets of disjoint elements. Vectorisation is achieved in each color, yielding good performances [Erhel, 1989]. However, this technique is not applicable to MIMD computers, because it would induce too much synchronisation points. On distributed memory machines, a natural idea is to partition the domain into subdomains and to allocate one subdomain per processor. Communications then occur on the boundaries of the subdomains. Many implementations use efficiently this technique [Fezoui et al., 1991]. It has also been adapted to a shared virtual memory computer, the KSR, achieving much better performances than a straightforward automatic parallelisation [Lanteri et Farhat, 1993]. However, this approach requires preprocessing to define the subdomain partitioning and mainly the data structures. Moreover, communication time is often reduced at the price of load unbalancing. Using shared variables, either with a physically or virtually shared memory, it is easy to include the assembling phase into critical sections or to lock the conflicting variables by some other means. However, this solution is efficient only if the overhead is relatively low compared to the time of computation.

Here we propose a new technique using implicitely subdomains but using automatic parallelisation and providing a well-balanced partition. The idea is to assemble the vectors locally in each processor and to sum the contributions in a later phase. In [Bodin et Priol, 1992], this is done for all nodes and gives good performances on a ipsc2 with the system KOAN [Lahjomri et Priol, 1992]. However, for other applications and/or computers, this accumulation must be done only for neighbouring nodes to avoid memory traffic. Moreover, a renumbering of the mesh improves data locality, thus minimizing communications.

We have implemented this technique in a finite element code solving the Helmholtz equations from Acoustics and two-dimensional Electro-Magnetics. The key idea here is to go back to the original wave equation and to apply exact controllability [Lions, 1988] in order to find its time periodic solutions. This method combines a least squares formulation and a finite element discretization. Numerical experiments concerning the scattering of planar waves by nonconvex obstacles show the efficiency of the new algorithm [Bristeau et al., 1993]. They are implemented on the KSR1, which provides a hardware facility to virtually share the physically distributed memory.

The paper is organised as follows. The second section is devoted to the physical problem and the numerical method to solve it. From this example, we derive a general structure of the assembling phase in finite element codes. The third section describes briefly the KSR1 architecture and software, explains the parallelisation and compares it to other solutions. The last section presents numerical experiments and performance results on various domains.

2 The Helmholtz Equation and its Equivalent Wave Problem

Let us consider a scattering body B, of boundary $\partial B = \gamma$, illuminated by an *incident monochromatic wave* of frequency $f = kc/2\pi$ (see figure 1) ; we denote by Γ the artificial boundary of the domain Ω.

The solution U of the Helmholtz equation

$$\left\{ \begin{array}{ll} \Delta U + k^2 U = 0 & \text{in } \mathbb{R}^d \backslash \bar{B} (d = 2, 3), \\ U = G & \text{on } \gamma, \\ \frac{\partial U}{\partial n} - ikU = 0 & \text{on } \Gamma \end{array} \right. \tag{1}$$

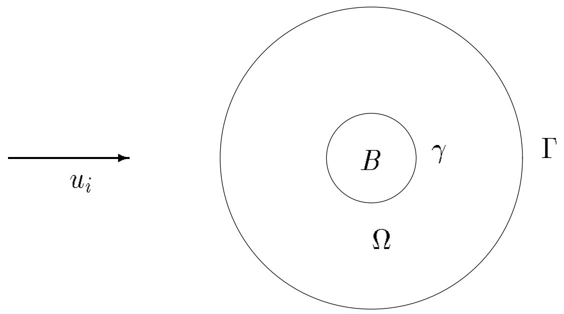

Figure 1: Scattering body (u_i : incident field)

is related to the T-periodic solution ($T = 2\pi/k$) $u(x,t) = Re(e^{-ikt}U(x))$of the following wave equation and associated boundary conditions

$$\begin{cases} u_{tt} - \Delta u = 0 & \text{in } Q(= \Omega \times (0,T)), \\ u = g & \text{on } \sigma(= \gamma \times (0,T)), \\ \frac{\partial u}{\partial n} + \frac{\partial u}{\partial t} = 0 & \text{on } \Sigma(= \Gamma \times (0,T)) \end{cases} \tag{2}$$

where $g(x,t)$ is a periodic function related to G by $g(x,t) = Re(e^{-ikt}G(x))$.

The main goal of this article is to find periodic solutions to system (2) without solving the Helmholtz problem (1) ; this means solutions satisfying (2) and

$$u(0) = u(T), \quad u_t(0) = u_t(T). \tag{3}$$

Introducing the control $e = (e_0, e_1)$ which must satisfy $e_0 = u(0) = u(T)$ and $e_1 = u_t(0) = u_t(T)$, problem (2)-(3) can be formulated as an *exact controllability* problem and solved by methods directly inspired by the J.L. Lions *Hilbert Uniqueness Method* (HUM)[Lions, 1988].

A right choice for the space E containing $e = (e_0, e_1)$ is fundamental. We advocate

$$\begin{aligned} E &= V_g \times L^2(\Omega), \\ where\ V_g &= \{\varphi | \varphi \in H^1(\Omega), \varphi|_\gamma = g(0)\}. \end{aligned}$$

We use a *least squares* formulation solved by a *preconditioned conjugate gradient algorithm* [Bristeau et al., 1993] ; each iteration of the algorithm requires the solution of a state equation (a forward wave equation), of an adjoint equation (a backward wave equation) and of an elliptic problem for the preconditioning of the gradient. The algorithm starts from from a given initial control (e_0^0, e_1^1) and computes in an initialisation step the initial gradient (g_0^0, g_1^1) with the initial descent direction (w_0^0, w_1^1) . Then the sketch of an iteration, updating the control (e_0^{n+1}, e_1^{n+1}) from the previous one (e_0^n, e_1^n) is as follows :

1. Solve the state equation :

$$\left\{ \begin{array}{ll} \overline{y}_{tt}^n - \Delta \overline{y}^n = 0 & \text{in } Q, \\ \overline{y}^n = 0 & \text{on } \sigma, \\ \frac{\partial \overline{y}^n}{\partial n} + \frac{\partial \overline{y}^n}{\partial t} = 0 & \text{on } \Sigma, \\ \overline{y}^n(0) = w_0^n; \quad \overline{y}_t^n(0) = w_1^n. & \end{array} \right. \tag{4}$$

2. Solve the adjoint equation :

$$\left\{ \begin{array}{ll} \overline{p}_{tt}^n - \Delta \overline{p}^n = 0 & \text{in } Q, \\ \overline{p}^n = 0 & \text{on } \sigma, \\ \frac{\partial \overline{p}^n}{\partial n} + \frac{\partial \overline{p}^n}{\partial t} = 0 & \text{on } \Sigma \end{array} \right. \tag{5}$$

   where $\overline{p}^n(T)$ and $\overline{p}_t^n(T)$ are computed by

$$\left\{ \begin{array}{l} \overline{p}^n(T) = \overline{y}_t^n(T) - w_1^n, \\ \int_\Omega \overline{p}_t^n(T) z dx = \int_\Gamma \overline{p}^n(T) z d\Gamma - \int_\Omega \vec{\nabla}(\overline{y}^n(T) - w_0^n)\vec{\nabla} z dx \quad \forall z \in V_0 \end{array} \right.$$

3. Solve an elliptic problem to compute the preconditioned gradient $\overline{g}^n = (\overline{g}_0^n, \overline{g}_1^n)$:

$$\left\{ \begin{array}{l} \int_\Omega \vec{\nabla}\overline{g}_0^n \vec{\nabla} z dx = \\ \int_\Omega \vec{\nabla}(w_0^n - \overline{y}^n(T))\vec{\nabla} z dx - \int_\Omega \overline{p}_t^n(0) z dx + \int_\Gamma \overline{p}^n(0) z d\Gamma \quad \forall z \in V_0, \\ \overline{g}_1^n = p^0(0) + w_1^n - \overline{y}_t^n(T) \end{array} \right. \tag{6}$$

4. Update the control, the gradient and the descent direction :

$$\left\{ \begin{array}{l} \rho_n = \int_\Omega (|\vec{\nabla} g_0^n|^2 + |g_1^n|^2) dx / \int_\Omega (\vec{\nabla}\overline{g}_0^n \vec{\nabla} w_0^n + \overline{g}_1^n w_1^n) dx, \\ e^{n+1} = e^n - \rho_n w^n, \\ g^{n+1} = g^n - \rho_n \overline{g}^n, \\ \gamma_n = \int_\Omega (|\vec{\nabla} g_0^{n+1}|^2 + |g_1^{n+1}|^2) dx / \int_\Omega (|\vec{\nabla} g_0^n|^2 + |g_1^n|^2) dx, \\ w^{n+1} = g^{n+1} + \gamma_n w^n. \end{array} \right. \tag{7}$$

5. Test for convergence :

$$\text{If } \gamma_n^{1/2} \le \epsilon \text{ then STOP.} \tag{8}$$

The practical implementation is based on a *time discretization* by a *centered second order* in time *explicit finite difference* scheme. This scheme is combined to *piecewise linear finite element* approximations for the space variables ; we use *mass lumping* to obtain a *diagonal* mass matrix for the acceleration terms.

For each conjugate gradient iteration, the most CPU expensive parts are the resolution of the state and adjoint equations. They rely on the unstructured mesh and the finite element method so that the value at a node is the accumulation of variables computed at the elements containing this node. The time discretisation involves a loop on the time steps, which are global and constant here, containing an assembling phase followed by an update phase. This structure is typical for finite element codes and can be summarized as below :

```
Loop on time steps n
  Loop on elements j of the mesh
    1/ compute XEL(j) using X(ME(j))
    2/ assemble XEL(j) in Y(ME(j)) : Y(ME(j))=Y(ME(j))+XEL(j)
  End
  Loop on nodes i of the mesh
    3/ update X(i) using Y(i)
  End
End
```

where ME is an array giving for each element the numbers of its nodes.

Each code is characterized by the relative cost of the three parts, namely the element computation 1/, the assembling 2/, the update 3/. In our application, the computation 1/ is not heavy so that the assembling phase 2/ is relatively costly. Furthermore, this accumulation is done at only one variable for each node of the element.

3 Parallelism

The only dependence which prevents from parallelism is the accumulation phase. Indeed, two neighbouring elements are conflicting to write into the same node. Moreover, these operations require the use of indirect addressing (gather for computing and scatter for accumulating).

The parallelisation depends on the target architecture. On vector computers, a colouring technique defines a partition into sets of elements with no common node. On distributed memory computers, a partition by subdomains allows to distribute the nodes and the elements [Fezoui et al., 1991]. On shared memory computers, parallelism can also be achieved by control distribution. Either the accumulation step is done in a critical section or it makes use of local variables which are in turn assembled into a global shared vector. On shared virtual memory, both approaches can a priori be studied. The domain decomposition method has been successfully implemented in [Lanteri et Farhat, 1993] for solving Navier-Stokes equations. However this solution requires a careful partitioning to balance the tasks and minimize the communications. Furthermore it relies on data structures to recover the connectivity.

Here we advocate the use of local variables which seems more promising to design an almost automatic run time tool. We do not retain the critical section solution which is definitely too expensive in a shared virtual memory environment for our application. Generally, the unit of coherency between local memories is a page or subpage and a mechanism allows to lock then unlock this subpage. This solution appears efficient if the lock operation is not too costly and/or the computation phase 1/ is relatively time consuming and/or the assembling phase 2/ involves several node variables stored in the same subpage. In our application, these requirements are not satisfied at all so that we got very poor performances when we implemented this technique [Feat, 1993].

Therefore we modify the code by introducing new variables, at the cost of some extra memory storage. The loop on the time steps remains sequential and the loop on elements and nodes are both parallel :

```
Sequential loop on time steps
  Parallel loop on elements j of the mesh, process p
    1/ compute XEL(j) using X(ME(j))
    2/ assemble XEL(j) in YLOC(p,ME(j))
  End
  Parallel loop on nodes i of the mesh
    Loop on process k
      2a/ Y(i) = Y(i)+YLOC(k,i)
    End
    3/ update X(i) using Y(i)
  End
End
```

However, this straightforward implementation does not work for our application, because the step 2a/ introduces a large overhead for at least two reasons [Feat, 1993]. First of all, this loop involves many unuseful operations on null terms ; secondly it requires many communications to transfer variables computed in one process and used in another. Therefore we had to optimize the technique to enhance the performances. The first problem is solved by distinguishing two kinds of nodes : local nodes which are assembled in a local variable YLOC by only one process and global nodes which are assembled by at least two processes. Currently we do not address the number of processes since we want a general and easy to implement technique. In the preprocessing step, a single loop on the elements allows to define an integer array giving for each node either the number of process computing the local node or a negative value (for example) if the node is global. The extra storage required is only one integer vector of size the number of nodes. Now the code becomes as below :

```
Sequential loop on time steps
  Parallel loop on elements j of the mesh, process p
    1/ compute XEL(j) using X(ME(j))
    2/ assemble XEL(j) in YLOC(p,ME(j))
  End
  Parallel loop on nodes i of the mesh
    If the node is local Then
      2b/ Y(i) = YLOC(noproc(i),i)
    Else
      Loop on processus k
        2a/ Y(i) = Y(i)+YLOC(k,i)
      End
    Endif
    3/ update X(i) using Y(i)
  End
End
```

This simple optimisation improves considerably the performances. However, many communications still occur if the elements and nodes are partitioned at random. Indeed, variables X computed in one process must be read by another and converserly variables YLOC must be copied in another local memory. Since the memory is really distributed and hierarchically organised, a good data locality is mandatory. Therefore, we try to minimize communications by assigning to a same process elements and their nodes.

This is achieved by a renumbering similar to a subdomain partitioning. The elements and also the nodes are renumbered consecutively in one subdomain. Hence a partition by blocks of the parallel loops recovers implicitely the subdomain partitioning and work is automatically well balanced. Most nodes are local, used and computed by the same process, except the nodes at the boundaries which are global and require communications. The implementation relies heavily on shared variables and on automatic parallelisation. Below is the preprocessing step :

```
Read the mesh
Renumber nodes and elements by a subdomain partitioning
Define local nodes and global nodes
```

Currently we use a simple partitioning by slices which is not optimal to minimize the size of the boundaries but which is straightforward to implement. We plan to test more sophisticated tools in further experiments.

4 Numerical Experiments

We illustrate here the method and the performances by two numerical results (others are described in [Bristeau et al., 1993]) on non-convex reflectors.

We consider first the scattering of a plane monochromatic wave by a semi-open cavity. We choose $f = 3Ghz$ implying that the wavelength λ is 0.10 m. The inside dimensions are $4\lambda \times \lambda$ and the thickness of the wall is $\lambda/5$. The finite element triangulation has 17629 vertices and 34452 triangles. The value of the time step Δt corresponds to 40 time steps per period. We consider an illuminating wave of incidence $\alpha = 30°$. The contours of the scattered field e_o (real part) are shown on figure 3 and figure 4 plots the convergence of the cost function introduced in the least-squares formulation ; The convergence is reached within 156 iterations.

We consider also the scattering of a plane monochromatic wave by a 2D shape given by the cross section of a Falcon 50(Dassault Aviation) in the symmetry plane. The length of the plane is about 18 m and the height 6 m. We choose f=0.3 Ghz, so the wavelength is 1 m ; the incidence of the wave is 30 degrees. Figure 2 shows an enlargment of the mesh which is done of 60590 nodes and 119332 triangles. The contours of the scattered field is presented on figure 5, while figure 6 shows the decrease of the cost function.

We implemented the code on a KSR1 computer which has 32 processors connected by a ring. Each processor has a local cache of 32 MB. The ALL-

CACHE memory implements a shared address space storing data in units of pages and subpages. Pages contain 16 KB divided into 128 subpages of 128 bytes. The unit of allocation in a local cache is a page, whereas the unit of transfer and sharing between local caches is a subpage. Coherency is achieved by an invalidation protocol. Multiple copies of a same subpage are allowed, but all other copies are marked invalid as soon as one of them is requested for writing. Each processor is a 64-bit processor, so that single precision is 64-bit and double precision is 128-bit. Each processor is register-to-register oriented and has a subcache divided into 0.25 MB of instruction subcache and 0.25 MB of data subcache.

The KSR1 memory architecture provides the familiar shared-memory programming model thanks to the ALLCACHE memory. Parallel constructs are expressed by compiler directives as usual. The KAP precompiler automatically inserts directives at the loop level when possible. Variables are shared by defaults but private variables can be declared in directives. In our application, we inserted manually directives because we had to use the process number. The loops on elements and nodes are parallelized with a partition by slices. Figure 7 is the code of the procedure to solve the state equation. The procedure to solve the adjoint equation is similar since it solves also a wave equation.

We did not yet parallelize other parts of the program, but only the procedures to solve the wave equation (state and adjoint). Most loops on nodes are automatically parallel, for example the computation of the gradient and the update of the solution. The main drawback remains the preconditioning by an elliptic problem (6) which involves solving two sparse unstructured triangular systems (lower and upper) at each gradient iteration. Further research should lead to a subdomain preconditioning.

We give now performances on a KSR1 for both numerical experiments. We only give the speed-ups for the parallel procedures since other parts are still sequential as shown by the timings. The tables 1 and 2 give CPU times in seconds after 10 iterations of conjugate gradient, including the initialisation step, but excluding the preprocessing step (in particular I/O to read the mesh, computation of some geometric data structures, renumbering). It should be noted that these timings were taken in a multiple-user environment.

The parallel performances are quite good for a small number of processors. The speed-up almost stops increasing after 10 processors for the small mesh and 14 processors for the second mesh. However the parallelism seems scalable in the sense that performances increase noticeably from a small mesh to a largest one. The resolution of the wave equation requires a small number of operations since the problem is two-dimensional and since the numerical scheme of integration is simple. This small granularity mainly explains the limitations of the speed-up to a small number of processors. The performances will improve when dealing with three-dimensional problems. The scalar performance is far below the peak performance because of indirect addressing and memory-bound computations. Indeed the number of floating-point operations is relatively small compared to the memory requests. The sequential part represents roughly 40 % of the computation and should be parallelized to get satisfactory global results. In this sequential part, only the elliptic solver is difficult to parallelize. It is planned to study either direct method with a subdomain approach or an iterative solver.

5 Conclusion

We have discussed in this paper the implementation of a finite element code on a shared virtual memory computer. The numerical method is based on exact controllability to solve Helmholtz and two-dimensional harmonic Maxwell equations with good convergence properties which are conserved for *large wave numbers* and for complex geometries. The main computational parts are the assembling of a vector, inducing write conflicts between neighboring elements. Parallelism is done at a loop level. For local nodes, elementary variables are assembled directly whereas global nodes require the accumulation of contributions from other tasks. Renumbering of the mesh improves data locality by minimizing the number of global nodes. Numerical experiments on a KSR computer show good speed-ups for sufficiently large meshes. We plan now to adapt this method for assembling a matrix with a reasonable memory extra-storage. This technique is promising to design a run-time compiler tool in order to parallelize almost automatically a finite element code.

References

[Bodin et Priol, 1992] Bodin, F., Priol, T. 1992. Performance Evaluation of BECAUSE Test-Program 2.5.1 on an iPSC/2 running KOAN shared virtual memory. In *Because European Workshop, Sophia-Antipolis, France.*

[Bristeau et al., 1993] Bristeau, M-O., Glowinski, R., Périaux, J. 1993. Scattering waves using exact controllability methods. *AIAA Paper* 93-0460, Reno (NV), USA.

[Erhel, 1989] Erhel, J. 1989. Simulation of compressible flows on supercomputers. In *7th Int. Conf. on Finite Element Methods in Fluid Problems*, Hunstville (Alabama), USA.

[Feat, 1993] Féat, P. 1993. Parallélisation d'un algorithme de résolution de l'équation des ondes. Rapport de stage, IRISA, France.

[Fezoui et al., 1991] Fezoui, L., Loriot, F., Loriot M., Regère, J. 1991. 2-D finite volume/finite element solver on a MIMD parallel machine. In *High Performance Computing II, Montpellier, France* M. Durand and F. El Dabaghi Ed., Elsevier Science Publishers, North-Holland.

[Lahjomri et Priol, 1992] Lahjomri, Z., Priol, T. 1992. Koan : a shared virtual memory for the ipsc/2 hypercube. In *CONPAR/VAPP92, France.*

[Lanteri et Farhat, 1993] Lanteri, S., Farhat, C. 1993. Viscous Flow Computations on MPP Systems : Computational/Implementational Issues and Performance Results for Unstructured Grids. In *6th SIAM conference on Parallel Processing for Scientific Computing*, Norfolk (Virginia), USA.

[Lax et Phillips, 1989] Lax, P.D., Phillips, R.S. 1989. *Scattering Theory.* Academic Press.

[Lions, 1988] Lions, J-L. 1988. Exact Controllability, stabilization and perturbations for distributed systems, *SIAM Review* 30 :1-68.

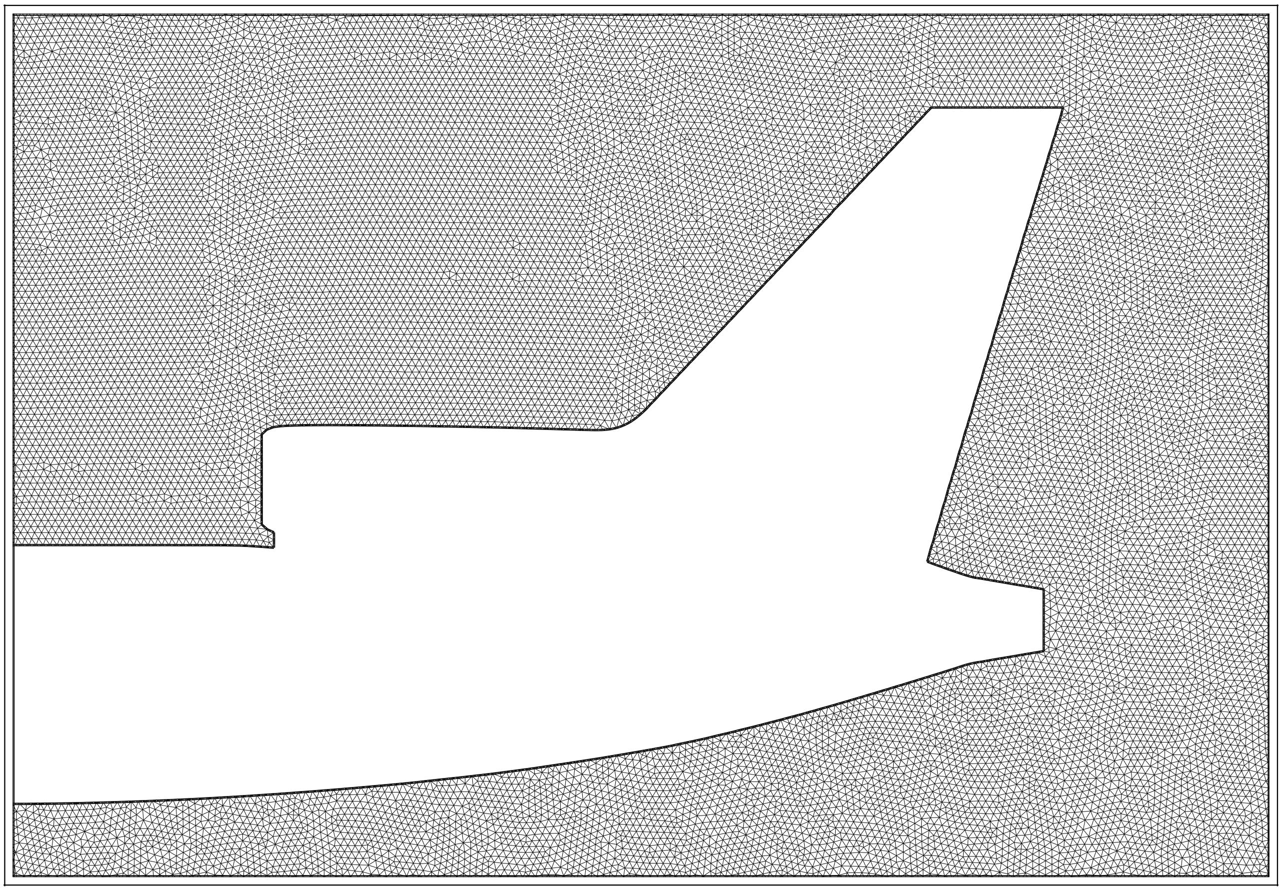

Figure 2: Detailed mesh near the air intake for the airplane

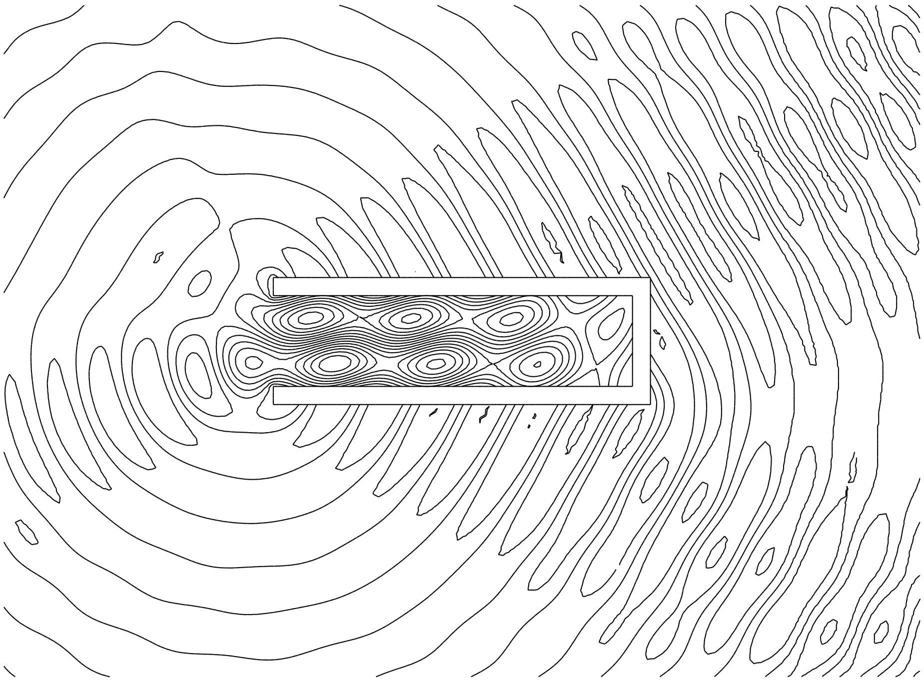

Figure 3: Contours of the scattered field e_0 (real component) for the cavity

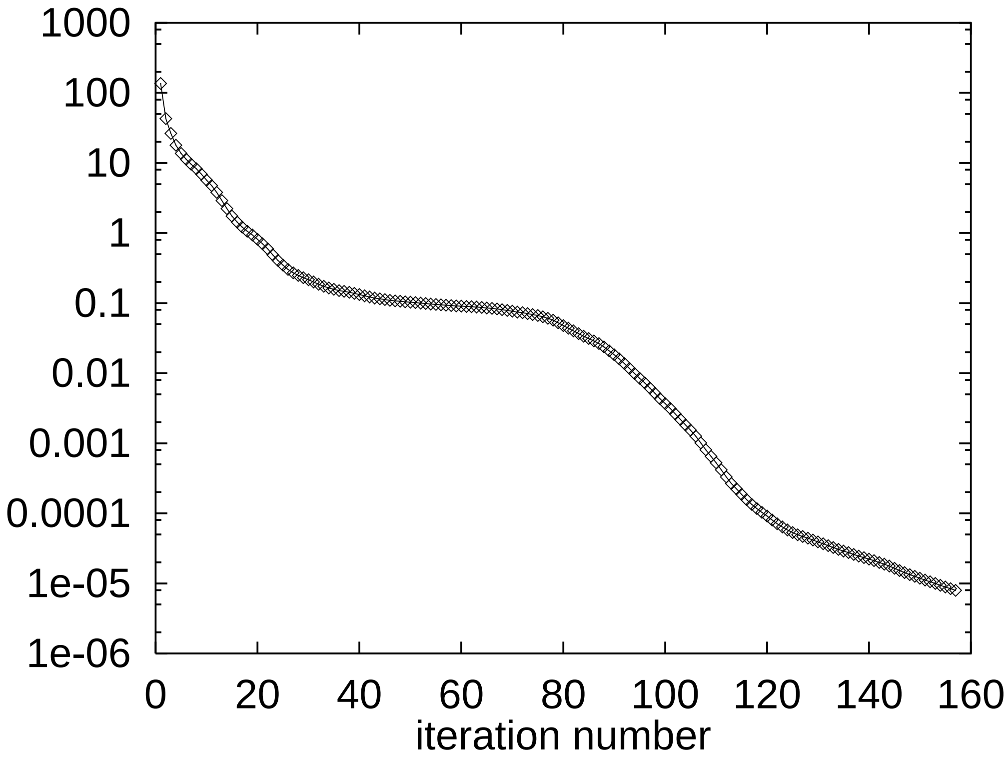

Figure 4: Convergence history of the cost function for the cavity

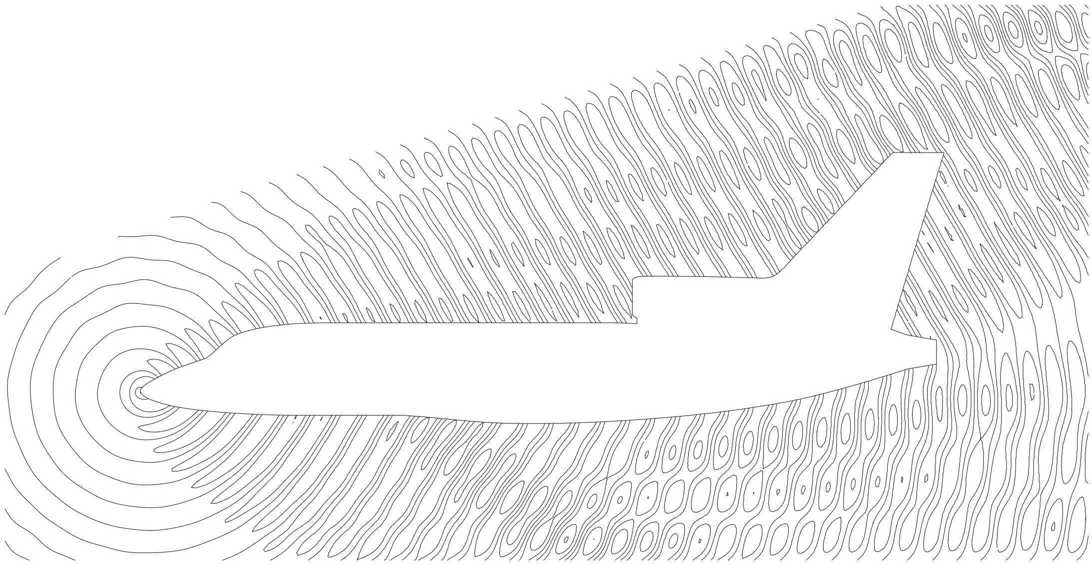

Figure 5: Contours of the scattered field e_0 (real component) for the 2D airplane shape

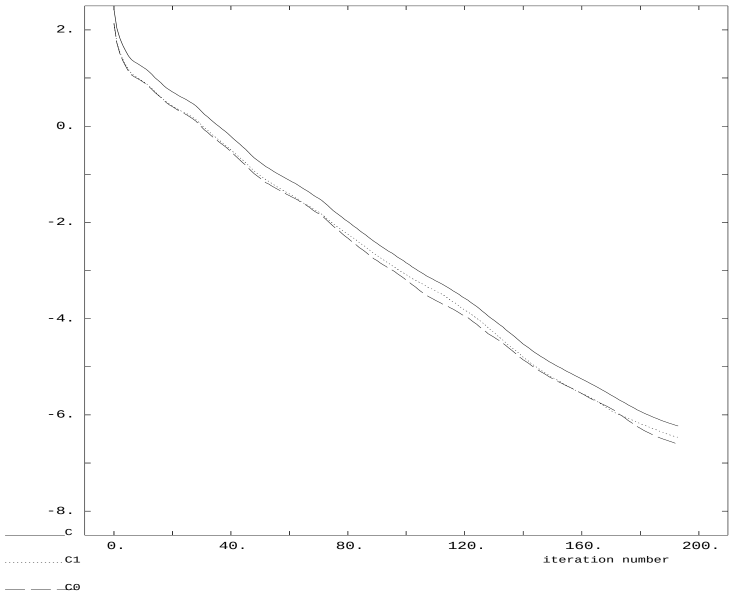

Figure 6: Convergence history of the cost function for the 2D airplane shape

```
C
C     TIME DISCRETISATION
C     *******************
C
      DO L=1,N
c                              -- one step further
C*KSR* PTILE(I)
       DO I=1,NS
         YNM1(I)=YN(I)
         YN(I)=YNP1(I)
         YNP1(I)=0.
       END DO
C
c                              -- elementary computation 1/
c                              -- and local assembling 2/
C*KSR* USER TILE(J,PRIVATE=(K,I,GXN,GYN,Y1,Y2,Y3))
       DO J=1,NT
        K=IPR_MID()+1
        DO I=1,NS
          YLOC(I,K)=0.
        ENDDO
         GXN=
     S YN(ME(1,J))*CQ(1,J)+YN(ME(2,J))*CQ(3,J)+YN(ME(3,J))*CQ(5,J)
         GYN=
     S YN(ME(1,J))*CQ(2,J)+YN(ME(2,J))*CQ(4,J)+YN(ME(3,J))*CQ(6,J)
         Y1=AIRE(J)*(GXN*CQ(1,J)+GYN*CQ(2,J))
         Y2=AIRE(J)*(GXN*CQ(3,J)+GYN*CQ(4,J))
         Y3=AIRE(J)*(GXN*CQ(5,J)+GYN*CQ(6,J))
         YLOC(ME(1,J),K)=YLOC(ME(1,J),K)+Y1
         YLOC(ME(2,J),K)=YLOC(ME(2,J),K)+Y2
         YLOC(ME(3,J),K)=YLOC(ME(3,J),K)+Y3
      END DO
C*KSR* END TILE
C
c                              -- assembling at nodes 2a and 2b/
c                              -- and update 3/
C*KSR* PTILE(I)
      DO I=1,NS
        YNP1(I)=0
```

```
        IF (NOPROC(I).LT.0 ) THEN
           DO K=1,NPROC
              YNP1(I)=YNP1(I)+YLOC(I,K)
           END DO
        ELSE
              YNP1(I)=YLOC(I,NOPROC(I))
        ENDIF
          AUX =AIRS(I)/3.+0.25*DT*SFR(I)
          YNP1(I)=(  (AIRS(I)/3.)*(2.*YN(I)-YNM1(I))
     S    -DT**2*YNP1(I)
     S    +0.25*DT*SFR(I)*YNM1(I)
     S    )/AUX
          IF(NG(I).EQ.ICO  )  YNP1(I)=0.
        END DO
C                              -- end of time steps
      END DO

C FINAL SOLUTION TO PREPARE THE ADJOINT
C ***********************************

C*KSR* PTILE(I)
      DO I=1,NS
        YN(I)=AC0*(YN(I)-V0(I))
        YNP1(I)=AC1*(0.5*(YNP1(I)-YNM1(I))/DT-V1(I))
        IF(NG(I).EQ.ICO) YNP1(I)=0.
      END DO
```

Figure 7: KSR1 Code to solve the wave equation

Table 1: CPU time (s) and speed-up for the cavity (17000 nodes)

Number of processors	1	2	4	6	10	14
Total CPU	347.3	269.5	227.1	231.3	210	205.2
CPU (STATE + ADJOINT)	212	134.2	89.3	77.9	71.5	68.1
CPU (Sequential part)	135.3	135.3	137.8	153.4	138.5	137.1
Speed-up (ETAT + ADJOINT)	1	1.6	2.4	2.7	3	3.11

Table 2: CPU time (s) and speed-up for the airplane (60000 nodes)

Number of processors	1	2	4	6	8	14	20
CPU (STATE + ADJOINT)	1021	551	351	280	265	224	210
Speed-up	1	1.85	2.91	3.65	3.85	4.56	4.86