TASK QUARTERLY 3 No 1 (1999), 17–37

A PARALLEL ADAPTIVE CODE FOR COMPRESSIBLE NAVIER-STOKES SIMULATIONS

KRZYSZTOF BANAŚ

Section of Applied Mathematics UCK,
Cracow University of Technology,
Warszawska 24, 31-155 Cracow, Poland;
email: Krzysztof.Banas@pk.edu.pl

Abstract: The paper presents a finite element code for compressible flow simulations. The code has two important features: adaptivity to increase accuracy of computations by selectively refining a finite element mesh and efficient parallel performance due to a special implementation based on the concept of patches of elements. The algorithm for approximating the compressible Navier-Stokes equations is a version of the stabilized finite element method. Three time integration strategies are implemented, explicit, linear implicit and nonlinear implicit, and the GMRES method is used to solve the systems of linear equations. For parallel simulations the code uses a special algorithm for mesh partition. The performance of the code is tested for two examples of supersonic flows: one inviscid and one viscous.

Keywords: numerical simulations, compressible flow, Navier–Stokes solver, finite element methode, paralel computing, adaptive meshes

1. Introduction

Parallel efficiency becomes nowadays a necessary feature of all competitive CFD codes. Different organizations of parallel systems pose specific requirements for a software. A distributed system with multiple processing units, each of which is equipped with its local memory, forms a logical organization that can be realized on different hardware platforms. The existence of popular software, like PVM and MPI implementations, for handling data communication on distributed systems creates a possibility for developing portable parallel computer codes.

Adaptivity is an important way for improving efficiency of numerical software for approximating field equations of mathematical physics. Based on a posteriori error estimates leads to optimal meshes that allow one to solve a given problem at a minimal computational cost. In the context of compressible flow equations, despite the lack of proven error estimates, adaptivity brings substantial savings in memory and CPU time usage due to the use of locally refined meshes in such regions as shocks and boundary layers.

The present article describes a parallel code designed for h-adaptive finite element simulations of laminar, compressible fluid flow. The paper is organized as follows: in Section 2 an algorithm for compressible flow simulations is described. Section 3 describes the finite element code with special stress on parallel implementation of the GMRES method for solving systems of linear equations and an algorithm for mesh partition. In Section 4 examples of numerical simulations are reported, including convergence and parallel performance tests. Section 5 presents final conclusions and the direction of further development of the code.

2. An algorithm for flow simulations

The Navier-Stokes equations, used as a mathematical model for laminar compressible flows, can be expressed in terms of conservation variables as follows:

$$\boldsymbol{U}(\boldsymbol{x},t)_{,t} + \boldsymbol{f}_i^E(\boldsymbol{U})_{,i} = \boldsymbol{f}_i^\mu(\boldsymbol{U},\nabla\boldsymbol{U})_{,i} \tag{1}$$

where:

$\boldsymbol{x}$ — point inside a computational domain $\boldsymbol{\Omega}_C \subset \mathbb{R}^s$, $s = 1, 2$ or 3,

t — time instant,

$\boldsymbol{U} = (\rho, \rho u_j, \rho e)^T$ — vector of conservation variables (ρ — density, u_j — velocity components ($j = 1,\ldots, s$) , e — total specific energy),

$\boldsymbol{f}_i^E = (\rho u_i, \rho u_i u_j + p\delta_{ij}, (\rho e + p)\, u_i)^T$ — vectors of Eulerian fluxes ($p = (\gamma - 1)(\rho e - {}^1/_2 \rho u_i\, u_i)$ — pressure, γ — ratio of specific heats, δ_{ij} – Kroneckers delta),

$\boldsymbol{f}_i^\mu = (0, \sigma_{ij}, \sigma_{ij}\, u_j - q_i)^T$ — vectors of viscous and heat fluxes (stress tensor $\sigma_{ij} = \mu\,(u_{i,j} + u_{j,i}) - {}^2/_3\, \mu\delta_{ij}\, u_{k,k}$),

$\mu = \frac{1.45T^{\frac{3}{2}}}{T+110} \cdot 10^{-6}$ — natural viscosity coefficient given by Sutherland's law in terms of temperature, $T = p/R\rho$,

$q_i = -\kappa\, T_{,i}$ — heat flux,

$Pr = \mu\gamma c_V/\kappa = 0.72$ — Prandtl relation,

$_{,t}$, $_{,i}$ — time and space partial derivatives $\frac{\partial}{\partial t}$, $\frac{\partial}{\partial x_i}$ (summation convention is always implied by repeated indices).

We assume that the system is accompanied by typical Dirichlet boundary conditions met in practice, namely: all unknowns specified on supersonic inflow, density and velocity given on subsonic inflow, pressure set on subsonic outflow, no boundary conditions on supersonic outflow and vanishing of velocity together with specified temperature or heat flux on a solid wall boundary [1].

A stabilized finite element method used for the space discretization of (1) reads: Find $\boldsymbol{U}^{\mathcal{D}}(\boldsymbol{x}) \in [H^1(\boldsymbol{\Omega}_C)]^{s+2}$ satisfying Dirichlet boundary conditions and such that for every est function $\boldsymbol{W} \in [H^1(\boldsymbol{\Omega}_C)]^{s+2}$, vanishing wherever Dirichlet boundary conditions are set, the following holds:

$$\int_{\Omega_C} \boldsymbol{W}^T \boldsymbol{U}_{,t}^{\Theta} dV - \int_{\Omega_C} \boldsymbol{W}_{,i}^T \boldsymbol{f}_i^E\left(\boldsymbol{U}^{\Theta}\right) dV +$$
$$+ \int_{\partial\Omega_C} \boldsymbol{W}^T \left(\boldsymbol{f}_i^E\left(\boldsymbol{U}^{\Theta}\right) - \boldsymbol{f}_i^{\mu}\left(\boldsymbol{U}^{\Theta}\right) n_i ds + \int_{\Omega_C} \boldsymbol{W}_{,i}^T \boldsymbol{K}_{ij}\left(\boldsymbol{U}^{\Theta}\right) \boldsymbol{U}_{,j}^{\Theta} dV = 0, \qquad (2)$$

where:

- n_i — outward normal unit vector
- $\boldsymbol{K}_{ij} = \boldsymbol{K}_{ij}^{\mu} + \boldsymbol{K}_{ij}^{L} + \boldsymbol{K}_{ij}^{AV}$: $\boldsymbol{K}_{ij}^{\mu}$ — natural viscosity matrices ($\boldsymbol{f}_j^{\mu} = \boldsymbol{K}_{ij}^{\mu}(\boldsymbol{U})\ \boldsymbol{U}_j$),
- $\boldsymbol{K}_{ij}^{L}$ — linear stabilization matrices,
- $\boldsymbol{K}_{ij}^{AV}$ — nonlinear artificial viscosity matrices.

Linear stabilization matrices, introduced to prevent oscillations that appear when standard Galerkin procedures are applied to equations of convection dominated flows, have the form [2]:

$$\boldsymbol{K}_{ij}^{L} = \boldsymbol{A}_i \frac{h}{2}\left(\boldsymbol{A}_1^2 + \boldsymbol{A}_2^2\right)^{-1/2} \boldsymbol{A}_j$$

($\boldsymbol{A}_i = (\boldsymbol{f}_i^E)_{,U}$ and h denotes a linear element size).

For simplified 1D or scalar cases stabilization by matrices $\boldsymbol{K}_{ij}^{L}$ can be shown to be equivalent to upwind differencing of flux term [3, 4].

Nonlinear artificial viscosity matrices $\boldsymbol{K}_{ij}^{AV}$ (nonlinearity is understood with respect to the gradient of unknowns) are based on the residual of the steady state Euler equations ($\boldsymbol{f}_{i,i}^{E} = 0$) [5]:

$$\boldsymbol{K}_{ij}^{AV} = \frac{h}{2} \frac{\sqrt{\left(\boldsymbol{f}_{k,k}^E\right)^T \boldsymbol{A}_0^{-1} \boldsymbol{f}_{k,k}^E}}{\sqrt{\boldsymbol{U}_{,l}^T \boldsymbol{A}_0^{-1} \boldsymbol{U}_{,l} + \varepsilon}} \delta_{ij} \boldsymbol{I}.$$

Here ε is some small positive constant, $\boldsymbol{I}$ denotes the identity matrix and $\boldsymbol{A}_0^{-1}$ is the Hessian of the entropy function $\eta = -\rho \ln(p\rho^{-\gamma})$ with respect to the conservation variables ($\boldsymbol{A}_0^{-1} = \eta_{,UU}$) [6].

There are three time integration strategies implemented in the code: explicit, linear implicit and nonlinear implicit. Each strategy transforms (2) into a sequence of one step problems that advance the solution from time level t^n to t^{n+1}. All strategies use first the order finite difference formula

$$\boldsymbol{U}_{,t}^{\Theta} = \frac{\boldsymbol{U}\left(t^{n+1}\right) - \boldsymbol{U}\left(t^n\right)}{t^{n+1} - t^n} = \frac{\boldsymbol{U}^{n+1} - \boldsymbol{U}^n}{\Delta t}.$$

Explicit strategy corresponds to the choice

$$\boldsymbol{U}^{\Theta} = \boldsymbol{U}^n,$$

and the lumping of the mass matrix obtained from the term

$$\frac{1}{\Delta t} \int_{\Omega_C} \boldsymbol{W}^T \boldsymbol{U}^{n+1} dV.$$

Both implicit strategies assume

$$\boldsymbol{U}^{\Theta} = \boldsymbol{U}^{n+1},$$

but the linear method additionally uses the substitutions

$$\boldsymbol{f}_i^E\left(\boldsymbol{U}^{\Theta}\right) = \boldsymbol{f}_i^E\left(\boldsymbol{U}^n\right), \qquad \boldsymbol{f}_i^{\mu}\left(\boldsymbol{U}^{\Theta}\right) = \boldsymbol{f}_i^{\mu}\left(\boldsymbol{U}^n\right).$$

These substitutions together with standard finite element procedures of using element shape functions transform the problem (2) into a system of linear equations.

The fully nonlinear implicit scheme requires the solution of nonlinear equation at each time step. This is achieved by an approximate Newton method that finds the solution $\boldsymbol{U}^{n+1}$ after a sequence of linear problems:

For $k = 0,\ 1,\ 2,\ldots,\ k_{max}$ find $\boldsymbol{U}^{k+1}$ satisfying Dirichlet conditions at specified boundary nodes and such that for every test function $\boldsymbol{W}$, vanishing wherever Dirichlet boundary conditions are set:

$$\frac{1}{\Delta t}\int_{\Omega_C} \boldsymbol{W}^T\boldsymbol{U}^{k+1}dV - \int_{\Omega_C} \boldsymbol{W}_{,i}^T\boldsymbol{A}_i\left(\boldsymbol{U}^k\right)\boldsymbol{U}^{k+1}dV + \int_{\partial\Omega_C} \boldsymbol{W}^T\boldsymbol{A}_i\left(\boldsymbol{U}^k\right)\boldsymbol{U}^{k+1}n_i dS +$$
$$+\int_{\Omega_C} \boldsymbol{W}_{,i}^T\boldsymbol{K}_{ij}\left(\boldsymbol{U}^k\right)\boldsymbol{U}_{,j}^{k+1}dV = \frac{1}{\Delta t}\int_{\Omega_C} \boldsymbol{W}^T\boldsymbol{U}^k dV + \int_{\partial\Omega_C} \boldsymbol{W}^T\boldsymbol{f}_i^{\mu}\left(\boldsymbol{U}^k\right)n_i dS, \tag{3}$$

where $\boldsymbol{U}^0 = \boldsymbol{U}^n$, $\boldsymbol{U}^{n+1} = \boldsymbol{U}^{k_{max}}$ and the number of iterations k_{max} is dictated by the required accuracy. Each linear problem is transformed into a system of linear equations, hence at each time step we have to solve k_{max} systems of linear equations.

The algorithms described are used for transient, as well as for steady state calculations. In the latter case the time increment Dt is computed locally for each element, based on a global value of the stability CFL number [1].

Each one step calculations in all algorithms form a separate problem that can be solved on a different mesh, thus allowing for mesh adaptation between any two time steps.

3. Finite element code

The standard finite element procedures for solving a given problem consist in creation of element stiffness matrices and load vectors, assembling them into a global stiffness matrix and a global load vector and then solving a resulting system of linear equations. The latter task is often performed by a separate, general purpose library procedure. The parallelization of such a solver is done independently of the finite element mesh and the problem solved.

In the code that we describe solvers are built into the program. For both explicit and implicit strategies they use the same particular data structure related to the finite element mesh and do not create a global stiffness matrix and a load vector. Instead, the computational domain is divided into small overlapping subdomains, called patches of elements, and for each subdomain local, explicit or implicit,

procedures are used. Each patch corresponds to a single finite element node and consist of all elements belonging to the support of node's shape function (in the case of regular meshes this means that the node is just a vertex of patch elements). Patches are recreated only after adaptations of the mesh. At each time step (or Newton iteration in the nonlinear algorithm) element stiffness matrices and load vectors are computed and assembled into patch stiffness matrices and load vectors. Patch stiffness matrices consist of non-zero entries in the rows of the global stiffness matrix corresponding to a given node (the same concerns patch load vectors and the global load vector). This defines a distributed storage scheme for the global stiffness matrix and the load vector based on a mesh topology.

Explicit procedures consist in computing element stiffness matrices and load vectors for all patch elements and advancing the solution according to the formula (2). Since the mass matrix is constant for all one step problems, patch stiffness matrices are computed only once, at the beginning of time integration, lumped, inverted and stored. Then at each time step only load vectors are computed to advance the solution.

Implicit procedures form a part of a linear equations solver. Since the linear problems appear within time integration schemes iterative solvers are the most appropriate, thanks to the existence of a perfect initial guess in the form of the solution from the previous time step. There are three iterative solvers in the code: block Jacobi, block Gauss—Seidel and generalized minimal residual (GMRES). Since in our implementation single iterations of block solvers are also used in the GMRES algorithm we will describe only the latter.

The GMRES algorithm [7] is one of the most successful and widely used iterative methods for solving non symmetric systems of linear equations. The performance of the GMRES depends on the conditioning of a system of equations so usually it is used with left or right preconditioning. In the case of left preconditioning instead of solving the original system of equations $Ax = b$ (with A the global stiffness matrix, x the vector of unknown degrees of freedom and b the global load vector), the preconditioned system $M^{-1}Ax = M^{-1}b$ is solved. The preconditioning matrix M should be easily invertible and designed in such a way that the preconditioned system has better convergence properties than the original one (the condition number of the product $M^{-1}A$ closer to one). The preconditioned GMRES algorithm schematically looks as follows:

set an initial guess x_0
repeat until convergence
  compute the residual of the preconditioned problem: $r_0 = M^{-1}(b - A x_0)$
  normalize the residual: $\bar{r}_0 = r_0 / \|r_0\|$
  for $i = 1, 2, \ldots, k$
    compute matrix-vector product: $\hat{r}_i = M^{-1} A \bar{r}_{i-1}$
    orthonormalize $\hat{r}_i$ *wrt all previous* $\bar{r}_j$, $j=1, \ldots, i-1$ *by the modified Gram-Schmidt procedure obtaining* $\bar{r}_i$
  solve the GMRES minimization problem to get the approximate solution
  check convergence, if attained leave GMRES

In the restarted version, the number of Krylov space vectors is limited to some small number (in our case $k = 10$) and the initial guess for restarts is taken as the current approximate solution.

Preconditioning of the GMRES algorithm can be efficiently achieved by using any of the basic iterative methods such as the block Jacobi or block Gauss-Seidel methods [8]. In such a case single iterations of block methods are used to perform the most time consuming parts of the GMRES involving matrix-vector products $M^{-1}(b - Ax_0)$ and $M^{-1} A \bar{r}_{i-1}$ [9]. The iterations consist of loops over all finite element nodes (patches of elements) for which local problems, with patch stiffness matrices and load vectors, are solved with values on the boundary of patches taken as Dirichlet boundary conditions.

3.1 Adaptivity

Mesh adaptivity constitutes important step in achieving high efficiency of numerical codes. Drastic reductions of the number of unknowns (finite element nodes) , as compared to the solutions of the same accuracy but on the uniform grids, can be obtained when mesh is refined only in certain indicated regions. A crucial ingredient is to base refinements on a reliable indicator that, in turn, should be based on proven a-posteriori error estimates. Unfortunately there are no error estimates for the compressible Navier-Stokes equations and one has to use indicators based either on approximate equations or some physical considerations.

In the code, adaptations were based on an indicator [10]:

$$\varepsilon = h^2 \cdot \boldsymbol{f}_{k,k}^T \boldsymbol{A}_0^{-1} \boldsymbol{f}_{k,k}$$

which in turn is based on an error estimate for scalar convection-diffusion problems [11]. Refinement-unrefinement strategy consists in dividing elements for which e was greater than some assumed limiting value and clustering back elements, resulting from the division of the same element, for which the sum of all indicators is smaller than some percentage (usually 20%) of the limiting value.

The code uses linear triangular elements. Mesh modifications are achieved by divisions of triangles into four sub-triangles, that maintain a regular shape of elements but requires the introduction of constrained (hanging) nodes [12]. Their existence complicates the creation of patches. Meshes with hanging nodes are called irregular.

3.2 Parallel algorithms

Parallelization of all presented solvers (direct for explicit strategy and iterative for implicit strategies) is based on distributing the data concerning the finite element mesh among different processing units of a system. It is assumed that each processing unit consists of one processor and a local memory. Each processor has access only to its local memory and can explicitly exchange data with other processors.

At the beginning of computations the mesh is divided into subdomains and each subdomain is assigned to a given processor. Several subdomains can be assigned to

one processor but an optimal situation occurs when the correspondence between subdomains and processors is one-to-one. In the case of different processors in a system with different processing powers a special algorithm for load balancing, based on differentiating the size of subdomains, is used [13].

Each processor recreates data structure concerning its subdomain and creates patches for all owned nodes. Patch stiffness matrices and load vectors are computed and explicit or implicit procedures of solvers performed. In all solvers after each loop over local patches processors exchange information on nodes on intersubdomain boundaries. For the explicit solver there is only one exchange of data per time step, but time steps are very frequent, for block Jacobi and block Gauss-Seidel solvers the number of exchanges is equal to the number of iterations, for the GMRES the number of iterations is equal to the dimension of the Krylov space, which in the inexact Newton algorithm is limited to some small number (in our case 20). However, in each iteration of GMRES there are several other procedures (computing vector norms and scalar products) that require interprocessor communication. Thus all methods are sensitive to the speed of interprocessor communication.

3.3 Mesh partition

The optimal domain decomposition, in the finite element context equivalent to partitioning the mesh, should lead to the minimal execution time for the whole problem solved by the finite element method. The partitioning process must optimize load balance and minimize interprocessor communication. Each subdomain should contain the number of degrees of freedom proportional to the computational power of processors (which is usually different as long as a cluster of workstations is considered) and a minimal number of interface nodes.

In the code an algorithm for the mesh partition based on the advancing front of nodes is used [14] wieden. Subdomains are created sequentially, each one by starting with a node chosen from the initial front (group) of nodes. Then all elements sharing the node are added to the subdomain and all vertices of elements are incorporated into the front. Further nodes are added to the subdomain from the front according to certain geometrical or topological criteria. Nodes and elements are added to the subdomain until the number of nodes reaches some specified limit. The front at the end of creation of one subdomain forms the initial front for the next subdomain.

The algorithm is flexible allowing for creation of subdomains with either required shapes or other geometrical features. It is also robust and works for arbitrary domains and finite element meshes, including irregular meshes with hanging nodes.

3.4 Implementation

The code is written in C language and uses typical for C features, such as defined types and dynamic memory allocation. The data concerning elements and nodes are stored in arrays of structures. During adaptations new structures are

allocated for new nodes, while for nodes deleted from the mesh only data are erased and the structure becomes available for new created nodes.

Due to the importance of patch computations in all described algorithms patches of elements have their own data type:

```
typedef struct {
int Nreles; int *Eles;
int Nrnoac; int *Noac;
int Nrnoin; int *Noin;
double **Ain;
double *Rin;
} PATCHES;
```

where:

`Nreles, Eles` — number and list of patch elements,

`Nrnoac, Noac` — number and list of patch internal nodes,

`Nrnoin, Noin` — number and list of patch boundary nodes,

`Ain` — assembled patch stiffness matrix,

`Rin` — assembled patch load vector.

The exchange of data between subdomains is based on arrays storing, in a suitable order, numbers of all nodes belonging to the overlap between subdomains. Due to different handling of four groups of nodes each subdomain creates four arrays:

`nodexch1` — internal nodes owned by a given subdomain,

`nodexch2` — internal nodes owned by other subdomains,

`nodexch1`— internal nodes on the boundary of other subdomains,

`nodbond` — boundary nodes (always owned by other subdomains).

The necessary differentiation between nodexch1 and nodexch2 nodes comes from the requirement that each node should possess only one subdomain that "owns" it. This requirement comes in turn from the parts of algorithms where each node must be considered only once (e.g. computing of vector norms for vector of unknowns).

Message passing between different processing units is implemented using the Parallel Virtual Machine (PVM) software. PVM consist of a Unix deamon controlling the exchange of data between different operating systems of different processors and C or Fortran routines, called from application programs, that perform all tasks connected with message passing. Additionally, there is a user interface, in the form of a terminal console, to create or modify a virtual machine and start or stop PVM applications.

The practical realization of the whole computation in the program is achieved using master–slave paradigm. There exists one master program (in PVM nomenclature master task) that controls the solution procedure and several slave

programs (tasks) performing in parallel most calculations. The master program reads input data, including control parameters, partitions the mesh and sends data to slave tasks. Slave tasks perform all the steps of solver algorithms and communicate with the master program only when dumpout of data or adaptation of the mesh is necessary.

4. Numerical examples

We present the capabilities and performance of the code for two numerical examples, one for inviscid and one for viscous flow calculations. Inviscid model is obtained by neglecting viscous and heat fluxes, together with corresponding matrices $\boldsymbol{K}_{ij}^{m}$, in equations (1) and algorithms (2) and (3) respectively. Boundary conditions for solid walls are also modified and enforce vanishing of the component of velocity normal to the wall only.

4.1 Inviscid flow around a bi–NACA0012 wing profile

The first example is the Mach 3 flow around a bi-NACA0012 wing profile [15] agard with zero angle of attack. The geometry of flow domain and the initial finite element mesh, with 821 nodes and 1501 elements, are depicted in Figure 1.

Figure 2 presents comparison of the convergence process for three time integration techniques — explicit, linear implicit (TG) and nonlinear implicit (NLE). Test runs were performed on a single MIPS R10000 (180 MHz) processor

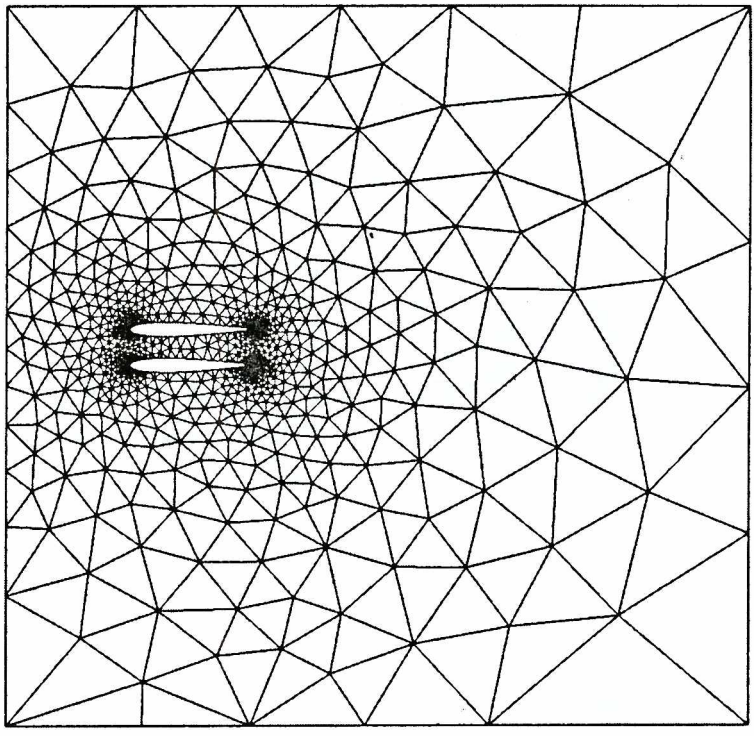


Figure 1. *Initial mesh and flow domain for bi-NACA0012 examples.*

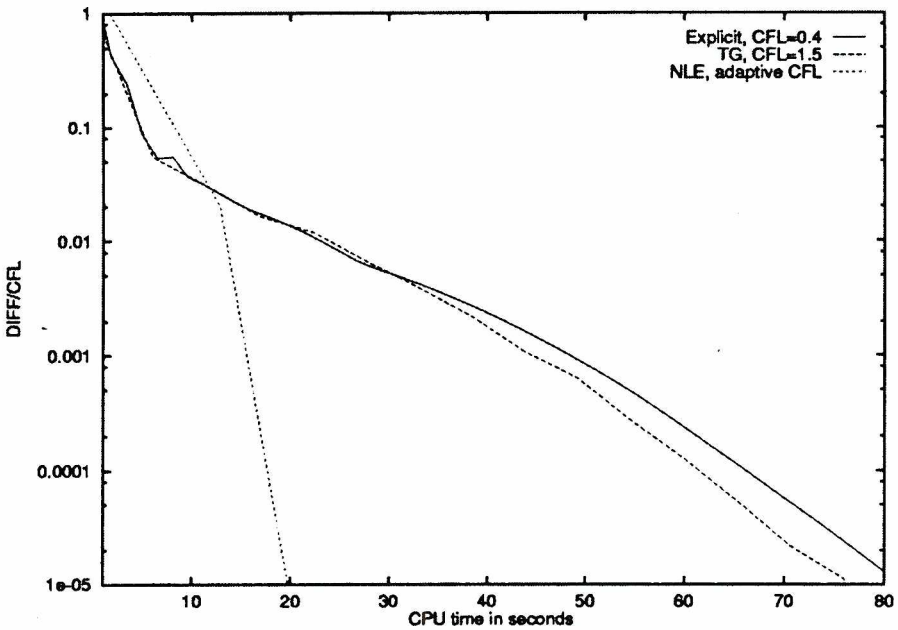


Figure 2. *Convergence for flow around a bi-NACA0012 profile — initial mesh*

$$(DIFF/CFL = max_{N=1,N_{NOD}} \frac{\rho_N^{n+1} - \rho_N^n}{CFL}).$$

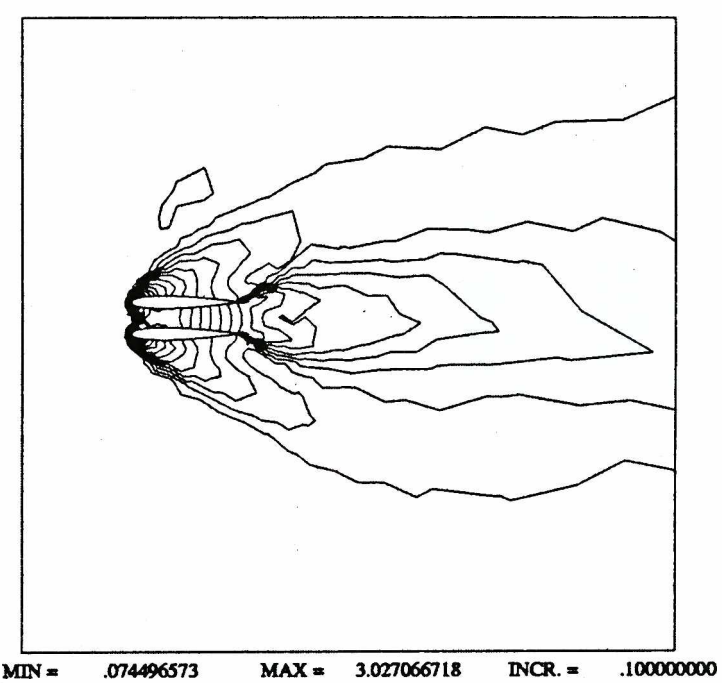


Figure 3. *Mach number contours for Mach 3 flow around bi-NACA0012 profile — initial mesh.*

of Silicon Graphics Origin 200 computer (≈100MFLOPS, SPECfpbase95 = 14.4). For each algorithm different values of CFL parameter were used. For explicit and implicit integration CFL was constant and equal to 0.4 and 1.5 respectively. These values were the biggest for which the algorithms remained stable. For the nonlinear (backward Euler) method adaptive strategy based on using two iterations of the inexact Newton method per time step and keeping (by the proper choice of CFL number) the convergence rate close to 0.5 was applied. The maximal CFL value obtained in this example was equal to 512. For all iterations the maximum norm of density changes at all finite element nodes was used to measure the error.

The solution obtained on the initial mesh is shown in Figure 3 in the form of Mach number contours.

After convergence the mesh is adapted using the value of the error indicator ε equal to 10^{-6} and computations performed on adapted mesh depicted in Figure 4. For further meshes we report results only for the most efficient nonlinear implicit algorithm.

For parallel computations meshes are divided into a number of subdomains, equal to the number of processors in the simulation. Different strategies of mesh partition can be used and we show in Figure 5 four example divisions of the mesh from Figure 4 into eight submeshes.

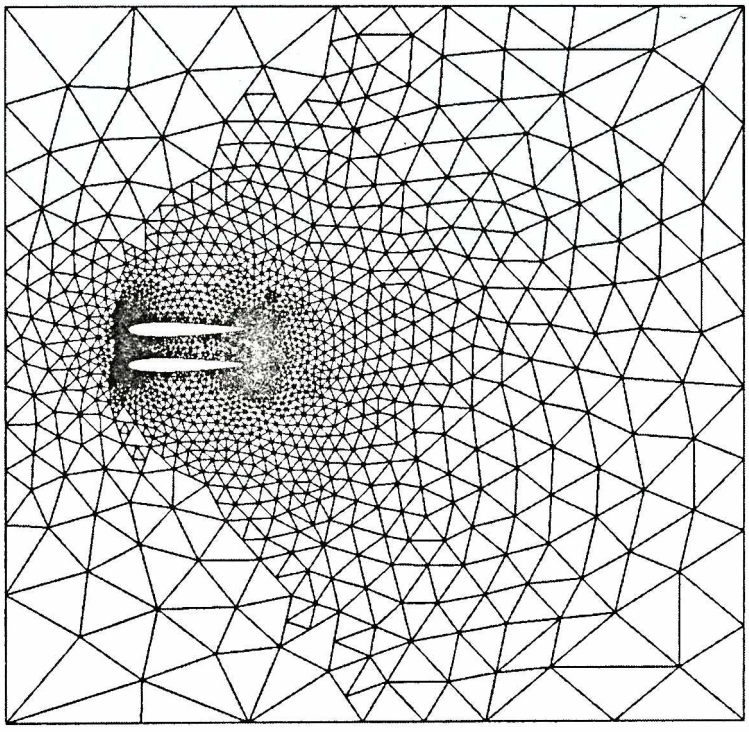
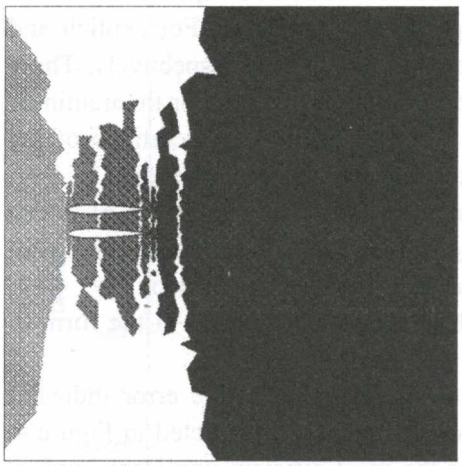
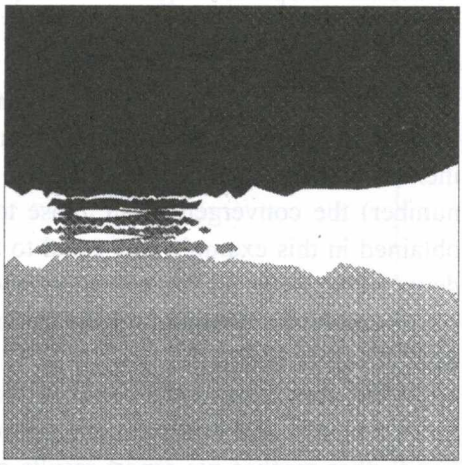


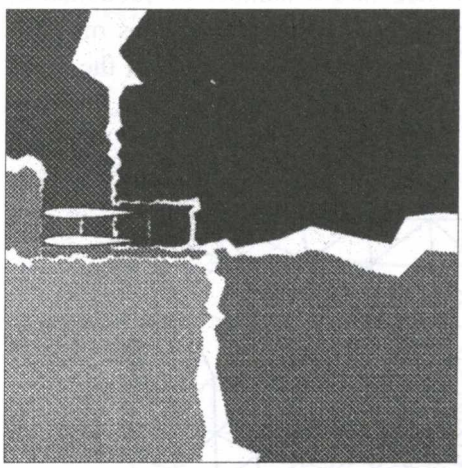
Figure 4. *First adapted mesh for bi-NACA0012 simulations.*



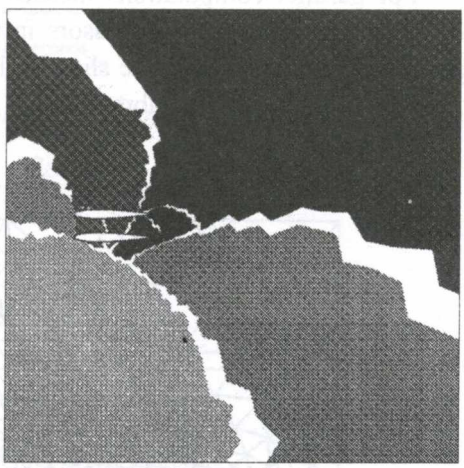
a) vertical subdomains



a) horizontal subdomains



a) square subdomains



a) oval subdomains

Figure 5. *Different mesh partitions for bi-NACA0012 mesh with one level of refinement.*

They correspond to strategies of creating: vertical, horizontal, square or oval subdomains respectively. In each case it was assumed that processors are of the same computational power so the subdomains should be of the same size (measured by the number of nodes) for the best load balance. However, due to the existence of the overlap between subdomains (white regions in Figure 5) different strategies resulted in different average numbers of nodes in subdomains: vertical — 428, horizontal — 437, square — 389, oval — 387. A smaller number of nodes indicates shorter length of intersubdomain boundary (again measured in number of nodes) and, in consequence, less interprocessor communication. In all simulations we used decomposition into oval subdomains, default in the code.

The procedure of mesh adaptation can be repeated several times. For each mesh, the computations are continued until a steady state is reached. Mesh partition and patch creation is performed only once for each mesh. Table 4.1 and Figure 6 show parallel performance of the code for the third refined mesh with 21515 nodes and 45814 elements.

Table 4.1 *Averaged results for one time step parallel computations on the mesh with 21515 nodes for an HP Exemplar SPP1600 computer.*

No. processors	*Execution time*	*Speed up*	*Efficiency*
1 (1 node)	112.97		
2 (1 node)	57.98	1.95	97.5%
4 (1 node)	29.56	3.82	95.5%
4 (2 nodes)	30.91	3.65	91.5%
8 (2 nodes)	16.23	6.96	87%
12 (3 nodes)	10.88	10.38	86.5%
16 (4 nodes)	8.91	12.68	79%

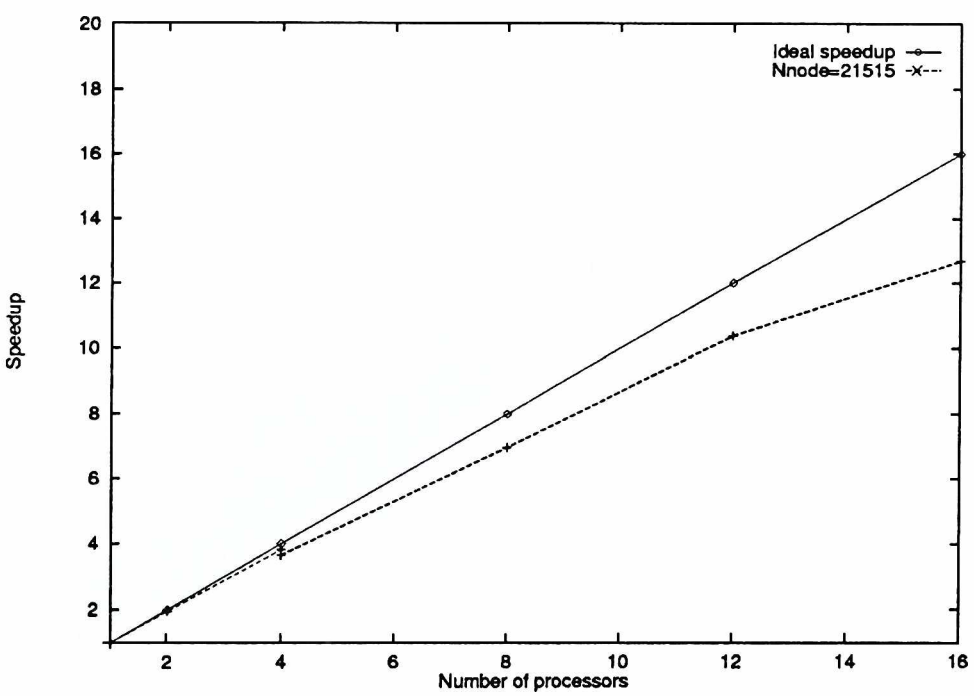


Figure 6. *Parallel speedup on the mesh with 21515 nodes for an HP Exemplar SPP1600 computer.*

The table presents execution times, speedup and efficiency for different number of processors used. The results correspond to one time step computations and are obtained by averaging over the whole simulation. Speedup is defined as the ratio of the sequential execution time to the parallel execution time and efficiency (expressed in percentage) as the ratio of speedup to the number of processors. We do not take into account changes in GMRES or Newton convergence resulting from parallelization of the algorithm (negligible in all cases), so the efficiency represents only the efficiency of numerical implementation. The reported results were obtained on an HP Exemplar SPP1600 multiprocessor system in a dedicated mode. HP Exemplar SPP1600 has a modular architecture and consists of a certain number of computational nodes, each of which comprises 8 processors. When computations are performed on one node, only the local node memory is used, when more nodes are used the use of slower global memory is necessary. For the purpose of comparison processors worked in a dedicated mode, however, buses were shared with other users, which affected the overall performance and made it random within the range of several percents. The sensitivity of the algorithm to interprocessor communication speed can be visible for two cases with four processors. The use of two computational nodes with slower inter-node communication decreased the efficiency of computations by 4%.

The results for the final for this example fifth refined mesh with 102822 nodes and 238812 elements are shown in Figure 7 and Figure 8. Figure 7 presents a detail of the mesh near the trailing edge and Figure 8 shows the final Mach number contours obtained in that simulation.

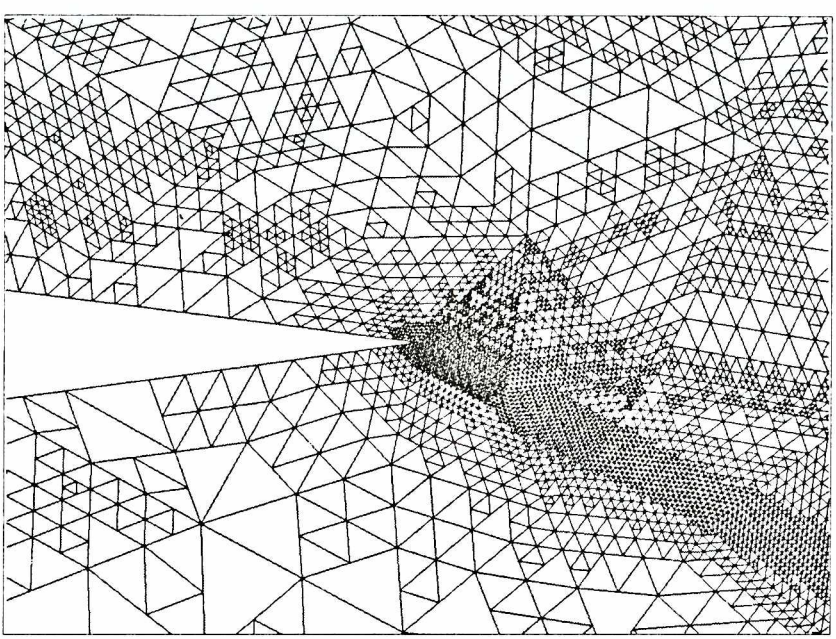


Figure 7. *Detail of the fifth adapted mesh for bi-NACA0012 simulations.*

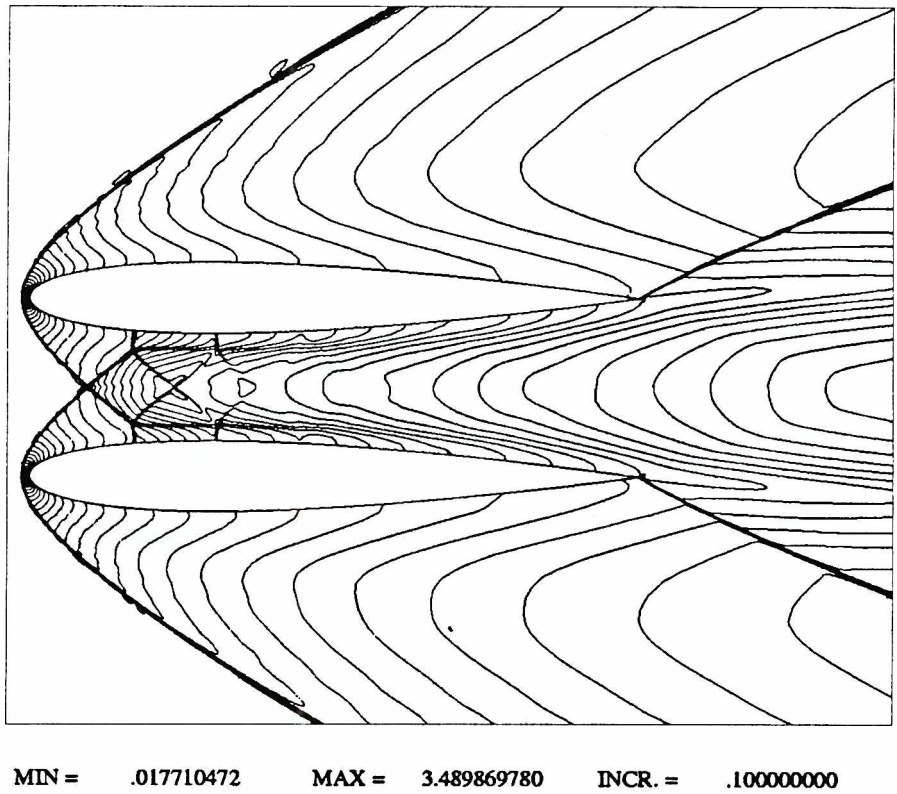


Figure 8. *Mach number contours for Mach 3 flow around bi-NACA0012 profile — mesh with five levels of refinement.*

4.2 Viscous flow over a flat plate

The next example is a viscous flow with Mach number 3 and Reynolds number 1000 over a flat plate. The definition of the problem is shown in Figure 9. Figure 10 depicts the initial mesh and Figure 11 presents density contours obtained on that mesh.

The strategy of refining the mesh and converging the solution to the steady state is repeated three times. The convergence on four consecutive meshes for the nonlinear algorithm is presented in Figure 12.

Figure 13 depicts the final mesh, density contours obtained on it are shown in Figure 14.

For the last mesh with 14262 nodes and 30771 elements Table 4.2 shows the performance for parallel execution on HP Exemplar SPP1600 computer. The graph of obtained speedup is shown in Figure 15.

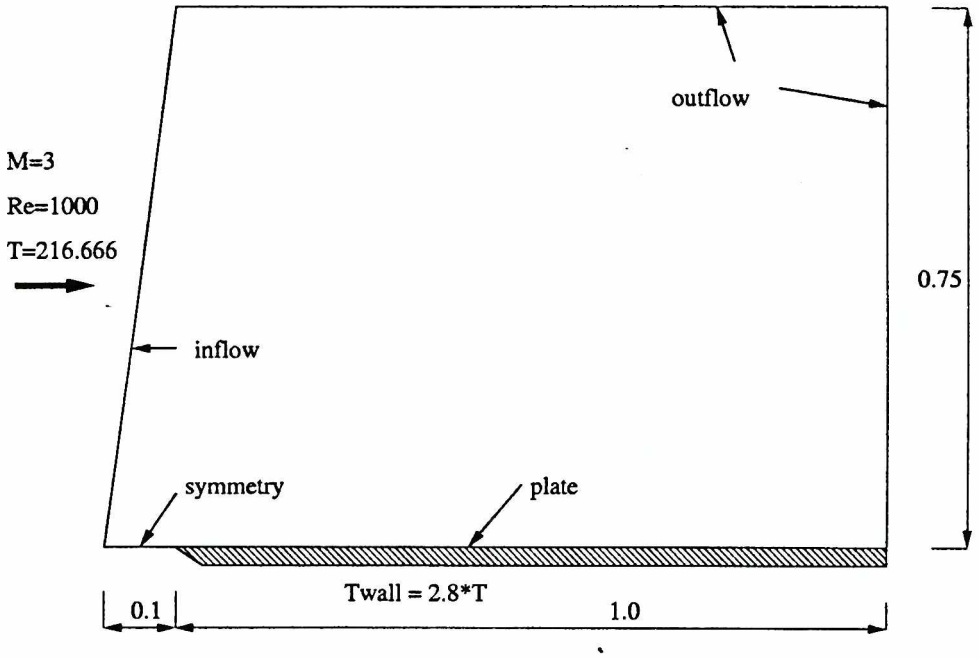


Figure 9. *Flat plate problem — geometry and definition.*

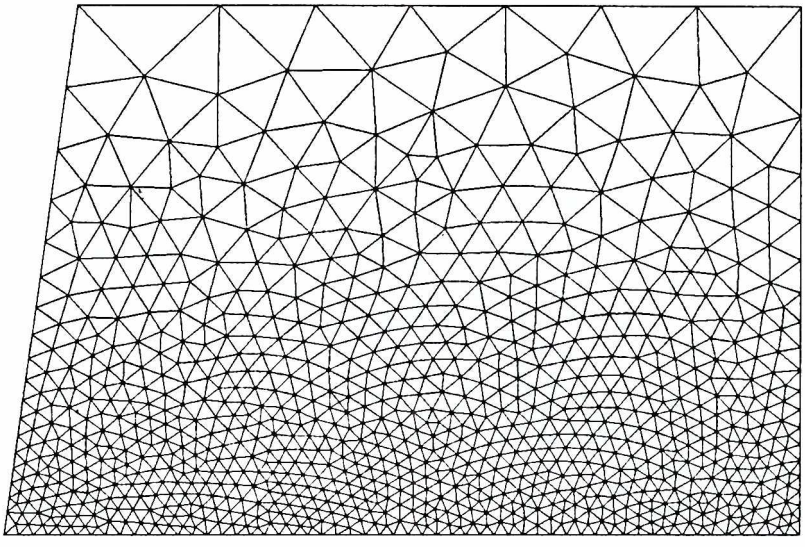


Figure 10. *Flat plate problem - initial mesh.*

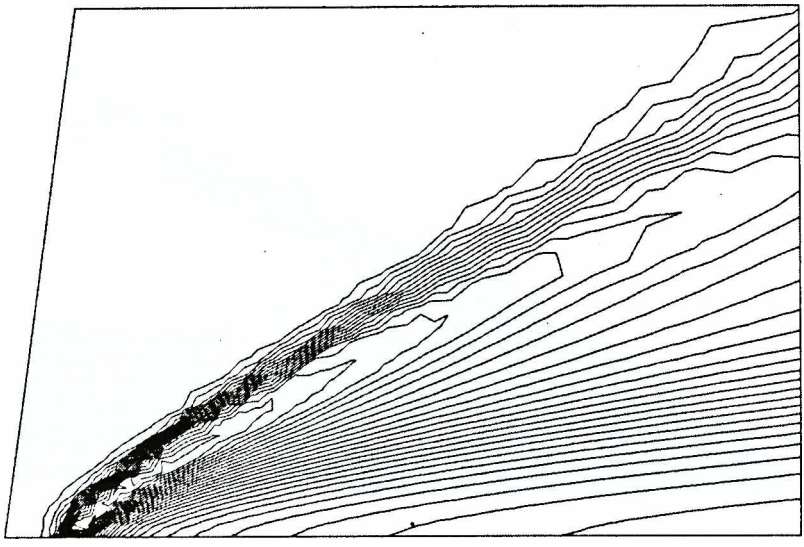


Figure 11. *Flat plate problem — Mach number contours on the initial mesh.*

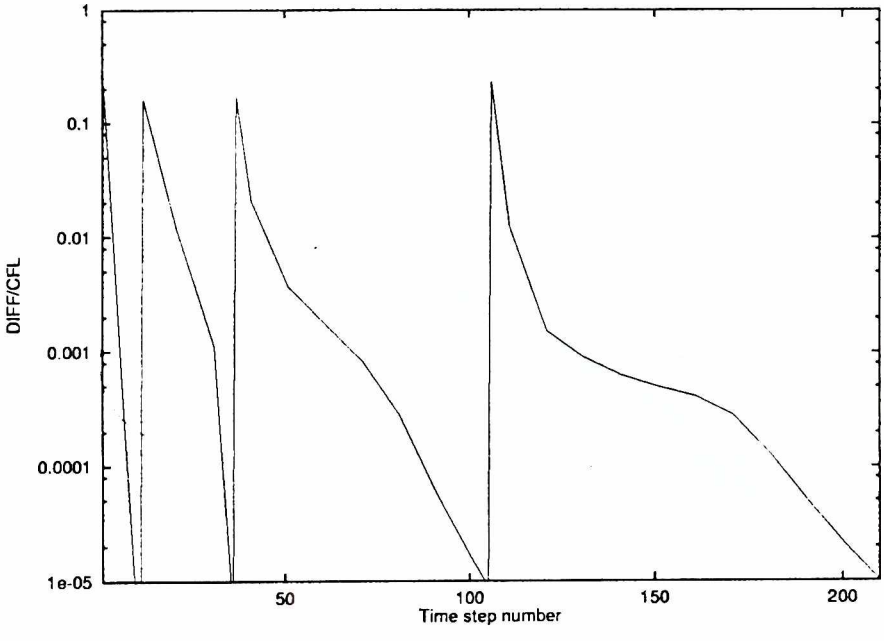


Figure 12. *Convergence for flow over a flat plate on four consecutive adapted meshes.*

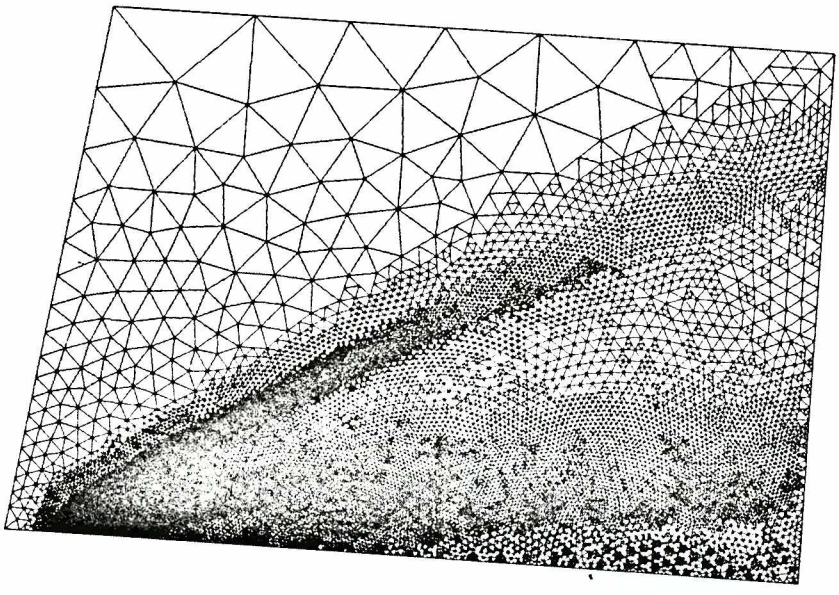


Figure 13. Flat plate problem — mesh with three levels of refinement.

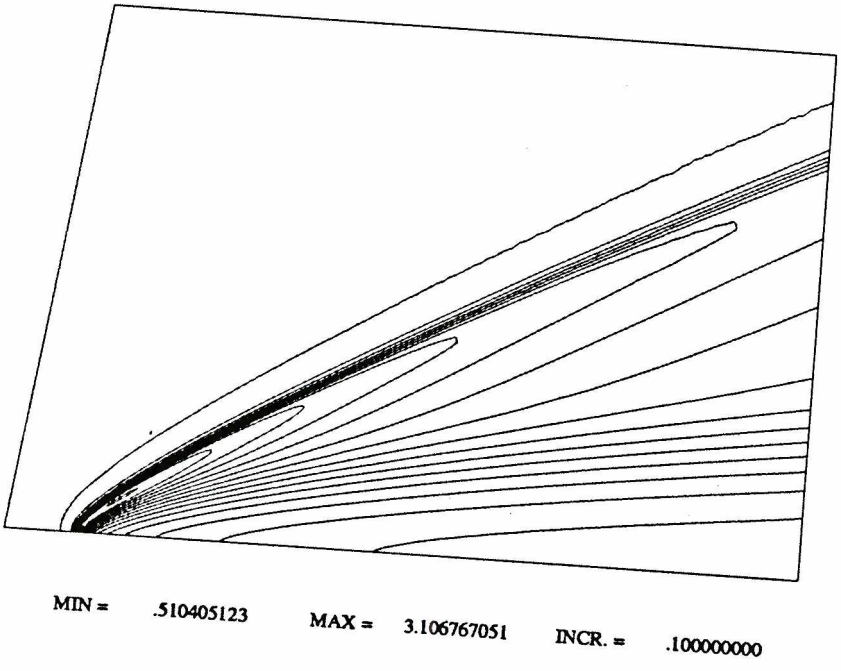


Figure 14. Flat plate problem — Mach number contours on the mesh with three levels of refinement.

Table 4.2 Averaged results for one time step parallel computations on the mesh with 14262 nodes for an HP Exemplar SPP1600 computer.

No. processors	Execution time	Speed up	Efficiency
1 (1 node)	73.64		
2 (1 node)	37.98	1.94	97%
4 (1 node)	19.73	3.73	93%
4 (2 nodes)	20.37	3.62	90.5%
8 (2 nodes)	10.40	7.08	88.5%
12 (3 nodes)	7.26	10.15	84.5%
16 (4 nodes)	5.53	13.31	83%

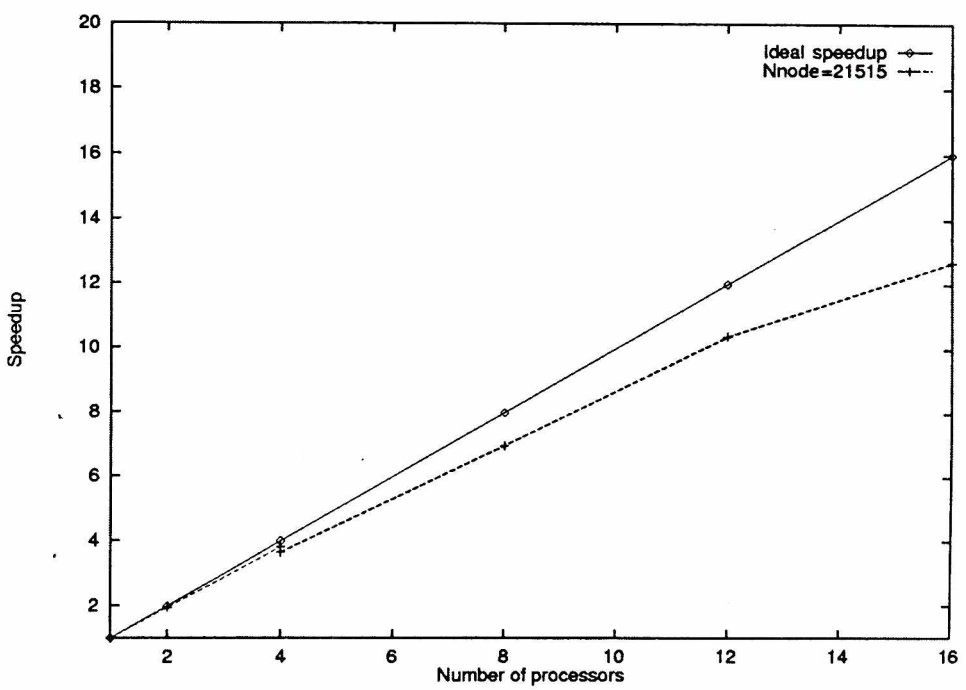


Figure 15. Parallel speedup on the mesh with 14262 nodes for an HP Exemplar SPP1600 computer.

5. Conclusions

The presented finite element code allows for efficient large scale parallel simulations of compressible flows. Still there is a room for further improvement. On the algorithmic side turbulence modeling has to be included, as well as special

error estimates for boundary layers. Finite element techniques will be extended by including hybrid adaptive meshes, with triangular and quadrilateral elements combined together, and anisotropic refinements, especially in boundary layers. Further work will also include the development of iterative solvers operating on hierarchical multi-level grids for better convergence. The described code forms a basis for this future development, in the form of the data structure, basic finite elements techniques including adaptivity, parallel linear equations solvers and algorithms for approximating the compressible Navier-Stokes equations.

Acknowledgment

The support of this work by the Polish Committee for Scientific Research under Grant 8 T11F 003 12 is gratefully acknowledged.

References

[1] Hirsch C., *Numerical Computation of Internal and External Flows*, Wiley, Chichester, 1988

[2] Hansbo P., *Explicit Streamline Diffusion Finite Element Methods for the Compressible Euler Equations in Conservation Variables*, Journal of Computational Physics, **109**, 274–288, (1993)

[3] Brooks A. N., Hughes T. J. R., *Streamline upwind/Petrov-Galerkin formulations for convection dominated flows wuth the particular emphasis on the incompressible Navier-Stokes equations*, Computer Methods in Applied Mechanics and Engineering, **32**, 199–259, (1982)

[4] Carette J. C., Deconinck H., Paillere H., Roe P. L., Multidimensional upwinding: its relation to finite elements, International Journal for Numerical Methods in Engineering, **20**, 935–955, (1995)

[5] Shakib F., Hughes T. J. R., Johan Z., *A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier-Stokes equations*, Computer Methods in Applied Mechanics and Engineering, **89**, 141–219, (1991)

[6] Banas K., Demkowicz L., *Entropy controlled adaptive finite element simulations for compressible gas flow*, Journal of Computational Physics, **126**, 181–201, (1996)

[7] Saad Y., Schultz M., *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal of Scientific and Statistical Computing, **7**, 856–869, (1986)

[8] LeTallec P., *Domain decomposition method in computational mechanics*, Computational Mechanics Advances, J.T.Oden ed., North Holland, Amsterdam, 1994

[9] Banaś K., Plażek J., *Dynamic load balancing for the preconditioned GMRES solver in a parallel, adaptive finite element Euler code*, in Proceedings of the Third ECCOMAS Computational Fluid Dynamics Conference, 9–13 September 1996, Paris, France, eds., J.-A. Desideri, C.Hirsch, P.Le Tallec, M.Pandolfi, and J.Periaux, pp. 1025–1031, Chichester, (1996), Wiley

[10] Hansbo P., Johnson C., *Adaptive streamline diffusion method for compressible flow using conservation variables*, Computer Methods in Applied Mechanics and Engineering, **87**, 267–280, (1991)

[11] Eriksson K., Johnson C., *Adaptive streamline diffusion finite element methods for stationary convection diffusion problems*, Mathematics of Computation, 60, 167–188, (1993)

[12] Demkowicz L., Oden J. T., Rachowicz W., Hardy O., *Towards a universal rm h-p adaptive finite element strategy*, Part.1 Constrained approximation and data structure, Computer Methods in Applied Mechanics and Engineering, **77**, 79–112, (1989)

[13] Banaś K., Plażek J., *Parallel h-adaptive simulations of inviscid flows by the finite element method*, Mechanika Teoretyczna i Stosowana, **35**, 249–262, (1997)

[14] Plażek J., Banaś K., Kitowski J., Boryczko K., *Exploiting two-level parallelism in FEM applications*, in Proceedings of the International Conference on High Performance Computing and Networking, Vien, Austria, April 1997, eds., B. Hertzberger and P. Sloot, pp. 272–281, Springer

[15] Dervieux A., v.Leer B., Periaux J., Rizzi A. (eds.), *Numerical simulation of compressible Euler flows*, Vieweg, Braunschweig, 1989