FlowVision software: numerical simulation of industrial CFD applications on parallel computer systems.

Andrey A. Aksenov[1*], Sergey A. Kharchenko[2], Vladimir N. Konshin[2], Victor I. Pokhilko[3]

[1]Institute for Computer Aided Design of RAS; 2-nd Brestskaya str., 19/18, Moscow, Russia
[2]Dorodnicyn Computing Centre of RAS, Vavilov str., 40, Moscow, Russia
[3]Institute for Mathematical Modelling of RAS, Miusskaya sq,. 4a, Moscow, Russia

Abstract

FlowVision is CFD code for modeling industrial three-dimensional turbulent fluid flows [1].

FlowVision is based on the finite-volume method and rectangular adaptive mesh with local refinement. **FlowVision** uses a subgrid geometry resolution to approximate the curvilinear shape of a computational domain with high accuracy. This technology provides importing a geometry from CAD packages and exchanging data with FEA systems.

The presented technology was implemented as a portable parallel software using the MPI message passing interface. It has been proven to be very reliable when solving industrial CFD problems. Results of numerical simulations of time dependent flow using network of workstations are presented.

Introduction

FlowVision is a CFD tool designed for accurate and reliable modelling of industrial flow problems on parallel computer platforms. This multi-disciplinary investigation brings together the advanced results from the fields of mathematical simulation, approximation theory, numerical linear algebra, and parallel computing.

The basic features of the numerical finite volume scheme used in **FlowVision** are presented below:
second order truncation error;
spectral-like resolution, i.e., low phase errors and a dissipative mechanism which suppresses or filters only spurious solution modes;
positive definiteness and compatibility of the resulting discrete systems;
discrete conservation, i.e., the approximation is based on the integral form of the system of the conservation laws;
high accuracy and stability on rectangular grid with local refinement;
a stable cell-centered formulation based on the high order stabilization;
fully implicit time stepping with Newton's method for the solution of resulting nonlinear systems.

FlowVision technique has been successfully used for many industrial applications. The numerical simulating a time-dependent flow around aircraft is presented as an example. The streamlines are shown in Fig.1a and the part of grid with local refinement is shown in Fig.1b.

The simulation was conducted without simplifying assumptions on the aircraft geometry on a single CPU as well as on the network of workstations.

* E-mail: anda@tesis.com.ru

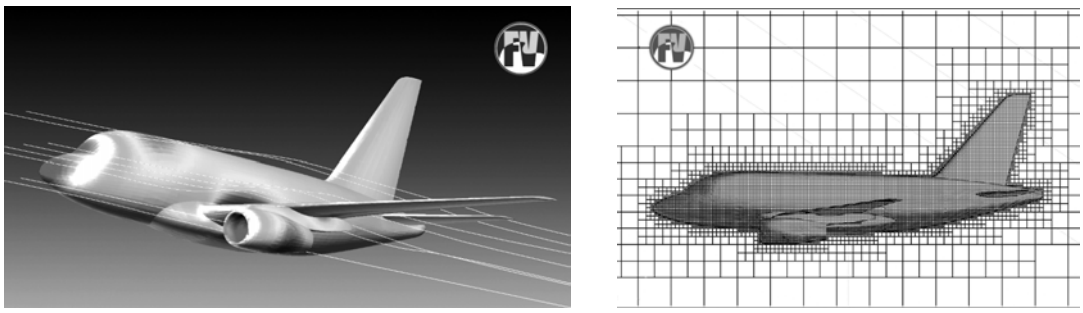

Fig. 1. Flow near aircraft (a) and grid (b).

Numerical methods

Consider the nonlinear problem described by the Navier-Stokes equations. In the operator form the discrete problem can be written as follows

$$\mathcal{L}u = f, \tag{1}$$

where the nonlinear discrete operator $\mathcal{L}$ corresponds to the approximation of the Navier-Stokes equations.

The solution method for the system (1) is the Newton-like method which can be written in the following form

$$\mathcal{L}_f(u^{k+1} - u^k) + \mathcal{L}u^k = f, \tag{2}$$

here k is the iteration number corresponding to the k –th time step and the linear operator $\mathcal{L}_f$ is a suitable linearization of $\mathcal{L}$. In order to compute the iterate u^{k+1} from (2) it is necessary to solve the pre- and post- conditioned linear system of equations

$$\mathcal{L}_f u^{k+1} = r, \tag{3}$$

where $r = f + \mathcal{L}_f u^k - \mathcal{L}u^k$.

From the algorithmic point of view, the solution method is split into three major blocks:

1) the computation of residuals, i.e., the computations of vectors $\mathcal{L}u^k$ and $\mathcal{L}_f u^k$;
2) generation of matrix $\mathcal{L}_f$;
3) iterative solution of linear system (3).

FlowVision generates different types of the systems of linear equations. Consider the system of linear equations in the general formulation:

$$Ax = b, \tag{4}$$

where A is large sparse matrix with irregular structure of non-zero elements. Irregularity of the sparsity structure of matrix A is induced by the local grid refinement.

The matrix A may have different algebraic properties. In particular, **FlowVision** generates symmetric positive matrices ($A = A^T > 0$), unsymmetric matrices ($A \neq A^T$), block 3x3 unsymmetric matrices ($A_{3\times3} \neq A_{3\times3}^T$). Matrices of the first type are generated for elliptic equations, matrices of the second type are generated for scalar parabolic equations and matrices of the third type – for vector parabolic equations (for example, Navier-Stokes vector equations).

Incomplete Cholessky preconditioned conjugate gradient method is used to solve linear systems with the matrices of the first type. Incomplete LU and incomplete block LU preconditioned Lanczos method is used to solve linear systems with the matrices of the second and third type.

Parallel implementation technique for the second type of matrices is described below, while for the first and for the third types of matrices the parallel implementation is similar.

Parallel implementation

Consider the main factors which influence the parallel efficiency [2].

Load balancing problem. Uniform load of the processes substantially affects the overall efficiency of the parallel implementation such as parallel scalability. We use special techniques

to balance the load. In particular, we have used MeTiS [3] software to generate the partitioning of the computational nodes over the processes. Partitioning that is computed by MeTiS splits all nodes of the matrix graph into p subgroups, here p is the number of processes. Each group of nodes have almost equal number of nodes, and the number of edges of the graph that have nodes from different groups is as minimal as possible. Each group of nodes is assigned to some process. Partitioning of the nodes induces the ordering of the nodes in which the matrix takes the following block bordered form:

$$A = \begin{bmatrix} A_1 & & 0 & C_1 \\ & \ddots & & \vdots \\ 0 & & A_p & C_p \\ B_1 & \cdots & B_p & D \end{bmatrix}. \tag{5}$$

We also applied RCM [4] profile optimization algorithm to optimize the profile of the diagonal subblocks. The sparsity patterns of the original and of the resulting reordered coefficient matrices for p = 2, 4, 8 for the model problem described above are presented on figures below.

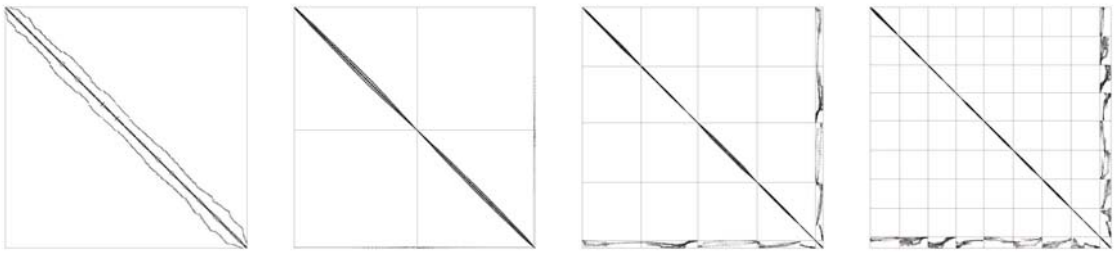

Fig. 2. Original and reordered sparsity patterns of the matrix for p =2,4,8.

Serial part of the code. The serial part of the code should be as minimal as possible. Major serial part of the code in our implementation is related to the computations with the nodes from the bordering. Thus, by minimizing the number of bordering nodes we minimize as much as possible the serial part of the code.

Communication costs. The above described mathematical technologies were implemented on distributed memory parallel computer in a message-passing style. In this case the total wall time of the application consists of the computation and communication costs. In order to get better parallel scalability it is necessary to minimize communication costs. Major communication costs in our implementation connected with the data related to the nodes from the bordering. Again, by minimizing the number of nodes in the bordering we minimize as much as possible the communication costs of the algorithm. Moreover, we extensively use the overlap of communications and computations whenever possible.

Distribution of the data between processes. The data of the linear system are distributed among processes according to the block bordered form (5) as follows. The data of each diagonal block row of the permuted coefficient matrix and the corresponding data of the right hand side and solution vectors are assigned to the corresponding process. The data of the bordering block row and the corresponding right hand side and solution vectors data are assigned to the process with identification number $id = 0$. During incomplete LU preconditioner generation stage the L and U data of the preconditioner are distributed over the processes as follows. The data of each diagonal block row of U and the data of each diagonal block column of L are assigned to the corresponding process. Diagonal subblock corresponding to the bordering are assigned to the process with $id = 0$.

The parallel computation of the ILU preconditioner is implemented as follows. At the first stage each process computes ILU factorization of its block row/block column according to the Dongarra/Eisenstadt algorithm separately without synchronizations or data exchanges. At the second stage each process sends the computed bordering data to the process with $id = 0$. And at

the third stage the process with $id = 0$ performs computation of the Schur complement for the bordering diagonal subblock of the reordered coefficient matrix with the use of received data, and then computes its ILU factorization.

Taking into account the distribution of the coefficient matrix over the processes, the parallel generation of the ILU preconditioner requires to exchange before factorization only small size data from the bordering part of the coefficient matrix.

After completion of the parallel incomplete factorization redistribution of some part of the preconditioner data among processes is performed. As a result, diagonal block rows of L and diagonal block columns of U are assigned to the process to which the corresponding diagonal block row of the coefficient matrix is assigned to. Such redistribution of the preconditioner data requires to exchange the data corresponding to the bordering only.

Iterative solution of the linear system by the ILU preconditioned Lanczos algorithm requires at each iteration the following major computations: multiplication of the vector by the coefficient matrix and by its transpose, solution of the linear system with the lower (L and U^{T}) and the upper triangular (U and L^{T}) matrices, and computation of some scalar products and some vector updates.

During the iterative solution stage all vector data are distributed over the processes according to the distribution of the block rows of the coefficient matrix.

Parallel multiplication on the coefficient matrix and on transposed matrix requires to exchange only a small part of vector data. These data correspond to nonzero entries of the coefficient matrix in the off-diagonal part of the bordering block row/block column.

Parallel solution of the linear system with the lower triangular matrices (L and U^{T}) is performed as follows. At the first stage each process separately solves its own lower triangular subsystem corresponding to the diagonal subblock. After that each process sends some part of the obtained vector data to the process with id=0. At the final stage process with $id = 0$ updates its own vector data from the bordering using the received data, and then solves system with the lower triangular submatrix of L or U^{T} from the bordering diagonal block.

Parallel solution of the linear system with the upper triangular matrices (U and L^{T}) is performed as follows. At the first stage process solves system with the upper triangular submatrix of U or L^{T} from the bordering diagonal block. Then process with $id = 0$ sends a part of the solution vector data from the bordering to each process. And at the final stage each process updates its own vector data from the block diagonal part using the received solution vector data from the bordering, and then solves system with the upper triangular submatrix of U or L^{T} from the diagonal block.

Computation of the scalar products in the parallel environment is implemented as follows. First of all each process performs computation of its own part of the scalar product, and then the results obtained by all process are collected. All vector updates are performed locally.

Numerical results

All numerical experiments were performed under Windows 2000 operating system on the network of PC workstations (500MHz Pentium II processors) with slow network (100Mbit per second).

Figure 3 shows timing results up to 4 processors for the problem considered. The results indicate that the parallel algorithm is scalable for small number of processors. For larger number of processors it is necessary to use more complicated parallel algorithms, e.g., multilevel parallel algorithm, to get better parallel efficiency.

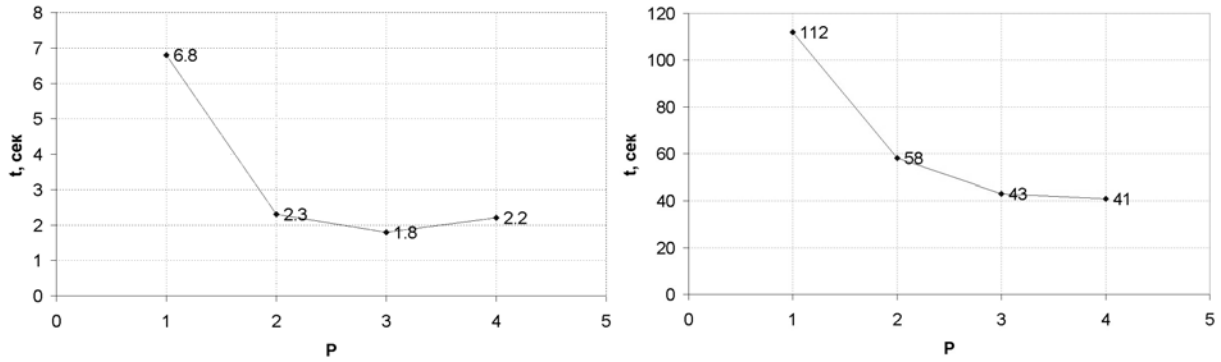

Fig. 3. Timing results for incomplete factorization (a) and iterative solution (b) stages.

Conclusion

FlowVision is proven to be reliable and efficient tool for simulating complicated 3D CFD problems. Parallel version of **FlowVision** demonstrates reasonable scalability on rather slow network of a small number of PC workstations. The basic approach implemented in **Flow-Vision** can be efficiently advanced to large number of processors by using multilevel parallel algorithms.

References

1. A.A.Aksenov, A.A.Dyadkin, V.I.Pokhilko. Overcoming of Barrier between CAD and CFD by Modified Finite Volume Method, Proc 1998 ASME Pressure Vessels and Piping Division Conference, San Diego, ASME PVP-Vol. 377-1, 1998
2. V.N.Konshin, V.A.Garanzha. Highly accurate numerical methods for incompressible 3D fluid flows on parallel architectures. In: Lecture Notes in Computer Science. Vol.1662. 1999. Pp. 68-76.
3. G.Karypis and V.Kumar. "MeTiS: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing ordering of sparse matrixes, version 4.0. 1998".
4. A. George, J.W. Liu, "Computer Solution of Large Sparse Positive Definite Systems", Prentice Hall, 1981.