Parallel Implementation of the Unified Flow Solver

S.A. Zabelok[1], V.V. Aristov[1], A.A. Frolova[1], V.I. Kolobov[2] and R.R. Arslanbekov[2]

[1]*Dorodnicyn Computing Center of the Russian Academy of Sciences, Ul. Vavilova - 40, Moscow, 119991, Russian Federation*
[2]*CFD Research Corporation, 215 Wynn Drive, Huntsville, Alabama 35805, USA*

Abstract. The parallelization of Unified Flow Solver (UFS), the software for modeling gas flows in a wide range of Knudsen numbers is described. Domain decomposition into arbitrary shape subdomains is implemented with different weights assigned to kinetic and continuum cells. Dynamic load balancing among processors is achieved to obtain good parallel performance. Space filling curves method is applied in the UFS in order to obtain optimal partitioning. MPI library is used to organize data exchanges between processors. Speedup for Navier-Stokes, Boltzmann and coupled regimes are considered.

Keywords: Boltzmann equation, hybrid schemes, parallel algorithms, dynamic load balancing.

PACS: 51.10+y, 05.20Dd

INTRODUCTION

The presence of rarefied and continuum domains is a typical feature of many complex gas flows. In rarefied domains, the mean free path of gas molecules is comparable or larger than a characteristic scale of the system. These domains are described by the Boltzmann kinetic equation for the particle distribution function. The continuum domains are best described by the Euler or Navier-Stokes (NS) equations in terms of average flow velocity, gas density and temperature. The development of hybrid solvers combining kinetic and continuum models has been an important area of research over the last decade (see [1, 2]).

The software product Unified Flow Solver (UFS), which is now under development, is considered in the present paper. The UFS is intended to model the flows in a wide range of the Knudsen numbers. In the regions of large local Knudsen numbers the Boltzmann equation is directly solved (for methods of direct solution of the Boltzmann equation see [3, 4] in order to obtain the correct flow pattern and Euler or Navier-Stokes equations are solved (using kinetic scheme, see [5, 6]) for small local Knudsen numbers for continuum parts of the computational domain. Several switching criteria for domain decomposition into kinetic and continuum regions have been proposed and coupling algorithm has been developed.

The open source GERRIS Flow Solver (GFS) [7] was selected as framework for the UFS. The original Gerris code contained a tree-based incompressible flow solver with a dynamically adaptive grid and support of complex solid boundaries with automatic locally refined mesh generation. Using the GFS framework, all the UFS components have been added: deterministic Boltzmann solver, compressible Euler and NS solvers based on kinetic scheme, practical criteria for domain decomposition and coupling kinetic and continuum solvers. The UFS can automatically generate Cartesian mesh around embedded solid boundaries defined through standard files, perform dynamic adaptation of the mesh to the solution and geometry, detect kinetic and continuum domains and select appropriate solvers based on continuum breakdown criteria. The UFS dynamically allocates and frees the memory for 3D distribution function in when the cell becomes a part of kinetic and continuum regions respectively.

The problem of parallelization is very important for every CFD code, which is supposed to work with 3D problems and complex geometries. For the UFS this problem is even more urgent, because Boltzmann solver deals with 6D phase space and requires huge computational resources and so, in order to use multiprocessor supercomputers, parallel processing must be implemented for the UFS. The initial version of the GFS code had possibility of parallel domain decomposition using cubical boxes or squares (in 2D). For parallel execution, the computational domain could be subdivided into several cubical (square) sub-domains, and a selected set of these

sub-domains assigned to different processors. Such decomposition is static (since it does not change during the computation) and rather inefficient. First, for the domain decomposition into the cubic sub-domains it is not always possible to achieve good load balance between the processors, even for the cases when the load is known for each computational cell, and the computational grid is also known. When adaptive grid is used, static domain decomposition can result in large load disbalance. Moreover, for the UFS, the computational load in each cell could vary by orders of magnitude depending on weather the cell is kinetic or continuum. For these reasons, it was decided to abandon the parallel option built into original GFS and develop new parallel capabilities using subdomains of arbitrary shape with dynamic load balancing between the processors depending on local grid refinement and different weight of kinetic and continuum cells.

PARALLEL ALGORITHM

First, the capability of performing computations in a selected part of the computational domain of arbitrary shape was implemented in UFS. This possibility allows using any domain decomposition algorithm in order to distribute the computational domain between processors. To illustrate details of the implementation, consider the procedure of accessing different cells in the GFS code illustrated in Figure 1. The computational grid in the GFS is generated by subsequent division of square boxes into smaller boxes with linear dimensions equal to half of the initial dimension (left part of Figure 1). The procedure of creating the computational mesh can be represented by a tree (the right part of Figure 1). The root of the tree (0^{th} level) corresponds to initial cube, the first level corresponds to 4 (in 2D) or 8 (in 3D) cubes obtained by division of the initial cube, etc. The computational cells correspond to leaves of the tree. The order of cell traversing is shown in Fig. 1 by dashed lines with arrows. All computation procedures are called only for the leaves of the tree.

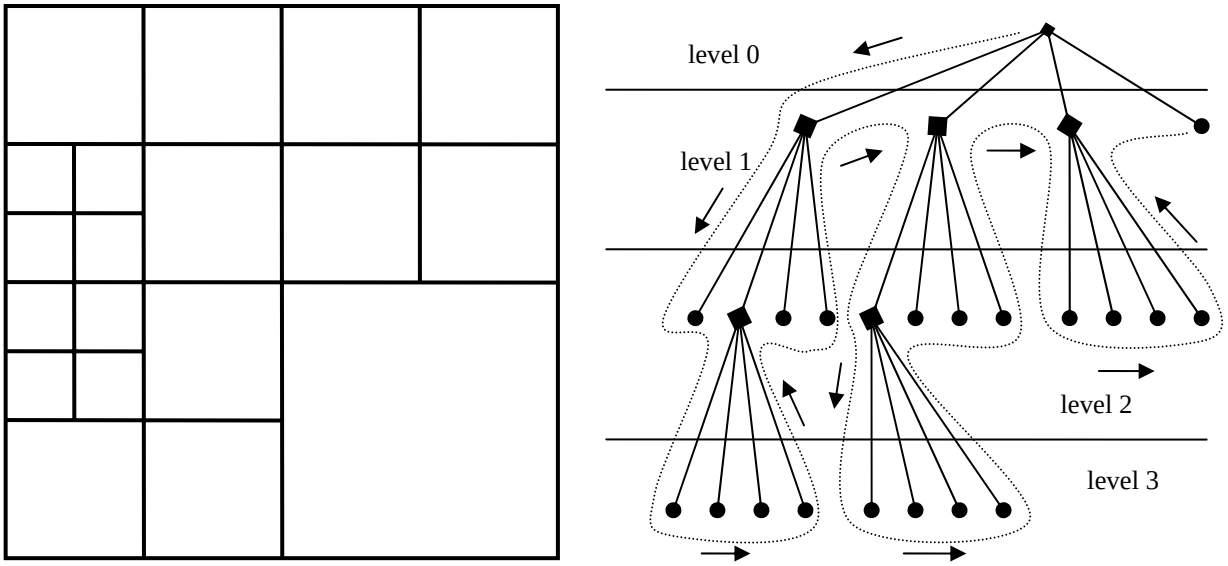

FIGURE 1. Computational mesh and the tree corresponding to it.

To perform computations only for a part of the domain (for instance, the sub-domain shown by blue color in Figure 2), one introduces a flag for each leaf to identify whether or not the cell belongs to the selected sub-domain. If a cell is a parent cell for a set of cells, it is flagged only if at least one child cell belongs to the selected sub-domain. After introducing flags, the procedure of cell traversing is modified in such a way as to visit only the cells belonging to the selected sub-domain. As shown on the right part of Figure 2, only the branches connected by solid lines are traversed. Moreover, it is necessary to specify boundary conditions at the boundary of the selected sub-domain. For this purpose, additional cells (hatched in Figure 2) – ghost cells – are used. The branches corresponding to the ghost cells are shown by dashed lines on the graph. The ghost cells are marked using a different flag, and can be traversed separately by the code.

The implementation of the approach described above enabled us to perform simulations only in selected parts of the computational domain. An additional outcome from this part of the work is the possibility to simulate one-dimensional problems (in the original GFS, these problems had to be solved as two-dimensional problems).

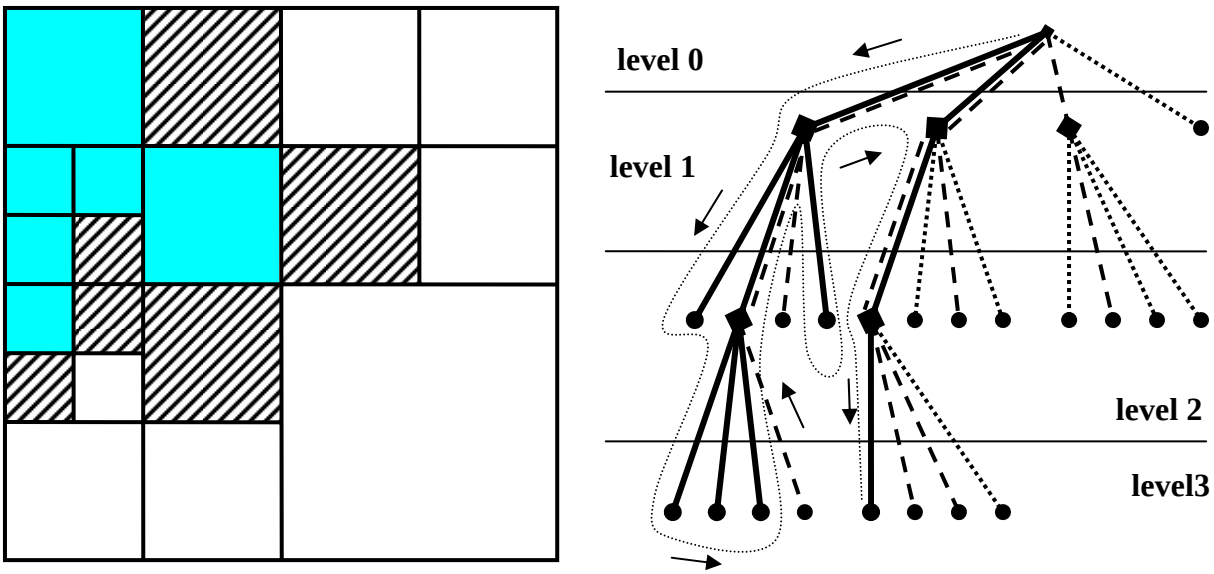

FIGURE 2. The part of computational domain and the sub-tree corresponding to it.

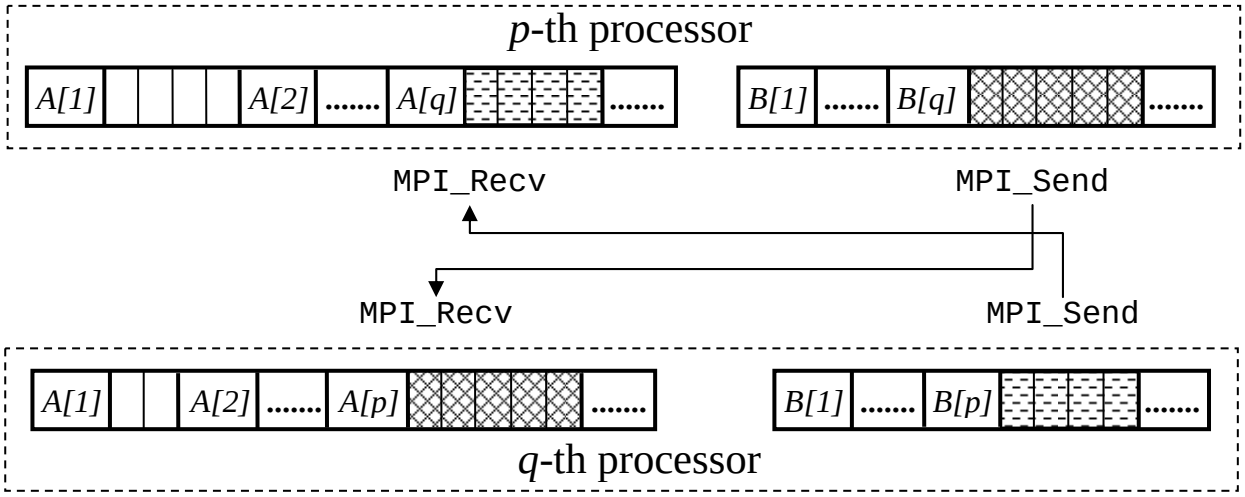

FIGURE 3. Scheme of data exchanges between processors *p* and *q*. *p*-th processor: the list *A[q]* contains cells, which are ghost for processor *p* and belong to *q*-th sub-domain, the list *B[q]* contains cells which belong to *p*-th sub-domain and are ghost for processor *q*. The lists *A[p]* and *B[p]* for *q*-th processor are obtained in analogous way. The data exchange is performed by call `MPI_Send` procedure for cells from *B[q]* and *B[p]* and `MPI_Recv` for cells from *A[q]* and *A[p]*.

The next stage of the UFS parallelization consisted in static parallelization when different sub-domains were assigned to different processors and remained unchanged during the computations, while parallel computing being performed for the entire domain, and different processors transmitting data from sub-domain boundaries to

corresponding processors. This was done using MPI library. In order to correctly set the values of all quantities of necessity in ghost cells two arrays *A* and *B* of lists have been created for each processor. For a given processor the *p*-th element of the first array *A[p]* contains list of all ghost cells for a sub-domain of a given processors, which belong to sub-domain of *p*-th processor, and the *p*-th element of the second array *B[p]* is the list of cells of a given processor sub-domain, which are ghost cells to *p*-th processor. These lists are ordered in the same way (natural tree traverse order) and the list *A[p]* for *q*-th processor and *B[q]* for *p*-th processor reference the same cells. These arrays are created dynamically after the mesh has been changed. So by calling `MPI_Send` and `MPI_Recv` procedures all necessary quantities can be set to correct values in ghost cells (see Figure 3).

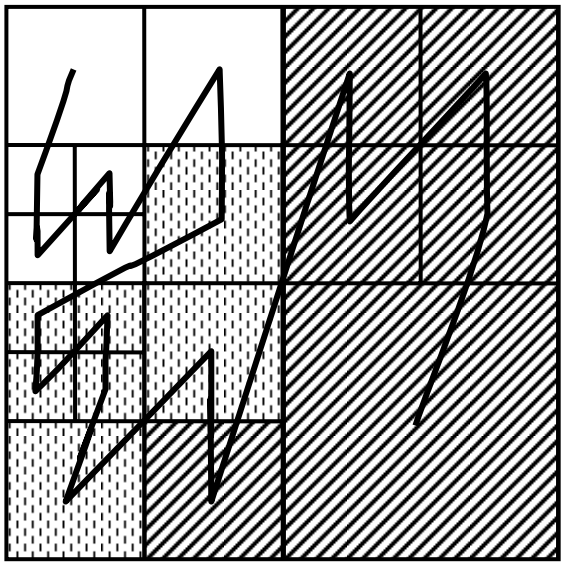

FIGURE 4. Domain partitioning obtained with SFC algorithm for the case of 3 processors.

After completion of this stage, the possibility of dynamic domain decomposition was implemented. The dynamic load balance between different processors was achieved separately for kinetic and continuum equations. The procedure of domain decomposition was performed using space filling curves (SFC, see [8]) as illustrated in Figure 3. During sequential traversing of the cells by natural order, the physical space is filled with curves in N-order (Morton ordering). After this ordering of cells, all cells can be considered as a one-dimensional array. A weight is assigned to each cell, which is proportional to the CPU time required to perform computations in this cell. Furthermore, the array modified with corresponding weights, is subdivided into sub-arrays equal to the number of processors, in such a way that the weight of the sub-arrays are approximately the same. This method allows rather efficient domain decomposition between different processors. Figure 4 shows an example of domain decomposition between several processors for supersonic gas flow around a cylinder.

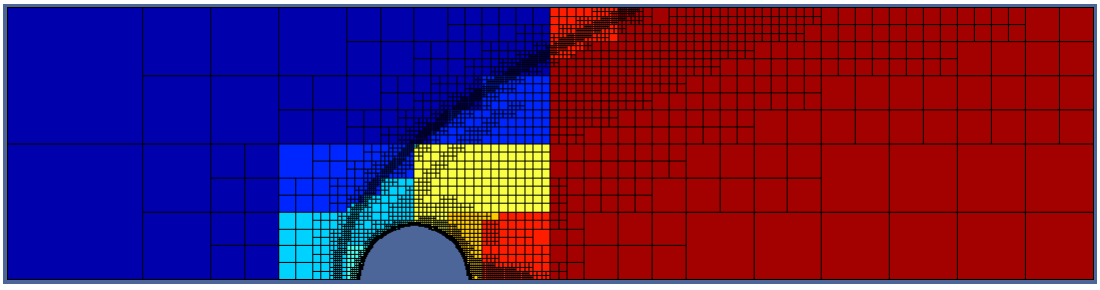

FIGURE 5. Example of domain decomposition for the problem of a flow about the cylinder for the case of processors. Sub-domains corresponding to different processors are indicated with different colors.

RESULTS

The parallel efficiency of the UFS has been studied cluster of Dorodnicyn Computing Center of Russian Academy of Sciences, which is a 8-node cluster, with each node having 2 Xeon 2.6 GHz processors and 4 Gigabytes of memory, and nodes being connected to each other via 2 GBit Myrinet network. Figure 5 shows speedup for different number of processors for coupled Euler-Boltzmann run of the UFS for the problem of a flow about the cylinder. One can see that UFS allows obtaining 9 times speedup for 14 processors.

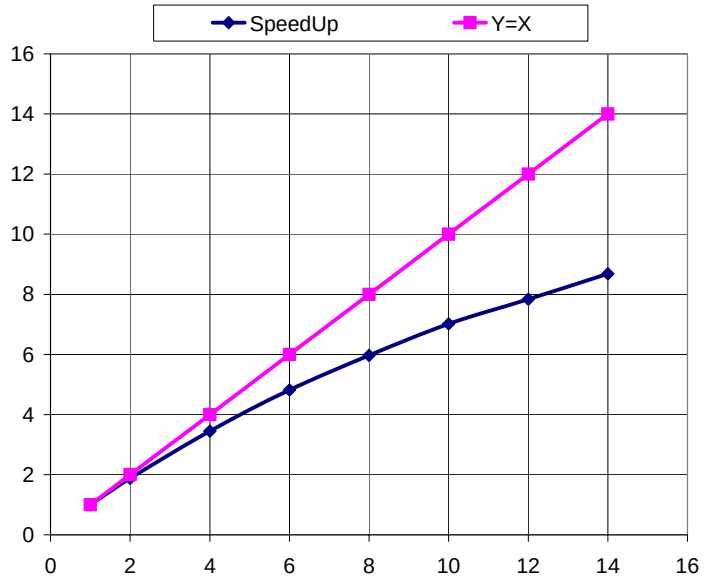

FIGURE 6. Speedup for coupled Euler-Boltzmann runs for the problem of a flow about the cylinder.

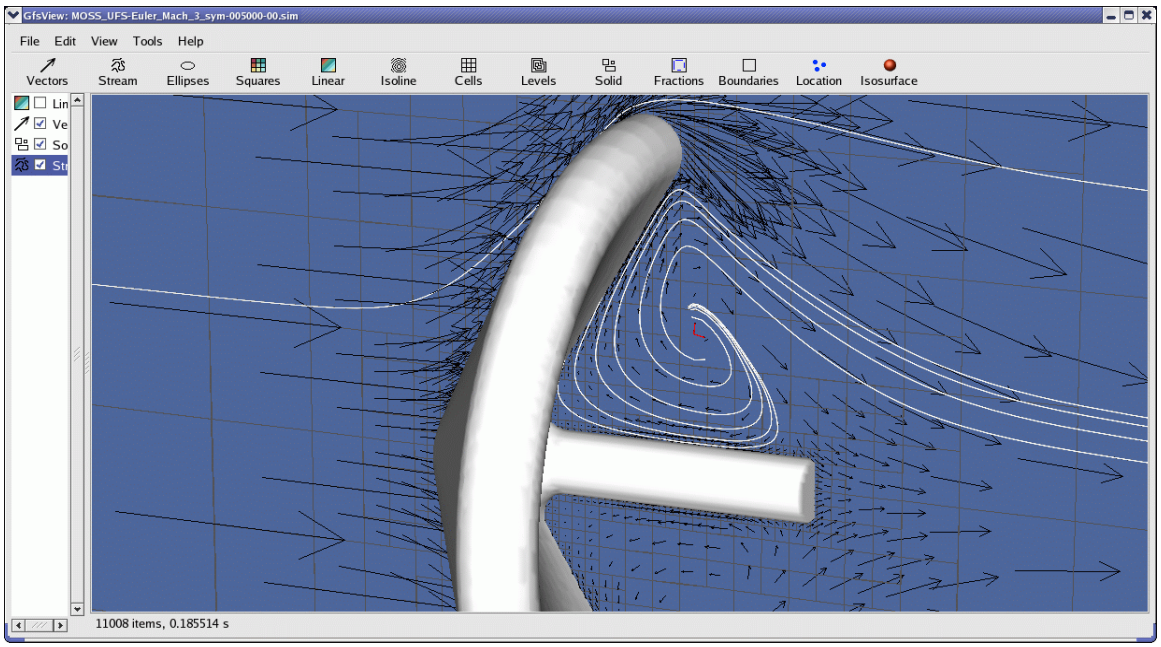

FIGURE 7. 3D simulation with conditions of Inflatable Reentry Vehicle Experiment (IRVE). Streamlines, computational mesh and field of velocity.

Current parallelization allowed us to compute several complex problems. As an example in Figure 6 one can see the results of computations for a 3D problem with conditions of Inflatable Reentry Vehicle Experiment (IRVE), obtained with 7-processor Linux cluster.

CONCLUSION AND PERSPECTIVES

We have obtained good speedup for rather simple test problem for up to 14 processors. The greater number of processors can require the speedup curve be closer to the ideal linear speedup. Nevertheless this speedup could be enough for the realistic problems, because the computational cost of the data exchanges between processors for such problems does not grow as fast, as the amount of computations per time step. So the parallel version of the UFS should be more thoroughly tested for the greater number of processors and for more complex problems. This will be done in the nearest future. Also non-parallel parts of the code which prevent the efficiency to be close to unity must be clearly identified in order to obtain estimates for their influence on the efficiency for realistic problems and to find other possibilities to optimize the parallel performance of the UFS.

ACKNOWLEDGMENTS

This research is supported by the US Air Force SBIR Project F33615-03-M-3326, by the Russian Foundation for Basic Research, Grant 04-01-00347 and by project 14 of the Presidium of the Russian Academy of Sciences.

REFERENCES

1. H.S. Wijesinghe and N.G.Hadjiconstantinou, *International Journal for Multiscale Computational Engineering* **2**, 189 (2004).
2. V.I.Kolobov, S.A.Bayyuk, R.R.Arslanbekov, V.V.Aristov, A.A.Frolova, and S.A.Zabelok, *AIAA Journal of Spacecraft and Rockets* **42**, 598 (2005).
3. V.V.Aristov, "Direct Methods for Solving the Boltzmann Equation and Study of Non-Equilibrium Flows", Kluwer Academic Publishers, 2001
4. F.G.Tcheremissine, "Direct Numerical Solution of the Boltzmann Equation", in RAREFIED GAS DYNAMICS: 24th International Symposium on Rarefied Gas Dynamics, AIP Conference Proceedings 762, 677 (2005)
5. K.Xu, *J. Comp. Phys*. **171**, 289 (2001).
6. T.Ohwada and S.Fukata, *J.Comp. Phys*. **211**, 424 (2006).
7. http://gfs.sourceforge.net/
8. M.J. Aftosmis, M.J. Berger, S.M. Murman, *AIAA 2004-1232*