A Tangled Web Strategy for Numerical and Parallel Scalability in Aerospace Simulation

Jochem Häuser
Dept. of Parallel Computing, Center of Logistics and Expertsystems GmbH, Germany

Roy D. Williams
CACR, California Institute of Technology, USA

Ralf Winkelmann
Dept. of Parallel Computing, Center of Logistics and Expertsystems GmbH, Germany

1. Parallelism for Large Scale CFD Applications

Parallel computing has become a key component of high performance computing in the 90's. In order to exploit this technology in science and engineering, in particular for aerospace, automotive, and turbomachinery applications as well as in environmental simulation, highly complex geometries have to be dealt with, often generated by CAD systems. In order to bring CFD in the design loop, quick turnaround times are mandatory.

Grids are multiblock hexahedra allowing any kind of topology and comprise any number of blocks. Grid generation is completely non-interactive. Only a wireframe and the geometry description are provided. Surface and volume grids are generated together. Grids are slope continuous. Grids are constructed using a high-level grid generation language. Grids consist of objects that are reusable. A problem/user specific grid database may be constructed that allows to use more complex entities from which to generate new grids. Complex grids are built in an object oriented fashion.

2. Parallelization Strategy for CFD Codes

There are basically three ways of parallelizing a code. First, a simple and straightforward approach is to **parallelize the do loops** in the code. Many so called automatic parallelizers analyze do loops and suggest a parallelization strategy based on this analysis. This concept, however, is not scalable to hundreds or thousands of processors, and results in a very limited speedup.

Most applications in science and engineering can be described by a set of equations in some kind of solution space. A second approach therefore is **to parallelize the numerical solution process for these equations**. For example, if a matrix-vector multiplication occurs, this multiplication could be distributed on the various processors and performed in parallel. Again, scalability to a large number of processors cannot be obtained. Moreover, this technique would work only for large regular matrices. If a problem was represented by a large number of smaller matrices (often the case in practice), parallelization would be impossible.

The third approach is denoted as **domain decomposition, sometimes also referred to as grid partitioning**. The idea is simple. The solution domain is subdivided into a set of subdomains that exchange information to update each other during the solution process. The numerical solution takes place within each domain, and is thus independent of the other domains. The solution space can be the actual space-time continuum, or it can be some abstract space. For the computer simulation, this space is discretized and thus is described by a set of points. Domain decomposition is the most general and versatile approach. It also leads to the best parallel efficiency, since the number of points per subdomain (or block) can be freely varied as well as the number of subdomains per processor. A large number of codes in science and engineering

paper to be published in the proceedings of CFD 97, University of Victoria, BC, 25-27 May 1997

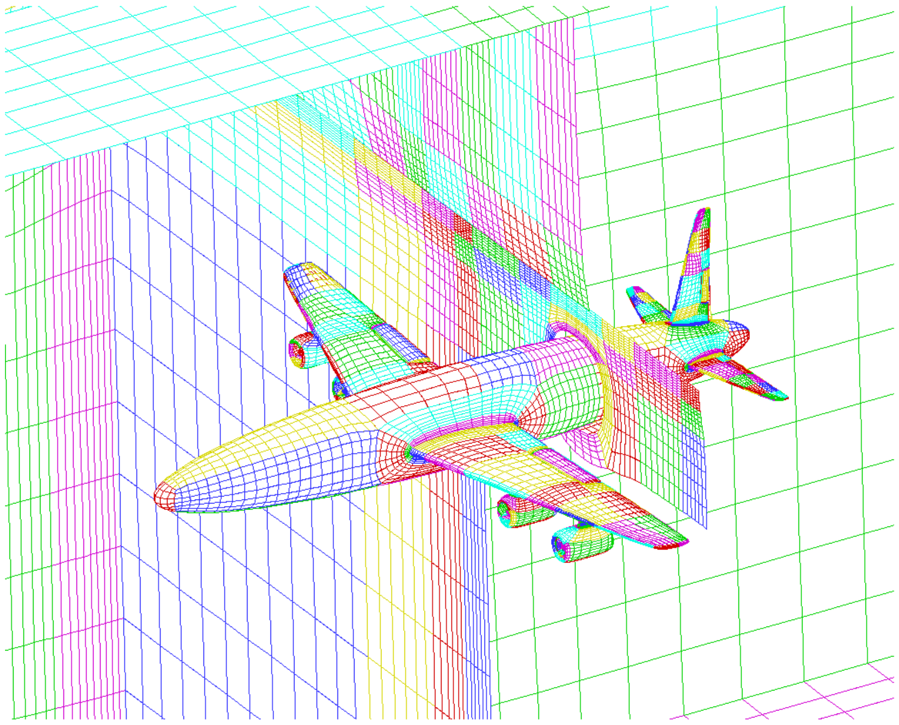


Fig 1: Generic aircraft configuration.

There is, however, an important aspect in parallelization, namely the geometrical complexity of the solution domain. In the following, a brief discussion on **geometrical complexity** is given and how it affects parallelization. If the solution domain comprises a large rectangle or box, domain decomposition is relatively straightforward. For instance, the rectangle can be decomposed into a set of parallel stripes, and a box can be partitioned into a set of planes. This leads to a one-dimensional communication scheme where messages are sent to left and right neighbors only. However, more realistic simulations in science and engineering demand a completely different behavior. For example, the calculation past an entire aircraft (see Fig. 1) configuration leads to a partitioning of the solution domain that results in a large number of subdomains of widely different size, i.e. the number of grid points in the various blocks differs. As a consequence, **it is unrealistic to assume that a solution domain can be partitioned into a number of equally sized subdomains**. In addition, **it is also unrealistic to assume a nearest neighbor communication**. On the contrary, the set of subdomains is unordered (unstructured) on the subdomain level, leading to **random communication among subdomains**. In other words, the communication distance cannot be limited to nearest neighbors, but any distance on the processor topology is possible (processor topology describes how the processors are connected, for instance as a 2D mesh, as a torus or as a hypercube etc.). Hence, the efficiency of the parallel algorithm must not depend on nearest neighbor communication. Therefore, the parallelization of solution domains of complex geometry requires a more complex communication pattern to ensure a loadbalanced application. It also demands more sophisticated message passing among neighboring blocks, which may reside on the same, on a neighboring, or on a distant processor. The basic parallelization concept for this kind of problem is the **introduction of a new type of boundary condition, namely the inter-domain boundary condition** that is updated in the solution process by neighboring subdomains via message passing. **Parallelization then is simply achieved by the introduction of a new type of boundary condition.** Thus, parallelization of a large class of complex problems has been logically reduced to the well known problem of specifying boundary conditions.

1. A Tangled Web for the Solution of the Navier-Stokes Equations

It is important to note that the successful solution of the large scale parallel N-S equations can only be performed by a **Triad** numerical solution procedure. Numerical Triad is the concept of using **grid generation, domain decomposition**, and the **numerical solution scheme** itself. Each of the three **Triad** elements has its own unique contribution in the numerical solution process. However, in the past, these topics were considered mainly separately and their close interrelationship has not been fully recognized. In fact, it is not clear which of the three topics will have the major contribution to the accurate and efficient solution of the N-S equations. While it is generally accepted that grid quality has an influence on the overall accuracy of the solution, the solution dynamic adaptation process leads to an intimate coupling of numerical scheme and adaptation process, i.e. the solution scheme is modified by this coupling as well as the grid generation process. When domain decomposition is used, it may produce a large number of independent blocks (or subdomains). Within each subdomain a block-implicit solution technique is used, leading to a decoupling of grid points. Each domain can be considered to be completely independent of its neighboring domains, **parallelism** simply being achieved by introducing **a new boundary condition**, denoted as inter-block or inter-domain boundary condition. Updating these boundary points is done by message-passing. It should be noted that exactly the same approach is used when the code is run in serial mode, except that no messages have to be sent to other processors. Instead, the updating is performed by simply linking the receive buffer of a block to its corresponding neighboring send buffer. Hence, **parallelizing a multi-block code neither demands rewriting the code nor changing its structure**.

A major question arises in how the decomposition process affects the convergence rate of the implicit scheme. First, it should be noted that the N-S equations are not elliptic, unless the time derivative is omitted and inertia terms are neglected (Stokes equations). This only occurs in the boundary layer when a steady state has been reached or has almost been reached. However, in this case the Newton method will converge quadratically, since the initial solution is close to the final solution. The update process via overlap boundaries therefore should be sufficient. In all other cases, the N-S equations can be considered hyperbolic. Hence, a full coupling of all points in the solution domain would be unphysical, because of the finite propagation speed, and is therefore not desirable and not needed. To retain the numerical order across block (domain) boundaries, an overlap of two points in each coordinate direction has been implemented. This guarantees that the numerical solution is independent of the block topology. The only restriction comes from the computation of flow variables along the diagonals on a face of a block, needed to compute the mixed derivatives the viscous terms.

To continue the discussion of convergence speed it should be remembered that for steady state computations implicit techniques converge faster than fully explicit schemes. The former are generally more computationally efficient, in particular for meshes with large variations in grid spacing. However, since a full coupling is not required by the physics, decomposing the solution domain should result in a convergence speedup, since the inversion of a set of small matrices is faster than the inversion of the single large matrix, although boundary values are dynamically updated. On the other hand, if the decomposition leads to a blocksize of 1 point per block, the scheme is fully explicit and hence computationally less efficient than the fully implicit scheme. Therefore, an **optimal decomposition topology must exist** that most likely depends on the flow physics and the type of implicit solution process. So far, no theory has been developed. However, a number of numerical experiments has been performed with the *ParNSS* code, clearly demonstrating the convergence speedup. Block numbers have been varied from 2 to 1024 in 2D and from 6 to 384 in 3D, using an otherwise identical grid.

In this paper, the basis of the numerical solution technique is the Newton method, combined with a Conjugate-Gradient technique (*GMRES*) for convergence acceleration within a Newton-Iteration. In the preconditioning process used for the Conjugate-Gradient technique, domain decomposition is used to decrease the condition number (ratio of largest to smallest eigenvalues) of the matrix forming the left hand side, derived from the discretized N-S equations. In other words, the eigenvalue spectrum is compressed, because the resulting matrices are smaller. It is shown in [1] that this ratio is a measure of the convergence speed for generalized conjugate residual algorithms. Having smaller matrices the condition number should not increase; using physical reasoning it is concluded that in general the condition number should decrease.

From these remarks, it should be evident that only a combination of grid generation scheme, numerical solution procedure, and domain decomposition approach will result in an effective, general numerical solution strategy for the parallel N-S equations on complex geometries. Because of their mutual interaction these approaches must not be separated. Thus, the concept of numerical solution procedure is much more general than devising a single numerical scheme for discretizing the N-S equations. **Only the implementation of this interconnectedness in a parallel solver will lead to the optimal design tool**.

2. Speedup Results by Tangled Web Approach

So far we have investigated the following acceleration schemes:

1. **Adaptive coupling** during the solution process, that is, the computation is started with an explicit scheme, followed by a block implicit GMRES technique using a large number of blocks. Automatic block merging is used to increase coupling strength. Merging of blocks is based on the residual of the solution. Results of this strategy are presented in Fig.2.

2. Results in Table 1 are obtained by varying block number only, that is, the computational scheme is not changed.

3. Computing a coarse grid flow solution and transferring it onto a finer grid. Repeating this process in sequence, i.e. using a sequence of grids from coarse to fine, accelerates convergence to a steady state solution. Results are given in Table 2.

4. Combination of acceleration techniques: Grid sequencing plus adaptive coupling are presented in Fig. 4 and Fig 5.

# of block	# of points per block	# of iterations	computing time	speedup
2	24000	253	52519	1.00
32	1560	305	33930	1.55
120	435	317	22577	2.326
256	213	333	19274	2.725
480	**119**	**349**	**17752**	**2.958**
1024	61	380	18012	2.916

Table 1: Results for solution acceleration by variation of block number for a 48000 point NACA0012 airfoil. The Euler solution is computed by the implicit GMRES algorithm for a Mach number of 1.7. The computation stops after the residual drops to 10e-12. The optimal speedup is obtained for 480 blocks.

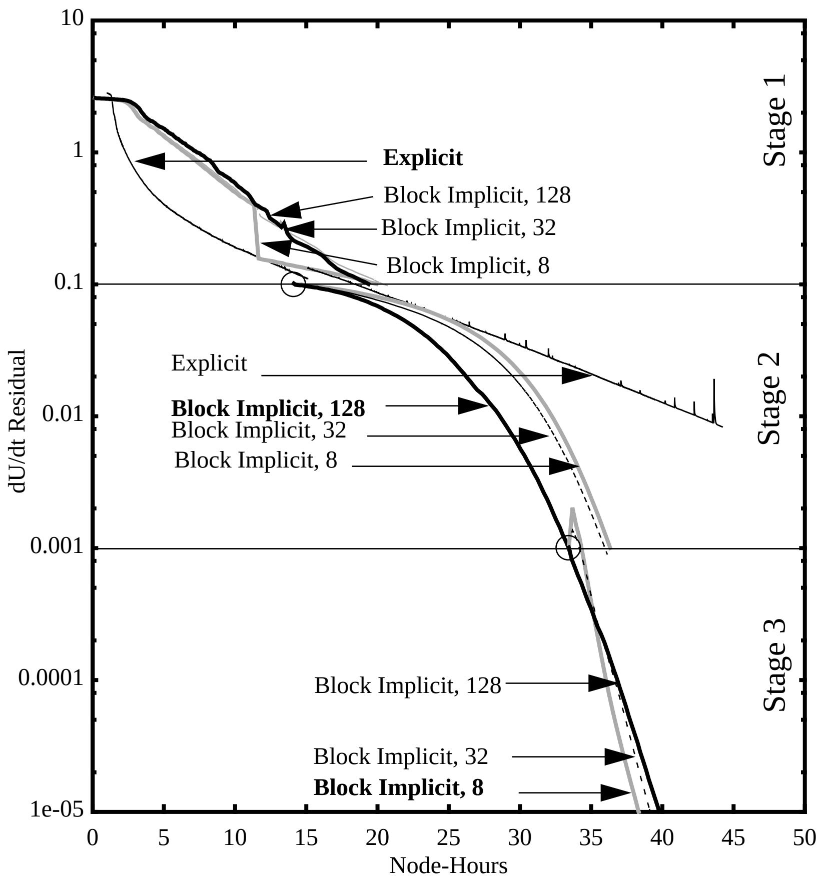


Fig 2: Solution acceleration using adaptive coupling. Computation was done for a NACA0012 airfoil.

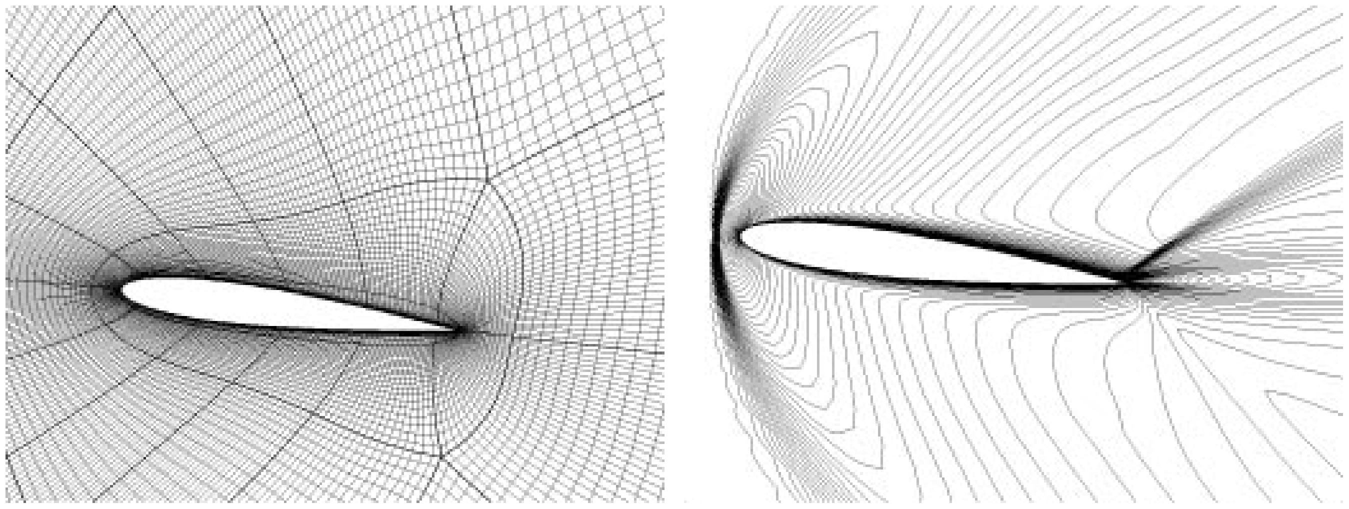
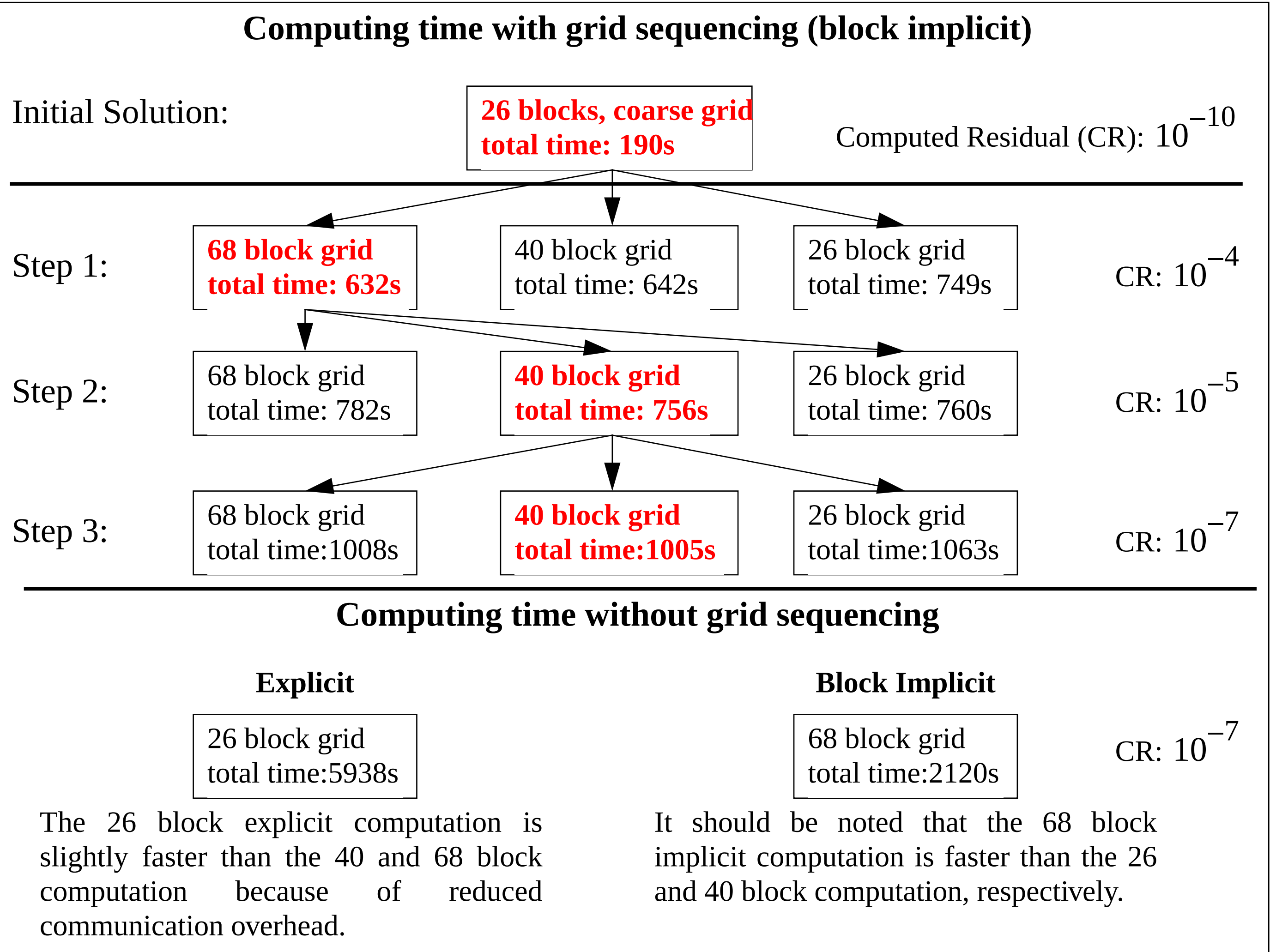


Fig 3: NACA 0012 airfoil. Navier-Stokes grid and Mach number contour plot. The fine grid consists of 26 blocks and some 29,000 cells. The coarse grid comprises about 7,500 cells.



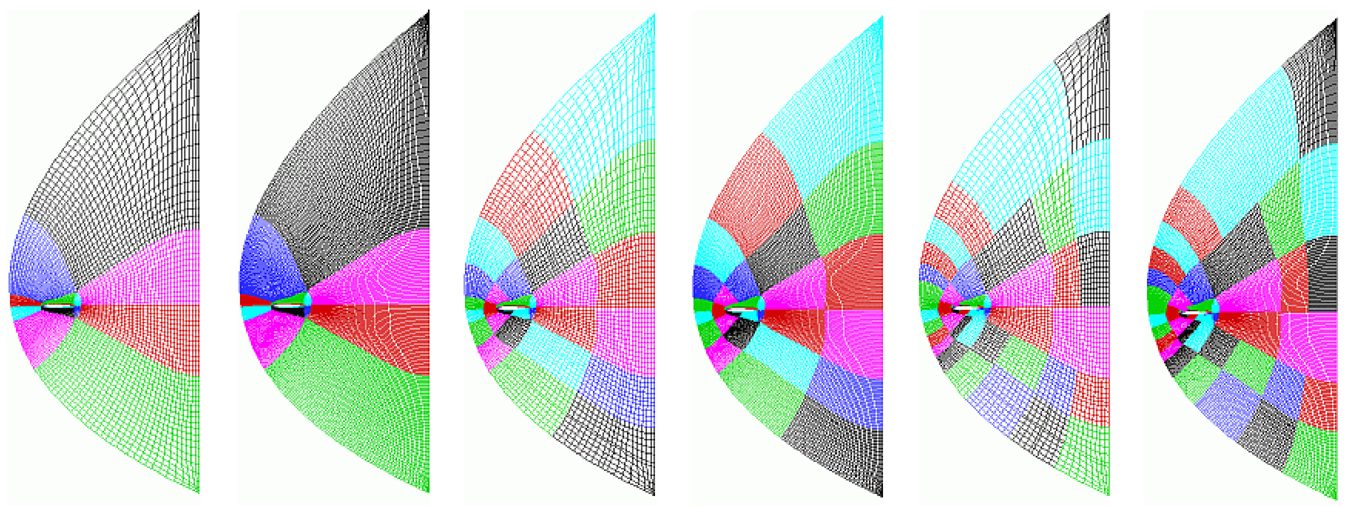
The 26 block explicit computation is slightly faster than the 40 and 68 block computation because of reduced communication overhead.

It should be noted that the 68 block implicit computation is faster than the 26 and 40 block computation, respectively.

Fig 4: Grid sequencing, in combination with adaptive coupling, was used in the Navier-Stokes computation of the NACA0012 airfoil. First, a converged solution was computed on the 26 block coarse grid (see Fig. 5a) which served as initial condition for the computations in the steps 1 to 3. It turns out that the combination of grid sequencing and adaptive coupling results in substantial speedups in comparison with fully explicit or a block implicit solution on a fine grid only. The speedups are 5.91 and 2.11, respectively . It should to be mentioned that further improvements of the total execution time will be achieved by using dynamic load balancing.

Grid level	grid points	[CPU+SYS-time] s	[total CPU+SYS-time] s
3	832	134	134
2	3162	775	909
1	12322	6010	6919
0	48000	23741	30660
0 (no grid sequencing)	48000	51578	51578

Table 2: Results for solution acceleration by using multi-level grids for a 48,000 point NACA0012 airfoil. The Euler solution is computed by implicit GMRES for a Mach number of 1.7. The algorithm switches to the next finer grid, if the residual has dropped to 10e-12 on the coarse grid level. A speedup of 1.68 is obtained by this techniques.



a) coarse grid 26 blocks | b) fine grid 26 blocks | c) coarse grid 40 blocks | d) fine grid 40 blocks | e) coarse grid 68 blocks | f) fine grid 68 blocks

Fig 5: A sequence of coarse and fine grids along with varying number of blocks has been used to compute the steady state solution. Decomposing and coarsening of grids is handled automatically by *ParNSS*. In addition, the code performs the transfer of the numerical solution from the coarse to the fine grid. In case a varying number of blocks is computed, the solution transfer is also handled by *ParNSS*.

This work is part of the PhD thesis of Ralf Winkelmann.

References

[1] Gene Golub, James M. Ortega, Scientific Computing, Parallel Computing, Academic Press, INC, 1993, ISBN 0-12-289253-4

[2] R. D. Williams, J. Hauser, and R. Winkelmann, “Efficient Convergence Acceleration for a Parallel CFD Code.” In: Parallel Computational Fluid Mechanics 1996, A. Ecer et. al., Eds., Elsevier North-Holland, to be published.

[3] J. Hauser, R. D. Williams, H.-G. Paap, M. Spel, J. Muylaert, and R. Winkelmann, “A Newton-GMRES Method for the Parallel Navier-Stokes Equations.” In: Parallel Computational Fluid Mechanics 1995, A. Ecer et. al., Eds., Elsevier North-Holland, 1995.

[4] J. Hauser, M. Spel, J. Muylaert, and R. Williams, ParNSS: An Efficient Parallel Navier-Stokes Solver for Complex Geometries, AIAA paper 94-2263.

[5] J. Hauser and R. D. Williams, “Strategies for Parallelizing a Navier-Stokes Code on the Intel Touchstone Machines,” Int. J. Numerical Methods in Fluids, 15(51), 1992.

[6] P. R. Eiseman et al., GridPro/az3000 User’s Manual, Program Development Corporation, White Plains, NY, Oct. 1996.

[7] J. Hauser, J. et al., “Euler and Navier-Stokes Grid Generation for Halis Configuration with Body Flap.” In: Numerical Grid Generation for Computational Fluid Dynamics, B. Soni et al., Eds., NSF Engineering Research Center for Computational Field Simulation, Mississippi State Univ., ISBN 0-9651627-0-2

[8] R.D. Williams, J. Hauser and R. Winkelmann, Hypersonic Flow Around the Halis Orbiter, Concurrent Supercomputing Consortium, California Institute of Technology Caltech Anual Report 1996, http://www.cacr.caltech.edu/publications/annreps/annrep96/cfd1.html, 1996

[9] R.D. Williams, J. Haeuser and R. Winkelmann, Annual Report 1995 (page 56-57), Concurrent Supercomputing Consortium, California Institute of Technology, USA and http://www.cacr.caltech.edu/publications/annreps/annrep95/cfd1.html