FLO67P: a multi-block version of FLO67 running within PARAGRID

F. Dellagiacoma, M. Vitaletti
IBM ECSEC; Rome, Italy

A. Jameson, L. Martinelli
Princeton University, MAE Dept.
Princeton, NJ, USA

S. Sibilla, L. Visintini
Aermacchi S.p.A.; Varese, Italy

Abstract

In this paper we describe the implementation of FLO67P, a version of the FLO67 code which is built upon the PARAGRID parallel multi-block environment. The emphasis here is on those features of FLO67 which determine the strategy followed to adapt the original code to the PARAGRID application programming interface. Results are given for the simulation of supersonic flows around complex configurations. Parallel speed-up figures and scalability issues are discussed.

Introduction

This paper illustrates the integration of FLO67, a widely known code for the analysis of aerodynamic flows, within the PARAGRID multi-block environment. A brief description of the features of the two codes is given in this section. The subsequent sections are devoted to the illustration of the following items:

- integration design,
- multi-block topology issues,
- accuracy and efficiency tests, and
- parallel performance.

FLO67

FLO67 [?] is a computer code capable of analyzing inviscid, compressible fluid flows by solving the conservation form of the Euler equations in the subsonic, transonic, and supersonic regimes. FLO67 uses a cell vertex discretization suitable for calculations on meshes with sharp corners between grid lines. The numerical discretization is carried out via a finite volume formulation using body fitted curvilinear meshes with hexahedral cells. Integration in time is carried out by a multi-stage Runge-Kutta method. No upwind bias is introduced in the convective fluxes at the cell interfaces. Spurious oscillations are filtered by an adaptive dissipative correction which is of fourth order in smooth flow regions, while it becomes of second order near discontinuities. This method guarantees perfect global conservation of the physical quantities such as mass, momentum and energy, and also provides second order accuracy on smooth meshes.

The code is capable of computing both steady and unsteady flows. For steady state calculations several acceleration techniques are implemented, including the use of a variable *local time step*, *implicit residual smoothing* and *multi-grid* (MG). Such techniques provide very fast convergence of the iterative process, and the calculation of a typical steady state solution requires only about 25 steps.

The original code handles a single-block structured grid. However, the flow solver and the multigrid modules are essentially independent of the topological description of the mesh, making it suitable to deal efficiently with more complex geometries through a multi-block approach.

This feature has been exploited in the implementation of FLO67P, a version of the original code which is based upon PARAGRID [?] and is capable of computing multi-block structured grids in parallel on clusters of workstations.

PARAGRID

PARAGRID is a parallel multi-block framework developed by the IBM European Center for Engineering and Scientific Computing (ECSEC). A program solving the discretized field equations on a *single-block structured grid* can be integrated in the proposed environment as a *subdomain solver*, so that *multiple copies* of the code can work concurrently on different grid blocks and therefore exploit the computing power of parallel computers.

Each one of the six faces of a block can either be of *physical* or *internal* type. The main topological constraint on admissible multi-block grids is that internal faces are entirely shared by *only two* neighboring blocks. Nodes lying at the interface shared by the two subdomains are *duplicated*, being included in both grids. On the other hand, there is no explicit assumption concerning the number of blocks which share a given edge or corner. PARAGRID features an automatic analysis of the grid topology which detects all the adjacency relations between blocks.

In PARAGRID, an arbitrary number of fields can be associated to the following *grid sites*, namely the *nodes*, the *cell centers*, and the grid *cell face centers.*

The field variables are stored in multi-dimensional arrays and managed according to their *type.* There are three types of data: *static*, *dynamic* and *scratch.*

PARAGRID applications access data from two different program interfaces, one for *global* operations and one for *local* operations. The subprogram designed for *global* operations usually performs ANALYSIS (e.g. post-processing) functions, while the subprogram designed for *local* operations is mainly responsible for the UPDATE of the fields within each individual block.

The task of the application UPDATE subprogram is to advance the computation of the *dynamic* field variables associated to a given block by one step. In addition to the status of the field as computed in the previous step, the information available to the UPDATE subprogram includes the status of the field over a *halo* of grid cells. The latter extends from the block boundaries into the adjacent blocks by a *depth* which is specified by the user.

PARAGRID maintains in a *transparent fashion* those elements of arrays storing dynamic data which are associated to points falling in the *halo* region. This is achieved through the following stages:

- in the *export* stage data from the *core* region of one block which fall within the halo region of other neighbours are copied to the sequential chunks reserved for each of the target blocks. Chunks relative to those blocks which are assigned to a *remote host* are actually sent through the network.
- In the *import* stage, chunks containing exported data, are used to *replace* elements associated to the halo region.
- Eventually, in the UPDATE stage, the user's application subprogram is CALLed to advance dynamic field data in time on each individual block.

More details on the PARAGRID application programming interface can be found in [?].

FLO67P: Integration Design

Adapting to the PARAGRID interface did not require any substantial change of the code kernels. The core of the Runge-Kutta (RK) time integration is the computation of the net flux entering the control volume associated to each node, which is formed by the 8 surrounding cells. Therefore, the computation of convective terms would only require one extra layer of nodes in the halo region. One additional layer is required to compute fourth order artificial dissipation terms, giving a total of *two layers* of data in the halo region.

Some choices made in the design phase were aimed at increasing the level of *locality* in the calculations, in order to achieve a good parallel efficiency. In practice, this approach was also the one requiring the minimum recoding efforts, because the original integration scheme is applied, with minimum variations, to each individual block. In particular, each step of advancement in time is performed in *two* PARAGRID UPDATE steps. The update of the finest grid level is performed at the *odd* steps while the *even* steps are responsible to compute the *coarse grid corrections*. This mechanism causes halo data to be exchanged only before the initial fine grid update (the fine grid *relaxation* in the MG terminology) and before the the successive collection of residuals to the first coarser grid:

1. PARAGRID replaces *halo* data with the most up to date values.
2. *Odd step*
   - The whole sequence of RK stages is applied to the finest grid level with *frozen halo data*;
3. PARAGRID replaces *halo* data with the most up to date values.
4. *Even step*
   - *Residuals* on the finest grid level are computed and *injected* to the first coarse grid.
   - The whole MG cycle is completed within each block with *frozen halo data* on all grid levels.

Block Multi-Grid

This is approach is different compared to the one usually taken in the implementation of serial multi-block codes. In the latter case, a major goal is to ensure the identity of results, at each phase of the calculations, when using different block splittings of the same single-block grid. In the PARAGRID environment, achieving this goal would require to exchange halo data among adjacent blocks at each stage of the RK sequence, with a severe increase of the communication overheads.

The lack of automatic halo management for multiple grid levels, in the current version of PARAGRID, forces to confine the computation of coarse grid corrections within each individual block. Moreover, the halo of coarser grids is managed explicitely by the application. Only *one* layer of halo data is needed on coarser grids, where fourth order dissipation terms are *not* computed. As a result, the halo can be created in the ordinary way only of the first coarse level, while the doubling of spacings on the successive coarse levels cannot affect the direction normal to an internal boundary. The situation is depicted in in Figure ??.

Multi-Block topology Issues

Common grid topologies involve *irregular edges*, and *corners* for which it is not possible to establish a one-to-one correspondence between the grid points falling in the halo region and the storage elements of a three-dimensional regular array. In general, the proper treatment of these irregular points would require to adopt an unstructured storage format for points in the halo region, which is currently not supported by PARAGRID. Irregular *edges*, in particular, can be classified according to the following *four* categories:

1. *Conflicts on points of the halo region.*
   This occurs, for example, in C-type grids around isolated wings with a *sharp trailing edge.* In this case, the upper and lower surfaces of the body coincide at the trailing edge. The halo region of blocks behind the trailing edge has storage elements where one should store data from both the lower and the upper surface.
2. *Only 2 blocks around the edge.*
   This occurs, for example, in C-type grids around isolated wings with a *sharp tip.* In this case, splitting the region beyong the tip into a lower and a upper part, one sees that there are only two blocks around the leading edge.
3. *Only 3 blocks around the edge.*
   This configuration is very convenient when covering the region of space surrounding a convex object.
4. *More than 4 blocks around the edge.*
   This configuration is frequently encountered when covering geometrically complex regions.

The second and third of the above irregular edges are easily handled. In both cases, no use is done in computational space of the halo points associated to the edge itself, being all the relevant information already available from the halo chunks relative to the contiguous block faces. The first and the last cases are more difficult to treat. Here we only mention the strategy followed to solve the first problem.

The presence of multiple grid points conflicting on the same storage element of the block halo causes PARAGRID to store only one of the values and to flag the occurrence of such condition to the application. In FLO67, these pathological elements enter in the computation of the *edge boundary* grid points. Using the pathological elements as is would produce different values to be associated to the same edge node in the four surrounding blocks. The solution implemented in FLO67P computes the edge values as an average of the neighboring nodes which occupy the lower and the upper position, respectively, with respect to the separating body. This *directional* averaging can be computed consistently in all the surrounding blocks, since it does not involve the pathological halo elements. In the case of a wing trailing edge, this approach is consistent with the enforcement of a *Kutta condition.*

Accuracy and efficiency tests

3. Efficiency tests and solution quality

The FLO67 multi-block implementation within Paragrid 1.0 has been tested on several geometries. A simple blunt nosed body has been chosen to check the quality of the solution and the convergence rate, in comparison with the results obtained from the original code. Two different multi-block grids have been tested on the same geometry, in order to study the influence of non-symmetric topologies and geometric boundary conditions on the behaviour of the code. Solutions from FLO67P have been compared with existing single-block solutions on a wing-body-canard configuration to investigate the effects of extensive block decompositions. Finally, the code has been tested on the real geometry of a launch vehicle; in this case, the multiply connected physical domain is much better described with a multi-block grid.

3.1. Blunt nosed body test case

3.1.1. Grid topologies

The test geometry consists in a cylindrical body ending with a spherical cap of unit radius, simulating a simplified aeronautical shape (like a fuselage or a missile). The length of the cylindrical section is 8 times the radius of the cap. A single block C-O grid of 97*41*17 nodes has been used to compare the original and the modified codes. The grid has been studied for the analysis of a high supersonic flow with bow-shock wave formation: the test condition has been set at Mach 2 and zero incidence. The grid has been also divided into 4 blocks to check, on the same topology, the effects of block decomposition. One block boundary lies normal to the surface, dividing the cap from the cylindrical part; the other lies parallel to the surface, at a distance suitable to the description of the bow shock wave. Cells in the direction normal to the surface are equispaced in the first layer of blocks, while in the second layer they are distributed with a Vinokur stretching function. The 4 resulting blocks have 33*25*17, 65*25*17, 33*17*17 and 65*17*17 nodes. Finally, a different 3-block grid has been generated in order to avoid the use of a singular line boundary; this grid allows also to investigate the effects of a non-symmetric topology in the analysis of symmetric geometries and flows. The surface grid consists in a patch of H topology on the nose region connected with a C-O description of the cylindrical region identical to the previuosly described grid; the cell distribution normal to the surface is also identical to the previous one. The grid is therefore composed of a nose block of 17*41*17 nodes and two blocks of 81*41*17 nodes.

3.1.2. Convergence rate analysis

Two grid levels have been employed in all four cases, computing 50 multigrid steps for both the coarse and the fine grid with a CFL number of 6. If the maximum residual is examined, no real difference in convergence is noted between original FLO67 and the single-block solution obtained with FLO67P : the maximum residual loses in both cases 3 orders of magnitude. In fact, a better level of convergence is obtained on the coarse grid with the new FLO67P; actually, FLO67P provides a slightly different analysis of the singular line boundary present in the C-O topology. FLO67P obtains also a better convergence of the aerodynamic drag coefficient on the coarse grid. The final value of Cd is reached in less than 20 iterations; anyway, no real difference is observed in the convergence on the fine grid, where, within 15 steps, both computations converge to the final value. The same rate of convergence on aerodynamic coefficients is achieved if the 4-block decomposition is introduced. On the other hand, the maximum residual does not diminish more than two orders of magnitude under its initial value on the fine grid of the multi-level scheme; in fact, the residual seems not able to converge to a lower value on some points of the internal block boundaries. In any case, this does not affect the quality of the solution, as it will be shown later. A slightly lower rate of convergence of residuals has been observed also on the 3-block grid, where such lower rate is expected as a consequence of the more complex non-symmetric topology; nevertheless, this fact is counterbalanced by the higher level of accuracy which can be obtained in the solution at the same number of iterations.

3.1.3. Solution analysis

No difference appears between the single-block and the multi-block solutions on the 97*49*17 C-O grid: a pressure iso-curve map shows identical levels everywhere in the flowfield. The bow-shock wave is located approximately at x = -1.35; a plot of the pressure coefficient on the stagnation streamline shows perfect coincidence between sigle-block and 4-block results. Better resolution on the shock wave is shown by the 3-block non-symmetric grid: the pressure discontinuity is described in a lower number of mesh cells and the solution at the stagnation point (Cp = 1.668) appears to be closer to the theorical solution Cp = 1.657, while the other calculations give Cp = 1.637. Actually, original FL067, single-block and 4-block FL067P solutions give all the same result, showing that no difference is introduced either by the Paragrid modification of FL067, or by the block decomposition; anyway, the 3-block description with a nose patch seems to be more suitable than a singular line grid for the analysis of strong shocks. The global drag coefficient is therefore slightly higher (0.85 the last case:

code	grid	C
FL067	1-block, singular line	0.7761
FL067P	1-block, singular line	0.7764
FL067P	4-blocks, singular line	0.7764
FL067P	3-blocks, nose patch	0.7830

3.1.4. Time requirements

The use of FL067 in the Paragrid environment seems to raise the time requirements of the computation of 20-35 of nodes and of grid blocks. CPU times of the present computations are given in the table below; all the cases have been run on a single machine IBM RISC System/6000 540.

code	grid	CPU time (s)		nodes	time * node * iteration (s)	
		coarse	fine		coarse	fine
FL067	1-block	255	2439	134521	2.782 10	3.626 10
FL067P	1-block	173	1482	67609	3.736 10	4.384 10
FL067P	4-blocks	178	1497	67609	3.844 10	4.428 10
FL067P	3-blocks	339	3101	124763	3.942 10	4.971 10

3.2. Analysis of complex geometries

3.2.1. Wing-body-canard configuration

A wing-body-canard configuration was chosen as an example of a fairly complex geometry featuring vortex flow and requiring multiple blocks. Such geometry was defined for the International Vortex Flow Experiment and a large number of theoretical and experimental results are available. This geometry consists in a canard surface with a 60 deg swept sharp leading edge and a 65 deg swept sharp leading edge wing, both mounted on a simple fuselage; the wing consists in a biconvex airfoil over the forward 40 NACA 64A005 over the aft 60 A suitable modification in the boundary conditions allows to employ a single-block 193*33*49 C-H mesh for the analysis of the flow with the original FL067 code . This mesh has been also divided in 32 computational blocks to investigate the effects of block decomposition in the Paragrid environment. The test condition has been chosen at Mach 0.85 and angle of attack alpha = 10 , to analyse a transonic vortex flow. The simulation has been run for 100 multigrid iterations on the coarse grid and 150 on the fine grid, with a CFL number of 4. Some problems have been met when running with an higher CFL; actually, while good convergence is achieved by FL067 at CFL = 8 on the same test case after 50 multigrid iterations on the coarse grid and 100 on the fine one, FL067P in Paragrid diverges after few steps on the fine grid. Maximum residual analysis at CFL = 4 shows that the block decomposition does not affect really the convergence rate, although the residual converges to a slightly higher value than the simple FL067 case. No difference at all is anyway found in the aerodynamic force coefficients convergence: both the lift and drag coefficients converge with the same rate, and 60 iterations are roughly sufficient to reach the final value. No real problem in the force evaluation can be ascribed to the block decomposition, as the final values do not differ for more than 0.3

code	grid	C	C
FL067	single-block	0.0865	0.494
FL067P	32-blocks	0.0868	0.493

The flowfield analysis shows some problem when the block boundaries cross the vortex region; actually, a slight unnatural deflection of the isolines is found in the plots of the pressure coefficient and of the total pressure ratio; anyway, both computations give the same results in the prediction of the vortex position and of the pressure distribution on the canard and wing surfaces. As previously noted, Paragrid implementation raises CPU time requirements, as shown below (always referring to an IBM RISC System/6000 540):

code	grid	CPU time (s)		nodes	time * node * iteration (s)	
		coarse	fine		coarse	fine
FL067	1-block	1659	17398	312081	4.024 10	3.717 10
FL067P	32-blocks	2109	22042	312081	5.116 10	4.709 10

3.2.2. Small satellite launch vehicle

The FL067P code in the Paragrid environment has been used in the aerothermal analysis of a small satellite launch vehicle; the vehicle consists in a core surrounded by 4 boosters. The computational domain has been limited by a conical boundary co-axial to the core, with radius varying between 60 and 80 lenghth of the core; the grid has an O-H topology around the core and around each booster; two different grid topologies have been considered: in the first one the O-H grid is closed around each nose, generating a singular line boundary; in the other, "patch blocks" have been used to avoid the presence of singular line boundaries. The first grid allows the analysis of an eighth of the launch vehicle (20 blocks, 250.000 cells) for zero incidence flows, while the second one forces to analyse at least a quarter of the vehicle (65 blocks, 400.000 cells). Finally each grid has been enlarged (to 80 and 130 blocks respectively)

Figure 1: Performance figures on a cluster of 6 workstations

to surround half missile, in order to study the flow at incidence. The second grid has proved more effective, especially in the analysis of high supersonic and hypersonic flows: actually, the singular line boundary in the first grid imposes the presence of block boundaries on the noses of core and boosters; the communication of flow data between upstream and downstream blocks through this division surface can be difficult at high Mach number and the resolution of the bow shock wave can be quite poor. Good results, in a large range of Mach numbers (0.95 - 6.0) were obtained with the second grid topology, especially in the analysis of the complex shock-wave reflections and shock-shock interactions in the core-booster interference region. These computations were run on IBM RISC System/6000 540, 550 and 53H in parallel; a speedup of 1.6 was reached when running on the first two machines, while when running on all three the maximum speedup has been 2.3.

Parallel performance

The parallel speed-up was measured in the computation of a realistic test case, involving a grid with *228888* points, on a cluster of *six* IBM RISC/6000 workstations mod 560 running at 50 Mhz. Two types of communication network were experimented: Token Ring adapters capable of 16 mbits/sec and FDDI adapters capable of 100 mbits/sec. PARAGRID implements the communication among nodes by TCP/IP sockets. The grid was made of *24 blocks* of the same size, giving a good load-balancing on *2*, *3*, *4* and *6* nodes. The parallel speed-up is illustrated in figure cite??. The speed-up measured on six nodes is is about 5 with both Token Ring and FDDI. The larger bandwidth of the latter gains only a little with respect to Token Ring. The reason is in the small difference in the workload of the computational nodes, which is due to the different computational weigth of physical boundaries with respect to the internal ones. Although relatively small, this causes a degradation of the paralle efficiency whenthe number of nodes is increased.