

AIRCRAFT COMPUTATIONS USING MULTIGRID AND AN UNSTRUCTURED PARALLEL LIBRARY

Paul I. Crumpton Michael B. Giles

This paper examines the application of unstructured multigrid, using a sequence of independent tetrahedral grids. The test cases examined are for inviscid flow over an aircraft and an M6 wing. The sensitivity of the method to grid sequence and cycling strategy are investigated.

All of the calculations were performed on a parallel computer. This was achieved by using the OPlus library which, by the straightforward insertion of subroutine calls, facilitates parallelisation of the resulting code. A single source OPlus application code can be compiled to executed on either a parallel or sequential machine. This greatly increases the usability of the parallel machine, and the maintainability of the code.

This work was presented at the AIAA meeting, January 1995, (Paper AIAA 95-0210).

Key words and phrases: unstructured grids, multigrid, parallel library

This work was performed within Oxford Parallel, with financial support from Rolls-Royce plc, DTI and SERC.

Oxford University Computing Laboratory
Numerical Analysis Group
Wolfson Building
Parks Road
Oxford, England OX1 3QD

April, 1997

1 Introduction

The eventual goal of this work is to accurately model flow over realistic configurations. Consequently there is a need for:

1. a physically realistic mathematical model,
2. an unstructured mesh to resolve the complex geometry,
3. an efficient numerical algorithm such as multigrid,
4. the effective utilisation of modern supercomputers, which entails parallel computation.

This paper addresses the latter three items. An Unstructured Multigrid algorithm is described and applied to inviscid flow over an aircraft configuration. The relative merits and complexities of the algorithm are discussed. This algorithm is parallelised using the OPlus subroutine library. This, by means of the straightforward insertion of subroutine calls at a loop level, enables a *single source* FORTRAN 77 program to be executed on either a sequential or a parallel machine.

2 Unstructured Multigrid

Multigrid has become a crucial algorithm for CFD, especially for structured grid applications [1]. Multigrid for unstructured tetrahedral grids is still at an early stage of development, [2], [3] and needs further validation and evaluation to become mainstream. This section discusses and analyses the merits and complexities of the approach.

Multigrid is an acceleration technique that requires three components:

1. A sequence of grids ranging from coarse to fine grid resolution;
2. Grid transfer operators, Restriction I_h^H from fine to coarse, and Prolongation I_H^h from coarse to fine.
3. A smoother, or iterative solver, that quickly damps all high frequency error components in the solution, thus leaving only low frequency components, that can be well approximated on a coarse grid;

These three components are discussed in detail in the following sections.

⁰Copyright ©1994 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

2.1 Sequence of grids

Here, as in [2], [3], a sequence of non-nested, independent tetrahedral grids are generated for a specific geometry (see Fig. 4, Fig. 1). The reason for adopting this approach is the availability of a “black box” grid generator. This leads to the extra flexibility of being able to choose coarse grids for optimal performance, but the problem of linking sequences of unstructured tetrahedral grids needs to be addressed. That is, for all nodes on a particular grid the tetrahedra which contain those nodes on the coarser and finer grids are required. At curved boundaries of the mesh this is not always possible, in which case “nearest” tetrahedra are required. An efficient algorithm based on directed line searches has been developed, which takes $O(M)$ time, where M is the size of the largest grid being linked.

The grid generation was performed using the advancing front method of [4]. This requires the definition of a background grid, which contains point/line/plane sources which define exponential functions to control local mesh density. These functions are scaled by ϕ such that $\phi = 1$ is the fine grid, and $\phi > 1$ produces a coarser grid. In this way it is easy to generate a sequence of grids for multigrid calculations.

An alternative to this “independent grid” approach is agglomeration multigrid [5], where a finite volume method is formulated using edge based coefficients, which can be “agglomerated” to form coarser grids. This has the advantage that the coarse grids are automatically generated from finest grid; however, there are limitations in the data structure (edge based), and the order of the prolongation ([5]).

2.2 Transfer Operators

The current implementation uses linear interpolation for both prolongation and restriction. Using a superscript to denote the grid, for a node \mathbf{x}_j^K on grid K a scalar q_j would be transferred from grid $K - 1$ by

$$q_j^K = \sum_{(i \in C_j^{K \rightarrow K-1})} \Phi_i^{K-1}(\mathbf{x}_j^K) q_i^{K-1} \quad (2.1)$$

where Φ_i^K are the piecewise linear basis functions centred at node \mathbf{x}_i^K on grid K and the set $C_j^{K \rightarrow K-1}$ contains the nodes of the tetrahedra on grid $K - 1$ that contains \mathbf{x}_j^K . As already mentioned, near curved boundaries \mathbf{x}_j^K may not be inside mesh $K - 1$ and consequently some of the Φ weights will be negative, corresponding to extrapolation. Where this occurred the negative weights were set to zero and the sum of the weights re-normalised to unity, so as to prevent the interpolation creating oscillations. It is believed any inaccuracies due to this will only cause minor local oscillations that should be immediately damped by the smoother. This transfer operation is simpler than that required in [3] to obtain stable computations.

If a nested coarse grid could be generated, eg. by skipping alternate points in a structured hexahedral grid that has been split into tetrahedra, then this restriction is equivalent to injection (ie copying the residual at the nested nodes between grids). This is well known to produce non-optimal performance for Elliptic equations; the optimal treatment uses a full weighted restriction, which averages the copied residual with its neighbours. However, for the unstructured tetrahedral grids, the interpolation procedure itself will produce some averaging, and for the Euler application, further averaging was found to be ineffective.

As part of the grid transfer operation boundary conditions are imposed. For example, as corrections are prolonged from coarse to fine grid, the corrections are set to zero on Dirichlet boundaries, and the normal component of the momentum correction is disregarded on an inviscid wall boundary.

2.3 Smoother

The purpose of the smoother is to damp high frequency error modes in the solution. Assuming that $N(W) = f$ is a nonlinear system whose solution W approximates a partial differential equation over the tetrahedral mesh, the smoothing iteration can be expressed as

$$W^{n+1} = W^n + J^{-1}(f - N(W^n)) \quad n = 1, 2, \dots \quad (2.2)$$

where J is some approximation to the Jacobian of $N(W)$.

To demonstrate the applicability of the multigrid method, the Euler equations

$$\nabla \cdot \mathcal{F} = 0, \quad \mathcal{F} = (f, g, h)^T, \quad \mathcal{F} = \mathcal{F}(q) \quad (2.3)$$

were discretised using a cell vertex finite volume discretisation, with a blend of fourth and second order artificial viscosity. The resulting nodal equations can be expressed as

$$N_j(W) = \frac{1}{V_j} \sum_{\alpha \in C_j} V_\alpha [D_{\alpha,j} \mathbf{R}_\alpha + \mathbf{A}_{\alpha,j}]$$

where

W_j are the conserved variables $(\rho, \rho u, \rho v, \rho w, E)$,

C_j is the set of cells surrounding node j ,

\mathbf{R}_α is the cell based residual ($\oint_\alpha \mathcal{F} \cdot \mathbf{n} ds$) for tetrahedral cell α ,

$D_{\alpha,j}$ are distribution matrices mapping the cell residual to the nodes, derived from a Lax-Wendroff procedure,

$\mathbf{A}_{\alpha,j}$ is a cell based scalar blend of 2nd and 4th order artificial dissipation, which is transparent to a linear solution.

The philosophy adopted in this approach for the spatial discretisation is a generalisation of that described in [6]. The Jacobian is approximated by a simple pseudo-timestep, hence the resulting iteration is explicit in nature, and so is suitable for parallelising. This algorithm requires two loops over tetrahedra (1) to accumulate an undivided Laplacian for the 4th order dissipation, and (2) to perform a Lax–Wendroff update with 2nd and 4th order smoothing. Over 90% of the cpu time are spent in these loops, consequently the work associated with the Euler solver can be assumed to be directly proportional to the number of tetrahedra. This will become an issue when considering the parallel aspects of the method.

2.4 Multigrid Algorithm

The FAS multigrid algorithm is summarised below:

1. Pre-smooth errors on the fine grid by μ_1 relaxations

$$W_h^{n+1} = W_h^n + J_h^{-1}(f_h - N_h(W_h^n)) \quad (2.4)$$

2. Form a coarse grid right hand side:

$$f_H = I_h^H(f_h - N_h(W_h^n)) + N_H(I_h^H W_h) \quad (2.5)$$

and do γ iterations of multigrid on $N_H(W_H) = f_H$ unless it's the coarsest mesh, in which case do n_{crs} smoothing iterations,

3. Prolong the coarse grid correction:

$$W_h := W_h + I_H^h(W_H - I_h^H W_h) \quad (2.6)$$

4. Post-smooth errors on the fine grid by μ_2 relaxations:

$$W_h^{n+1} = W_h^n + J_h^{-1}(f_h - N_h(W_h^n)) \quad (2.7)$$

- 5.

The multigrid cycling parameters are μ_1 and μ_2 for the smoothing and $\gamma = 1$ for V-cycles, $\gamma = 2$ for W-cycles. n_{crs} , the number of iteration on the coarsest grid was always chosen to be 10. The above algorithm is exactly the same as that used on traditional structured grids. Fundamentally there is no extra complexity in the unstructured multigrid over the structured multigrid. However, in practice careful thought needs to put into the design of the software. Unlike the structured grid case;

1. the sizes of the grids bear no relation to each other,

2. the grid connectivity needs to be explicitly stored,
3. extra connectivity is required to link grids together.

The software is written in FORTRAN 77 and arranged so that a table of starting addresses is constructed; these point into a large one dimensional array, which is made multidimensional through subroutine calls. This gives the flexibility to have any number of grids, reuse workspace, and allocate workspace arrays on each grid.

2.5 Multigrid Performance

2.5.1 M6 wing

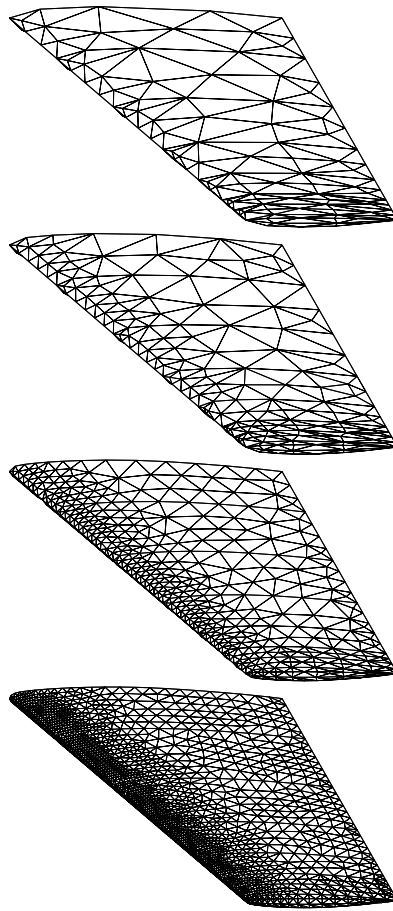


Figure 1: M6 wing sequence of grids for $N=4$

In an attempt to find the best grid sequence, for a fixed finest and coarsest grids, a variety of intermediate grids were generated and the performance analyzed. An M6 wing geometry was chosen for this test. The coarsening parameter

s_N	1.51	1.68	2.0	2.82	1.0
N	6	5	4	3	1
ncell	250000	250000	250000	250000	250000
r	3.1	4.2	6.7	16.3	
ncell	81400	60000	38000	15000	
r	3.1	3.9	5.5	8.9	
ncell	26000	15000	6900	1700	
r	2.8	3.3	3.9		
ncell	9300	4600	1700		
r	2.4	2.7			
ncell	3800	1700			
r	2.2				
ncell	1700				

Table 1: The number of tetrahedra on the differing grid sequences for the M6 wing

ϕ was set to 8 for the coarsest grid and $\phi = 1$ on the finest grid, see Fig. 1. $\phi = 8$ was the largest ϕ for which the grid generator was successful; failure was due to the granularity of the surface spline patches. Grid sequences of N grids were generated by taking powers of s_N , that is

$$\phi = s_N^{i-1} \quad \text{for } i = 1, \dots, N \quad (2.8)$$

where $s_N = 8^{1/(N-1)}$. Table 1 gives the resulting number of tetrahedra on each grid, and the ratio between grids, for all sequences. Clearly, this ratio is not constant for each grid sequence, although this may be a desirable property and is a consequence of the non-linear mesh density functions. Fig 1 shows the surface triangles for the $N = 4$ case.

All convergence plots are work units against \log_{10} of the residual norm, where a work unit is a residual evaluation on the finest grid. The oscillations near the origin is because of the Full MultiGrid (FMG) startup procedure, where the initial solution on each grid is prolonged from a coarser grid.

In order to test the grid sequence efficiency, a robust cycling strategy was chosen, $\mu_1 = \mu_2 = 6$, $\gamma = 2$. Fig 2 shows the convergence for each grid sequence except $N = 2$ that failed. The failure is attributed to an excessive jump in grid density. All the other grid sequences showed a good speed up over the single grid performance, that is, a healthy invariance to the grid sequence is observed.

Fig. 2 also shows the convergence plots for the $N = 4$ grid sequence, with $\mu_1 = \mu =$ set to 6, 3, 2, 1. The actual work required for convergence seems invariant to the amount of pre- and post-smoothing.

Finally, Fig. 2 compares the performance for V and W cycles for $N = 4$, $\mu_2 = \mu_1 = 2$. The W-cycles exhibit a two-fold speed-up over the V-cycles. This

superior performance of W-cycles is often observed in the literature, see [2].

One conclusion of the above experiments is that a good cycling strategy is $\mu_2 = \mu_1 = 2$ with W-cycles, with a tetrahedra ratio between grids of about 4. More importantly, the achieved convergence rates are relatively insensitive to the grid sequence and the cycling strategy used. Fig. 3 shows the solution, and the convergence history of the optimised multigrid against the single grid iteration. A ten-fold speedup is observed.

2.5.2 Aircraft simulation

Using the above strategy, the following grid sequence, for an aircraft configuration, was generated:

ncells	746286
r	4.6
ncells	163768
r	5.9
ncells	27831

The surface triangulations for this grid sequence can be seen in Fig 4. Fig 5 shows the surface pressure contours along with the comparison of single and multigrid convergence. Clearly multigrid is an effective acceleration procedure for such large complex configurations.

3 OPlus: Parallel Computing

The aforementioned multigrid application leads to a complex and intricate code. A one-off parallelisation of such a code using a message passing library such as PVM would be a time-consuming and error prone task, and the resulting code would be difficult to maintain and develop. The purpose of the OPlus library is to take the burden from the application programmer [7]. Emphasis is put on

generality: OPlus uses arbitrary data structures which can include edge/cell/face based data structure for any grid type (triangle/quadrilaterals (2D) or tetrahedra/prisms/hexahedra etc... (or any combination),

performance: Messages are only sent when necessary, are concatenated to reduce latency, and computation is overlapped with computation wherever possible,

portability: OPlus is a library that can be interfaced to general message passing libraries such as PVM and MPI, or the native message passing library of the machine,

single source: A single source code can be executed either sequentially or in parallel, this greatly simplifies development and maintenance of parallel code.

There have been several authors who have pursued the idea of constructing a library for parallelising programs for MIMD machines. These are mainly aimed at structured grids. De Keyser developed a software tool called LOCO [8] for structured grids with refined regions. Dellagiacoma *et al* constructed PARAGRID [9] which uses overlapping domain decomposition techniques on block-structured grids. Williams created DIME (Distributed Irregular Mesh Environment) [10] and Das *et al* developed PARTI (Parallel Automated Runtime Toolkit at ICASE) [11] for unstructured mesh algorithms on distributed machines. However, DIME restricts the user to two dimensional triangular grid calculations. PARTI parallelises problems in any dimensions, but is not believed to be as communication efficient and easy to use as the current work.

The OPlus framework uses a data parallel approach. Consequently certain algorithmic restrictions must be enforced to ensure the solutions from parallel and sequential executions are identical, to within machine accuracy. For example, algorithms involving matrix-vector multiplications can be handled, such as conjugate gradient methods with simple preconditioning, explicit time stepping methods and Jacobi relaxation. Excluded algorithms are globally implicit methods, such as ADI and sweeping relaxation schemes like Gauss-Seidel.

3.1 Top Level Concepts

The concept behind the OPlus framework is that unstructured grids can be described by a number of *sets*. Depending on the application, these sets might be of nodes, edges, triangular faces, quadrilateral faces, cells of a variety of types, far-field boundary nodes, wall boundary faces, etc. Associated with these sets are both *data* (e.g. coordinate data at nodes, volumes at cells, normals on faces) and *pointers* to other sets (e.g. edge pointers to the two nodes at each end of the edge). All of the numerically-intensive operations can then be described as a *function* operating on all members of a set (eg. looping over cells, calculating volumes from the nodal coordinate values). The function can operate on data associated directly with the set or with another set through a pointer.

The OPlus framework makes the important restriction that a function will operate on all members of a set and the order it operates on the members will not affect the final result. If a function needs to be applied on a subset of members then another set should be formed containing only these members. Consequently, the OPlus routines can choose an ordering to achieve maximum parallel efficiency. It is this restriction that negates the framework being applied to Gauss-Seidel relaxation or globally implicit methods.

Before parallel execution can proceed, three main initialisation phases take place:

partitioning All sets are partitioned and each partition is assigned an *owner* processor. Processors must compute and store the values of the members they own.

halo construction If the application of a function to an element of a set (and other data pointed to by that element) requires data which is not locally owned that element is referred to as a “halo” member. The data at the halo member itself may, or may not, be owned. Copies of the required data must be communicated before halo execution.

local renumbering All sets are locally renumbered to minimise the memory requirements of each process. The pointers have to be similarly renumbered to ensure consistency.

It is important to note that all of the above operations are performed automatically by the OPlus library, not the application, which only need specify the sets and pointers.

3.2 Input/Output

Input/Output

An aim of the framework is to allow users to write a *single source code* which will execute either sequentially or in parallel depending on how the executable is linked. To achieve this it is necessary for the program to handle all disk and terminal i/o via appropriate subroutines.

1. For sequential execution the user’s main program is linked to user-written subroutines which handle all i/o. This will enable the user to develop, debug and maintain their sequential code without any parallel message passing libraries.
2. For parallel execution the OPlus framework creates server and client programs from the user’s single source, Fig. 6. The server program is formed by linking the OPlus server process to the user’s i/o routines, while the client program is created by linking the user’s compute process to OPlus client routines.

The current implementation of this client server model uses two message passing systems.

PVM to communicate between the server and clients, thus giving the flexibility to having the server on any machine, and the possible poor

performance of using PVM is irrelevant because i/o should not be a bottleneck within the OPlus framework.

BSP FORTRAN is used for client–client communication [12]; this is designed to be portable and give high parallel performance for SPMD applications on homogeneous machines and so is ideally suited for the client communication.

Together this gives a flexible and efficient solution to the portable performance problem.

3.3 Loop Syntax

The following example is given to illustrate how a loop is parallelised. Suppose that a triangular cell area, **AREAC**, is distributed to the cell's nodes using a pointer, **NCELL**, which points from the cell to its three nodes. This can be carried out for all the cells in the triangular mesh by the following DO-loop:

FORTRAN loop

```

DO IC = 1, NCELLS
  I1 = NCELL(1, IC)
  I2 = NCELL(2, IC)
  I3 = NCELL(3, IC)
  AREAN(I1) = AREAN(I1) + AREAC(IC)/3.0
  AREAN(I2) = AREAN(I2) + AREAC(IC)/3.0
  AREAN(I3) = AREAN(I3) + AREAC(IC)/3.0
ENDDO

```

This becomes, using the OPlus framework:

FORTRAN OPlus loop

```

DO WHILE(OP_PAR_LOOP(NCELLS, ISTART, IFINISH))
  CALL OP_ACCESS_R8('read' , AREAC, 1, NCELLS, NULL , 0, 0, 1, 1)
  CALL OP_ACCESS_R8('update', AREAN, 1, NNODES, NCELL, 1, 1, 1, 3)
  DO IC = ISTART, IFINISH
    I1 = NCELL(1, IC)
    I2 = NCELL(2, IC)
    I3 = NCELL(3, IC)
    AREAN(I1) = AREAN(I1) + AREAC(IC)/3.0
    AREAN(I2) = AREAN(I2) + AREAC(IC)/3.0
    AREAN(I3) = AREAN(I3) + AREAC(IC)/3.0
  ENDDO
END WHILE

```

Essentially the DO WHILE loop is similar to a colouring loop that would be necessary for vectorisation. OP_PAR_LOOP is a logical function which returns as arguments the start and finish indices of the inner loop. For sequential execution, ISTART and IFINISH are set to 1 and NCELLS respectively. OP_ACCESS tells the library how the arrays in the main loop are to be accessed and enables the library to perform runtime data dependency analysis. The contents of the inner loop have not changed during parallelisation. Full details of the arguments and other routines can be found in [13]. With this syntax the OPlus library can concatenate client to client messages to reduce latency, minimise the size of message sent and overlap communication with computation.

3.4 Parallel Performance

Multigrid using W-cycles on five aircraft grids represents a challenge to both the flexibility and the performance of the OPlus library. The sets on each grid associated with this problem are:

1. tetrahedra,
2. nodes,
3. boundary faces,
4. boundary nodes.

The grid sequence is that seen in Fig. 4 with intermediate grids. The pointers per grid level required for this calculation are:

pointer from	to	length
nodes	fine nodes	4
nodes	coarse nodes	4
tetrahedral cell	nodes	4
boundary faces	nodes	3
boundary nodes	nodes	1

The partitioning strategy used is to partition the finest grid by geometric inertial bisection, and let the coarse grids “inherit” the fine grid partition by looking through the grid-grid pointer tables. This is performed automatically by the OPlus library. The advantage with this method is that data remains local throughout the multigrid cycle; however, the coarse grids will not be load balanced, ie. inheriting the partitions on the coarse grids will not ensure the same number of tetrahedra on each partition. Another load balancing issue is the redundant computations that are performed because of the OPlus parallel strategy. Thus, there is a limit L on the achievable speed up caused by partitioning and redundant computation given by

$$L = \frac{\text{sequential work}}{\text{max slave work}} \quad .$$

Since all the work in the Euler code is performed over tetrahedra this can be evaluated for grid i to be

$$L_i = \frac{T_i}{\max_{\alpha} T_i^{\alpha} \times p}$$

where T_i is the number of tetrahedra on the complete grid i and T_i^{α} is the number of tetrahedra executed on partition α . For a multigrid iteration this becomes

$$L_{MG} = \frac{\sum_{i=1}^{N-1} c\gamma^{i-1} T_i + n_{crs} T_N}{\sum_{i=1}^{N-1} c\gamma^{i-1} \max_{\alpha} T_i^{\alpha} + n_{crs} \max_{\alpha} T_N^{\alpha}}$$

where $c = \mu_1 + \mu_2 + 1$. For the five level multigrid on eight processors these limits are

L_5	5.2
L_4	5.9
L_3	6.2
L_2	6.2
L_1	6.6
L_{MG}	6.1

where L_1 is the finest grid and L_5 is the coarsest grid. Clearly the effect of inheriting the partitions from grid 1 is manifesting itself in a decreasing L for coarser grids. However, since less work is performed on the coarse meshes, L_{MG} is dominated by the partitioning of the finest grid.

Calculations were performed on the following machines:

- an 8 node distributed memory IBM SP1,
- 4 processor shared memory SGI Power Challenge.

The time spent partitioning, calculating local numbers and reading all the pointer tables was less than 60 sections elapsed time; this is thought to be negligible. The elapsed times, in seconds, per multigrid cycle, along with the speed up achieved (SU) is given in the following table.

p	L_{MG}	IBM SP1		SGI PC	
		time	S U	time	S U
1	1.0	1006	1.0	419	1.0
2	1.9	556	1.8	216	1.9
3	2.7	384	2.6	149	2.8
4	3.5	310	3.2	116	3.6
5	4.1	270	3.7	—	—
6	4.8	234	4.3	—	—
7	5.4	211	4.8	—	—
8	6.1	190	5.3	—	—

Clearly the L_{MG} limit on the performance is more significant than the communication cost, and so if faster times are sought the partitioning strategy should be reevaluated. It is possible with the OPlus framework to independently partition each grid but this would incur a communication penalty when performing grid transfers. The SGI Power Challenge managed to run faster than L_{MG} using four processors, this is accounted for by better cache utilisation on the smaller partitioned grids.

The key point of this table is that worthwhile parallel speed up has been achieved for a highly complex, worst case multigrid method with little user intervention.

4 Visualisation

For such large applications it becomes prohibitive to rely on graphics workstations to manipulate and render the complete solution. To remedy this, the compute intensive part of the visualisation is performed on the parallel machine, along with the flow solver, while all the rendering is performed on the graphics workstation, thus maximising the use of both sets of hardware. This uses the pV3 software [14], which fits easily in the OPlus framework.

5 Conclusions

An unstructured multigrid algorithm has been found to be effective for large scale geometrically complex configurations. The performance of the multigrid has been

found to be insensitive to the grid sequence and multigrid cycling strategy.

A flexible and general approach has been demonstrated to parallelise unstructured grid applications. This involves the programmer adopting the OPlus loop style of programming and all i/o being sent through specific subroutine calls. The resulting code will execute on a sequential machine (without the need for *any* parallel libraries) or in parallel (on a MIMD architecture). This single source is of major benefit for the development and maintenance of the code.

The OPlus parallel execution is fully optimised to concatenate messages, minimise number of messages sent and overlap communication with computation. This library is intended for large applications, which warrant the use of parallel machines, and has been demonstrated by the aforementioned 3D Euler multigrid solver for a complete aircraft configuration. For this realistic industrial application a worthwhile speed up has been achieved with very little effort from the application programmer.

References

- [1] P. Wesseling. *An introduction to multigrid methods*. John Wiley, 1992.
- [2] D. Mavriplis and A. Jameson. Multigrid solution of the Euler Equations on unstructured and adaptive meshes. *ICASE Report 87-53*, 1987.
- [3] J. Peraire, J. Peiro, and K. Morgan. A 3-D finite element multigrid solver for the Euler equations. AIAA Paper 92-0449, 1992.
- [4] K. Morgan, J. Peraire, and J. Peiró. Unstructured grid methods for compressible flows. *AGARD*, R-787:5.1-5.39, May 1992.
- [5] V. Venkatakrishnan and D.J. Mavriplis. Agglomeration multigrid for the tree-dimensional Euler equations. ICASE Report No. 94-5, 1994.
- [6] P.I. Crumpton, J.A. Mackenzie, and K.W. Morton. Cell vertex algorithms for the compressible Navier-Stokes equations. *J. Comput. Phys.*, 109(1):1-15, 1993.
- [7] D. A. Burgess, P. I. Crumpton, and M. B. Giles. A parallel framework for unstructured mesh solvers. IFIP WG10.3 Working Conference on Programming Environments for Massively Parallel Distributed Systems, 1994.
- [8] J.De Keyser. *LOCO1.0: a library supporting data parallelism on MIMD computers*. Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, March 1993.

- [9] F. Dellagiacoma, S. Paoletti, F. Poggi, and M. Vitaletti. PARAGRID: a parallel multi-block environment for Computational Fluid Dynamics. IBM ECSEC, Viale Oceano Pacifico 173, 00144 Rome, Italy.
- [10] R. D. Williams. *DIME Distributed Irregular Mesh Environment*. Report C3P 861, Cal. Tech. Pasadena, CA, 1990.
- [11] H. Berryman, J. Saltz, and J. Scroggs. Execution time support for adaptive scientific algorithms on distributed memory architectures. *Concurrency: Practice and experience*, 3(3), 1991.
- [12] R. Miller. A library for bulk-synchronous parallel programming. British computer society parallel processing <http://www.comlab.ox.uk/oucl/oxpara/bsplib.html>, 1993.
- [13] P.I. Crumpton and M.B. Giles. OPlus programmer's manual. Oxford University Computing Laboratory, 1993.
- [14] R. Haimes. pV3: A distributed system for large scale unsteady CFD visualisation. *AIAA Paper 94-0321*, 1994.

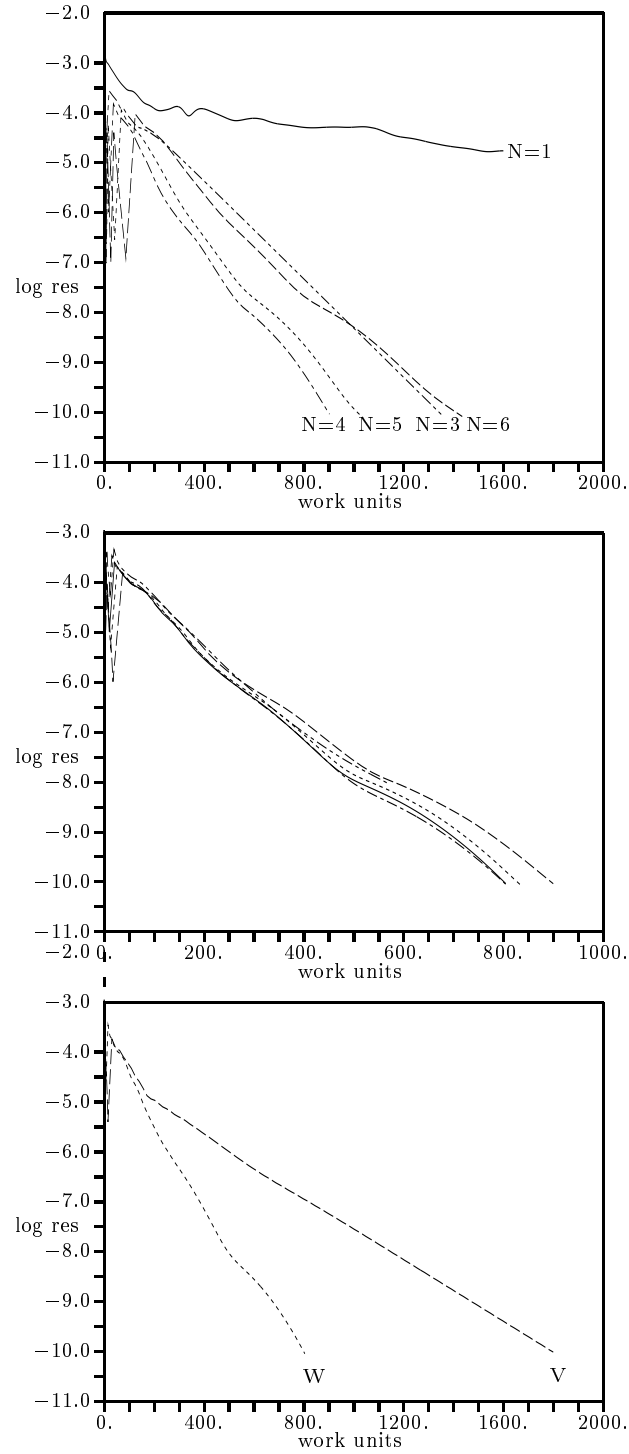


Figure 2: Top: comparison of convergence for grid sequences $N=1,2,3,6$; Middle: comparison of convergence for $\mu_1 = \mu_2 = 1, 2, 3, 6$, Bottom: comparison of convergence for W and V cycles, $\mu_1 = \mu_2$.

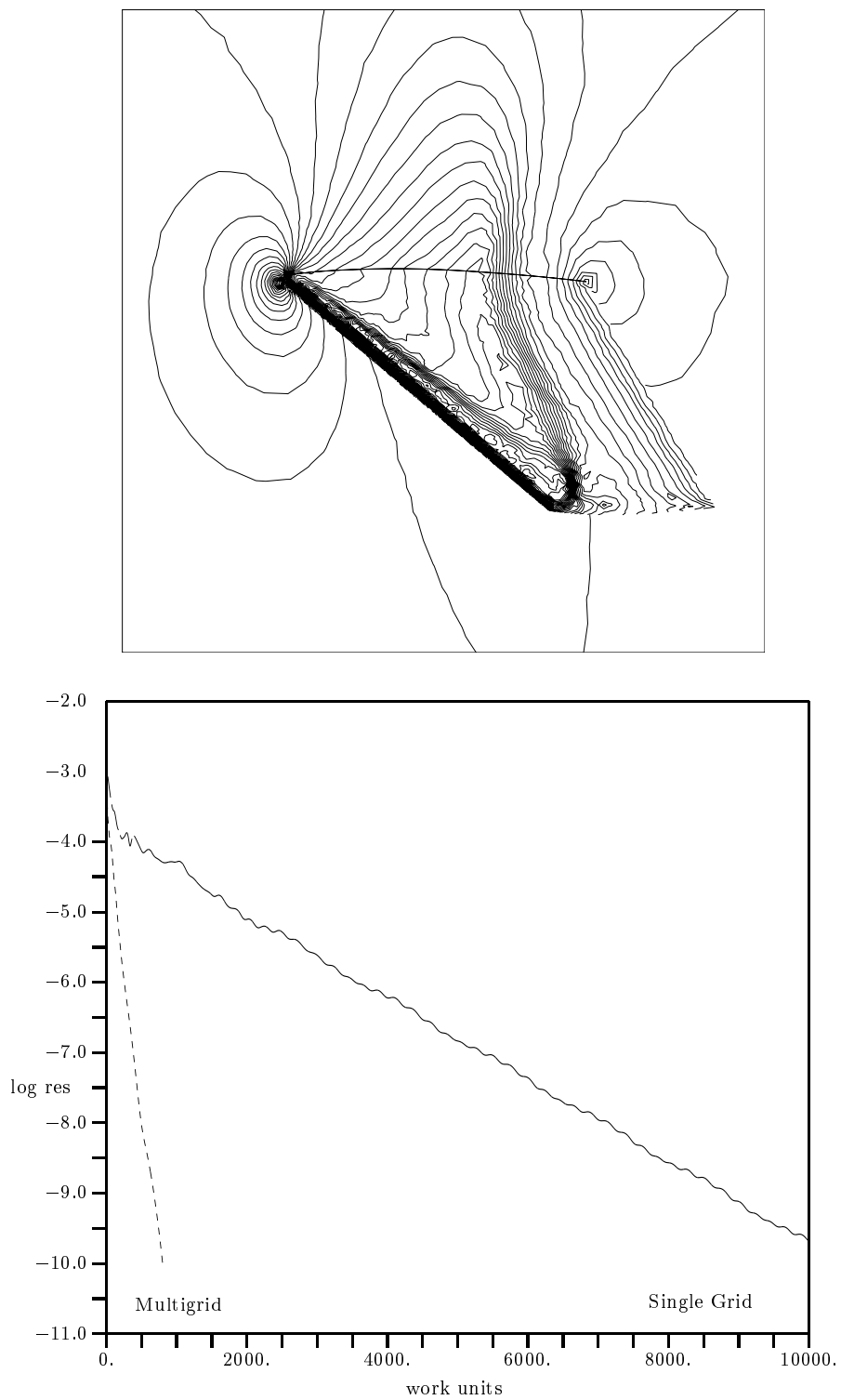


Figure 3: Contours of pressure and convergence history for the M6 wing

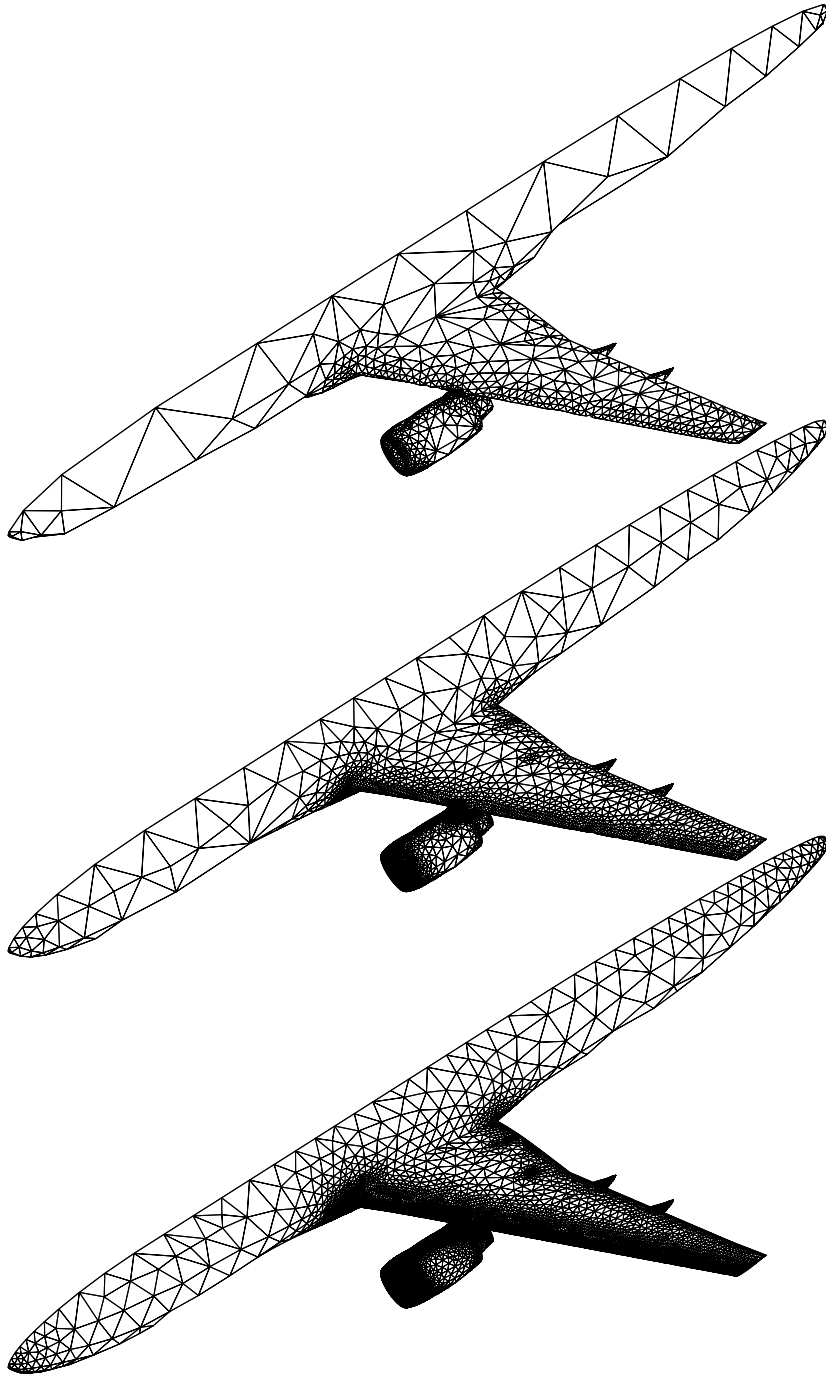


Figure 4: The sequence of grids used for the aircraft configuration

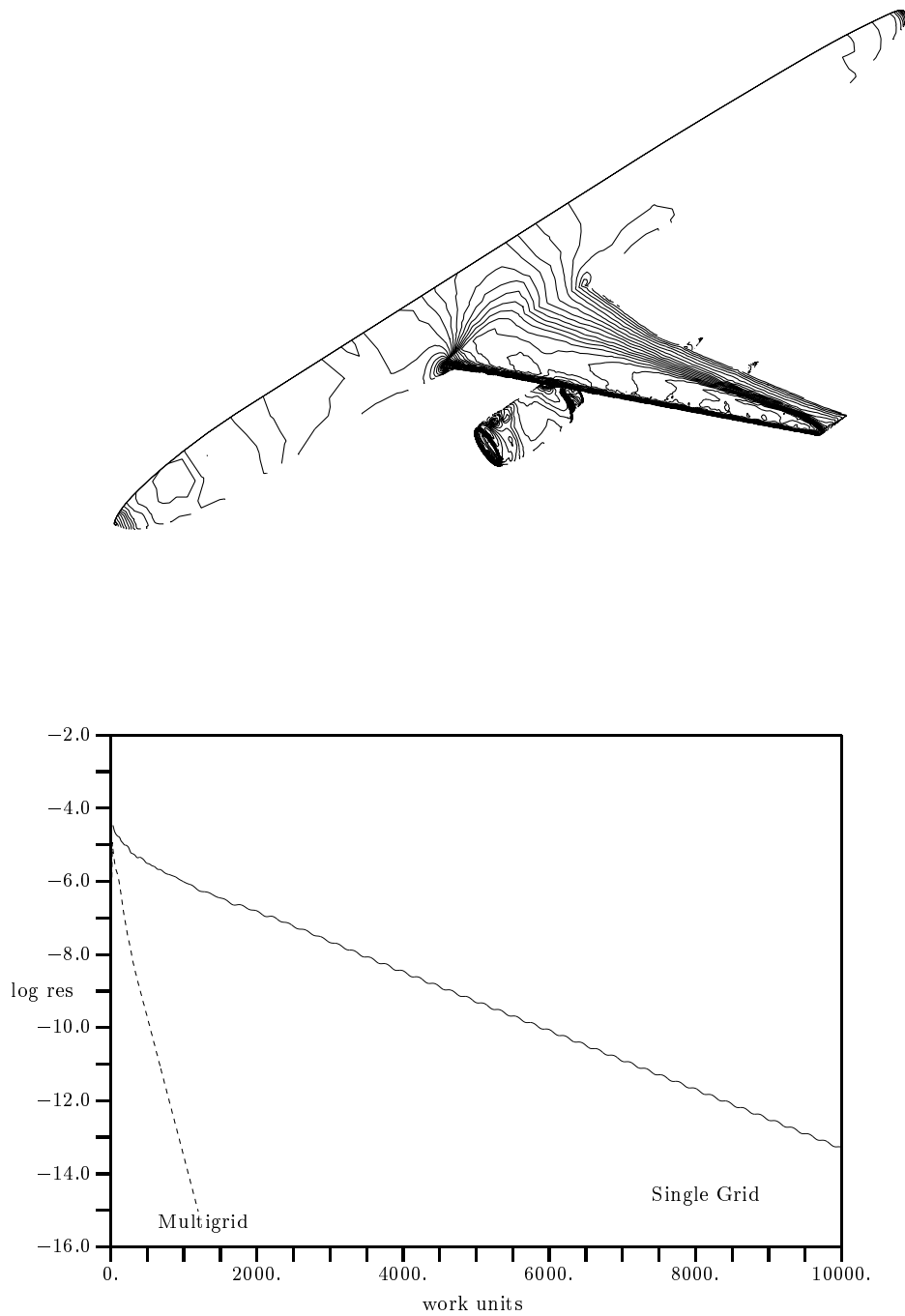


Figure 5: Contours of pressure and convergence history for the aircraft configuration

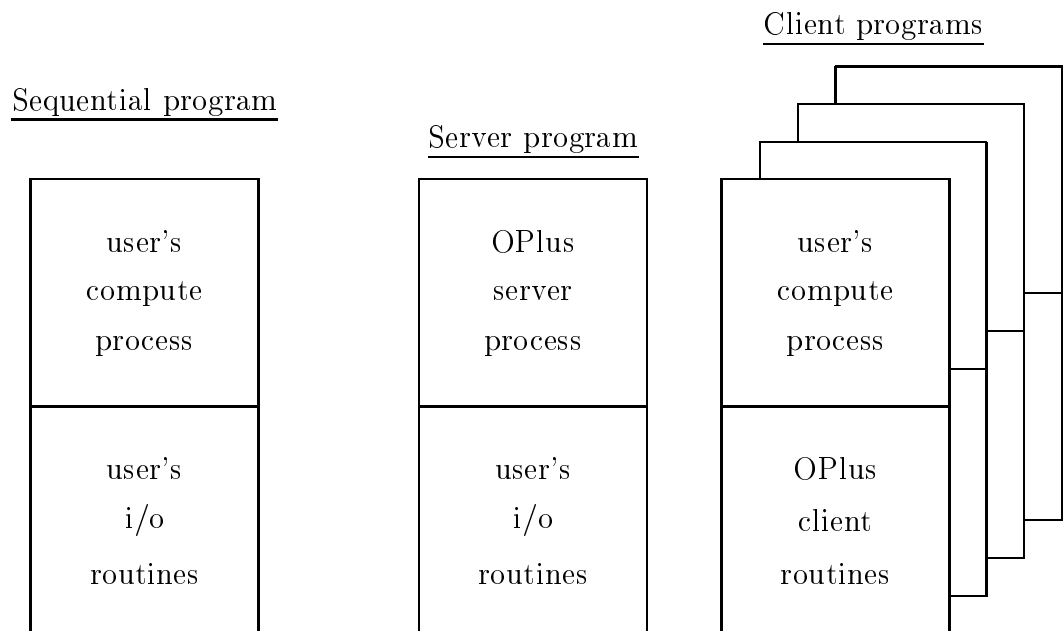


Figure 6: Sequential and parallel versions of user's program