University of Glasgow

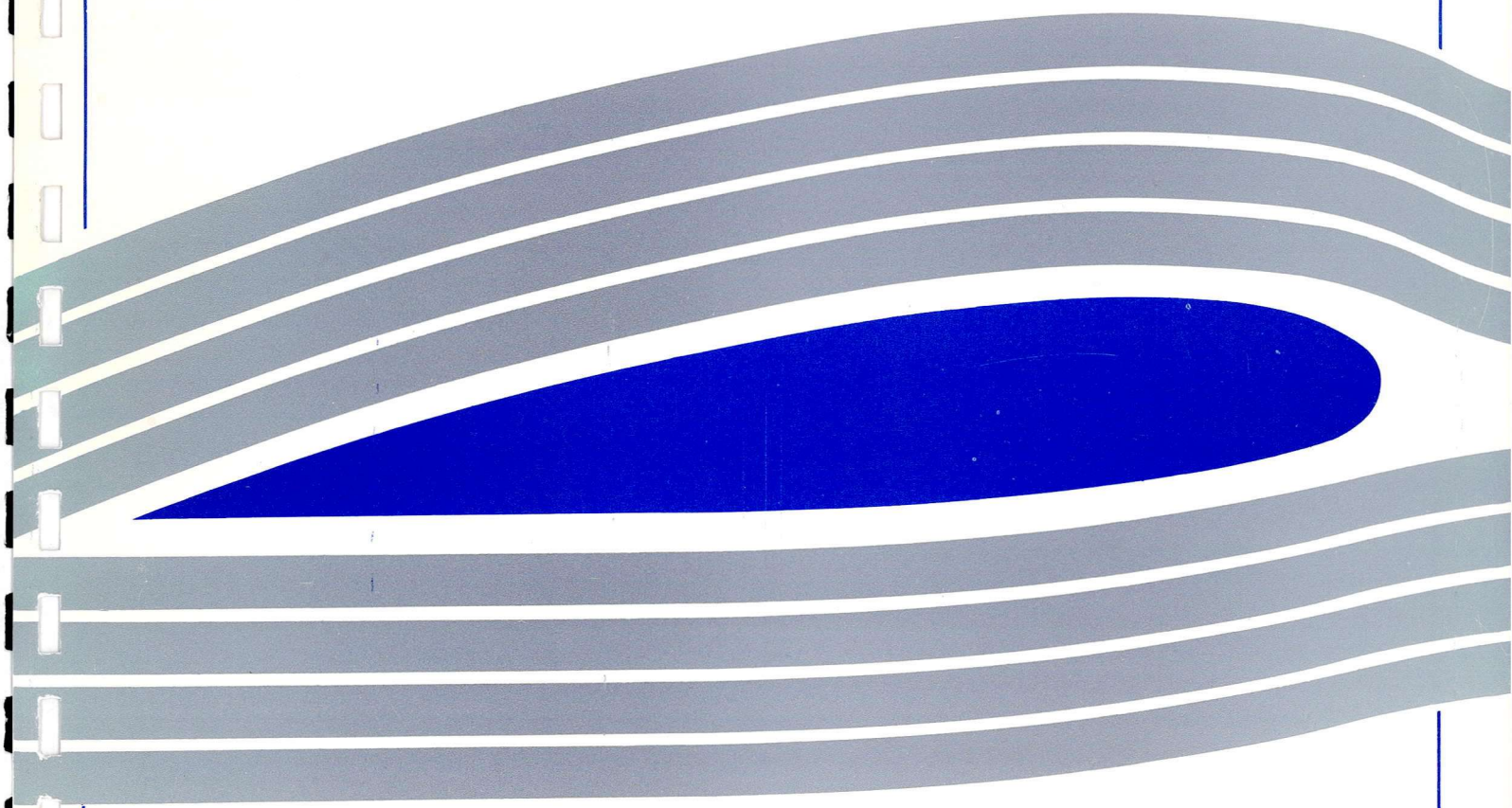
DEPARTMENT OF

AEROSPACE ENGINEERING

Progress Report : Application of the Multiblock Method in Computational Aerodynamics

Brian J. Gribben

Aero. Report 9621
12 September 1996



Progress Report : Application of the Multiblock Method in Computational Aerodynamics

Brian J. Gribben

Aero. Report 9621
12 September 1996

Aerospace Engineering Department
University of Glasgow,
Glasgow, G12 8QQ, U.K.

Abstract

This report serves as a record of the progress made since October 1995 as a postgraduate research student studying in the field of computational aerodynamics. The area of interest is the application of the multiblock method to examine real problems in aerodynamics. The experience gained in using various multiblock grid generation packages is discussed, along with an examination of the load balancing problem for parallel execution of aerodynamic flow solvers. Some initial results from the development of a static load balancer based on the method of simulated annealing are presented.

1 Introduction

The structured grid approach in CFD affords the advantage of easier calculation management than the unstructured approach, but the disadvantage that grid generation is considerably more difficult for complex configurations. An unstructured grid can generally be created around any complex shape, whereas single-block structured grids can only be used on domains which can be mapped onto rectangular parametric space. A useful compromise between the two approaches is provided by the multiblock technique [1][2]. The multiblock grid consists of an unstrutured arrangement of structured grids. This preserves in essence the advantages of each method, and at the same time generates a host of other considerations. Grid generation is a fundamental one. In this subdiscipline of CFD single-block structured grid generation, especially around single aerofoils in two dimensions, is a solved problem. However, the generation of multiblock structured grids, although quite obviously based on single-block algorithms, is still a restrictively time-consuming task even in two dimensions. The problem lies in the need for detailed user input and judgement at nearly every stage in the process, i.e. the lack of automation. Progress in this area seems to have centred on facilitating user evaluation and input via graphical user interfaces rather than trying to automate the task. Reference [3] provides a very good review of this area.

The management of the interfaces between blocks in the flow solver can be handled in a variety of ways, making a compromise between flow solver simplicity, efficiency and grid generation issues. For example, some flow solvers require that grid lines are continuous across block interfaces. This makes the task of managing the calculation across the interface easiest but has an efficiency penalty; grid point clustering in one block, for example to resolve a boundary layer, must be carried through to the adjacent blocks where such a node density is not required. Allowing only one boundary condition per block face certainly keeps the interface management simple but means that a large number of small blocks may be necessary, which could have a detrimental effect e.g. on preconditioning [4]. Examples in the literature of succesful applications of the multiblock method for complex configurations are numerous ([5]-[12]), and in each case a suitable block management scheme is employed. However, a general scheme which is efficient and does not require special treatments for each case is difficult to obtain.

The solution of large CFD problems requires considerable computational effort. Parallel computing is one increasingly attractive route to obtaining the computer power needed for such simulations, both in terms of memory and processing speed. The domain decomposition inherent in the multiblock method provides an obvious division of work for parallelization. To use a parallel resource efficiently for any given problem, the issue of load balancing must be addressed. In CFD, this is largely concerned with mesh partitioning i.e. allocating sub-domains to processors. Communication time between processors must also be taken into account. Successful applications of CFD on parallel platforms are varied and numerous, see for example [13], but the impetus to develop improved techniques to enable consideration of larger, more complex problems still exists.

Considerable effort is being made by the CFD group at Glasgow to develop a generally applicable parallel multiblock flow solver [4]. The code employs a cell-centered finite volume method using a high-order upwind discretisation which is solved using an implicit scheme. The author's par-

ticular contribution to the group effort takes the form of examining available grid generation tools for complex configurations, developing a method for efficient parallel implementation, and applying the code to complex configurations of interest to DRA Bedford. At present there is a working two-dimensional unsteady Euler code, with work on the viscous, turbulent code nearly complete. Message passing is impemented using PVM [14]. The present Euler code has been verified and validated.

To provide a comparison with the results of the parallel multiblock code (PMB), results are also presented for the EAGLE inviscid flow solver. The run times given throughout are for a Silicon Graphics Indy R4400. To demonstrate that EAGLE does produce reasonable results (making timing comparisons valid), results are presented for transonic flow around the NACA0012 aerofoil with a freestream Mach number of 0.8 at zero incidence, Figure 1. Both the PMB and EAGLE results show good agreement with results from two established inviscid solvers [15],[16]. The oscillation in the EAGLE results downstream of the shock is the only real difference. For this case, PMB took 190s CPU for the residual to reduce 12 orders, and the EAGLE run took 4400s for the residual to reduce 4 orders.

In the remainder of this report, the progress made towards developing a multiblock grid generation capability will be reveiwed and the relative merits of the packages used will be discussed. Also presented are the initial steps taken in creating a mesh partitioning and load balancing tool for the parallel multiblock flow solver. Finally the future directions to be taken are discussed.

2 Grid generation

2.1 EAGLE

EAGLE (Eglin Arbitrary Geometry ImpLicit Euler) [17] is a multiblock grid generation and flow solution code developed at Mississippi State University for the U.S.A.F. The grid generation part, often referred to itself as "EAGLE", uses elliptic smoothing on initial algebraic grids. EAGLE has its own command language, the user producing a "script" of commands which is executed by the code. The multiblock grid is constructed in a step-by-step manner starting with points in space and progressing through curve definitions to the final grid. The user has a large degree of control over geometries, node spacings and how the grid is to be generated. From the point of view of numerical grid generation techniques, EAGLE is a very complete software module and has become something of a benchmark for structured grid generation. However the need to learn a complicated command language and the lack of a graphical user interface (GUI) means that EAGLE is in general difficult and time-consuming to use.

As a first attempt at creating a multiblock grid, EAGLE was used to create an eight block grid around a tandem aerofoils configuration, Figure 2. There are some obvious imperfections in this grid, notably cell skewness in several regions. This is not the fault of the grid generation algorithm but rather the result of poor shaping of the block edges. Fine manipulation of the curves forming the block edges is difficult in EAGLE. A point in favour of EAGLE is the relatively good grid line slope continuity achieved here in a somewhat sub-standard topology.

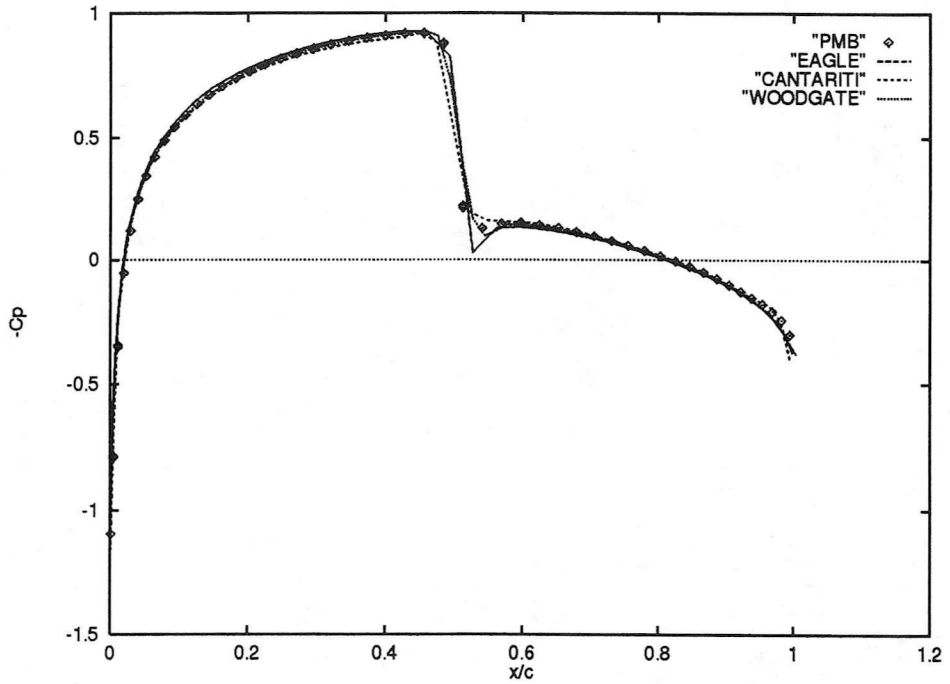


Figure 1: *Pressure coefficient, NACA0012 aerofoil (M=0.8, α=0.0)*

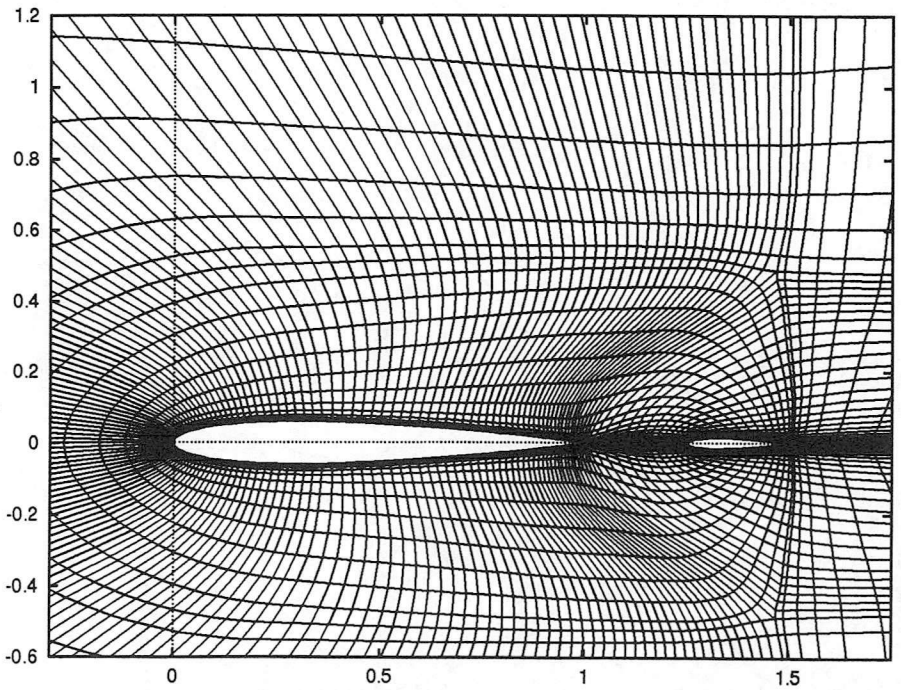


Figure 2: *EAGLE grid for tandem aerofoils case*

2.2 Geomesh

Geomesh is the grid generation module for the RAMPANT commercial CFD package. Geomesh can be used to generate 2-D or 3-D, structured or unstructured meshes and includes an impressive modern GUI. Complex geometry modelling is possible using a CAD preprocessor. The package is aimed at novice CFD users so strong emphasis is placed on user-friendliness making Geomesh a very easy tool to use for designing multiblock grids.

To define the initial geometry, the user can choose to use the associated CAD package, read in a list of coordinates, or import IGES formatted geometries from other CAD packages. Each block is constructed using a 'rubber- banding' technique where the user shapes a given block template as desired. Fine control over the shape of the block edges and the stretch functions which define node spacings are features of Geomesh.

Despite the superior GUI and associated short design cycle, Geomesh has some serious drawbacks. Firstly, the grid generation algorithms themselves are fairly basic compared to, say, those in EAGLE. Despite the array of adjustable factors at the user's disposal to control the grid generation, it is not in general possible to achieve better than a basic algebraic grid. The smoothing and orthogonality capabilities of Geomesh are very poor. Another problem is that in the RAMPANT package, the Geomesh grids are designed to be passed to the RAMPANT flow solver (in the form of unstructured binary dumps) and not to an external flow solver. This means that obtaining a grid created in Geomesh in an external format is difficult, and requires the use of data filters to interpret the native format. Writing these filters is a non-trivial task. In addition, unwanted merging of grid points can occur in Geomesh when dimensions are small, turning a quadrilateral cell into a triangular cell. Geomesh does not produce an error message when this occurs so it is difficult to detect. To circumvent this problem it is necessary to scale up the model by a large factor (say 1000) before generating the grid, and to scale back down before using the grid in the flow solver. Geomesh is also unable to produce grids with singular lines, as often required in missile-type configurations.

Geomesh was used to create a grid around the tandem aerofoils configuration (Figure 3) for comparison with the grid produced in EAGLE (Figure 2). This grid has a different, improved topology from its EAGLE-generated counterpart. This can be put down partly to the author's improved experience when this grid was generated and partly to the ease with which new topologies can be visualised and created in Geomesh. The Geomesh grid has far less cell skewness but poorer orthogonality at the solid surfaces and some slope discontinuity problems at block boundaries. An inviscid version of the flow solver was run for a transonic case on this mesh. The pressure contours obtained are shown in Figure 4, and the pressure contours in Figure 5. The parallel multiblock code (PMB) took 1866s CPU for the residual to reduce 8 orders, and the EAGLE run took 15861s for the residual to reduce 4.5 orders.

As another exercise in multiblock grid generation, Geomesh was used to create a grid around a staggered biplane configuration consisting of two NACA0012 aerofoils, Figure 6. Two five-point singularities are present in this mesh. There are some areas of the mesh which could be worked on to improve orthogonality and smoothness, but an important point to take from this example is that the grid took less than one morning to generate from scratch and a reasonable flow solution was obtained, Figure 7. Again the EAGLE flow solver was also used for com-

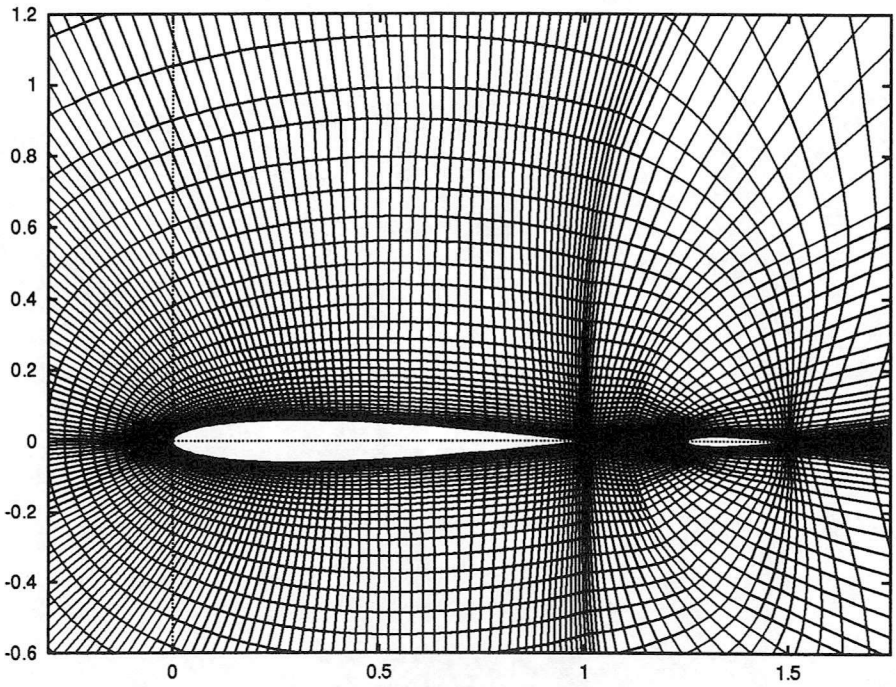


Figure 3: *Geomesh grid for tandem aerofoils case*

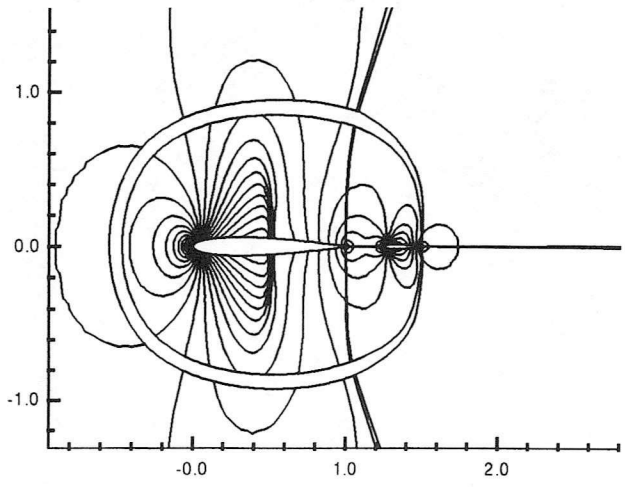


Figure 4: *Tandem aerofoils case pressure countours (M=0.8, α=0.0)*

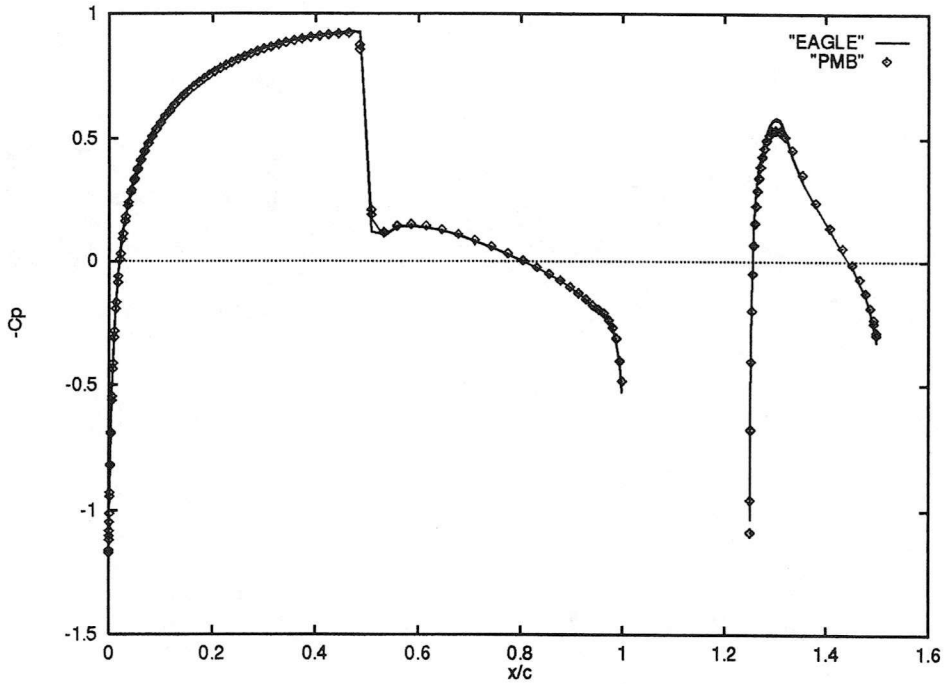


Figure 5: *Tandem aerofoils case pressure coefficient (M=0.8, α=0.0)*

parison, Figure 8. For this case, PMB took 2084s to reduce the residual by 8 orders, and EAGLE took 16852s to reduce the residual by 4 orders.

Geomesh was used to construct a multiblock grid for the Williams aerofoils, configuration B [18], Figure 9. One five point singularity was used in the slot region. Figure 10 shows the pressure contours obtained for transonic conditions with a freestream Mach no. of 0.58 and zero incidence. Comparison is possible with a full potential solution [19], Figure 11. Again a timing comparison can be made with the results from the EAGLE flow solver: for this case PMB took 1905s to reduce the residual by 8 orders and EAGLE took 23558s to reduce the residual by 5 orders. The present results are very similar to other inviscid solutions to this case which appear in the literature [20], [21].

2.3 EAGLEView

EAGLEView is a grid generation package which is based on EAGLE but incorporating a GUI. The aim of EAGLEView is to add ease of use to the attractive features that EAGLE already has. Put simply, EAGLEView's GUI enables EAGLE commands to be executed by pointing and clicking. The GUI is not as advanced as that of Geomesh but is generally sufficient. The user also has the option to edit a script as in EAGLE if desired. Several options are available for input and output formats. By taking advantage of this, the entire multiblock grids generated in Geomesh for the tandem aerofoils and Williams configuration B cases were read in to EAGLEView with the intention of improving them using EAGLE's superior elliptic smoothing technology. Figure 12 shows the tandem aerofoils grid obtained using EAGLEView. This grid has improved smoothness and orthogonality compared to the original Geomesh grid, Figure 3. The

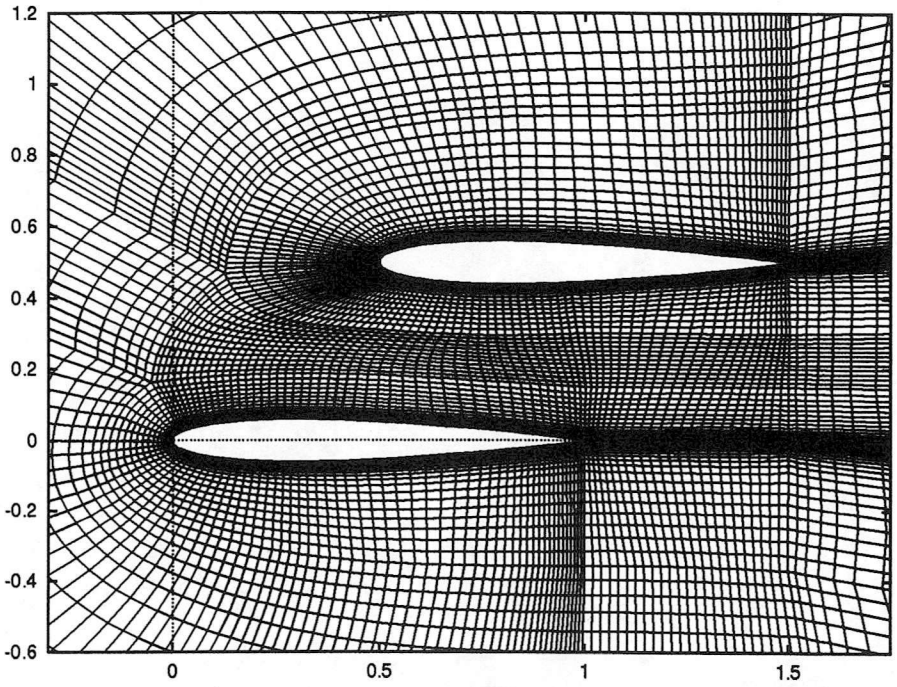


Figure 6: *Geomesh grid for biplane configuration*

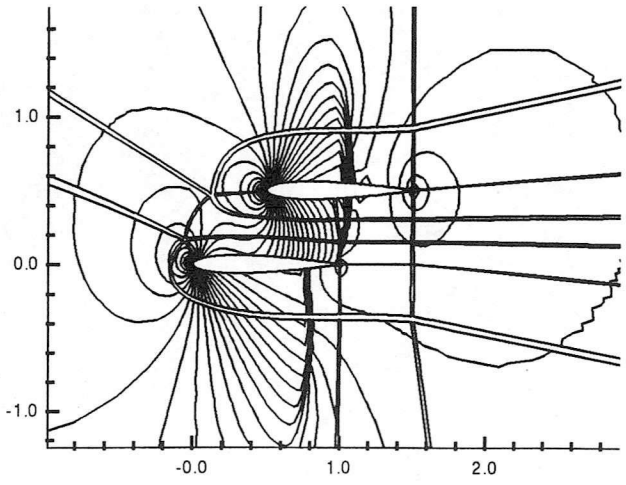


Figure 7: *Biplane configuration pressure countours (M=0.8, α=0.0)*

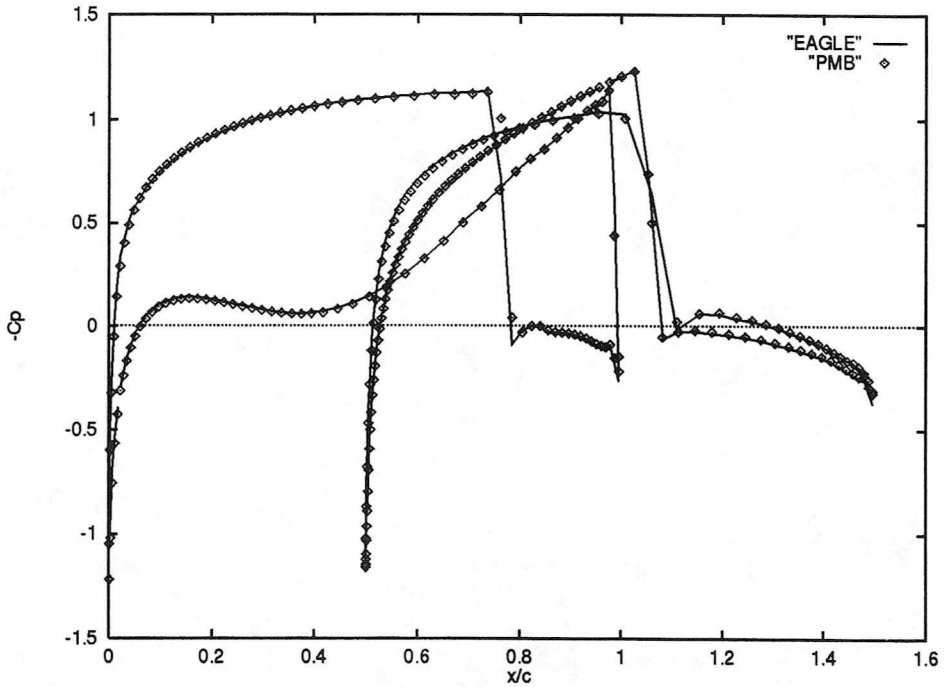


Figure 8: *Biplane configuration, pressure coefficient (M=0.8, α=0.0)*

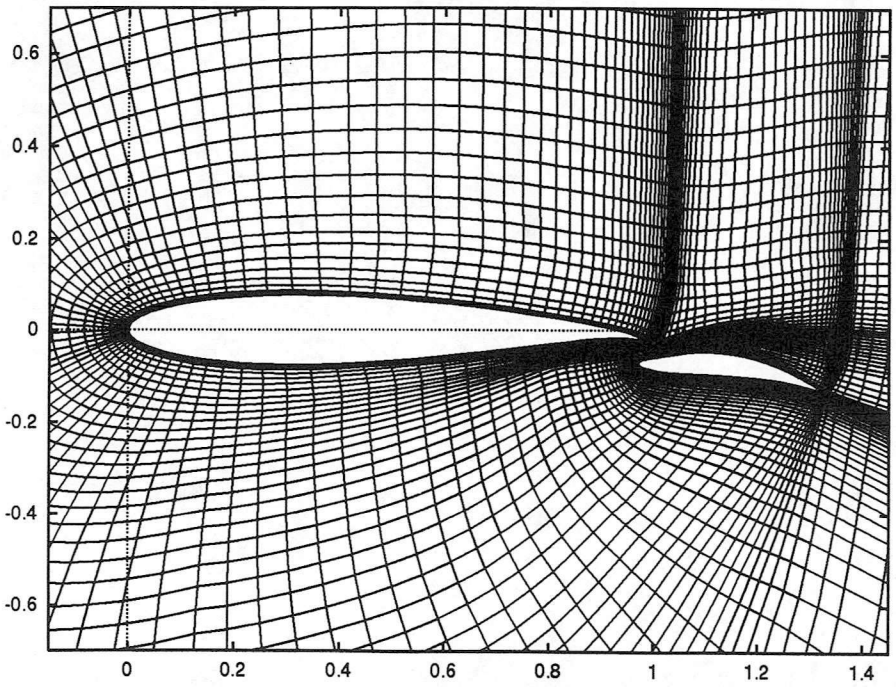


Figure 9: *Geomesh grid for Willaims aerofoil, configuration B*

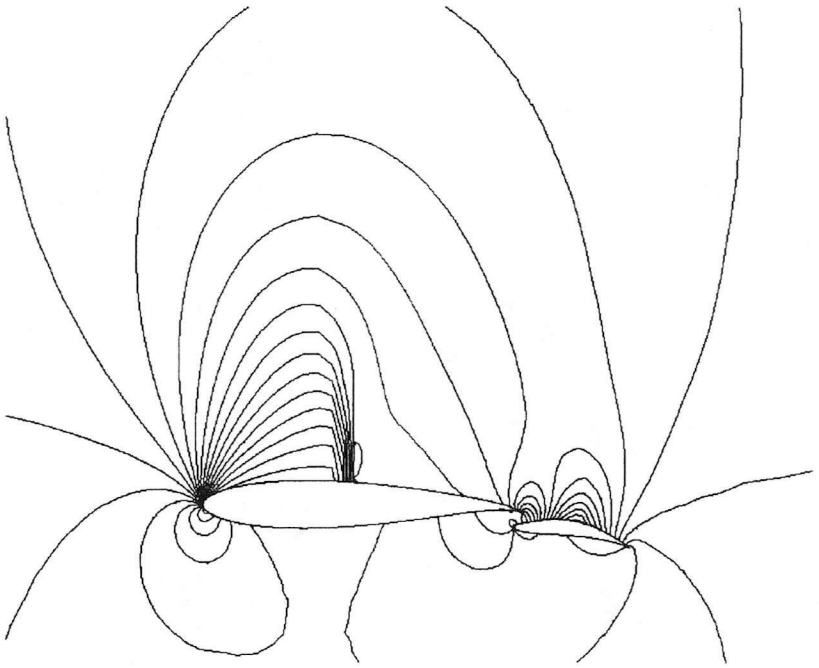


Figure 10: *Pressure contours for Willaims aerofoil, configuration B (M=0.58, α=0.0)*

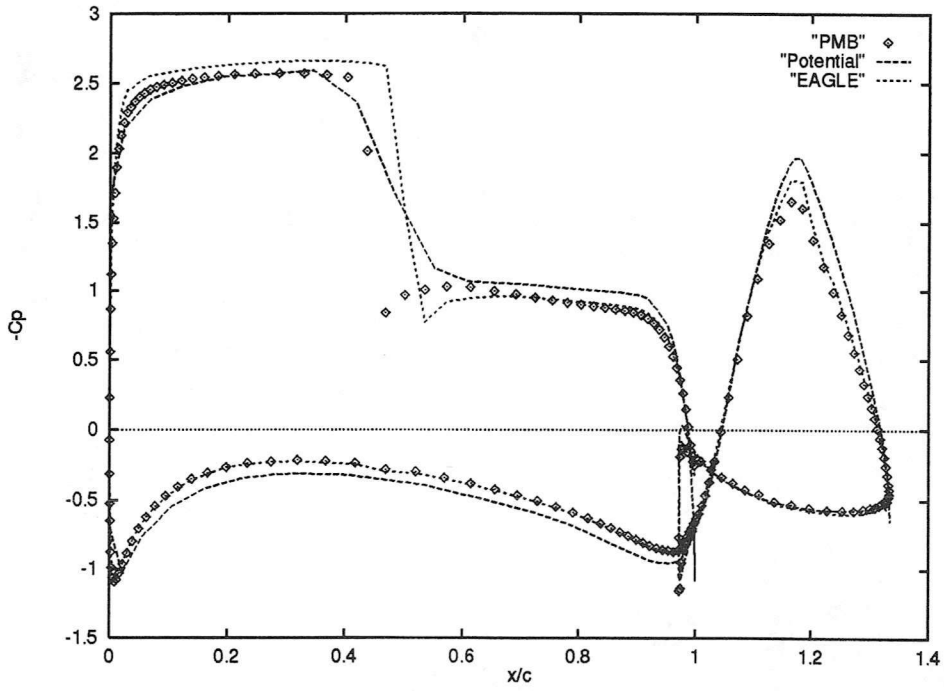


Figure 11: *Pressure coefficient for Willaims aerofoil, configuration B(M=0.58, α=0.0)*

flow solver gives identical results on each grid (the node distributions at block edges are identical), but the solution converges more quickly on the EAGLEView grid, Figure 13. The area of difficulty in the Geomesh grid for the Williams B case was in the slot region where it is difficult to maintain smoothness around the five-point singularity, Figure 14. The EAGLEView grid for the same case is noticably better, Figure 15, particularly regarding continuity of slope across the block boundaries emanating from the five-point singularity and orthogonality at the surface of the flap. Again the flowfield solutions obtained are identical for each grid, but the EAGLEView grid converges more quickly, Figure 16. The relative speed-up would probably be greater if a more appropriate, slightly coarser grid were used.

2.4 Discussion

There is no advantage to using EAGLE over EAGLEView. The real question is whether the superior GUI of Geomesh gives sufficient compensation for the poorer grids it produces compared to EAGLEView, and the other problems associated with Geomesh mentioned above. If a user has no experience of either package and does not have access to the data filters needed to extract Geomesh grids, EAGLEView should be the chosen tool because the time and effort needed to create the filters is prohibitive. Unfortunately a new user may not realise immediately that this problem exists because there is no mention of it in the manuals. If access to data filters for Geomesh is available, Geomesh would be the preferred tool where speed of construction of a new grid is at a premium. It is the author's opinion that Geomesh is a very good tool for generating reasonable grids quickly, particularly in 2-D, and is superior to EAGLEView in this respect. However the high quality of grids produced in EAGLEView is the overriding factor in making it the best tool in general. Finally, there is no choice if it is desired to create a grid with a singular line. It cannot be done in Geomesh, but is straightforward in EAGLEView.

3 Load balancing and mesh partitioning

To use most effectively a parallel platform an intelligent mapping of subsets of the total computational work onto processors must be performed. There is more than one possible objective: to minimise execution time for a given run, to maximise throughput (minimise execution time for all runs, weighted by job priorities) and also to minimise network traffic. The chosen objective (or mixture of objectives) may vary with circumstances eg. time of day for a given task. Ideally, an automatic mapping would be performed which partitions the computational domain taking account of the heterogeneous nature of the set of processing nodes and the communication network before run-time. This would be a 'static' partition. During run-time it is generally true that the number of calculations performed is not exactly predictable so a utility to create a new mapping 'dynamically' should be available.

In CFD, the problem is usually decoupled into two parts, mesh partitioning and allocation of the partition segments onto the parallel machine. Mesh partitioning is at its simplest for single-block structured grids, but is far more complex for unstructured grids due to the large number of possible permutations. As well as attempting to obtain equally sized segments, unstructured grid partitioning algorithms try to minimise the interface length i.e. the amount of communication between processors. Multiblock grid partitioning is easier in that there are

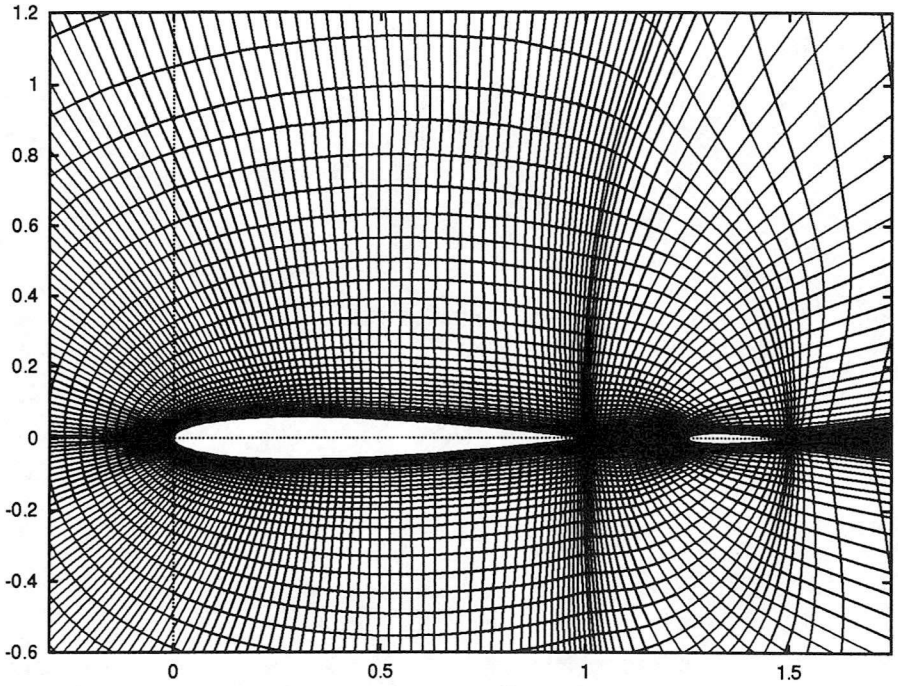


Figure 12: *EAGLEViewgrid for tandem aerofoils case*

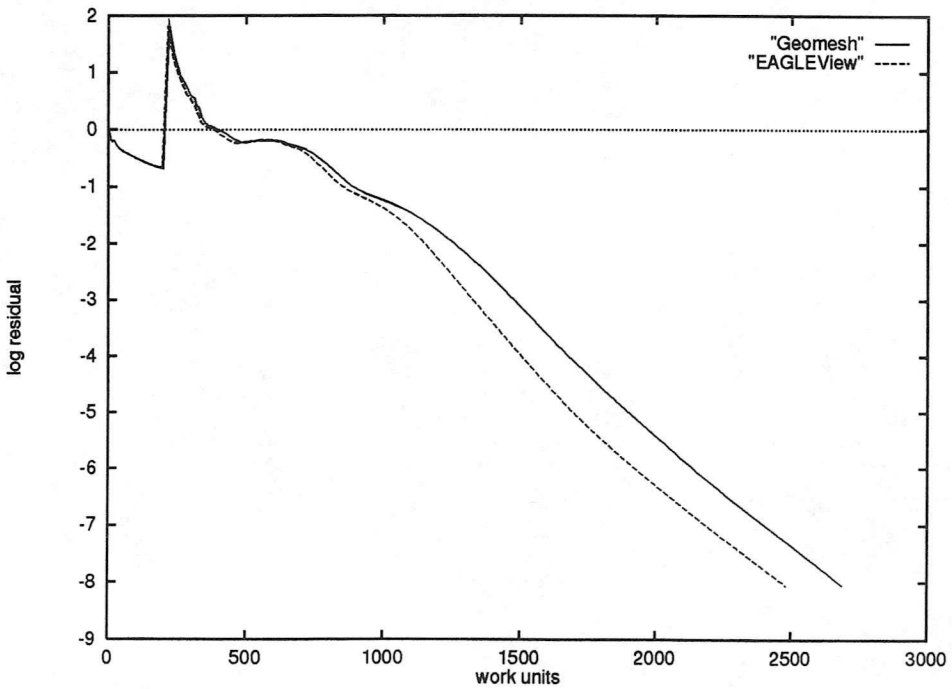


Figure 13: *Comparison of solution convergence rates using the EAGLEView and Geomesh grids, tandem aerofoils case*

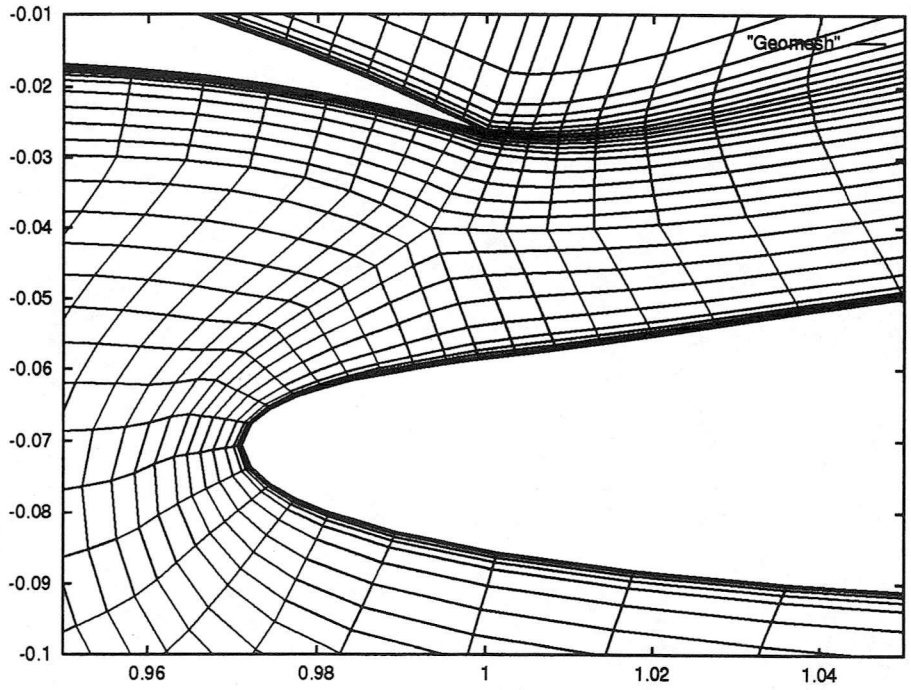


Figure 14: *Geomesh grid in slot region for Williams aerofoil, configuration B*

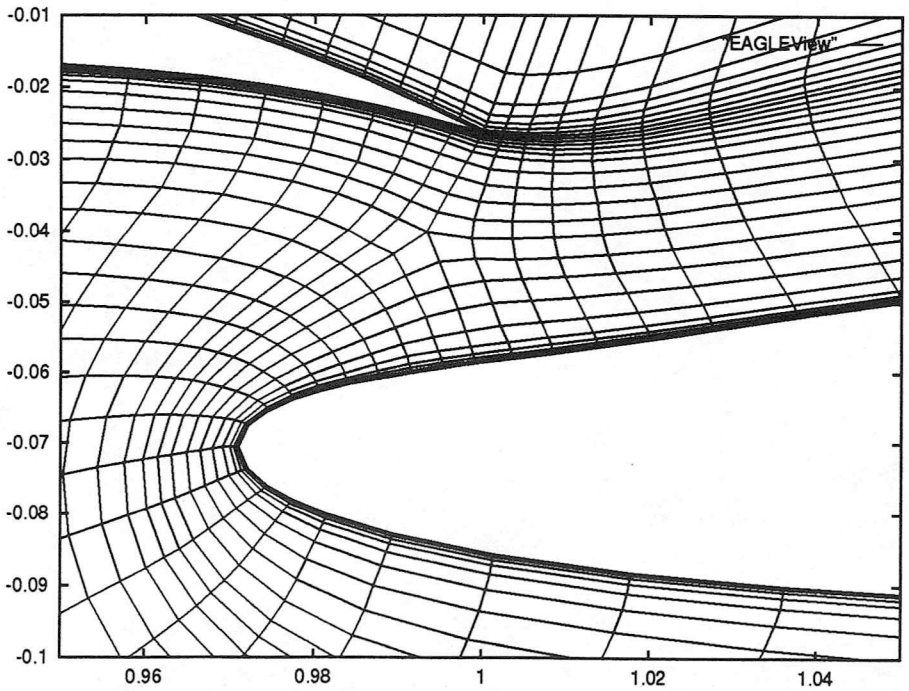


Figure 15: *EAGLEView grid in slot region for Williams aerofoil, configuration B*

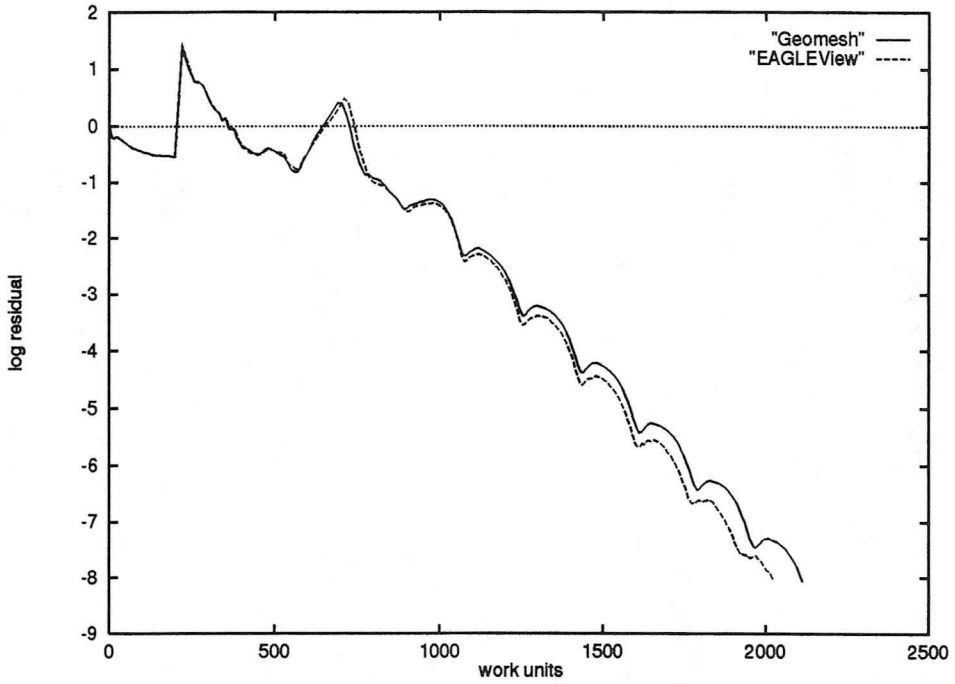


Figure 16: *Comparison of solution convergence rates using the EAGLEView and Geomesh grids, Williams aerofoil case*

less possible boundary path permutations but harder in that geometrical constraints must be considered in the partitioning algorithm. A complete model for mapping the partition onto the parallel machine should take into account variations in processor speeds, memory sizes and communication and create a mapping appropriate to the objective. The task of gathering this information especially in a dynamic multi-user environment is difficult, and simplifying assumptions are often made to make network modelling easier.

For unstructured grids, there are many examples of successful applications of recursive bisection heuristics to obtain load-balanced partitions, usually with some effort made to minimise communication in the process. Reference [22] provides a good overview of this approach, where [23] adds a simulated annealing procedure to minimise a computational cost function as a second step. Simulated annealing is one example of a non-deterministic method of finding a minimum to an objective function. The methods of stochastic evolution and tabu searching have also been used for unstructured partitioning [23] and genetic algorithms [24] for structured grids. Interestingly the relative importance of load balancing versus communication costs appears in all these examples as an artificial balance parameter in the cost function. Far more complex communication models are used for the problem of mapping existing partitions onto processors [25] [26].

3.1 Simulated Annealing

The method of simulated annealing [27],[28],[29] is a relatively new method for the minimization of objective functions. It is particularly suited to discrete, very large configuration spaces i.e. for combinatorial minimization problems. The title of the

method is due to an analogy with the slow cooling of metals.

The annealing algorithm is straightforward to program. Some initial configuration of the state (random will do) is required, along with a cost function definition. In an iterative manner, a small change based on a random selection is made to the system and this 'basic move' is either accepted or rejected. If the acceptance criterion is that the cost associated with the system decreases, and the rejection criterion is that the cost increases, then the algorithm is of the 'greedy' or 'hill-climbing' variety. This type of algorithm cannot avoid local minima. This is the kernel of the annealing algorithm, except that the acceptance/rejection criteria are different. If a proposed 'basic move' reduces the cost, then the move is accepted as before. If the cost is increased, then the move is only rejected with a certain probability, called the Metropolis criterion [30]. Included in this criterion is an artificial system 'temperature' such that at high temperatures almost any basic move however bad is accepted, and at low temperatures effectively zero bad moves are accepted i.e. the algorithm becomes of the 'greedy' type. A high starting temperature is used, and the temperature is periodically forced down by some factor after a large number of basic moves have been proposed. The intention is to explore the entire state-space with the Metropolis criterion providing a means of escape from local minima.

The attraction of simulated annealing is its simplicity, and that it is very easy to program the algorithm to provide reasonable solutions. The drawback is that it is difficult to determine other than through trial and error what values to use for the various parameters that appear in the algorithm (initial temperature, cooling schedule, how many basic moves to attempt at each temperature level) to optimise convergence to the global minimum.

A good introduction to the application of simulating annealing to a model load balancing problem is provided by Williams [22]. An attempt was made to replicate his results. The system represents a 1-D simplification of a general mesh partitioning problem. Two hundred equally sized data blocks with a simple communication interdependency have to be mapped onto four processors such that a cost function is minimised. The cost function contains a load balancing and a communication overhead term:

$$H = \frac{P^2}{N^2} \sum_q {N_q}^2 + \left(\frac{\mu P}{N}\right)^{\frac{2}{3}} \sum_{e \leftrightarrow f} 1 - \delta_{p(e),p(f)}$$

The first term is a minimum if the N data blocks are divided equally amongst the P processors. The second term is small if the number of communication boundaries is small, i.e. if the number of neighbouring data blocks on different processors is small. A value of 0.9 was chosen for the artificial balance coefficient μ. For this simple model problem, the cost H is minimised if groups of fifty data blocks are assigned to each processor.

The heuristic used for the 'basic move' is very important in simulated annealing. A good choice of basic move based on an understanding of the system being modelled can make a huge difference to the execution time, but care must be made not to introduce too complicated an expression which degrades the exploration of the state-space. The evolution of the cost of the present state as the temperature is reduced is shown for four different heuristics, Figure 17. The first heuristic (move a) is simply to allocate a random block onto a random processor. The best state found was very near the minimum but not the optimum. Some more insight was used for the second heuristic. It is known that the optimum state will consist

largely of groups of neighbouring blocks on each processor, so the second heuristic (move b) allocates a random block onto the processor of a randomly selected neighbouring block. This method found a local minimum very quickly. However the initial state did not include any blocks allocated to one of the four processors so attaining the optimum state was not possible. In this case the optimum distribution onto three processors is found. To avoid this problem, the third heuristic (move c) combines the first and second methods; the basic move at each step is the second heuristic with a large probability, but the first heuristic with a small probability to "seed" the allocation of blocks onto an empty processor. This attempt successfully finds the optimum state. The last heuristic (move d) is to use the third heuristic on a cluster of neighbouring blocks rather than on a single block. This attempt also finds the optimum state, and in a marginally quicker time. The results of the experiments agree well with those of Williams.

As a first step to developing a practical load balancing tool for the parallel multiblock flow solver, the method of simulated annealing was used to produce a load balanced allocation of blocks onto processors. The cost function used was that above without the second communication term for simplicity. No mesh partitioning was included in the algorithm, the blocks being those occuring from the grid generation process. The heuristic used was fairly complex. With low probability a random block is moved onto a random processor. With high probability a cluster of blocks on one processor was swapped with a cluster on another processor, the clusters being formed in a probability-based manner. The evolution of the cost as the temperature reduces is shown in Figure 18 for the Williams aerofoil grid. There are 15 blocks of widely varying size, the largest having 4408 points and the smallest having 150, which are to be mapped onto 3 processors. The best state found has a cost of fractionally above 3 units so it can be concluded that the annealing algorithm was applied successfully (a perfect mapping onto 3 processors would have a cost of 3.0, if the mapping were possible). The progress of the run appears somewhat haphazard compared to Figure 17. This is because the state-space for this problem is far more uneven, having a large number of steep-sided local minima, and as such poses a very difficult minimisation problem.

4 Future directions

At present the 2-D multiblock flow solver imposes the restriction on the block topology that each block can have only one boundary condition per face. This has two disadvantages. In 3-D this restriction would impose severe limitations on load balancing potential. Also, it is preferable to have a smaller number of larger blocks when considering the performance of the preconditioner in the flow solver. Multiple boundary conditions per face should for these reasons be included in the 3-D solver. For the 2-D solver this is less important, but inclusion of this feature would be useful to gain experience in the implications for calculation management and load balancing/mesh partitioning on less computationally demanding problems. The possibility of using the solver on grids including blocks with one face collapsed to a singular line must also be catered for if the goal of obtaining a flow solver for truly arbitrary 3-D configurations is to be achieved.

The next steps in the static load balancing problem are to add a mesh partitioner to the simulated annealing heuristics and to develop a communication cost model for inclusion in the cost function. It should be possible to base this on real-

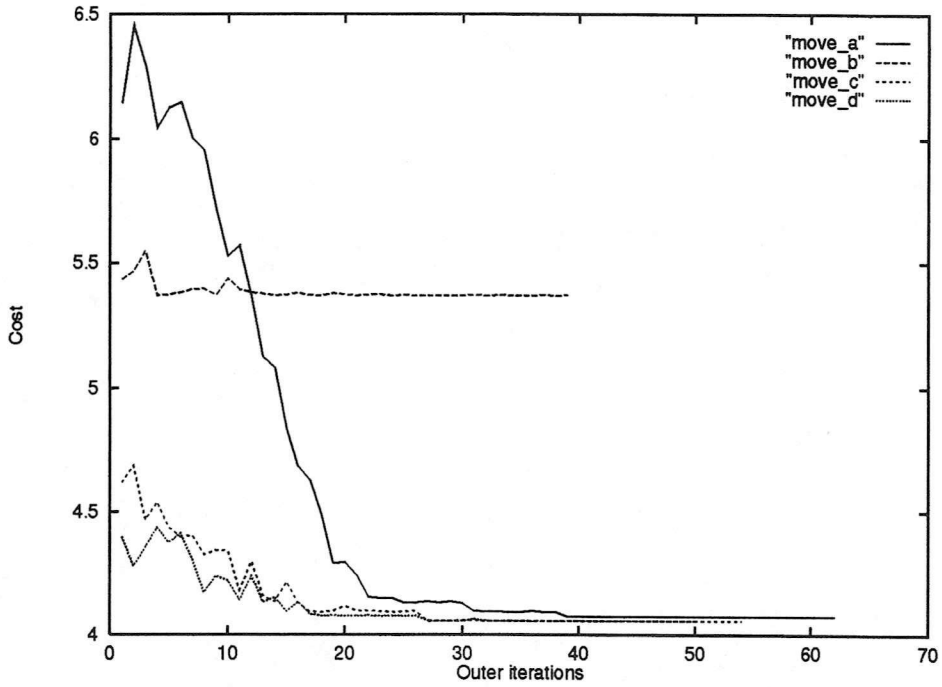


Figure 17: *Comparison of heuristics in simulated annealing for model problem*

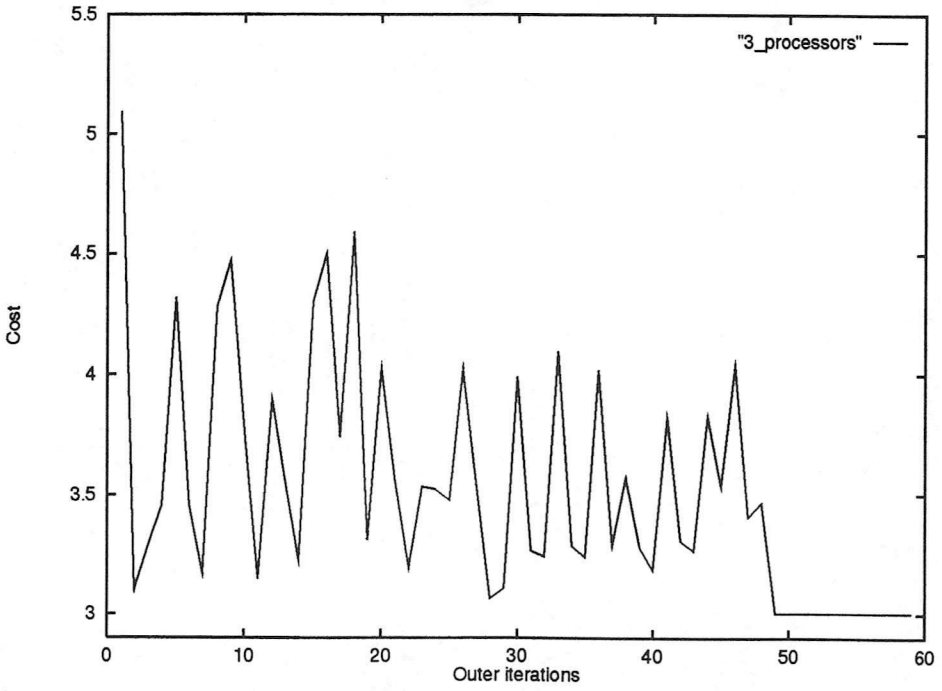


Figure 18: *Cost function evolution from simulated annealing for 2-D multiblock grid case*

ity (through network timing experiments or similar) rather than the artificial balance coefficient favoured in the literature. A static load balancer developed to this level of complexity would already be a very useful tool, but for optimal static partitioning the final step is to take into account network loading at the time of calculation. This is of particular interest for parallel machines consisting of a network of workstations, such as that at Glasgow, where there is a dynamic multi-user environment.

An interesting sideways step would be to decide for the user how many processors should be used, referring to the objective discussed at the beginning of Section 3. The studies of optimal partitioning appearing so far assume that the number of processors has been decided prior to the partition calculation. For this to work well, for every run the user must consider what the objective is, monitor the load on the network, and decide accordingly. It is unrealistic to expect the user to be capable of making the best choice on a large network, and preferable if the task is automated.

Solving the dynamic problem can be based on the method developed for the static problem, with the addition of a system for monitoring the development of load imbalance during run-time. During the parallel execution several copies of the flow solver are running at once. In some areas of the domain the solution will be converging more quickly than others. This is a load imbalance due to the flow solver, which can be monitored in the flow solver. However, the dynamic load imbalance caused by other users must be monitored using system software. A challenge is that the dynamic monitoring and re-allocating must be fast, which almost precludes the use of simulated annealing. It may be necessary to introduce some simplifications eg. assume all re-allocations will be small in magnitude or to adopt a more advanced stochastic approach for cost function minimisation eg. genetic algorithms.

There is also the possibility of using different types of flow solver in different sub-domains, eg. an inviscid solution may be adequate in an upstream far-field block. This would of course add complexity to the flow solver let alone the dynamic partitioner but the savings in computation time could be significant.

Acknowledgements

Many thanks to Ken Badcock, Franck Cantariti, Wark Woodgate, and Laurent Dubuc for their generous assistance and advice during this project. Thanks also to Trevor Birch at DRA Bedford for his support. This work is funded by a University of Glasgow Research Scholarship and sponsorship form DRA Bedford.

References

[1] N.P. Weatherill and C.R. Forsey, 'Grid Generation and Flow Calculations for Aircraft Geometries', *Journal of Aircraft, Vol. 22, No. 10, pp. 855-860,* (October 1985).

[2] A. Rizzi, P. Eliasson, I. Lindblad, C. Hirsch, H. Lacor and J. Haeuser, 'The Engineering of Multiblock/Multigrid Software for Navier-Stokes Flows on Structured Meshes', *Computers and Fluids, Vol. 22, Nos. 2-3, pp. 341-367,* (1992).

[3] J.F. Thompson and N.P. Weatherill, 'Aspects of Numerical Grid Generation: Current Science and Art', *AIAA Applied Aerodynamics Conference, Monterey, California,* (August 9-11, 1993).

[4] K.J. Badcock, S. Porter and B.E. Richards, 'Unfactored Multiblock

Methods - Part 1: Initial Development', *University of Glasgow, Aero Report 9511*, (1995).

[5] S. Rill, K. Becker, 'Simulation of Transonic Flow over Twin-Jet Transport Aircraft', *Journal of Aircraft, Vol. 29, No. 4, pp. 640-646*, (July-Aug, 1992).

[6] G. Freskos, O. Penanhoat, 'Numerical Simulation of the Flow Field Around Supersonic Air-Intakes', *Journal of Engineering for Gas Turbines and Power, Vol. 116, pp. 116-123*, (January 1994).

[7] P.A. Gnoffo, K.J. Weilmuenster, S.J. Alter, 'Multiblock Analysis for Shuttle Orbiter Re-entry Heating form Mach 24 to Mach 12', *Journal of Spacecraft and Rockets, Vol. 31, No. 3, pp. 367-377*, (May-Jun 1994).

[8] Th. Streit, 'Euler and Navier-Stokes Solutions for Supersonic Flow Around a Complex Missile', *Journal of Spacecraft and Rockets, Vol. 31, No. 4, pp. 600-608*, (July-Aug, 1994).

[9] M. Kathong, R.E. Smith, S.N. Timari, 'Application of Multiple Grids Topology to Supersonic Internal/External Flow Interactions', *Journal of Aircraft, Vol. 27, No. 3, pp. 245-252*, (March, 1990).

[10] G. Kalitzin, A.R.B. Gould, J.J. Benton, 'Application of Two-Equation Turbulence Models in Aircraft Design', *AIAA 96-0327, 34th Aerospace Sciences Meeting and Exhibit, Reno, NV*, (Jan 15-18, 1996).

[11] F. Ghaffari, J.M. Luckring, J.L. Thomas, B.L. Bates and R.T. Biedron, 'Multiblock Navier-Stokes Solutions bout the F/A-18 Wing-LEX-Fuselage Configuration', *Journal of Aircraft, Vol. 30, No. 3, pp. 293-303*, (May-June 1993).

[12] T.E. Nelson, D.W. Zingg and G.W. Johnston, 'Compressible Navier Stokes Computations of Multielement Airfoil Flows Using Multiblock Grids', *AIAA Journal, Vol. 32, No.3, pp. 506-511*, (March 1994).

[13] A. Ecer, J. Periaux, N. Satofuka and S. Taylor (eds.), 'Proceedings of the Parallel CFD 95 Conference', *Published in 1996 by Elsevier Science B.V.*, (1996).

[14] 'A User's Guide to PVM Parallel Virtual Machine', *Document ORNL/TM-12187, Oak Ridge National Laboratory*, (May 1993).

[15] Franck J-J. Cantariti, 'Computation of External Aerodynamic Flows using Differential Reynolds Stress modelling and Unstructured Grids', *PhD Thesis, UMIST*, (1988).

[16] P.I. Crumpton, J.A. Mackenzie and K.W. Morton, 'Cell Vertex Algorithms for the Compressible Navier-Stokes Equations', *Journal of Computational Physics Vol. 109, No. 1, pp. 1-15*, (1993).

[17] J.F. Thompson, 'Composite Grid Generation Code for Composite 3-D Regions — the EAGLE code', *AIAA Journal Vol. 26, No. 3, pp. 271-272*, (1988).

[18] B.R. Williams, 'An Exact Test Case for the Plane Potential Flow About Two Adjacent Lifting Aerofoils', *ARC R&M 3717*, (1973).

[19] A. Suddhoo and I.M. Hall, 'Inviscid Compressible Flow Past a Multi-

Element Aerofoil', *AGARD CP 365*, (1984).

[20] L. Stolcis and L.J. Johnston, 'Solution of the Euler Equations on Unstructured Grids for Two-Dimensional Compressible Flow', *Aeronautical Journal June/July 1990, pp. 181-195*, (1990).

[21] P.M. Sinclair, 'An Exact Integral (Field Panel) Method for the Calculation of Two-Dimensional Transonic Potential Flow Around Complex Configurations', *Aeronautical Journal June/July 1986, pp. 227-236*, (1986).

[22] R.D. Williams, 'Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations', *Concurrency: Practice and Experience Vol. 3(5), pp. 457-481*, (October 1991).

[23] D. Vanderstraeten and R. Keunings, 'Optimized Partitioning of Unstructured Finite Element Meshes', *International Journal for Numerical Methods in Engineering Vol. 38, pp. 433-450*, (1995).

[24] D. Quagliarella, 'Genetic Algorithms Applications in Computational Fluid Dynamics', *Genetic Algorithms in Engineering and Computer Science (John Wiley & Sons Ltd.), pp. 417-442*, (December 1995).

[25] Y.P. Chien, A. Ecer, H.U. Akay, F. Carpenter and R.A. Blech, 'Dynamic Load Balancing on a Network of Workstations for Solving Computational Fluid Dynamics Problems', *Comput. Methods Appl. Mech. Engrg. Vol. 119, pp. 17-33*, (1994).

[26] Y.P. Chien, F. Carpenter A. Ecer and H.U. Akay, 'Load-Balancing for Parallel Computation of Fluid Dynamics Problems', *Comput. Methods Appl. Mech. Engrg. Vol. 120, pp. 119-130*, (1995).

[27] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery, 'Numerical Recipes in C: The Art of Scientific Computing', *Cambridge University Press*, (1992).

[28] R.A. Rutenbar, 'Simulated Annealing Algorithms: An Overview', *IEEE Circuits and Devices Magazine, Jan. 1989, pp. 19-26*, (January 1989).

[29] R.H.J.M. Otten and L.P.P.P. van Ginneken, 'The Annealing Algorithm', *Kluwer Academic Publishers*, (1989).

[30] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller 'Equation of State Calculations by Fast Computing Machines', *Journal of Chemical Physics Vol. 21, pp. 1087-1092*, (1953).