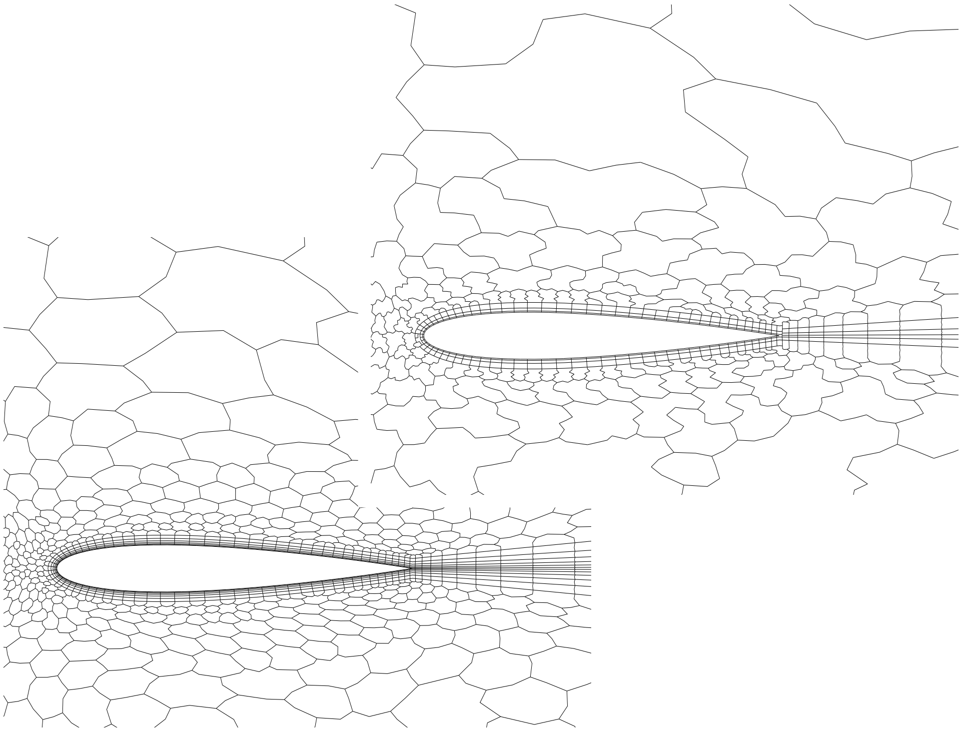
AIAA 99-3338

Development of an Agglomeration Multigrid Algorithm with Directional Coarsening

J. V. Lassaline and D. W. Zingg
Institute for Aerospace Studies
University of Toronto
Toronto, Ontario, Canada



14th AIAA Computational Fluid Dynamics Conference

June 28-July 1, 1999/Norfolk, VA

Development of an Agglomeration Multigrid Algorithm with Directional Coarsening

J. V. Lassaline* and D. W. Zingg†
Institute for Aerospace Studies
University of Toronto
Toronto, Ontario, Canada

A two-dimensional agglomeration multigrid solver is presented for the computation of steady turbulent flows on unstructured grids. The solver utilizes a finite-volume method to solve the Favre-averaged compressible Navier-Stokes equations. Utilizing a variety of source grids, an agglomeration process is used to automatically generate coarse grids suitable for use with a multigrid solver. Variations of the agglomeration algorithm including directional coarsening are presented. Programming issues with respect to the solver are discussed. Results are presented demonstrating the accuracy and efficiency of the solver, and the effect of directional coarsening on multigrid convergence rates.

Introduction

THE goal of this research is the development and implementation of a two-dimensional turbulent flow solver that provides fast convergence rates for a wide variety of flow conditions. A secondary goal of the research is to develop a solver that automates as much of the solution process as possible.

Multigrid methods are often used to accelerate the convergence of explicit iterative methods. Typically grid generation can be a significant task requiring user intervention. Generating suitable coarse grids for a multigrid sequence can add to the difficulty. A coarse grid can be formed from a structured grid by removing alternate rows of points but cannot be as easily generalized for unstructured grids. One approach is to generate multiple non-nested meshes using the same grid generator as used to create the finest grid.[1,2] Our solver utilizes agglomeration to heuristically generate a suitable sequence of coarse grids for use with the multigrid method.[3] Agglomeration fuses groups of fine grid cells to build a nested coarse grid. The algorithm may be applied to structured, unstructured and hybrid grids; however the resulting coarse grids are unstructured.

Our aim is to utilize hybrid (or mixed) grids as this class of grids has been demonstrated to provide excellent results for turbulent flows.[4,5] As the agglomeration algorithm is applied to the dual of the source grid, the algorithm works on any edge-connected collection of points. The agglomeration algorithm requires the development of generalized finite-volume discretizations on polygonal cells as encountered with hybrid grids.

Multigrid has been shown in the past to work well for inviscid flows but can converge an order of magnitude slower for viscous flows. Morano et al.[6] have demonstrated that the convergence properties of the multigrid method depend largely upon the "quality" of the grid. Agglomeration in its simplest form provides grids that are suitable for inviscid flows, but suffer from stiffness in regions of highly stretched cells as used to resolve high Reynolds number turbulent flows. Mavriplis[7] has demonstrated a variation of agglomeration that couples directional-coarsening and line-implicit smoothing resulting in an improved viscous convergence rate. It is the aim of this paper to demonstrate an agglomeration method that encompasses the best features of an automatic grid generator.

In addition, several programming issues are discussed with respect to the development of this solver. An object-oriented language was used in the development and the advantages to this approach are discussed. The modular implementation of the solver is discussed, and how it improves the design and flexibility of the solver.

Governing Equations And Turbulence Model

The two-dimensional Favre-averaged Navier-Stokes equations are solved with turbulence effects modeled by the Spalart-Allmaras turbulence model. The spatial discretization is accomplished using a finite-volume method. Iteration to convergence is achieved using a Runge-Kutta time-marching method accelerated using multigrid.

In two-dimensional integral form, the equations can be written as

$$\frac{d}{dt}\int_{\Omega}\mathbf{Q}dA + \oint_{d\Omega}\mathbf{n}\cdot\left(\overline{\mathbf{F}}-\overline{\mathbf{G}}\right)ds = \int_{\Omega}\mathbf{S}dA \qquad (1)$$

where $\mathbf{Q}$ represents the vector of conserved variables, $\overline{\mathbf{F}}$ is the inviscid flux tensor, $\overline{\mathbf{G}}$ is the viscous flux ten-

*Graduate Student, jason@oddjob.utias.utoronto.ca.
†Professor, Member AIAA.

sor, and $\mathbf{S}$ contains the source terms arising from the turbulence model.

The one-equation Spalart and Allmaras[8] turbulence model is written in non-dimensional form as

$$\begin{aligned}\frac{\partial\tilde{\nu}}{\partial t}+u\frac{\partial\tilde{\nu}}{\partial x}+v\frac{\partial\tilde{\nu}}{\partial y}&=\frac{c_{b1}}{Re}\left[1-f_{t2}\right]\tilde{S}\tilde{\nu}\\&+\frac{1}{Re\sigma}\left[\nabla\cdot\left(\left(\nu+\tilde{\nu}\right)\nabla\tilde{\nu}\right)+c_{b2}\nabla^2\tilde{\nu}\right]\\&-\frac{1}{Re}\left[c_{w1}f_w-\frac{c_{b1}}{\kappa^2}f_{t2}\right]\left[\frac{\tilde{\nu}}{d}\right]^2+Ref_{t1}\Delta U^2\end{aligned}\quad(2)$$

where the kinematic eddy viscosity ν_t is determined from the turbulence parameter $\tilde{\nu}$ using

$$\nu_t=\tilde{\nu}f_{v1}\quad(3)$$

$$f_{v1}=\frac{\chi^3}{\chi^3+c_{vi}^3}\quad(4)$$

$$\chi=\frac{\tilde{\nu}}{\nu}\quad(5)$$

where ν is the laminar kinematic viscosity.

The turbulence production, destruction and trip terms are computed using the following relations

$$S=\left|\frac{\partial v}{\partial x}-\frac{\partial u}{\partial y}\right|\quad(6)$$

$$f_{v2}=1-\frac{\chi}{1+\chi f_{v1}}\quad(7)$$

$$f_w=g\left(\frac{1+C_{w3}^6}{g^6+C_{w3}^6}\right)^{\frac{1}{6}}\quad(8)$$

$$g=r+C_{w2}(r^6-r)\quad(9)$$

$$r=\frac{\tilde{\nu}}{\tilde{S}\kappa^2d^2}\quad(10)$$

$$\tilde{S}=SRe+\frac{\tilde{\nu}f_{v2}}{\kappa^2d^2}\quad(11)$$

$$f_{t2}=C_{t3}e^{-C_{t4}\chi^2}\quad(12)$$

where d is the distance from the point in question to the closest boundary wall.

The functions f_{t1} and g_t of the trip term are calculated as

$$f_{t1}=C_{t1}g_te^{-C_{t2}\frac{\omega_t^2}{\Delta U^2(d^2+g_t^2d_t^2)}}\quad(13)$$

$$g_t=\min\left(0.1,\frac{\Delta U}{\omega_t\Delta x}\right)\quad(14)$$

where d_t is the distance to the closest trip point and ω_t is the vorticity at the closest trip point.

The following constants are used

$$\begin{array}{ccc}C_{b1}=0.1355 & \sigma=2/3 & C_{b2}=0.622\\ \kappa=0.41 & C_{t1}=1 & C_{t2}=2\\ C_{t3}=1.1 & C_{t4}=2 & C_{w2}=0.3\\ C_{w3}=2 & C_{v1}=7.1 & \end{array}\quad(15)$$

$$C_{w1}=\frac{C_{b1}}{\kappa^2}+\frac{1}{\sigma}(1+C_{b2})$$

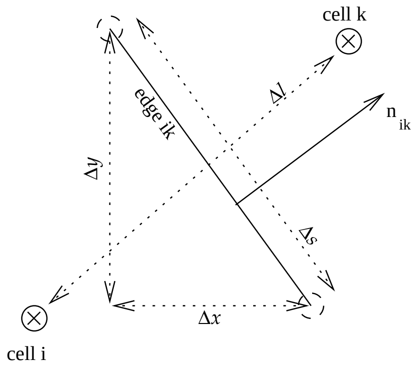


Fig. 1 An example edge between cell i and neighbour k indicating pre-computed dimensions.

Spatial Discretization

Assuming a piecewise constant solution vector across the cell, the semi-discrete form can be written for cell i as

$$\Omega_i\frac{d}{dt}\mathbf{Q}_i=\sum_k^{neigh.}\left(\overline{\mathbf{F}}_{ki}-\overline{\mathbf{G}}_{ki}\right)\cdot\mathbf{n}_{ki}+\Omega_i\mathbf{S}_i\quad(16)$$

where Ω_i is the cell area and $\mathbf{n}_{ki}=\begin{bmatrix}\Delta y_{ki} & -\Delta x_{ki}\end{bmatrix}$ is the edge-scaled outward normal between neighbouring cells. An example edge between cells i and k indicating the edge dimensions appears in Fig. 1.

The calculation of viscous terms in the Navier-Stokes equations requires the computation of the spatial second derivatives of the flow variables. A Taylor expansion reveals the necessity of at least six points to approximate the second derivatives to first order. In a regular triangulated grid there are typically at least five neighbours to a point, and thus there is a sufficient number of points to form a stencil. However in conjunction with a more generic agglomerated grid, the number of neighbours may be less than required.[9] Thus to resolve the full viscous fluxes, it is necessary to involve the next-to-nearest neighbours in the computation of the spatial derivatives, adding computational cost to the solver.

In forming the full viscous fluxes, the terms of Eq. 16 are computed as

$$\mathbf{Q}=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ e\\ \rho\tilde{\nu}\end{bmatrix}\quad\mathbf{S}=\begin{bmatrix}0\\0\\0\\0\\S_t\end{bmatrix}$$

for the solution and source vectors, and the fluxes are computed as

$$\left(\overline{\mathbf{F}} - \overline{\mathbf{G}}\right) \cdot \mathbf{n} =$$

$$\begin{bmatrix} \rho u \\ \rho u^2 + p - \frac{1}{Re}\tau_{xx} \\ \rho uv - \frac{1}{Re}\tau_{xy} \\ u(e+p) - \frac{1}{Re} f \\ \rho u\tilde{\nu} - \frac{1}{Re\sigma}\rho(\nu + \tilde{\nu})\tilde{\nu}_x \end{bmatrix} n_x + \begin{bmatrix} \rho v \\ \rho uv - \frac{1}{Re}\tau_{xy} \\ \rho v^2 + p - \frac{1}{Re}\tau_{yy} \\ v(e+p) - \frac{1}{Re} g \\ \rho v\tilde{\nu} - \frac{1}{Re\sigma}\rho(\nu + \tilde{\nu})\tilde{\nu}_y \end{bmatrix} n_y \quad (17)$$

$$f = u\tau_{xx} + v\tau_{xy} + \frac{1}{\gamma - 1}\left(\frac{\mu}{Pr} + \frac{\mu_t}{Pr_t}\right)\frac{\partial a^2}{\partial x} \quad (18)$$

$$g = u\tau_{xy} + v\tau_{yy} + \frac{1}{\gamma - 1}\left(\frac{\mu}{Pr} + \frac{\mu_t}{Pr_t}\right)\frac{\partial a^2}{\partial y} \quad (19)$$

A number of the terms in Eq. 2 may be condensed into one source term, and in the absence of any additional source terms, the term $\mathbf{S}$ is greatly simplified. The solver assumes a piecewise constant value of the flow variables and their derivatives across the cell.

First derivatives may be computed by a sum about all the neighbours of a cell as follows for a general variable ϕ

$$\nabla\phi = \frac{1}{\Omega}\sum_{k}^{neigh.}\left[\phi_{ik}\Delta y_{ki}\mathbf{i} - \phi_{ik}\Delta x_{ki}\mathbf{j}\right] \quad (20)$$

where $\Delta x_{ki}, \Delta y_{ki}$ are the edge dimensions and ϕ_{ik} is the average value of ϕ for edge ik.

An additional matrix dissipation term formed from the undivided Laplacian and biharmonic is included in the residual for numerical stability and resolving shocks.[10] The dissipation vector is formed as the edge-based sum

$$\sum_{k}^{neigh.} |\mathbf{A}|_{ik}(e_{ik}^{(2)}(\mathbf{Q}_k - \mathbf{Q}_i) - e_{ik}^{(4)}(\mathbf{L}_k - \mathbf{L}_i)) \quad (21)$$

$$\mathbf{L}_i = \sum_{k}^{neigh.}(\mathbf{Q}_k - \mathbf{Q}_i) \quad (22)$$

This formulation approximates the undivided Laplacian operator as $\sum(\mathbf{Q}_k - \mathbf{Q}_i)$. The matrix $|\mathbf{A}|$ is formed as

$$|\mathbf{A}| = \mathbf{X}|\Lambda|\mathbf{X}^{-1} \quad (23)$$

where $\mathbf{X}$ is the matrix formed from the eigenvectors of the flux jacobian normal to the edge, and $|\Lambda|$ is a diagonal matrix containing the absolute values of the eigenvalues. The matrix $|\mathbf{A}|_{ik}$ is the average computed for the left and right cell.

The numerical boundary conditions are applied to all boundary edges in a manner consistent with the physical boundary conditions. For outflow boundary edges Riemann invarients are used to prevent the reflection of outgoing waves. For solid wall boundary conditions a no-slip, adiabatic wall condition is applied. For all boundary edges, a solution vector is retained to be used in computing the residual at the first interior cells. Values at the boundary are extrapolated from the interior cells.[11]

Iterative Method

The solution is time-marched to convergence using a 5-stage Runge-Kutta method. Generically this may be written for an m stage method as

$$\mathbf{Q}_0 = \mathbf{Q}^n \quad (24)$$

$$\cdots$$

$$\mathbf{Q}_i = \mathbf{Q}_0 - \frac{\alpha_i}{\Omega}\mathrm{CFL}\left[\Delta \mathsf{t}\right]\mathbf{R}_i \quad (25)$$

$$\cdots$$

$$\mathbf{Q}^{n+1} = \mathbf{Q}_m \quad (26)$$

where the residual $\mathbf{R}_i$ is computed from the convective and dissipative terms of Eq. 16 and $[\Delta \mathsf{t}]$ is the matrix time-step. The matrix time-step can be written for cell i as

$$[\Delta \mathsf{t}]_i^{-1} = [\Delta \mathsf{t_H}]_i^{-1} + [\Delta \mathsf{t_P}]_i^{-1} \quad (27)$$

$$[\Delta \mathsf{t_H}]_i^{-1} = \frac{1}{2\Omega_i}\sum_{k}^{neigh.}|\mathbf{A}_{ik}| \quad (28)$$

$$[\Delta \mathsf{t_P}]_i^{-1} = \frac{2}{\Omega_i^2}\sum_{k}^{neigh.}\lambda_{ik}\mathbf{I} \quad (29)$$

$$\lambda = \left(\frac{\mu}{Pr} + \frac{\mu_t}{Pr_t}\right)\frac{\Delta l\gamma}{\rho Re} \quad (30)$$

where $|\mathbf{A}_{ik}|$ and λ_{ik} use the average of the neighbouring solution vectors. In the case of our solver, $\alpha = \begin{bmatrix}0.25 & \frac{1}{6} & \frac{3}{8} & 0.5 & 1.0\end{bmatrix}$. To lower the computational cost of a residual calculation, dissipation is only calculated for the first, third and fifth stage while viscous fluxes are only calculated for the first stage.[11]

In the case of a single grid, the time-march is repeated until convergence was obtained. To accelerate the convergence, a multigrid cycle is used where the computed correction is obtained by time-marching a sum of the fine grid equations with the addition of a forcing term on the coarse grids. The multigrid cycle takes advantage of the fact that low-frequency errors on the fine grid become high-frequency errors on the coarse grid. By using a time-marching method tailored to high-frequency errors on all grids, convergence of the fine grid solution is accelerated as compared to time-marching on the fine grid alone. An example four-grid multigrid cycle is presented in Fig. 2. A W-cycle performs more time-steps on the coarse grids and

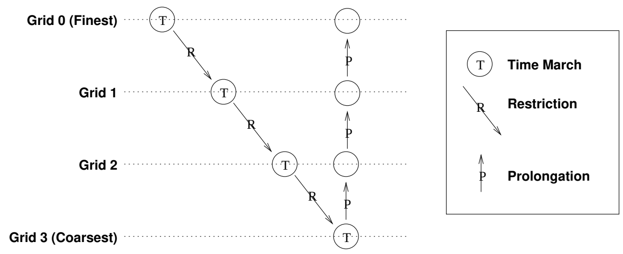


Fig. 2 An example four grid multigrid sawtooth cycle.

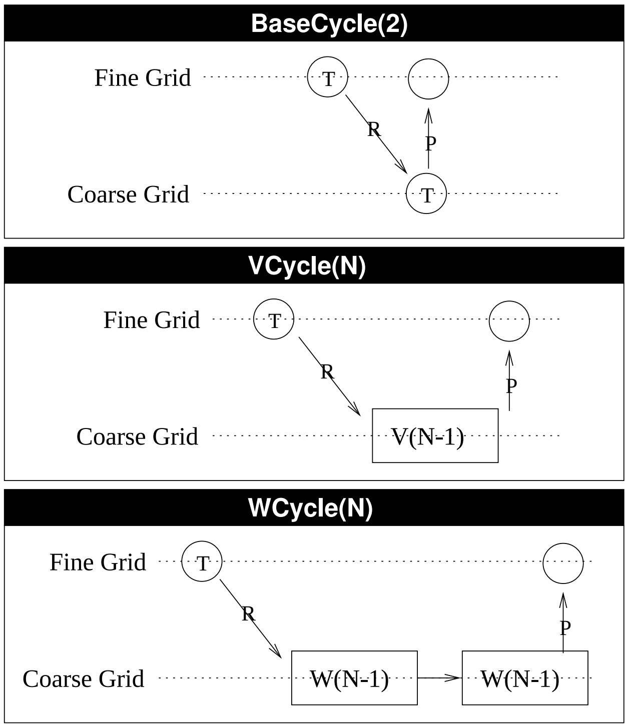


Fig. 3 A recursive cycle definition.

will typically further improve convergence rates. Recursive definitions of a V-cycle and a W-cycle appear in Fig. 3.

With nested coarse grids, the coarse grid equations are formed as a sum of their fine grid constituents with an additional forcing function. On a coarse grid, Eq. 25 of the time-marching method is re-written to include a component of the residual of all finer grids.

$$\mathbf{Q}_i = \mathbf{Q}_0 - \frac{\alpha_i}{\Omega}\left[\Delta \mathrm{t}\right]\left(\mathbf{R}_i + \mathbf{P}\right) \tag{31}$$

where the forcing function $\mathbf{P}$ is formed from the restricted residual $I_f^c\mathbf{R}$ and the zeroth stage coarse grid residual. The new coarse grid residual is represented by $\mathbf{R}_i + \mathbf{P}$.

The restriction operator restricts the current fine grid solution $\mathbf{Q}$ and residual $\mathbf{R}$ to form the initial coarse-grid solution and the forcing function $\mathbf{P}$. On the coarse grid this is defined as

$$\mathbf{Q}_c^n = I_{f_Q}^c \mathbf{Q}_f^n \tag{32}$$

$$\mathbf{P}_c = I_{f_R}^c \mathbf{R}_f - \mathbf{R}\left(I_{f_Q}^c \mathbf{Q}_f\right) \tag{33}$$

The prolongation operator transfers the computed corrections from the coarse to the finer grid using injection. The prolongation operator can be defined by

$$\mathbf{Q}_f^{n+1} = \mathbf{Q}_f^n + I_c^f\left(\mathbf{Q}_c^{n+1} - I_{f_Q}^c \mathbf{Q}_f^n\right) \tag{34}$$

The interpolation operators used for restriction of the solution and residual are defined using the nested nature of the agglomerated cells.

$$I_{f_Q}^c \mathbf{Q}_c = \frac{1}{\Omega_c}\sum_i^{fine} \Omega_{fi}\mathbf{Q}_{fi} \tag{35}$$

$$I_{f_R}^c \mathbf{R}_c = \sum_i^{fine} \mathbf{R}_{fi} \tag{36}$$

The interpolation operator used for the prolongation simply passes the coarse-grid solution correction $\mathbf{\Delta Q}$ to the constituent fine grid cells. Point-Jacobi smoothing can be applied to the fine grid correction but is not typically required.

Agglomeration

The agglomeration process is a heuristic technique for the automatic generation of a coarse grid sequence suitable for use with a multigrid solver. For a given source grid a centroidal-median dual is generated. By selectively fusing fine grid cells, a nested coarse grid may be generated from the dual. This process may be repeated recursively to generate a sequence of successively coarser grids.

The current algorithm can handle two-dimensional grids from a variety of sources provided the following descriptions of the grid are defined or may be inferred:

A list of points defined by co-ordinates,

A list of cells defined by bounding edges,

A list of edges defined by end-points and neighbouring cells.

This definition of a grid may be used to represent compactly either the source grid, the resulting dual, or any of the coarse grids. Thus the agglomeration algorithm may be applied to any grid stored in this format.

The agglomeration algorithm proceeds by fusing available fine grid cells according to specified criteria. It is presented in pseudo-code Algorithm 1. If no selection criterion is specified, the resulting algorithm coarsens in a greedy, wave-front manner. This may produce less than desirable results, with small cells being surrounded by a few large neighbours. However,

Algorithm 1 Agglomeration method

Form a priority queue of cells (weighting boundary cells higher than non-boundary cells.)
while priority queue is not empty **do**
  Pop a seed cell from the priority queue.
  Form new coarse cell from seed cell.
  for all neighbouring cells of seed **do**
    if neighbour matches criteria and not already fused **then**
      Add the neighbour cell to coarse cell
    end if
  end for
end while

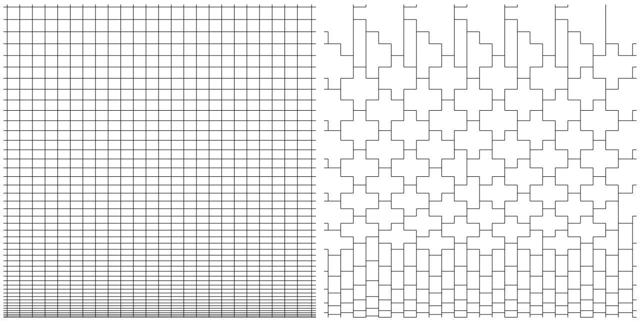
it does provide an optimal coarsening ratio of approximately 4 : 1.[12]

Venkatakrishnan and Mavriplis[12] suggested a variation in which the aspect ratio of the coarse cells is minimized. The resulting algorithm adds the additional fusing criterion that the new coarse cell must minimize P^2/Ω where P is the perimeter and Ω is the area of the new cell. This algorithm is slightly more expensive then the original. Although it produces a more uniform cell shape, it has the disadvantage that the resulting coarsening ratio is approximately 2:1. This can be overcome by repeating the agglomeration twice, and dropping the intermediate grid, thus recovering a 4:1 coarsening ratio.

Mavriplis[13] has demonstrated an agglomeration variation termed directional-coarsening that coarsens in a direction normal to the local cell stretching. The fuse selection criterion in this case is formed using a pre-computed weighting for each neighbour. The weighting is such that large weights are associated with neighbours in the direction normal to stretching and small weights in the direction parallel to the stretching direction. The agglomeration algorithm is modified such that only neighbours with weights greater than βw_{max} are fused; w_{max} representing the maximum edge weight for the seed cell. For $\beta = 0.5$ the algorithm behaves like the greedy wave-front method in isotropic regions, and coarsens directionally in regions of highly stretched cells. Directional coarsening results in a 2:1 coarsening ratio which may be overcome by agglomerating twice. The value $\beta = 0$ corresponds to isotropic coarsening.

An example of directional coarsening as applied to a structured grid varying from highly stretched cells at the bottom to isotropic cells at the top is presented in Fig. 4. In this case the agglomeration is applied twice to generate the coarser grid. Note that the cells near the bottom are coarsened in the direction normal to the cell stretching while the cells near the top are fused in a more isotropic manner.

In combination with the directional coarsening, a line construction algorithm is used to form lines along which an implicit smoother is applied. Constructing the line using the same weights as the directional coarsener results in lines formed perpendicular to the boundary layer and wake. The line construction algorithm begins by ordering cells according to the precomputed ratio of maximum to average edge weights. Beginning with the highest stretched cells, the algorithm constructs a line to a neighbouring cell if the maximum to average ratio of the neighbouring cell is greater than α (typically $\alpha = 4.0$.) For cells in regions of high stretching, a line-implicit smoothing may be applied, and for isotropic regions, the smoothing reduces to a point-Jacobi pre-conditioned scheme.



a) Source grid **b)** Agglomerated Grid

Fig. 4 An example of directional agglomeration applied to a region varying from highly stretched cells to isotropic cells.

Programming Issues

Our solver was implemented using an object-oriented programming language, in particular the `C++` language. Object-oriented languages may be easily applied to numerical solvers, and provide several key features we believe to be beneficial to the development of a reliable solver. Object classes provide encapsulation of data and their corresponding functions, forcing a design to be consistent in the handling of data.

Operator-overloading is a powerful feature that greatly simplifies code without sacrificing efficiency. Given a defined vector or matrix class, operations can be performed using the primitive operators $+, -, *, /, =$ directly on vectors or matrices. For example, instead of writing code to index the components of an expression, the expressions

$$\mathtt{a} = \mathtt{b} * 0.5 \tag{37}$$

$$\mathtt{x} = \mathtt{a}/\mathtt{A} \tag{38}$$

where `a`, `b` , and `x` are vectors and `A` is a matrix may be used directly in the code. Several public-license libraries are available which provide a wide range of numerical operations for a variety of vector and matrix classes.

Inheritance is another useful feature of object-oriented languages. Deriving a new class definition from another (inheritance) provides common functionality amongst all classes derived from the same base class. Our solver has implemented several features through inheritance such as

- persistence of objects beyond run-time,
- configuration of objects at run-time, and
- visualization for a variety of visualizer formats

that can be added to new classes simply by deriving them from the appropriate base class.

Data structures for our solver are implemented using the Standard Template Library (STL) freely provided by SGI Inc.[14] This template library allows us to develop code quickly using STL defined data structures such as lists and maps. One disadvantage to utilizing templates is that without careful design and compilation techniques, the resulting executable can quickly become quite large.

Our solver also makes use of a conceptual class called methods. Methods are similar to functions or procedures in that they execute some task. However, methods are created at run-time, and can be added, deleted or changed at run-time. Four basic method classes are defined

simple method performs one task,

serial method contains a list of methods, where the order of execution is important.

parallel method contains a list of methods, where the order of execution is not important.

iterative method contains three tasks (initializer, iterator, and finalizer) plus a completion criterion.

The last three methods may be defined using any other method.

Each method provides the following basic functionality

Constructor establishes parameters of tasks, and allocates memory for method

Execution completes tasks as applied to parameters.

Destructor cleans up memory allocation and method.

The constructor of the method is given all necessary parameters for the task, and to save memory these are typically passed as a reference. The execution may be called repeatedly to complete the defined task, and handles synchronizing steps of a task. The destructor performs any necessary clean-up for the method, such as memory deallocation.

Although at first glance, it may appear excessively complex, we feel that this technique holds many advantages. One major advantage of this technique is that it forces the designer of the solver to break the solver into its constituent tasks. A detailed analysis of the tasks within the solver will tend to lead towards a more reliable and well documented solver. Synchronization of tasks in a solver is required for the design of both serial and parallel processor solvers. The synchronization of steps of each method achieves this automatically. Breaking a solver into its constituent tasks also provides fine granularity for parallelization.

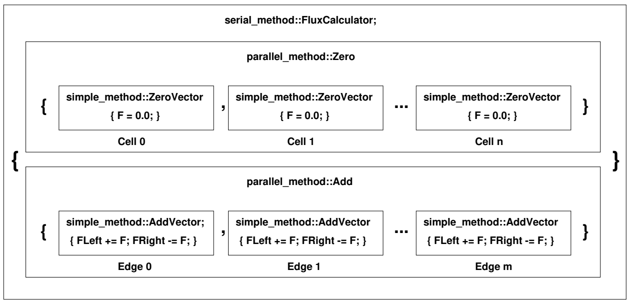


Fig. 5 An example flux calculator implemented using the conceptual class 'method'.

Passing parameters by reference means that variables appear as local variables to the execution section of the method. Thus the designer can write the execution section of the method as if the parameters were local.

As an example, consider the computation of flux vectors in forming the residual of this solver. A simplified version of tasks consists of setting all flux vectors to zero (one for every time-marched cell) and computing an edge flux (one for every edge.) An example of a fine granularity implementation of this method is presented in Fig. 5. As presented, the serial method 'FluxCalculator' consists of two steps: zero all vectors and add edge contributions.

The parallel method 'Zero' consists of a set of simple methods that set vector `F` to zero. Note that the method 'ZeroVector' makes use of the fact that the assignment operator is defined for the vector class. Hence there is no need for the 'ZeroVector' class to set all indices of `F`. In this example, one 'ZeroVector' method object is instantiated for every cell, yielding the finest granularity of the task. The n parallel method tasks could be split amongst several processors or completed on one processor.

Although storing a reference costs a small amount of memory, less memory could have been used at the cost of granularity. For example, if only 4 processors were to be used, four 'ZeroVector' method objects could be instantiated, each one operating on a larger vector `F` that represents essentially one-quarter of the number of cells. The memory cost per method object remains the same, but the granularity of the task is much coarser.

In computing the edge flux contributions, the flux vector would be computed for each edge and added to the left and right neighbouring cell. In this example method, 'AddVector' computes a vector `F` which

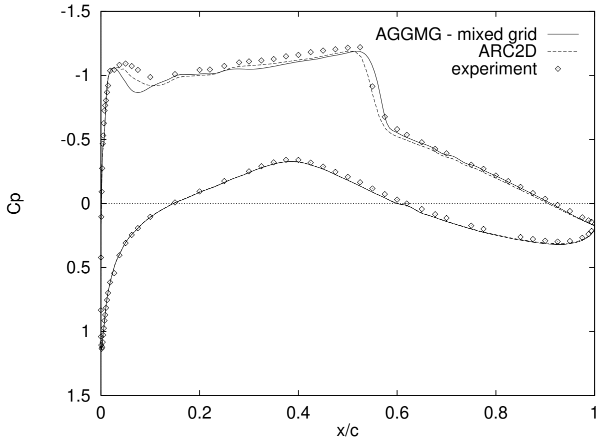


Fig. 6 RAE 2822 test case with a moderately strong shock on the upper surface $M = 0.729$ $\alpha = 2.31°$.

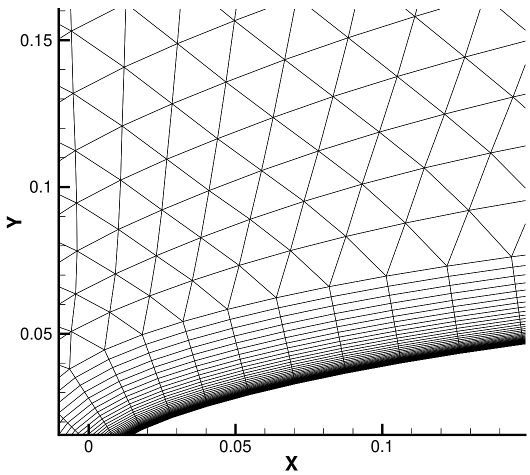


Fig. 7 Close-up near leading edge of RAE 2822 airfoil with a mixed-element grid.

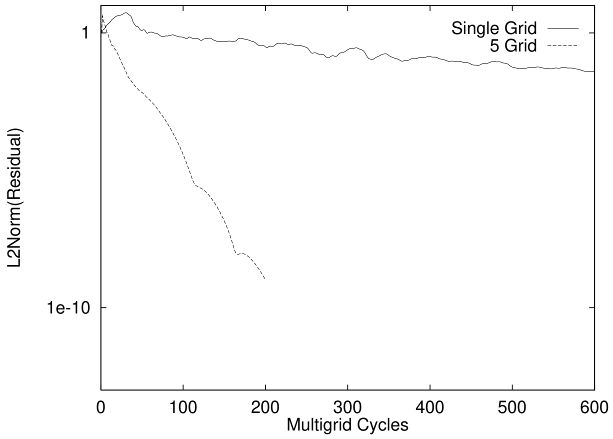


Fig. 8 Inviscid convergence history for single and 5 grid case $M = 0.73$ $\alpha = 2.31°$.

is added and subtracted to the parameters `FLeft` and `FRight` respectively. The vector incremental assignment operators $+=$ and $-=$ are defined within the vector class. In this case the parallel method consists of m 'AddVector' methods providing fine granularity.

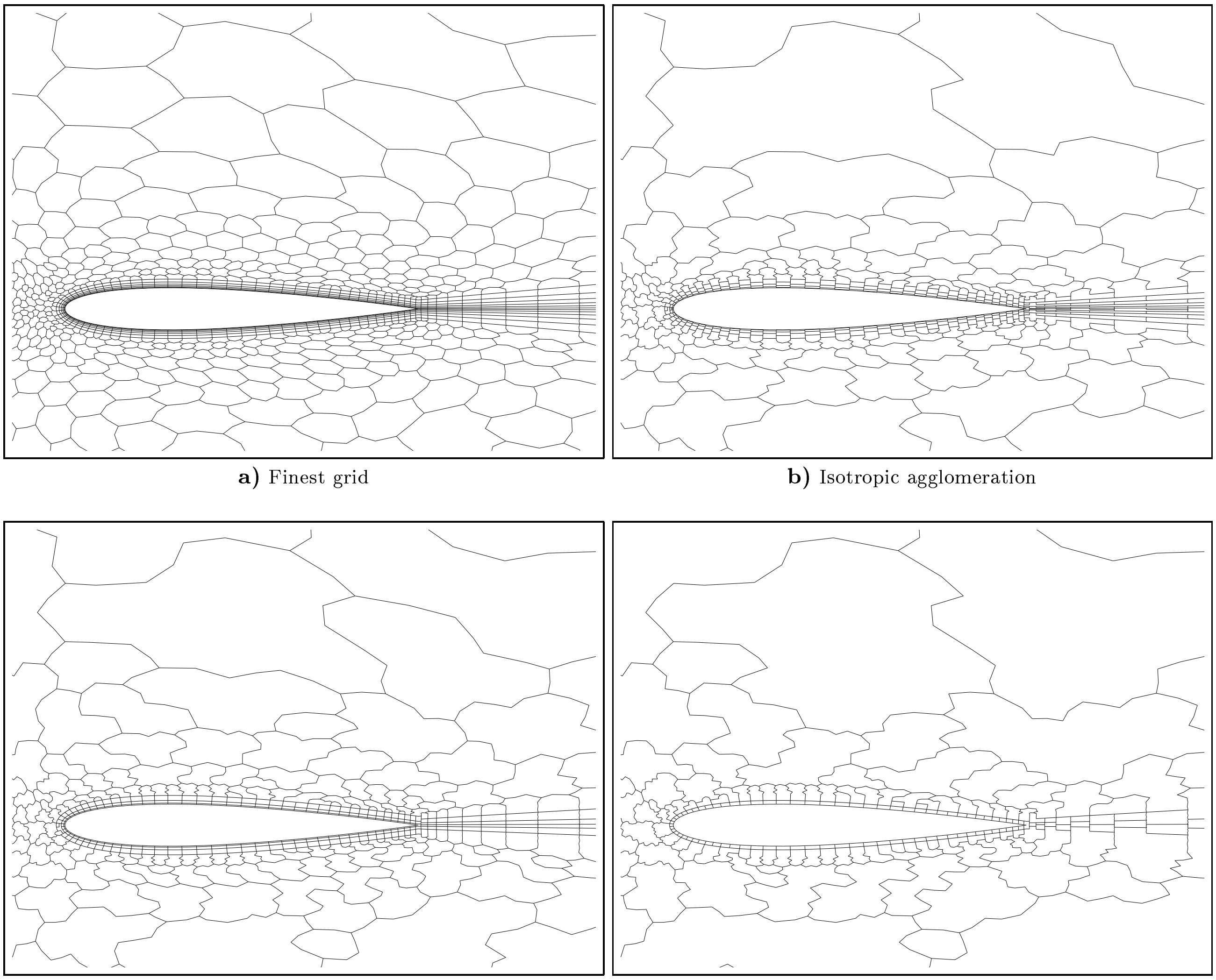
Although this example demonstrates two parallel methods, each consisting of a list of the same type of methods, a mix of methods may also be used. For example, if certain boundary conditions may be applied at the same time, they too can be added to the parallel method provided synchronization of data access is assured. Due to the fact that the solver is built at run-time by instantiating objects of defined methods, the solver can be configured and re-configured as desired. Although this adds complexity to the development of the solver, we believe this is overcome by the structured nature of the defined methods.

Results

To verify the accuracy of the solver, a viscous turbulent test case on an asymmetric airfoil was compared to experimental values and to computations using the well-known flow solver ARC2D from the NASA Ames Research Center.[15] The solution was computed on a mixed-element mesh with 31,354 nodes, as appears in Fig. 7. The ARC2D solution was computed on a C-grid with 30,291 nodes. As can be seen by the C_p plot in Fig. 6, the results agree favourably with both ARC2D and the experimental results. Figure 8 demonstrates the convergence acceleration provided by agglomeration multigrid with isotropic coarsening, for an inviscid transonic flow about the NACA 0012 airfoil on a grid with 5,700 nodes.

Figure 9 demonstrates the different variations of agglomeration. A hybrid grid with a structured viscous region and an unstructured inviscid region produces the cell-based dual presented in Fig. 9(a). A relatively coarse grid was chosen for ease of visualization. Isotropic agglomeration fusing all available neighbouring cells produces the coarse grid in Fig. 9(b). Directional coarsening ($\beta = 0.5$) produces the coarse grid in Fig. 9(c). Note in the viscous region the cell shape approaches an aspect ratio of 1.0. The difference between Fig. 9(b) and 9(c) is much more apparent in the wake region where the aspect ratio is increased in the former. Applying directional coarsening twice produces a grid that obtains an overall coarsening ratio of at least $4:1$ as shown in Fig. 9(d). In this example, directional coarsening applied twice produces a coarsening ratio larger than $4:1$ owing to a coarsening ratio of greater than $2:1$ in Fig. 9(c).

Figure 10 demonstrates the convergence history for laminar flow over the NACA 0012 airfoil at a Reynolds number of 5000. The maximum cell aspect ratio is 200. The directional coarsening ratio β was varied with $\beta = 0$ representing the isotropic coarsening case. All cases were completed with the same dissipation coefficients. The results indicate that $\beta = 0.5$ is optimal



a) Finest grid **b)** Isotropic agglomeration

c) Directional coarsening **d)** Directional coarsening applied twice

Fig. 9 An example hybrid grid agglomerated in with three strategies.

and provides a small improvement over isotropic coarsening. Larger improvements are expected on grids suitable for turbulent flows, which have higher cell aspect ratios.

Conclusions

We have presented progress towards a reliable efficient agglomeration multigrid solver for turbulent aerodynamic flows. Aspects of an object-oriented implementation of the solver have been discussed. Directional coarsening has been shown to provide a small improvement in the convergence rate for a laminar flow. Larger gains are expected for turbulent flows. Future work involves the development of an optimal directionally-coarsened agglomeration strategy for turbulent flows.

References

[1] Mavriplis, D. J., "Three Dimensional Unstructured Multigrid For The Euler Equations," *AIAA J.*, Vol. 30, No. 7, 1992, pp. 1753–1761.

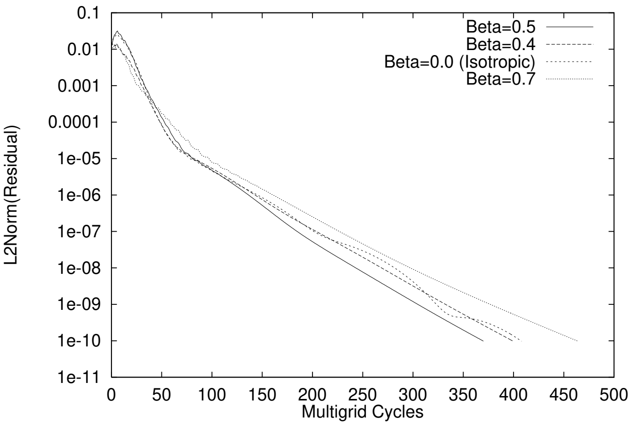


Fig. 10 Convergence history for a 4-grid V-cycle applied to laminar flow over a NACA 0012 airfoil $Re = 5000$ $M = 0.5$ $\alpha = 0$.

[2]Ollivier-Gooch, C., "Multigrid Acceleration of an Upwind Euler Solver on Unstructured Meshes," *AIAA J.*, Vol. 33, No. 10, 1995, pp. 1822–1827.

[3]Lallemand, M.-H., Stève, H., and Dervieux, A., "Unstructured Multigridding by Volume Agglomeration Current Status," *Comput. And Fluids*, Vol. 21, 1992, pp. 397–433.

[4]Mavriplis, D. J. and Venkatakrishnan, V., "A Unified Multigrid Solver for the Navier-Stokes Equations on Mixed Element Meshes," *Int. J. Comput. Fluid. D.*, Vol. 8, No. 4, 1997, pp. 247–263.

[5]Haselbacher, A. C., McGuirk, J. J., and Page, G. J., "Finite-Volume Discretisation Aspects for Viscous Flows on Mixed Unstructured Grids," AIAA-97-1946, June 1997.

[6]Morano, E., Mavriplis, D. J., and Venkatakrishnan, V., "Coarsening Strategies For Unstructured Multigrid Techniques with Application to Anisotropic Problems," *SIAM J. Sci. Comput.*, Vol. 20, No. 2, 1999, pp. 393–415.

[7]Mavriplis, D. J., "Multigrid Strategies for Viscous Flow Solvers on Anisotropic Unstructured Meshes," *J. Comput. Phys.*, Vol. 145, No. 1, 1998, pp. 141–165.

[8]Spalart, P. R. and Allmaras, S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," AIAA-92-0439, January 1992.

[9]Greiner, K., *A Finite Volume Solver for Viscous Turbulent Flows on Mixed Element Unstructured Meshes*, Master's thesis, University of Toronto Institute For Aerospace Studies, 1998.

[10]Swanson, R. and Turkel, E., "On Central-Difference and Upwind Schemes," *Computational Physics*, Vol. 101, 1992, pp. 292–306.

[11]Swanson, R. and Turkel, E., "Multistage Schemes with Multigrid for Euler and Navier-Stokes Equations," Tech. Rep. 3631, NASA, August 1997.

[12]Venkatakrishnan, V. and Mavriplis, D. J., "Agglomeration Multigrid for the Three-Dimensional Euler Equations," *AIAA J.*, Vol. 33, No. 4, 1995, pp. 633–640.

[13]Mavriplis, D. J., "Directional Agglomeration Multigrid Techniques for High Reynolds Number Viscous Flow Solvers," AIAA-98-0612, January 1998.

[14]"Introduction to the Standard Template Library," http://www.sgi.com/Technology/STL/, Address subject to change.

[15]Pulliam, T. H., "Efficient Solution Methods For The Navier-Stokes Equations," Lecture Notes For The Von Karman Institute For Fluid Dynamics Lecture Series: *Numerical Techniques For Viscous Flow Computation In Turbomachinery Bladings*, Jan. 1986.