Steady three-dimensional turbulent flow computations with a parallel Newton-Krylov-Schur algorithm

Michal Osusky[*] and David W. Zingg[†]

Institute for Aerospace Studies, University of Toronto, Toronto, Ontario, M3H 5T6, Canada

We present computations of steady three-dimensional turbulent flows with a parallel Newton-Krylov-Schur solution algorithm. The algorithm solves the Reynolds-Averaged Navier-Stokes equations coupled with the Spalart-Allmaras one-equation turbulence model, with the optional use of quadratic constitutive relations. The governing equations are discretized on multi-block structured grids using summation-by-parts operators with simultaneous approximation terms to enforce boundary and block interface conditions. We present a discussion on algorithm performance metrics, including speed, accuracy, and efficiency, and suggest several metrics which can be used for algorithm comparisons, such as convergence time, equivalent right-hand-side evaluations, computing time per grid node, and time required to reach specific functional error levels as compared to grid-converged values. The suggested metrics are applied to the current algorithm in the solution of subsonic and transonic flows around the ONERA M6 and NASA Common Research Model geometries. The results show effective algorithm convergence as grid resolution is increased, converging the residual by 12 orders of magnitude for all cases. A third geometry, based on the 1st AIAA High Lift Prediction Workshop delta wing configuration, provides difficulty in obtaining full convergence, leveling the residual off at a reduction of 4 to 5 orders of magnitude. However, the partially converged solutions show excellent agreement of lift and drag coefficients with experimental data in the angle of attack range of 1° to 40°.

I. Introduction

The widespread availability and ever-increasing use of massively parallel computational resources has allowed various fields of computational science to tackle problems that would have been considered out of reach a decade ago. Such is the case in the field of computational fluid dynamics (CFD), where access to supercomputer clusters has paved the way for the solution of large-scale simulations of flows around entire aircraft configurations. This ties into the results from the AIAA Drag Prediction Workshop series,[1] which indicate that grids with over $O(10^8)$ grid nodes are required for grid-converged lift and drag values for flows over a wing-body configuration. However, even with access to computational resources at an all-time high, algorithms that scale well with thousands of processors are required to make efficient use of those resources.

Flow solvers used for the solution of the Reynolds-Averaged Navier-Stokes (RANS) equations implement a wide variety of techniques. For example. popular RANS solvers such as OVERFLOW,[2] FUN3D,[3] Flo3xx,[4] and NSU3D,[5] make use of structured and unstructured grids, finite-volume or finite-difference approximations, and a plethora of combinations of linear solvers and preconditioners. The choices made in designing an algorithm can have an impact on the accuracy of the solutions, particularly the trend by which the algorithm approaches a grid-converged solution. This has been demonstrated through previous efforts, such as the various AIAA Drag Prediction Workshops.[6,7,8,9,10]

Due to the complexities of these algorithms and the significant programming skill required to develop a fast flow solver, detailed comparisons of algorithm performance have often been neglected. Instead, only blanket statements on algorithm performance are presented, such as the amount of time required to converge a particular case. While providing a general picture of the power of an algorithm, such statements can often

[*]Postdoctoral Fellow, Member AIAA (currently Mechanical Engineer, GE Global Research)

[†]Professor and Director, Tier 1 Canada Research Chair in Computational Fluid Dynamics and Environmentally Friendly Aircraft Design, J. Armand Bombardier Foundation Chair in Aerospace Flight, Associate Fellow AIAA

paint a misleading picture. In order to understand the performance of an algorithm, in particular when comparing to other solvers, it is important to understand how algorithm performance can be measured, and how each measure can impact the perceived performance.

First, one can consider the *speed* of an algorithm, or the CPU time to achieve a steady-state solution on a given mesh. While seemingly straightforward, using this metric for comparisons is not at all easy. The natural way to measure speed is in terms of iterations to convergence, but this does not allow for comparisons across different algorithms. Hence one can measure speed in terms of CPU time, but this approach is hardware dependent and also depends on programming effort and skill. One possibility to remove some algorithm and hardware dependency is to consider equivalent residual (RHS) evaluations. This measure is useful, but is strongly affected by the nature of the RHS (i.e. the spatial discretization). An expensive RHS (or an inefficiently coded one) can actually make a code appear faster by this measure. Other aspects of speed include scaling with mesh size and parallel scaling.

Next one can consider the *robustness* of the algorithm, that is how reliably does it converge over a wide range of geometries and flows. This is difficult to measure, but suggests that algorithm comparisons should include a suite of test cases of varying complexity. Robustness and speed are often competing objectives, so this must also be taken into account in choosing algorithm parameters and assessing algorithms. In particular, this can be the case when finely tuned parameters are required for each production run, adding overhead to the algorithm run time that is not typically reported.

Further difficulties are encountered when comparisons of algorithms of different spatial discretization order are performed. Higher-order algorithms will be inherently more expensive per iteration, but provide benefits elsewhere. With the more accurate discretizations, coarser meshes can be employed, reducing overall computation time. For this reason, algorithm comparisons should not rely on cases where a single grid level is utilized, which would negatively impact the higher-order algorithm comparison. The concept of *efficiency* rather than speed, should then be considered, where the amount of time to achieve a certain error, or level of accuracy, in the solution is monitored, rather than just the amount of time required to obtain a converged solution. To categorize efficiency, extra effort in determining solution error is required; the use of Richardson extrapolation[11] on a set of consistently refined grids is one possibility, and does not require extensive modifications to algorithms that do not have error prediction capabilities built in.

In an effort to identify the state-of-the-art in CFD algorithms and compile a list of potential best practices the technical discussion group of solver technology was established. This paper is part of a special session devoted to assessing the current trends in algorithm development for the solution of the Reynolds-Averaged Navier-Stokes equations. It presents the application of a Newton-Krylov-Schur parallel solution algorithm for the Reynolds-Averaged Navier-Stokes equations coupled with the Spalart-Allmaras one-equation turbulence model to a suite of flow problems with a view to characterizing the algorithm's performance. The algorithm employs summation-by-parts operators on multi-block structured grids, while simultaneous approximation terms are used to enforce boundary conditions and coupling at block interfaces.

The paper is divided into the following sections. Section II presents a brief overview of the governing equations and the flow solution algorithm. Section III presents a description of the algorithm benchmarking conducted, as well as notes about algorithm comparisons. Section IV contains results obtained with the current algorithm for steady flow solutions, with a focus on three-dimensional flows. Conclusions are given in Section V.

II. Flow Solution Algorithm

The flow solution algorithm considered in the current work is a parallel implicit finite-difference RANS equation flow solver called DIABLO. The following sections present an overview of the algorithm, with further details provided by Osusky and Zingg.[12]

A. Governing Equations

1. *The Navier-Stokes Equations*

The three-dimensional Navier-Stokes equations, after the coordinate transformation $(x, y, z) \rightarrow (\xi, \eta, \zeta)$, are given by

$$\partial_t \hat{\mathbf{Q}} + \partial_\xi \hat{\mathbf{E}} + \partial_\eta \hat{\mathbf{F}} + \partial_\zeta \hat{\mathbf{G}} = \frac{1}{Re}\Big(\partial_\xi \hat{\mathbf{E}}_\mathbf{v} + \partial_\eta \hat{\mathbf{F}}_\mathbf{v} + \partial_\zeta \hat{\mathbf{G}}_\mathbf{v}\Big), \tag{1}$$

with

$$
\begin{aligned}
&\hat{\mathbf{Q}} = J^{-1}\mathbf{Q}, \\
&\hat{\mathbf{E}} = J^{-1}\Big(\xi_x\mathbf{E} + \xi_y\mathbf{F} + \xi_z\mathbf{G}\Big), \quad \hat{\mathbf{E}}_\mathbf{v} = J^{-1}\Big(\xi_x\mathbf{E}_\mathbf{v} + \xi_y\mathbf{F}_\mathbf{v} + \xi_z\mathbf{G}_\mathbf{v}\Big), \\
&\hat{\mathbf{F}} = J^{-1}\Big(\eta_x\mathbf{E} + \eta_y\mathbf{F} + \eta_z\mathbf{G}\Big), \quad \hat{\mathbf{F}}_\mathbf{v} = J^{-1}\Big(\eta_x\mathbf{E}_\mathbf{v} + \eta_y\mathbf{F}_\mathbf{v} + \eta_z\mathbf{G}_\mathbf{v}\Big), \\
&\hat{\mathbf{G}} = J^{-1}\Big(\zeta_x\mathbf{E} + \zeta_y\mathbf{F} + \zeta_z\mathbf{G}\Big), \quad \hat{\mathbf{G}}_\mathbf{v} = J^{-1}\Big(\zeta_x\mathbf{E}_\mathbf{v} + \zeta_y\mathbf{F}_\mathbf{v} + \zeta_z\mathbf{G}_\mathbf{v}\Big),
\end{aligned}
$$

where $Re = \frac{\rho_\infty a_\infty l}{\mu_\infty}$, ρ is the density, a the sound speed, l the chord length, μ the viscosity, and J the metric Jacobian that results from the coordinate transformation. The '∞' subscript denotes a free-stream value for the given quantity. The notation ξ_x, for example, is a shorthand form of $\partial_x \xi = \frac{\partial \xi}{\partial x}$. The conservative variables and inviscid and viscous fluxes are given by

$$
\mathbf{Q} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ e \end{bmatrix}, \quad
\mathbf{E} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(e+p) \end{bmatrix}, \quad
\mathbf{F} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ v(e+p) \end{bmatrix}, \quad
\mathbf{G} = \begin{bmatrix} \rho v \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ w(e+p) \end{bmatrix},
$$

$$
\mathbf{E}_\mathbf{v} = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ E_{v,5} \end{bmatrix}, \quad
\mathbf{F}_\mathbf{v} = \begin{bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ F_{v,5} \end{bmatrix}, \quad
\mathbf{G}_\mathbf{v} = \begin{bmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ G_{v,5} \end{bmatrix},
$$

where e is the energy, p the pressure, u, v, and w the Cartesian velocity components, and τ the Newtonian stress tensor. The preceding variables have been made dimensionless by the use of the free-stream values of density and sound speed, as well as chord length. Laminar viscosity is calculated using a dimensionless form of Sutherland's law:[13]

$$
\mu = \frac{a^3(1 + S^*/T_\infty)}{a^2 + S^*/T_\infty}, \tag{2}
$$

where $S^* = 198.6°\text{R}$, and T_∞ is typically set to 460°R.

2. *Spalart-Allmaras One-Equation Turbulence Model*

In order to solve turbulent flows, a turbulent, or eddy, viscosity, μ_t, can be added to the viscosity, μ, based on the Boussinesq approximation.[14] The Spalart-Allmaras one-equation turbulence model[15] is used to compute the turbulent viscosity. The model solves a transport equation for a turbulence variable, $\tilde{\nu}$, that is related to turbulent viscosity. The model itself is a sixth equation that is solved concurrently with the five Navier-Stokes equations in a fully-coupled implicit manner.

The version of the model used in this work, labeled "SA" on the NASA Turbulence Modeling Resource (TMR) website,[16] is given by

$$
\begin{aligned}
\frac{\partial \tilde{\nu}}{\partial t} + u_i \frac{\partial \tilde{\nu}}{\partial x_i} = \frac{c_{b1}}{Re}[1 - f_{t2}]\tilde{S}\tilde{\nu} + \frac{1 + c_{b2}}{\sigma_\text{t} Re}\nabla \cdot [(\nu + \tilde{\nu})\nabla\tilde{\nu}] - \frac{c_{b2}}{\sigma_\text{t} Re}(\nu + \tilde{\nu})\nabla^2\tilde{\nu} \\
- \frac{1}{Re}\left[c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2}\right]\left(\frac{\tilde{\nu}}{d}\right)^2,
\end{aligned} \tag{3}
$$

where $\nu = \frac{\mu}{\rho}$.

The spatial derivatives on the left side of the equation represent advection. The first term on the right side represents production, while the second and third terms account for diffusion. The fourth term represents destruction. Reference 15 contains the values of all of the terms and constants that appear in (3). Importantly, precautions are taken to ensure that neither the vorticity, S, nor the vorticity-like term, $\tilde{S}$, approaches zero or becomes negative, which could lead to numerical problems. The off-wall distance at each computational node is denoted by d.

An important consideration in the use of the Spalart-Allmaras turbulence model is the method by which the off-wall distance, d, is calculated. Many implementations, in an effort to save computational time and reduce code complexity, make use of approximations in calculating the distance from each volume node to the closest solid boundary. These range from following grid lines towards a surface, a method that itself becomes very complex when dealing with multi-block grids, to calculating distances to individual grid nodes on the surface. While these methods provide reasonably accurate representations of d, certain grid topologies can result in values that have a substantial effect on the force coefficients. This effect is due to the sensitivity of the Spalart-Allmaras turbulence model to the values used for d. It is therefore highly recommended that a method which identifies the entire surface, including edges and surface segments between the nodes on a solid boundary, be used in calculating d, as it will provide more accurate off-wall distance values on a wide variety of grids, regardless of blocking or gridding anomalies. Such an approach is used in the current algorithm.

A notable feature of the current algorithm is the use of quadratic consitutive relations (QCR). Spalart[17] presented a modification to the Reynolds stress tensor that results from the Boussinesq approximation, producing a nonlinear stress model:

$$\tau_{ij} = \bar{\tau}_{ij} - c_{nl1}\left(O_{ik}\bar{\tau}_{jk} + O_{jk}\bar{\tau}_{ik}\right), \tag{4}$$

where

$$\begin{aligned} O_{ik} &\equiv \frac{\partial_k u_i - \partial_i u_k}{\sqrt{\partial_n u_m \partial_n u_m}}, \\ \partial_n u_m \partial_n u_m &= (\partial_x u)^2 + (\partial_x v)^2 + (\partial_x w)^2 \\ &\quad + (\partial_y u)^2 + (\partial_y v)^2 + (\partial_y w)^2 \\ &\quad + (\partial_z u)^2 + (\partial_z v)^2 + (\partial_z w)^2, \end{aligned}$$

u_i are the velocity components (u, v, w), and $c_{nl1} = 0.3$. Additionally, $\bar{\tau}_{ij}$ is the Reynolds stress tensor obtained from the Boussinesq approximation. The implementation of the above modification does not add substantial development time and has negligible effect on the convergence characteristics of the algorithm,

The use of the above modification to the Reynolds stress tensor has seen limited adoption. However, some results presented during the fourth and fifth DPW have incorporated QCR,[9,18,10] with positive effects in the solutions presented. In particular, the use of QCR has a substantial impact on the size of the recirculation region at the wing-body junction once a critical angle of attack is reached. The current algorithm has demonstrated the effective use of QCR in the solution of the cases for the fifth DPW.[19]

Where appropriate, the use of QCR in the results section will be labeled as "SA-QCR2000," as prescribed on the TMR website.

B. Solution Methodology

The following sections present a brief overview of the solution algorithm used for this work, DIABLO, including the spatial discretization method used, the overall solution strategy, as well as some basic implementation details. Further details of the algorithm can be found in references 20, 21, and 22, with a complete description of the RANS solution methodology of the current algorithm presented in reference 12.

1. Spatial Discretization

The spatial discretization of the Navier-Stokes equations and the turbulence model is obtained with second-order Summation-By-Parts (SBP) operators, while inter-block coupling and boundary conditions are enforced weakly by the use of Simultaneous Approximation Terms (SATs). We decompose our domains into multiple blocks, resulting in multi-block structured grids. Not only does this type of blocking strategy work well with the SBP-SAT discretization, it also allows for relatively straightforward creation of meshes around complex geometries.

The SBP-SAT approach requires minimal information from adjoining blocks in order to obtain a discretization of the governing equations at block interfaces. This results in a reduced requirement for information sharing between blocks, which is especially advantageous for parallel algorithms, reducing the time spent in communication. Additionally, the fact that this discretization does not need to form any derivatives across interfaces reduces the continuity requirements for meshes at interfaces. In fact, only C^0 continuity is necessary, allowing the algorithm to provide accurate solutions even on grids with relatively high incidence angles for grid lines at interfaces. This feature substantially reduces the burden placed on the grid generation process, especially for complex three-dimensional geometries, such as the NASA Common Research Model (CRM).[23]

Numerical dissipation is added using either the scalar dissipation model developed by Jameson *et al.*[24] and later refined by Pulliam,[25] or the matrix dissipation model of Swanson and Turkel.[26]

It should be emphasized that while the current spatial discretization is second-order accurate, the SBP-SAT approach allows for straightforward extension to higher orders.

2. *Iteration to Steady State*

Applying the SBP-SAT discretization to the governing equations results in a large system of nonlinear equations:

$$\mathcal{R}(\mathcal{Q}) = 0, \tag{5}$$

where $\mathcal{Q}$ represents the complete solution vector. When time-marched with the implicit Euler time-marching method and a local time linearization, this results in a large set of linear equations of the form:[27]

$$\left(\frac{\mathcal{I}}{\Delta t} + \mathcal{A}^{(n)} \right) \Delta \mathcal{Q}^{(n)} = -\mathcal{R}^{(n)}, \tag{6}$$

where n is the outer (nonlinear) iteration index, Δt is the time step, $\mathcal{I}$ is the identity matrix, $\mathcal{R}^{(n)} = \mathcal{R}(\mathcal{Q}^{(n)})$, $\Delta\mathcal{Q}^{(n)} = \mathcal{Q}^{(n+1)} - \mathcal{Q}^{(n)}$, and

$$\mathcal{A}^{(n)} = \frac{\partial \mathcal{R}^{(n)}}{\partial \mathcal{Q}^{(n)}}$$

is the Jacobian.

In the infinite time step limit, the above describes Newton's method and will converge quadratically if a suitable initial iterate, $\mathcal{Q}^{(0)}$, is known. This initial iterate must be sufficiently close to the solution of (5). Since it is unlikely that any initial guess made for a steady-state solution will satisfy this requirement, the present algorithm makes use of a start-up phase whose purpose is to find a suitable initial iterate.

To solve the large set of discrete equations, we use a parallel Newton-Krylov algorithm using the flexible GMRES Krylov subspace iterative method with the approximate-Schur preconditioner.[28] Pseudo-transient continuation is used to march the solution from an initial guess to steady state, employing a spatially varying time step.

The general framework of the algorithm has also been leveraged to create an efficient time-accurate solution strategy.[29]

C. Implementation Details

Implementation details can have a substantial impact on the performance of an algorithm. Hence, this section presents some basic information on the algorithm presented in this paper, providing a way to clearly identify inherent differences that may affect comparisons to other codes.

At the most basic level, the vast majority of the code is written in Fortran90, with a few legacy subroutines written in Fortran77. All parallel communication is performed using the Message-Passing Interface (MPI). Recent structural analysis modules are written in C++ and coupled to the remainder of the algorithm using Python. However, none of these subroutines were used for the cases presented here.

The results presented in this paper were run on the Scinet GPC cluster, which consists of 3864 8-core nodes. Each node contains two Intel Xeon E5540 processors running at 2.53GHz, with 16GB RAM per node (roughly 2GB per core, less the memory required for the operating system). The majority of the nodes are connected with 5:1 blocked QDR InfiniBand, while approximately a quarter of the cluster is interconnected with non-blocking DDR InfiniBand. The code was compiled on the Linux operating system with the Intel Fortran compiler using IntelMPI.

In order to make efficient use of the parallel computational systems, the meshes used for this algorithm are load balanced *a priori*. This is achieved either during the mesh creation process, when blocks of equal sizes are created, or with a load balancing tool that can subdivide existing meshes. The latter approach, as detailed by Apponsah and Zingg,[30] can be used to reduce the computational time required substantially on meshes with poor load balancing.

Finally, it should be noted that DIABLO is primarily an academic development code, and has therefore not been subjected to the extensive computer software optimization that full-blown production codes are. Additional performance improvements could likely be extracted through a rigorous coding overhaul, including the use of OpenMP, but this is not an objective of the present work.

III. Algorithm Benchmarking and Comparison Notes

A. Note on Algorithm Comparisons

One of the most convenient ways of measuring algorithm performance involves comparing the CPU time that is required to converge to a steady state, more commonly referred to as the *speed* of an algorithm. This approach, however, has many pitfalls. First, CPU time is hardware dependent, and will often vary widely based on programmer skill and hardware set-up, particularly for large distributed systems. Cache size and the type of parallel interconnect architecture could slow down an otherwise fast algorithm. In other words, an algorithm that is fast on one supercomputing cluster might not behave as well when installed on a different system. Additionally, various flow solution algorithms use different residual convergence criteria for identifying converged solutions. Hence, comparisons have to be made with solutions that are converged to the same tolerance.

In addition, an important topic to be aware of is the importance of algorithm speed versus algorithm *robustness*. For representative results, the parameters used in obtaining a converged solution, such as time step parameters, should belong to a default set. This would prevent a misrepresentation of algorithm speed, such as when cases that converge with aggressive parameter values are shown to converge rapidly, even if those parameters would lead to divergence of most other cases considered. At the same time, it would be unwise and wasteful to use parameters that allow all cases to converge, as that would lead to unnecessarily conservative choices. A compromise must be reached where a clear default set of parameters is selected, and only minimal tuning is required to converge the more problematic cases. This approach is especially important for flow solution algorithms that are part of an optimization framework; they need to be robust due to the number of potential geometries they will encounter during an optimization, but also fast, since they will be called many times during the course of an optimization. The specific choices are left to each operator, but they should be stated clearly. Ideally, algorithm comparisons should include a suite of test cases of varying complexity, giving a clearer picture of algorithm capability and performance.

Whenever algorithm speed is being considered, it is also important to consider the notion of *efficiency*, defined as the time required to reach a prescribed error level in the solution. This becomes especially important when comparing algorithms of different spatial discretization accuracy levels. For example, to achieve the same error in a converged solution, a higher order algorithm, which is more expensive on a per-grid-node basis, would require a coarser mesh than a lower order algorithm. However, on a single mesh, the higher-order algorithm would require more time to obtain a converged solution, potentially leading to the conclusion that the algorithm is slower. This clearly demonstrates the need to not only report the time required in obtaining a solution, but also the error in the obtained solution. Hence, a means of consistently quantifying solution error is needed, such as a grid convergence study using Richardson extrapolation.

Understanding code performance relies heavily on detailed information about the code and the hardware it is run on. While hardware characteristics are easily communicated, the subtleties of code development are more difficult to describe, or perhaps even proprietary. However, at least a basic effort should be made. In addition, pre- and post-processing steps that are often ignored when discussing code performance should be included, with a common start and end point used for quantifying computations. Hence, instead of simply reporting how long the iterations to converge the flow solution took, the overall time could span the instant the code begins execution (once it is provided with a grid and input parameter file), to the instant the complete solution file is written. This would provide a clearer picture of any initialization costs, such as for MPI or any computations that are done before the iterative process, such as an off-wall distance calculation. This would also provide some sense of IO costs. This type of discussion will not only help show key differences between algorithms, but also serve in identifying best practices in parallel algorithm development that can then be adopted by more codes.

The above is not meant to dissuade the use of any of the stated metrics in determining algorithm performance. The purpose here is to encourage transparency in algorithm testing, both in terms of implementation and the hardware used, as this will make it easier and more straightforward to compare different algorithms on different systems. By understanding the limitations of each metric, a thorough algorithm comparison would present several of the metrics along with hardware and software specifications, providing a clearer picture of the true performance of the algorithm.

B. Benchmarking Methodology

The current algorithm has been thoroughly verified and validated with a range of two- and three-dimensional test cases. These included the solution of turbulence model verification and validation cases provided on the TMR website, as well as the solution of three-dimensional flows around wings and wing-body configurations. The results of these tests can be found in references 12, 19, 21, 22, 31, and 32. The references also provide results of parallel scaling tests that demonstrate the parallel scaling performance of the algorithm with up to 6656 processors.

For the purposes of this paper, several metrics are presented to provide more straightforward comparisons to other solution algorithms. For all cases run, CPU-time-based metrics will serve as the basis of most comparisons, spanning the following values:

Total time Time required for the complete execution of the algorithm. This includes initialization of the code, pre-iteration calculations (such as off-wall distance calculation), and solution output/cleanup.

Iteration (solution) time Time required to converge the solution (12 orders of magnitude residual reduction), starting at the first iteration.

Equivalent right-hand-side evaluation time Iteration time divided by the time required to form the residual vector. Used for facilitating comparison of current implicit algorithm to explicit algorithms.

CPU time/grid nodes A measure designed to provide computational cost per grid node, allowing comparison of cases using different grid sizes and numbers of processors. This metrics is calculated as (total time)*(number of processors)/(total number of grid nodes).

For the purpose of defining efficiency, all cases are accompanied by a grid convergence study. A grid convergence study can be used to analyze the behaviour of a functional, F, which can represent quantities such as coefficients of lift, drag, and moment, (C_L, C_D, and C_M, respectively). Solving each case on at least three consistently refined grid levels allows for the calculation of a grid-converged functional, F^*, as well as convergence order, p, obtained from

$$\frac{F_F - F_M}{F_M - F_C} = \frac{r^p - 1}{r^{2p} - r^p} \tag{7}$$

$$F^* = F_F + \frac{F_F - F_M}{r^p - 1} \tag{8}$$

where r is the grid refinement factor and (F_C, F_M, F_F) are the functionals computed on the coarse, medium, and fine meshes, respectively. The error is calculated as

$$\text{error} = \left| \frac{F_a - F^*}{F^*} \right| \tag{9}$$

where $a = (C, M, F)$. Error is plotted versus solution time for each case to provide a measure of efficiency for the algorithm. When more than three grid levels are available, the value of F^* is determined from the finest set of consecutive grid levels that produce a monotonic sequence of functional values.

IV. Results

In order to demonstrate the performance of the current algorithm, DIABLO, this section presents a variety of cases, including subsonic and transonic flow solutions over wings and wing-body configurations. All cases contain grid convergence studies that help characterize the efficiency of the algorithm, along with the convergence characteristics of the algorithm.

Finally, we present a flow solution that has proven to be problematic for the current algorithm, and discuss the potential causes for the lack of deep convergence.

Table 1. **Flow solution cases**

Case	*Ma*	*Re*	Angle of attack
ONERA M6 - t	0.8395	11×10^6	3.06°
ONERA M6 - s	0.40	3×10^6	2.00°
CRM - t1	0.85	5×10^6	2.28°
CRM - t2	0.85	5×10^6	2.229°
CRM - s1	0.60	5×10^6	3.50°
CRM - s2	0.60	5×10^6	5.00°

Table 2. **Grid information**

Grid level	Blocks	Nodes, N	Average off-wall spacing
ONERA M6 - **C**oarse	128	628,824	4.35×10^{-6}
ONERA M6 - **M**edium	128	4,599,936	1.92×10^{-6}
ONERA M6 - **F**ine	128	35,152,000	9.01×10^{-7}
CRM - **T**iny	6656	1,164,800	1.03×10^{-5}
CRM - **C**oarse	6656	3,261,440	6.84×10^{-6}
CRM - **M**edium	6656	7,008,768	5.13×10^{-6}
CRM - **F**ine	6656	21,372,416	3.42×10^{-6}
CRM - **X**-fine	6656	48,089,600	2.56×10^{-6}

A. Flow solution case details

Two three-dimensional geometries are considered for the studies presented in Section IV.B, namely the ONERA M6 wing and the NASA Common Research Model (CRM) wing-body geometry.[23] Both subsonic and transonic flow solutions were performed for both geometries, as outlined in Table 1. In particular, the transonic cases for the CRM geometry reproduce the flow solutions from the 5th AIAA Drag Prediction Workshop (DPW5), using the target angles of attack for the medium and extra-fine grid levels attaining the required lift coefficient of $C_L = 0.5$. For the current study, the angle is held constant for all grid levels. The ONERA M6 solutions were computed with the standard turbulence model, labeled "SA", while the CRM solutions were computed using the "SA-QCR2000" model variant based on experiences from DPW5.

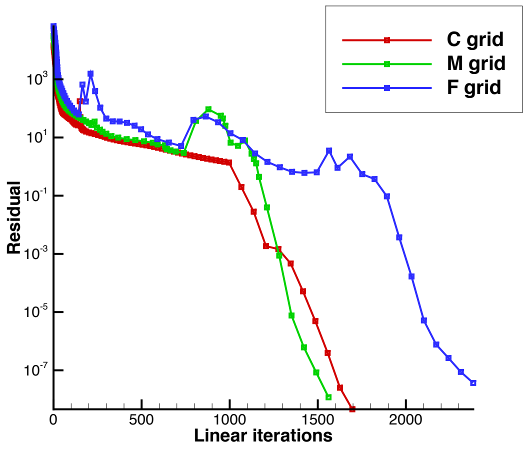
Three grid levels were used for all ONERA M6 studies, while five grid levels were used with the CRM. While the CRM grids, taken from DPW5[33] and subdivided into equally-sized blocks, make use of 1.5-times refinement in each direction, the ONERA M6 grids use the more typical 2-times refinement. This has to be taken into account when calculating the Richardson extrapolation order and grid-converged functional values. The grid details are outlined in Table 2.

For the purpose of the current study, the number of processors for obtaining solutions on each geometry were held constant. For the ONERA M6 cases, 128 processors were used, while the CRM solutions used 832 processors, all with perfect load balancing. The residual was converged by 12 orders of magnitude.

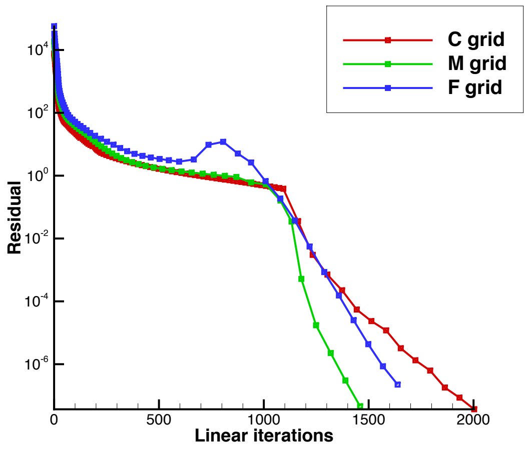
B. Functional convergence and algorithm efficiency

The residual convergence histories for the flow solutions presented in this section are shown in Figure 1. The plots present the convergence versus the total number of linear iterations, rather than CPU time, allowing a more direct comparison of the algorithm on the various grid levels. The symbols in the plots represent the outer (Newton) iterations of the solution algorithm. As can be seen from the plots, the current algorithm maintains very good convergence characteristics for the majority of grids considered, for both subsonic and transonic cases. The convergence slows down once very fine grids are used, due to the increase in linear iterations required to converge the linear system in late stages of convergence. It should be noted, though, that the solution algorithm parameters were held constant for all solution sets, with the algorithm exhibiting good convergence behavior on grids differing in size by more than two orders of magnitude.

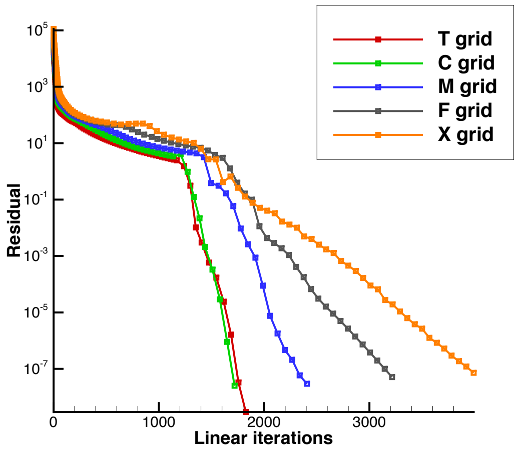
Table 3 summarizes the functional values obtained on all grid levels, along with the convergence order, p,



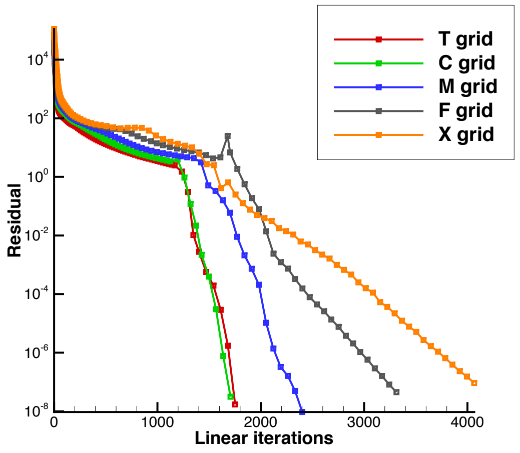
(a) ONERA M6 - t



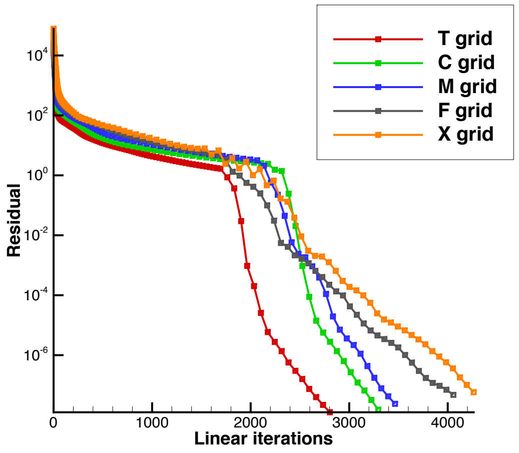
(b) ONERA M6 - s



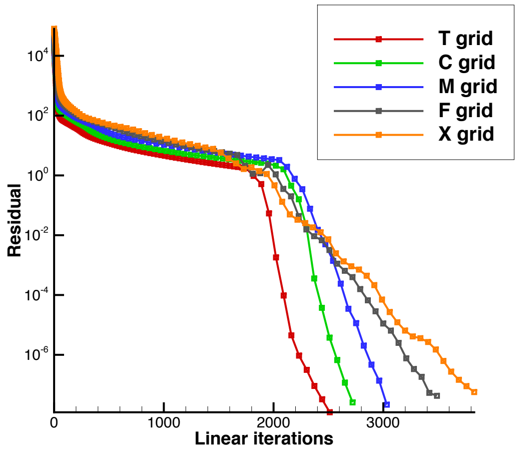
(c) CRM - t1



(d) CRM - t2



(e) CRM - s1



(f) CRM - s2

Figure 1. Residual convergence (12 orders of magnitude reduction)

Table 3. Functional values

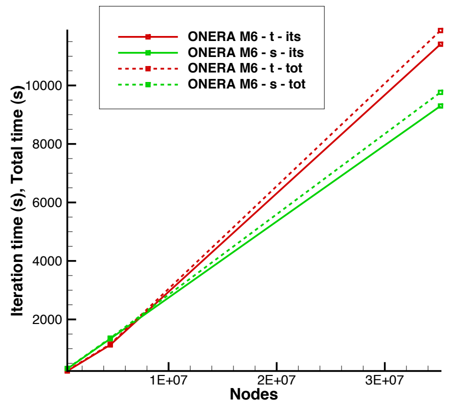
Case	Grid level	C_L	F^*	p	C_D	F^*	p
ONERA M6 - t	1-C	0.25581			0.020159		
	2-M	0.26532			0.017898		
	3-F	0.26986	0.27401	1.07	0.017086	0.016630	1.48
ONERA M6 - s	1-C	0.12905			0.015179		
	2-M	0.12925			0.011237		
	3-F	0.12998	NM	NM	0.010525	0.010368	2.47
CRM - t1	1-T	0.49254			0.027521		
	2-C	0.50225			0.025891		
	3-M	0.50619	0.50726	2.22	0.025575	0.025555	4.04
	4-F	0.51110	NM	NM	0.025599	NM	NM
	5-X	0.51397	0.51588	1.32	0.025685	NM	NM
CRM - t2	1-T	0.48538			0.027104		
	2-C	0.49482			0.025484		
	3-M	0.49865	0.49969	2.22	0.025169	0.025149	4.04
	4-F	0.50341	NM	NM	0.025189	NM	NM
	5-X	0.50619	0.50805	1.32	0.025270	NM	NM
CRM - s1	1-T	0.50863			0.027499		
	2-C	0.51934			0.025828		
	3-M	0.52359	0.52469	2.28	0.025362	0.025303	3.15
	4-F	0.52678	0.53185	0.71	0.025265	0.025257	3.86
	5-X	0.52781	0.52799	2.78	0.025281	NM	NM
CRM - s2	1-T	0.61471			0.036027		
	2-C	0.64208			0.033583		
	3-M	0.65372	0.65723	2.11	0.328380	0.032725	2.92
	4-F	0.66155	0.66963	0.98	0.032625	0.032597	3.10
	5-X	0.66379	0.66408	3.09	0.032614	0.032614	7.21

and grid-converged functional value, F^*. When more than three grid levels are present, the p and F^* values are calculated based on three consecutive grid levels. Hence, more than one set of values will be present for each case. Additionally, "NM" stands for non-monotonic, which signifies that the functionals do not follow consistent trends of either increasing or decreasing in proportion to changing grid size, which prevents the use of Richardson extrapolation. In many cases, the second-order algorithm attains its design order, or better, especially for the drag coefficient. For error calculations, which are presented later, the F^* value used is taken from the finest available set of three grids that produce a monotonic sequence of functional values.

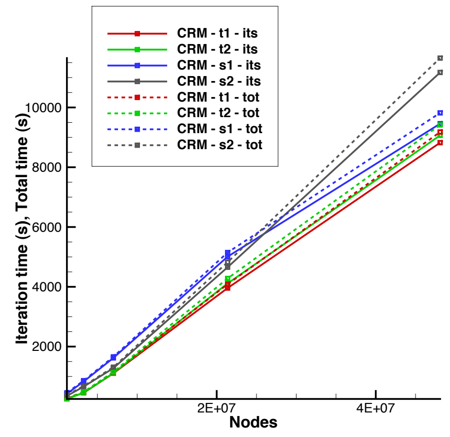
Figure 2 provides a graphical representation of solution time (both total and iteration), showing that we do not incur a set-up time penalty as the grid size is increased. This signifies that any overhead costs, such as off-wall distance calculation or reading and writing of files, scales as well as the solution algorithm itself.

Figures 3 and 4 show the equivalent RHS evaluations for each grid level and the total CPU time divided by number of nodes in a grid, respectively. Both of these measures can be used to compare the current solution algorithm to other codes, as they provide a metric that is independent of algorithm type (explicit or implicit). The second metric, which effectively provides an estimate of computational expense per grid node, highlights the effectiveness of the current algorithm as grid size is increased, signified by the relatively level plot line. This metric can further be made more general by dividing the value by the number of orders the residual was decreased, in this case 12, allowing for easier comparison of algorithms that attain various levels of convergence.

Figures 5 and 6 demonstrate the behaviour of error in the lift and drag coefficients, respectively. The errors are calculated relative to the grid-converged value obtained using Richardson extrapolation, using (9).

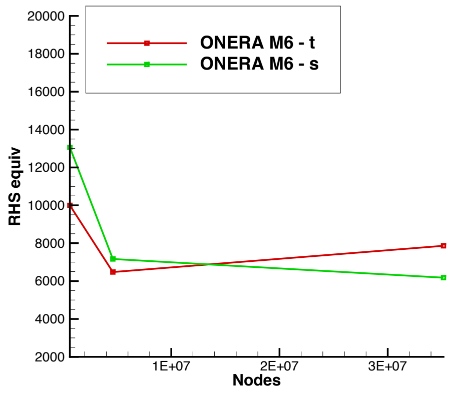


(a) ONERA M6

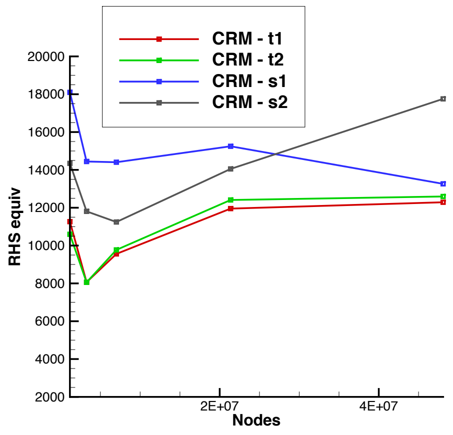


(b) NASA CRM

Figure 2. Convergence times

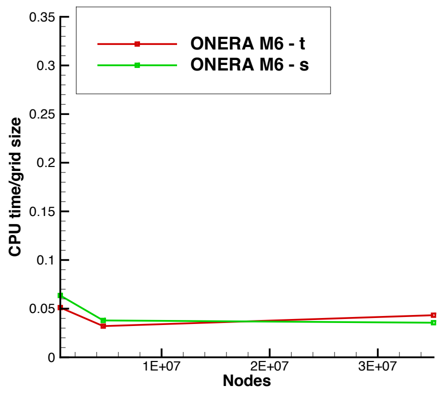


(a) ONERA M6

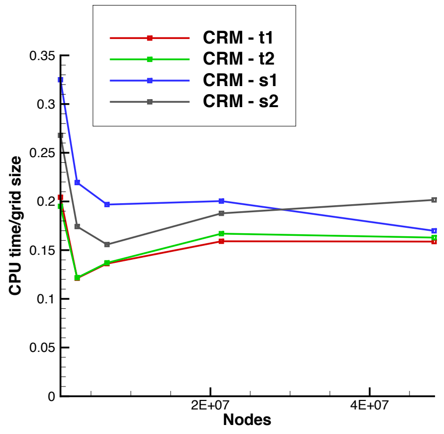


(b) NASA CRM

Figure 3. Equivalent RHS evaluations

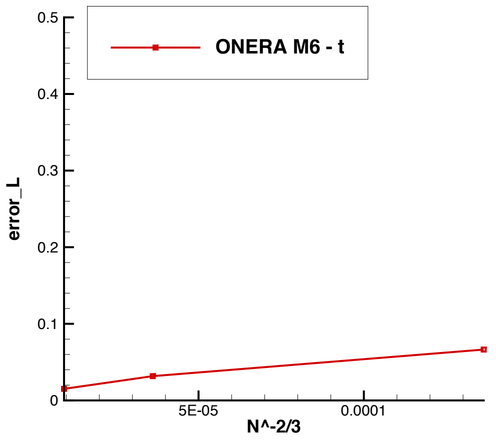


(a) ONERA M6

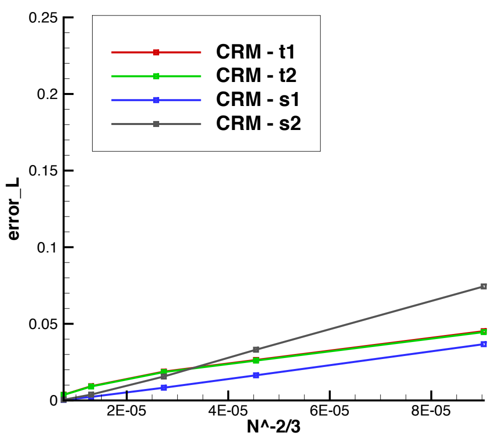


(b) NASA CRM

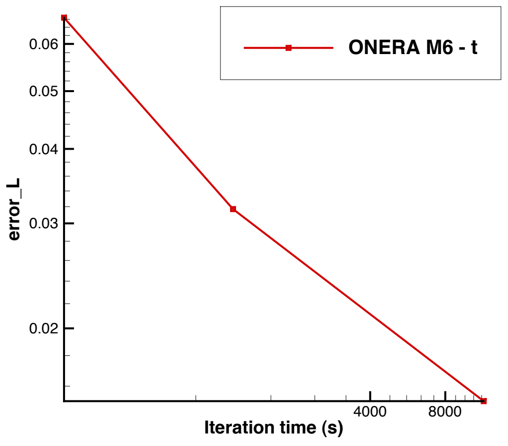
Figure 4. Total CPU time/Grid size



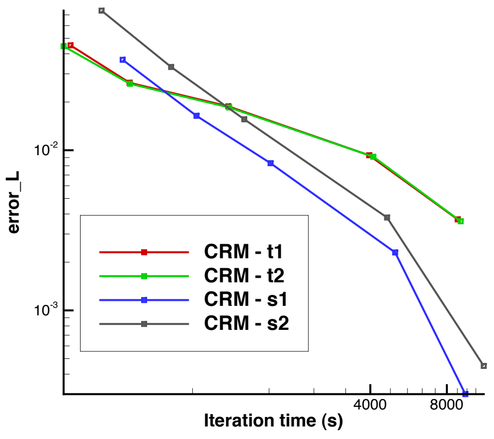
(a) ONERA M6



(b) NASA CRM



(c) ONERA M6



(d) NASA CRM

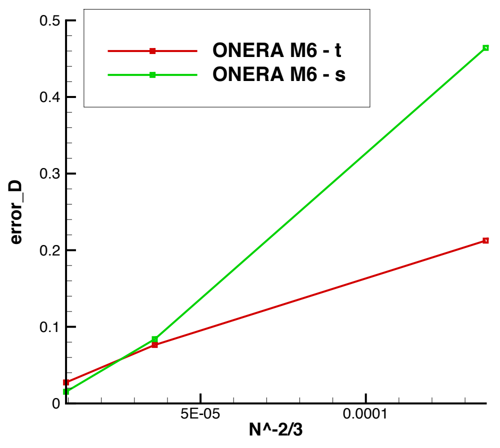
Figure 5. Error in lift coefficient

The errors are plotted against grid size and iteration time, characterizing the behaviour of the algorithm as grid density is increased. Of note, if non-monotonic functional behaviour was observed on the finer grid levels, the error values for those grids are not included in the plots. Plots such as these can be used to make decisions on grid resolution requirements, especially when a functional estimate within a certain error range is acceptable. For example, one can see both the mesh density and CPU requirements to reduce both C_L and C_D errors below 1%. For case "CRM - s1," using 832 processors, the algorithm attains C_L and C_D errors of 0.83% and 0.41%, respectively, in approximately 30 minutes on the 6656-block grid with 7 million nodes.

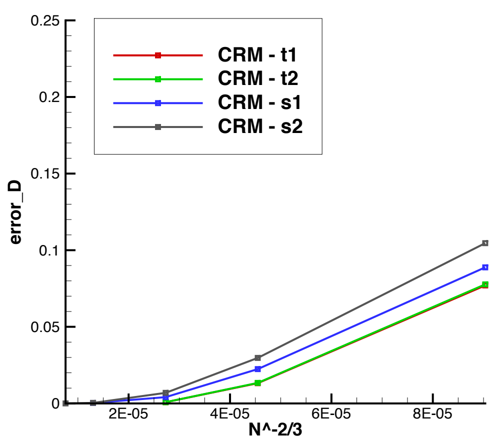
C. High-Lift Configuration

The final case presented is the solution of flow around the delta wing geometry, which was the subject of the 1st AIAA CFD High Lift Prediction Workshop (HiLiftPW-1).[34] The geometry consists of a main wing, along with a deployed slat and flap, referred to in the workshop as "Config 1," and is subject to the following flow conditions:

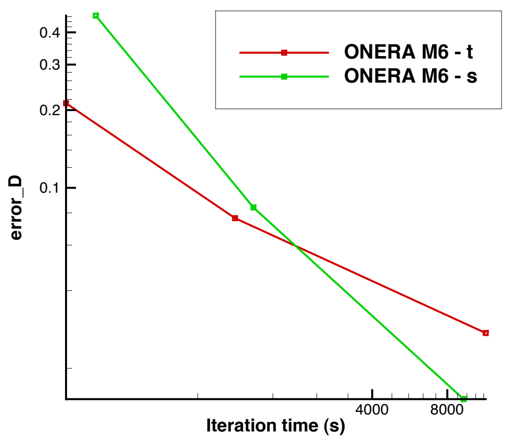
$$Ma = 0.20,\ Re = 4.3 \times 10^6,\ \alpha = [1°,13°,19°,28°,33°,35°,40°].$$



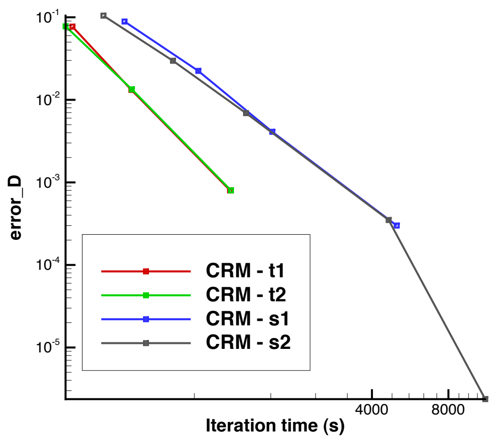
(a) ONERA M6



(b) NASA CRM



(c) ONERA M6



(d) NASA CRM

Figure 6. Error in drag coefficient

The choice of α values is based on the range of flow conditions available in experimental data.[a] The grid used in this study was obtained from the HiLiftPW-1 website, based on the JAXA-supplied coarse grid. The original grid was subdivided into 1395 blocks, with a total of 13.3 million nodes and an average off-wall spacing of 4.8×10^{-6}.

The algorithm experiences a stall in residual convergence once the inexact-Newton stage of the solution process is reached. This type of behaviour is typically seen when the solution produced in the approximate-Newton stage is not a suitable initial iterate for the final phase. However, for the solutions around the delta wing geometry, changing the residual reduction at which the switch between phases occurs has no impact on the final phase convergence. Figure 7 presents a comparison of the residual convergence at several angles of attack, demonstrating the stall. The deepest convergence is observed for the flow solution at $\alpha = 33°$, but with substantial convergence slowdown. Of note, the linear solver (GMRES) convergence during the final iterations is on par with that achieved for the fully converged ONERA M6 and CRM solutions presented in the previous section; hence, the convergence stall does not appear to be related to insufficient numerical convergence of the linear system. This solver behaviour was observed with both variants of the turbulence model (SA and SA-QCR2000).

[a]Experimental data accessed from `http://hiliftpw.larc.nasa.gov/Workshop1/experiment.html` on November 14, 2013.

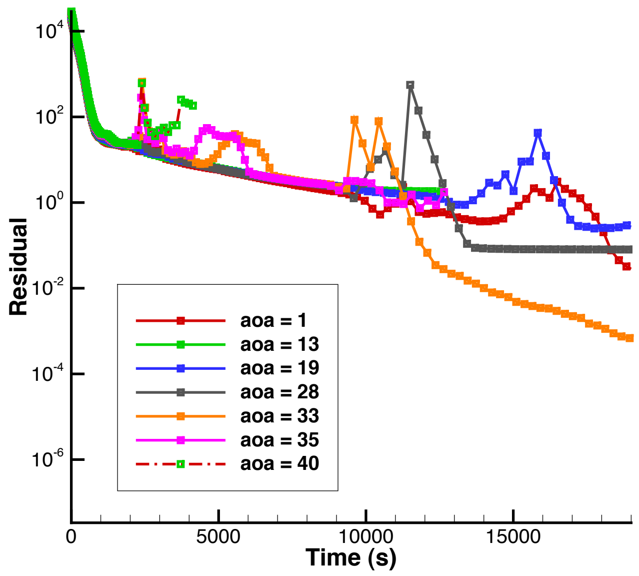


Figure 7. Residual convergence for delta wing flow solutions

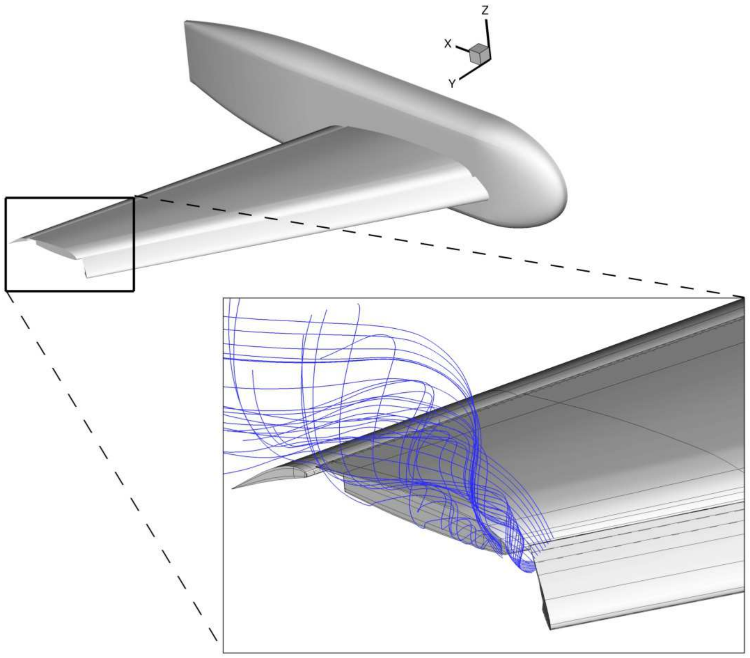
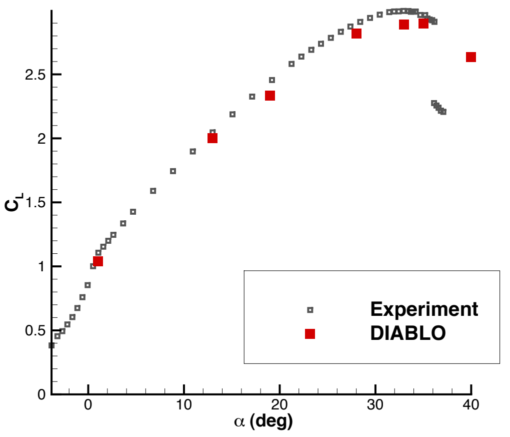
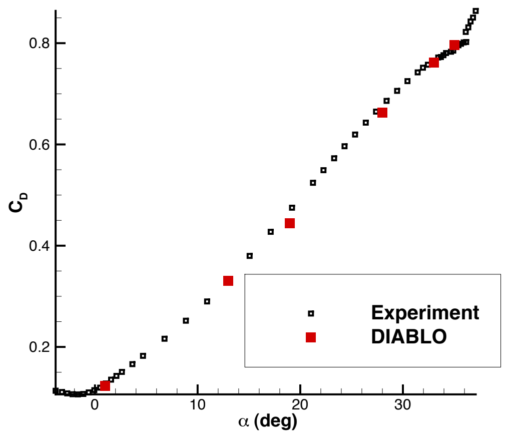


Figure 8. Vortical structures present at leading edge wing-tip for flow at $\alpha = 28^\circ$

It is unclear what is causing the stall in residual. From a careful examination of the flow field, and the values of the residual, it was observed that the largest residual values are located near the region of the leading edge wing-tip. Large vortical structures are present, as shown in Figure 8, which may be contributing to the residual stall. In addition, it is possible that the cases are experiencing at least partially unsteady flow, which could account for the poor nonlinear residual convergence.



(a) Coefficient of lift



(b) Coefficient of drag

Figure 9. Force coefficients for high lift delta wing geometry

However, even with the stalled residual convergence, the coefficients of lift and drag at the various angles of attack follow the expected trends, as shown in Figure 9. The numerical results are plotted with experimental data for the geometry. In fact, the discrepancy from the experimental values of lift coefficient is likely due to the lower grid resolution of the current grid, rather than the incomplete convergence, and the data show remarkable agreement for the drag coefficient. To provide some correlation between force coefficients and residual convergence, by 4 levels of residual reduction, the lift and drag coefficients are only changing in the 4th significant figure for the $\alpha = 33°$ solution.

V. Conclusions

This paper presented the performance characteristics of a Newton-Krylov-Schur algorithm for the solution of the RANS equations and the Spalart-Allmaras one-equation turbulence model, with an optional implementation of quadratic constitutive relations. In an effort to facilitate flow solver algorithm comparisons, several performance metrics were presented that can provide a basis for a more straightforward comparison of different algorithms, including explicit and implicit codes, taking into account differing convergence tolerances.

The algorithm performs well when solving the flow around three-dimensional configurations. Both subsonic and transonic solutions were performed around the ONERA M6 wing and the CRM wing-body configuration. Grid convergence studies provided the solution error behavior of the code, along with an estimate of the weak scaling characteristics of the algorithm. All solutions were converged by reducing the residual 12 orders of magnitude.

A third geometry was studied, providing difficulty in full residual convergence. The high-lift delta wing geometry resulted in stalled convergence at angles of attack between 1° and 40°, with the residual typically leveling off at a reduction of 4 to 5 orders of magnitude. However, the stalled solutions still exhibit the correct trends in the lift and drag coefficient data, as compared to experimental data, signifying that the incomplete convergence of the solutions is not hampering the functional value accuracy.

Acknowledgments

The authors gratefully acknowledge financial assistance from the Natural Sciences and Engineering Research Council (NSERC), MITACS, Bombardier, the Canada Research Chairs program, and the University of Toronto.

Computations were performed on the GPC supercomputer at the SciNet HPC Consortium, part of Compute Canada. SciNet is funded by: the Canada Foundation for Innovation under the auspices of Compute

Canada; the Government of Ontario; Ontario Research Fund - Research Excellence; and the University of Toronto.

References

[1]Mavriplis, D. J., Vassberg, J. C., Tinoco, E. N., Mani, M., Brodersen, O. P., Eisfeld, B., Wahls, R. A., Morrison, J. H., Zickuhr, T., Levy, D., and Murayama, M., "Grid quality and resolution issues from the drag prediction workshop series," *46th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA–2008–0930, Reno, Nevada, Jan. 2008.

[2]Jespersen, D., Pulliam, T., and Buning, P., "Recent enhancements to OVERFLOW (Navier-Stokes code)," *35th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA–97–0644, Reno, Nevada, Jan. 1997.

[3]Nielsen, E. J., Walters, R. W., Anderson, W. K., and Keyes, D. E., "Application of Newton-Krylov methodology to a three-dimensional unstructured Euler code," *12th AIAA Computational Fluid Dynamics Conference*, AIAA–95–1733, San Diego, California, United States, June 1995.

[4]May, G. and Jameson, A., "Unstructured Algorithms for Inviscid and Viscous Flows Embedded in a Unified Solver Architecture: Flo3xx," *43rd AIAA Aerospace Sciences Meeting and Exhibit*, AIAA–2005–0318, Reno, Nevada, Jan. 2005.

[5]Mavriplis, D. J., "Grid Resolution of a Drag Prediction Workshop using the NSU3D Unstructured Mesh Solver," *17th AIAA Computational Fluid Dynamics Conference*, AIAA–2005–4729, Toronto, Canada, June 2005.

[6]Levy, D. W., Zickuhr, T., Vassberg, J. C., Agarwal, S., Wahls, R. A., Pirzadeh, S., and Hemsch, M. J., "Summary of Data from the First AIAA CFD Drag Prediction Workshop," AIAA–2002–0841, 2002.

[7]Laflin, K. R., Klausmeyer, S. M., Zickuhr, T., Vassberg, J. C., Wahls, R. A., Morrison, J. H., Brodersen, O. P., Rakowitz, M. E., Tinoco, E. N., and Godard, J.-L., "Data summary from second AIAA computational fluid dynamics drag prediction workshop," *Journal of Aircraft*, Vol. 42, No. 5, 2005, pp. 1165–1178.

[8]Vassberg, J. C., Tinoco, E. N., Mani, M., Brodersen, O. P., Eisfeld, B., Wahls, R. A., Morrison, J. H., Zickuhr, T., Laflin, K. R., and Mavriplis, D. J., "Abridged summary of the third AIAA computational fluid dynamics drag prediction workshop," *Journal of Aircraft*, Vol. 45, No. 3, 2008, pp. 781–798.

[9]Vassberg, J. C., Tinoco, E. N., Mani, M., Rider, B., Zickuhr, T., Levy, D. W., Brodersen, O. P., Eisfeld, B., Crippa, S., Wahls, R. A., Morrison, J. H., Mavriplis, D. J., and Murayama, M., "Summary of the Fourth AIAA CFD Drag Prediction Workshop," *28th AIAA Applied Aerodynamics Conference*, AIAA–2010–4547, Chicago, Illinois, United States, June 2010.

[10]Levy, D. W., Laflin, K. R., Tinoco, E. N., Vassberg, J. C., Mani, M., Rider, B., Rumsey, C. L., Wahls, R. A., Morrison, J. H., Brodersen, O. P., Crippa, S., Mavriplis, D. J., and Murayama, M., "Summary of Data from the Fifth AIAA CFD Drag Prediction Workshop," *51st AIAA Aerospace Sciences Meeting and Aerospace Exposition*, AIAA–2013–0046, Grapevine, Texas, United States, Jan. 2013.

[11]Baker, T. J., "Mesh generation: Art or science?" *Progress in Aerospace Sciences*, Vol. 41, No. 1, 2005, pp. 29–63.

[12]Osusky, M. and Zingg, D. W., "Parallel Newton-Krylov-Schur Flow Solver for the Navier-Stokes Equations," *AIAA Journal*, Vol. 51, No. 12, 2013, pp. 2833–2851.

[13]White, F. M., *Viscous Fluid Flow*, McGraw–Hill Book Company, New York, 1974.

[14]Wilcox, D. C., *Turbulence modeling for CFD*, Vol. 2, DCW industries La Canada, 1998.

[15]Spalart, P. R. and Allmaras, S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," *30th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA–92–0439, Reno, Nevada, United States, Jan. 1992.

[16]"NASA Turbulence Modeling Resource," http://turbmodels.larc.nasa.gov.

[17]Spalart, P. R., "Strategies for turbulence modeling and simulations," *International Journal of Heat and Fluid Flow*, Vol. 21, 2000, pp. 252–263.

[18]Yamamoto, K., Tanaka, K., , and Murayama, M., "Comparison study of drag prediction for the 4th CFD Drag Prediction Workshop using structured and unstructured mesh methods," *28th AIAA Applied Aerodynamic Conference*, AIAA–2010–4222, Chicago, Illinois, United States, 2010.

[19]Osusky, M., Boom, P. D., and Zingg, D. W., "Results from the Fifth AIAA Drag Prediction Workshop obtained with a parallel Newton-Krylov-Schur flow solver discretized using summation-by-parts operators," *31st AIAA Applied Aerodynamics Conference*, San Diego, California, United States, June 2013.

[20]Hicken, J. E. and Zingg, D. W., "A parallel Newton-Krylov solver for the Euler equations discretized using simultaneous approximation terms," *AIAA Journal*, Vol. 46, No. 11, Nov. 2008, pp. 2773–2786.

[21]Osusky, M., Boom, P. D., Del Rey Fernández, D. C., and Zingg, D. W., "An efficient Newton-Krylov-Schur parallel solution algorithm for the steady and unsteady Navier-Stokes equations," *7th International Conference on Computational Fluid Dynamics*, ICCFD7–1801, Big Island, Hawaii, USA, July 2012.

[22]Osusky, M. and Zingg, D. W., "A parallel Newton-Krylov-Schur flow solver for the Reynolds-Averaged Navier-Stokes equations," *50th AIAA Aerospace Sciences Meeting and Aerospace Exposition*, AIAA–2012–0442, Nashville, Tennessee, United States, Jan. 2012.

[23]Vassberg, J. C., DeHann, M. A., Rivers, S. M., and Wahls, R. A., "Development of a Common Research Model for Applied CFD Validation Studies," *26th AIAA Applied Aerodynamics Conference*, AIAA–2008–6919, Honolulu, Hawaii, United States, Aug. 2008.

[24]Jameson, A., Schmidt, W., and Turkel, E., "Numerical solution of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes," *14th Fluid and Plasma Dynamics Conference*, AIAA–81–1259, Palo Alto, California, United States, 1981.

[25]Pulliam, T. H., "Efficient solution methods for the Navier-Stokes equations," Tech. rep., Lecture Notes for the von Kármán Inst. for Fluid Dynamics Lecture Series: Numerical Techniques for Viscous Flow Computation in Turbomachinery Bladings, Rhode-Saint-Genèse, Belgium, Jan. 1986.

[26]Swanson, R. C. and Turkel, E., "On central-difference and upwind schemes," *Journal of Computational Physics*, Vol. 101, No. 2, 1992, pp. 292–306.

[27]Lomax, H., Pulliam, T. H., and Zingg, D. W., *Fundamentals of Computational Fluid Dynamics*, Springer–Verlag, Berlin, Germany, 2001.

[28]Saad, Y. and Sosonkina, M., "Distributed Schur complement techniques for general sparse linear systems," *SIAM Journal of Scientific Computing*, Vol. 21, No. 4, 1999, pp. 1337–1357.

[29]Boom, P. D. and Zingg, D. W., "Time-Accurate Flow Simulations Using an Efficient Newton-Krylov-Schur Approach and High-Order Temporal and Spatial Discretization," *51st AIAA Aerospace Sciences Meeting and Aerospace Exposition*, AIAA–2013–0383, Grapevine, Texas, United States, Jan. 2013.

[30]Apponsah, K. P. and Zingg, D. W., "A Load Balancing Tool for Structured Multi-Block Grid CFD Applications," *20th Annual Conference of the CFD Society of Canada*, Canmore, Alberta, Canada, May 2012.

[31]Osusky, M., Hicken, J. E., and Zingg, D. W., "A parallel Newton-Krylov-Schur flow solver for the Navier-Stokes equations using the SBP-SAT approach," *48th AIAA Aerospace Sciences Meeting and Aerospace Exposition*, AIAA–2010–116, Orlando, Florida, United States, Jan. 2010.

[32]Hicken, J. E., Osusky, M., and Zingg, D. W., "Comparison of parallel preconditioners for a Newton-Krylov flow solver," *6th International Conference on Computational Fluid Dynamics*, St. Petersburg, Russia, July 2010.

[33]Vassberg, J. C., "A Unified Baseline Grid about the Common Research Model Wing-Body for the Fifth AIAA CFD Drag Prediction Workshop," *29th AIAA Applied Aerodynamics Conference*, AIAA–2011–3508, Honolulu, Hawaii, United States, June 2011.

[34]Slotnick, J. P., Hannon, J. A., and Chaffin, M., "Overview of the First AIAA CFD High Lift Prediction Workshop," *49th AIAA Aerospace Sciences Meeting*, AIAA–2011–0862, Orlando, Florida, Jan. 2011.