A parallel Newton-Krylov flow solver for the Euler equations on multi-block grids

Jason E. Hicken* and David W. Zingg †

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

We present a parallel Newton-Krylov algorithm for solving the three-dimensional Euler equations on multi-block structured meshes. The Euler equations are discretized on each block independently using second-order accurate summation-by-parts operators and scalar numerical dissipation. Boundary conditions are imposed and block interfaces are coupled using simultaneous approximation terms (SATs). The resulting discrete equations are solved iteratively using an inexact Newton method. At each Newton iteration, the linear system is solved inexactly using a Krylov subspace iterative method, and both additive Schwarz and approximate Schur preconditioners are considered. The algorithm is tested on the ONERA M6 wing. The results show that a discretization based on SATs is well suited to a parallel Newton-Krylov solution strategy, and that the approximate Schur preconditioner is more efficient than the Schwarz preconditioner in terms of CPU time and Krylov iterations, for both flow and adjoint solves.

I. Introduction

State-of-the-art flow solvers are capable of finding accurate solutions for flows with moderate separation;[1] however, run times remain an issue for three-dimensional configurations, especially for applications such as unsteady flows and optimization. This motivates the development of efficient parallel flow solvers. While serial solvers will continue to have a role in, for example, preliminary design, improvements in parallel computing architectures and libraries fuel interest in ever more complex large-scale problems. In this paper we describe an algorithm to tackle such problems; the algorithm combines a technique for handling block interfaces and boundaries (simultaneous approximation terms) and a solution strategy (Newton-Krylov) which together produce an efficient parallel solver.

Our choice of grid type and discretization is based on experience with serial two- and three-dimensional flow solvers.[2–4] The semi-structured, or multi-block, approach provides the flexibility to fit complex shapes while permitting accurate and efficient discretizations. In particular, multi-block finite-difference discretizations can be readily extended to high-order schemes.

The use of multi-block finite-difference schemes in parallel does present some challenges. In particular, how do we discretize the equations of motion at points on the edges and corners of blocks? The treatment of these "exceptional" points can be difficult, even for serial finite-difference solvers. In parallel computations the exceptional points can introduce additional communication overhead if not treated carefully.

The problems associated with exceptional points can be avoided with simultaneous approximation terms (SATs). The SAT methodology was originally developed to enforce boundary conditions in an accurate and time-stable manner,[5] but the method has also been extended to handle domain interfaces.[6–8] SATs have been successfully used by Mattsson *et al.*[9] for 3rd- and 5th-order discretizations of the Euler equations. In the present work we use a second-order discretization, so our primary interest in SATs is the elegant parallelization they provide; however, future work will consider high-order discretizations.

Coupling blocks with SAT penalties has been shown to reduce the effective CFL number significantly.[10,11] This suggests that a Newton-Krylov solution strategy may be well suited to SAT discretizations. For serial

*PhD Candidate, AIAA Student Member

†Professor and Director, Tier 1 Canada Research Chair in Computational Aerodynamics, Associate Fellow AIAA

computations, Newton-Krylov solution strategies have proven to be efficient, both in flow simulation[4,12–17] and optimization.[3,18] There are also many examples demonstrating that the excellent serial performance of Newton-Krylov algorithms can be extended to parallel algorithms.[19–23]

Krylov solvers are easily parallelizable with the possible exception of the preconditioner. The preconditioner poses a difficulty, because many of the best serial preconditioners tend to be inefficient when parallelized directly — consider, for example, incomplete lower upper factorizations[24] — and many parallel preconditioners tend to scale poorly. In the current work we consider an additive Schwarz preconditioner[24] and an approximately factored Schur complement preconditioner.[25]

In summary, the objectives of this work are as follows:

- to develop SAT penalty terms suitable for flows with shocks and vanishing wave speeds,
- to evaluate and compare two parallel preconditioners, and
- to demonstrate that a discretization employing SATs is well matched to a parallel Newton-Krylov solution algorithm.

The paper is divided as follows. Section II focuses on the governing equations and their discretization. In particular, we summarize the use of SATs for one-dimensional problems. In Section III we review the Newton-Krylov method for solving the nonlinear equations and resulting distributed linear systems. Results are presented in Section IV, including an assessment of the discretization and a comparison of the parallel preconditioners applied to flow and adjoint problems. Conclusions can be found in Section V.

II. Governing Equations and Discretization

A. The Transformed Euler Equations

We consider the three-dimensional Euler equations:

$$\partial_t \mathbf{Q} + \partial_{x_i} \mathbf{E}_i = \mathbf{0}, \tag{1}$$

where $(x_1, x_2, x_3) = (x, y, z)$,

$$\mathbf{Q} = \begin{pmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ e \end{pmatrix}, \qquad \text{and} \qquad \mathbf{E}_i = \begin{pmatrix} \rho u_i \\ \rho u_1 u_i + p\delta_{1i} \\ \rho u_2 u_i + p\delta_{2i} \\ \rho u_2 u_i + p\delta_{3i} \\ (e+p)u_i \end{pmatrix}.$$

δ_{ij} is the Kronecker delta symbol. Our objective is to solve for the conservative flow variables, $\mathbf{Q}$, on multi-block structured grids; thus, we consider diffeomorphisms of the following form.

$$\begin{gathered} \mathbf{x} = \mathbf{F}(\boldsymbol{\xi}), \\ \mathbf{F} : D \to P \quad \text{and} \quad \mathbf{F} \in C^1 \text{ on } D \setminus \partial D, \\ D = \left\{ \boldsymbol{\xi} \in \mathbb{R}^3 | \xi_i \in [0, L_i], i = 1, 2, 3 \right\}. \end{gathered}$$

The hexahedral domain D represents one block in computational space. Applying the diffeomorphism, the Euler equations become

$$\partial_t \hat{\mathbf{Q}} + \partial_{\xi_i} \hat{\mathbf{E}}_i = \mathbf{0}, \tag{2}$$

where $(\xi_1, \xi_2, \xi_3) = (\xi, \eta, \zeta)$,

$$\hat{\mathbf{Q}} = \frac{1}{J} \begin{pmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ e \end{pmatrix}, \qquad \text{and} \qquad \hat{\mathbf{E}}_i = \frac{1}{J} \begin{pmatrix} \rho U_i \\ \rho u_1 U_i + p\partial_x \xi_i \\ \rho u_2 U_i + p\partial_y \xi_i \\ \rho u_2 U_i + p\partial_z \xi_i \\ (e+p)U_i \end{pmatrix}.$$

The scalar J denotes the Jacobian of the mapping, and the U_i are the contravariant velocities defined by $U_i = u_j \partial_{x_j} \xi_i$.

The discretization of the transformed Euler equations (2) is performed independently on each computational block using second-order centered differences in the interior and first-order one-sided differences at the block sides. Boundary conditions are introduced and blocks are coupled using simultaneous approximation terms (SATs).[5] Section II.B introduces the SAT methodology in a one-dimensional setting; the results can be extended to three-dimensions using Kronecker products; see, for example, Nordström and Carpenter.[8]

B. Discretization for One-Dimensional Problems

In an effort to keep our presentation self-contained, this section summarizes summation-by-parts (SBP) operators and SATs applied to one-dimensional problems, specifically constant-coefficient advection and the quasi-one-dimensional Euler equations. For further information on SBP operators and SATs, we direct the reader to the literature on these topics.[5,7–9,26]

1. Summation-By-Parts Operators

In both the boundary and interface cases, the SAT method relies on difference operators that satisfy a discrete SBP property. The difference operator $\mathcal{P}^{-1}\mathcal{Q}$ is an SBP operator if it satisfies the following three properties:[27]

1. The vector of discrete derivatives, $\mathbf{u}_x$, has the form

$$\mathcal{P}\mathbf{u}_x - \mathcal{Q}\mathbf{u} = \mathbf{0},$$
$$\mathcal{P}\partial_x\mathbf{v} - \mathcal{Q}\mathbf{v} = \mathcal{P}\mathbf{T},$$

   where $\mathbf{u}$ is the discrete solution vector, $\mathbf{v}$ is the exact solution evaluated at the node locations, and $\mathbf{T}$ is the truncation error. For an m^{th} order operator, the truncation error satisfies $\|\mathbf{T}\| = \mathrm{O}(\Delta x_{\max}^m)$. This form of discrete first derivative encompasses both compact and non-compact schemes.

2. The matrix $\mathcal{P}$ is symmetric positive definite. In particular,

$$c\Delta x \leq \lambda(\mathcal{P}) \leq C\Delta x$$

   where $\lambda(\mathcal{P})$ denotes an eigenvalue of $\mathcal{P}$, and c, C are real, positive constants independent of the number of grid points.

3. The matrix $\mathcal{Q}$ is nearly skew-symmetric in the sense that $\mathcal{Q} + \mathcal{Q}^T = \mathcal{D}$, where $\mathcal{D}$ is a diagonal matrix of the form $\mathcal{D} = \mathrm{diag}(-1, 0, \ldots, 0, 1)$. Furthermore, $Q_{0,0} = -\frac{1}{2}$ and $Q_{N,N} = \frac{1}{2}$.

The SBP definition encompasses operators of all orders; however, for the present work we consider only second-order operators. Note, the standard second-order centred-difference operator with first-order differences at the boundaries is an SBP operator if we define

$$\mathcal{Q} = \frac{1}{2}\begin{bmatrix} -1 & 1 & & & & \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 0 & 1 \\ & & & & -1 & 1 \end{bmatrix}, \qquad \mathcal{P} = \begin{bmatrix} \Delta x_0 & & & \\ & \Delta x_1 & & \\ & & \ddots & \\ & & & \Delta x_N \end{bmatrix},$$

where $\Delta x_0 = (x_1 - x_0)/2$, $\Delta x_N = (x_N - x_{N-1})/2$, and $\Delta x_i = (x_{i+1} - x_{i-1})/2, i = 2, 3, \ldots, N-1$.

2. Constant-Coefficient Advection

The SAT method adds a penalty term that forces the solution at the boundary or interface towards the desired value. To introduce the method, we consider a constant-coefficient one-dimensional linear advection problem

$$\partial_t u + a\partial_x u = 0, \tag{3}$$

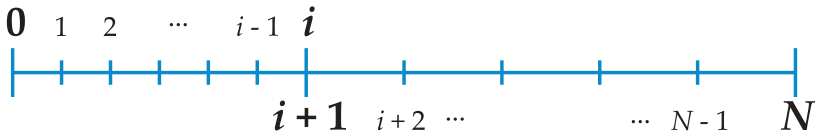


Figure 1. Example domain consisting of two subdomains with an interface at $x_i = x_{i+1}$.

where a is the advection speed. Equation (3) is discretized on the mesh $\mathbf{x} = (x_0, x_1, \ldots, x_i, x_{i+1}, \ldots, x_N)^T$ with $N+1$ points and an interface at $x_i = x_{i+1}$; see Figure 1. For simplicity, we will apply the SAT at the interface only; the application of SATs for the boundary conditions is similar.[5]

Let the discrete solutions on the left and right domains be $\mathbf{u}_L = (u_0, u_1, \ldots, u_i)^T$ and $\mathbf{u}_R = (u_{i+1}, u_{i+2}, \ldots, u_N)^T$, respectively. The systems to be solved on each domain are

$$\mathcal{P}_L \partial_t \mathbf{u}_L + a\mathcal{Q}_L \mathbf{u}_L = \sigma_1 (u_i - u_{i+1}) \mathbf{e}_L, \tag{4}$$

$$\mathcal{P}_R \partial_t \mathbf{u}_R + a\mathcal{Q}_R \mathbf{u}_R = \sigma_2 (u_{i+1} - u_i) \mathbf{e}_R. \tag{5}$$

Note that the unit vectors $\mathbf{e}_L = (0, 0, \ldots, 1)^T$ and $\mathbf{e}_R = (1, 0, \ldots, 0)^T$ have lengths $(i+1)$ and $(N-i)$, respectively.

The terms appearing on the right sides of (4) and (5) are the penalty terms that couple the two domains. For smooth solutions they introduce no truncation errors since the points coincide. As an aside, if we were applying a boundary condition at x_i instead of an interface condition, u_{i+1} would be replaced with u_{BC} in equation (4), where u_{BC} is the appropriate boundary value of u.

The coefficients σ_1 and σ_2 are determined using time-stability and conservation arguments. First, consider conservation. Premultiplying (4) by the constant vector $\mathbf{1} = (1, 1, \ldots, 1)^T$, and ignoring boundary contributions, we obtain

$$\begin{aligned}
\mathbf{1}^T \mathcal{P}_L \partial_t \mathbf{u}_L + a\mathbf{1}^T \mathcal{Q}_L \mathbf{u}_L &= \sigma_1 (u_i - u_{i+1}) \mathbf{1}^T \mathbf{e}_{(i)} \\
\frac{d}{dt}(\mathbf{1}^T \mathcal{P}_L \mathbf{u}_L) + a\mathbf{1}^T (\mathcal{D}_L - \mathcal{Q}_L^T) \mathbf{u}_L &= \sigma_1 (u_i - u_{i+1}) \\
\frac{d}{dt}(\mathbf{1}^T \mathcal{P}_L \mathbf{u}_L) - a\mathbf{u}_L^T \mathcal{Q}_L \mathbf{1} &= \sigma_1 (u_i - u_{i+1}) - a u_i \\
\frac{d}{dt}(\mathbf{1}^T \mathcal{P}_L \mathbf{u}_L) &= \sigma_1 (u_i - u_{i+1}) - a u_i
\end{aligned}$$

Note that $\mathbf{u}_L^T \mathcal{Q}_L \mathbf{1} = 0$, since the constant vector is in the null space of $\mathcal{Q}_L$. Adding a similar expression for the right domain we find

$$\frac{d}{dt}(\mathbf{1}^T \mathcal{P}_L \mathbf{u}_L + \mathbf{1}^T \mathcal{P}_R \mathbf{u}_R) = (u_i - u_{i+1})(\sigma_1 - \sigma_2 - a). \tag{6}$$

We want the right-hand side of (6) to vanish for conservation. In general, u_i and u_{i+1} will not have the same value. Thus, to ensure conservation on the whole domain, the expression (6) implies

$$\sigma_2 = \sigma_1 - a. \tag{7}$$

The relation (7) has also been shown to hold for nonlinear fluxes using a weak form of conservation.[7]

To further fix the penalty parameters, a time-stability requirement is imposed. Consider the time evolution of the norm $\|\mathbf{u}\|_{\mathcal{P}}^2 = \mathbf{u}_L^T \mathcal{P}_L \mathbf{u}_L + \mathbf{u}_R^T \mathcal{P}_R \mathbf{u}_R$. Premultiplying equation (4) from the left by $\mathbf{u}_L^T$, and adding the transpose, we find

$$\begin{aligned}
\frac{d}{dt}(\mathbf{u}_L^T \mathcal{P}_L \mathbf{u}_L) + a\mathbf{u}_L^T (\mathcal{Q}_L + \mathcal{Q}_L^T) \mathbf{u}_L &= 2\sigma_1 (u_i^2 - u_i u_{i+1}) \\
\frac{d}{dt}(\mathbf{u}_L^T \mathcal{P}_L \mathbf{u}_L) + a\mathbf{u}_L^T \mathcal{D}_L \mathbf{u}_L &= 2\sigma_1 (u_i^2 - u_i u_{i+1}) \\
\frac{d}{dt}(\mathbf{u}_L^T \mathcal{P}_L \mathbf{u}_L) &= 2\sigma_1 (u_i^2 - u_i u_{i+1}) - a u_i^2
\end{aligned}$$

Adding a similar expression for the right domain, we obtain the following "energy" estimate:

$$\frac{d}{dt}\|u\|_{\mathcal{P}}^2 = \begin{pmatrix} u_i & u_{i+1} \end{pmatrix} \begin{bmatrix} 2\sigma_1 - a & -(\sigma_1+\sigma_2) \\ -(\sigma_1+\sigma_2) & 2\sigma_2 + a \end{bmatrix} \begin{pmatrix} u_i \\ u_{i+1} \end{pmatrix}. \tag{8}$$

The eigenvalues of the symmetric matrix on the right-hand-side of (8) are $\lambda_1 = 0$ and $\lambda_2 = -2a + 4\sigma_1$. To ensure that the norm does not grow with time, we need $\lambda_2 \leq 0$ which implies $\sigma_1 \leq \frac{a}{2}$. To satisfy this stability requirement, as well as the conservation requirement (7), we tentatively adopt the following values for σ_1 and σ_2, although other choices are possible:

$$\begin{aligned} \sigma_1 &= 0, \quad \sigma_2 = -a, \quad \text{if} \quad a \geq 0, \\ \sigma_1 &= a, \quad \sigma_2 = 0, \quad \text{if} \quad a < 0. \end{aligned} \tag{9}$$

If we use (9) without modification, both penalty parameters vanish if $a = 0$. This can cause problems for Euler flows if the interface is located at a sonic or stagnation point; thus, in analogy with Swanson and Turkel's matrix dissipation model,[28] we propose limiting the penalty parameters as follows:

$$\begin{aligned} \sigma_1 &= -\frac{1}{2}\left[\max\left(|a|, V\right) - a\right], \\ \sigma_2 &= -\frac{1}{2}\left[\max\left(|a|, V\right) + a\right], \end{aligned} \tag{10}$$

where $V > 0$ is a constant. These values of the penalty parameters satisfy the conservation requirement (7). What about time-stability? If $|a| \geq V$, then $\lambda_2 = -2|a| < 0$ as required. Moreover, if $|a| < V$ then $\lambda_2 = -2V < 0$, so the penalty parameters defined by (10) ensure stability. Notice that the SAT penalties are activated depending on the direction of wave propagation, which gives this boundary treatment a characteristic variable "flavour."

3. The Quasi-One-Dimensional Euler Equations

In this section, we review the use of SBP operators and SATs in discretizing the Euler equations. Consider the (transformed) quasi-one-dimensional Euler equations applied to a converging-diverging nozzle:

$$\partial_t \hat{\mathbf{Q}} + \partial_\xi \hat{\mathbf{E}} - \hat{\mathbf{G}} = \mathbf{0}, \tag{11}$$

where

$$\hat{\mathbf{Q}} = \frac{1}{J}\begin{pmatrix} \rho S \\ \rho u S \\ \rho E S \end{pmatrix}, \qquad \hat{\mathbf{E}} = \frac{1}{J}\begin{pmatrix} \xi_x \rho u S \\ \xi_x(\rho u^2 + p) S \\ \xi_x \rho u H S \end{pmatrix}, \qquad \hat{\mathbf{G}} = \frac{1}{J}\begin{pmatrix} 0 \\ p \partial_x S \\ 0 \end{pmatrix},$$

$J = \xi_x = (x_\xi)^{-1}$ is the metric Jacobian, and S is the nozzle area.

Suppose the one-dimensional domain is divided into two subdomains. As before, assume the grid points are located at $\mathbf{x} = (x_0, x_1, \ldots, x_i, x_{i+1}, \ldots, x_N)^T$, and let $x_i = x_{i+1}$ define the interface between the two subdomains. The semi-discrete form of (11) becomes

$$\begin{aligned} \mathcal{P}_L \partial_t \mathbf{q}_L + \mathcal{Q}_L \mathbf{f}_L - \mathcal{P}_L \mathbf{g}_L &= \mathbf{s}_L, \\ \mathcal{P}_R \partial_t \mathbf{q}_R + \mathcal{Q}_R \mathbf{f}_R - \mathcal{P}_R \mathbf{g}_R &= \mathbf{s}_R, \end{aligned} \tag{12}$$

where

$$\begin{aligned} \mathbf{q}_L &= (\hat{\mathbf{Q}}_0^T, \hat{\mathbf{Q}}_1^T, \ldots, \hat{\mathbf{Q}}_i^T)^T, & \mathbf{q}_R &= (\hat{\mathbf{Q}}_{i+1}^T, \hat{\mathbf{Q}}_{i+2}^T, \ldots, \hat{\mathbf{Q}}_N^T)^T, \\ \mathbf{f}_L &= (\hat{\mathbf{E}}_0^T, \hat{\mathbf{E}}_1^T, \ldots, \hat{\mathbf{E}}_i^T)^T, & \mathbf{f}_R &= (\hat{\mathbf{E}}_{i+1}^T, \hat{\mathbf{E}}_{i+2}^T, \ldots, \hat{\mathbf{E}}_N^T)^T, \\ \mathbf{g}_L &= (\hat{\mathbf{G}}_0^T, \hat{\mathbf{G}}_1^T, \ldots, \hat{\mathbf{G}}_i^T)^T, & \mathbf{g}_R &= (\hat{\mathbf{G}}_{i+1}^T, \hat{\mathbf{G}}_{i+2}^T, \ldots, \hat{\mathbf{G}}_N^T)^T \end{aligned}$$

The subscript denotes the location of the variable or flux (e.g. $\hat{\mathbf{Q}}_i = \hat{\mathbf{Q}}(x_i)$).

The penalty terms in (12) are $3(i+1)$ and $3(N-i)$ column vectors given by

$$\mathbf{s}_L = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ -\frac{1}{2}(|A| - A)(\hat{\mathbf{Q}}_i - \hat{\mathbf{Q}}_{i+1}) \end{pmatrix}, \qquad \mathbf{s}_R = \begin{pmatrix} -\frac{1}{2}(|A| + A)(\hat{\mathbf{Q}}_{i+1} - \hat{\mathbf{Q}}_i) \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix}$$

where A is the flux Jacobian matrix at an averaged state; for the present work we use the simple average $\frac{1}{2}(\mathbf{Q}_i + \mathbf{Q}_{i+1})$. The matrix $|A| = X|\Lambda|X^{-1}$ where X denotes the right eigenvectors of A, and

$$|\Lambda| = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

where

$$\begin{aligned} \lambda_1 &= \max\left(|u + a|, V_n \rho(A)\right) \\ \lambda_2 &= \max\left(|u - a|, V_n \rho(A)\right), \\ \lambda_3 &= \max\left(|u|, V_l \rho(A)\right). \end{aligned}$$

$\rho(A)$ denotes the spectral radius of A. The constants V_n and V_l are used to scale the spectral radius. For subsonic flows $V_n = 0.025$ and for transonic flows $V_n = 0.25$. $V_l = 0.025$ is used for all flows.

We have omitted numerical dissipation in (12) to focus on the SATs. In the present work we use the JST scalar dissipation model,[29,30] with second- and fourth-difference dissipation. Mattsson *et al.*[9] have shown that the dissipation based on the standard difference operators does not produce an energy estimate. Their result does not imply that the standard dissipation operators will always produce time-unstable schemes, but that conclusions regarding time-stability must be carried out on a grid-by-grid basis.[9] Indeed, we have not experienced problems when using the standard dissipation operators in conjunction with SATs *for steady-state flows* solved using implicit methods; however, unsteady simulations may require us to revisit the dissipation operators.

III. Solution Method

A. Newton-Krylov Approach

Discretizing the transformed, steady Euler equations at each computational node produces a set of nonlinear algebraic equations, represented by the vector equation

$$\boldsymbol{\mathcal{F}}(\mathbf{q}) = \mathbf{0}, \tag{13}$$

where $\mathbf{q}$ is a block column vector; each block represents the conservative flow variables, $\mathbf{Q}$, at a node.

Applying Newton's method to the discrete equations (13) we obtain the following linear equation for each (outer) iteration n:

$$\mathcal{A}^{(n)} \boldsymbol{\Delta}\mathbf{q}^{(n)} = -\boldsymbol{\mathcal{F}}^{(n)}, \tag{14}$$

where $\boldsymbol{\mathcal{F}}^{(n)} = \boldsymbol{\mathcal{F}}(\mathbf{q}^n)$, $\boldsymbol{\Delta}\mathbf{q}^{(n)} = \mathbf{q}^{(n+1)} - \mathbf{q}^{(n)}$, and

$$\mathcal{A}_{ij}^{(n)} = \frac{\partial \mathcal{F}_i}{\partial q_j}\left(\mathbf{q}^{(n)}\right).$$

Newton's method will converge quadratically provided that the initial iterate, $\mathbf{q}^{(0)}$, is sufficiently close to the solution of (13).[31] As is well known, finding a suitable initial iterate for Newton's method can be difficult; thus, our algorithm is broken into two phases:

1. an approximate-Newton, start-up phase whose objective is to find a suitable initial iterate, as efficiently as possible, and;
2. the inexact-Newton phase, which uses the initial iterate and a slightly modified form of equation (14).

1. Approximate-Newton Phase

The approximate-Newton phase uses a form of pseudo-transient continuation to find the initial iterate.[23,30,32] This strategy is similar to discretizing the unsteady equations (1) using implicit Euler in time; however, since we are seeking an initial iterate for Newton's method, and not a time accurate solution, there are several important modifications that we can make to the implicit Euler scheme. These modifications include a first-order Jacobian matrix, a lagged Jacobian update, and a spatially varying time step.

A first-order Jacobian matrix can be effective during start-up,[4,32] and is obtained here by eliminating the fourth-difference dissipation terms from $\mathcal{A}^{(n)}$, and increasing the coefficient for the second-difference dissipation. Let κ_4 and κ_2 denote the fourth- and second-difference dissipation coefficients used in the discrete equations, $\boldsymbol{\mathcal{F}}$, and let $\tilde{\kappa}_4$ and $\tilde{\kappa}_2$ denote the corresponding coefficients used in the modified Jacobian matrix during start-up. Then,

$$\tilde{\kappa}_4 = 0, \qquad \tilde{\kappa}_2 = \kappa_2 + \sigma\kappa_4$$

Previous work suggests that the optimal value for the lumping coefficient σ is between 4 and 6 for three-dimensional inviscid flows.[4] Lumping the dissipation coefficients in this way produces a modified Jacobian which is only first-order accurate: this does not affect the accuracy of the steady solution. We will use $\mathcal{A}_1$ to denote the first-order Jacobian, in order to emphasize its accuracy and distinguish it from the exact Jacobian.

The first-order Jacobian is factored using ILU (or block ILU) with a level of fill p to produce the pre-conditioners *during both phases*. Factoring the matrix is one of the most expensive tasks required by the algorithm. The approximate-Newton phase often requires many outer iterations, so the cost of the factorization can be particularly acute if it is performed each iteration. This suggests periodically updating the first-order Jacobian and, therefore, the factorization.[33] Let m be the number of outer iterations between updates. Then, $\mathcal{A}_1$ is updated and factored on iteration n if $\mathrm{mod}(n, m) = 0$. In Section IV we will investigate the influence of the update period m on efficiency.

Finally, a spatially varying time step has been shown to improve the rate of convergence for schemes based on approximate factorizations[30] as well as Newton-Krylov algorithms.[4,34] The time step used in the current work is, for node (j, k, m),

$$\Delta t^{(n)}_{j,k,m} = \frac{\Delta t^{(n)}_{\mathrm{ref}}}{J_{j,k,m}(1 + \sqrt[3]{J_{j,k,m}})}. \tag{15}$$

This time step roughly approximates a constant CFL. The appearance of the first J in the denominator of (15) is due to the use of $\mathbf{Q}$ rather than $\hat{\mathbf{Q}}$ in the column vector $\mathbf{q}$; see equation (2). The reference time step is steadily increased according to the geometric formula

$$\Delta t^{(n)}_{\mathrm{ref}} = a(b)^{m\lfloor \frac{n}{m} \rfloor},$$

where $\lfloor \cdot \rfloor$ is the floor operator[a]; this operator ensures that updates to $\Delta t^{(n)}_{\mathrm{ref}}$ are consistent with the update period m. Typical values for a and b in the present work are $a = 0.1$ and $b \in [1.2, 1.7]$.

To summarize, during start-up we replace (14) with the approximate-Newton update equation

$$\tilde{\mathcal{A}}^{(n)} \boldsymbol{\Delta}\mathbf{q}^{(n)} = -\boldsymbol{\mathcal{F}}^{(n)}, \tag{16}$$

with

$$\tilde{\mathcal{A}}^{(n)} \equiv \mathcal{T}^{(n^*)} + \mathcal{A}_1^{(n^*)},$$

where $n^* = m\lfloor n/m \rfloor$ and $\mathcal{T}^{(n)}$ is a diagonal matrix containing the (inverse) local time steps appropriate to each equation. Finally, we emphasize that the update equation (16) is not solved exactly, but rather inexactly to a relative tolerance of 0.5 using a Krylov iterative solver; the solution of the linear system is outlined in Section III.B.

[a] $\lfloor x \rfloor$ gives the largest integer less than or equal to x

2. Switching Between Phases

The algorithm should switch to the inexact-Newton phase as soon as a suitable initial (Newton) iterate has been obtained; thus, we must determine what qualifies as a suitable initial "guess." Several authors have suggested switching when the nonlinear relative residual is reduced below a certain threshold.[4,34] For the present discretization using SATs, we have found that the residual by itself is a poor measure for convergence during start-up: the nodes on interfaces and boundaries typically have much larger residuals than the interior nodes. Hence, our algorithm switches to the inexact-Newton phase if

$$\frac{\|\boldsymbol{\mathcal{F}}^{(n)}\|_2}{\|\boldsymbol{\mathcal{F}}^{(0)}\|_2} \equiv R_d^{(n)} \leq \tau \qquad \text{and} \qquad \frac{\|\boldsymbol{\Delta}\mathbf{q}^{(n)}\|_2}{\|\boldsymbol{\Delta}\mathbf{q}^{(m)}\|_2} \equiv S_d^{(n)} \leq \tau$$

where $\tau = 0.1$. The additional condition on $\boldsymbol{\Delta}\mathbf{q}^{(n)}$ safe-guards against switching to the inexact-Newton phase prematurely. Notice that $\|\boldsymbol{\Delta}\mathbf{q}^{(n)}\|_2$ is measured relative to the norm of the solution update after the first Jacobian update period; this is because the updates produced during the initial period fluctuate considerably.

3. Inexact-Newton Phase

As with the start-up phase, a diagonal matrix of spatially varying time-steps is used during the inexact-Newton phase. The reference time step during the Newton phase is based on a method first used by Mulder and van Leer:[35]

$$\Delta t_{ref}^{(n)} = \max\left[\alpha\left(R_d^{(n)}\right)^{-\beta}, \Delta t_{\text{ref}}^{(n-1)}\right],$$

where, for the present results, we have used $\beta = 2$. The value of α is calculated to avoid an abrupt change between the approximate Newton and inexact-Newton time steps. Specifically, if n_{Newt} is the first inexact-Newton iteration, then

$$\alpha = a(b)^{m\lfloor\frac{n_{\text{Newt}}}{m}\rfloor}\left(R_d^{(n_{\text{Newt}})}\right)^{\beta}$$

Each outer iteration during the inexact-Newton phase produces the following linear system:

$$\left(\mathcal{T}^{(n)} + \mathcal{A}^{(n)}\right)\boldsymbol{\Delta}\mathbf{q}^{(n)} = -\boldsymbol{\mathcal{F}}^{(n)}, \tag{17}$$

where, as before, $\mathcal{T}^{(n)}$ is a diagonal matrix of inverse time steps. Note that the diagonal time step matrix tends to the zero matrix quadratically with $R_d^{(n)}$ due to the value $\beta = 2$. Unlike the approximate Newton phase, the matrices on the left-hand side of (17) are recomputed at each iteration, and the Jacobian matrix, $\mathcal{A}^{(n)}$, is not explicitly modified. Indeed, since we use Krylov subspace methods, we need only the Jacobian-vector products. These products can be approximated using a first-order accurate, forward difference:

$$\mathcal{A}^{(n)}\mathbf{v} \approx \frac{\boldsymbol{\mathcal{F}}(\mathbf{q}^n + \epsilon\mathbf{v}) - \boldsymbol{\mathcal{F}}(\mathbf{q}^n)}{\epsilon}.$$

The perturbation parameter must be chosen carefully to minimize truncation error and avoid round-off errors.[36] For this work we have used[13]

$$\epsilon = \sqrt{\frac{N\delta}{\mathbf{v}^T\mathbf{v}}},$$

where $\delta = 10^{-13}$ and N is the number of unknowns.

The inexact-Newton algorithm does not solve equation (17) exactly, but rather to a certain relative tolerance:

$$\left\|\boldsymbol{\mathcal{F}}^{(n)} - \left(\mathcal{T}^{(n)} + \mathcal{A}^{(n)}\right)\boldsymbol{\Delta}\mathbf{q}^{(n)}\right\|_2 \leq \eta_n\|\boldsymbol{\mathcal{F}}^{(n)}\|_2.$$

The forcing parameter $\eta_n \in [0, 1)$ controls the accuracy of solution update $\boldsymbol{\Delta}\mathbf{q}^{(n)}$ and the convergence rate of the inexact-Newton method. If η_n is too small, we obtain quadratic convergence at the expense of oversolving the linear system. If η_n is too large, the linear system will be cheap to solve, but outer iterations will not converge quadratically.

For this work we use the following formula for the forcing parameter, originally proposed by Eisenstat and Walker:[37]

$$\eta_n = \frac{\left| \|\boldsymbol{\mathcal{F}}^{(n)}\|_2 - \|\boldsymbol{\mathcal{F}}^{(n-1)} - \left(\mathcal{T}^{(n-1)} + \mathcal{A}^{(n-1)}\right) \boldsymbol{\Delta}\mathbf{q}^{(n-1)}\| \right|}{\|\boldsymbol{\mathcal{F}}^{(n-1)}\|_2}, \tag{18}$$

together with the safeguard

$$\eta_n \leftarrow \min\left\{0.5, \max\left\{\eta_n, \eta_{n-1}^{(1+\sqrt{5})/2}\right\}\right\}, \qquad \text{whenever } \eta_{n-1}^{(1+\sqrt{5})/2} > 0.1$$

With this choice of forcing parameter, the inexact-Newton method can be shown to converge q-superlinearly.[37] We limit the number of Krylov iterations to between 60 and 80 per iteration, so the asymptotic convergence of our algorithm is often linear; nevertheless, we have found that formula (18) is very useful, because it avoids oversolving during the early iterations of the inexact-Newton phase.

B. Solving The Distributed Linear System

During both the start-up and Newton phases we use a Krylov solver — for example, the generalized minimal residual method (GMRES)[38] — to inexactly solve sparse systems of the form

$$A\mathbf{x} = \mathbf{b}. \tag{19}$$

To solve the above equation in parallel, the unknowns, $\mathbf{x}$, and their corresponding equations are assigned to unique processes according to some domain decomposition; for the present work each block is assigned to a process. For a given process, i, three types of unknowns can be identified for the linear system (19):

1. internal unknowns which appear only in equations on the process i;
2. internal-interface unknowns which are assigned to process i but coupled to unknowns on another process $j \neq i$, and;
3. external-interface unknowns assigned to other subdomains but appearing in equations on process i.

For example, when the Euler equations are discretized using SATs, the internal- and external-interface unknowns correspond to nodes which are coincident for adjacent blocks.

If the unknowns and equations are grouped, i.e. ordered, by subdomain, then we can write the equations corresponding to process i as

$$A_i\mathbf{x}_{(i)} + E_i\mathbf{y}_{(i,\text{ext})} = \mathbf{b}_{(i)}, \tag{20}$$

where $\mathbf{x}_{(i)}$ and $\mathbf{b}_{(i)}$ denote the unknowns and right-hand-sides assigned to process i, and $\mathbf{y}_{(i,\text{ext})}$ are the external-interface unknowns coupled with unknowns on process i. With this grouping, the global linear system for four subdomains has the structure shown in Figure 2. Note that internal-interface unknowns are ordered last in each subdomain. This convention allows more efficient interprocessor communication, and reduced local indirect addressing during matrix-vector multiplication.[25]

With the internal-interface unknowns ordered last, we obtain the (local) partitioning

$$\mathbf{x}_{(i)} = \begin{pmatrix} \mathbf{u}_{(i)} \\ \mathbf{y}_{(i)} \end{pmatrix}, \qquad \mathbf{b}_{(i)} = \begin{pmatrix} \mathbf{f}_{(i)} \\ \mathbf{g}_{(i)} \end{pmatrix},$$

where $\mathbf{u}_{(i)}$ are the local internal variables, and $\mathbf{y}_{(i)}$ are the local internal-interface variables. The subvectors $\mathbf{f}_{(i)}$ and $\mathbf{g}_{(i)}$ are the analogous partitions of $\mathbf{b}_{(i)}$. Hence, the local equations (20) on process i take the form

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} \mathbf{u}_{(i)} \\ \mathbf{y}_{(i)} \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j\in N_i} E_{ij}\mathbf{y}_{(j)} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{(i)} \\ \mathbf{g}_{(i)} \end{pmatrix}. \tag{21}$$

The neighbouring subdomains of subdomain i are denoted by the set N_i.

In using a Krylov-subspace method to solve the distributed system (19), we must parallelize the inner-products, the matrix-vector products, and the preconditioner. The inner-products are straightforward: they are computed by summing the local products, e.g. $\mathbf{v}_{(i)}^T\mathbf{z}_{(i)}$, using the MPI command `MPI_Allreduce()`. Parallelizing the matrix-vector products and the preconditioner requires more care.

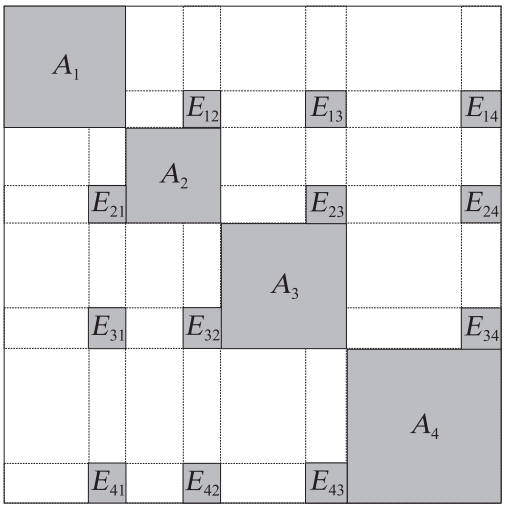


Figure 2. The sparsity structure of the global system matrix with unknowns grouped by subdomain, and internal-interface variables listed last. Sparse submatrices with non-zero elements are shaded.

For the matrix-vector products, we see that only internal-interface unknowns are affected by the product $E_i\mathbf{y}_{(i,\text{ext})} = \sum_{j\in N_i} E_{ij}\mathbf{y}_{(j)}$. Thus, communication time can be partially hidden by using a non-blocking communication of the interface variables.[39] During the communication the local matrix is multiplied, and, once the external-interface unknowns are received, the contribution due to $E_i\mathbf{y}_{(i,\text{ext})}$ is calculated and added to $\mathbf{y}_{(i)}$. Note that explicit matrix-vector products are only required for the approximate-Newton stage or for adjoint solves.

Preconditioners are the most critical component of an efficient parallel implicit linear solver. While excellent serial preconditioners exist for Newton-Krylov flow solvers,[4,15,16] these preconditioners cannot be implemented efficiently in parallel; for example, while using ILU(p) on the global system matrix, A, has proven to work well in serial, a parallel version results in substantial idle time and communication. The essence of the challenge is that a good parallel preconditioner must balance the competing objectives of scalability and serial performance.

C. Preconditioning

For the current work, two parallel preconditioners were investigated: one based on the additive Schwarz method and one based on an approximate Schur method. Both of the underlying methods require an exact or inexact inversion of local submatrices. The preconditioners presented here use an incomplete lower-upper factorization of the *local* submatrix of the modified Jacobian:

$$L_iU_i = \left[\mathcal{T}^{(n)} + \mathcal{A}_1^{(n)}\right]_i + R_i,$$

where R_i is the error in the factorization. Incomplete lower-upper factorization with a level of fill p, ILU(p)[40] is used to obtain the factorization L_iU_i. Notice that the factorization itself does not require inter-processor communication, since ILU(p) is applied to the local submatrices only. For the remaining sections, the submatrix A_i refers to the modified Jacobian submatrix $[\mathcal{T}^{(n)} + \mathcal{A}_1^{(n)}]_i$.

1. Additive Schwarz Preconditioner

The simplest form of additive Schwarz preconditioning is essentially a block Jacobi iteration; see, for example, Saad.[41] Given the vector w, the local component of the preconditioned vector z is given by the exact or inexact solution to the system

$$A_iz_i = w_i. \tag{22}$$

Equation (22) can be solved using a direct method or iteratively using, for example, GMRES. For this work, we solve (22) approximately using a single application of the ILU factorization: $z_i = U_i^{-1}L_i^{-1}w_i$. We have

investigated the use of preconditioned GMRES to solve (22), but solving the local system more accurately in this way was not found to be competitive.

Additive Schwarz methods can employ overlapping domains to improve the quality of the preconditioner, i.e. reduce the number of Krylov iterations. Numerical experiments by Gropp *et al.*[42] suggest that domain overlap, while capable of reducing the number of Krylov iterations, increases the overall CPU time; hence, we do not consider overlapping as a convergence strategy in this work.

2. Approximate Schur Preconditioner

The idea behind Schur complement methods is the elimination of the internal unknowns to form a reduced system of equations, called the Schur complement system. Saad and Sosonkina[25] have proposed a preconditioning technique based on an approximate factorization of the Schur complement system. Their contribution is summarized below.

Considering (21), we see that the internal variables can be written as

$$\mathbf{u}_{(i)} = B_i^{-1}(\mathbf{f}_{(i)} - F_i\mathbf{y}_{(i)}). \tag{23}$$

Substituting $\mathbf{u}_{(i)}$ into the equation for $\mathbf{y}_{(i)}$ we obtain the following system for the internal-interface variables on process i:

$$S_i\mathbf{y}_{(i)} + \sum_{j\in N_i} E_{ij}\mathbf{y}_{(j)} = \mathbf{g}_{(i)} - E_iB_i^{-1}\mathbf{f}_{(i)} \equiv \mathbf{g}'_{(i)}. \tag{24}$$

where $S_i = C_i - E_iB_i^{-1}F_i$ is the "local" Schur complement matrix. Assembling all the local Schur complement systems for each process, we obtain a linear system for all the internal-interface unknowns:

$$\underbrace{\begin{pmatrix} S_1 & E_{12} & \dots & E_{1P} \\ E_{21} & S_2 & \dots & E_{2P} \\ \vdots & & \ddots & \vdots \\ E_{P1} & E_{P2} & \dots & S_P \end{pmatrix}}_{S} \begin{pmatrix} \mathbf{y}_{(1)} \\ \mathbf{y}_{(2)} \\ \vdots \\ \mathbf{y}_{(P)} \end{pmatrix} = \begin{pmatrix} \mathbf{g}'_{(1)} \\ \mathbf{g}'_{(2)} \\ \vdots \\ \mathbf{g}'_{(P)} \end{pmatrix}. \tag{25}$$

The coefficient matrix, S, appearing in (25) is the Schur complement.[41] We will refer to this matrix as the global Schur complement to distinguish it from the S_i, which are the diagonal blocks of S.

We could assemble the global Schur complement matrix, solve the system (25), and then solve for the local internal unknowns on each process using (23). In practice, however, forming the Schur complement matrix and solving for the interface unknowns exactly is not competitive with other methods.[25] Instead, we consider systems that approximate (25) and act as preconditioners for the global system (19).

Consider the following block factorization of A_i;

$$A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} = \begin{pmatrix} B_i & 0 \\ E_i & S_i \end{pmatrix} \begin{pmatrix} I & B_i^{-1}F_i \\ 0 & I \end{pmatrix}. \tag{26}$$

Next, suppose A_i has been factored instead into $A_i = L_iU_i$, where

$$L_i = \begin{pmatrix} L_{B_i} & 0 \\ E_iU_{B_i}^{-1} & L_{S_i} \end{pmatrix}, \qquad \text{and} \qquad U_i = \begin{pmatrix} U_{B_i} & L_{B_i}^{-1}F_i \\ 0 & U_{S_i} \end{pmatrix}.$$

Comparing the factors in (26) with L_i and U_i we can show that[25]

$$S_i = L_{S_i}U_{S_i}. \tag{27}$$

So, we can obtain an LU decomposition of the local Schur complement by extracting the relevant blocks from the LU decomposition of A_i. Similarly, and more relevant to preconditioning, *we can obtain an approximate factorization of S_i by extracting the relevant blocks from the ILU factorization of A_i.*

Suppose the local matrix has been factored as $A_i = L_i U_i + R_i$. Then we can define the following approximate local-Schur complement system using (27) and (24);

$$\mathbf{y}_{(i)} = U_{S_i}^{-1} L_{S_i}^{-1} \left(\mathbf{g}'_{(i)} - \sum_{j \in N_i} E_{ij} \mathbf{y}_{(j)} \right). \tag{28}$$

The above equation is a single iteration of block Jacobi on the local internal-interface unknowns. This system can be further accelerated using a Krylov-subspace method such as GMRES.[41] Once the approximations to the $\mathbf{y}_{(i)}$ have been exchanged, we can substitute the $\mathbf{y}_{(i,\text{ext})}$ into (20), and apply the ILU factorization to obtain approximate values for $\mathbf{x}_{(i)}$.

We have implemented an approximate Schur preconditioner which is closely based on algorithm 3.1 of Saad and Sosonkina.[25] Two important changes to the original algorithm involve the complete forward and backward solves, $(L_i U_i)^{-1}$, on lines 5 and 25 of their algorithm. These forward-backward solves are modified with partial operations involving U_{S_i} and L_{S_i}, such that, mathematically our algorithm is equivalent, but computationally it is approximately 20% more efficient. The relevant modifications can be found on lines 1, 22, and 23 of algorithm 1 in the appendix. These modifications, while transparent in a mathematical sense, are essential if the approximate Schur preconditioner is to be competitive with additive Schwarz.

A linear solver that uses the approximate Schur preconditioner must be a flexible variant, that is, the solver must be compatible with preconditioning that varies from iteration to iteration. For this reason, we use Flexible GMRES (FGMRES)[43] with the Schur preconditioner. FGMRES uses approximately twice the memory of GMRES, although it is essentially identical in terms of CPU time.

In the context of the approximate Schur preconditioner, one advantage of using SATs to couple the blocks is that the reduced system size is independent of the order of the interior scheme; therefore, we anticipate that the approximate Schur preconditioner will be well suited to parallel implicit high-order finite-difference schemes.

IV. Results

The results presented in the following subsections have been obtained on grids for the ONERA M6 wing. The dimensions of the grids are listed in Table 1. All grids use an HH topology and 25 chord lengths to the far field. For a given grid, each block has identical N_j, N_k, and N_m dimensions. Sizing the blocks this way allows for better load balancing; future work may consider block decomposition as a means of load balancing arbitrary structured grids.

Table 1. ONERA M6 grids and their dimensions.

grid name	blocks	dimensions ($N_j \times N_k \times N_m$)	grid size
M6-12-1094	12	$45 \times 45 \times 45$	1093500
M6-12-146	12	$23 \times 23 \times 23$	146004
M6-48-458	48	$33 \times 17 \times 17$	457776

A. Mitigation of Memory Contention Issues

We have obtained all results on an SMP HP Itanium Beowolf-class cluster. Each node on the cluster consists of 4 Itanium 2 processors with 6 MBytes L3 cache and a clock speed of 1500 MHz. The nodes each have at least 8 GBytes of RAM, and are connected with a high-bandwidth low-latency Myrinet network.

While validating the parallel solver, we observed that some operations which should have been close to 100% efficient were not. Memory contention was identified as the problem; when the same cases were run with 1 process per node the efficiency improved to the expected levels. Clearly, running with only 1 process per node is not a suitable resolution of the memory contention issue.

One way to reduce memory contention for this architecture is by improving cache residency. If the number of floating point operations per memory access can be increased, i.e. fewer cache misses, then the likelihood of contention should decrease. The subroutines that experience the most cache misses are the

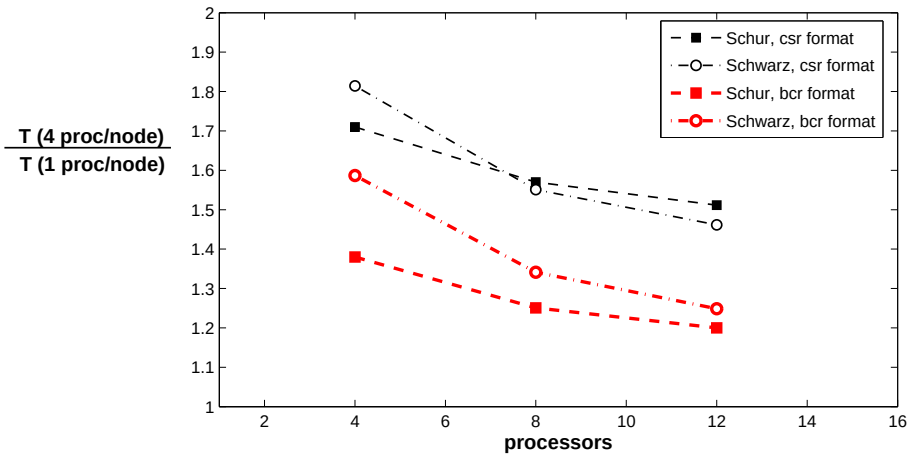


Figure 3. Ratio of solver run times using 4 processors per node to solver run times using 1 processor per node, as a function of number of processors.

explicit matrix-vector products and the $U_i^{-1}L_i^{-1}$ forward-backward solves. These subroutines loop through compressed matrix rows and access non-sequential elements of a vector. This non-sequential access of vector elements is responsible for most of the memory contention.

The Jacobian matrix for the Euler and Navier-Stokes equations has a natural 5×5 block structure. If the matrix is stored as rows of blocks rather than rows of numbers then matrix-vector operations should experience fewer cache misses. This matrix storage scheme is referred to as a block compressed row (bcr) storage format; the original code used the traditional compressed sparse row (csr) storage scheme.

For the csr format, the ILU(p) factorization is slightly modified to allow fill-in for any block that contains non-zeros, i.e. Block-Fill ILU(p) (BFILU(p)[44]). For the bcr format, we use block ILU(p) (BILU(p)), which is analogous to ILU(p) except that matrix operations replace scalar operations. For the Euler equations with scalar dissipation, BFILU(p) and BILU(p) lead to almost identical ILU factorizations for csr and bcr formats, respectively; the only differences are due to the 5×5 SAT blocks at interface nodes. It follows that differences in CPU time between the two formats are attributable primarily to changes in cache residency.

Flow solutions for the M6-48-458 grid were obtained for 4, 8, and 12 processors at a Mach number of 0.699 and an angle of attack of 3.06 degrees. Both the csr and bcr matrix formats were used together with the approximate Schur and additive Schwarz preconditioners. Run times using 4 processors per node, T(4 proc/node), were scaled by run times using 1 processor per node, T(1 proc/node); these ratios are plotted in Figure 3. For all cases, memory contention decreases as the number of processors increases, since the local vectors become smaller. The results demonstrate that the bcr format outperforms the csr format on this architecture, independent of preconditioner. The figure also shows that the additive Schwarz preconditioner suffers more memory contention than the approximate Schur. More GMRES iterations are required when using the Schwarz preconditioner, which means more calls to the subroutines that suffer the greatest memory contention.

The remaining results presented in this paper use the bcr format and BILU(1) factorizations.

B. Preconditioner Comparison

To test the parallel efficiency of the preconditioners, flow solutions were timed for the M6-12-146 and M6-48-458 grids. The operating conditions were fixed at a Mach number of 0.699 and an angle of attack of 3.06 degrees. The discrete equations were solved to a relative tolerance of 10^{-10} using the Newton-Krylov algorithm. The linear sub-problems were solved using FGMRES limited to 60 and 80 iterations for the 12 and 48 block grids, respectively.

Figure 4 compares the speed-up and parallel efficiency of the additive Schwarz and approximate Schur preconditioners for the M6-12-146 grid. In the figure, T is the run time and T_{ref} the reference serial run time: both preconditioners reduce to ILU(1) for serial computations. To avoid complications introduced by

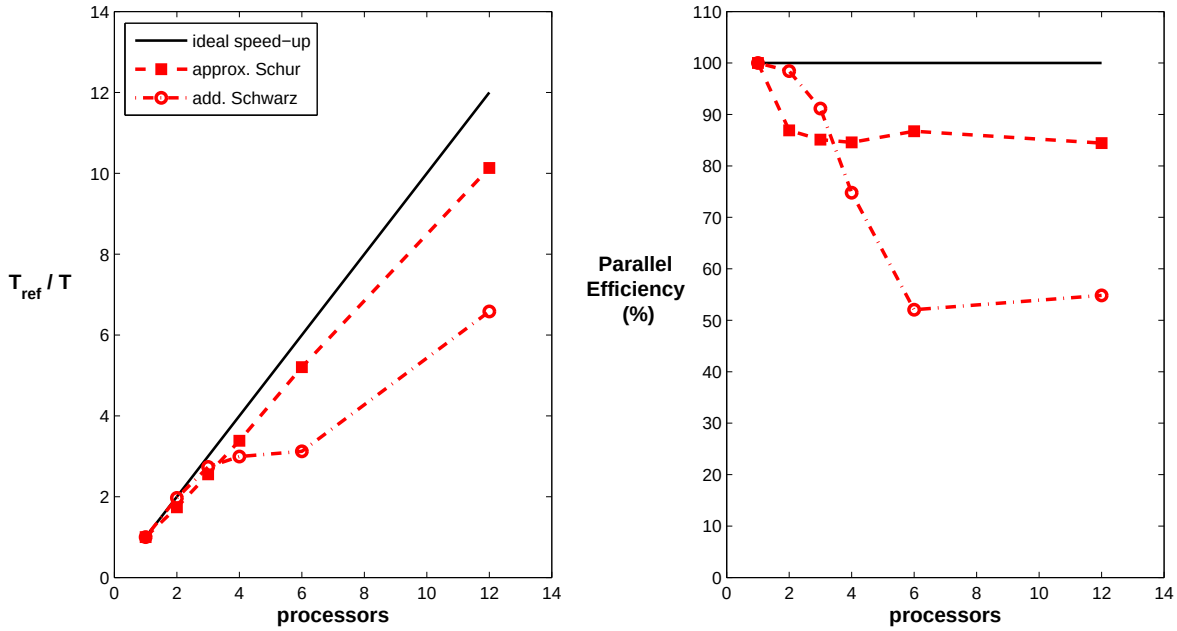


Figure 4. Speed-up and parallel efficiency of the approximate Schur and additive Schwarz preconditioners for the grid M6-12-146. Results were obtained for 1 processor per node to eliminate issues related to memory contention.

memory contention, the timings for this grid were obtained using 1 processor per node. Except for the runs with 2 and 3 processors, the approximate Schur preconditioner has the best efficiency. Moreover, the Schur preconditioner demonstrates excellent scalability as the number of processors increases.

Speed-up and efficiency results for the M6-48-458 grid are presented in Figure 5. The reference time was chosen, arbitrarily, to be the 24 processor approximate Schur timing: $\mathrm{T_{ref}} = 24 \times \mathrm{T_{Schur}}(24\,\mathrm{proc})$. In order to obtain results for 48 processors, it was necessary to run with 4 processors per node; hence, the results in Figure 5 are confounded by memory contention. The effects of contention are reflected in the increase in efficiency as the number of processors increases from 4 to 24. The approximate Schur preconditioner performs slightly better than the additive Schwarz preconditioner for the M6-48-458 grid, but not to the degree of the 12 block grid. It is not clear why the additive Schwarz preconditioner performs poorly on the smaller grid.

The preconditioners can also be assessed by comparing the total number of FGMRES iterations used during a flow solution. Figure 6 shows that the iterations required by the approximate Schur preconditioner are relatively independent of the number of processors, for both grids considered here. In contrast, the additive Schwarz preconditioner requires more iterations as more processors as added. This difference reflects the implicit coupling between domains achieved by the approximate Schur preconditioner.

For the grids considered, the results suggest that the approximate Schur preconditioner is a better choice than the additive Schwarz preconditioner, in terms of CPU time and Krylov iterations. The Schwarz preconditioner may be useful if memory considerations are important, since the Schur preconditioner requires FGMRES while the Schwarz preconditioner can use GMRES (see the discussion at the end of Section III.C.2).

C. Jacobian Update Period

Recall that the parameter m controls the frequency with which the modified Jacobian matrix is calculated and factored during the approximate-Newton phase. The M6-12-146 ONERA M6 wing grid was used to obtain flow solutions for the update period range $m \in \{1, 2, \ldots, 7\}$. Mach numbers of 0.5, 0.699, and 0.84 were considered.

Figure 7 plots the solution time versus the number of iterations between Jacobian updates. The time is

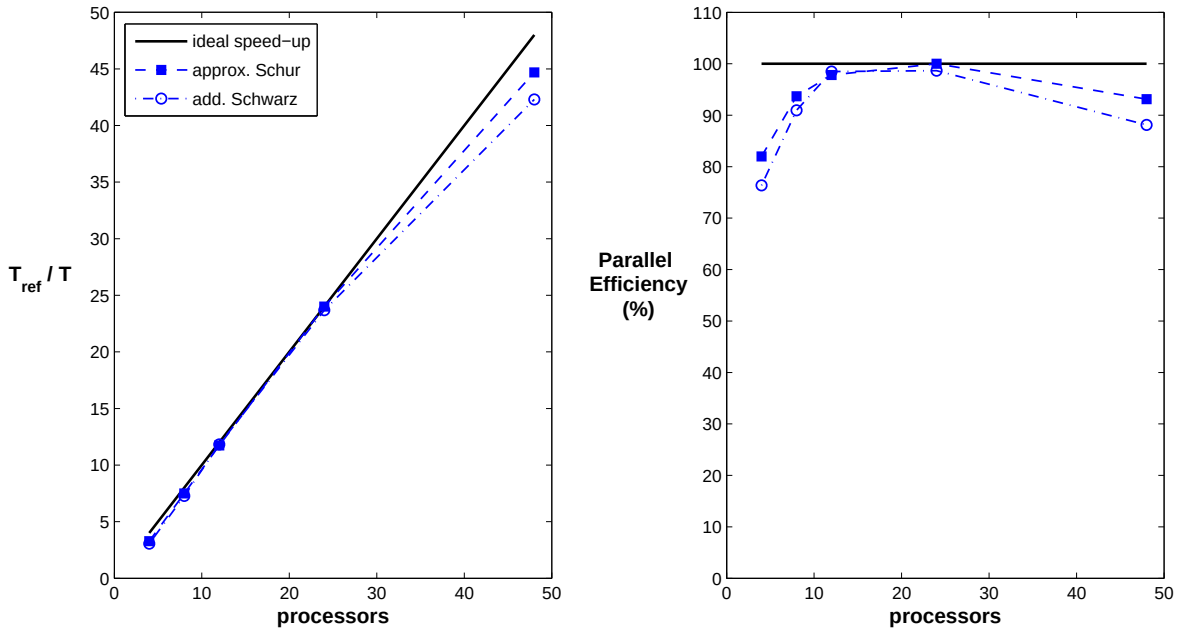


Figure 5. Speed-up and parallel efficiency of the approximate Schur and additive Schwarz preconditioners for the grid M6-48-146. Results are for 4 processors per node, so memory contention is present.

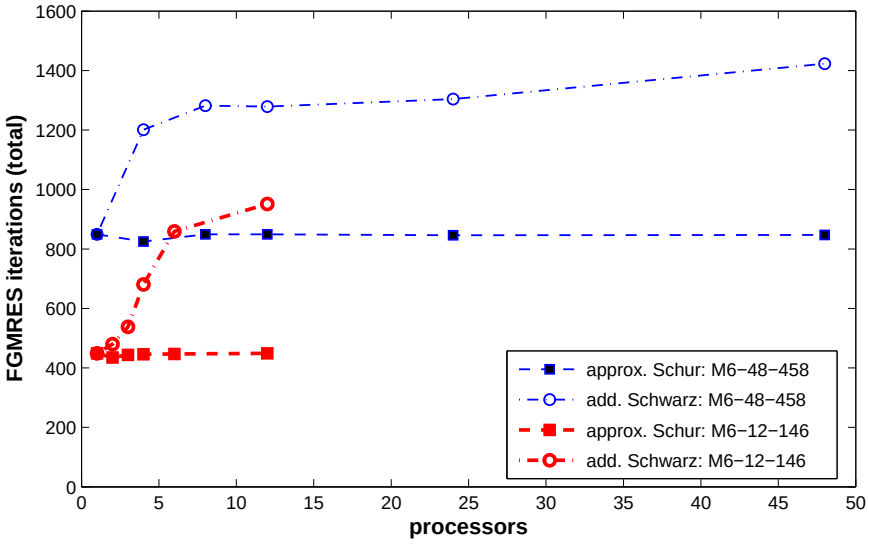


Figure 6. Total number of FGMRES iterations as a function of the number of processors, for grids M6-12-146 and M6-48-458.

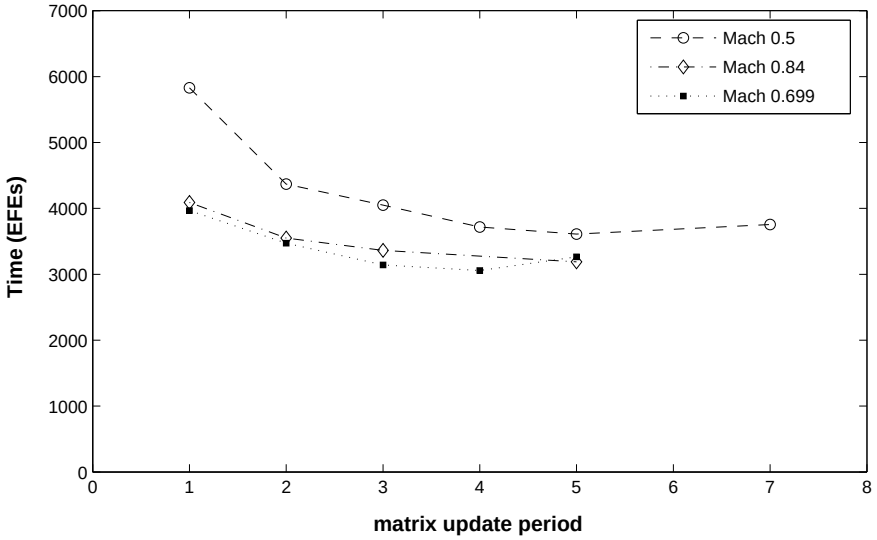


Figure 7. Flow solve CPU time, measured in equivalent function evaluations, as a function of the Jacobian update period m.

given in equivalent function evaluations (EFEs); this measure scales the CPU time by the time required to evaluate $\boldsymbol{\mathcal{F}}$ once. The result shows that m can have a significant impact on the algorithm, reducing CPU time between 20% and 30% as m increases from 1 to 5. As the update period increases beyond $m = 5$, convergence success becomes less likely and non-physical pressures and densities are encountered. These results suggest that values of $m \in \{3, 4, 5\}$ provide a good tradeoff between CPU time and convergence success. We have used $m = 5$ for the results presented in this paper.

D. Solution Quality

The flow over the ONERA M6 wing was used to validate the flow solver. Solutions were obtained for the ONERA M6 grid M6-12-1094 at Mach numbers 0.699 and 0.84. The angle of attack was fixed at 3.06 degrees.

Figures 9 and 10 compare the numerical solutions with the experimental results of Schmitt and Charpin.[45] The comparisons are purely qualitative, since viscous effects are neglected in the numerical results. The comparisons are intended to verify the discretization based on the SBP operators and the SATs.

Figure 8 plots the contours of pressure on the symmetry plane of the ONERA M6 computation, at a Mach number of 0.699. A multivalued solution, permitted by the SAT formulation, is evident near the leading edge; however, the grid used is coarse, and the discrepancy between blocks does not affect stability or accuracy.

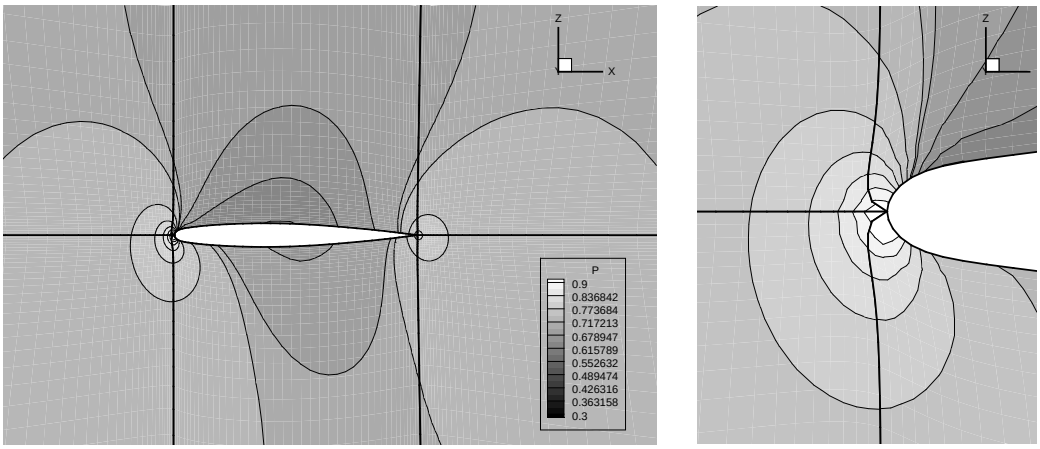
E. Adjoint Solution

The inexact-Newton strategy used in the flow solver requires relatively few FGMRES (or GMRES) iterations. This is an important point, since the computational work needed by GMRES grows quadratically with the number of iterations. If we decrease the desired tolerance, a preconditioner which requires fewer iterations becomes more attractive.

The discrete adjoint variables, $\boldsymbol{\psi}$, used in optimization are solutions to equations of the form

$$\mathcal{A}^T \boldsymbol{\psi} = \mathbf{b} \tag{29}$$

where $(\mathbf{b})_i = \partial \mathcal{J} / \partial q_i$ and $\mathcal{J}$ is an objective function. The adjoint equation (29) must be solved to a sufficiently small tolerance to ensure that the gradient calculation is accurate.[46] This leads us to the question: how do the two preconditioners compare when used to solve the adjoint problem?



(a) pressure contours at the symmetry plane (b) close-up near leading edge

Figure 8. Pressure contours at the symmetry plane of the ONERA M6 wing at a Mach number of 0.699.

To answer this question, the adjoint equations for $\mathcal{J} = C_l/C_d$ were formed and solved on the ONERA M6 grids M6-12-146 and M6-48-458. The same flow conditions, $M = 0.699$ and $\alpha = 3.06$, were used for both grids, and 12 and 48 processors were used for the small and large grids, respectively. The flow equations were first solved to a relative tolerance of 10^{-10}, and the adjoint systems were subsequently formed and solved to a relative tolerance of 10^{-6}.

In addition to comparing the two preconditioners, we also consider the use of BiCGSTAB[47] in the solution of the adjoint system (29). Recent work in two-dimensional, unsteady optimization[48] suggests that BiCGSTAB is significantly faster than GMRES in this context. No flexible variant of BiCGSTAB exists, so we only consider the additive Schwarz preconditioner for this solver.

Table 2 compares the preconditioner-solver pairs applied to the adjoint equations on the two grids. When using FGMRES, we see that the approximate Schur preconditioner is 50% faster than the additive Schwarz preconditioner. The approximate Schur-FGMRES pair is also significantly faster than the Schwarz-BiCGSTAB combination, but this difference reduces as the problem size grows. The reason, as mentioned above, is that work increases quadratically with the number of iterations for FGMRES while it increases linearly for BiCGSTAB. Thus, for larger problems BiCGSTAB will eventually be faster than FGMRES.

Memory is another consideration that may favour BiCGSTAB for larger problems. For example, on the larger grid the approximate Schur-FGMRES pair requires 384 Krylov subspace vectors (768 vectors total), which is equivalent in terms of memory to approximately 12 Jacobian matrices. Similarly, the memory required by additive Schwarz-GMRES is approximately 10 Jacobian matrices on the larger grid. Preconditioned BiCGSTAB requires only 8 vectors, independent of the number of iterations.

Table 2. Comparison of the preconditioner-solver pairs applied to the adjoint system.

		Grid M6-12-146		Grid M6-48-458	
solver	preconditioner	CPU Time (s)	Krylov Iterations	CPU Time (s)	Krylov Iterations
BiCGSTAB	Schwarz (0 iter)	86.40	411	141.63	857
FGMRES	Schwarz (0 iter)	73.41	365	165.54	676
FGMRES	Schur	35.23	179	91.75	384

V. Conclusions

We draw the following conclusions.

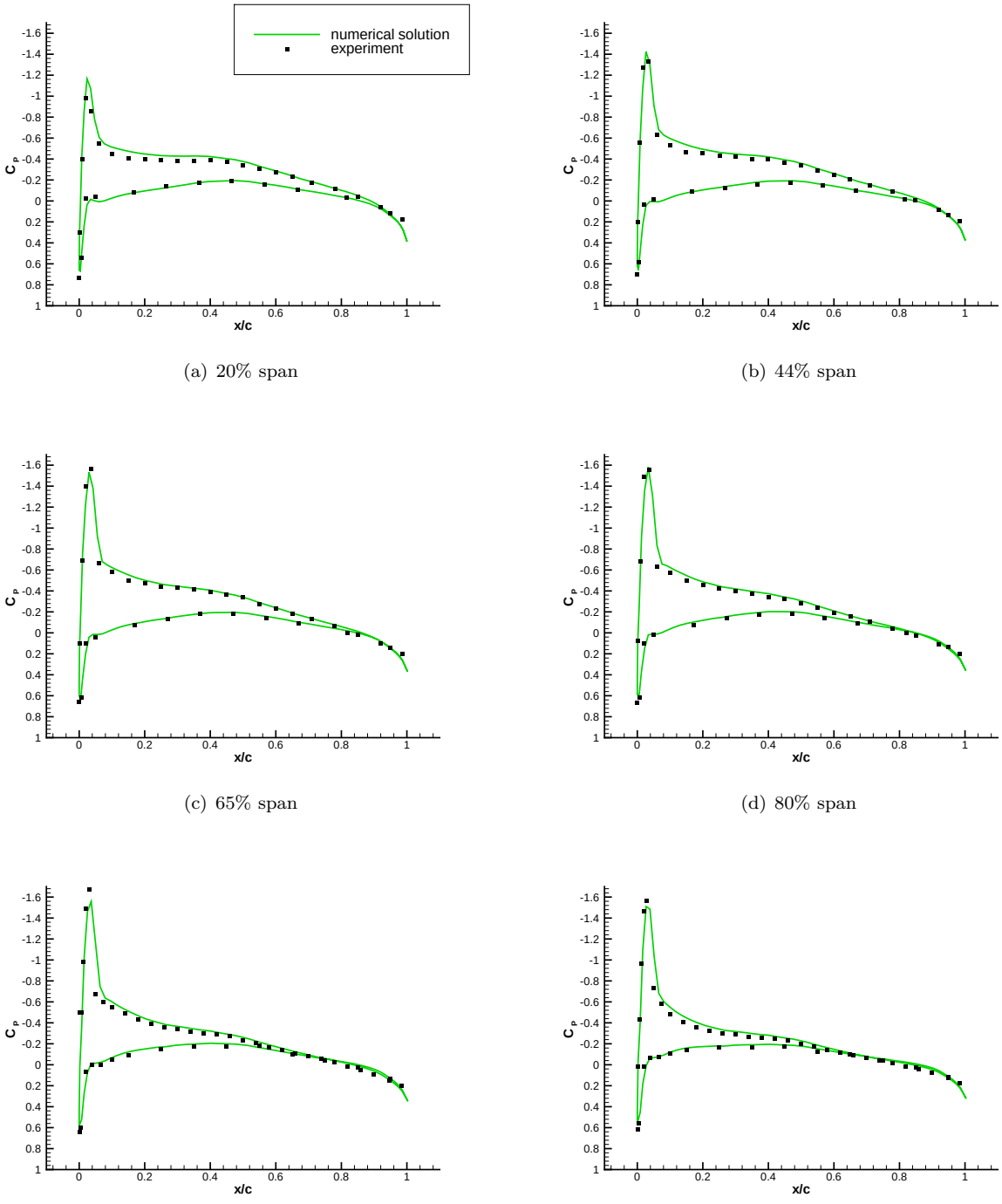
- A second-order discretization of the Euler equations using SATs at the boundaries and interfaces is well suited to a parallel Newton-Krylov solution strategy.
- For the Euler equations, the approximate Schur preconditioner outperforms the additive Schwarz preconditioner in terms of CPU time and Krylov iterations.
- The approximate Schur preconditioner must use FGMRES, which requires approximately twice the memory of GMRES; therefore, the Schwarz preconditioner is a good choice when memory is limited.
- For the grids considered, with between 10^5 to 10^6 nodes, the most efficient choice for the adjoint equations is FGMRES with the approximate Schur preconditioner. For larger grids, BiCGSTAB with additive Schwarz may be a better choice, but further investigation is needed.

Appendix

Algorithm 1: Approximate Schur Preconditioner

Data: m, η, $\mathbf{u}$
Result: $\mathbf{v}$

$\mathbf{v} = U_{S_i}^{-1} L_i^{-1} \mathbf{u}$ `// get residual of reduced system for a guess of zero`
$\mathbf{w}_{(i)}^{(1)} = P\mathbf{v}$ `// P projects onto the space of internal-interface unknowns`
set $\beta = \|\mathbf{w}_{(i)}^{(1)}\|_2$, $\mathbf{w}_{(i)}^{(1)} \leftarrow \mathbf{w}_{(i)}^{(1)}/\beta$, and $H = 0$
for $j = 1, m$ **do**
    obtain external interface values, $\mathbf{w}_{(i,\text{ext})}^{(j)}$
    $\mathbf{w}_{(i)}^{(j+1)} = E_i \mathbf{w}_{(i,\text{ext})}^{(j)}$ `// perform external matrix-vector product`
    $\mathbf{w}_{(i)}^{(j+1)} \leftarrow U_{S_i}^{-1} L_{S_i}^{-1} \mathbf{w}_{(i)}^{(j+1)}$ `// apply diagonal block of Schur complement`
    $\mathbf{w}_{(i)}^{(j+1)} \leftarrow \mathbf{w}_{(i)}^{(j)} + \mathbf{w}_{(i)}^{(j+1)}$ `// finish the matrix-vector product, equation` (28)
    for $k = 1, j$ **do** Gram-Schmidt orthogonalization
        $H_{k,j} = (\mathbf{w}_{(i)}^{(j+1)})^T \mathbf{w}_{(i)}^{(k)}$
        $\mathbf{w}_{(i)}^{(j+1)} \leftarrow \mathbf{w}_{(i)}^{(j+1)} - H_{k,j} \mathbf{w}_{(i)}^{(k)}$
    end
    $H_{j+1,j} = \|\mathbf{w}_{(i)}^{(j+1)}\|_2$
    $\mathbf{w}_{(i)}^{(j+1)} \leftarrow \mathbf{w}_{(i)}^{(j+1)}/H_{j+1,j}$
    if *reduced system residual tolerance* $\leq \eta$ **then**
        set $m = j$ and exit
    end
end
Define $W_m = [\mathbf{w}_{(i)}^{(1)}, \ldots, \mathbf{w}_{(i)}^{(m)}]$
Compute $\mathbf{y}_{(i)} = W_m \mathbf{z}^{(m)}$ where $\mathbf{z}^{(m)} = \min_z \|\beta e_1 - Hz\|_2$ and $e_1 = [1, 0, \ldots, 0]^T$
obtain external interface preconditioned values, $\mathbf{y}_{(i,\text{ext})}$
$\mathbf{v} \leftarrow \mathbf{v} + P^T P(\mathbf{u} - E_i \mathbf{y}_{(i,\text{ext})} - \mathbf{v})$ `// updates internal-interface only`
$\mathbf{v} \leftarrow U_i^{-1} \left[I + P^T P(L_i^{-1} - I)\right] \mathbf{v}$ `// forward solve is applied to interface only`



(a) 20% span

(b) 44% span

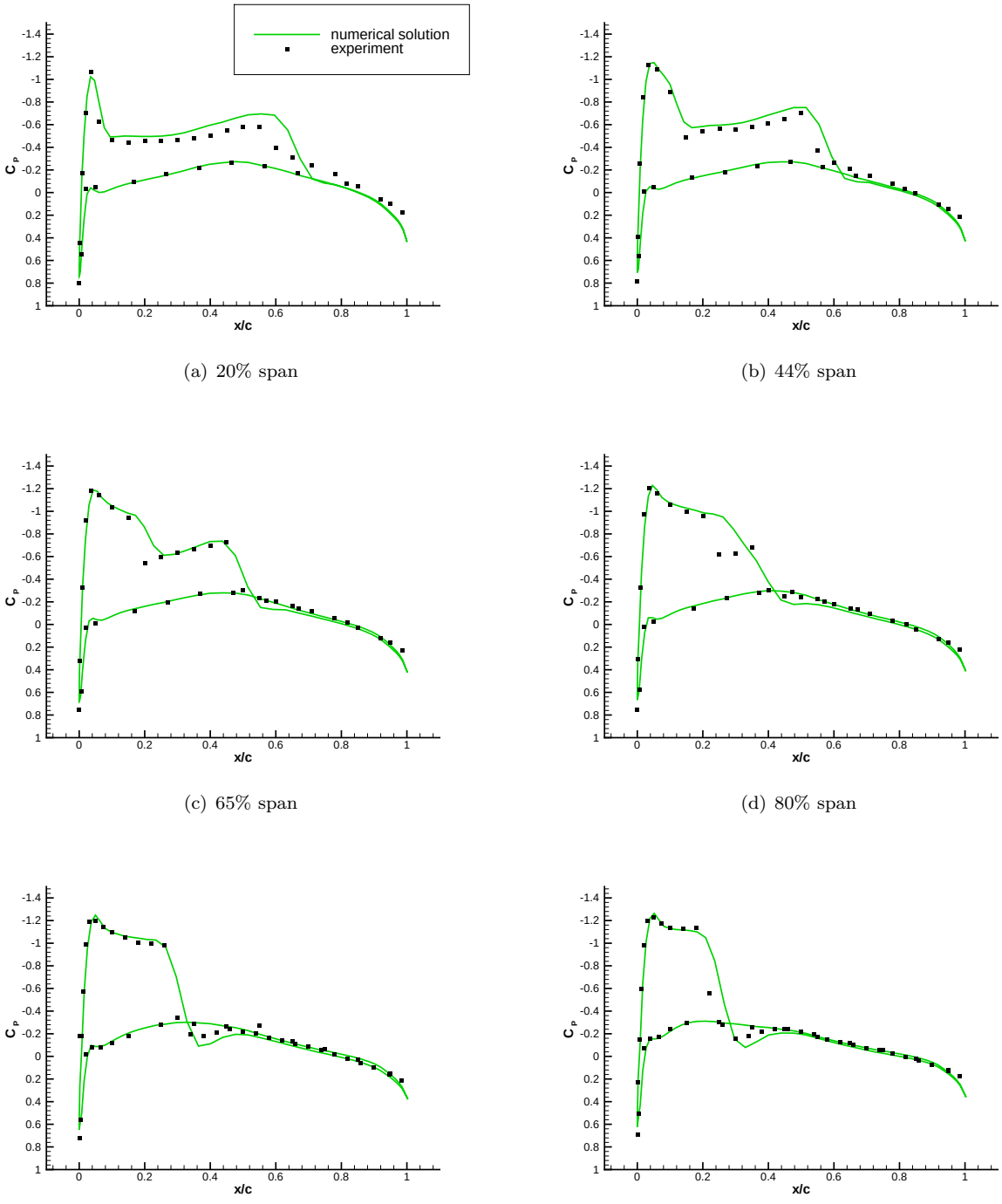
(c) 65% span

(d) 80% span

(e) 90% span

(f) 96% span

Figure 9. Comparison of the coefficient of pressure at a Mach number of 0.699 around the ONERA M6 wing.



(a) 20% span

(b) 44% span

(c) 65% span

(d) 80% span

(e) 90% span

(f) 96% span

Figure 10. Comparison of the coefficient of pressure at a Mach number of 0.84 around the ONERA M6 wing.

Acknowledgments

The first author would like to thank Edwin van der Weide and Magnus Svärd for their generous help explaining the implementation of the SAT. The authors gratefully acknowledge financial assistance from the Natural Sciences and Engineering Research Council (NSERC), the Canada Research Chairs program, and the University of Toronto.

References

[1]Agarwal, R., "Computational fluid dynamics of whole-body aircraft," *Annual Review of Fluid Mechanics*, Vol. 31, 1999, pp. 125–169.

[2]De Rango, S. and Zingg, D. W., "Higher-order spatial discretization for turbulent aerodynamic computations," *AIAA Journal*, Vol. 39, No. 7, July 2001, pp. 1296–1304.

[3]Nemec, M., Zingg, D. W., and Pulliam, T. H., "Multipoint and multi-objective aerodynamic shape optimization," *AIAA Journal*, Vol. 42, No. 6, 2004, pp. 1057–1065.

[4]Nichols, J. and Zingg, D. W., "A three-dimensional multi-block Newton-Krylov flow solver for the Euler equations," *17th AIAA Computational Fluid Dynamics Conference*, No. AIAA–2005–5230, Toronto, Canada, June 2005.

[5]Carpenter, M. H., Gottlieb, D., and Abarbanel, S., "Time-stable boundary conditions for finite-difference schemes solving hyperbolic systems: methodology and application to high-order compact schemes," *Journal of Computational Physics*, , No. 111, 1994, pp. 220–236.

[6]Hesthaven, J. S., "A stable penalty method for the compressible Navier-Stokes equations: III. Multidimensional domain decomposition schemes," *SIAM Journal of Scientific Computing*, Vol. 20, No. 1, 1998, pp. 62–93.

[7]Carpenter, M. H., Nordström, J., and Gottlieb, D., "A stable and conservative interface treatment of arbitrary spatial accuracy," *Journal of Computational Physics*, , No. 148, 1999, pp. 341–365.

[8]Nordström, J. and Carpenter, M. H., "High-order finite difference methods, multidimensional linear problems, and curvilinear coordinates," *Journal of Computational Physics*, , No. 173, 2001, pp. 149–174.

[9]Mattsson, K., Svärd, M., and Nordström, J., "Stable and accurate artificial dissipation," *Journal of Scientific Computing*, Vol. 21, No. 1, 2004, pp. 57–79.

[10]Hesthaven, J. S. and Gottlieb, D., "A stable penalty method for the compressible Navier-Stokes equations: I. open boundary conditions," *SIAM Journal of Scientific Computing*, Vol. 17, No. 3, 1996, pp. 579–612.

[11]Nordström, J. and Carpenter, M. H., "Boundary and interface conditions for high-order finite-difference methods applied to the Euler and Navier-Stokes equations," *Journal of Computational Physics*, , No. 148, 1999, pp. 621–645.

[12]Johan, Z., Hughes, T. J. R., and Shakib, F., "A globally convergent matrix-free algorithm for implicit time-marching schemes arising in finite element analysis in fluids," *Computer Methods in applied mechanics and engineering*, , No. 87, 1991, pp. 281–304.

[13]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*, San Diego, CA, 1995, AIAA Paper 95–1733.

[14]Anderson, W. K., Rausch, R. D., and Bonhaus, D. L., "Implicit/multigrid algorithms for incompressible turbulent flows on unstructured grids," *Journal of Computational Physics*, , No. 128, 1996, pp. 391–408.

[15]Pueyo, A. and Zingg, D. W., "Efficient Newton-Krylov solver for aerodynamic computations," *AIAA Journal*, Vol. 36, No. 11, Nov. 1998, pp. 1991–1997.

[16]Blanco, M. and Zingg, D. W., "Fast Newton-Krylov method for unstructured grids," *AIAA Journal*, Vol. 36, No. 4, April 1998, pp. 607–612.

[17]Chisholm, T. T. and Zingg, D. W., "A Newton-Krylov algorithm for turbulent aerodynamic flows," *39th AIAA Aerospace Sciences Meeting and Exhibit*, No. AIAA 2003–0071, Reno, Nevada, 2003.

[18]Pulliam, T., Nemec, M., Holst, T., and Zingg, D., "Comparison of evolutionary (genetic) algorithm and adjoint methods for multi-objective viscous airfoil optimizations," AIAA Paper 2003–0298, Jan. 2003.

[19]Barth, T. J. and Linton, S. W., "An unstructured mesh Newton solver for compressible fluid flow and its parallel implementation," *33rd AIAA Aerospace Sciences Meeting and Exhibit*, No. AIAA–95–0221, Reno, Nevada, 1995.

[20]Keyes, D. E., "Aerodynamic applications of Newton-Krylov-Schwarz solvers," *Proceedings of the 14th International Conference on Numerical Methods in Fluid Dynamics*, Springer, New York, 1995, pp. 1–20.

[21]Kaushik, D. K., Keyes, D. E., and Smith, B. F., "Newton-Krylov-Schwarz methods for aerodynamics problems: compressible and incompressible flows on unstructured grids," *11th International Conference on Domain Decomposition Methods*, 1998.

[22]Groth, C. P. and Northrup, S. A., "Parallel implicit adaptive mesh refinement scheme for body-fitted multi-block mesh," *17th AIAA Computational Fluid Dynamics Conference*, No. AIAA–2005–5333, Toronto, Ontario, Canada, June 2005.

[23]Knoll, D. and Keyes, D., "Jacobian-free Newton-Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, , No. 193, 2004, pp. 357–397.

[24]Cai, X.-C., Gropp, W. D., Keyes, D. E., and Tidriri, M. D., "Newton-Krylov-Schwarz methods in CFD," *International Workshop on Numerical Methods for the Navier-Stokes Equations*, Heidelberg, 1993.

[25]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.

[26]Svärd, M., *Stable High-Order Finite Difference Methods for Aerodynamics*, Ph.D. thesis, Uppsala University, 2004.

[27]Kreiss, H.-O. and Scherer, G., "Finite element and finite difference methods for hyperbolic partial differential equations," *Mathematical aspects of finite elements in partial differential equations*, edited by C. de Boor, Mathematics Research Center, the University of Wisconsin, Academic Press, 1974.

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

[29]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*, Palo Alto, CA, 1981, AIAA Paper 81–1259.

[30]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, Brussels, Belgium, Jan. 1986.

[31]Kelley, C. T., *Solving Nonlinear Equations With Newton's Method*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2003.

[32]Kelley, C. T. and Keyes, D. E., "Convergence analysis of pseudo-transient continuation," *SIAM Journal of Numerical Analysis*, Vol. 35, No. 2, April 1998, pp. 508–523.

[33]Kim, D. B. and Orkwis, P. D., "Jacobian update strategies for quadratic and near-quadratic convergence of Newton and Newton-like implicit schemes," *31st AIAA Aerospace Sciences Meeting and Exhibit*, No. AIAA–93–0878, Reno, Nevada, 1993.

[34]Chisholm, T. T., *A Fully Coupled Newton-Krylov Solver With a One-Equation Turbulence Model*, Ph.D. thesis, University of Toronto, 2007.

[35]Mulder, W. A. and van Leer, B., "Experiments with implicit upwind methods for the Euler equations," *Journal of Computational Physics*, , No. 59, 1985, pp. 232–246.

[36]Zingg, D. W. and Chisholm, T. T., "Jacobian-free Newton-Krylov Methods: issues and solutions," *Proceedings of The Fourth International Conference on Computational Fluid Dynamics*, Ghent, Belgium, July 2006, to appear.

[37]Eisenstat, S. C. and Walker, H. F., "Choosing the forcing terms in an inexact Newton method," *SIAM Journal of Scientific Computing*, Vol. 17, No. 1, Jan. 1996, pp. 16–32.

[38]Saad, Y. and Schultz, M. H., "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, No. 3, July 1986, pp. 856–869.

[39]Saad, Y. and Malevsky, A. V., "P-SPARSLIB: a portable library of distributed memory sparse iterative solvers," Tech. rep., University of Minnesota, May 1995.

[40]Meijerink, J. A. and van der Vorst, H. A., "An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix," *Mathematics of Computation*, Vol. 31, No. 137, Jan. 1977, pp. 148–162.

[41]Saad, Y., *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, PA, 2nd ed., 2003.

[42]Gropp, W. D., Kaushik, D. K., Keyes, D. E., and Smith, B. F., "High-performance parallel implicit CFD," *Parallel Computing*, , No. 27, 2001, pp. 337–362.

[43]Saad, Y., "A flexible inner-outer preconditioned GMRES algorithm," *SIAM Journal on Scientific and Statistical Computing*, Vol. 14, No. 2, 1993, pp. 461–469.

[44]Orkwis, P. D., "Comparison of Newton's and quasi-Newton's method solvers for the Navier-Stokes equations," *AIAA Journal*, Vol. 31, No. 5, 1993, pp. 832–836.

[45]Schmitt, V. and Charpin, F., "Pressure distributions on the ONERA-M6-wing at transonic mach numbers," Tech. rep., Office National d'Etudes et Recherches Aerospatiales, 92320, Chatillon, France, 1979.

[46]Nemec, M., *Optimal Shape Design of Aerodynamic Configurations: A Newton-Krylov Approach*, Ph.D. thesis, University of Toronto, 2003.

[47]van der Vorst, H. A., "BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, Vol. 13, No. 2, March 1992, pp. 631–644.

[48]Rumpfkeil, M. and Zingg, D. W., "A general framework for the optimal control of unsteady flows with applications," *45th AIAA Aerospace Sciences Meeting and Exhibit*, No. AIAA–2007–1128, Reno, Nevada, 2007.