AIAA JOURNAL
Vol. 46, No. 11, November 2008

Parallel Newton–Krylov Solver for the Euler Equations Discretized Using Simultaneous-Approximation Terms

Jason E. Hicken* and David W. Zingg†
University of Toronto, Toronto, Ontario M3H 5T6, Canada

DOI: 10.2514/1.34810

We present a parallel Newton–Krylov algorithm for solving the three-dimensional Euler equations on multiblock 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. The summation-by-parts with simultaneous-approximation-terms approach is time-stable, requires only C^0 mesh continuity at block interfaces, accommodates arbitrary block topologies, and has low interblock-communication overhead. 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 investigated. The algorithm is tested on the ONERA M6 wing geometry. We conclude that the approximate Schur preconditioner is an efficient alternative to the Schwarz preconditioner. Overall, the results demonstrate that the Newton–Krylov algorithm is very efficient: using 24 processors, a transonic flow on a 96-block, 1-million-node mesh requires 12 minutes for a 10-order reduction of the residual norm.

I. Introduction

STATE-OF-THE-ART flow solvers are capable of finding accurate solutions for flows with moderate separation [1,2]; however, run times remain a problem for three-dimensional configurations, especially for applications such as unsteady flows and shape optimization. This motivates the development of efficient parallel flow solvers. Although 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: an algorithm that combines a technique for handling block interfaces and boundaries (simultaneous-approximation terms) with a solution strategy (Newton–Krylov) to produce an efficient parallel solver.

Our choice of grid type and discretization is based on experience with serial two- and three-dimensional flow solvers [3–5]. The semistructured, or multiblock, approach provides the flexibility to fit complex shapes while permitting accurate and efficient discretizations. In particular, multiblock finite difference discretizations can be readily extended to high-order schemes.

Despite their apparent simplicity, multiblock finite difference schemes present some challenges that must be addressed. Consider the discretization at nodes along a block interface. If we use the interior discretization for the interface nodes (for example, through halo nodes), then the mesh must be sufficiently smooth at the interface to maintain the desired accuracy. This often motivates the use of elliptic smoothing; however, although elliptic smoothing can improve the grid continuity between blocks, it cannot eliminate mesh singularities along edges and vertices that are inherent in the geometry or the block topology. Resolving these singularities (via isotropic mesh spacing, for example) is inefficient and unnecessary. Finally, the requirement for smoothness in the interface-normal direction places significant restrictions on how the mesh can be refined: one cannot, for example, refine one block independently of another.

Another issue posed by multiblock finite difference schemes is the discretization at points along edges and vertices of the blocks. At these points, the coordinate directions are either nonsmooth, as already discussed, or ambiguous: consider an edge on which only three blocks meet. An accurate and stable treatment of these exceptional points is not obvious, even for second-order methods. As the order of accuracy of the scheme increases, more points are exposed to this difficulty. In parallel computations, the exceptional points can introduce additional difficulties that diminish the appeal of finite difference schemes.

The problems associated with block interfaces and exceptional points can be eliminated with summation-by-parts (SBPs) operators and simultaneous-approximation terms (SATs). The SAT methodology was originally developed to enforce boundary conditions in an accurate and time-stable manner [6], but the method has also been extended to handle domain interfaces [7–9]. SATs have been successfully used by Mattsson et al. [10] for third- and fifth-order discretizations of the Euler equations. High-order SBP–SAT methods retain the advantages of the second-order scheme described here (C^0 mesh interfaces, time-stability, and exceptional point treatment) without the increasing communication overhead that is typical of halo approaches. This paper introduces the SBP–SAT discretization and focuses on the solution strategy; a future paper will consider high-order SBP–SAT algorithms.

Coupling blocks with SAT penalties has been shown to significantly reduce the maximum stable Courant–Friedrichs–Lewy (CFL) number for explicit schemes [11,12]. This suggests that a Newton–Krylov solution strategy may be well suited to SAT discretizations. For serial computations, Newton–Krylov solution strategies have proven to be efficient, both in flow simulation [5,13–18] and optimization [4,19]. There are also many examples demonstrating that the excellent serial performance of Newton–Krylov algorithms can be extended to parallel algorithms [20–24].

Krylov solvers are readily 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 (ILU) [25]] and many parallel preconditioners tend to scale poorly. In the current work, we evaluate an additive Schwarz preconditioner [25] and an approximately factored Schur-complement preconditioner [26].

Presented as Paper 4333 at the 18th AIAA Computational Fluid Dynamics Conference, Miami, FL, 25–28 June 2007; received 26 September 2007; accepted for publication 17 July 2008.

*Ph.D. Candidate, Institute for Aerospace Studies. Student Member AIAA.

†Professor and Director, Tier 1 Canada Research Chair in Computational Aerodynamics, Institute for Aerospace Studies. Associate Fellow AIAA.

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

II. Governing Equations and Discretization

A. Transformed Euler Equations

We consider the three-dimensional Euler equations:

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

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

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

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

$$\mathbf{x} = \mathbf{F}(\boldsymbol{\xi}), \qquad \mathbf{F}\colon D \to P \quad \text{and} \quad \mathbf{F} \in C^1 \quad \text{on } D \backslash \partial D$$
$$D = \{\boldsymbol{\xi} \in \mathbb{R}^3 | \xi_i \in [0, L_i], i = 1, 2, 3\}$$

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

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

The scalar J denotes the Jacobian of the mapping, and 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 SATs [6]. 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 [9]).

B. Discretization for One-Dimensional Problems

In an effort to keep our presentation self-contained, this section summarizes 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 [6,8–10,27–29].

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

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

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 mth-order operator, the truncation error satisfies $\| \mathbf{T} \| = \mathcal{O}(\Delta x_{\max}^m)$. This form of discrete first derivative encompasses both compact and noncompact schemes.

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

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

where $\lambda(\mathcal{P})$ denotes an eigenvalue of $\mathcal{P}$, and c and C are real positive constants that are 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 that the standard second-order centered-difference operator with first-order differences at the boundaries is an SBP operator if we define

$$\mathcal{Q} = \frac{1}{2}\begin{bmatrix} -1 & 1 & & & & & \\ -1 & 0 & 1 & & & & \\ & -1 & 0 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -1 & 0 & 1 & \\ & & & & -1 & 1 & \end{bmatrix}$$

$$\mathcal{P} = \begin{bmatrix} \Delta x_0 & & & & \\ & \Delta x_1 & & & \\ & & \ddots & & \\ & & & \Delta x_N & \end{bmatrix}$$

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 toward the desired value. To introduce the method, we consider a constant-coefficient one-dimensional linear advection problem:

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

where a is the advection speed. Equation (3) is discretized on the mesh $\mathbf{x} = (x_0, x_1, \ldots, x_s, x_{s+1}, \ldots, x_N)^T$ with $N+1$ points and an interface at $x_s = x_{s+1}$ (see Fig. 1). For simplicity, we will apply the SATs at the interface only; the application of SATs for the boundary conditions is similar [6].

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

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

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

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

The terms appearing on the right-hand sides of Eqs. (4) and (5) are the penalty terms that couple the two domains. For smooth solutions, they introduce no truncation errors because the points coincide. Therefore, although the discrete solution may develop a discontinuity at the interface, this will not affect the global order of accuracy. As an aside, if we were applying a boundary condition at x_s rather than an interface condition, u_{s+1} would be replaced with u_{BC} in Eq. (4), in which u_{BC} is the appropriate boundary value of u.

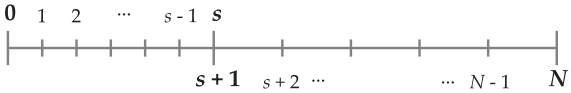


Fig. 1 Example domain consisting of two subdomains with an interface at $x_s = x_{s+1}$.

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

$$\begin{aligned}
&\mathbf{1}^T\mathcal{P}_L\partial_t\mathbf{u}_L + a\mathbf{1}^T\mathcal{Q}_L\mathbf{u}_L = \sigma_1(u_s - u_{s+1})\mathbf{1}^T\mathbf{e}_L\\
&\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{1}^T\mathcal{P}_L\mathbf{u}_L) + a\mathbf{1}^T(\mathcal{D}_L - \mathcal{Q}_L^T)\mathbf{u}_L = \sigma_1(u_s - u_{s+1})\\
&\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{1}^T\mathcal{P}_L\mathbf{u}_L) - a\mathbf{u}_L^T\mathcal{Q}_L\mathbf{1} = \sigma_1(u_s - u_{s+1}) - au_s\\
&\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{1}^T\mathcal{P}_L\mathbf{u}_L) = \sigma_1(u_s - u_{s+1}) - au_s
\end{aligned}$$

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

$$\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{1}^T\mathcal{P}_L\mathbf{u}_L + \mathbf{1}^T\mathcal{P}_R\mathbf{u}_R) = (u_s - u_{s+1})(\sigma_1 - \sigma_2 - a) \tag{6}$$

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

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

Relation (7) has also been shown to hold for nonlinear fluxes using a weak form of conservation [8].

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 Eq. (4) from the left by $\mathbf{u}_L^T$ and adding the transpose, we find

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{u}_L^T\mathcal{P}_L\mathbf{u}_L) + a\mathbf{u}_L^T(\mathcal{Q}_L + \mathcal{Q}_L^T)\mathbf{u}_L = 2\sigma_1(u_s^2 - u_su_{s+1})\\
&\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{u}_L^T\mathcal{P}_L\mathbf{u}_L) + a\mathbf{u}_L^T\mathcal{D}_L\mathbf{u}_L = 2\sigma_1(u_s^2 - u_su_{s+1})\\
&\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{u}_L^T\mathcal{P}_L\mathbf{u}_L) = 2\sigma_1(u_s^2 - u_su_{s+1}) - au_s^2
\end{aligned}$$

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

$$\frac{\mathrm{d}}{\mathrm{d}t}\|\mathbf{u}\|_{\mathcal{P}}^2 = \begin{pmatrix} u_s & u_{s+1}\end{pmatrix}\begin{bmatrix} 2\sigma_1 - a & -(\sigma_1 + \sigma_2)\\ -(\sigma_1 + \sigma_2) & 2\sigma_2 + a\end{bmatrix}\begin{pmatrix} u_s\\ u_{s+1}\end{pmatrix} \tag{8}$$

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

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

If we use Eq. (9) without modification, both penalty parameters vanish if $a = 0$. This can cause problems if the interface is located at a sonic or stagnation point; thus, in analogy with Swanson and Turkel's [30] matrix dissipation model, we propose limiting the penalty parameters as follows:

$$\sigma_1 = -\tfrac{1}{2}[\max(|a|, V) - a], \qquad \sigma_2 = -\tfrac{1}{2}[\max(|a|, V) + a] \tag{10}$$

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

3. *Quasi-One-Dimensional Euler Equations*

In this section, we review the use of SBP operators and SATs when 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

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

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

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

$$\begin{aligned}
&(\mathcal{P}_L \otimes \mathcal{I}_3)\partial_t\mathbf{q}_L + (\mathcal{Q}_L \otimes \mathcal{I}_3)\mathbf{f}_L - (\mathcal{P}_L \otimes \mathcal{I}_3)\mathbf{g}_L = \mathbf{\Sigma}_L\\
&(\mathcal{P}_R \otimes \mathcal{I}_3)\partial_t\mathbf{q}_R + (\mathcal{Q}_R \otimes \mathcal{I}_3)\mathbf{f}_R - (\mathcal{P}_R \otimes \mathcal{I}_3)\mathbf{g}_R = \mathbf{\Sigma}_R
\end{aligned} \tag{12}$$

where $\mathcal{I}_3$ is the 3×3 identity matrix and

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

The subscript denotes the location of the variable or flux [e.g., $\hat{\mathbf{Q}}_i = \hat{\mathbf{Q}}(x_i)$]. The operator $\otimes$ denotes the Kronecker product for matrices: if $A \in \mathcal{M}^{m\times n}$ and $B \in \mathcal{M}^{p\times q}$, then $C = A \otimes B \in \mathcal{M}^{mp\times nq}$ is defined by

$$C_{p(i-1)+k,q(j-1)+l} = A_{ij}B_{kl}$$

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

$$\mathbf{\Sigma}_L = \begin{pmatrix}\mathbf{0}\\ \mathbf{0}\\ \vdots\\ -\frac{1}{2}(|A| - A)(\hat{\mathbf{Q}}_s - \hat{\mathbf{Q}}_{s+1})\end{pmatrix}$$

$$\mathbf{\Sigma}_R = \begin{pmatrix}-\frac{1}{2}(|A| + A)(\hat{\mathbf{Q}}_{s+1} - \hat{\mathbf{Q}}_s)\\ \mathbf{0}\\ \vdots\\ \mathbf{0}\end{pmatrix}$$

where A is the flux Jacobian matrix at an averaged state; for the present work, we use the simple average $\frac{1}{2}(\mathbf{Q}_s + \mathbf{Q}_{s+1})$. 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

$$\lambda_1 = \max(|u + a|, V_n\rho(A)), \qquad \lambda_2 = \max(|u - a|, V_n\rho(A))$$
$$\lambda_3 = \max(|u|, V_l\rho(A))$$

where $\rho(A)$ denotes the spectral radius of A, and 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 omitted numerical dissipation in Eq. (12) to focus on the SATs. In the present work, we use the Jameson–Schmidt–Turkel scalar dissipation model [31,32] with second- and fourth-difference dissipation; however, the discretization can easily be adapted to use matrix dissipation [30] or upwinding [10].

Mattsson et al. [10] have shown that the dissipation based on the standard difference operators does not produce an energy estimate. Nevertheless, we did not experience problems when using the standard dissipation operators in conjunction with SATs for steady-state flows.

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 equation (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

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

Newton's method will converge quadratically, provided that the initial iterate $\mathbf{q}^{(0)}$ is sufficiently close to the solution of Eq. (13) [33]. 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) The first phase is an approximate-Newton startup phase for which the objective is to find a suitable initial iterate as efficiently as possible.

2) The second phase is an inexact-Newton phase, which uses the initial iterate and a slightly modified form of Eq. (14).

Both of these phases include a number of parameters. In most cases, these parameters are bounded by robustness on one hand and by CPU time on the other hand. The parameter values we present next have been validated for a range of cases and provide a good compromise between these two objectives. For the results in Sec. IV, the parameters are fixed unless otherwise stated.

1. Approximate-Newton Phase

The approximate-Newton phase uses a form of pseudotransient continuation to find the initial iterate [24,32,34]. This strategy is similar to discretizing the unsteady equation (1) with the implicit Euler time-marching scheme. However, because 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 startup [5,34] 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 startup. 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 [5]. Lumping the dissipation coefficients in this way produces a modified Jacobian that is only first-order-accurate, but this does not affect the accuracy of the steady solution. We will use $\mathcal{A}_1$ to denote the first-order Jacobian, to emphasize its accuracy and distinguish it from the exact Jacobian.

The first-order Jacobian is factored using block ILU with a level of fill p to produce the preconditioner. Factoring the matrix is one of the most expensive tasks required by the algorithm. The approximate-Newton phase often requires many outer iterations, and 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 [35]. Let m be the number of outer iterations between Jacobian updates. Then $\mathcal{A}_1$ is updated and factored on iteration n if $\mathrm{mod}(n, m) = 0$. Values $m = \{3, 4, 5\}$ provide a good compromise between CPU time and robustness [36].

Finally, a spatially varying time step has been shown to improve the rate of convergence for schemes based on approximate factorizations [32] as well as Newton–Krylov algorithms [5,37]. The time step used in the current work for node (j, k, m) is

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

This time step roughly approximates a constant CFL. The appearance of the first J in the denominator of Eq. (15) is due to the use of $\mathbf{Q}$ rather than $\hat{\mathbf{Q}}$ in the column vector $\mathbf{q}$ [see Eq. (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 ($\lfloor x \rfloor$ gives the largest integer less than or equal to x); this operator ensures that updates to $\Delta t^{(n)}_{\mathrm{ref}}$ are consistent with the update period m. Values for a and b used in the present work are $a = 0.1$ and $b = 1.7$.

To summarize, during startup we replace Eq. (14) with the approximate-Newton update equation:

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

where

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

$n^* = m\lfloor n/m \rfloor$, 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 discussed further in Sec. III.B.

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 [5,37]:

$$\frac{\|\boldsymbol{\mathcal{F}}^{(n)}\|_2}{\|\boldsymbol{\mathcal{F}}^{(0)}\|_2} \equiv R_d^{(n)} \leq \tau$$

For the Euler equations, $\tau = 0.1$ is usually sufficient and has been used for all of the results presented in this paper.

3. Inexact-Newton Phase

As with the startup 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 [38]:

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

where, for the present results, we 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 $n_{\text{Newt}} - 1$ is the last approximate-Newton iteration, then

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

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

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

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

$$\mathcal{A}^{(n)}\mathbf{v} \approx \frac{\boldsymbol{\mathcal{F}}(\mathbf{q}^n + \epsilon\mathbf{v}) - \boldsymbol{\mathcal{F}}(\mathbf{q}^n)}{\epsilon} \tag{18}$$

The perturbation parameter must be chosen carefully to minimize truncation error and avoid round-off errors [39]. For this work, we used [14]

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

where $\delta = 10^{-13}$ and N is the number of unknowns. In light of approximation (18), we do not need to compute or store the Jacobian matrix. We do compute and store the first-order Jacobian $\mathcal{A}_1$ at the beginning of each inexact-Newton iteration, because this smaller matrix is needed to build the preconditioner.

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

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

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 the number of outer iterations will increase.

For this work, the forcing parameter is gradually decreased from its startup value of 0.5 to 0.01 using a safeguard proposed by Eisenstat and Walker [40]:

$$\eta_n = \max\{0.01, \eta_{n-1}^{(1+\sqrt{5})/2}\} \tag{19}$$

We found that formula (19) helps avoid oversolving during the early iterations of the inexact-Newton phase. Eisenstat and Walker [40] also give an adaptive formula for the forcing parameter that allows q-superlinear convergence. Although their adaptive η formula reduces the number of Newton iterations, the CPU and memory costs do not warrant the additional Krylov iterations.

B. Solving The Distributed Linear System

During both the startup and Newton phases we use a Krylov solver [for example, the generalized minimal residual method (GMRES) [41]) to inexactly solve sparse systems of the form

$$A\mathbf{x} = \mathbf{b} \tag{20}$$

To solve the preceding 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, one or more blocks are assigned to a process. For a given process i, three types of unknowns can be identified for linear system (20):

1) Internal unknowns appear only in equations on the process i.

2) Internal-interface unknowns are assigned to process i but are coupled to unknowns on another process $j \neq i$.

3) External-interface unknowns are assigned to other subdomains but appear in equations on process i.

For example, when the Euler equations are discretized using SATs, the internal- and external-interface unknowns correspond to nodes that 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

$$A_i\mathbf{x}_{(i)} + E_i\mathbf{y}_{(i,\text{ext})} = \mathbf{b}_{(i)} \tag{21}$$

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 Fig. 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 [26].

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 Eqs. (21) on process i take the form

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix}\begin{pmatrix} \mathbf{u}_{(i)} \\ \mathbf{y}_{(i)} \end{pmatrix} + \begin{pmatrix} 0 \\ \sum\limits_{j\in N_i} E_{ij}\mathbf{y}_{(j)} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_{(i)} \\ \mathbf{g}_{(i)} \end{pmatrix} \tag{22}$$

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

When using a Krylov-subspace method to solve the distributed system (20), 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()`].

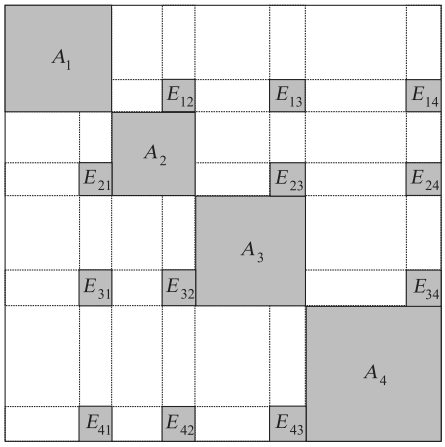


Fig. 2 Sparsity structure of the global system matrix with unknowns grouped by subdomain and internal-interface variables listed last; sparse submatrices with nonzero elements are shaded.

Parallelizing the matrix-vector products and the preconditioner requires more care.

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 nonblocking communication of the interface variables [42]. 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.

Preconditioners are the most critical component of an efficient parallel linear solver. Although excellent serial preconditioners exist for Newton–Krylov flow solvers [5,16,17], these preconditioners cannot be implemented efficiently in parallel; for example, although 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:

$$L_i U_i = [\mathcal{T}^{(n)} + \mathcal{A}_1^{(n)}]_i + R_i$$

where R_i is the error in the factorization. A block version of incomplete lower/upper factorization with a level of fill p [ILU(p)] [43] is used to obtain the factorization $L_i U_i$. The blocks are composed of the flow unknowns at each node. Notice that the factorization itself does not require interprocessor communication, because block ILU(p) [BILU(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 [44]). Given the vector w, the local component of the preconditioned vector z is given by the exact or inexact solution to the system:

$$A_i z_i = w_i \tag{23}$$

Equation (23) can be solved using a direct method or iteratively using, for example, GMRES. For this work, we solve Eq. (23) approximately using a single application of the ILU factorization: $z_i = U_i^{-1} L_i^{-1} w_i$. We investigated the use of preconditioned GMRES to solve Eq. (23), 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. [45] suggested that domain overlap, although 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 [26] proposed a preconditioning technique based on an approximate factorization of the Schur-complement system. Their contribution is summarized next.

Considering Eq. (22), we see that the internal variables can be written as

$$\mathbf{u}_{(i)} = B_i^{-1}(\mathbf{f}_{(i)} - F_i \mathbf{y}_{(i)}) \tag{24}$$

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

$$S_i \mathbf{y}_{(i)} + \sum_{j \in N_i} E_{ij} \mathbf{y}_{(j)} = \mathbf{g}_{(i)} - E_i B_i^{-1} \mathbf{f}_{(i)} \equiv \mathbf{g}'_{(i)} \tag{25}$$

where $S_i = C_i - E_i B_i^{-1} F_i$ is the *local* Schur-complement matrix [26]. Assembling all of the local Schur-complement systems for each process, we obtain a linear system for all of the internal-interface unknowns:

$$\underbrace{\begin{pmatrix} S_1 & E_{12} & \dots & E_{1P} \\ E_{21} & S_2 & \dots & E_{2P} \\ \vdots & & \ddots & \vdots \\ E_{P1} & E_{P2} & \dots & S_P \end{pmatrix}}_{S} \begin{pmatrix} \mathbf{y}_{(1)} \\ \mathbf{y}_{(2)} \\ \vdots \\ \mathbf{y}_{(P)} \end{pmatrix} = \begin{pmatrix} \mathbf{g}'_{(1)} \\ \mathbf{g}'_{(2)} \\ \vdots \\ \mathbf{g}'_{(P)} \end{pmatrix} \tag{26}$$

The coefficient matrix S appearing in Eq. (26) is the Schur complement [44]. Following Saad and Sosonkina [26], 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 system (26), and then solve for the local internal unknowns on each process using Eq. (24). In practice, however, forming the Schur-complement matrix and solving for the interface unknowns exactly is not competitive with other methods [26]. Instead, we consider systems that approximate Eq. (26) and act as preconditioners for the global system (20).

Consider the following block factorization of A_i:

$$A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} = \begin{pmatrix} B_i & 0 \\ E_i & S_i \end{pmatrix} \begin{pmatrix} I & B_i^{-1} F_i \\ 0 & I \end{pmatrix} \tag{27}$$

Next, suppose that A_i has been factored instead into $A_i = L_i U_i$, where

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

Comparing the factors in Eq. (27) with L_i and U_i, we can show that [26]

$$S_i = L_{S_i} U_{S_i} \tag{28}$$

Thus, we can obtain a lower/upper (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,

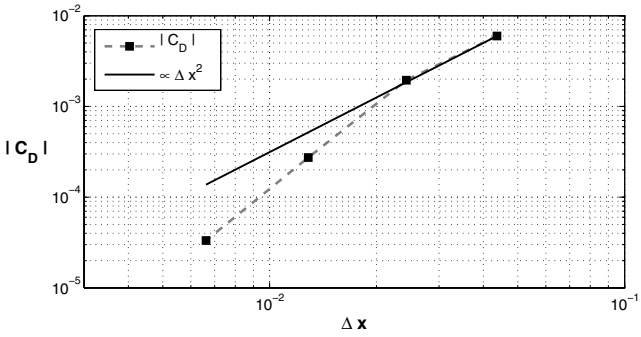


Fig. 3 Coefficient of drag as a function of nominal mesh spacing for a subsonic zero-angle-of-attack flow.

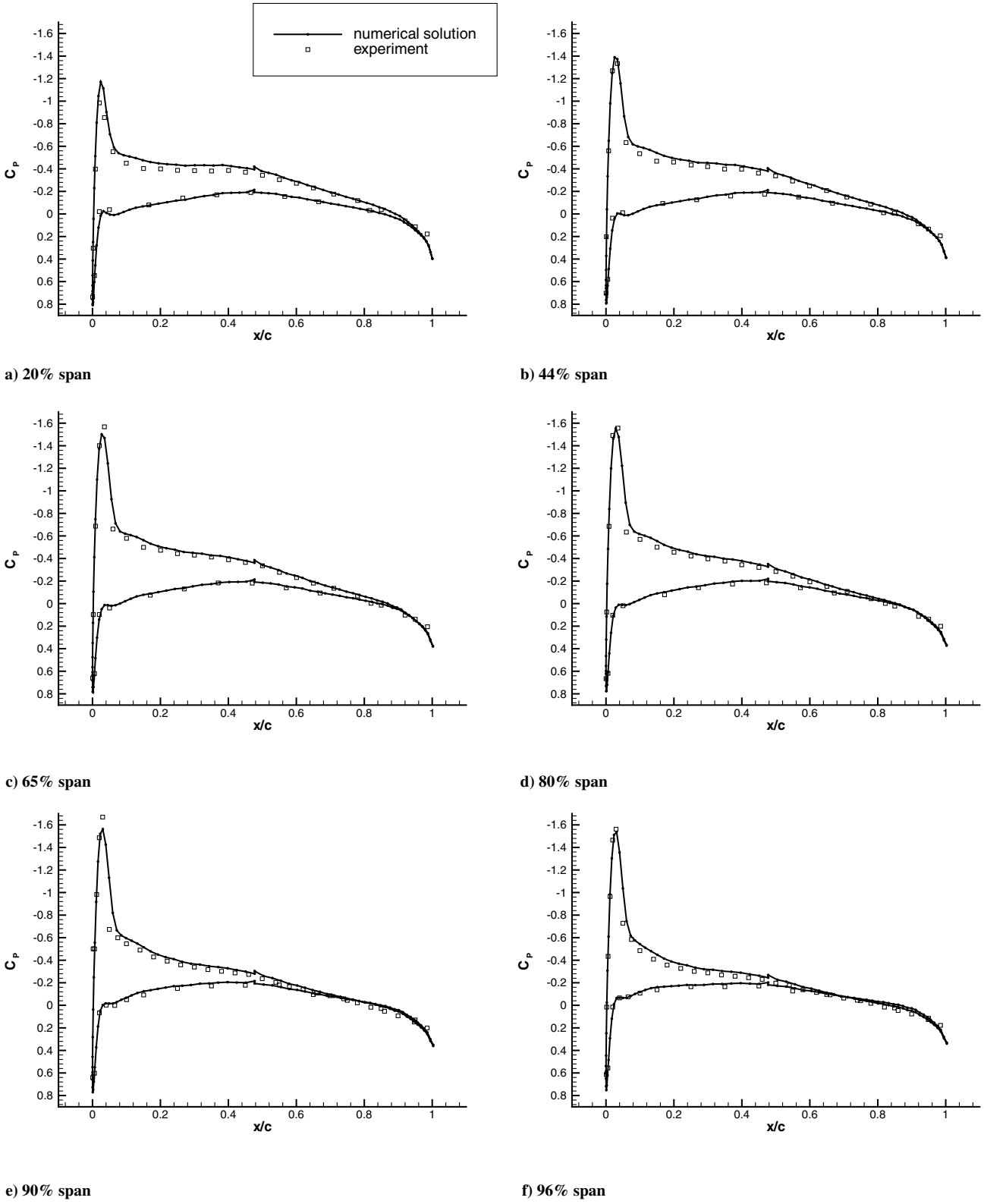
we can obtain an approximate factorization of S_i by extracting the relevant blocks from the ILU factorization of A_i.

Suppose that 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 Eqs. (25) and (28):

$$\mathbf{y}_{(i)} = U_{S_i}^{-1} L_{S_i}^{-1} \left(\mathbf{g}'_{(i)} - \sum_{j \in N_i} E_{ij} \mathbf{y}_{(j)} \right) \tag{29}$$

The preceding 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 [44]. Once the approximations to the $\mathbf{y}_{(i)}$ have been exchanged, we can substitute the $\mathbf{y}_{(i,\text{ext})}$ into Eq. (21) and apply the ILU factorization to obtain approximate values for $\mathbf{x}_{(i)}$.

We implemented an approximate Schur preconditioner that is closely based on Algorithm 3.1 of Saad and Sosonkina [26]. Two important clarifications to the original algorithm involve the



a) 20% span **b) 44% span**

c) 65% span **d) 80% span**

e) 90% span **f) 96% span**

Fig. 4 Comparison of the coefficient of pressure at a Mach number of 0.699 around the ONERA M6 wing.

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 our algorithm is equivalent mathematically, but it is approximately 20% more efficient computationally. The relevant modifications can be found on lines 1, 22, and 23 of algorithm A1 in Appendix A. These modifications, although 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

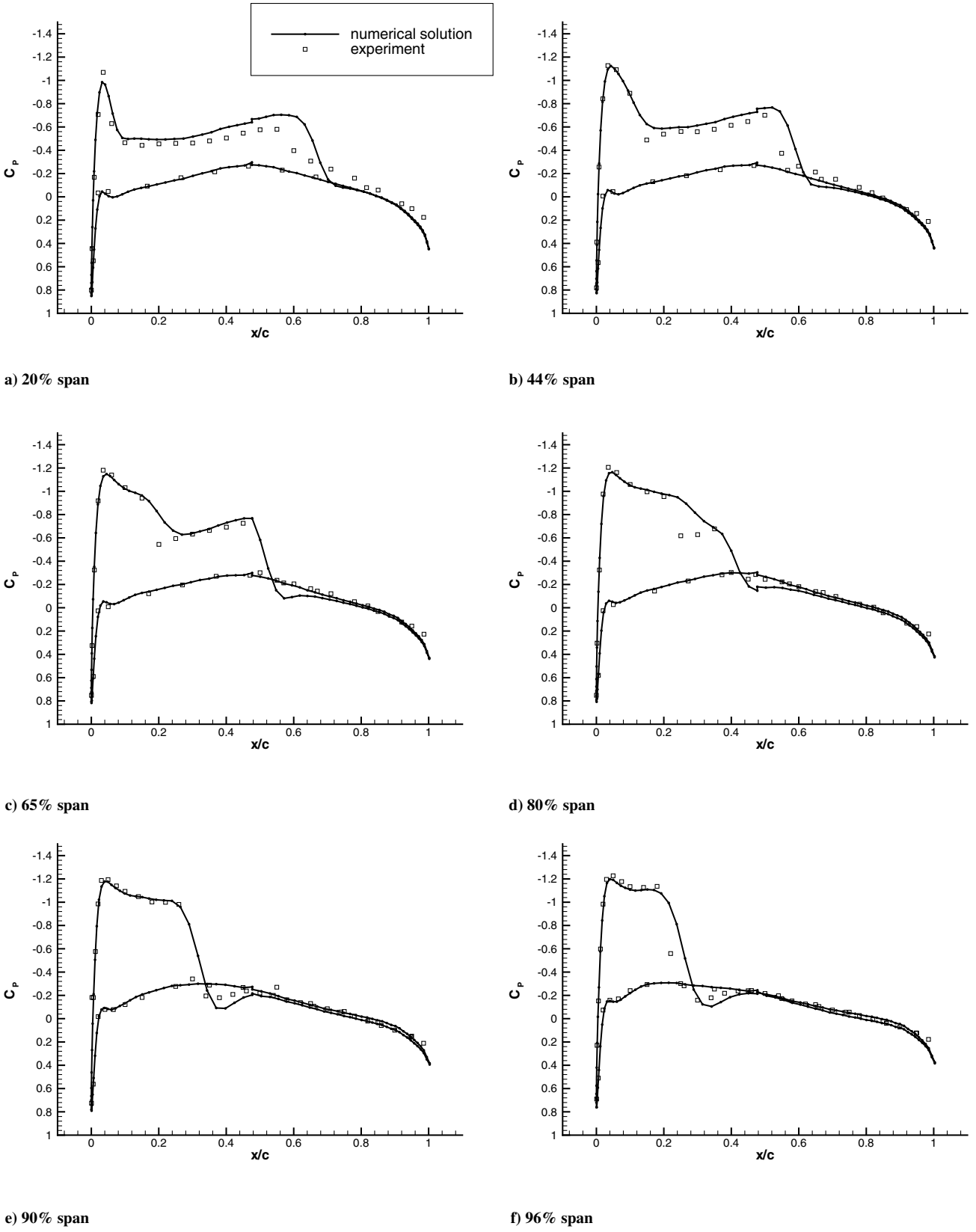
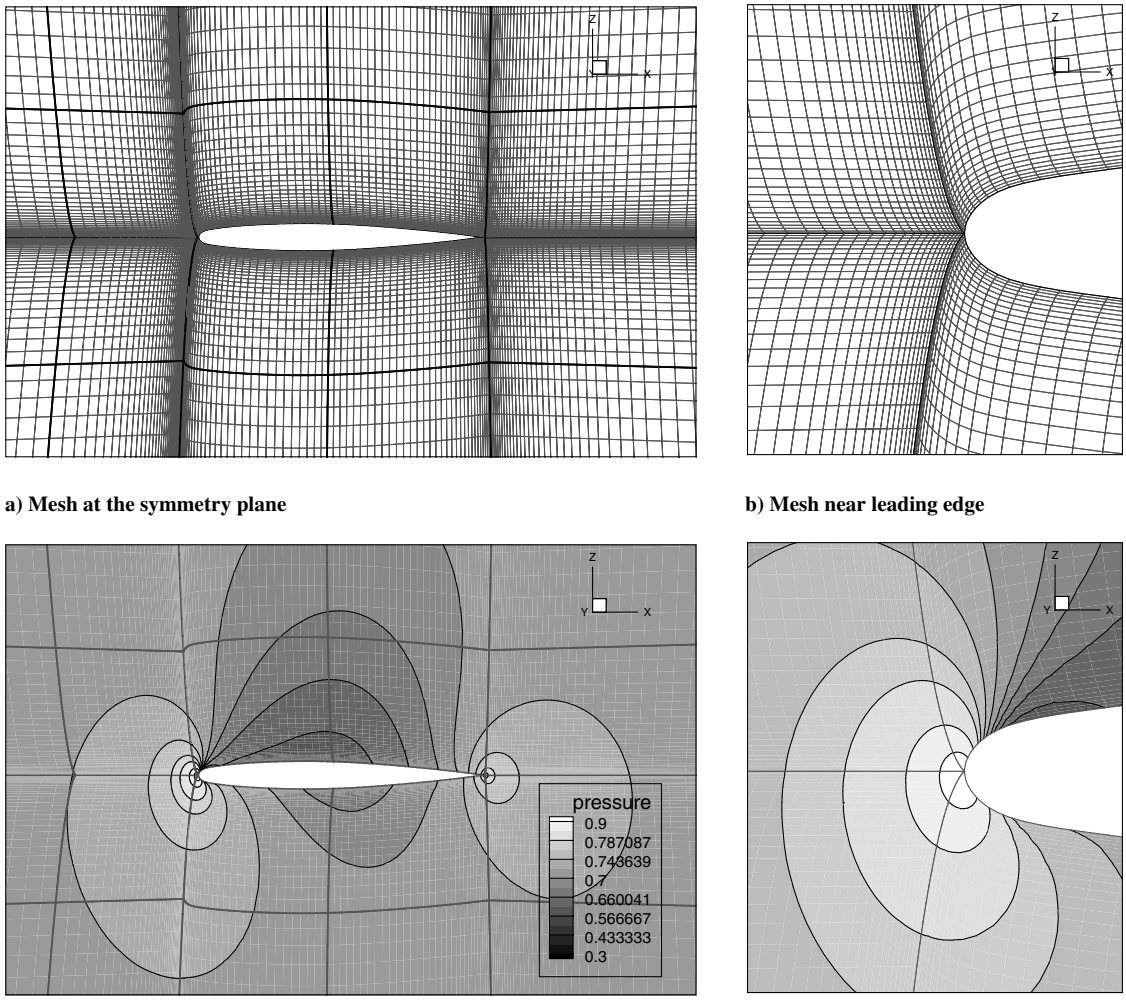


Fig. 5 Comparison of the coefficient of pressure at a Mach number of 0.84 around the ONERA M6 wing.



a) Mesh at the symmetry plane

b) Mesh near leading edge

c) Pressure contours at the symmetry plane

d) Contours near leading edge

Fig. 6 Mesh and pressure contours at the symmetry plane of the ONERA M6 wing; Mach number of 0.699.

reason, we use flexible GMRES (FGMRES) [46] 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

We obtained all results on a symmetric multiprocessor HP Itanium Beowulf-class cluster. Each node on the cluster consists of 4 Itanium 2 processors with 6 MB L3 cache and a clock speed of 1500 MHz. The nodes each have at least 8 GB of RAM and are connected with a high-bandwidth low-latency Myrinet network. We used only 1 processor per node to eliminate memory contention observed with this architecture [36].

The results presented in the following subsections have been obtained on grids for the ONERA M6 wing. All grids use an H–H 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 will consider block decomposition as a means of load-balancing arbitrarily structured grids.

A. Discretization Assessment

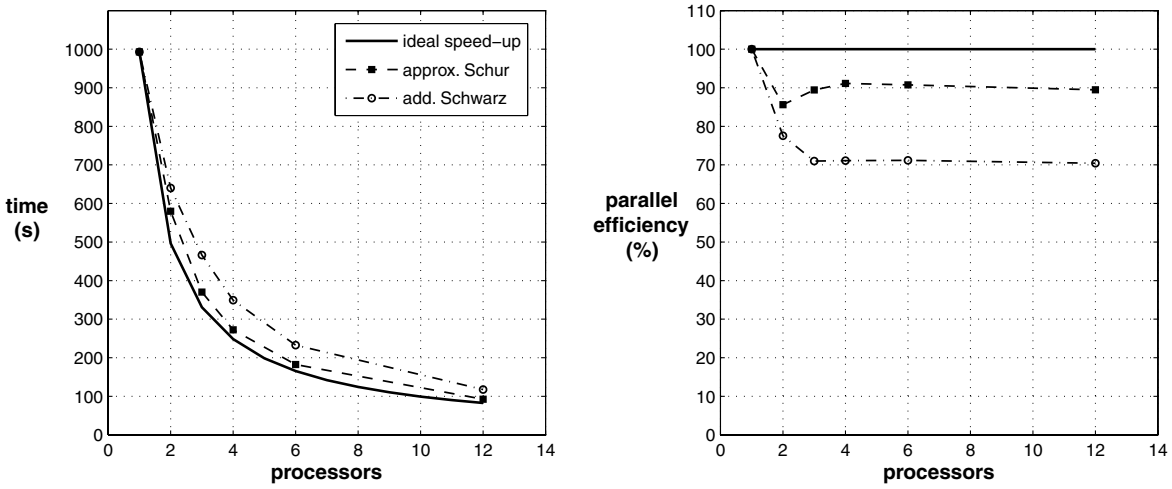
The use of SATs in computational aerodynamics is not common, and so a demonstration of solution accuracy and convergence is warranted. We assess the SAT discretization using four grids for the ONERA M6 wing. Each grid contains 96 blocks, and the finest grid has $33 \times 33 \times 33$ nodes on each block. Successively coarser grids are obtained by removing every other node in each direction from the next finer grid; hence, the coarser grids have $17 \times 17 \times 17$ nodes, $9 \times 9 \times 9$ nodes, and $5 \times 5 \times 5$ nodes per block.

We use the coefficient of drag to establish the order of accuracy of the scheme: recall that for a symmetric inviscid subsonic flow, C_D vanishes. Figure 3 plots $|C_D|$ as a function of nominal mesh spacing defined by $\Delta x \equiv N^{-\frac{1}{3}}$, where N is the total number of nodes. We calculated the coefficient of drag resulting from a far-field flow with a Mach number of 0.5 and zero angle of attack; the ONERA M6 has symmetric sections. Figure 3 demonstrates that the discretization is at least second-order-accurate, as expected.

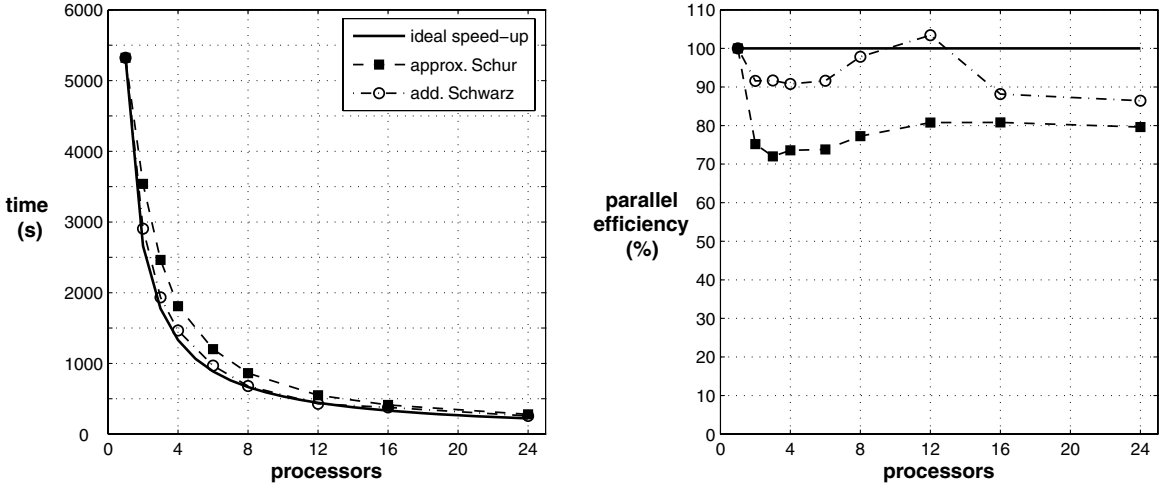
Validating the code using experimental results is also useful, although only qualitative comparisons can be made using the inviscid code. We obtained flow solutions at Mach numbers of 0.699 and 0.84 on the finest grid (96 blocks with 3.45×10^6 nodes). The angle of attack was fixed at 3.06 deg. Figures 4 and 5 compare our results with the experimental results of Schmitt and Charpin [47]. Discontinuities in the solutions are visible near the midchord and mark the location of block interfaces. As discussed earlier, such discontinuities are permitted by the SAT methodology and do not affect the stability or the order of accuracy, as shown in Fig. 3.

Consequently, any dependence the solution has on the block topology is on the order of the discretization.

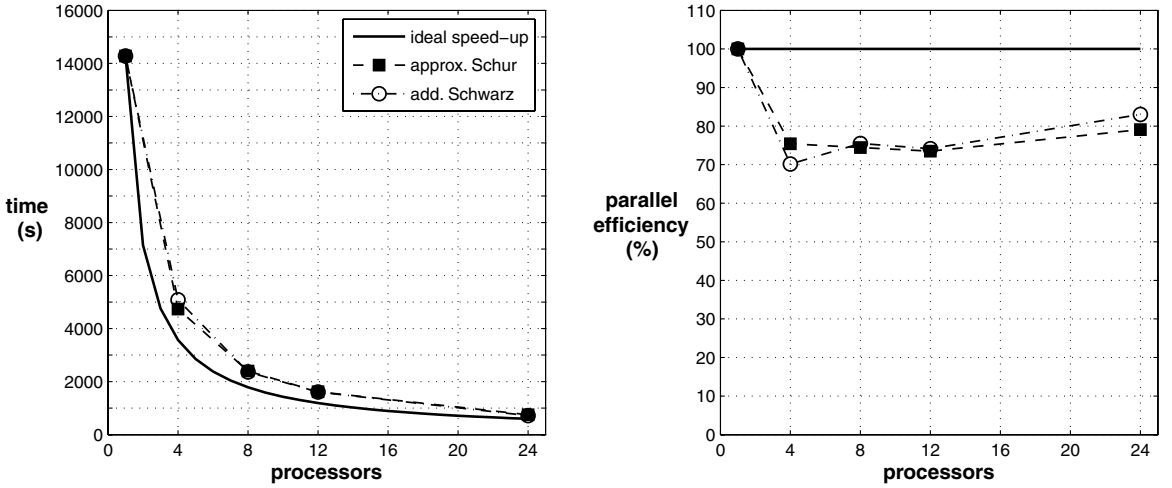
Recall that the SBP–SAT methodology does not require slope continuity of grid lines at block interfaces. Indeed, the preceding grids used were obtained using transfinite interpolation without elliptic smoothing. Figure 6 shows the symmetry plane of the finest mesh and the pressure contours at a Mach number of 0.699 obtained on that mesh. A close-up of the leading edge shows the slope discontinuities that the scheme can handle.



a) CPU time and parallel efficiency on the 12 block, 1.46×10^5 node grid



b) CPU time and parallel efficiency on the 48 block, 4.58×10^5 node grid



c) CPU time and parallel efficiency on the 96 block, 1.168×10^6 node grid

Fig. 7 CPU time (in seconds) and parallel efficiency of the approximate Schur and additive Schwarz preconditioners.

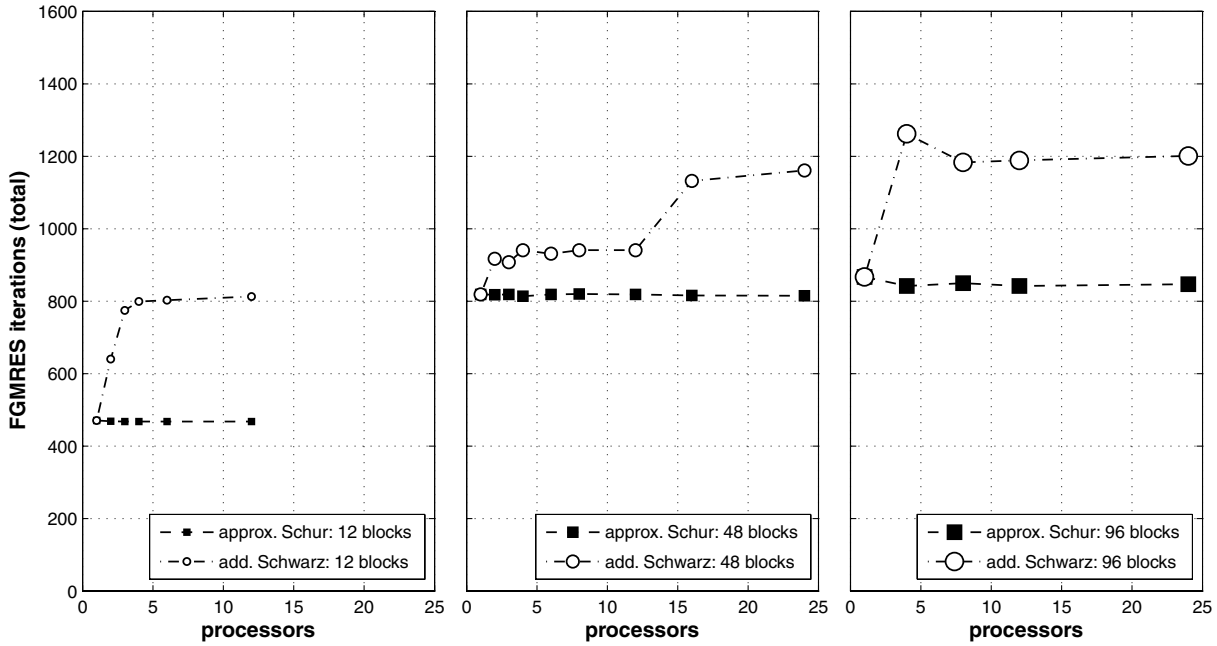


Fig. 8 Total number of FGMRES iterations as a function of the number of processors for the 12-, 48-, and 96-block grids in Table 1.

B. Preconditioner Comparison

To test the parallel efficiency of the preconditioners, flow solutions were timed for the grids listed in Table 1. The grids allow us to evaluate the preconditioners for different numbers of blocks and mesh sizes on the same ONERA M6 geometry.

The operating conditions were fixed at a Mach number of 0.699 and an angle of attack of 3.06 deg. The discrete equations were solved to a relative tolerance of 10^{-10} using the Newton–Krylov algorithm. The linear subproblems were solved using FGMRES limited to 60 iterations for the 12-block grid and 80 iterations for the 48- and 96-block grids. During the approximate-Newton phase, a Jacobian update period of $m = 5$ was used for the 12-block grid and a period of $m = 4$ was used for the two larger grids.

Figures 7a–7c compare the CPU time and parallel efficiency of the two preconditioners. For serial computations, note that both preconditioners reduce to BILU(1). The results demonstrate that the best preconditioner is grid-dependent, although both preconditioners scale well on all three grids.

Some remarks are necessary regarding the superlinear speedup for the 12-processor additive Schwarz time observed in Fig. 7b. Both parallel preconditioners change as the number of processors increases; therefore, the solution algorithm with N processors is not strictly the same as the serial algorithm. The superlinear speedup in Fig. 7b would not appear if we had used an identical block Jacobi preconditioner for the serial computation. This leads to another question: why is global BILU(1) worse than additive Schwarz in this case? One possible explanation could be the additional cache misses experienced in the serial computation. Another explanation is that ILU is sensitive to the ordering of unknowns [18], and so the 12-processor case may have a better ordering.

The preconditioners can also be assessed by comparing the total number of FGMRES iterations used during a flow solution. Figure 8 shows that the iterations required by the approximate Schur preconditioner are independent of the number of processors for the range of processors considered here. In contrast, the additive Schwarz preconditioner requires more iterations as more processors are added. This difference reflects the implicit coupling between domains achieved by the approximate Schur preconditioner.

Table 1 ONERA M6 grids and their dimensions used to evaluate the preconditioners

Blocks	Dimensions ($N_j \times N_k \times N_m$)	Grid size (nodes)
96	$23 \times 23 \times 23$	1,168,032
48	$33 \times 17 \times 17$	457,776
12	$23 \times 23 \times 23$	146,004

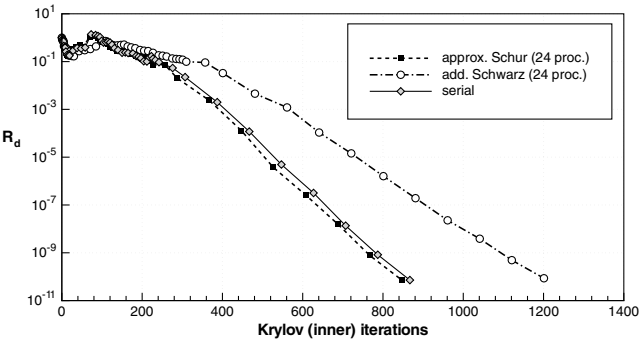


Fig. 9 Relative residual R_d versus the number of Krylov iterations; symbols denote the outer iterations; results for the 96-block 1.168×10^6 node grid, Mach number of 0.699, and angle of attack of 3.06 deg.

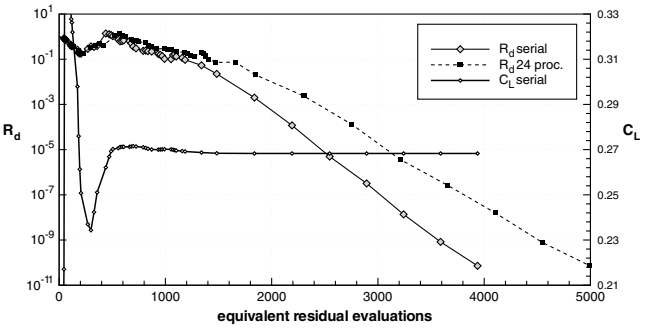


Fig. 10 Relative residual R_d and C_L versus equivalent residual evaluations; 24-processor residual history corresponds to the approximate Schur preconditioner; results for the 96-block 1.168×10^6 node grid with a Mach number of 0.699 and angle of attack of 3.06 deg.

Although the approximate Schur preconditioner requires fewer Krylov iterations, the results suggest that, on average, both preconditioners are comparable in terms of CPU time. The Schwarz preconditioner is preferable if memory considerations are important, because the Schur preconditioner requires FGMRES, whereas the Schwarz preconditioner can use GMRES (see the discussion at the end of Sec. III.C.2).

C. Algorithm Performance

We evaluate the performance of the algorithm from two perspectives. First, we consider the number of Krylov iterations needed to reduce the relative residual R_d by 10 orders. We hope this measure will be useful for other developers of Newton–Krylov solvers. Second, we plot the residual convergence versus equivalent residual evaluations, a form of normalized CPU time that offers an architecture-independent comparison with other flow solvers.

The relative residual is plotted versus Krylov iterations in Fig. 9. We define a Krylov, or inner, iteration as the application of a matrix-vector product with preconditioning. The symbols in the figure denote the outer (Newton) iterations. Residual histories are shown for serial and parallel computations; for the parallel runs, we used both preconditioners and 24 processors. The flow solution is for the 96-block 1.168×10^6 node grid, and the far-field Mach number and angle of attack are 0.699 and 3.06 deg, respectively. The figure shows that for the inexact-Newton phase, the maximum number of allowable matrix-vector products (80) are used during each outer iteration. As a result, we obtain linear convergence. As noted earlier, it is possible to achieve superlinear convergence using a smaller tolerance and more matrix-vector products; however, we find that superlinear convergence increases CPU time.

Although counting inner iterations is useful for comparing with other Krylov solvers, this measure ignores preconditioner factorization time and the change in matrix-vector products [i.e., from explicit to the matrix-free Eq. (18)]. For more general comparisons, we include Fig. 10, which plots R_d versus equivalent residual evaluations. An equivalent residual evaluation is the CPU time needed to compute the residual equation $\boldsymbol{\mathcal{F}}$, which includes all flux and SAT evaluations. We use the same grid and flow conditions as in Fig. 9. The residual history for the serial and 24-processor cases are shown. We used the Schur preconditioner for the 24-processor case, but the results for the Schwarz preconditioner are similar (see Fig. 7c). The coefficient of lift obtained at each outer iteration of the serial computation is also included in Fig. 10. The history of C_L for 24 processors is similar. The algorithm obtains 3-digit accuracy in C_L in 1483 and 1845 equivalent residual evaluations for the serial and 24-processor runs, respectively.

V. Conclusions

We described a second-order SBP–SAT solver for the Euler equations. The SBP–SAT discretization can accommodate grids with C^0 continuity at the block interfaces, and no special treatment of the points at block edges and vertices is required. The discrete equations are solved using a parallel Newton–Krylov approach. We compared two preconditioners for the Krylov iterative solver: an additive Schwarz and an approximate Schur preconditioner.

We draw the following conclusions:

1) 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.

2) For the Euler equations, the approximate Schur preconditioner outperforms the additive Schwarz preconditioner in terms of Krylov iterations, but the preconditioners have similar CPU time requirements.

3) Because the approximate Schur preconditioner must use a flexible iterative solver (e.g., FGMRES), which may require more memory, the Schwarz preconditioner is a good choice when memory is limited.

Appendix A: Pseudocode for Approximate Schur Preconditioner

Algorithm A1 Approximate Schur preconditioner

Data: m, η, $\mathbf{u}$
Results: $\mathbf{v}$
Get the residual of the reduced system for a guess of zero: $\mathbf{v} = U_{S_i}^{-1} L_i^{-1} \mathbf{u}$.
Project onto the space of internal-interface unknowns using P: $\mathbf{w}_{(i)}^{(1)} = P\mathbf{v}$.
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: $\mathbf{w}_{(i,\text{ext})}^{(j)}$.
  Perform the external matrix-vector product: $\mathbf{w}_{(i)}^{(j+1)} = E_i \mathbf{w}_{(i,\text{ext})}^{(j)}$.
  Apply the diagonal block of the Schur complement: $\mathbf{w}_{(i)}^{(j+1)} \leftarrow U_{S_i}^{-1} L_{S_i}^{-1} \mathbf{w}_{(i)}^{(j+1)}$.
  Finish the matrix-vector product [Eq. (29)]: $\mathbf{w}_{(i)}^{(j+1)} \leftarrow \mathbf{w}_{(i)}^{(j)} + \mathbf{w}_{(i)}^{(j+1)}$.
  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: $\mathbf{y}_{(i,\text{ext})}$.
Update the internal interface only: $\mathbf{v} \leftarrow \mathbf{v} + P^T P(\mathbf{u} - E_i \mathbf{y}_{(i,\text{ext})} - \mathbf{v})$.
Apply forward solve to the interface and then backsolve: $\mathbf{v} \leftarrow U_i^{-1}[I + P^T P(L_i^{-1} - I)]\mathbf{v}$.

Acknowledgments

The authors gratefully acknowledge financial assistance from the Natural Sciences and Engineering Research Council (NSERC), the Canada Research Chairs program, Mathematics of Information Technology and Complex Systems (MITACS), Bombardier Aerospace, and the University of Toronto. The first author would like to thank Edwin van der Weide and Magnus Svärd for their generous help explaining their implementation of simultaneous-approximation terms.

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 Paper 2008-0930, Reno, NV, Jan. 2008.

[2] Agarwal, R., "Computational Fluid Dynamics of Whole-Body Aircraft," *Annual Review of Fluid Mechanics*, Vol. 31, 1999, pp. 125–169. doi:10.1146/annurev.fluid.31.1.125

[3] 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. doi:10.2514/2.1472

[4] 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. doi:10.2514/1.10415

[5] Nichols, J., and Zingg, D. W., "A Three-Dimensional Multiblock Newton-Krylov Flow Solver for the Euler Equations," 17th AIAA

Computational Fluid Dynamics Conference, AIAA Paper 2005-5230, Toronto, June 2005.

[6] 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*, Vol. 111, No. 2, 1994, pp. 220–236. doi:10.1006/jcph.1994.1057

[7] Hesthaven, J. S., "A Stable Penalty Method for the Compressible Navier-Stokes Equations, 3: Multidimensional Domain Decomposition Schemes," *SIAM Journal on Scientific Computing*, Vol. 20, No. 1, 1998, pp. 62–93. doi:10.1137/S1064827596299470

[8] Carpenter, M. H., Nordström, J., and Gottlieb, D., "A Stable and Conservative Interface Treatment of Arbitrary Spatial Accuracy," *Journal of Computational Physics*, Vol. 148, No. 2, 1999, pp. 341–365. doi:10.1006/jcph.1998.6114

[9] Nordström, J., and Carpenter, M. H., "High-Order Finite Difference Methods, Multidimensional Linear Problems, and Curvilinear Coordinates," *Journal of Computational Physics*, Vol. 173, No. 1, 2001, pp. 149–174. doi:10.1006/jcph.2001.6864

[10] Mattsson, K., Svärd, M., and Nordström, J., "Stable and Accurate Artificial Dissipation," *Journal of Scientific Computing*, Vol. 21, No. 1, 2004, pp. 57–79. doi:10.1023/B:JOMP.0000027955.75872.3f

[11] Hesthaven, J. S., and Gottlieb, D., "A Stable Penalty Method for the Compressible Navier-Stokes Equations, 1: Open Boundary Conditions," *SIAM Journal on Scientific Computing*, Vol. 17, No. 3, 1996, pp. 579–612. doi:10.1137/S1064827594268488

[12] 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*, Vol. 148, No. 2, 1999, pp. 621–645. doi:10.1006/jcph.1998.6133

[13] 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*, Vol. 87, Nos. 2–3, 1991, pp. 281–304. doi:10.1016/0045-7825(91)90009-U

[14] 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.

[15] Anderson, W. K., Rausch, R. D., and Bonhaus, D. L., "Implicit/Multigrid Algorithms for Incompressible Turbulent Flows on Unstructured Grids," *Journal of Computational Physics*, Vol. 128, No. 2, 1996, pp. 391–408. doi:10.1006/jcph.1996.0219

[16] Pueyo, A., and Zingg, D. W., "Efficient Newton–Krylov Solver for Aerodynamic Computations," *AIAA Journal*, Vol. 36, No. 11, Nov. 1998, pp. 1991–1997. doi:10.2514/2.326

[17] Blanco, M., and Zingg, D. W., "Fast Newton–Krylov Method for Unstructured Grids," *AIAA Journal*, Vol. 36, No. 4, April 1998, pp. 607–612. doi:10.2514/2.412

[18] Chisholm, T. T., and Zingg, D. W., "A Newton–Krylov Algorithm for Turbulent Aerodynamic Flows," 39th AIAA Aerospace Sciences Meeting and Exhibit, AIAA Paper 2003-0071, Reno, NV, 2003.

[19] 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.

[20] 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, AIAA Paper 95-0221, Reno, NV, 1995.

[21] 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.

[22] 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*, DDM.org, Greenwich, England, U.K., 1998, pp. 513–520.

[23] Groth, C. P., and Northrup, S. A., "Parallel Implicit Adaptive Mesh Refinement Scheme for Body-Fitted Multiblock Mesh," 17th AIAA Computational Fluid Dynamics Conference, AIAA Paper 2005-5333, Toronto, June 2005.

[24] Knoll, D., and Keyes, D., "Jacobian-Free Newton-Krylov Methods: A Survey of Approaches and Applications," *Journal of Computational Physics*, Vol. 193, No. 2, 2004, pp. 357–397. doi:10.1016/j.jcp.2003.08.010

[25] 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*, Vieweg Verlag, Braunschweig, Germany, 1993, pp. 17–30.

[26] Saad, Y., and Sosonkina, M., "Distributed Schur Complement Techniques for General Sparse Linear Systems," *SIAM Journal on Scientific Computing*, Vol. 21, No. 4, 1999, pp. 1337–1357. doi:10.1137/S1064827597328996

[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, Academic Press, New York, 1974.

[28] Strand, B., "Summation by Parts for Finite Difference Approximations for d/dx," *Journal of Computational Physics*, Vol. 110, No. 1, 1994, pp. 47–67. doi:10.1006/jcph.1994.1005

[29] Svärd, M., "Stable High-Order Finite Difference Methods for Aerodynamics," Ph.D. Thesis, Uppsala Univ., Uppsala, Sweden, 2004.

[30] Swanson, R. C., and Turkel, E., "On Central-Difference and Upwind Schemes," *Journal of Computational Physics*, Vol. 101, No. 2, 1992, pp. 292–306. doi:10.1016/0021-9991(92)90007-L

[31] 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, AIAA Paper 81-1259, 1981.

[32] Pulliam, T. H., "Efficient Solution Methods for the Navier-Stokes Equations," *Numerical Techniques for Viscous Flow Computation in Turbomachinery Bladings*, von Kármán Inst. for Fluid Dynamics Lecture Series, von Kármán Inst. for Fluid Dynamics, Rhode-Saint-Genèse, Belgium, Jan. 1986.

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

[34] Kelley, C. T., and Keyes, D. E., "Convergence Analysis of Pseudo-Transient Continuation," *SIAM Journal on Numerical Analysis*, Vol. 35, No. 2, Apr. 1998, pp. 508–523. doi:10.1137/S0036142996304796

[35] 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, Reno, NV, AIAA Paper 93-0878, 1993.

[36] Hicken, J. E., and Zingg, D. W., "A Parallel Newton-Krylov Flow Solver for the Euler Equations on Multiblock Grids," 18th AIAA Computational Fluid Dynamics Conference, Miami, FL, AIAA Paper 2007-4333, June 2007.

[37] Chisholm, T. T., "A Fully Coupled Newton–Krylov Solver With a One-Equation Turbulence Model," Ph.D. Thesis, Univ. of Toronto, Toronto, 2007.

[38] Mulder, W. A., and van Leer, B., "Experiments with Implicit Upwind Methods for the Euler equations," *Journal of Computational Physics*, Vol. 59, No. 2, 1985, pp. 232–246. doi:10.1016/0021-9991(85)90144-5

[39] 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* (to be published).

[40] Eisenstat, S. C., and Walker, H. F., "Choosing the Forcing Terms in an Inexact Newton Method," *SIAM Journal on Scientific Computing*, Vol. 17, No. 1, Jan. 1996, pp. 16–32. doi:10.1137/0917003

[41] 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. doi:10.1137/0907058

[42] Saad, Y., and Malevsky, A. V., "P-Sparslib: A Portable Library of Distributed Memory Sparse Iterative Solvers," Minnesota Supercomputer Inst., Rept. UMSI-95-180, Minneapolis, MN, 1995; also available online at http://www-users.cs.umn.edu/~saad/PDF/umsi-95-180.pdf.

[43] 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.
doi:10.2307/2005786

[44] Saad, Y., *Iterative Methods for Sparse Linear Systems*, 2nd ed., Society for Industrial and Applied Mathematics, Philadelphia, 2003.

[45] Gropp, W. D., Kaushik, D. K., Keyes, D. E., and Smith, B. F., "High-Performance Parallel Implicit CFD," *Parallel Computing*, Vol. 27, No. 4, 2001, pp. 337–362.
doi:10.1016/S0167-8191(00)00075-2

[46] Saad, Y., "A Flexible Inner-Outer Preconditioned GMRES Algorithm," *SIAM Journal on Scientific and Statistical Computing*, Vol. 14, No. 2, 1993, pp. 461–469.
doi:10.1137/0914028

[47] Schmitt, V., and Charpin, F., "Pressure Distributions on the ONERA-M6-Wing at Transonic Mach Numbers," *Experimental Data Base for Computer Program Assessment*, AGARD, Neuilly-sur-Seine, France, 1979.

K. Powell
Associate Editor