Comparison of parallel preconditioners for a Newton-Krylov flow solver

Jason E. Hicken, Michal Osusky, and David W. Zingg

1 Introduction

Analysis of the results from the AIAA Drag Prediction workshops (Mavriplis et al, 2008) suggest that at least $O(10^8)$ grid nodes are necessary for grid-converged lift and drag values in computational aerodynamics. This motivates the development of efficient solution algorithms that scale well using thousands of processors. In particular, we are interested in parallel Newton-Krylov methods, since serial implementations of these methods have proven to be highly efficient for turbulent flows (Chisholm and Zingg, 2009).

Parallel preconditioning is a critical part of scalable Newton-Krylov algorithms, and this remains an active area of research for large-scale problems. In this paper, we compare the performance of two parallel preconditioners for a Newton-Krylov flow solver: an additive-Schwarz preconditioner and an approximate-Schur preconditioner. A previous study using only 24 processors was inconclusive regarding the relative performance of these preconditioners (Hicken and Zingg, 2008). In the present study, the approximate-Schur preconditioner is shown to scale well to at least 1000 processors and is superior to the additive-Schwarz preconditioner when more than 100 processors are used.

2 Overview of the Newton-Krylov Solver

This section briefly reviews the parallel Newton-Krylov solution algorithm: for details see Hicken and Zingg (2008, 2009) and Osusky et al (2010). The steady governing equations are discretized in space, leading to a coupled set of nonlinear algebraic equations:

$$\mathbf{R}(\mathbf{q}) = \mathbf{0}, \tag{1}$$

Jason E. Hicken, postdoctoral fellow
e-mail: jehicken@oddjob.utias.utoronto.ca

Michal Osusky, PhD candidate
e-mail: michal@oddjob.utias.utoronto.ca

David W. Zingg, Professor and Director
University of Toronto Institute for Aerospace Studies,
4925 Dufferin St. Toronto, Ontario, Canada, M3H 5T6
e-mail: dwz@oddjob.utias.utoronto.ca

where $\mathbf{R} : \mathbb{R}^n \to \mathbb{R}^n$ is the residual and $\mathbf{q} \in \mathbb{R}^n$ is a column vector representing the discrete solution. The results presented in this paper are based on a summation-by-parts finite-difference discretization (Kreiss and Scherer, 1974); however, the Newton-Krylov algorithm is relatively independent of the chosen discretization.

Newton's method is applied to (1), leading to the sequence of linear systems

$$\mathsf{A}^{(k)} \Delta \mathbf{q}^{(k)} = -\mathbf{R}^{(k)}, \qquad k \geq 0, \tag{2}$$

where $\Delta \mathbf{q}^{(k)} = \mathbf{q}^{(k+1)} - \mathbf{q}^{(k)}$, $\mathbf{R}^{(k)} = \mathbf{R}(\mathbf{q}^{(k)})$, and $\mathsf{A}^{(k)}_{ij} = \partial R^{(k)}_i / \partial q_j$. Newton's method converges provided $\mathsf{A}^{(k)}$ is non-singular and the initial iterate $\mathbf{q}^{(0)}$ is sufficiently close to the solution: a globalization strategy is necessary to ensure the algorithm converges when $\mathbf{q}^{(0)}$ is far from the solution. The globalization strategy adopted here is the dissipation-based continuation of Hicken and Zingg (2009).

Solving the linear system (2) exactly is unnecessary and inefficient, particularly in the early iterations of Newton's method. This motivates the class of inexact-Newton methods (Dembo et al, 1982), in which the linear-update system is solved to a tolerance η_k:

$$\| \mathbf{R}(\mathbf{q}^{(k)}) + \mathsf{A}^{(k)} \Delta \mathbf{q}^{(k)} \|_2 \leq \eta_k \| \mathbf{R}(\mathbf{q}^{(k)}) \|_2.$$

A Newton-Krylov algorithm is an inexact-Newton method that uses a Krylov iterative solver. In this work, we use the Krylov iterative solver GCROT(m,k) (Hicken and Zingg, 2010), a variant of de Sturler's truncated GCRO algorithm (de Sturler, 1999). Krylov methods require only Jacobian-vector products to solve (2), and these products can be obtained using a finite-difference approximation. Hence, the Jacobian matrix does not need to be explicitly formed and stored, leading to significant CPU-time and memory savings[1]

3 Parallel Preconditioners

A Newton-Krylov algorithm consists of three primary operations: residual evaluations, dot products, and preconditioner applications. The first two operations are easily parallelized and scale well. In contrast, finding an effective parallel preconditioner can be challenging. Saad and Sosonkina (1999) proposed an approximate-Schur preconditioner that we have found to be an efficient choice for CFD applications. For comparison, we also consider the additive-Schwarz preconditioner with no overlap (block Jacobi), which is a popular preconditioner for Newton-Krylov algorithms. Both preconditioners are described below in the context of the generic system

$$\mathsf{A}\mathbf{x} = \mathbf{b}, \tag{3}$$

where $\mathsf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.

[1] While most Newton-Krylov algorithms require an approximate Jacobian matrix for preconditioning, this matrix typically requires less storage and computation than the full Jacobian.

ILU-based Additive Schwarz Let P_i be the rectangular matrix that projects a global n-vector onto a local vector corresponding to the unknowns stored on process i. Applying P_i to the linear system (3), and considering local solutions of the form $\mathbf{x} = \mathsf{P}_i^T \mathbf{x}_i$, we obtain the block diagonal system

$$\mathsf{A}_i \mathbf{x}_i = \mathbf{b}_i, \tag{4}$$

where $\mathbf{b}_i \equiv \mathsf{P}_i \mathbf{b}$ and $\mathsf{A}_i \equiv \mathsf{P}_i \mathsf{A} \mathsf{P}_i^T$. The local matrix A_i is factored into $\mathsf{L}_i \mathsf{U}_i$ using ILU(p) (Meijerink and van der Vorst, 1977). The additive-Schwarz preconditioner (with no overlap) is obtained by applying the local factorizations to $\mathbf{b}_i$, and then summing the local contributions to create the global preconditioned vector:

$$\mathbf{x}_{\text{schwarz}} \equiv \sum_i \mathsf{P}_i^T (\mathsf{U}_i^{-1} \mathsf{L}_i^{-1}) \mathsf{P}_i \mathbf{b}.$$

The operations in this block-Jacobi preconditioner are local and parallelize well. However, the preconditioner becomes less effective as more processors are added, because the coupling between domains is ignored.

Approximate Schur Consider a local ordering in which variables assigned to processor i and coupled to unknowns on processor $j \neq i$ are ordered last. This ordering partitions $\mathsf{P}_i \mathsf{A} \mathbf{x} = \mathsf{P}_i \mathbf{b}$ into the following block structure:

$$\begin{pmatrix} \mathsf{B}_i & \mathsf{F}_i \\ \mathsf{H}_i & \mathsf{C}_i \end{pmatrix} \begin{pmatrix} \mathbf{u}_i \\ \mathbf{y}_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_j \mathsf{E}_{ij} \mathbf{y}_j \end{pmatrix} = \begin{pmatrix} \mathbf{f}_i \\ \mathbf{g}_i \end{pmatrix}. \tag{5}$$

The variables $\mathbf{u}_i$ are not coupled across processors and can be eliminated using

$$\mathbf{u}_i = \mathsf{B}_i^{-1} (\mathbf{f}_i - \mathsf{F}_i \mathbf{y}_i). \tag{6}$$

Substituting $\mathbf{u}_i$ into (5) we find the following system for the variables coupling the domains:

$$\underbrace{\begin{pmatrix} \mathsf{S}_1 & \mathsf{E}_{12} & \dots & \mathsf{E}_{1P} \\ \mathsf{E}_{21} & \mathsf{S}_2 & \dots & \mathsf{E}_{2P} \\ \vdots & & \ddots & \vdots \\ \mathsf{E}_{P1} & \mathsf{E}_{P2} & \dots & \mathsf{S}_P \end{pmatrix}}_{\mathsf{S}} \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_P \end{pmatrix} = \begin{pmatrix} \mathbf{g}_1' \\ \mathbf{g}_2' \\ \vdots \\ \mathbf{g}_P' \end{pmatrix}, \tag{7}$$

where $\mathsf{S}_i \equiv \mathsf{C}_i - \mathsf{H}_i \mathsf{B}_i^{-1} \mathsf{F}_i$, and $\mathbf{g}_i' \equiv \mathbf{g}_i - \mathsf{H}_i \mathsf{B}_i^{-1} \mathbf{f}_i$. The coefficient matrix S is the Schur complement corresponding to the variables coupled between processors. Suppose we solve (7) using block Jacobi. This approach would parallelize well, but it requires S_i — more precisely, its inverse — which can be expensive to form explicitly. Saad and Sosonkina (1999) recognized that an ILU factorization of S_i can easily be extracted from an ILU(p) factorization of A_i. Their Schur-based preconditioner consists of a GMRES-accelerated approximate solution of (7), with S_i replaced by its ILU factorization. Once approximate solutions to the $\mathbf{y}_i$ are obtained, they are substituted into (6), with B_i replaced with its ILU(p) factorization, to obtain $\mathbf{u}_i$. The

Table 1 Description of cases used to compare preconditioners

case	grid size	a.o.a.†	Mach number	Reynolds number
case I	1.0×10^7 nodes	3.00	0.50	∞ / 600
case II	1.0×10^7 nodes	3.06	0.84	∞ / 600
case III	3.8×10^7 nodes	3.00	0.50	∞ / 600
case IV	3.8×10^7 nodes	3.06	0.84	∞ / 600

† angle of attack in degrees

approximate-Schur preconditioner is nonstationary, so we use the flexible variant of GCROT(m, k).

4 Results

The two preconditioners are compared using the four test cases listed in Table 1. The geometry for all cases is the ONERA M6 wing. Both inviscid and viscous flow conditions are considered for each case, with the off-wall mesh spacing modified appropriately. The preconditioners for the inviscid runs are constructed using ILU(2), while the viscous preconditioners are based on an ILU(3) factorization. Fourth-difference scalar dissipation is used in all cases, and second-difference dissipation is activated as necessary. For a given case, the parallel performance of the preconditioner *prec* using p processors is measured using a relative efficiency: $\text{efficiency}_p^{prec} \equiv \left(p_{\min} T_{p_{\min}}^{\text{Schur}}\right) / \left(p T_p^{prec}\right)$, where $T_{p_{\min}}^{\text{Schur}}$ is the CPU time for a nine-order rresidual-norm reduction when the approximate-Schur preconditioner is used with the fewest possible processors permitted by memory constraints, $p_{\min}$.

Figure 1 plots the results for the inviscid runs. Cumulative matrix-vector products are plotted versus processors in the left figure, and relative efficiency is shown in the right figure. As the number of processors is increased, the additive-Schwarz preconditioner generally uses more matrix-vector products, as expected. In contrast, the number of products used by the approximate-Schur preconditioner remains steady or declines slightly. Consequently, the approximate-Schur preconditioner has a better relative efficiency when more processors are used (0.7–0.9) compared with the additive-Schwarz preconditioner (0.5–0.6).

The results from the viscous runs are shown in Figure 2. Both preconditioners exhibit better scalability compared with the inviscid runs: the viscous Jacobian-vector product is more expensive but the number of products remains on the same order as the inviscid runs, so the ratio of computation to communication goes up. Relative performance is similar to the inviscid runs: the approximate-Schur preconditioner outperforms the additive-Schwarz preconditioner when more than 100 processors are used.

Table 2 summarizes the CPU times required to reduce the L^2 norm of the residual by nine orders of magnitude, when the maximum number of processors are used.

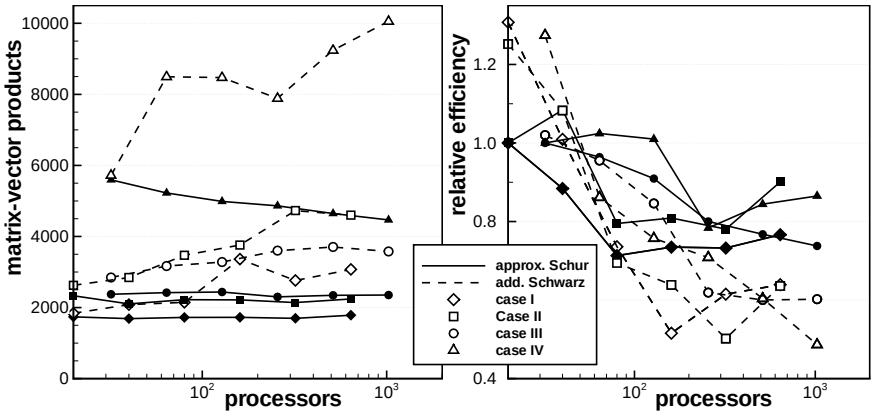

Fig. 1 Number of matrix-vector products versus processors (left) and relative parallel efficiency (right) for the inviscid runs.

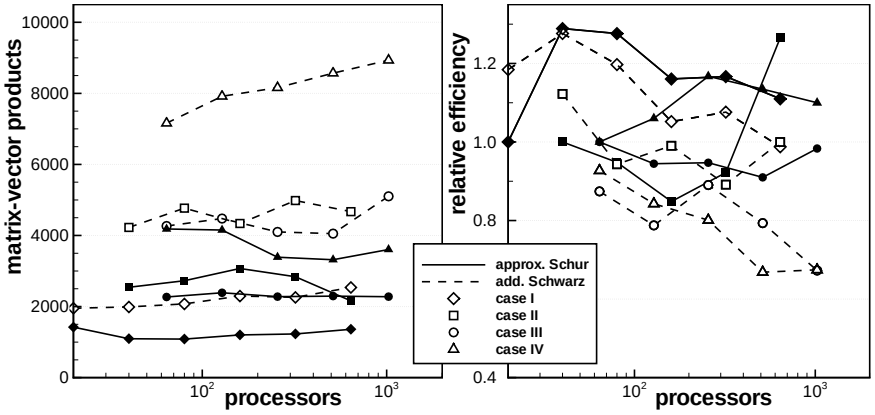

Fig. 2 Number of matrix-vector products versus processors (left) and relative parallel efficiency (right) for the viscous runs.

Table 2 CPU times, in seconds, for a 10^{-9} residual reduction using the maximum number of processors considered.

		640 processors		1024 processors	
		case I	case II	case III	case IV
inviscid	add. Schwarz	294	436	1123	2842
	approx. Schur	245	308	917	1598
viscous	add. Schwarz	741	1324	3582	6508
	approx. Schur	659	1046	2451	3992

5 Conclusions and Future Work

For a problem of fixed size, the number of matrix-vector products required by the additive-Schwarz (block-Jacobi) preconditioner generally increases as the number of processors increases. The approximate-Schur preconditioner, in contrast, is much less sensitive to the number of processors and scales well out to at least 10^3 processors. The approximate-Schur preconditioner is straightforward to implement and outperforms the Schwarz preconditioner when more than 100 processors are used.

References

Chisholm TT, Zingg DW (2009) A Jacobian-free Newton-Krylov algorithm for compressible turbulent fluid flows. Journal of Computational Physics 228(9):3490–3507, DOI 10.1016/j.jcp.2009.02.004

Dembo RS, Eisenstat SC, Steihaug T (1982) Inexact Newton methods. SIAM Journal on Numerical Analysis 19(2):400–408, DOI 10.1137/0719025

Hicken JE, Zingg DW (2008) A parallel Newton-Krylov solver for the Euler equations discretized using simultaneous approximation terms. AIAA Journal 46(11):2773–2786, DOI 10.2514/1.34810

Hicken JE, Zingg DW (2009) Globalization strategies for inexact-Newton solvers. In: 19th AIAA Computational Fluid Dynamics Conference, San Antonio, Texas, United States, AIAA-2009-4139

Hicken JE, Zingg DW (2010) A simplified and flexible variant of GCROT for solving nonsymmetric linear systems. SIAM Journal on Scientific Computing 32(3):1672–1694

Kreiss HO, Scherer G (1974) Finite element and finite difference methods for hyperbolic partial differential equations. In: de Boor C (ed) Mathematical Aspects of Finite Elements in Partial Differential Equations, Mathematics Research Center, the University of Wisconsin, Academic Press

Mavriplis DJ, Vassberg JC, Tinoco EN, Mani M, Brodersen OP, Eisfeld B, Wahls RA, Morrison JH, Zickuhr T, Levy D, Murayama M (2008) Grid quality and resolution issues from the drag prediction workshop series. In: The 46rd AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, AIAA–2008–0930

Meijerink JA, van der Vorst HA (1977) An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. Mathematics of Computation 31(137):148–162

Osusky M, Hicken JE, Zingg DW (2010) A parallel Newton-Krylov-Schur flow solver for the Navier-Stokes equations using the SBP-SAT approach. In: 48th AIAA Aerospace Sciences Meeting and Exhibit, Orlando, Florida, AIAA–2010–116

Saad Y, Sosonkina M (1999) Distributed Schur complement techniques for general sparse linear systems. SIAM Journal of Scientific Computing 21(4):1337–1357

de Sturler E (1999) Truncation strategies for optimal Krylov subspace methods. SIAM Journal on Numerical Analysis 36(3):864–889