NASA/CR-1998-208435
ICASE Report No. 98-24

Globalized Newton-Krylov-Schwarz Algorithms and Software for Parallel Implicit CFD

W.D. Gropp
Argonne National Laboratory, Argonne, Illinois

D.E. Keyes
Old Dominion University, Norfolk, Virginia and
ICASE, Hampton, Virginia

L.C. McInnes
Argonne National Laboratory, Argonne, Illinois

M.D. Tidriri
Iowa State University, Ames, Iowa

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA
Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center under
Contracts NAS1-19480 and NAS1-97046

August 1998

GLOBALIZED NEWTON-KRYLOV-SCHWARZ ALGORITHMS AND SOFTWARE FOR PARALLEL IMPLICIT CFD *

W. D. GROPP†, D. E. KEYES‡, L. C. MCINNES§, AND M. D. TIDRIRI¶

Abstract. Implicit solution methods are important in applications modeled by PDEs with disparate temporal and spatial scales. Because such applications require high resolution with reasonable turnaround, "routine" parallelization is essential. The pseudo-transient matrix-free Newton-Krylov-Schwarz (ΨNKS) algorithmic framework is presented as an answer. We show that, for the classical problem of three-dimensional transonic Euler flow about an M6 wing, ΨNKS can simultaneously deliver

- globalized, asymptotically rapid convergence through adaptive pseudo-transient continuation and Newton's method;
- reasonable parallelizability for an implicit method through deferred synchronization and favorable communication-to-computation scaling in the Krylov linear solver; and
- high per-processor performance through attention to distributed memory and cache locality, especially through the Schwarz preconditioner.

Two discouraging features of ΨNKS methods are their sensitivity to the coding of the underlying PDE discretization and the large number of parameters that must be selected to govern convergence. We therefore distill several recommendations from our experience and from our reading of the literature on various algorithmic components of ΨNKS, and we describe a freely available, MPI-based portable parallel software implementation of the solver employed here.

Key words. Newton-Krylov-Schwarz algorithms, parallel CFD, implicit methods

Subject classification. Computer Science

1. Introduction. Disparate temporal and spatial scales arise in CFD applications such as transonic or high Reynolds number flows in which the flow velocity goes to zero at a stagnation point or no-slip surface, flows containing shocks or combustion fronts, and multidisciplinary phenomena such as aeroelasticity. Local equilibration of wavespeeds (as in the "preconditioning" of [75]) and temporal subcycling (as in the "three-field" method of [26]) are two strategies that permit explicit integration of the flowfields of some such problems; nevertheless, implicit solution methods are playing increasingly important roles in CFD. Whereas explicit methods based on localized stencil updates with nearest-neighbor communication lend themselves straightforwardly to scalable parallelization, implicit methods based on

*This work was supported in part by the National Aeronautics and Space Administration under NASA Contract Nos. NAS1-97046 and NAS1-19480, while the authors were in residence at the Institute for Computer Applications in Science and Engineering, MS 403, NASA Langley Research Center, Hampton, VA 23681-2199.

† Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439-4844 USA `www.mcs.anl.gov/~gropp`

‡ Computer Science Department, Old Dominion University, Norfolk, VA 23529-0162 USA & ICASE, NASA Langley Res. Ctr., Hampton, VA 23681-2199 USA `www.cs.odu.edu/~keyes`

§ Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439-4844 USA `www.mcs.anl.gov/~mcinnes`

¶ Mathematics Department, Iowa State University, Ames, IA 50011-2064 USA

the global recurrences of forward and backward triangular solves (exact or approximate) and/or the global inner products of optimal descent or Krylov iterative methods appear to be intrinsically less scalable. At the same time, the demand for high resolution with reasonable turnaround requires scalable parallelism and in particular — for cost effectiveness — latency-tolerant parallelism for applicability to networked clusters. For the common situation in which the discretization is applied in a uniform way over a grid that is adapted only infrequently (and may therefore be cost-effectively load balanced to high precision), the pseudo-transient matrix-free Newton-Krylov-Schwarz (ΨNKS) algorithmic framework is presented as one answer to the conflicting demands of implicitness and parallelism.

In the final section, we discuss the role of ΨNKS methods in more dynamic simulation environments. For problems in which convergence to a low-residual steady state is required, as is often the case when CFD analyses are finite-differenced for sensitivities in computational design and optimization, a Newton method is asymptotically cost effective. Alternatively, the ability to solve implicit linear systems with the Jacobian, a by-product of Newton's method, can be exploited in deriving sensitivities. Motivated by the requirements of NASA, DOE, industry, and others to employ CFD in the design process, we regard the ability to apply the inverse action of the Jacobian as ultimately necessary to the optimization process, and therefore to be exploited during the analysis process as well. While Newton methods sometimes converge from simply specified initial iterates even in challenging CFD problems (e.g., [13]), they must usually be "robustified" through a continuation scheme, such as pseudo-transience (e.g., [40]). Newton methods for PDEs require the solution of large, sparse nonsymmetric linear systems, to which we apply Krylov methods, such as GMRES [65]. To control the number of Krylov iterations, while obtaining concurrency proportional to the number of processors, we precondition them with domain-decomposed additive Schwarz methods [69].

Effective use of ΨNKS in CFD codes requires attention to several details. We describe, in particular, the sensitivity of the methodology to the implicitness of the boundary conditions, the presence of limiters in the discretization, the scaling of the differencing parameter in the matrix-free application of the Jacobian, the convergence of the inner Krylov iterations, and the aggressiveness of the pseudo-transient continuation. Some of these considerations are generic to any system modeled by PDEs.

Various aspects of the ΨNKS framework have been pioneered by others over the past two decades. While this paper (and its unstructured-grid companions [38, 44]) contributes some new, architecturally oriented advances, our principal goal is to integrate algorithmic progress and to reconcile trade-offs between interrelated algorithmic components so as to promote robustness, rapid convergence, and parallel scalability in the context of an important family of applications.

The use of pseudo-transience as a means of approaching steady state (typically in the form of time-parabolization of an elliptic boundary value problem) has been independently reinvented in contexts far too numerous to mention. We have been particularly influenced by two forms: the "successive evolution-relaxation" strategy of Mulder and Van Leer [56] and the temporal truncation error strategy described in [45], both of which smoothly adapt the aggressiveness of the timestepping to the progress of the iterations toward steady state, ultimately leading to Newton-like asymptotic superlinear or quadratic convergence rates. Some sufficient conditions for globalized convergence for such strategies are given in [40].

An exact Newton method is rarely optimal in terms of memory and CPU resources for large-scale problems, such as finely resolved multidimensional PDE simulations. The pioneering work of

Dembo, Eisenstat, & Steihaug [20] showed that properly tuned inexact Newton methods can save enormous amounts of work (through approximating the Newton corrections, which can in turn permit approximation of the Jacobian matrix) over a sequence of Newton iterations, while still converging quadratically. This theory was revisited to provide inexpensive, constructive formulae for the sequence of inexact tolerances by Eisenstat & Walker [24]. Smooke [70] and Schreiber & Keller [67] devised Newton-chord methods with models for cost-effective frequency of Jacobian reevaluation. The use of various approximate Newton methods in CFD emerged independently in various regimes. Vanka [77] implemented Newton solvers in primitive-variable Navier-Stokes problems. Venkatakrishnan [78], Orkwis [61], and Whitfield & Taylor [84] established Newton-like methods in transonic problems. These works employed various direct or stationary iterative methods for the linear Newton correction equation and were based on explicit matrix representations of the Jacobian operator.

The advent of Krylov iterative methods (see, e.g., [28] for a survey) inside of inexact Newton iterations in a matrix-free context can be traced to the ODE-oriented papers of Gear & Saad [29], Chan & Jackson [17], and Brown & Hindmarsh [8] and the PDE-oriented work of Brown & Saad [9]. (The term "Newton-Krylov" seems first to have been applied to such problems in [9].) The GMRES [65] method was firmly established in CFD following the work of Wigton, Yu, & Young [86] and Johann, Hughes, & Shakib [37, 68]. Venkatakrishnan & Mavriplis showed in [81] that NK methods (preconditioned with a global incomplete factorization) were competitive with multigrid methods for large-scale CFD problems; a similar comparison for the matrix-free form of such methods was given by Keyes in [42]. Various practical aspects of NK methods in CFD were explored in [1, 6, 36, 51, 52, 59, 63, 73].

The application of domain decomposition-based preconditioners to nonlinearly implicit CFD algorithms has been our focus for the past decade [43]. Cai's doctoral dissertation [10] extended overlapping Schwarz theory to the nonselfadjoint operators of convection-diffusion problems and first articulated their optimality — even without the benefit of a coarse grid component — in the parabolic case. The term "Newton-Krylov-Schwarz" was coined in [14]. NKS methods have been taken up by Cai and collaborators [11, 12, 13, 15], Knoll and collaborators [46, 47, 48, 49, 52], and Pernice and collaborators [62], among many others.

One of the main contemporary motivations for domain decomposition methods is divide-and-conquer concurrency. Scalability studies based on dimensionless ratios of communication and computation parameters for message-passing aspects of domain-decomposed iterative methods appeared in [30, 31]. Recently, the cache-based motivation for domain decomposition has become very apparent [82]. Actual parallel implementations of NKS methods, however, are relatively few. We mention the shared-memory implementation of [53] and the distributed-memory implementations of [13, 15, 38, 44].

We believe that the most important capabilities of ΨNKS algorithms have been brought together in freely available, widely portable, and widely distributed parallel software for the first time in the Portable, Extensible Toolkit for Scientific Computation (PETSc) package [5], through which the illustrative results of this paper have been obtained. We justify the broad scope of the presentation by the need to appreciate the framework as a whole. If any aspect of the ΨNKS algorithmic framework is missing or "mistuned" (in various senses to be illustrated in Section 6), significant performance will be "left on the table". On the other hand, this is by no means a comprehensive paper. It considers only one example and omits important strategies such as multilevel iteration and solution-adaptive partitioning. It should be regarded as one installment of a "metapaper" that PETSc has been created to help us

continue to write.

We begin by detailing the pseudo-transient continuation strategy and the NKS algorithmic framework in Section 2. We review the compressible Euler equations in Section 3, followed by additional algorithmic details (specifically, CFL advancements and matrix-free issues) in Section 4. Section 5 discusses the parallel implementation of NKS in a legacy CFD code (the JULIANNE code of Whitfield and Taylor [84]) using PETSc to replace the solver while preserving the discretization, and Section 6 presents numerical experiments. We summarize and extrapolate in Section 7.

2. Algorithmic Framework. ΨNKS methods are designed to solve steady-state systems of nonlinear boundary value problems discretized as

$$f(u) = 0, \tag{2.1}$$

where u is a vector of unknowns representing the state of the system (typically nodal values of multiple fields defined at the same set of grid locations, though other interpretations in terms of expansion coefficients for the fields, or staggered grid layouts of nodal values, are included), with solution u^*, and where $f(u)$ is a vector-valued function of residuals of the governing equations. We are primarily interested in three-dimensional settings in which the number of gridpoints is in the millions and the number of components of u is correspondingly larger by a factor of (typically) five (more with turbulence or reaction models, fewer for various special incompressible or irrotational models). For problems of this size, typical of full airplane external aerodynamics or of complex ASCI-scale systems, the ordering of unknowns can be crucial to performance on a hierarchical memory computer system, and we remark on this issue later. In this paper, we consider a problem on a mapped, structured grid. For related implementations on unstructured grids, see [38].

2.1. Pseudo-Transient Continuation. Pseudo-transient continuation solves the steady-state problem (2.1), for which a solution is presumed to exist, through a series of problems

$$g_\ell(u) \equiv \frac{u - u^{\ell-1}}{\tau^\ell} + f(u) = 0, \quad \ell = 1, 2, \ldots, \tag{2.2}$$

which are derived from a method-of-lines model

$$\frac{\partial u}{\partial t} = -f(u),$$

each of which is solved (approximately) for u^ℓ. The physical transient is followed when the timestep τ^ℓ is sufficiently small, leading the iterations through a physically feasible sequence of states. Furthermore, the Jacobians associated with $g_\ell(u) = 0$ are well conditioned when τ^ℓ is small. τ^ℓ is advanced from $\tau^0 \ll 1$ to $\tau^\ell \to \infty$ as $\ell \to \infty$, so that u^ℓ approaches the root of $f(u) = 0$. We emphasize that pseudo-transient continuation does *not* require reduction in $||f(u^\ell)||$ at each step, as do typical linesearch or trust region globalization strategies [21]; it can climb hills.

Strict Newton iteration applied to (2.2) yields

$$u^{\ell,k} = u^{\ell-1} - (I + \tau^\ell f'(u^{\ell,k}))^{-1}(u^{\ell,k} + \tau^\ell f(u^{\ell,k}) - u^{\ell-1}), \quad k = 0, 1, \ldots. \tag{2.3}$$

If we take $u^{\ell,0} = u^{\ell-1}$ (the simplest initial iterate), then the first correction step is

$$u^{\ell,1} = u^{\ell-1} - (\frac{1}{\tau^\ell} I + f'(u^{\ell-1}))^{-1} f(u^{\ell-1}). \tag{2.4}$$

In some problems, it may be required to iterate the Newton corrector (2.3) more than once [36] or until it converges ($\lim_{k\to\infty} u^{\ell,k} \equiv u^\ell$), thus leading in the limit to following the transient implicitly. In this paper, however, we prefer to advance in pseudo-time after just one Newton step (2.4).

A timestep scheme is required to complete the algorithm. One choice is successive evolution-relaxation (SER) [56], which lets the timestep grow in inverse proportion to residual norm progress:

$$\tau^\ell = \tau^{\ell-1} \cdot \frac{||f(u^{\ell-2})||}{||f(u^{\ell-1})||}. \tag{2.5}$$

Alternatively, a temporal truncation error strategy bounds the maximum temporal truncation error in each individual component, based on a local estimate for the leading term of the the error. (The idea is not to control the error, *per se*, but to control the stepsize through its relationship to the error.) Another approach sets target maximum magnitudes for change in each component of the state vector and adjusts the timestep so as to bring the last measured change to the target. All such devices are "clipped" into a range about the current timestep in practice. Typically, the timestep is not allowed to more than double in a favorably converging situation, or to be reduced by more than a factor of ten in an unfavorable one, unless feasibility is at stake, in which case the timestep may be drastically cut [40].

The globalization theory of [40] employs a three-phase approach, whose phases in practice may or may not be cleanly demarcated in residual norm convergence plots. Initially, $||u^0 - u^*|| \gg 1$ and $\tau^0 \ll 1$. During an "induction phase" the solution is marched in a method-of-lines sense with relatively small timestep until $||u - u^*||/||u^0 - u^*|| \ll 1$. Success of this phase is governed by stability and accuracy of the integration scheme (we simply use the backward Euler method) and by the choice of initial iterate. For problems in which a complex feature, such as a shock or a flamefront, must arise from a structure-free initial condition, the induction phase is typically by far the longest. (We are reminded of an observation attributed to Samarskii: "The slower you start, the sooner you finish.") In a grid-sequenced problem, in which the initial iterate on a given fine grid is interpolated from a converged solution on a coarser grid, and in which solution features are correctly located (if not fully resolved), the induction phase on the finest grid can be relatively brief [71]. During a second "transition phase" the timestep is built up in the neighborhood of the solution. The critical assumption is existence of a β such that $\|(I + \tau f'(u))^{-1}\| \leq (1 + \beta\tau)^{-1}$ for all $\tau \geq 0$ if $\|u - u^*\| \leq \epsilon$. Finally comes a "polishing phase," during which the the timesteps approach infinity (or some user-imposed upper bound) and iterates approach the root with asymptotic Newton-like convergence. This phase is treated by a conventional local analysis, as in [39].

The main result of the theory is that there is either convergence from u^0 to u^* or an easily detectable contraction of τ^ℓ toward 0, allowing recovery actions. The main hypotheses of the theory, including smooth differentiability of $f(u)$, are difficult to verify in practice. They are also rarely respected in practice, since instantaneous analytical approximations of $f'(u)$ are too expensive in memory and execution time.

2.2. Inexact Newton Methods. We use the term "inexact Newton method" to denote any nonlinear iterative method for solving $f(u) = 0$ through a sequence $u^\ell = u^{\ell-1} + \lambda^\ell \delta u^\ell$, where δu^ℓ approximately satisfies the true Newton correction equation

$$f'(u^{\ell-1})\delta u^\ell = -f(u^{\ell-1}), \tag{2.6}$$

in the sense that the linear residual norm $||f'(u^{\ell-1})\delta u^\ell + f(u^{\ell-1})||$ is sufficiently small. Typically the right-hand side of the linear Newton correction equation, which is the nonlinear residual $f(u^{\ell-1})$, is evaluated to full precision, so the inexactness arises from an incomplete convergence employing the true Jacobian, freshly evaluated at $u^{\ell-1}$, or from the employment of an inexact Jacobian for $f'(u^{\ell-1})$. Typical choices would be

- a matrix whose action on a vector is constructed from finite differences of f, rather than analytic formulae;
- a matrix that is lagged (or some of whose assembly elements are lagged) from the evaluation at some previous state u^m, $m < l-1$;
- a matrix derived from a discretization related to, but not the same as, that used for f itself; or
- a matrix that has been simplified by omission of elements that are inconvenient with a particular storage scheme or approximate parallelizable inversion process.

In this paper, we consider the first possibility. The latter three possibilities for economizing on the Jacobian are also employed in this paper, but not in the matrix used in the Newton correction equation — only in the construction of its preconditioner. In other contexts, the second possibility is referred to as a "modified Newton method" [70]. The latter two we regard as so inexact that they are demoted to "defect correction methods" [42].

The first choice above is divided into two categories of finite-difference approximations:

- one in which the approximate Jacobian is explicitly constructed, element by element, from a sequence of finite differences (usually chosen with the aid of graph coloring on the discrete stencil), each of which supplies one or more columns of $f'(u)$, for example, $[f'(u^\ell)]_{ij} = \frac{\partial f_i}{\partial u_j}(u^\ell) \approx \frac{1}{h}[f_i(u^\ell + he_j) - f_i(u^\ell)]$, where h is a differencing parameter and e_j is the jth unit vector; and
- one in which the Jacobian-vector action is approximated in one or two function evaluations (besides that of $f(u^{\ell-1})$, itself) in a discrete analog of Fréchet derivatives of smooth functions, for example, $f'(u^\ell)v \approx \frac{1}{h}[f(u^\ell + hv) - f(u^\ell)]$.

We compare both methods in this paper. We note that the first method permits the differencing parameter to be chosen with respect to individual state vector components, in order to maximize the signal-to-noise ratio in the numerical differentiation, whereas the second method requires a differencing parameter that is the same for all state vector perturbations. In this sense, the first method can be made more robust than the second in finite precision, but it does require the explicit construction of the Jacobian. It is proved in [21] that the first method permits the discrete Newton method to inherit the quadratic asymptotic convergence of the true Newton method.

Inexact Newton methods require a strategy for terminating the inner linear iterations, in effect choosing η_ℓ, in

$$||f(u^{\ell-1}) + f'(u^{\ell-1})(u - u^{\ell-1})|| \leq \eta_\ell ||f(u^{\ell-1})|| \ . \tag{2.7}$$

One of the Eisenstat-Walker [24] criteria is

$$\eta_\ell = \frac{\left| \ ||f(u^{\ell-1})|| - ||f(u^{\ell-1}) + f'(u^{\ell-1})(u - u^{\ell-1})|| \ \right|}{||f(u^{l-1})||}. \tag{2.8}$$

Ajmani et al. [1] adopt: $\eta_\ell = \frac{c}{\log \tau^\ell}$. Venkatakrishnan & Mavriplis [81] adaptively choose η_ℓ so that work and storage per Newton step are bounded at some fixed expenditure. When the latter strategy

is combined with the fixed single Newton correction per pseudo-timestep, a constant linear work per timestep results.

The first strategy is theoretically elegant and appears to assure the minimum linear work consistent with a guaranteed asymptotically superlinear convergence. However, the resulting stringent linear convergence requirements may be difficult to meet in large, ill-conditioned problems. Moreover, superlinear nonlinear convergence may be too expensive a goal in practice, where the objective is to minimize execution time rather than number of inexact Newton steps.

The last strategy seems common in compressible external aerodynamics codes, particularly with problems whose memory requirements approach the maximum available. Often in such large ill-conditioned problems, relatively little progress is made in a given inner linear iteration, with the consequence that the Newton correction is effectively underdamped and the steady-state residual norm improves only slightly. This provides direct feedback limiting the increase of the timestep (and possibly decreasing it), which maintains or improves the linear conditioning of the next step, rather than letting the conditioning deteriorate with increasing pseudo-timestep. (See [25] for a polyalgorithmic method that exploits the effect of the pseudo-timestep on the linear conditioning.)

We have experimented a good deal with the first and last strategies, and we find that a hybrid approach is the most cost-effective in large transonic flow problems. Such an approach involves an initially loose convergence criterion (to avoid "oversolving," in the sense of [24]) evolving to a tighter criterion, but subject to a linear-work-per-step bound (in the sense of [81]).

For problems sufficiently small and well conditioned (linearly) to apply (2.8), overall cost is not as important.

2.3. Newton-Krylov Methods. A Newton-Krylov method uses a Krylov method, such as GMRES [65], to solve (2.6) for δu^ℓ. From a computational point of view, one of the most important characteristics of a Krylov method for the linear system $Ax = b$ is that information about the matrix A needs to be accessed only in the form of matrix-vector products in a relatively small number of carefully chosen directions. When the matrix A represents the Jacobian of a discretized system of PDEs, each of these matrix-vector products is similar in computational and communication cost to a stencil update phase (or "global flux balance") of an explicit method applied to the same set of discrete conservation equations or to a single finest-grid "work unit" in a multigrid method. NK methods are suited for nonlinear problems in which it is unreasonable to compute or store a true full Jacobian, where the action of A can be approximated by discrete directional derivatives.

Some Krylov methods for nonsymmetric problems require matrix-vector products with A^T as well as A [27]. It does not seem possible to approximate the action of A^T from finite differences of the original function evaluation. Other nonsymmetric Krylov solvers, such as CGS [72], BiCGSTAB [76], and TFQMR [27], could be substituted for GMRES and converge about as well in terms of the total number of matrix-vector products. In our experience with model problems (see, e.g., [41]), most such methods employ two matrix-vector products per step and converge in about half as many steps. It should be borne in mind, however, that their behaviors can differ wildly, and in nonuniformly rankable ways, for specially chosen problems [57]. Our experience with such solvers in the matrix-free ΨNKS context is less favorable than that with GMRES. They have the advantage of requiring less memory, and the potential of requiring fewer global reductions (for inner products), but the disadvantage of nonmonotonic and, in some cases, wildly oscillating residual norm histories, leading to decreased numerical stability. Another

advantage of GMRES is its use of matrix-vector products with unit-norm vectors v, which tend to be well suited for finite-difference approximations involving scale-sensitive perturbations, for example, $f(u^{\ell-1} + hv)$ [52].

2.4. Newton-Krylov-Schwarz Methods. A Newton-Krylov-Schwarz method combines a Newton-Krylov (NK) method, such as nonlinear GMRES [86], with a Krylov-Schwarz (KS) method, such as restricted additive Schwarz [16]. If the Jacobian A is ill-conditioned, the Krylov method will require an unacceptably large number of iterations. The system can be transformed into the equivalent form $B^{-1}Ax = B^{-1}b$ through the action of a preconditioner, B, whose inverse action approximates that of A, but at smaller cost. It is in the choice of preconditioning that the battle for low computational cost and scalable parallelism is usually won or lost. In KS methods, the preconditioning is introduced on a subdomain-by-subdomain basis through a conveniently computable approximation to a local Jacobian. Such Schwarz-type preconditioning provides good data locality for parallel implementations over a range of parallel granularities, allowing significant architectural adaptability.

Domain-based parallelism is recognized as the form of data parallelism that most effectively exploits contemporary microprocessors with multilevel memory hierarchy [18, 82]. Schwarz-type domain decomposition methods have been extensively developed for finite difference/element/volume PDE discretizations over the past decade, as reported in the annual proceedings of the international conferences on domain decomposition methods, of which the two most recent volumes (containing about 150 contributions) are [7] and [50].

We use the *restricted* additive Schwarz Method (RASM), which eliminates interprocess communication during the interpolation phase of the additive Schwarz technique [16]. In particular, if we decompose a problem into a set of possibly overlapping subdomains Ω_i, the conventional additive Schwarz method can be expressed as

$$M_{ASM}^{-1} = \sum_i {R_i}^T A_i^{-1} R_i, \tag{2.9}$$

where the three-phase solution process consists of first collecting data from neighboring subdomains via global-to-local restriction operators R_i, then performing a local linear solve on each subdomain A_i^{-1}, and finally sending partial solutions to neighboring subdomains via the local-to-global prolongation operators ${R_i}^T$. The RASM preconditioner is expressed in operator notation as

$$M_{RASM}^{-1} = \sum_i {R_i'}^T A_i^{-1} R_i. \tag{2.10}$$

It performs a complete restriction operation but does not use any communication during the interpolation phase, ${R_i'}^T$. This provides the obvious benefit of a 50% reduction in nearest-neighbor communication overhead. In addition, experimentally, it preconditions better than the original additive Schwarz method over a broad class of problems [16], for reasons that are beginning to be understood in terms of the H^1 "energy" of the global interpolant.

As originally introduced [23], additive Schwarz preconditioning includes a coarse grid term in the sum (2.9). Indeed, the coarse grid is essential for optimal conditioning in the scalar elliptic case. Success with coarse grids in Schwarz methods in the Euler and Navier-Stokes contexts has been modest, reflecting the comparable situation for multigrid methods for difficult flow problems. In this paper, we do not further consider coarse grids. We mention the Newton-based work of Venkatakrishnan [79] on

Euler problems and the defect correction-based work of Jenssen & Weinerfelt on Euler [34] and Navier-Stokes [35]. In the former context, the convergence rate enhancement of the coarse grid is nearly wiped out by the cost per iteration of its parallel implementation. In the latter cases, coarse grid corrections encouragingly pay for themselves.

2.5. Contrast with Defect Correction. A typical Euler (or Navier-Stokes) code employs a defect correction solver. To solve the sequence of nonlinear problems created upon implicitly temporally differencing (2.2), a left-hand-side matrix (related to a Jacobian) is created in whose construction computational short-cuts are employed and which may be stabilized by a degree of first-order upwinding that would not be acceptable in the discretization of the residual itself. We denote this generic distinction in the update equation (2.11) by subscripting the residual "high" and the left-hand-side matrix "low":

$$J_{low}\delta u = -f_{high}. \tag{2.11}$$

Often, J_{low} is based on a low-accuracy residual for f:

$$J_{low} = \frac{D}{\tau} + \frac{\partial f_{low}}{\partial u}, \tag{2.12}$$

where D is a scaling matrix (which could be the identity or could be a diagonal matrix of cell volumes, for instance). The inconsistency of the left- and right-hand sides prevents the use of large timesteps, τ. Using J_{low} (or, more typically, some inexpensive approximation thereto, denoted $\tilde{J_{low}}$) as a preconditioner, we replace (2.11) with

$$(\tilde{J_{low}})^{-1} J_{high}\delta u = -(\tilde{J_{low}})^{-1} f_{high}, \tag{2.13}$$

in which the action of J_{high} on a vector is obtained through directional differencing, namely,

$$J_{high}(u^l)v \approx \frac{1}{h}\left[f_{high}(u^l + hv) - f_{high}(u^l)\right] + \frac{D}{\tau}v, \tag{2.14}$$

where h is a small parameter. Note that the operators on both sides of (2.13) are based on consistent high-order discretizations; hence, timesteps can be advanced to arbitrarily large values, recovering a true Newton method in the limit.

From the viewpoint of linear convergence rate, it would seem ideal to use a preconditioner based on J_{high} in (2.13), but such a preconditioner can be much more expensive and memory-consumptive than one based on J_{low}. In (2.13), we have merely shifted the inconsistency from the nonlinear to the linear iteration. From the point of overall execution time, it is not obvious which is better: many inexpensive nonlinear iterations in which the inner linear problem (2.11) is preconditioned by J_{low}, or fewer more expensive nonlinear iterations containing the inner problem (2.13). The answer is strongly affected by the sequence of timesteps employed in (2.12). When the parameter τ is small, J_{low} and J_{high} are both dominated by the same diagonal matrix, and the extra costs of working with J_{high} in the preconditioner may be unjustified.

The potential for (2.13) to outperform (2.11) is demonstrated in the CFD context in [15, 42]. In [32], the deterioration with advancing CFL number of a solver based on approximate factorization of the operator in (2.12) is contrasted with an advancing CFL approach based on (2.13). Dimensionally split approximate factorization schemes also require low CFL number. In spite of this disadvantage in their rate of convergence to steady states, dimensionally split schemes continue to enjoy memory

advantages over more implicit schemes, the fully matrix-free work of Qin, Ludlow, & Shaw [63] being a recent example.

The range of options for J_{low} and J_{high} are explored in the context of CFD in [36]. The combination of choice for obtaining low-residual steady-state solutions (designated "ALLMUS" therein) corresponds to the use of J_{high} on the left-hand side, as well as the right, as in (2.13).

It should be borne in mind that the margin of superiority of (2.13) over less nonlinearly implicit schemes is very sensitive to the frequency of reevaluation (and refactorization) of J_{low} and to the intimate coupling of the optimal reevaluation frequency with CFL advancement strategy and Krylov subspace size. Evaluation and refactorization of J_{low} are still expensive, comparable in arithmetic cost to the evaluation of f_{high} and typically more expensive in terms of communication. The frequency of J_{low} evaluation is a relatively neglected topic in the literature, since it is so problem dependent. An empirical sequential cost model is outlined in [46].

Other tuning parameters with a strong influence on the performance of (2.13) are those that define the difference between J_{low} and $\tilde{J_{low}}$. These include parameters defining incomplete factorization fill (whether position-based or threshold-based); relaxation or multilevel method parameters if the preconditioning is implemented by a number of sweeps of an iterative method; and, in the parallel context, the number of subdomains, subdomain overlap, the use of a coarse grid in the Schwarz method, and so forth. These algorithmic tuning choices are, in principle, amenable to systematic optimization with direct search methods [22] and should be explored before undertaking a series of "production" runs.

3. Compressible Euler Equations. To illustrate the ΨNKS algorithm in the parallel context, we solve the three-dimensional compressible Euler equations on mapped, structured grids using a second-order finite volume discretization. The basis for our implementation is the sequential Fortran77 code (JULIANNE) by Whitfield & Taylor [84]. Thus, we demonstrate the use of parallel ΨNKS algorithms implemented via PETSc in the "legacy" context. This section presents a brief overview of the governing equations and boundary conditions; additional details can be found in [83, 84]. Sections 3.1 and 3.2 are standard; the material in Sections 3.3 and 3.4 describes modifications of general importance to implicit solvers.

The original JULIANNE code has a discrete Newton-relaxation pseudo-transient continuation solver with explicit enforcement of boundary conditions. We retained the discretization as embodied in flux balance routines for steady-state residual construction and finite-difference Jacobian construction. The function evaluations are undertaken to second-order in the upwinding scheme, and the Jacobian matrix (used mainly as a preconditioner, but also tested as an explicit Jacobian) is evaluated to first order. We replaced the explicit boundary conditions with a fully implicit variant [73] and added the Van Albada limiter [2] option to three limiter options provided.

3.1. Governing Equations. The governing system of PDEs for inviscid steady-state flow can be expressed in coordinate-invariant form by

$$\nabla \cdot (\rho \mathbf{u}) = 0, \tag{3.1}$$

$$\nabla \cdot (\rho \mathbf{u}\mathbf{u} + pI) = 0, \tag{3.2}$$

$$\nabla \cdot ((\rho e + p)\mathbf{u}) = 0, \tag{3.3}$$

where ρ, $\mathbf{u}$, and e represent the density, three-dimensional velocity, and energy density fields, respectively, and the pressure field p is determined by an algebraic equation of state,

$$e = \frac{p}{\gamma - 1} + \frac{1}{2}\rho(u^2 + v^2 + w^2),$$

where γ is the ratio of specific heats.

In the computational example of this paper, we work on a mapped structured grid of "C-H" type, in which the Cartesian coordinates (x, y, z) are functions of (ξ, η, ζ), discretely indexed by (i, j, k). The Jacobian of the coordinate transformation,

$$J = x_\xi(y_\eta z_\zeta - z_\eta y_\zeta) - y_\xi(x_\eta z_\zeta - z_\eta x_\zeta) + z_\xi(x_\eta y_\zeta - y_\eta x_\zeta),$$

appears briefly in this section only and is not to be confused with the Jacobian matrix of Newton's method. In addition to the Cartesian velocities (u, v, w), it is convenient to consider the velocities in the mapped coordinate system, (U, V, W), the so-called contra-variant velocities, defined by

$$\begin{aligned} U &= \xi_x u + \xi_y v + \xi_z w, \\ V &= \eta_x u + \eta_y v + \eta_z w, \\ W &= \zeta_x u + \zeta_y v + \zeta_z w. \end{aligned}$$

In the standard notation (see, e.g., [33]), the transient form of the Euler equations (3.1) is given by

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}}{\partial \xi} + \frac{\partial \mathbf{G}}{\partial \eta} + \frac{\partial \mathbf{H}}{\partial \zeta} = 0, \tag{3.4}$$

where

$$\mathbf{Q} = J \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ e \end{pmatrix},$$

and the flux vectors take the form

$$\mathbf{F} = J \begin{pmatrix} \rho U \\ \rho u U + \xi_x p \\ \rho v U + \xi_y p \\ \rho w U + \xi_z p \\ (e+p)U \end{pmatrix}, \mathbf{G} = J \begin{pmatrix} \rho V \\ \rho u V + \eta_x p \\ \rho v V + \eta_y p \\ \rho w V + \eta_z p \\ (e+p)V \end{pmatrix}, \text{ and } \mathbf{H} = J \begin{pmatrix} \rho W \\ \rho u W + \zeta_x p \\ \rho v W + \zeta_y p \\ \rho w W + \zeta_z p \\ (e+p)W \end{pmatrix}.$$

The conservation form of (3.4) can also be written in the quasilinear form

$$\frac{\partial \mathbf{Q}}{\partial t} + A\frac{\partial \mathbf{Q}}{\partial \xi} + B\frac{\partial \mathbf{Q}}{\partial \eta} + C\frac{\partial \mathbf{Q}}{\partial \zeta} = 0,$$

which defines the pointwise flux Jacobians $A \equiv \frac{\partial \mathbf{F}}{\partial \mathbf{Q}}$, $B \equiv \frac{\partial \mathbf{G}}{\partial \mathbf{Q}}$, and $C \equiv \frac{\partial \mathbf{H}}{\partial \mathbf{Q}}$.

3.2. Finite Volume Scheme. The system (3.4) is discretized via a finite volume scheme in which simplicial control volumes are centered on the vertices [84]. Using first-order backward differencing in time, one can express the resulting discrete nonlinear system as

$$\frac{\mathbf{Q}_{i,j,k}^{n+1} - \mathbf{Q}_{i,j,k}^{n}}{\Delta t} + \delta_i \mathbf{F}(\mathbf{Q}^{n+1}) + \delta_j \mathbf{G}(\mathbf{Q}^{n+1}) + \delta_k \mathbf{H}(\mathbf{Q}^{n+1}) = 0, \tag{3.5}$$

where

$$\delta_i \mathbf{F}(\mathbf{Q}^{n+1}) = \mathbf{F}_{i+1/2}(\mathbf{Q}^{n+1}) - \mathbf{F}_{i-1/2}(\mathbf{Q}^{n+1}),$$

and $\mathbf{F}_{i+1/2}(\mathbf{Q}^{n+1})$ denotes the numerical flux at cell face $i + 1/2$; analogous definitions hold for the remaining terms of (3.5). The numerical flux is computed by augmenting the first-order term that results from Roe's approximate Riemann solver [33, 64] with a second-order component. Details of the formulation, which now can be considered standard, are beyond the scope of this paper but are presented in [84].

3.3. Flux Limiters. Flux limiters are typically used when upwind discretization techniques are applied to flows with supercritical phenonema or material interfaces in order to produce steady-state solutions that avoid unrealistic oscillations (that would be properly damped by the model if the scales on which molecular viscosity acts could affordably be represented). Differentiability of the limiter is required when using derivative information in the numerical scheme. Unfortunately, many popular limiters were designed for solution algorithms of defect correction type, in which the true Jacobian never appears on the left-hand side, and are nondifferentiable (e.g., Van Leer, Superbee, Minmod) and are therefore inappropriate for direct use in Newton methods [80].

As we show in Section 6.3, this problem is not just of theoretical concern but is a weakness of such limiters in the matrix-free context, since they can cause stagnation or breakdown of the numerical scheme. Thus, for all experiments in this paper, we use the Van Albada limiter [2].

3.4. Boundary Conditions. Ghost (or "phantom" or "halo") cells are used so that the interior Euler equations, discretized on a seven-point star stencil for each conservation law, may be employed on vertices up to and including the physical boundaries. Artificial values (generally depending upon adjacent interior state variable values) are specified at the ghost vertices to complete these stencils. The values at the ghost vertices are included in the unknown state vector, and the partial derivatives of the ghost-vertex boundary conditions with respect to the ghost unknowns are included in the implicitly defined Jacobian; however, these additional values do not represent any additional resolution of the physical problem. (For this reason, we regard the coarse, medium, and fine grids in Section 6 as having recursively "doubled" dimensions of the form $(2^p n_x + 2) \times (2^p n_y + 2) \times (2^p n_z + 2)$, for $p = 0, 1, 2$, respectively, even though the number of algebraic unknowns, including ghostpoints, does not precisely double.)

In the C-H mapped coordinate system used in our simulation (see Fig. 3.1), four types of bounding surfaces at extremal values of the three indices are used to enumerate the gridfunction values: k indexes the transverse direction, from low k at the root of the wing, to k_{tip} in the plane of the wingtip, to high k in the transverse freestream; j indexes the normal direction from low j on the wing itself to high j in the freestream; and i wraps longitudinally around the wing and along the C-cut, from low i on the lower side in the rear of the wake, forward through $i_{lower,te}$ at the trailing edge, through i_{le} at the

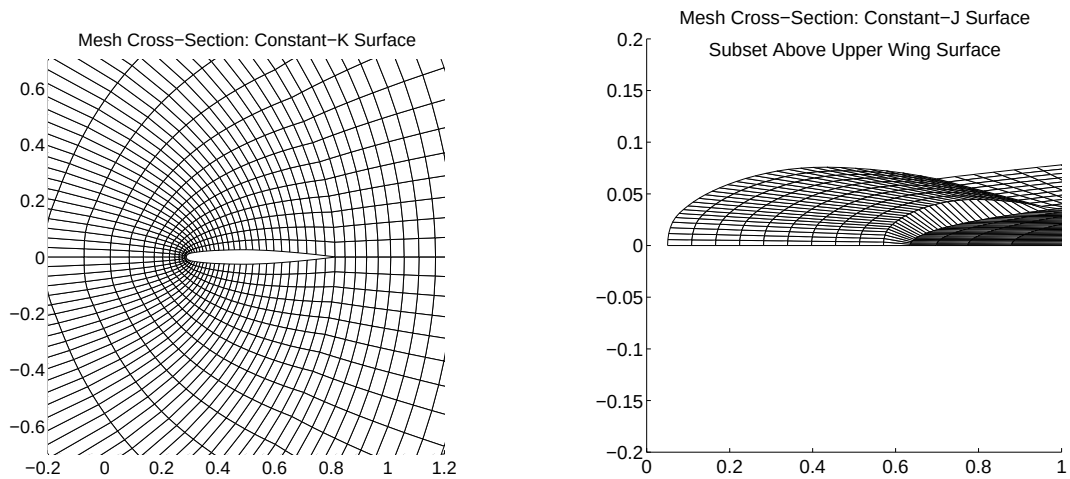

FIG. 3.1. *Constant-coordinate cuts of the "medium" grid: (a) a clipped cross-section of the constant-k surface five cells away from the wing root; (b) a clipped perspective of the constant-j surface on the upper side of the wing, looking in from the outboard front of the wing. (Index i wraps around the wing streamwise.)*

leading edge, rearward across the upper surface of the wing through $i_{upper,te}$, and finally to high i on the upper side in the rear of the wake region.

The root of the wing (low k) is considered to be a symmetry plane, with the $k = 1$ values set equal to their $k = 2$ counterparts. For points in the wake region and beyond k_{tip} of the wing, where for low j and for each k, a range of i indices maps gridpoints on either side of the C-cut back on themselves, the values at ghost vertices are set equal to the corresponding interior values on the other side of the cut.

For the freestream and impermeable wing surfaces, we use locally one-dimensional characteristic variable boundary conditions, which are briefly summarized below; see, for example, [83] for details. At each constant-coordinate surface on which a boundary condition must be derived, the nonconservative form of the Euler equations is locally linearized. Characteristic values (eigenvalues) and variables (right eigenvectors) are determined for the 5×5-block of the flux Jacobian matrix (A, B, or C) that premultiplies the derivatives of the primitive variables in the direction normal to the bounding surface. Terms involving derivatives in the plane of the bounding surface are set to zero. The sign of the each characteristic value determines whether the corresponding characteristic variable propagates information into, or out of, the computational domain.

In our test cases, we need consider only subsonic inflow and outflow boundaries and impermeable boundaries. The cases of supersonic inflow and outflow are considered in, for example, [83]. At subsonic inflow, four characteristics enter the domain and may be fixed at Dirichlet values from the freestream. One characteristic exits the domain and is set by extrapolation from adjacent interior values. Five algebraic relationships are thereby derived for the five values at each ghost vertex. At subsonic outflow, the opposite situation prevails: one characteristic variable is set from freestream conditions, and four are extrapolated. On impermeable surfaces, one characteristic enters the domain and is set by the condition of no-flow across the surface. This, in effect, provides the pressure. The remaining values are set by extrapolation so as to satisfy the no-flow constraint on the physical boundary. As illustrated in one of the experiments to follow, and as discussed by Whitfield & Taylor in [84] and for similar problems

by Tidriri [73, 74], the implicit form of these boundary conditions is needed to maintain stability as timesteps adaptively increase.

All of the boundary conditions for the ghost vertex unknowns are local (involving, at most, values at immediately interior vertices), with the exception of the C-cut ghost-to-interior identity mappings. For traditional lexicographic orderings of the gridpoints, the entries of the Jacobian matrix that tie together these spatially identical but logically remote degrees of freedom lie outside of the normal band structure, and would, if used, generate extensive fill in a full factorization. We thus choose to include these nonzeros in the matrix-free action of the Jacobian only, not in the Jacobian preconditioner.

4. ΨNKS Algorithmic Details. Careful attention to details of CFL advancement strategies and matrix-free Jacobian-vector product approximations is crucial for the successful implementation of ΨNKS methodology for large-scale problems. This section discuss these issues in some detail.

4.1. CFL Advancement. We use a locally adapted pseudo-timestep of the form $\tau_{ijk} = \alpha_{ijk} N_{CFL}$, where α_{ijk} is a ratio of signal transit time (based on local convective and acoustic velocities in each coordinate direction) to cell volume and N_{CFL} is a global dimensionless scaling factor, which would have to be kept of order unity to satisfy explicit stability bounds on the timestep and which should approach ∞ for a steady-state Newton method. The constraints on CFL advancement in our implicit context are the robustness of the nonlinear iterative process and the cost of the iterative linear solver. We employ an advancement strategy based on the SER technique [56],

$$N_{CFL}^{\ell} = N_{CFL}^{\ell-1} \frac{\|f(u^{\ell-2})\|}{\|f(u^{\ell-1})\|},$$

with clipping about the current timestep so that CFL increases by a maximum of two and decreases by no more than ten at each step. Experiments show that when the linearized Newton systems are solved sufficiently well at each step the CFL may often advance according to this strategy without further restrictions. Jiang & Forsyth [36] discuss conditions under which more stringent measures are needed to limit CFL advancement. Experiments on compressible flows with this strategy alone in [40] occasionally led to `NaN`s, which were attributed to negative densities. Whereas in a conventional Newton-like method an infeasible Newton update δu^{ℓ} can be caught before evaluation of $f(u^{\ell-1} + \lambda^{\ell} \delta u^{\ell})$, and λ^{ℓ} cut back accordingly for robustness, a matrix-free Newton code may "probe" $f(\cdot)$ at an infeasible point while building up the Krylov subspace. Thus, the input to every call to the subroutine that evaluates $f(\cdot)$ must be checked. In the pseudo-transient context an infeasible state vector may be handled the same way convergence stagnation is handled [40], namely, by restoring the state at the previous timestep and recommencing the current timestep with a drastically reduced N_{CFL}.

4.2. Matrix-Free Methods. Matrix-free Jacobian-vector products are defined by directional differencing of the form

$$f'(u)v \approx \frac{f(u + hv) - f(u)}{h},$$

where the differencing parameter h is chosen in an attempt to balance the relative error in function evaluation with the magnitudes of the vectors u and v. Selection of an appropriate parameter is nontrivial, as values either too small or too large will introduce rounding errors or truncation errors, respectively, that can lead to breakdowns. Investigators with relatively small, well-scaled discrete

problems (guided by a wise nondimensionalization) sometimes report satisfaction with a simple choice of h, approximately the square root of the "machine epsilon" or "unit roundoff" for their machine's floating-point system. A typical double-precision machine epsilon is approximately 10^{-16}, with a corresponding appropriate h of approximately 10^{-8}. More generally, adaptivity to the vectors u and v is desirable.

We choose the differencing parameter dynamically using the technique proposed by Brown and Saad [9], namely,

$$h = \frac{e_{rel}}{||v||^2} \max\left[|u^T v|, \text{typ} u^T |v|\right] \text{sign}(u^T v),$$

where $|v| = [|v_1|, ..., |v_n|]^T$, $\text{typ} u = [|\text{typ} u_1|, ..., |\text{typ} u_n|]^T$ for $\text{typ} u_j > 0$ being a user-supplied typical size of u_j and $e_{rel} \approx$ square root of relative error in function evaluations.

Determining an appropriate estimation of the relative noise or error in function evaluations is crucial for robust performance of matrix-free Newton-Krylov methods. Assuming double-precision accuracy (or $e_{rel} \approx 10^{-8}$) is inappropriate for many large-scale applications. A more appropriate relative error estimate for the compressible flow problems considered in this work is $e_{rel} = 10^{-6}$, as determined by noise analysis techniques currently under investigation by McInnes and Moré [54]. When evaluating gradients too close to the noise level in a given problem, we have found that otherwise identical executions may converge on one system with a given floating-point convention for rounding and fail to converge in a reasonable time on another with a different rounding convention. Backing off to larger values of h generally resolves such discrepancies.

Taking h too large is one of many ways of damping the nonlinear iteration in NKS methods, in that it replaces a tangent hyperplane estimate with a chordal plane estimate. However, we do not recommend using h to control the nonlinear convergence in this manner. It should generally be taken as close to the noise level as robustness requirements permit, and damping should be applied more consciously at a higher level in the code.

In addition to evaluating the Jacobian-vector products with directional differencing within GMRES, the preconditioner is constructed in a blackbox manner, without recourse to analytical formulae for the Jacobian elements, by directional differencing as described in [84] and as provided in the JULIANNE code.

Another approximate Jacobian-vector product derived from the same multivariate Taylor expansion that underlies the finite-difference approximations above, which is, however, free of subtractive cancellation error, has recently been rediscovered by Whitfield & Taylor [85]. It features an imaginary perturbation,

$$f(u + ihv) = f(u) + ihf'(u)v + c_2\mathcal{O}(h^2) + ic_3\mathcal{O}(h^3),$$

about any point u, where f, interpreted as a complex function of a complex-valued argument, is analytic. Here, u and v are real vectors. When f is real for real argument, as is true for the Euler equations, all quantities except for i in the expansion above are also real; therefore, by extracting real and imaginary parts, we can identify $f(u) = \text{Re}[f(u + ih)] + O(h^2)$ and $f'(u)v = \text{Im}[f(u + ihv)]/h + O(h^2)$. Special care is needed for Roe-type flux functions and any other nondifferentiable features of $f(u)$, but with minor code alterations, both $f(u)$ and $f'(u)v$ are available without subtraction from a single complex evaluation of $f(u)$. Implications for evaluation of sensitivity derivatives by this technique are explored in [58].

5. Parallel Implementation Using PETSc. This section discusses some issues that arise in the transition of a legacy code originally developed for uniprocessor vector architectures to a distributed-memory variant. After providing an overview of our conversion strategy, we discuss some performance optimizations for memory management, message passing, and cache utilization.

The parallelization paradigm we recommend in approaching a legacy code is a compromise between the "compiler does all" approach, for which some in the scientific and engineering communities have been waiting many years now, and the "hand-coded by expert" approach, which some others insist is still the only means of obtaining good parallel efficiency. We employ PETSc [4, 5], a library that efficiently handles, through a uniform interface, the low-level details of the distributed-memory hierarchy. Examples of such details include striking the right balance between buffering messages and minimizing buffer copies, overlapping communication and computation, organizing node code for strong cache locality, preallocating memory in sizable chunks rather than incrementally, and separating tasks into one-time and every-time subtasks using the inspector/executor paradigm. The benefits to be gained from these and from other numerically neutral but architecturally important techniques are so significant that it is efficient in both programmer time and execution time to express them in general-purpose code.

PETSc is a versatile package integrating distributed vectors, distributed matrices in several sparse storage formats, Krylov subspace methods, preconditioners, and Newton-like nonlinear methods with built-in trust region or line search strategies and continuation for robustness. It has been designed to provide the numerical infrastructure for application codes involving the implicit numerical solution of PDEs, and it sits atop MPI for portability to most parallel machines. The PETSc library is written in C, but may be accessed from user codes written in C, Fortran, and C++. PETSc has features relevant to computational fluid dynamics, including matrix-free Krylov methods, blocked forms of parallel preconditioners, and various types of timestepping.

5.1. Converting Legacy Codes. Converting a legacy code to a production parallel version involves two types of tasks: parallelization and performance optimization. Abstractly, parallelization includes the discovery or creation of concurrency, orchestration of data exchange between the concurrent processes, and mapping of the processes onto processors.

For converting structured-grid legacy codes, the major reprogramming steps are essentially: converting global data structures in the legacy code to distributed data structures provided by the domain decomposition library; replacing domain-wide loop bounds with subdomain-wide loop bounds in the routines that evaluate the governing equation residuals and Jacobian elements; and parameterizing the solution algorithm supplied by the library, which ordinarily replaces the solution algorithm in the legacy code.

A coarse diagram of the calling tree of a typical ΨNKS application appears in Fig. 5.1. The top-level user routine performs I/O related to initialization, restart, and postprocessing; it also calls PETSc subroutines to create data structures for vectors and matrices and to initiate the nonlinear solver. Subroutines with the PETSc library call user routines for function evaluations $f(u)$ and (approximate) Jacobian evaluations $J(u)$ at given state vectors. Auxiliary information required for the evaluation of f and J that is not carried as part of u is communicated through PETSc via a user-defined "context" that encapsulates application-specific data. (Such information would typically include dimensioning data, grid geometry data, physical parameters, and quantities that could be derived from the state u but are most conveniently stored instead of recalculated, such as constitutive quantities.)

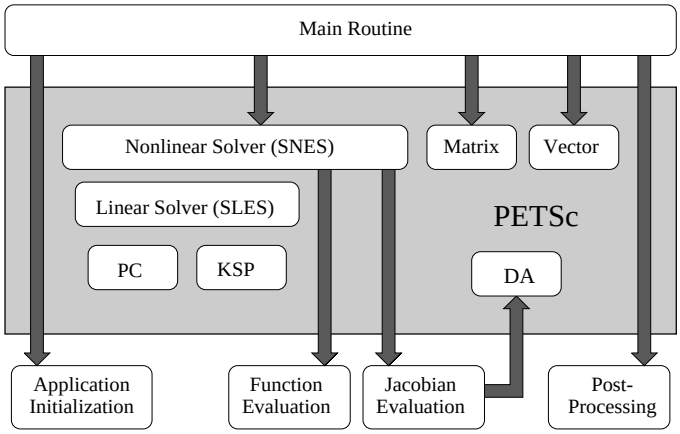

FIG. 5.1. *Coarsened calling tree of the JULIANNE-PETSc code, showing a user-provided main program and user-provided callback routines for supplying the initial nonlinear iterate, evaluating the nonlinear residual vector at a PETSc-requested state, and evaluating the Jacobian (preconditioner) matrix.*

We emphasize that the readiness of legacy codes for high-performance parallel ports of any kind varies considerably. Codes making heavy use of COMMON blocks should first be transformed to passed-argument form and made to execute at high computation rates on a cache-based uniprocessor. This process will often involve combining component fields of u found in separate arrays into a single interleaved structure, with contiguous locations in memory corresponding to unknowns associated with the same gridpoint, rather than with the same component field at an adjacent gridpoint. Codes in which solver, function evaluation, and Jacobian evaluation logic are interwoven should be modularized so that function and Jacobian evaluation routines can be cleanly and independently extracted. (Some codes use common gradient and flux evaluation logic in the subassembly of function and Jacobian evaluation, a practice we applaud. However, such common code should normally be isolated for separate calls from each major routine.)

For memory economization and high performance, we have found it advantageous to transfer elements of f and J into the distributed PETSc data structures in dense blocks of intermediate size, rather than to form an entire copy of f or J in some other user data structure and then transfer it.

5.2. Memory Management-Oriented Optimizations. Many code developers have observed that dynamic memory management within PDE-based simulations, particularly through the C library `malloc` and `free` routines, can consume significant amounts of time. In addition, even when implemented efficiently, such allocation can lead to memory fragmentation that is not well suited to cache-based memory hierarchies. Further, reallocation of memory space to enlarge a memory array often requires that data be copied from an old area to a new area. This memory copy does no useful work and can lead to a loss in performance. Since parallel sparse matrix memory management can be particularly challenging, we discuss some techniques to aid its efficiency; many of these ideas also apply to management of vectors, grids, and so forth.

5.2.1. Memory Preallocation. PETSc provides a number of ways to preallocate sparse matrix memory based on knowledge of the anticipated nonzero structure (corresponding to mesh connectivity). However, PETSc does not require preallocation; this approach avoids having programs fail simply

because sufficient memory was not preallocated. PETSc also keeps track of memory use; this profiling information can be very valuable in tracking usage patterns and identifying memory problems.

5.2.2. Aggregation in Assembly. A related issue is that of the granularity of operations, particularly for matrix assembly. It is common to define operations in terms of their most general, single-element form, such as "set matrix element to value" or "add value to matrix element." This approach is inefficient, however, because each operation requires a number of steps to find the appropriate entry in a data structure (particularly for sparse matrix formats). Thus, PETSc includes a variety of operations for handling larger numbers of elements, including logically regular dense blocks. Such aggregate optimizations significantly improve performance during operations such as matrix assembly.

Within the subject parallel compressible flow code, we specify in advance the sparsity pattern for the first-order approximation of the Jacobian that serves as the preconditioner. Thus, all matrix storage space is preallocated once and is then continually reused as the nonlinear simulation progresses. Matrix elements are assembled in aggregates of five, as they are naturally computed for this problem.

5.3. Message Passing-Oriented Optimizations. Any kind of communication of data between parallel processes involves two steps: the transfer of data and the notification that the transfer has completed. Message passing combines these two operations: for each message sent, there is a "synchronization" that indicates when the data is available for use. (In the case of shared-memory programming, this synchronization is implemented through locks, flags, or barriers.) Such synchronizations can be a source of performance problems; efficient code tries to defer any synchronization until the last possible moment. The PETSc approach to communication aims to balance ease of use and efficiency of implementation; it does not attempt to completely conceal parallelism from the application programmer. Rather, the user initiates combinations of high-level calls, but the library handles the detailed (data structure-dependent) message passing. For a detailed philosophy of PETSc implementation, see [3].

5.3.1. Multiphase Exchanges. A common way to avoid problems due to early synchronization is to divide an operation into two parts: an initiation and a completion (or ending) phase. For example, asynchronous I/O uses this approach. The MPI message-passing standard [55] provides asynchronous operations; send and receive operations are divided into starting (e.g., `MPI_Isend` or `MPI_Irecv`) and completion (e.g., `MPI_Wait`) phases. PETSc takes the same multiphased approach with other operations that would otherwise suffer from severe performance problems, including matrix assembly of nonlocal data and generalized vector scatter/gathers. For example, the starting version of these operations issues the appropriate MPI nonblocking communication calls (e.g., `MPI_Isend`). The ending version then concludes by using the appropriate completion routine (e.g., `MPI_Waitall`). Because the PETSc operations explicitly defer their completion, it is easy to change the underlying implementation to take advantage of different optimization approaches, including alternate MPI operations (e.g., persistent (`MPI_Rsend_init`)) or even non-MPI code (e.g., one-sided or remote memory operations).

5.3.2. Algorithmic Reduction in Synchronization Frequency. Pseudo-transient Newton-Krylov methods make extensive use of inner products and norms, which are examples of global reductions or commutatives and impose global synchronization on the parallel processes. The inner products are associated primarily with the conjugation process in the Krylov method. The norms are associated with the Krylov method, with convergence monitoring, and with various stability and robustness features in the selection of the timestep, the linesearch parameter, and the Fréchet differencing parameter.

To reduce the penalty of these synchronizations, PETSc offers options such as an unmodified Gram-Schmidt operation in GMRES [65], and lagged parameter selection. In severe circumstances it would be unwise to back off from the robust practices of modified Gram-Schmidt and frequent refreshing of the Fréchet parameter or the CFL number. All of the cases described herein, however, use deferred synchronization via unmodified Gram-Schmidt and reevaluate other parameters less frequently than dictated by conventional sequential practice.

5.4. Cache-Oriented Optimizations. Cache-oriented optimizations are crucial, since good overall parallel performance requires fast *per processor* computation as well as effective parallel algorithms and communication. Scalability studies often omit attention to single-node performance optimization and thereby demonstrate high scalability on code that nonetheless makes inefficient user of the hardware, overall. Here we discuss three optimization strategies: exploitation of dense block operations, field component interleaving, and grid reordering.

5.4.1. Exploitation of Dense Block Operations. The standard approach to improving the utilization of memory bandwidth is to employ "blocking". That is, rather than working with individual elements in a preconditioning matrix data structure, one employs blocks of elements. Since the use of implicit methods in CFD simulations leads to Jacobian matrices with a naturally blocked structure (with a block size equal to the number of degrees of freedom per cell), blocking is extremely advantageous. The PETSc sparse matrix representations use a variety of techniques for blocking, including

- a generic sparse matrix format (no blocking);
- a generic sparse matrix format, with additional information about adjacent rows with identical nonzero structure (so called *I-nodes*); this I-node information is used in the key computational routines to improve performance; and
- storing the matrices using a fixed (problem-dependent) block size.

The advantage of the I-node approach is that it is a minimal change from a standard sparse matrix format and brings a relatively large percentage of the improvement one obtains via blocking. Using a fixed block size delivers the absolute best performance, since inner loops can be hardwired to a particular size, removing their overhead entirely.

Table 5.1 presents the floating-point performance for a basic matrix-vector product and a triangular solve obtained from an ILU(0) factorization using these three approaches: a basic compressed row storage format, the same compressed row format using the I-nodes option, and a fixed block size code (with a block size of five). These rates were attained on one node of an IBM SP2 for a coarse grid Euler problem of dimension 25,000 (described in Section 6.1). The speeds of the I-node and fixed-block operations are several times those of the basic sparse implementations. These examples demonstrate that careful implementations of the basic sequential kernels in PETSc can dramatically improve overall floating-point performance relative to casually coded legacy kernels.

Table 5.1
Basic kernel flop rates (Mflop/s).

Kernel	Basic	I-Node Version	Fixed Block Size
Matrix-Vector Product	28	58	90
Triangular Solves from ILU(0)	22	39	65

Much of the approximate Jacobian matrix has block-band structure corresponding to the three-dimensional, seven-point stencil, with five degrees of freedom per node (three-dimensional momentum, internal energy, and density). We use the PETSc matrix format for block, compressed, sparse rows (block CSR) to exploit this structure.

5.4.2. Field Component Interleaving. For consistency with the matrix storage scheme and to exploit better cache locality within the application portion of code, we modified the original nonlinear function evaluation code to use the same interleaved ordering employed for matrix storage for the **Q**-vector instead of the original field-oriented ordering. Table 5.2 compares the performance of these two orderings for local function evaluations (excluding the global-to-local scatters needed to assemble ghost point data). The timings within this table for a single function evaluation were computed from overall execution times and iteration counts collected during a complete nonlinear simulation for a problem of matrix dimension 158,600 (see Section 6.1). These performance studies indicate a savings for the local function evaluation component of between 4 and 20% on the SP2, depending on the ratio of cache size to problem size. Similar results are reported in [19].

Table 5.2

Comparison of multicomponent orderings for local function evaluations with five degrees of freedom per node.

Number of Processors	Time for a Single Function Evaluation (sec)		Percentage Improvement
	Noninterlaced	Interlaced	
1	.983	.779	21
2	.477	.375	21
4	.237	.191	20
8	.115	.103	10
16	.061	.056	7

5.4.3. Grid Reordering. Another technique that can improve cache utilization is the reordering of grid entities. This is discussed in [44] for unstructured grids but not employed in the structured-grid computational examples of this paper, where its use would destroy one of the main advantages of structured grids, namely, the ability to employ direct addressing to locate data at neighboring vertices (or cells, in cell-centered codes). The idea behind grid reordering for enhanced cache residency in an edge-based CFD code is simple: vertices that share an edge need to have their data co-resident in the cache to compute the flux along the edge. If edges common to a vertex are ordered near each other, the data at that vertex may suffer as little as one (compulsory) cache miss during a flux-computation cycle. If edges are ordered in a greedy way, away from the initial set of edges, there may be low data miss rates on average throughout the entire domain for the entire cycle. It remains to be seen whether, in processors with deep memory hierarchies, the data locality enhancements possible with unstructured problems can overcome the overheads (in time and space) of indirect addressing.

5.5. Importance of Profiling. Profiling a code's overall performance for realistically sized problems, including timings, floating-point operations, computational rates, and message-passing activity (such as the number and size of messages sent and collective operations), is crucial for gaining an

understanding of sections that can benefit most from performance enhancements and parameter tuning. We have employed the automatic profiling capabilities within PETSc [4] to target performance enhancements for the parallel ΨNKS methods of this work.

Such profiling indicates that throughout a complete matrix-free nonlinear simulation using block sparse matrix data structures, the solution time (for various problem sizes and processor configurations, including all associated communication overheads) is roughly divided as follows:

- 1% for initialization (including parallel mesh partitioning and setup of vector scatter/gathers for handling ghost point data),
- 16% for formation of the linearized Newton systems (including right-hand-side vectors and lagged first-order preconditioner matrices),
- 82% for linear system solution, and
- 1% for other nonlinear solver operations.

Further breakdown of the linear solve phase indicates roughly 2% of overall time devoted preconditioner setup (including determination of additive Schwarz overlap and incomplete factorization of each processor's local submatrix), 17% for preconditioner application, 59% for matrix-free Jacobian-vector products, and the remainder for other vector operations. The efficiency of the sparse numerical linear algebra has resulted from tuned data structures and message-passing activity, as discussed above. The dominance of function evaluation time within the total solution process has led us to focus on the tuning of various numerical parameters to aim for the fewest possible overall fine grid function evaluations (or flux balances) needed to achieve steady state.

6. Numerical Experiments. We illustrate in this section the importance of some of the design decisions above in the context of a legacy CFD application that has been parallelized and accelerated through ΨNKS. We briefly describe the test problems and then discuss issues of boundary conditions, limiters, preconditioner quality, convergence tuning, and scalability.

6.1. Test Problems. The ONERA M6 wing is a standard three-dimensional test case, for which extensive experimental data is given in [66]. A frequently studied parameter combination combines a freestream Mach number of 0.84 with an angle of attack of 3.06^o. This transonic case gives rise to a characteristic λ-shock, as depicted within Fig. 6.1. As discussed in Section 3, the basis for our implementation is a legacy sequential Fortran77 code (JULIANNE) by Whitfield & Taylor [84].

Tests on three different successively doubled grid resolutions, the sizes of which are given in Table 6.1, provide a demonstration of the grid independence of the solution, as evidenced by the agreement of the overall values of C_L (lift) and C_D (drag) on the intermediate and finest grids in Fig. 6.2. Comparisons of the aerodynamic coefficients with the data in [66] replotted in Fig. 6.3 show reasonable agreement for an inviscid model.

Table 6.1

Test problem dimensions (system size includes boundary ghost nodes).

Grid Dimensions	System Size	Nonzeros in Jacobian Approx.
$50 \times 10 \times 10$	25,000	622,120
$98 \times 18 \times 18$	158,760	4,636,200
$194 \times 34 \times 34$	1,121,320	35,742,760

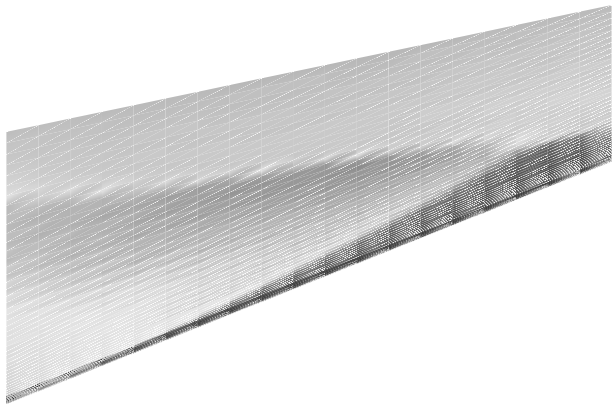

FIG. 6.1. *Mach number contours (the local tangential velocity magnitude divided by the local sound speed) of the converged flowfield on the upper wing surface for the finest grid.*

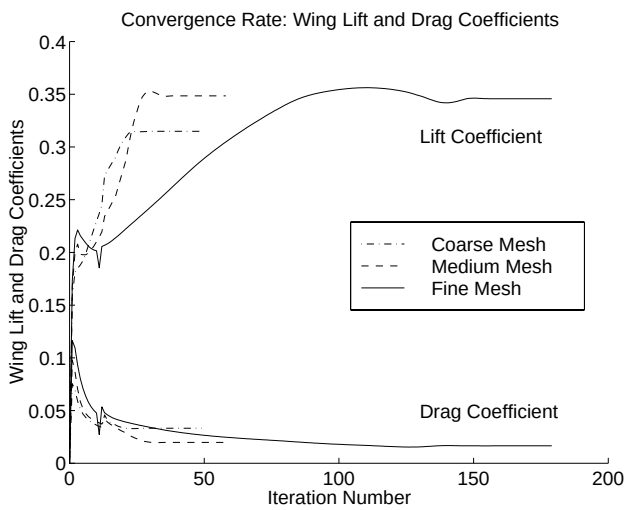

FIG. 6.2. *Dimensionless aerodynamic functionals C_L and C_D as functions of pseudo-transient iteration step on the coarse, medium, and fine grids.*

Thus far, we have employed only the simplest decomposition strategies in inducing locality in the Schwarz preconditioner: recursive bisection cuts along the i, j, and k coordinate directions to create subdomains. The order of the cuts is determined so as to produce subdomains that are as close to unit aspect ratio as possible *in logical space*, a strategy that minimizes communication volume. Given the 6:1 logical aspect ratio of the full domain, the first three cuts are made in the i direction. Beyond eight processors, the j and then the k directions are also partitioned. On the basis of investigations of domain decomposition preconditioners on other flow problems [11, 32], it is well known that there can be a strong interaction between flow direction and cutting planes. Our study of this phenomenon in the present problem and a collection of others will appear elsewhere. The main implication of the partitioning strategy in interpreting the scalability results of this section is that the linear conditioning is affected by the number and orientation of the cuts and that performance results for different ways of obtaining p-fold concurrency will generally differ from the samples included. Furthermore, for tunings

of ΨNKS in which the linear systems are only very incompletely converged, such as ours, the nonlinear trajectory of convergence can be affected by the cut.

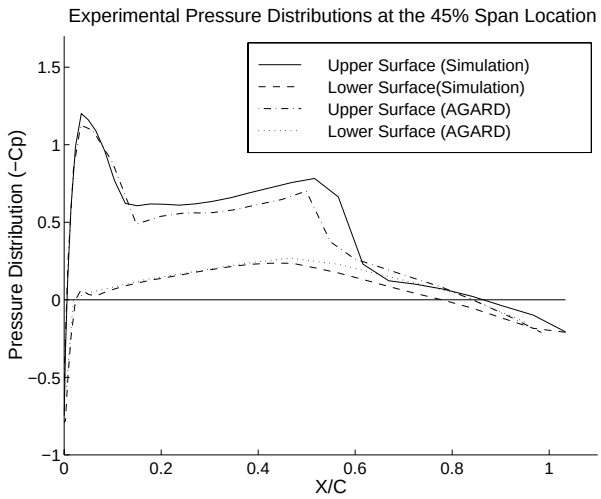

FIG. 6.3. *Dimensionless aerodynamic functional $C_P(x)$ from leading to trailing edge, upper and lower wing surfaces, at 45% of wingspan, and comparison with experiment.*

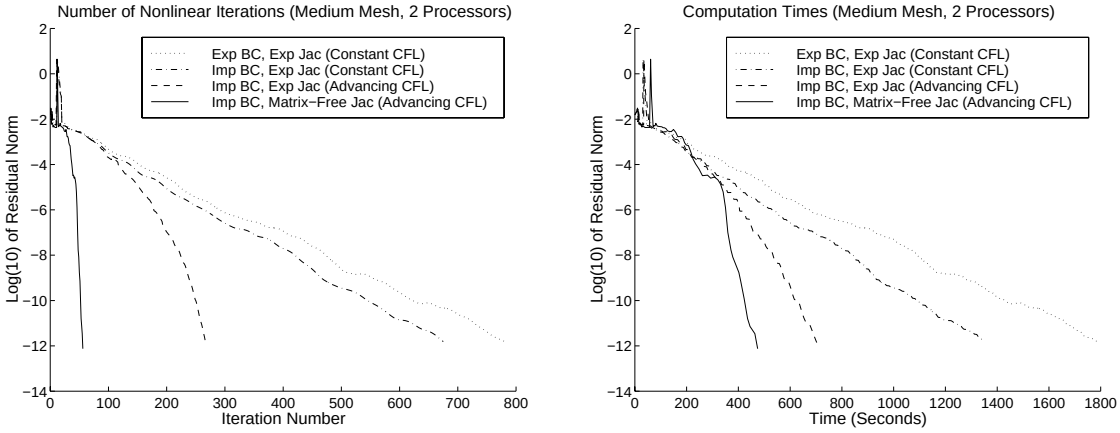

FIG. 6.4. *Comparison of four globalized solution algorithms, showing the cumulative advantages of implicit BCs, advancing CFL, and matrix-free representation of the Jacobian, in terms of both iteration count (number of pseudo-timesteps) and overall execution time on an IBM SP2.*

6.2. Implicit Boundary Conditions and Matrix-free Jacobian Action. Figure 6.4 compares the convergence in terms of both nonlinear iterations and total time of the various approaches under consideration: explicit boundary conditions (limiting CFL to 7.5), implicit boundary conditions with the same CFL, implicit boundary conditions with advancing CFL, and implicit boundary conditions with advancing CFL and matrix-free application of the Jacobian. This figure presents performance data for the medium mesh problem run on two processors of the SP2; the relative convergence rates and timings are analogous for the other test problems under consideration for various machines and processor configurations. Whenever required, the explicit Jacobian is computed to first-order accu-

racy in space via finite differences, stored with the block sparse scheme discussed in Section 5.4, and refreshed once every ten pseudo-timesteps. This Jacobian served as the left-hand-side matrix of the Newton systems (corresponding to (2.11)) or as the preconditioner for the matrix-free method. This frequency of Jacobian recomputation provides a reasonable trade-off in terms of convergence rate and the computational cost of matrix evaluation.

For the cases in Fig. 6.4, we solved the linearized Newton systems approximately with a relative linear convergence tolerance of 10^{-2} using preconditioned GMRES. For the matrix-free variant we employed the additive Schwarz preconditioner with an overlap of 1 cell, and for the other cases we used the cheaper but sufficiently powerful block Jacobi method (equivalent to additive Schwarz with no overlap); both versions used one subdomain block per processor, solved with ILU(0) (the issue of preconditioner quality is discussed in Section 6.4). When retaining a fixed CFL of 7.5, only 3–4 linear iterations were required for each step to reach the 10^{-2} tolerance for both explicit and implicit boundary condition variants. Experiments indicate that because of the mismatch of first-order and second-order schemes in the left- and right-hand sides of these defect correction methods, more accurate linear solves and more frequent Jacobian refreshment do not appreciably accelerate convergence. For the cases with increasing CFL, the linear systems become more challenging to solve as they transition toward true Newton systems, requiring up to 30 iterations to reach the specified 10^{-2} tolerance when CFL reaches its maximum value of 10^5. The quality of linear solves is further discussed in Section 6.5.

The key observation from this data is that the combination of implicit boundary conditions coupled with the higher-order discretization enabled by the matrix-free technique solves the nonlinear problem to machine precision several times faster than does the defect correction method. Neither the use of implicit boundary conditions alone nor the use of increasing CFL with a low-order Jacobian allows the approach of quadratic convergence. We also note, on the basis of analogous experiments with the other problem sizes considered in the paper, that the impact of using this combination increases with problem size.

In addition to pseudo-time continuation, the original JULIANNE code employs a subtle form of continuation in boundary conditions. The boundary conditions for mass density and energy density at the impermeable wing surface are initially of simple Neumann type, and are switched to full characteristic boundary conditions only after the tenth pseudo-timestep. (This accounts for the spikes seen in the steady-state residual norm histories in Figs. 6.4–6.8.) We have found this device to be important for robustness in a fully implicit approach in which the steady-state residuals are evaluated to second order from the outset. Alternatively, we have employed full characteristic boundary conditions for all variables from the outset in a first-order discretization, and then switched to a second-order discretization after approximately 10^{-2} reduction in the steady-state residual norm.

6.3. Differentiable Limiters. As discussed in Section 3.3, differentiability of limiters is essential when using derivative information in the numerical scheme. When using matrix-free differencing approximations of Jacobian-vector products, differentiable limiters are required to avoid breakdown (as manifested by the introduction of `NaN`s in the iterate update) or stagnation. Figure 6.5 shows convergence stagnation when the coarse mesh is run with the nondifferentiable Van Leer limiter and the penetration of this stagnation with the Van Albada limiter.

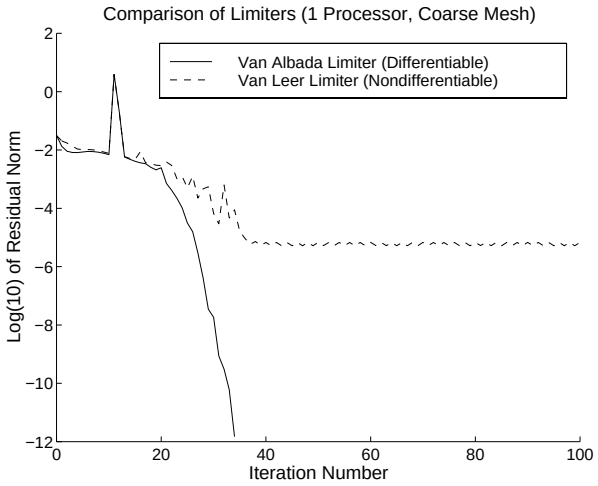

FIG. 6.5. *Illustration of the "hang up" characteristic of nondifferentiable limiters within a nonlinearly implicit critical flow.*

6.4. Preconditioner Quality. Preconditioner quality dramatically impacts the overall efficiency of the parallel ΨNKS methodology, as demonstrated in the closely related Figs. 6.6 and 6.7 for several variants of additive Schwarz techniques. The graphs within these figures compare convergence rate (in terms of relative residual norm) versus both nonlinear iteration number (left-hand graphs) and time (right-hand graphs) for the medium mesh on 16 processors of an IBM SP2; analogous results were achieved for other problem sizes and processor configurations. All plotted runs use preconditioned restarted GMRES with a Krylov subspace of maximum dimension 30 and a fixed relative convergence tolerance of 10^{-2} (this issue is discussed in the following section); each processor hosts a single preconditioner block, which is solved via point-block ILU(0).

We contrast in Fig. 6.6 a zero-overlap (subdomain-)block Jacobi preconditioner with a two-cell-overlap additive Schwarz method (ASM) and a two-cell-overlap restricted additive Schwarz method (RASM), which were presented in Section 2.4. We observe that modest overlap is important in providing a high-quality preconditioner that allows the overall numerical scheme to progress rapidly in the nonlinear sense. Further, we see that the RASM method, which eliminates communication during the interpolation phase, not only saves time in terms of communication overhead, but also provides more powerful preconditioning, as evidenced by faster convergence in terms of the nonlinear iteration count.

Figure 6.7 contrasts various degrees of overlap for RASM. In particular, we see that two-cell overlap provides a good balance in terms of power and cost. Less overlap trades off cheaper cost per iteration for a preconditioner that does not allow the nonlinear iterations to converge proceed as rapidly, while more overlap is costly to apply and does not sufficiently contribute to faster nonlinear convergence. Similar behavior was observed for structured-grid problems at other problem sizes and processor configurations, even when using other criteria to determine linear inner iteration convergence.

In the context of unstructured tetrahedral grids, where each successive level of subdomain overlap is defined by following edges incident on vertices belonging to the subdomain at the previous level, our experience with overlap for RASM has led to a more extreme trade-off. Cost per iteration rises rapidly with overlap, since a vertex may be a member of 15 or more tetrahedra, and iteration reduction does

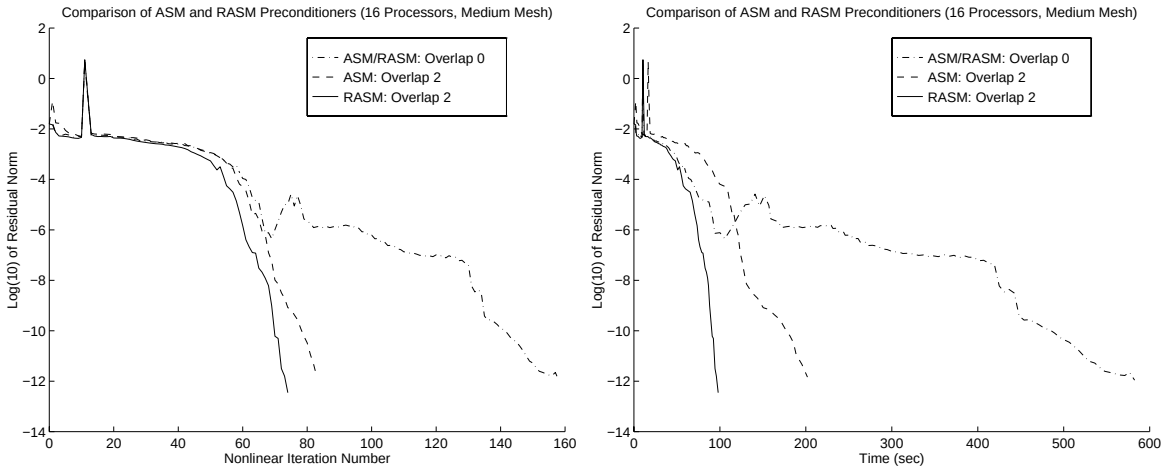

FIG. 6.6. *Comparison of three domain-decomposed preconditioners: subdomain-block Jacobi, standard additive Schwarz with overlap of 2 cells, and restricted additive Schwarz with overlap of 2 cells. All methods solve point-block ILU(0) on 16 subdomains on an IBM SP2.*

not significantly counterbalance this cost, for the weak levels of linear convergence required. We often use no overlap in such cases.

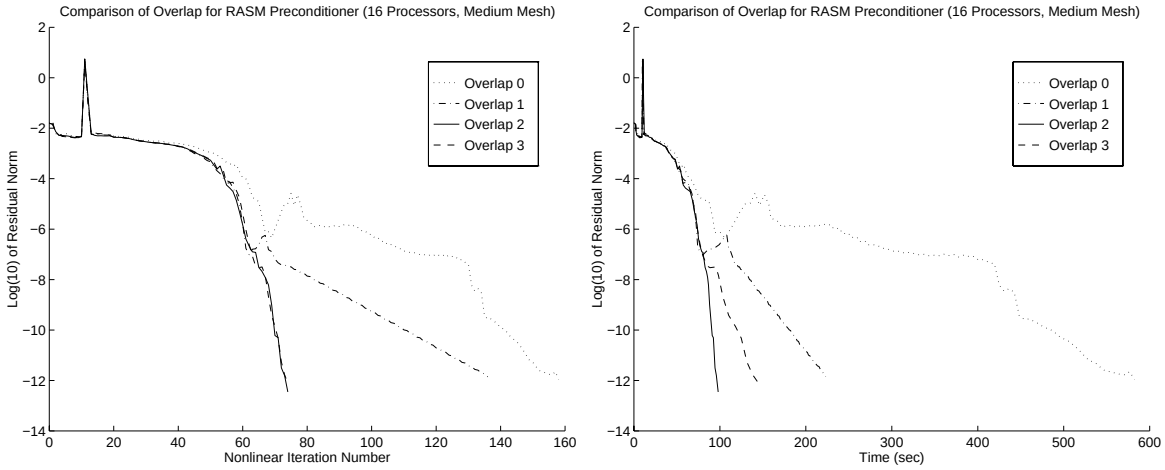

FIG. 6.7. *Comparison of four domain-decomposed preconditioners: subdomain-block Jacobi and restricted additive Schwarz with overlap of 1, 2, and 3 cells. All methods solve point-block ILU(0) on 16 subdomains on an IBM SP2.*

6.5. Convergence Tuning. Convergence tuning for terminating the inner Krylov iterations in the Newton correction is tightly coupled with preconditioner quality, and the two must be considered in concert. Figure 6.8 contrasts various convergence criteria using the restricted additive Schwarz preconditioner with an overlap of two cells, which was shown in Section 6.4 to be an effective choice for the problems considered in this work. This figure presents convergence data in terms of solution times and work per linear iteration for three methods: fixed Krylov subspace dimension, fixed relative tolerance for residual norm reduction, and a hybrid strategy that terminates upon the earlier of satisfying a relative residual norm reduction or exceeding a Krylov subspace dimension. These graphs focus on

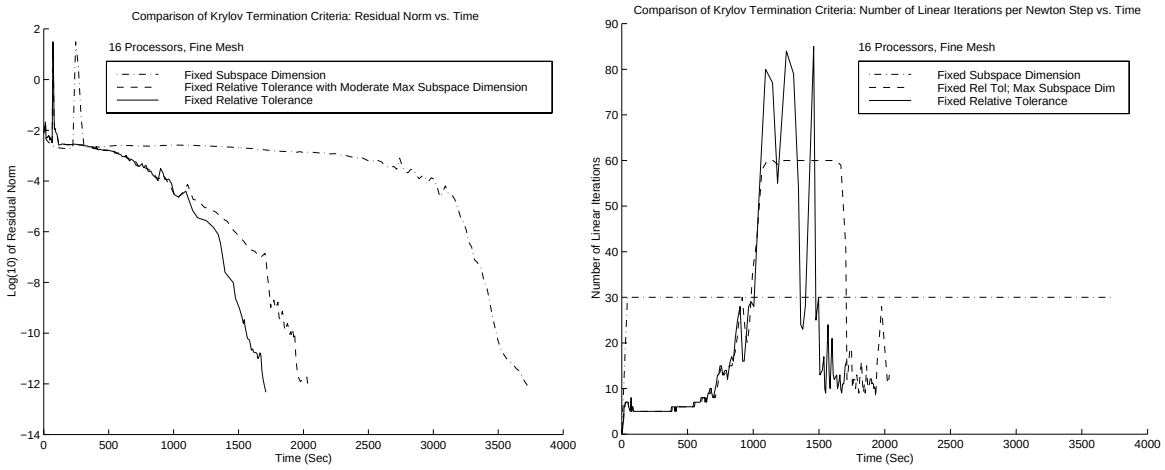

FIG. 6.8. *Comparison of three different tunings of the linear convergence tolerance: fixed work, fixed relative tolerance, and a loose hybrid.*

the 16-processor case with the fine mesh, although they are representative of experiments on various numbers of processors for different problem sizes. The left-hand graph within Fig. 6.8, which plots the log of the residual norm versus solution time, indicates that using a loose fixed relative tolerance of 10^{-2} is preferable over the other choices when a sufficiently robust preconditioner is employed.

The corresponding right-hand graph better illustrates where within the total simulation process the bulk of computational work occurs within the linear solves. In particular, we see that the common practice of specifying a fixed amount of work per nonlinear iteration does not fare as well as alternatives that focus linear solver work where it is truly needed during the second and third phases of the ΨNKS process (recall Section 2.1). The fixed-work case for this problem employed thirty iterations of GMRES; experiments with fewer GMRES iterations and more frequent restarts encountered stagnation before converging to steady state. In contrast, the approaches that use a fixed relative tolerance or hybrid scheme fared much better.

Strategies that dynamically adjust the linear tolerances to achieve greater relative residual reduction during steps of the final ΨNKS phase and thereby aim for quadratic convergence (e.g., [24]), have been of limited benefit when a sufficiently robust linear solver is employed. We note that others with strong incentives to watch the "bottom line" of execution time in production line design use have converged on fairly loose tolerances for inner linear iterations [87]. However, further investigation of tuning such strategies should prove valuable.

Figure 6.9 illustrates the evolution of the shock structure (characterized by the number of supersonic mesh points) as a function of time for the medium and fine meshes. Notice that the number of supersonic points scales approximately with the number of gridpoints (factor of 8 between the two curves), reflecting the convergence to a grid-independent feature in physical space. We also observe that each run from a uniform flow initial iterate passes through the same intermediate condition during which the supersonic region is larger than its asymptotic value. The duration of this period is inversely proportional to the resolution, suggesting that the shock advances in units of gridpoints per iteration, based on a cell-size-dependent CFL, not with physical temporal accuracy. Properly applied grid sequencing would suffer

this period of settling of shock location on the coarsest grid only.

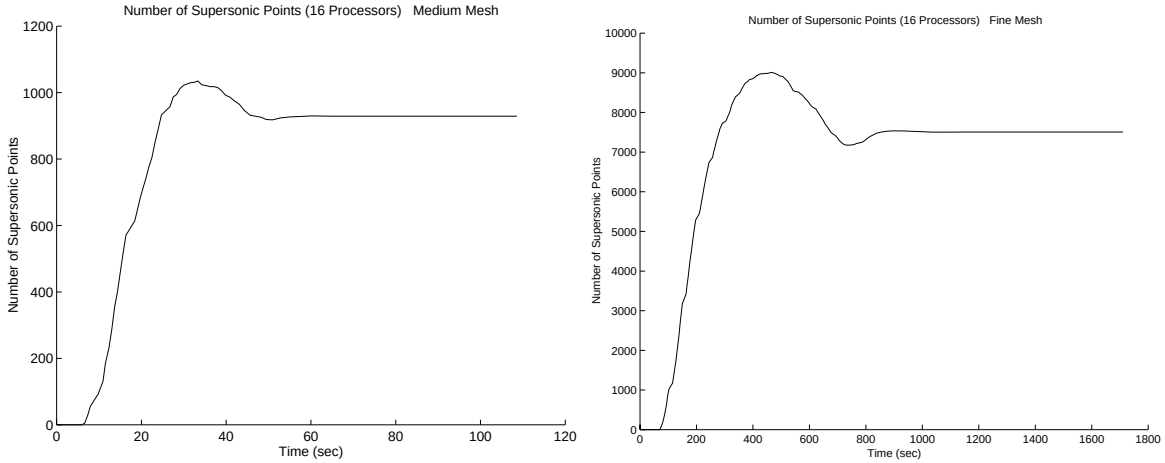

FIG. 6.9. *Illustration of the evolution of the shock structure as reflected in the number of gridpoints contained in the supersonic "bubble" on the upper surface of the airfoil, as a function of pseudo-timestep number, for the medium and fine grids.*

6.6. Scalability. There are many aspects to parallel scalability in a nonlinear PDE problem. We may usefully distinguish between the numerical scalability of the algorithm (reflecting how the number of iterations depends upon the partitioning, which makes the "best preconditioner" at each granularity algebraically different) and the implementation scalability (reflecting how well a given "market basket" of operations within a single iteration at some level executes at different granularities). We also report fixed-problem-size scalability and fixed-memory-per-node or "Gustafson" scalability.

In Table 6.2 we present computation rates on an IBM SP2 for the matrix-vector product and an entire linear solve using an explicitly stored Jacobian with implicit boundary conditions averaged over a fixed number of Newton corrections of a particular pseudo-timestep. The linear Newton systems are solved using restarted GMRES with a Krylov subspace of maximum dimension 30 and block Jacobi preconditioning, where each processor has one block that is solved with ILU(0). The speedup over two processors (the smallest number on which the entire problem fits, when both the explicit Jacobian and its preconditioner must be stored) is given in parentheses in the tables. To put in perspective the average single-node performance of 73.5 Mflop/s (parallel overheads included) for the block-sparse linear solution of the 2-processor case, we note that the peak performance of one processor of the quad-issue IBM SP2 is 266 Mflop/s, the *dense* LINPACK-100 benchmark produces 130 Mflop/s, and a *sparse* matrix-vector product that uses the standard compressed sparse row format (CSR) attains 27 Mflop/s.

We next present in Table 6.3 similar runs for the same problem on the fine mesh, which produces a system that is roughly eight times as large as the previous one. For this problem of 1,121,320 unknowns, the computation rate on sixteen processors for the matrix-vector product was 1.28 Gflop/s, while the complete linear solve achieves 1.01 Gflop/s. On sixty-four processors the matrix-vector product runs at 4.22 Gflop/s, while the complete linear solve achieves 3.74 Gflop/s. The data presented here is based on flop counters embedded in the PETSc library routines and pertains to the solvers only. The function evaluation and Jacobian evaluation application routines are not yet instrumented for floating-point operation counting; therefore, self-contained linear solutions (with explicitly stored matrix operations,

Table 6.2
Performance on medium grid problem (computation rate in Mflop/s).

Number of	Matrix-Vector Products		Linear Solves	
Processors	Mflop/s	Speedup	Mflop/s	Speedup
2	179	–	147	–
4	337	(1.9)	286	(2.0)
8	620	(3.5)	540	(3.7)
16	1137	(6.4)	994	(6.8)
32	2038	(11.4)	1785	(12.1)
64	3003	(16.8)	2546	(17.3)

Table 6.3
Performance on fine grid problem (computation rate in Mflop/s).

Number of	Matrix-Vector Products		Linear Solves	
Processors	Mflop/s	Speedup	Mflop/s	Speedup
16	1281	–	1011	–
32	2483	(1.9)	2154	(2.1)
64	4217	(3.3)	3744	(3.7)

not matrix-free approximations) were employed for these computational rate tests. Scalability analysis in terms of time for the matrix-free variants is presented in Fig. 6.10 and Table 6.6.

Table 6.4 presents timing data for a single NKS iteration, including evaluation of the right-hand-side vector as well as the linear solve, on the SP2 for 8 and 64 processor cases. We partition the 8-processor case as $8 \times 1 \times 1$ for the medium mesh and the 64-processor case as $16 \times 2 \times 2$ for the fine mesh. These particular cases have identical local mesh configurations, so that the problems require the same amount of memory per processor and therefore have similar caching profiles (which controls for one of the main effects that renders parallel performance studies difficult).

Table 6.4
Time (s) for one nonlinear iteration on an SP2.

Solution Technique	Medium Mesh 8 Procs (8x1x1)	Fine Mesh 64 Procs (16x2x2)	Percent Difference
Matrix-explicit	.39	.46	15
Matrix-free	1.20	1.60	25

It is encouraging that the parallel matrix-free version of the code also has good parallel efficiency in the Gustafson (memory-constrained) sense. For example, given two versions of the same problem, one eight times larger than the other (a factor of two mesh refinement in each of three dimensions), and given eight times as many processors to run the larger case, the two problems have per-iteration costs within 5–15% of being identical for the matrix-explicit case, and within 12–25% for the matrix-free

case. The slightly lower efficiency for the matrix-free case arises because of the dominance of time for function evaluations for this method.

Despite the slightly poorer scalability of the matrix-free method in the memory-constrained sense, the faster convergence of the matrix-free method relative to the matrix-explicit method becomes increasingly important as problem size increases. Table 6.5 presents the time for total nonlinear solution using both matrix-explicit and matrix-free approaches for the coarse, medium, and fine meshes on 1, 8, and 64, processors, respectively. We see that in each case the matrix-free method performs more than twice as fast as its matrix-explicit counterpart for a given problem size and processor configuration.

TABLE 6.5
Time (s) for total nonlinear solution on an SP2.

Grid	Processor Configuation	Matrix Free	Matrix Explicit	Ratio (ME/MF)
Coarse	1	64.4	142.6	2.2
Medium	8 ($8 \times 1 \times 1$)	217.5	499.2	2.3
Fine	64 ($16 \times 2 \times 2$)	1019.3	2353.5	2.3

The plots in Fig. 6.10 show the scaling of the complete nonlinear simulation for the fine mesh on 16, 32, and 64 processors of an IBM SP with 120 MHz P2SC nodes with two 128 MB memory cards each and a TB3 switch; analogous performance trends are seen also on the SP2. The data indicate a modest decrease in nonlinear convergence rate as the number of processors grows; overall solution times scale reasonably well.

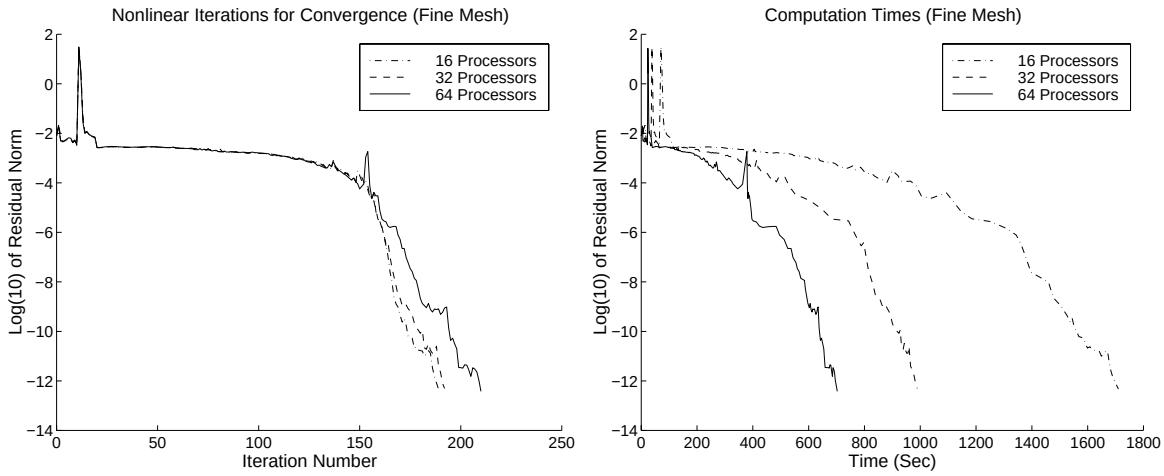

FIG. 6.10. *Scalability of nonlinear solver (fine mesh).*

Finally, Table 6.6 compares the scalability on two contemporary high-performance parallel computers, the IBM SP2 and the Cray T3E. Timings are given for a single matrix-free NKS iteration on the fine mesh, including evaluation of the right-hand-side vector as well as the linear solve. We consider 16, 32, and 64-processor configurations and present speedup over the 16-processor case.

Table 6.6

Scalability on the SP2, T3E: Timings for one nonlinear iteration on fine mesh.

Machine	Number of Processors	Time (s)	Speedup over 16 Procs.
IBM SP2	16	4.39	–
	32	2.46	1.8
	64	1.60	2.7
Cray T3E	16	5.15	–
	32	2.71	1.9
	64	1.71	3.0

7. Conclusion. We have shown that the pseudo-transient matrix-free Newton-Krylov-Schwarz (ΨNKS) methodology is robust and reasonably efficient for the parallel solution of large-scale CFD problems, as demonstrated here for the three-dimensional compressible Euler equations. For a given problem class, the various parameters that affect algorithmic performance can be systematically studied and tuned. We have also shown how any one of several missing or mistuned functionalities can cause convergence to slow or to hang at a level far from the best attainable. An integrated approach to investigating innovative numerical methods and developing software is crucial to designing efficient, reusable tools.

In our experience, conversion of a legacy code to the matrix-free NKS framework is a "filtering" process. During the conversion to a Newton method, initially unrecognized explicit updates may be discovered and unrecognized nondifferentiability or destabilizing sensitivity of local gradient information about a function may be stumbled upon.

Additional work in progress includes the following:

- Investigation of techniques for evaluating relative function noise. As indicated in Section 4.2, the robustness of matrix-free methods is potentially sensitive to the finite differencing parameter for the Fréchet derivative. Its selection should be automatically adaptive.
- Incorporation of multilevel methods. Coarse-to-fine grid sequencing is a continuation tool as powerful as pseudo-transient continuation. Linear multilevel methods that revisit the coarse levels are also arithmetically optimal solvers for the Newton correction equations. Although such methods may be very difficult for nonexperts to tune in anisotropic, nonmonotone-inverse problems, they can be used effectively as preconditioners, as in [48]. Nonlinear multilevel methods can also be directly accelerated by nonlinear Krylov methods, as in [60].
- Problem-adaptive domain partitioning. The deterioration of convergence rate of Schwarz methods with increasing subdomain granularity can be minimized when the cuts introduced to create the partitions are along edges in the Jacobian matrix with minimal coefficient weight (in some norm to be made more precise with further research). Obviously, load balance and communication-to-computation ratio criteria may conflict with coefficient weight criteria in partitioning, but there is often significant latitude (particularly in unstructured problems) in choosing partitions, which could be exploited by such problem-specific knowledge.
- Nonlinear Schwarz methods. The global Newton method advocated in this paper is an asymptotically attractive implicit strategy, but many important problems have the property of being strongly nonlinear in limited subregions and weakly nonlinear (or even linear) elsewhere. When

such problems are solved with global Newton methods, work is often wasted on the well-behaved regions while the solution slowly evolves in the strongly nonlinear regions (see, e.g., [13]). Nonlinear Schwarz methods, including asynchronous varieties, can adapt the work to the degree of nonlinearity in the way that adaptive mesh refinement methods can adapt the work to solution activity. We expect that NKS will fit into an overall solution strategy as a closing strategy, after nonlinear Schwarz (employing NK as a subdomain solver) is adaptively applied as an opening strategy.

Acknowledgments. The authors owe a large debt of gratitude to Professor Dave Whitfield of the Engineering Research Center at Mississippi State University for providing JULIANNE as the legacy code that drove several aspects of this work. Satish Balay and Barry Smith of Argonne National Laboratory co-developed the PETSc software employed in this paper, together with Gropp and McInnes, under the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. Dr. George K. Lea of the National Science Foundation supported the work of the Keyes and McInnes at Old Dominion University under ECS-9527169. Dr. James L. Thomas of the NASA Langley Research Center supported the work of Satish Balay at Old Dominion University under NAG-1-1692, and all four authors have collaborated while in residence at ICASE, where early results of this paper were employed in a "Bring Your Own Code" Workshop on the Parallelization of PDE-based Codes in December 1996. Barry Smith, Kees Oosterlee, and Alex Povitsky provided valuable feedback on early versions.

REFERENCES

[1] K. Ajmani, W.-F. Ng, and M.-S. Liou, *Preconditioned conjugate gradient methods for the Navier-Stokes equations*, J. Computational Physics, 110 (1994), pp. 68–81.

[2] G. D. V. Albada, B. van Leer, and W. W. Roberts, Jr., *A comparative study of computational methods in cosmic gas dynamics*, Astronomics and Astrophysics, 108 (1982), pp. 76–84.

[3] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhauser Press, 1997, pp. 163–202.

[4] ——, *PETSc 2.0 users manual*, Tech. Report ANL-95/11 - Revision 2.0.22, Argonne National Laboratory, April 1998.

[5] ——, *PETSc home page*. http://www.mcs.anl.gov/petsc, May 1998.

[6] T. J. Barth and S. W. Linton, *An unstructured mesh Newton solver for fluid flow and its parallel implementation*, Tech. Report 95-0221, AIAA, 1995.

[7] P. Bjorstad, R. Espedal, and D. E. Keyes, eds., *Proceedings of the Ninth International Conference on Domain Decomposition Methods*, Wiley, 1998 (to appear).

[8] P. N. Brown and A. C. Hindmarsh, *Matrix-free methods for stiff systems of ODE's*, SIAM J. Numerical Analysis, 23 (1986), pp. 610–638.

[9] P. N. Brown and Y. Saad, *Hybrid Krylov methods for nonlinear systems of equations*, SIAM J. Scientific and Statistical Computing, 11 (1990), pp. 450–481.

[10] X.-C. Cai, *Some domain decomposition algorithms for nonselfadjoint elliptic and parabolic partial differential equations (Ph.D. thesis)*, Tech. Report 461, Courant Institute, September 1989.

[11] X.-C. Cai, C. Farhat, and M. Sarkis, *Schwarz methods for the unsteady compressible Navier-Stokes equations on unstructured meshes*, in Proceedings of the Eighth International Conference on Domain Decomposition Methods, Wiley, 1997, pp. 453–460.

[12] ——, *A minimum overlap restricted additive Schwarz preconditioner and applications in 3d flow simulations*, in Proceedings of the Tenth International Conference on Domain Decomposition Methods, AMS, 1998, pp. 238–244.

[13] X.-C. Cai, W. D. Gropp, D. E. Keyes, R. G. Melvin, and D. P. Young, *Parallel Newton-Krylov-Schwarz algorithms for the transonic full potential equation*, SIAM J. Scientific Computing, 19 (1998), pp. 246–265.

[14] X.-C. Cai, W. D. Gropp, D. E. Keyes, and M. D. Tidriri, *Newton-Krylov-Schwarz methods in CFD*, in Proceedings of the International Workshop on Numerical Methods for the Navier-Stokes Equations, Vieweg, 1995, pp. 17–30.

[15] X.-C. Cai, D. E. Keyes, and V. Venkatakrishnan, *Newton-Krylov-Schwarz: An implicit solver for CFD*, in Proceedings of the Eighth International Conference on Domain Decomposition Methods, Wiley, 1997, pp. 387–400.

[16] X.-C. Cai and M. Sarkis, *A restricted additive Schwarz preconditioner for general sparse linear systems*, Tech. Report CU-CS 843-97, Computer Science Department, University of Colorado-Boulder, 1997. (accepted by SIAM J. of Scientific Computing).

[17] T. F. Chan and K. R. Jackson, *Nonlinearly preconditioner Krylov subspace methods for discrete Newton algorithms*, SIAM J. Scientific and Statistical Computing, 5 (1984), pp. 535–542.

[18] D. E. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture*, Morgan-Kaufman, 1998.

[19] V. D. Decyk, S. R. Karmesin, A. de Boer, and P. C. Kuewer, *Optimization of particle-in-cell codes on RISC processors*, Tech. Report CRPC-95-6, CRPC, Oct. 1995.

[20] R. Dembo, S. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numerical Analysis, 19 (1982), pp. 400–408.

[21] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, 1983.

[22] J. E. Dennis and V. Torczon, *Direct search methods on parallel machines*, SIAM J. Optimization, 1 (1991), pp. 448–474.

[23] M. Dryja and O. B. Widlund, *An additive variant of the Schwarz alternating method for the case of many subregions*, Tech. Report 339, Courant Institute, NYU, 1987.

[24] S. C. Eisenstat and H. F. Walker, *Choosing the forcing terms in an inexact Newton method*, SIAM J. Scientific Computing, 17 (1996), pp. 16–32.

[25] A. Ern, V. Giovangigli, D. E. Keyes, and M. D. Smooke, *Towards polyalgorithmic linear system solvers for nonlinear elliptic systems*, SIAM J. Scientific Computing, 15 (1994), pp. 681–703.

[26] C. Farhat, M. Lesoinne, and N. Maman, *Mixed explicit/implicit time integration of coupled aeroelastic problems: Three-field formulation, geometric conservation and distributed solution*, International J. for Numerical Methods in Fluids, 21 (1995), pp. 807–835.

[27] R. W. Freund, *A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems*, SIAM J. Scientific and Statistical Computing, 14 (1993), pp. 470–482.

[28] R. W. Freund, G. H. Golub, and N. M. Nachtigal, *Iterative solution of linear systems*, Acta Numerica, (1992), pp. 57–100.

[29] C. W. Gear and Y. Saad, *Iterative solution of linear equations in ODE codes*, SIAM J. Scientific and Statistical Computing, 4 (1983), pp. 583–601.

[30] W. D. Gropp and D. E. Keyes, *Complexity of parallel implementation of domain decomposition techniques for elliptic partial differential equations*, SIAM J. Scientific and Statistical Computing, 9 (1988), pp. 312–326.

[31] ——, *Domain decomposition on parallel computers*, Impact of Computing in Science and Engineering, 1 (1989), pp. 421–439.

[32] W. D. Gropp, D. E. Keyes, and J. S. Mounts, *Implicit domain decomposition algorithms for steady, compressible aerodynamics*, in Sixth International Symposium on Domain Decomposition Methods, A. Quarteroni, J. Periaux, Y. A. Kuznetsov, and O. B. Widlund, eds., Providence, 1994, AMS, pp. 203–213.

[33] C. Hirsch, *Numerical Computation of Internal and External Flows: Volume 2: Computational Methods for Inviscid and Viscous Flows*, John Wiley and Sons, 1988.

[34] C. B. Jenssen and P. A. Weinerfelt, *Coarse grid correction scheme for implicit multiblock Euler calculations*, AIAA J., 33 (1995), pp. 1816–1821.

[35] ——, *Parallel implicit time-accurate Navier-Stokes computations using coarse grid correction*, AIAA J., 36 (1995), pp. 946–951.

[36] H. Jiang and P. A. Forsyth, *Robust linear and nonlinear strategies for solution of the transonic Euler equations*, Computers and Fluids, 24 (1995), pp. 753–770.

[37] Z. Johann, T. J. R. Hughes, and F. Shakib, *A globally convergent matrix-free algorithm for implicit time-marching schemes arising in finite element analysis in fluids*, Computational Methods in Applied Mechanics and Engineering, 87 (1991), pp. 281–304.

[38] D. E. Kaushik, D. E. Keyes, and B. F. Smith, *On the interaction of architecture and algorithm in the domain-based parallelization of an unstructured grid incompressible flow code*, in Proceedings of the Tenth International Conference on Domain Decomposition Methods, AMS, 1998, pp. 311–319.

[39] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, SIAM, 1995.

[40] C. T. Kelley and D. E. Keyes, *Convergence analysis of pseudo-transient continuation*, SIAM J. Numerical Analysis, 35 (1998), pp. 508–523.

[41] D. E. Keyes, *Domain decomposition methods for the parallel computation of reacting flows*, Computer Physics Communications, 53 (1989), pp. 181–200.

[42] ——, *Aerodynamic applications of Newton-Krylov-Schwarz solvers*, in Proceedings of the 14th International Conference on Numerical Methods in Fluid Dynamics, Springer, 1995, pp. 1–20.

[43] D. E. Keyes and W. D. Gropp, *Domain decomposition techniques for nonsymmetric systems of elliptic boundary value problems: Examples from CFD*, in Second International Symposium on Domain Decomposition Methods, T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Philadelphia, 1989, SIAM, pp. 321–339.

[44] D. E. Keyes, D. K. Kaushik, and B. F. Smith, *Prospects for CFD on petaflops systems*, in

CFD Review 1997, Wiley, 1998 (to appear).

[45] D. E. Keyes and M. D. Smooke, *A parallelized elliptic solver for reacting flows*, in Parallel Computations and Their Impact on Mechanics, A. K. Noor, ed., ASME, 1987, pp. 375–402.

[46] D. A. Knoll, P. R. McHugh, and D. E. Keyes, *Newton-Krylov methods for low Mach number compressible combustion*, AIAA J., 34 (1996), pp. 961–967.

[47] D. A. Knoll and W. J. Rider, *A multilevel Newton-Krylov method for nonsymmetric, nonlinear boundary value problems*, tech. report, Los Alamos National Laboratory, 1997.

[48] ——, *A multilevel preconditioned Newton-Krylov method*, tech. report, Los Alamos National Laboratory, 1997.

[49] D. A. Knoll, W. J. Rider, and G. L. Olson, *Newton-Krylov methods applied to nonequilibrium radiation diffusion*, tech. report, Los Alamos National Laboratory, 1998.

[50] J. Mandel, C. Farhat, and X.-C. Cai, eds., *Proceedings of the Tenth International Conference on Domain Decomposition Methods*, AMS, 1998.

[51] D. J. Mavriplis, *On convergence acceleration techniques for unstructured meshes*, Tech. Report 98-2966, AIAA, 1998.

[52] P. R. McHugh and D. A. Knoll, *Inexact Newton's method solutions to the incompressible Navier-Stokes and energy equations using standard and matrix-free implementations*, in Proceedings of the AIAA Eleventh Annual Computational Fluid Dynamics Conference, 1993.

[53] P. R. McHugh, D. A. Knoll, V. A. Mousseau, and G. A. Hansen, *An investigation of Newton-Krylov solution techniques for low Mach number compressible flow*, in Proceedings of the ASME Fluids Engineering Division Summer Meeting, 1995.

[54] L. C. McInnes and J. J. Moré. Unpublished information, Mathematics and Computer Science Division, Argonne National Laboratory, 1998.

[55] MPI Forum, *MPI: A message-passing interface standard*, International J. for Supercomputing Applications, 8 (1994). http://www.mpi-forum.org.

[56] W. Mulder and B. van Leer, *Experiments with implicit upwind methods for the Euler equations*, J. Computational Physics, 59 (1985), pp. 232–246.

[57] N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen, *How fast are nonsymmetric matrix iterations?*, SIAM J. Matrix Analysis and Applications, 13 (1992), pp. 778–795.

[58] J. C. Newton, W. K. Anderson, and D. L. Whitfield, *Multidisciplinary sensitivity derivatives using complex variables*, Tech. Report 98-08, Mississippi State University Engineering Research Center, July 1998.

[59] E. J. Nielsen, R. W. Walters, W. K. Anderson, and D. E. Keyes, *Application of Newton-Krylov methodology to a three-dimensional unstructured Euler code*, Tech. Report 95-1733, AIAA, 1995.

[60] C. W. Oosterlee and T. Washio, *On Krylov subspace acceleration of nonlinear multigrid schemes for rotating flow problems*, tech. report, Institute for Algorithms and Scientific Computing, GMD, 1998. Submitted to SIAM J. of Scientific Computing.

[61] P. D. Orkwis, *Comparison of Newton's and quasi-Newton's method solvers for the Navier-Stokes equations*, AIAA J., 31 (1993), pp. 832–836.

[62] M. Pernice, L. Zhou, and H. F. Walker, *Parallel solution of nonlinear partial differential equations using a globalized inexact Newton-Krylov-Schwarz method*, Tech. Report 48, Univer-

sity of Utah Center for High Performance Computing, 1997.

[63] N. Qin, D. K. Ludlow, and S. T. Shaw, *A matrix-free preconditioned Newton/GMRES method for Navier-Stokes equations.* Submitted to J. of Computational Physics, 1997.

[64] P. L. Roe, *Approximate Riemann solvers, parameter vector, and difference schemes*, J. Computational Physics, 43 (1981), pp. 357–372.

[65] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Scientific and Statistical Computing, 7 (1986), pp. 856–869.

[66] V. Schmitt and F. Charpin, *Pressure distributions on the ONERA M6 wing at transonic Mach numbers*, Tech. Report AR-138, AGARD, May 1979.

[67] R. Schreiber and H. B. Keller, *Driven cavity flows by efficient numerical techniques*, J. Computational Physics, 49 (1983), pp. 310–333.

[68] F. Shakib, T. J. R. Hughes, and Z. Johan, *Element-by-element algorithms for nonsymmetric matrix problems arising in fluids*, in Superlarge Problems in Computational Mechanics, Plenum, 1987, pp. 1–34.

[69] B. F. Smith, P. Bjørstad, and W. D. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, 1996.

[70] M. D. Smooke, *An error estimate for the modified Newton method with applications to the solution of nonlinear two-point boundary value problems*, J. Optimization Theory and Applications, 39 (1983), pp. 489–511.

[71] M. D. Smooke and R. M. Mattheij, *On the solution of nonlinear two-point boundary value problems on successively refined grids*, Applied Numerical Mathematics, 1 (1985), pp. 463–487.

[72] P. Sonneveld, *CGS, a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Scientific and Statistical Computing, 10 (1989), pp. 36–52.

[73] M. D. Tidriri, *Krylov methods for compressible flows*, Tech. Report 95-48, ICASE, June 1995.

[74] ——, *Schwarz-based algorithms for compressible flows*, Tech. Report 96-4, ICASE, Jan. 1996.

[75] E. Turkel, *Review of preconditioning methods for fluid dynamics*, Applied Numerical Mathematics, 12 (1993), pp. 27–46.

[76] H. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging varient of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Scientific and Statistical Computing, 13 (1992), pp. 631–644.

[77] S. P. Vanka, *Block-implicit calculation of steady turbulent recirculating flows*, International J. of Heat and Mass Transfer, 28 (1985), pp. 2093–2103.

[78] V. Venkatakrishnan, *Newton solution of inviscid and viscous problems*, AIAA J., 27 (1989), pp. 885–891.

[79] V. Venkatakrishnan, *Parallel implicit unstructured grid Euler solvers*, AIAA J., 32 (1994), pp. 1985–1991.

[80] V. Venkatakrishnan, *Convergence to steady state solutions of the Euler equations on unstructured grids with limiters*, J. Computational Physics, 118 (1995), pp. 120–130.

[81] V. Venkatakrishnan and D. J. Mavriplis, *Implicit solvers for unstructured meshes*, J. Computational Physics, 105 (1993), pp. 83–91.

[82] G. Wang and D. K. Tafti, *Performance enhancement on microprocessors with hierarchical*

memory systems for solving large sparse linear systems, International J. for Supercomputer Applications and High Performance Computing, 12 (1998 (to appear)).

[83] D. L. Whitfield and J. M. Janus, *Three-dimensional unsteady Euler equations using flux vector splitting*, in Proceedings of the AIAA 17th Fluid Dynamics, Plasma Dynamics, and Lasers Conference, 1984.

[84] D. L. Whitfield and L. K. Taylor, *Discretized Newton-relaxation solution of high resolution flux-difference split schemes*, in Proceedings of the AIAA Tenth Annual Computational Fluid Dynamics Conference, 1991, pp. 134–145.

[85] ——, *Variants of a two-level method for the approximate numerical solution of field simulation equations*, Tech. Report 98-09, Mississippi State University Engineering Research Center, July 1998.

[86] L. B. Wigton, N. J. Yu, and D. P. Young, *GMRES acceleration of computational fluid dynamics codes*, Tech. Report 85-1494, AIAA, 1985.

[87] D. P. Young, R. G. Melvin, M. B. Bieterman, F. T. Johnson, S. S. Samant, and J. E. Bussoletti, *A locally regined rectangular grid finite element method: Applications to computational fluid dynamics and computational physics*, J. Computational Physics, 92 (1991), pp. 1–66.