Newton-Krylov-Schwarz Methods for Aerodynamics Problems: Compressible and Incompressible Flows on Unstructured Grids

D. K. Kaushik* D. E. Keyes† B. F. Smith‡

1 Introduction and Motivation

We review and extend to the compressible regime an earlier parallelization of an implicit incompressible unstructured Euler code [9], and solve for flow over an M6 wing in subsonic, transonic, and supersonic regimes. While the parallelization philosophy of the compressible case is identical to the incompressible, we focus here on the nonlinear and linear convergence rates, which vary in different physical regimes, and on comparing the performance of currently important computational platforms.

Multiple-scale problems should be marched out at desired accuracy limits, and not held hostage to often more stringent explicit stability limits. In the context of inviscid aerodynamics, this means evolving transient computations on the scale of the convective transit time, rather than the acoustic transit time, or solving steady-state problems with local CFL numbers approaching infinity. Whether time-accurate or steady, we employ Newton's method on each (pseudo-)timestep. The coupling of analysis with design in aerodynamic practice is another motivation for implicitness. Design processes that make use of sensitivity derivatives and the Hessian matrix require operations with the Jacobian matrix of the state constraints (i.e., of the governing PDE system); if the Jacobian is available for design, it may be employed with advantage in a nonlinearly implicit analysis, as well.

*Computer Science Department, Old Dominion University, Norfolk, VA 23529-0162 and ICASE, NASA Langley Res. Ctr., Hampton, VA 23681-2199, kaushik@cs.odu.edu. Supported in part by NASA under contract NAGI-1692 and by a GAANN fellowship from the U. S. Department of Education.

†Computer Science Department, Old Dominion University, Norfolk, VA 23529-0162 and ICASE, NASA Langley Res. Ctr., Hampton, VA 23681-2199, keyes@icase.edu. Supported in part by the National Science Foundation under grant ECS-9527169, by NASA under contracts NAS1-19480 and NAS1-97046, and by Argonne National Laboratory under contract 982232402.

‡Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439-4844, bsmith@mcs.anl.gov. Supported by U.S. Department of Energy, under Contract W-31-109-Eng-38.

Implicit methods tend to contain global operations, which makes them challenging to parallelize. Nevertheless, the increasing resolution requirements of PDE analyses require access to the large memories provided by parallelism.

2 Parallel ΨNKS Solvers and Software

Our framework for an implicit PDE solution algorithm, with pseudo-timestepping to advance towards an assumed steady state, has the form: $(\frac{1}{\Delta t^\ell})\mathbf{u}^\ell + \mathbf{f}(\mathbf{u}^\ell) = (\frac{1}{\Delta t^\ell})\mathbf{u}^{\ell-1}$, where $\Delta t^\ell \rightarrow \infty$ as $\ell \rightarrow \infty$, where $\mathbf{u}$ represents the fully coupled vector of unknowns, and the steady-state solution satisfies $\mathbf{f}(\mathbf{u}) = 0$.

Each member of the sequence of nonlinear problems, $\ell = 1, 2, \ldots$, is solved with an inexact Newton method. The resulting Jacobian systems for the Newton corrections are solved with a Krylov method, relying directly only on matrix-free operations. The Krylov method needs to be preconditioned for acceptable inner iteration convergence rates, and the preconditioning can be the "make-or-break" feature of an implicit code. A good preconditioner saves time and space by permitting fewer iterations in the Krylov loop and smaller storage for the Krylov subspace. An additive Schwarz preconditioner [4] accomplishes this in a concurrent, localized manner, with an approximate solve in each subdomain of a partitioning of the global PDE domain. Applying any preconditioner in an additive Schwarz manner tends to increase flop rates over the same preconditioner applied globally, since the smaller subdomain blocks maintain better cache residency, even apart from concurrency considerations. Combining a Schwarz preconditioner with a Krylov iteration method inside an inexact Newton method leads to a synergistic parallelizable nonlinear boundary value problem solver with a classical name: Newton-Krylov-Schwarz (NKS) [5, 7]. Combined with pseudo-timestepping, we write ΨNKS

The basic philosophy of any efficient distributed computation is "owner computes," together with message merging and overlapping communication with computation where possible with split transactions. To minimize communication, each processor "ghosts" its stencil dependences on its neighbors' data. Grid functions are mapped from a global (user) ordering into contiguous local orderings (which, in unstructured cases are designed to maximize spatial locality for cache line reuse). Scatter/gather operations are created between local sequential vectors and global distributed vectors, based on connectivity patterns determined at runtime. Global NKS operations are thus translated into local tasks and communication tasks.

We employ the PETSc package [3], which features distributed data structures — index sets, vectors, and matrices — as fundamental objects. Iterative linear and nonlinear solvers, implemented in as data structure-neutral a manner as possible, are combinable modularly, recursively, and extensibly through a uniform application programmer interface. Portability is achieved through MPI, but message-passing detail is not required in user code. We use MeTiS [8] to partition the unstructured grid.

3 Incompressible and Compressible Flows

Our discretization routines are adapted from FUN3D, a tetrahedral unstructured grid code developed by W. K. Anderson and co-workers at NASA Langley for compressible [1] and incompressible [2] Euler and Navier-Stokes equations. It is used in aeronautical and automotive external flow applications for analysis and (recently) design optimization. Unknown fields are defined at the vertices of tetrahedra, with a typical edge-connectivity to other vertices of approximately 15, when a control volume discretization is carried out on the polyhedral control volumes that are dual to the tetrahedra. The discrete $\mathbf{f}(\mathbf{u})$ is constructed in a conservative manner by looping over the edges of the tetrahedral grid, evaluating fluxes across the dual cell face pierced by the edge, and allocating the identical flux with opposite sign to the two control volumes on either side.

The incompressible version of the code employs four unknowns (pressure and three momenta) at each vertex and Chorin's artificial compressibility technique [6]. The compressible version uses five unknowns (density, momenta, and internal energy) at each vertex. Roe's flux-difference splitting is used to discretize the convective terms. This requires a local eigendecomposition of the flux Jacobian to determine the characteristic directions and speeds of the waves passing through each control volume face element — an operation that is significantly more complex, computationally, for the compressible case than for the incompressible. Either first- or second-order fluxes can be computed, a feature that we exploit in different ways in the pseudo-transient NKS technique. We never employ higher than first-order fluxes in the preconditioner, since the preconditioner matrix is incompletely factored without pivoting, which is stabilized by the artificial viscosity of the upwinded first-order discretization. We always employ second-order fluxes in the residual evaluation as steady state is approached, and thereby obtain close to second-order discretization accuracy in the converged solution. We switch from first-order to second-order fluxes in the residual evaluation as a "continuation" device in compressible flow problems, in which rapid Newton convergence is difficult to achieve from a second-order discretization alone.

Besides the discretization-order switch, we employ the traditional continuation device of false timestepping when pursuing steady states, as in this paper. (False timestepping may also be employed as a subiteration, if necessary, in transient problems.) The timestep is advanced towards infinity by a power-law variation of the switched evolution/relaxation (SER) heuristic of Van Leer & Mulder [11]. To be specific, within each of the first-order and second-order phases of computation, we adjust the timestep according to

$$N_{CFL}^{\ell} = N_{CFL}^{0} \left(\frac{\|f(u^0)\|}{\|f(u^{\ell-1})\|} \right)^p ,$$

where p is normally unity, but damped down to 0.75 for robustness in cases in which shocks are expected to appear.

The overall solution process for nonlinear steady states has been found to be competitive with FAS multigrid in execution time when compared in specific

two-dimensional external Euler flow contexts on vector computers [10]. We have not yet compared ΨNKS to FAS multigrid in the parallel three-dimensional context, but we know that, at a minimum, nonlinear grid sequencing is required for pseudo-transient NKS to scale acceptably as the mesh is refined.

4 Comparisons

In this paper, we present three sets of comparisons: (1) the parallel scalability of a transonic compressible flow problem on three different parallel machines (Cray T3E, SGI Origin, IBM SP), (2) the parallel scalability of four different flow problems on an Origin, and (3) the cache efficiency of two different flow problems on five common sequential processors. The first set of experiments exposes relative performance advantages of important machines. The second set exposes the relative difficulties of solving different regimes of flow. We consider incompressible, compressible subsonic, transonic, and mildly supersonic flows. The grid and geometry are held fixed. (This is somewhat unrealistic, since no special care is given to adaptively resolve shocks when they appear, but we are primarily interested in the algebraic aspects of the problem, not the discretization aspects.) The third set of experiments illustrates the sensitivity of per-processor floating-point performance to cache organization, and the organization of the "busy" data structures in the flow code.

Incompressible (Mach zero) flow is relevant to low velocity portions of flight envelope (and to automotive speeds), where the incompressible formulation offers considerable savings in storage and execution time over the compressible formulation while capturing the physics accurately for Mach numbers up to approximately 0.3. Our transonic (Mach 0.839) flow is a classical $\alpha = 3.06^\circ$ "lambda shock" test case. This is relevant to airliner cruise conditions. Runs at Mach 0.3 verify the adequacy of the incompressible model and help interpolate trends. Runs at Mach 1.2 explore changing nonlinear and linear character of the discretized system.

4.1 Scalability Across Platforms

Cross-platform performance comparisons of a medium-size wing problem, closed with a symmetry plane inboard, are given in Table 1. The 16-processor run has approximately 22,369 vertices per processor; the 80-processor run has approximately 4,473. Decreasing volume-to-surface ratios in the subdomains and increasing depth of the global reduction spanning tree of the processors lead to gradually decaying efficiency. The convergence rate, in terms of pseudo-time steps to achieve a relative reduction of steady-state residual norm of 10^{-12}, is not much affected by increased partitioning. Exactly one Newton iteration is performed on each pseudo-time step, and the Krylov space restart size is 30, with a maximum of one restart. The slight differences in the numbers of timesteps arise from slightly different floating point arithmetic and/or noncommutative summation of global inner products, which lead to slightly different trajectories

Table 1: Transonic flow over M6 wing; fixed-size grid of 357,900 vertices.

No.	Cray T3E			IBM SP			SGI Origin		
Procs.	Steps	Time	Eff.	Steps	Time	Eff.	Steps	Time	Eff.
16	55	2406s	—	55	1920s	—	55	1616s	—
32	57	1331s	.90	57	1100s	.87	56	862s	.94
48	57	912s	.88	57	771s	.83	56	618s	.87
64	57	700s	.86	56	587s	.82	57	493s	.82
80	57	577s	.83	59	548s	.70	57	420s	.77

to the same steady state. The Origin is the fastest per processor (achieving the highest percentage of peak sequentially). The T3E has the best scalability, due to its torus network, which is fast compared to sequential processor performance. The full problem fits on smaller numbers of processors on the Origin, but "false" superunitary parallel scalability results due to cache-thrashing when too many vertices are assigned to a processor; 5K to 20K vertices per processor is reasonable for this code.

4.2 Scalability Across Flow Regimes

Trans-Mach convergence comparisons of the same problem are given in Table 2. Here efficiencies are normalized by the number of timesteps, to factor convergence degradation out of the performance picture and measure implementation factors alone (though convergence degradation with increasing granularity is modest). The number of steps increases dramatically with the nonlinearity of the flow, as Mach rises; however, the linear work per step decreases on average. Reasons for this include: more steps spent in the cheaper, first-order discretization phase of the continuation process, smaller CFL in early steps, and the increased hyperbolicity of the flow. The compressible Jacobian is far more complex to evaluate, but it also concentrates locality, achieving much higher computational rates than the corresponding incompressible Jacobian.

4.3 Memory Hierarchy Aspects

As observed in [9] for the same unstructured flow code, data structure storage patterns for primary and auxiliary fields should adapt to hierarchical memory through: (1) interlacing, (2) blocking of degrees of freedom (DOFs) that are defined at the same point in point-block operations, and reordering (3) of edges for reuse of vertex data. Blocking allows efficient use of registers by reducing integer overhead and permitting hardwired unrolling of dense inner loops. Interlacing allows efficient reuse of cached operands, since components at the same point interact more intensely with each other than do the same fields at other points. Similarly, edge-reordering for vertex reuse reflects the fact that nearby points interact more intensely than distant points.

Table 2: Flow over M6 wing on SGI Origin; fixed-size grid of 357,900 vertices (1,431,600 DOFs incompressible, 1,789,500 DOFs compressible).

No. Procs.	Steps	Time per Step	Per-Step Speedup	Impl. Eff.	FcnEval Mflop/s	JacEval Mflop/s
Incompressible (4×4 blocks)						
16	19	41.6s	—	—	2,630	359
32	19	20.3s	2.05	1.02	5,366	736
48	21	14.1s	2.95	0.98	7,938	1,080
64	21	11.2s	3.71	0.93	10,545	1,398
80	21	10.1s	4.13	0.83	11,661	1,592
Subsonic (Mach 0.30) (5×5 blocks)						
16	17	55.4s	—	—	2,002	2,698
32	19	29.8s	1.86	0.93	3,921	5,214
48	19	20.5s	2.71	0.90	5,879	7,770
64	20	14.3s	3.88	0.97	8,180	10,743
80	20	12.7s	4.36	0.87	9,452	12,485
Transonic (Mach 0.84) (5×5 blocks)						
16	55	29.4s	—	—	2,009	2,736
32	56	15.4s	1.91	0.95	4,145	5,437
48	56	11.0s	2.66	0.89	5,942	7,961
64	57	8.7s	3.39	0.85	8,103	10,531
80	57	7.4s	3.99	0.80	9,856	12,774
Supersonic (Mach 1.20) (5×5 blocks)						
16	80	19.2s	—	—	2,025	2,679
32	81	10.6s	1.81	0.90	3,906	5,275
48	81	7.1s	2.72	0.91	6,140	7,961
64	82	5.8s	3.31	0.83	7,957	10,398
80	80	4.6s	4.20	0.84	9,940	12,889

Table 3 illustrates these three effects on five processors with different cache (and processor) parameters. The original ordering is the native FUN3D ordering, which is based on vector register-oriented multicoloring. The combination of the three effects can enhance overall execution time by a factor of 2.5 on the Pentium to as much as 7.5 on the Power2. We are currently studying hardware counter profiles of similar runs to build more detailed causal explanations.

5 Conclusions and Future Directions

Unstructured implicit CFD solvers are amenable to scalable implementation, but careful tuning is needed to obtain the best product of per-processor efficiency and parallel efficiency. We [9] and others have already solved problems of millions of vertices on hundreds of processors at rates in the tens of gigaflop/s,

and we believe such performance is extensible, with further effort, to the teraflop/s regime. In the future, we hope to enhance per-processor performance through improved spatial and temporal locality. We also hope to enhance parallel efficiency through algorithms that synchronize less frequently, and through multiobjective partitioning, which equidistributes communication work as well as computational work.

Acknowledgements

The authors thank W. Kyle Anderson of the NASA Langley Research Center for providing FUN3D. Satish Balay, Bill Gropp, and Lois McInnes of Argonne National Laboratory co-developed (with Smith) the PETSc software employed in this paper. Computer time was supplied by DOE (through Argonne and NERSC). Sandra Bittner graciously assisted in providing dedicated Origin access.

References

[1] W. K. Anderson and D. L. Bonhaus. An implicit upwind algorithm for computing turbulent flows on unstructured grids. *Computers and Fluids*, 23, 1994.

[2] W. K. Anderson, R. D. Rausch, and D. L. Bonhaus. Implicit/multigrid algorithms for incompressible turbulent flows on unstructured grids. AIAA 95-1740, 1995.

[3] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. The Portable, Extensible Toolkit for Scientific Computing, version 2.0.22. `http://www.mcs.anl.gov/petsc`, 1998.

[4] X.-C. Cai. Some domain decomposition algorithms for nonselfadjoint elliptic and parabolic partial differential equations. Courant Institute TR 461, 1989.

[5] 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*, pages 387–400. Wiley, 1997.

[6] A. Chorin. A numerical method for solving incompressible viscous flow problems. *J. Comp. Phys.*, 2:12–26, 1967.

[7] W. D. Gropp, L. C. McInnes, M. D. Tidriri, and D. E. Keyes. Parallel implicit PDE computations: Algorithms and software. In *Proceedings of Parallel CFD'97*, pages 333–344. Elsevier, 1998.

[8] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. to appear in SIAM J. Sci. Comput., 1998.

[9] D. K. 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*, pages 311–319. AMS, 1998.

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

[11] W. Mulder and B. Van Leer. Experiments with implicit upwind methods for the Euler equations. *J. Comp. Phys.*, 59:232–246, 1995.

Table 3: Flow over M6 wing; fixed-size grid of 22,677 vertices (90,708 DOFs incompressible; 113,385 DOFs compressible). Activation of a layout enhancement is indicated by a "×" in the corresponding column. Improvement ratios are averages over the entire code; different subroutines benefit to different degrees. The F77 and C compilers are the vendor's in each case, except for the Pentium, where versions 5.0 of Visual Fortran and C++ are used.

Enhancements			Results			
Field	Structural	Edge	Incompressible		Compressible	
Interlacing	Blocking	Reordering	Time/Step	Ratio	Time/Step	Ratio
Alpha 21164, 450MHz, cache: 8KB data / 8KB instr/ 96KB L2						
			153.2s	—	216.8s	—
×			67.8s	2.26	93.8s	2.31
×	×		56.0s	2.74	72.1s	3.00
		×	55.7s	2.75	91.0s	2.38
×		×	38.9s	3.94	58.3s	3.72
×	×	×	29.2s	5.24	40.8s	5.31
IBM P2SC ("thin"), 120MHz, cache: 128KB data / 32 KB instr						
			165.7s	—	237.6s	—
×			62.1s	2.67	85.8s	2.77
×	×		50.0s	3.31	65.7s	3.62
		×	43.3s	3.82	67.5s	3.52
×		×	33.5s	4.95	50.8s	4.68
×	×	×	22.1s	7.51	32.2s	7.37
MIPS R10000, 250MHz, cache: 32KB data / 32KB instr / 4MB L2						
			83.6s	—	140.0s	—
×			36.1s	2.31	57.5s	2.44
×	×		29.0s	2.88	43.1s	3.25
		×	29.2s	2.86	59.1s	2.37
×		×	23.4s	3.57	35.7s	3.92
×	×	×	16.9s	4.96	24.5s	5.71
Intel Pentium II (NT), 400MHz, cache: 16KB data / 16KB instr / 512KB L2						
			70.3s	—	108.5s	—
×			44.1s	1.59	70.1s	1.55
×	×		37.4s	1.88	57.3s	1.89
		×	43.8s	1.61	72.4s	1.50
×		×	34.0s	2.07	54.5s	1.99
×	×	×	27.6s	2.55	43.2s	2.51
Sun UltraSPARC II, 300MHz, cache: 2MB external						
			120.5s	—	185.0s	—
×			61.6s	1.96	86.3s	2.14
×	×		50.8s	2.37	70.9s	2.61
		×	51.0s	2.36	103.1s	1.79
×		×	37.8s	3.19	55.7s	3.32
×	×	×	28.5s	4.22	42.1s	4.39