ACHIEVING HIGH SUSTAINED PERFORMANCE IN AN UNSTRUCTURED MESH CFD APPLICATION

W.K. ANDERSON[*], W.D. GROPP[†], D.K. KAUSHIK[‡], D.E. KEYES[§], AND B.F. SMITH[¶]

Abstract. This paper highlights a three-year project by an interdisciplinary team on a legacy F77 computational fluid dynamics code, with the aim of demonstrating that implicit unstructured grid simulations can execute at rates not far from those of explicit structured grid codes, provided attention is paid to data motion complexity and the reuse of data positioned at the levels of the memory hierarchy closest to the processor, in addition to traditional operation count complexity. The demonstration code is from NASA and the enabling parallel hardware and (freely available) software toolkit are from DOE, but the resulting methodology should be broadly applicable, and the hardware limitations exposed should allow programmers and vendors of parallel platforms to focus with greater encouragement on sparse codes with indirect addressing. This snapshot of ongoing work shows a performance of 15 microseconds per degree of freedom to steady-state convergence of Euler flow on a mesh with 2.8 million vertices using 3072 dual-processor nodes of Sandia's "ASCI Red" Intel machine, corresponding to a sustained floating-point rate of 0.227 Tflop/s.

Key words. high-performance computing, parallel implicit solvers, computational aerodynamics, memory-centric computation

Subject classification. Computer Science

1. Overview. Many applications of economic and national security importance require the solution of nonlinear partial differential equations (PDEs). In many cases, PDEs possess a wide range of time scales—some (e.g., acoustic) faster than the phenomena of prime interest (e.g., convective), suggesting the need for implicit methods. In addition, many applications are geometrically complex and possess a wide range of length scales. Unstructured meshes are often employed in such cases to accomplish mesh generation and adaptation (almost) automatically and to resolve the PDE without requiring an excessive number of mesh points. The best algorithms for solving nonlinear implicit problems are often Newton methods, which themselves require the solution of very large, sparse linear systems. The best algorithms for these sparse linear problems, particularly at very large sizes, are often preconditioned iterative methods. This

[*]Fluid Mechanics and Acoustics Division, NASA Langley Research Center, Hampton, VA 23682, `w.k.anderson@larc.nasa.gov`.

[†]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, `gropp@mcs.anl.gov`. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

[‡]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439 and Computer Science Department, Old Dominion University, Norfolk, VA 23529, `kaushik@cs.odu.edu`. This work was supported by a GAANN Fellowship from the U.S. Department of Education and by Argonne National Laboratory under Contract 983572401.

[§]Mathematics & Statistics Department, Old Dominion University, Norfolk, VA 23529, ISCR, Lawrence Livermore National Laboratory, Livermore, CA 94551, and ICASE, NASA Langley Research Center, Hampton, VA 23681, `keyes@icase.edu`. This work was supported by the National Science Foundation under Grant ECS-9527169, by NASA under Contract Nos. NAS1-97046 and NAS1-19480 (while the author was in residence at ICASE), by Argonne National Laboratory under Contract 982232402, and by Lawrence Livermore National Laboratory under Subcontract B347882.

[¶]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, `bsmith@mcs.anl.gov`. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

nested hierarchy of tunable algorithms has proved effective in solving complex problems in areas such as aerodynamics, combustion, radiation transport, and global circulation. Typically, for steady-state solutions from a trivial initial guess, the number of "work units" (evaluations of the discrete residuals on the finest mesh on which the problem is represented) is around 10^3 (to achieve reductions in the norm of the residual of 10^{-8} to 10^{-12}). Although these algorithms are efficient (in the sense of using relatively few floating-point operations to arrive at the final result), they do not necessarily achieve the absolute flops-per-second (flop/s) ratings that less efficient or less versatile algorithms may [3].

Our submission focuses on the time to solution rather than the achieved floating-point performance as the figure of merit. We have achieved a performance of 15 microseconds per degree of freedom on a mesh with 2.8 million vertices using 3072 dual-processor nodes of ASCI Red, and 36 microseconds per degree of freedom on 1024 processors of an SGI/Cray T3E. These figures correspond to sustained floating-point rates of 227 Gflop/s and 76 Gflop/s, respectively. The code is also nearly scalable, showing linear scaling in computation rate between 128 and 3072 nodes for a fixed-size problem, and only a modest degradation in algebraic convergence rate over the same range.

The code spends almost all of its time in two phases: flux computations (to evaluate conservation law residuals) and sparse linear algebraic kernels. The linear algebraic kernels run at close to the aggregate memory-bandwidth limit on performance (as determined by the STREAM benchmarks [15]), and the flux computations are bounded either by memory bandwidth or instruction scheduling (see the analysis in [8]). This level of performance (in excess of 100 Gflop/s) is well above what is commonly considered achievable for sparse-matrix and unstructured mesh computations and requires a combination of scalable algorithms and data structure optimizations, as well as powerful, tightly networked computers. See, for example, the comments by the "High End Crusader" [6, 7], who has called for a benchmark to focus attention on the difficulty of sparse, unstructured problems.

As a bonus, our message-passing code relies on no special architectural features or proprietary compiler licenses, but is based on the MPI standard, allowing the application to take advantage of continuing improvements in hardware performance without further software development.

2. The Application. The application code, FUN3D, is a tetrahedral vertex-centered unstructured mesh code developed by W. K. Anderson of the NASA Langley Research Center for compressible and incompressible Euler and Navier-Stokes equations [1, 2]. FUN3D uses a control volume discretization with variable-order Roe schemes for approximating the convective fluxes and a Galerkin discretization for the viscous terms. FUN3D is being used for design optimization of airplanes, automobiles, and submarines, with irregular meshes comprising several million mesh points. The optimization loop involves many analysis cycles. Thus, reaching the steady-state solution in each analysis cycle in a reasonable amount of time is crucial to conducting the design optimization. From the beginning, our effort has been focused on minimizing the time to convergence without compromising scalability, by means of appropriate algorithms and architecturally efficient data structures.

We have ported FUN3D into PETSc framework and tuned it for good cache performance and distributed parallel systems, using the single program multiple data (SPMD) programming model. This new variant (PETSc-FUN3D) is being used to run Navier-Stokes applications with the Spalart-Almaras turbulence model [17] on modest-sized problems, and we expect to scale up these more phenomenologically complex problems in coming months, while also beginning to cope with parallelization of the preprocessing.

Thus far, our large-scale parallel experience with PETSc-FUN3D is with the compressible or incompressible Euler subset, but nothing in the solution algorithms or software changes with additional physical phenomenology. Of course, the convergence rate will vary with conditioning, as determined by Mach and Reynolds numbers and the correspondingly induced mesh adaptivity. Furthermore, robustness becomes more of an issue in problems admitting shocks or using turbulence models. The lack of nonlinear robustness is a fact of life that is largely outside of the domain of parallel scalability. In fact, when nonlinear robustness is restored in the usual manner, through pseudo-transient continuation, the conditioning of the linear inner iterations is enhanced, and parallel scalability may be improved. In some sense, the Euler code, with its smaller number of flops per point per iteration, and its aggressive pseudotransient buildup toward the steady-state limit, may be a *more*, not less, severe test of parallel performance.

3. Algorithms and Data Structures.

Achieving high sustained performance, in terms of solutions per second, involves three aspects. The first is a scalable algorithm in the sense of convergence rate. The second is good per-processor performance on contemporary cache-based microprocessors. The third is a scalable implementation, in the sense of time per iteration as the number of processors increases. Our nonlinear method, pseudo-transient Newton-Krylov-Schwarz (ΨNKS), is an efficient algorithm, as the chart of nonlinear iterations in Figure 5.1 shows. The per-processor performance is also quite good; in fact, it is close to the memory-bandwidth limit (a more realistic measure of achievable performance than peak floating-point for sparse problems [8]). Moreover, on any architecture with a sufficiently rich interconnection network, ΨNKS leads to good per-iteration scalability, as argued from a simple analytical model in [14].

3.1. ΨNKS Solver.

Our implicit algorithmic framework for advancing toward an assumed steady state, $\mathbf{f}(\mathbf{u}) = 0$, 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 $\Delta t^\ell \to \infty$ as $\ell \to \infty$, $\mathbf{u}$ represents the fully coupled vector of unknowns, and $\mathbf{f}(\mathbf{u})$ is the vector of nonlinear conservation laws.

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 [5] accomplishes this in a concurrent, localized manner, with an approximate solve in each subdomain of a partitioning of the global PDE domain. The coefficients for the preconditioning operator are derived from a lower-order, sparser, and more diffusive discretization than that used for $\mathbf{f}(\mathbf{u})$, itself. 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 [18]. 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) [9]. We combine NKS with pseudo-timestepping [13] and use the shorthand ΨNKS to describe the algorithm.

To implement this algorithm in FUN3D, we employ the PETSc package [4], which features distributed data structures—index sets, vectors, and matrices—as fundamental objects. Iterative linear and nonlinear solvers are implemented within PETSc in a data structure-neutral manner, providing a uniform application programmer interface. Portability is achieved in PETSc through MPI, but message-passing detail is not required in the application. We use MeTiS [10] to partition the unstructured mesh.

3.2. Memory-Centric Computation.

We view a PDE computation predominantly as a mix of loads and stores with embedded floating-point operations (flops). Since flops are cheap relative to memory references, we concentrate on minimizing the memory references and emphasize strong sequential performance as

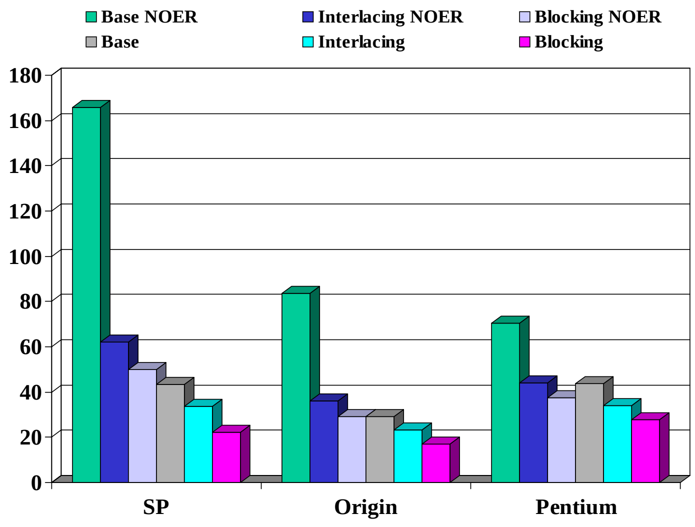

Fig. 3.1. *The effect of cache optimizations on the average execution time for one nonlinear iteration.* **BASE** *denotes the case without any optimizations, and* **NOER** *denotes no edge reordering. The performance improves by a factor of about 2.5 on the Pentium and 7.5 on the IBM SP. The processor details are: 120 MHz IBM SP (P2SC "thin", 128 KB L1), 250 MHz Origin 2000 (R10000, 32 KB L1, and 4 MB L2), 400 MHz Pentium II (running Windows NT 4.0, 16 KB L1, and 512 KB L2).*

one of the factors needed for aggregate performance worthy of the theoretical peak of a parallel machine. We use *interlacing* (creating spatial locality for the data items needed successively in time), *structural blocking* for a multicomponent system of PDEs (cutting the number of integer loads significantly, and enhancing reuse of data items in registers), and *vertex and edge reorderings* (increasing the level of temporal locality). Applying these techniques required whole-program transformations of certain loops of the original vector-oriented FUN3D, but raised the per-processor performance by a factor of between 2.5 and 7.5 (Figure 3.1), depending on the microprocessor and optimizing compiler [12].

The importance of memory bandwidth to the overall performance is suggested by the single-processor performance of PETSc-FUN3D shown in Figure 3.2. The performance of PETSc-FUN3D is compared to the peak performance and the result of the STREAM benchmark [15] which measures achievable performance for memory bandwidth-limited computations. These comparisons show that the STREAM results are much better indicators of realized performance than the peak numbers. The parts of the code that are memory bandwidth-limited (like the sparse triangular matrix solution phase, which is responsible for 25% of the overall execution time) are bound to show poor performance, as compared to dense matrix-matrix operations, which often come within 10–20% of peak. Even parts of the code that are not memory intensive often achieve much less than peak performance because of the limits on the number of basic operations that can be performed in a single clock cycle [8]. This is true for the flux calculation routine in PETSc-FUN3D, which consumes over 50% of the overall execution time. Instruction scheduling limits the performance to 47% of the peak on 250 MHz SGI Origin 2000 even under a perfect memory system (leading to an estimate of 235 Mflops/s), which is close to the value of 209 Mflops/s experimentally measured by the Origin's hardware counters.

The basic philosophy of any efficient parallel computation is "owner computes," with message merging and overlapping communication with computation where possible via split transactions. Each processor "ghosts" its stencil dependencies on its neighbors' data. Grid functions are mapped from a global (user)

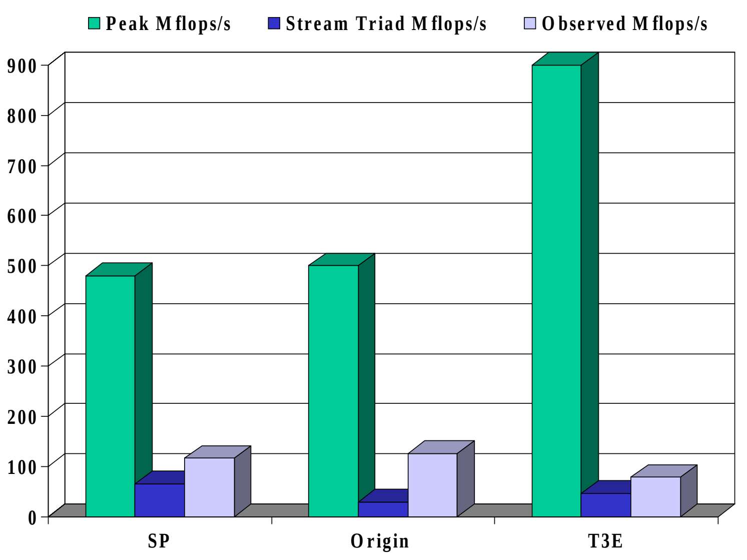

FIG. 3.2. *Sequential performance of PETSc-FUN3D for a coarse mesh of 22,677 vertices (with 4 unknowns per vertex). The processor details for IBM SP and Origin 2000 are the same as in Figure 3.1. The SGI/Cray T3E is based on a 450 MHz DEC Alpha 21164 with 8 KB L1 cache and 96 KB unified L2 cache.*

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 runtime-deduced connectivity patterns.

4. Measuring the Parallel Performance. We use PETSc's profiling and logging features to measure the parallel performance. PETSc logs many different types of events and provides valuable information about time spent, communications, load balance, and so forth, for each logged event. PETSc uses manual counting of flops, which are afterwards aggregated over all the processors for parallel performance statistics. We have observed that the flops reported by PETSc are close to (within 10 percent of) the values statistically measured by hardware counters on R10000 processor.

PETSc uses the best timers available in each processing environment. In our rate computations, we exclude the initialization time devoted to I/O and data partitioning. To suppress timing variations caused by paging in the executable from disk, we preload the code into memory with one nonlinear iteration, then flush, reload the initial iterate, and begin performance measurements.

Since we are solving large fixed-size problems on distributed memory machines, it is not reasonable to base parallel scalability on a uniprocessor run, which would thrash the paging system. Our base processor number is such that the problem has just fit into the local memory. We have employed smaller sequential cases to optimize cached data reuse [11, 12] to minimize the execution time. In the results below, we decompose the parallel efficiency into two factors: *algorithmic efficiency*, measuring the effect of increased granularity on the number of iterations to convergence, and *implementation efficiency*, measuring the effect of increased granularity on per-iteration performance.

5. Scalability Studies. We present three aspects of scalability in this section. Throughout we use unstructured tetrahedral meshes of the standard Onera M6 wing closed with a symmetry plane inboard, prepared for us by colleagues at the NASA Langley Research Center. On the two machines with the finest granularity available to us to date, a Cray T3E with 1024 600 MHz Alpha processors and a partition of

ASCI Red with 3072 333 MHz Pentium Pro dual-processor nodes, we show several metrics of fixed-size scalability on our finest mesh. On machines representative of the two ASCI Blue machines (an IBM SP and an SGI Origin) and on a T3E with 450 MHz processors, we compare executions of the same code on an intermediate fixed-size problem on up to 80 processors (the maximum available on our SP configuration). Finally, to convey some idea of the sensitivity of the Newton method to the severity of the nonlinearity, and of the sensitivity of the preconditioned Krylov solver with respect to different conditioning inherited from different Mach numbers of the simulation, we present some comparisons across Mach number (incompressible to supersonic). This study also gives an indication of the sensitivity of the floating point performance to the blocksize of the unknown vector, which is four in the incompressible case and five in the compressible cases.

5.1. Parallel Scalability on the T3E. The parallel scalability of PETSc-FUN3D is shown in Figure 5.1 for a mesh with 2.8 million vertices running on up to 1024 Cray T3E processors. We see that the implementation efficiency of parallelization (i.e., the efficiency on a per-iteration basis) is 82 percent in going from 128 to 1024 processors. The number of iterations is also fairly flat over the same eightfold range of processor number (rising from 37 to 42), reflecting reasonable algorithmic scalability. This is much less serious degradation than predicted by the linear elliptic theory (see [16]); pseudo-timestepping—required by the nonlinearity—is responsible. The overall efficiency is the product of the implementation efficiency and the algorithmic efficiency. The Mflop/s per processor are also close to flat over this range, even though the relevant working sets in each subdomain vary by nearly a factor of eight. This emphasizes the requirement of good serial performance for good parallel performance.

5.2. Parallel Scalability on ASCI Red. The same fixed-size problem is run on large ASCI Red configurations with sample scaling results shown in Figure 5.2. The implementation efficiency is 94% in going from 256 to 2048 nodes (and 95% in going from 128 to 2048 nodes, due to slightly worse cache performance in the 128-node run). For the data in Figure 5.2, we employed the `-procs 1` runtime option on ASCI Red, which dedicates a communication processor to every execution processor. The `-procs 2` runtime option enables 2-processor-per-node multithreading during threadsafe, communication-free portions of the code. We have activated this feature for the floating-point-intensive flux computation subroutine alone. On 2048 nodes, the resulting Gflop/s rate is 156, or 30% greater than for the single-threaded case on the same number of nodes. On 3072 nodes, the largest run we have been able to make on the unclassified side of the machine to date, the resulting GFlop/s rate is 227. Undoubtedly, further improvements to the algebraic solver portion of the code are also possible through multithreading, but the additional coding work does not seem justified at present.

5.3. Parallel Scalability across Architectures. Cross-platform performance comparisons of a medium-size wing problem over a common set of processor numbers are given in Table 1, which lists overall efficiencies. 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}, degrades only slowly with 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 to the same steady state. The Origin is the fastest per processor (achieving the highest percentage of peak sequentially). The T3E has the

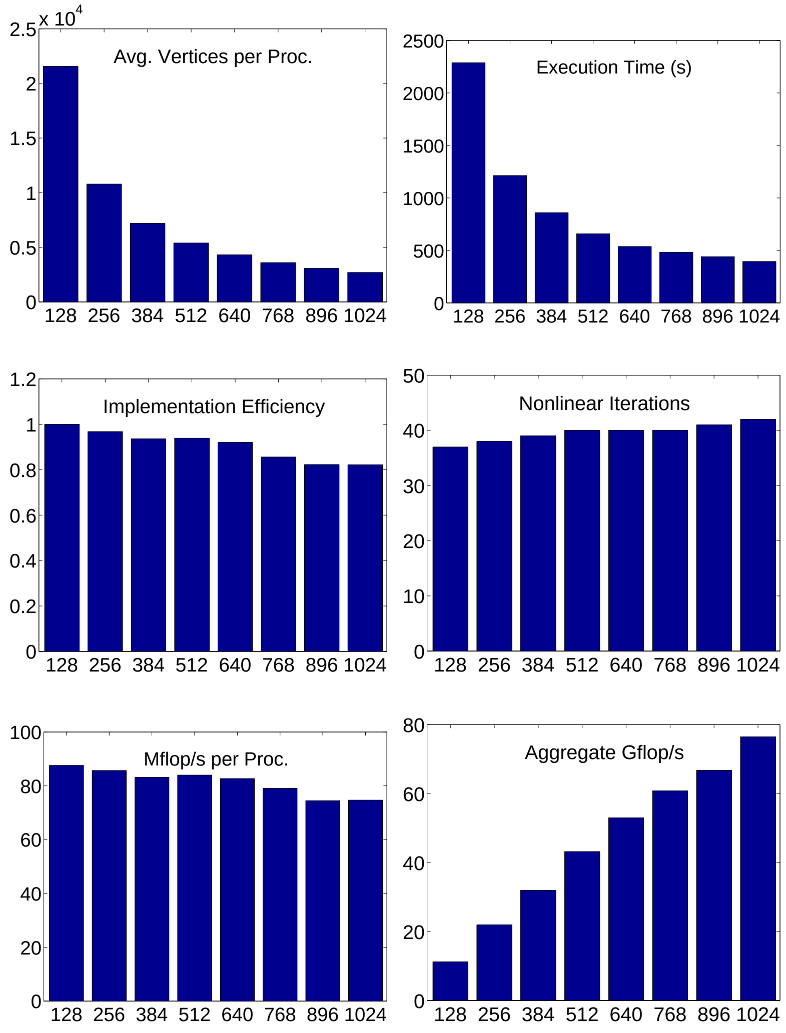

FIG. 5.1. *Parallel performance for a fixed size mesh of 2.8 million vertices run on up to 1024 Cray T3E 600 MHz Alpha processors.*

TABLE 5.1
Transonic flow over M6 wing; fixed-size mesh 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

best scalability, due to its torus network, which is fast compared with sequential processor performance. The full problem fits on smaller numbers of processors on the Origin, but "false" superunitary parallel scalability results because of the cache thrashing when too many vertices are assigned to a processor; 5,000 to 20,000 vertices per processor is a reasonable load for this code.

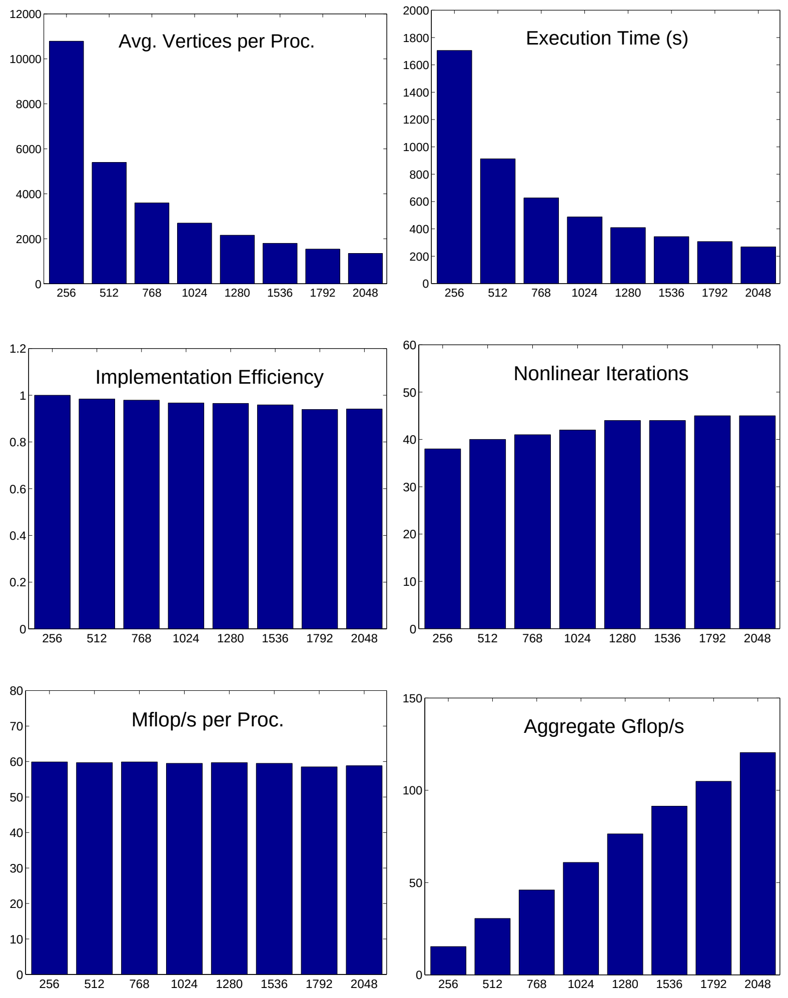

FIG. 5.2. *Parallel performance for a fixed size mesh of 2.8 million vertices run on up to 2048 ASCI Red 333 MHz Pentium Pro processors.*

A plot showing aggregate flop/s performance and a log-log plot showing execution time for our largest case on the three most capable machines to which we have thus far had access are shown in Figures 5.3 and 5.4. In both figures, lines of unit slope (positive and negative, resp.) show the departure from optimality. Note that although the ASCI Red flop/s rate scales nearly linearly, a higher fraction of the work is redundant at higher parallel granularities, so the execution time does not drop in exact proportion to the increase in flop/s.

5.4. Parallel 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 smaller pseudo-timesteps in early nonlinear iterations and the increased hyperbolicity of the flow. The compressible Jacobian is far more complex to evaluate, but its larger blocks (5×5 instead of 4×4) concentrate locality, achieving much higher computational rates than the corresponding

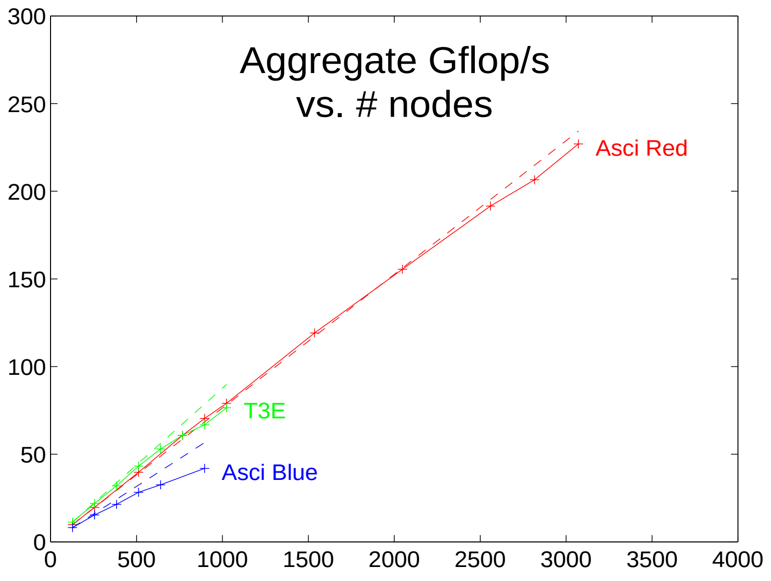

FIG. 5.3. *Fixed-size parallel scaling results: flop/s.*

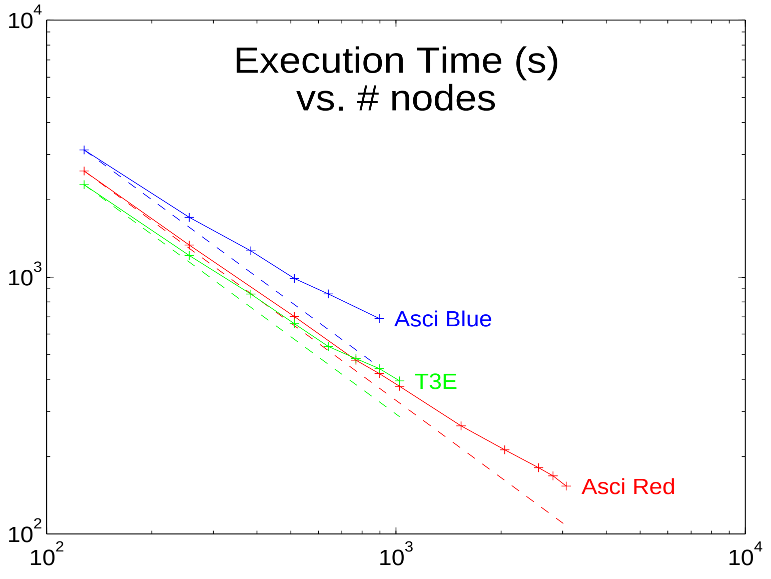

FIG. 5.4. *Fixed-size parallel scaling results: execution time.*

incompressible Jacobian.

6. Conclusion. High sustained scalable performance has been demonstrated on simulations that use implicit algorithms of choice for unstructured PDEs. In the history of the peak performance Bell Prize competition, PDE-based computations have led (or been part of leading entries containing multiple applications) in 1988, 1989, 1990, and 1996. All of these leading entries have been obtained on vector or SIMD architectures, and all were based on *structured* meshes. The last (1996) and most impressive of these PDE-based entries was executed on 160 vector nodes of the Japanese Numerical Wind Tunnel (NWT), and ran at 111 Gflop/s. The 227 Gflop/s sustained performance of our unstructured application on a hierarchical distributed memory multiprocessor in the SPMD programming style exceeds that of the 1996 entry by a

Table 5.2

Flow over M6 wing on SGI Origin; fixed-size mesh 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

factor of two.

The achieved flop/s rate is less important to computational engineers than are solutions per minute of discrete systems that are general enough to be employed in production design, as PETSc-FUN3D is now employed. In addition, PETSc-FUN3D is a portable message-passing application that runs on a variety of platforms with good efficiency, thus lowering the total cost of achieving high performance over the lifetime of the application.

7. Acknowledgments. Computer time was supplied by Argonne National Laboratory, Lawrence Livermore National Laboratory, NERSC, Sandia National Laboratory, and SGI-Cray.

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), pp. 1–21.

[2] W. K. Anderson, R. D. Rausch, and D. L. Bonhaus, *Implicit/multigrid algorithms for incompressible turbulent flows on unstructured grids*, Journal of Computational Physics, 128 (1996), pp. 391–408.

[3] D. F. Bailey, *How to fool the masses when reporting results on parallel computers*, Supercomputing Review, (1991), pp. 54–55.

[4] S. Balay, W. Gropp, L. C. McInnes, and B. Smith, *The Portable, Extensible,Toolkit for Scientific Computing (PETSc) ver. 22.* `http://www.mcs.anl.gov/petsc/petsc.html`, 1998.

[5] X. C. Cai, *Some domain decomposition algorithms for nonselfadjoint elliptic and parabolic partial differential equations*, Technical Report 461, Courant Institute, New York, 1989.

[6] H. E. Crusader, *Peak performance versus bandwidth.* HPCC Week, NOV 1998.

[7] ——, *Towards a U.S. sparse-matrix policy.* HPCC Week, DEC 1998.

[8] W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith, *Toward realistic performance bounds for implicit CFD codes*, in Proceedings of Parallel CFD'99, A. Ecer et al., eds., Elsevier, 1999.

[9] W. D. Gropp, L. C. McInnes, M. D. Tidriri, and D. E. Keyes, *Parallel implicit PDE computations*, in Proceedings of Parallel CFD'97, A. Ecer et al., eds., Elsevier, 1997, pp. 333–344.

[10] G. Karypis and V. Kumar, *A fast and high quality schema for partitioning irregular graphs*, SIAM J. Scientific Computing, 20 (1999), pp. 359–392.

[11] 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 10th International Conference on Domain Decomposition Methods, J. Mandel et al., eds., Wiley, 1997, pp. 311–319.

[12] ——, *Newton-Krylov-Schwarz methods for aerodynamic problems: Compressible and incompressible flows on unstructured grids*, in Proceedings of the 11th International Conference on Domain Decomposition Methods, C.-H. Lai et al., eds., Domain Decomposition Press, Bergen, 1999.

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

[14] D. E. Keyes, *How scalable is domain decomposition in practice?*, in Proceedings of the 11th International Conference on Domain Decomposition Methods, C.-H. Lai et al., eds., Domain Decomposition Press, Bergen, 1999.

[15] J. D. McCalpin, *STREAM: Sustainable memory bandwidth in high performance computers*, tech. report, University of Virginia, 1995. `http://www.cs.virginia.edu/stream`.

[16] B. F. Smith, P. Bjorstad, and W. Gropp, *Domain Decomposition*, Cambridge University Press, 1996.

[17] P. R. Spalart and S. R. Allmaras, *A one-equation turbulence model for aerodynamic flows*, La Recherche Aerospatiale, 1 (1994), pp. 5–21.

[18] G. Wang and D. K. Tafti, *Performance enhancements on microprocessors with hierarchical memory systems for solving large sparse linear systems*, Int. J. High Performance Computing Applications, 13 (1999), pp. 63–79.