

Achieving High Sustained Performance in an Unstructured Mesh CFD Application

<http://www.mcs.anl.gov/petsc-fun3d>

Satish Balay, Argonne National Laboratory

William Gropp, Argonne National Laboratory

Dinesh Kaushik, Argonne National Laboratory & ODU

David Keyes, Old Dominion University, LLNL & ICASE

Barry Smith, Argonne National Laboratory

Features of PETSc-FUN3D

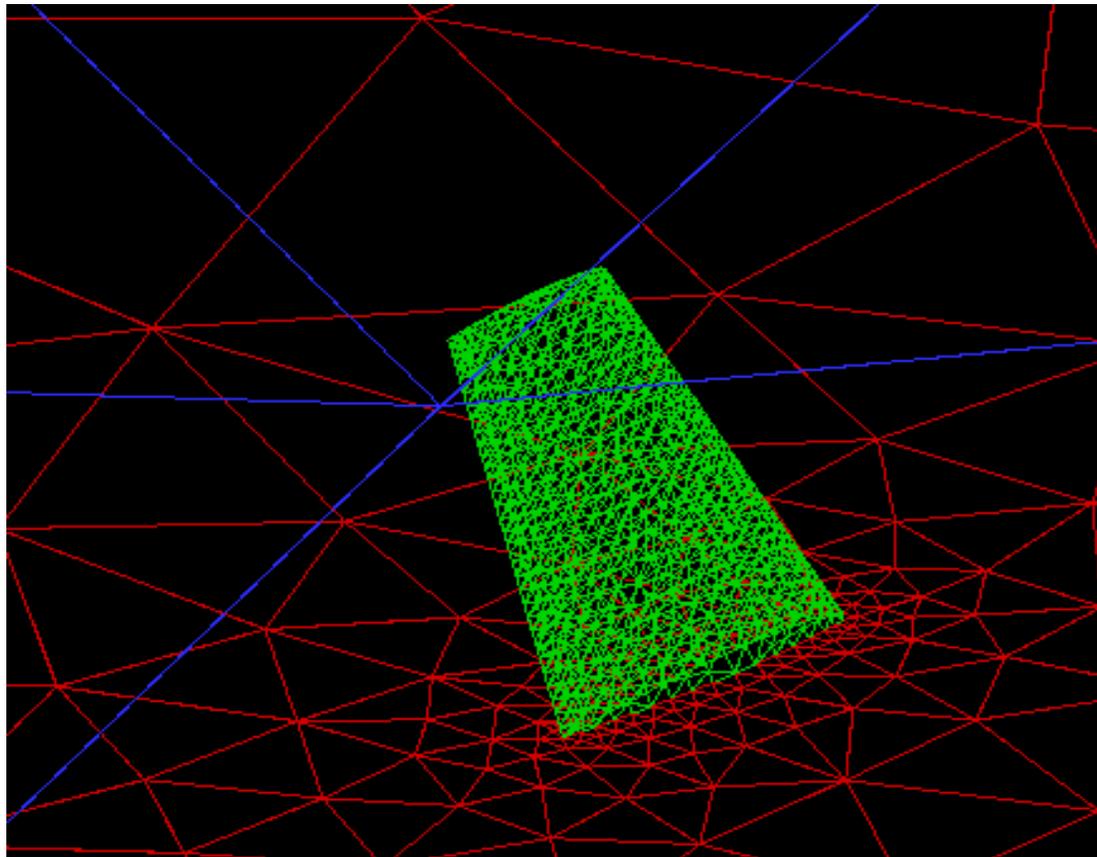
- Based on “legacy” (but contemporary) CFD application with significant F77 code reuse
- Portable, message-passing library-based parallelization, run on NT boxes through Tflop/s ASCI platforms
- Simple multithreaded extension (for SMP Clusters)
- Sparse, unstructured data, implying memory indirection with only modest reuse
- Wide applicability to other implicitly discretized multiple-scale PDE workloads - of interagency, interdisciplinary interest
- Extensive profiling has led to follow-on algorithmic research

Background of FUN3D Application

- Tetrahedral vertex-centered unstructured grid code developed by W. K. Anderson (Formerly at NASA Langley) for steady compressible and incompressible Euler and Navier-Stokes equations (with one-equation turbulence modeling)
- Used in airplane, automobile, and submarine applications for analysis and design
- Standard discretization is 2nd-order Roe for convection and Galerkin for diffusion
- Newton-Krylov solver with global point-block-ILU preconditioning, with false timestepping for nonlinear continuation towards steady state; competitive with FAS multigrid in practice
- Legacy implementation/ordering is vector-oriented

Surface Visualization of Test Domain for Computing Flow over an ONERA M6 Wing

- Wing surface outlined in green triangles
- Nearly 2.8 M vertices in this computational domain



Time-Implicit Newton-Krylov-MG-Schwarz

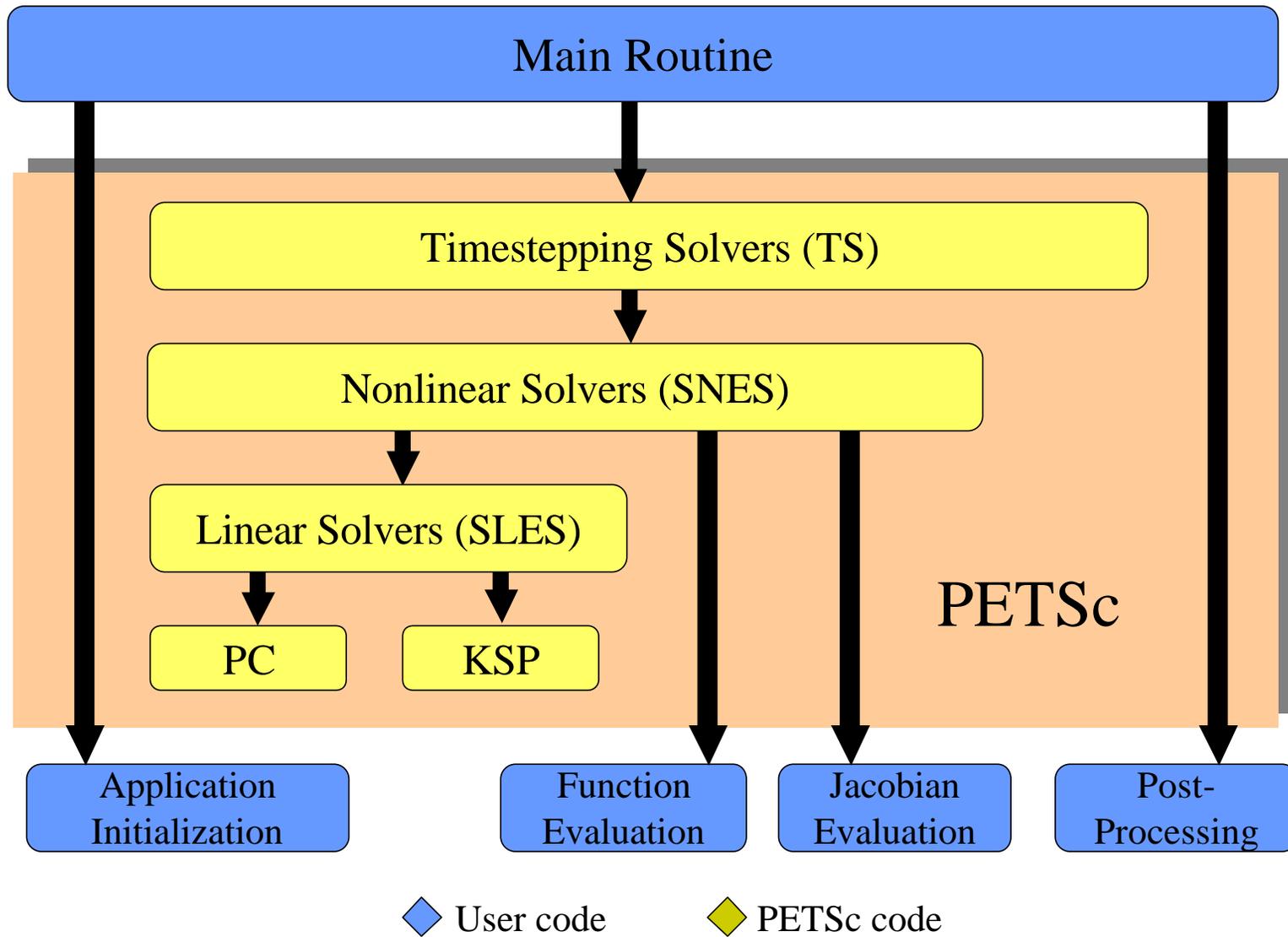
For nonlinear robustness, NKS iteration is wrapped in time-stepping:

```
for (l = 0; l < n_time; l++) {                               # n_time ~ 50
    select time step
    for (k = 0; k < n_Newton; k++) {                         # n_Newton ~ 1
        compute nonlinear residual and Jacobian
        for (j = 0; j < n_Krylov; j++) {                   # n_Krylov ~ 60
            forall (i = 0; i < n_Precon ; i++) {
                solve subdomain problems concurrently
            } // End of loop over subdomains
            perform Jacobian-vector product
            enforce Krylov basis conditions
            update optimal coefficients
            check linear convergence
        } // End of linear solver
        perform DAXPY update
        check nonlinear convergence
    } // End of nonlinear loop
} // End of time-step loop
```

Merits of NKS Algorithm/Implementation

- Relative characteristics: the “exponents” are *naturally good*
 - ⌚ Convergence scalability
 - weak (or no) degradation in problem size and parallel granularity (with use of small global problems in Schwarz preconditioner)
 - ⌚ Implementation scalability
 - no degradation in ratio of surface communication to volume work (in problem-scaled limit)
 - only modest degradation from global operations (for sufficiently richly connected networks)
- Absolute characteristics: the “constants” can be *made good*
 - ⌚ Operation count complexity
 - residual reductions of 10^{-9} in 10^3 “work units”
 - ⌚ Per-processor performance
 - up to 25% of theoretical peak
- Overall, machine-epsilon solutions require as little as 15 microseconds per degree of freedom!

Separation of Concerns: User Code/PETSc Library



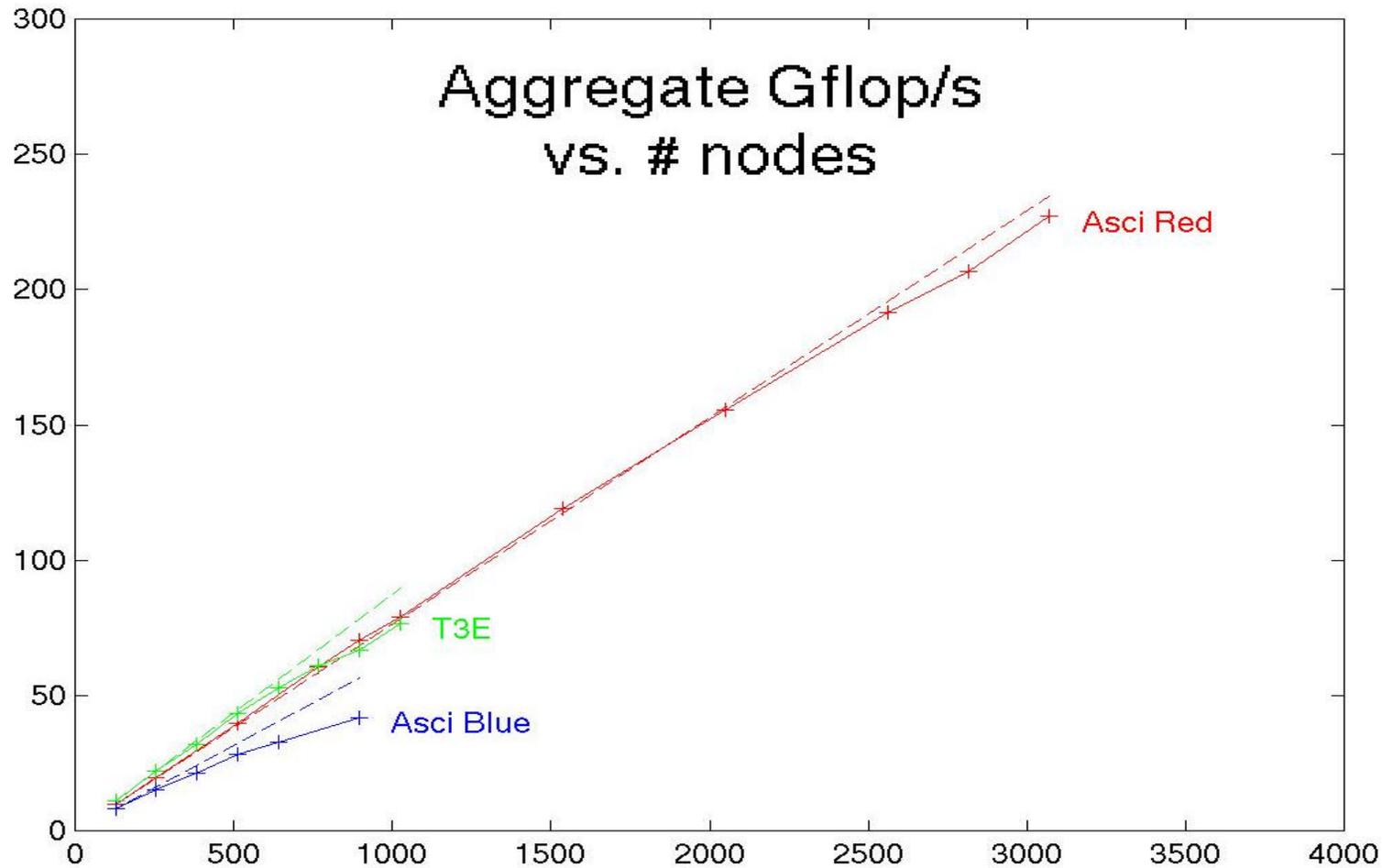
Key Features of Implementation Strategy

- Follow the “owner computes” rule under the dual constraints of minimizing the number of messages and overlapping communication with computation
- Each processor “ghosts” its stencil dependences in its neighbors
- Ghost nodes ordered after contiguous owned nodes
- Domain mapped from (user) global ordering into local orderings
- Scatter/gather operations created between **local sequential** vectors and **global distributed** vectors, based on runtime connectivity patterns
- Newton-Krylov-Schwarz operations translated into local tasks and communication tasks
- Profiling used to help eliminate performance bugs in communication and memory hierarchy

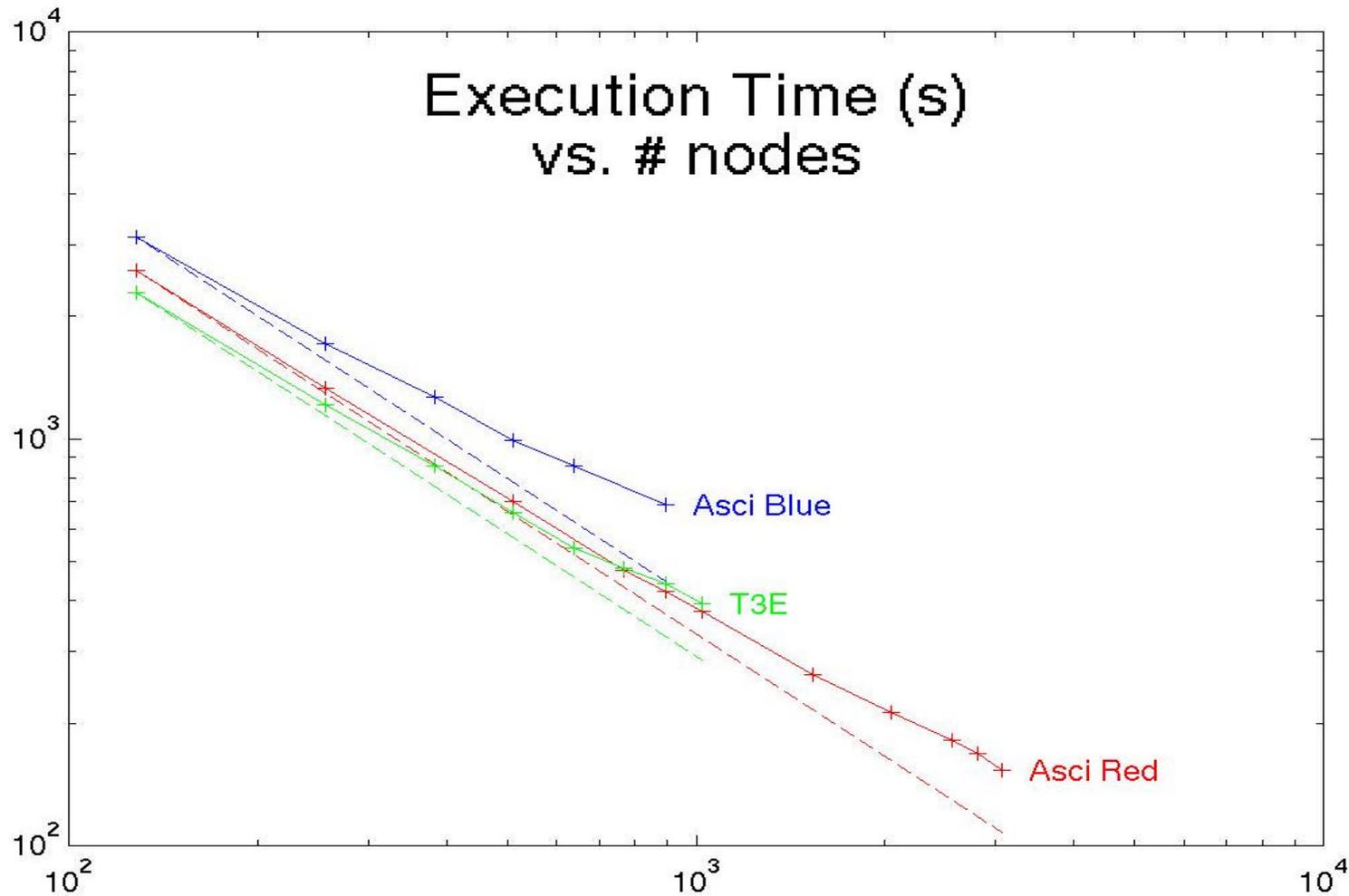
Single-processor Performance of PETSc-FUN3D

Processor	Clock MHz	Peak Mflop/s	Opt. % of Peak	Opt. Mflop/s	Reord. Only Mflop/s	Interl. only Mflop/s	Orig. Mflop/s	Orig. % of Peak
R10000	250	500	25.4	127	74	59	26	5.2
P3	200	800	20.3	163	87	68	32	4.0
P2SC (2 card)	120	480	21.4	101	51	35	13	2.7
P2SC (4 card)	120	480	24.3	117	59	40	15	3.1
604e	332	664	9.9	66	43	31	15	2.3
Alpha 21164	450	900	8.3	75	39	32	14	1.6
Alpha 21164	600	1200	7.6	91	47	37	16	1.3
Ultra II	300	600	12.5	75	42	35	18	3.0
Ultra II	360	720	13.0	94	54	47	25	3.5
Ultra II/HPC	400	800	8.9	71	47	36	20	2.5
Pent. II/LIN	400	400	20.8	83	52	47	33	8.3
Pent. II/NT	400	400	19.5	78	49	49	31	7.8
Pent. Pro	200	200	21.0	42	27	26	16	8.0
Pent. Pro	333	333	18.8	60	40	36	21	6.3

Fixed-size Parallel Scaling Results (Flop/s)

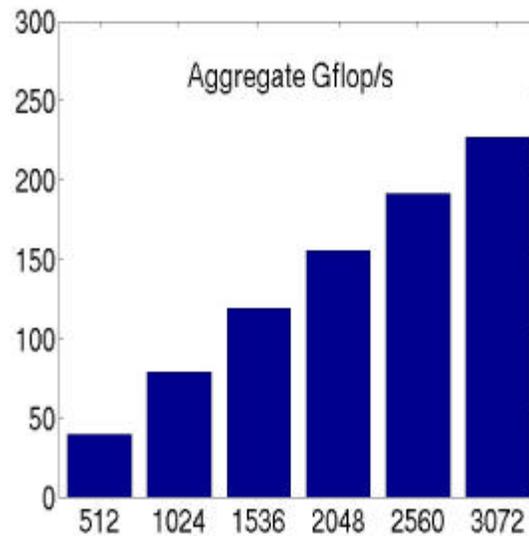
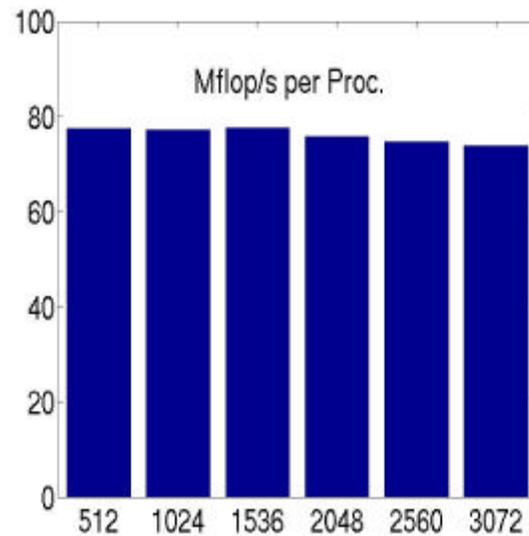
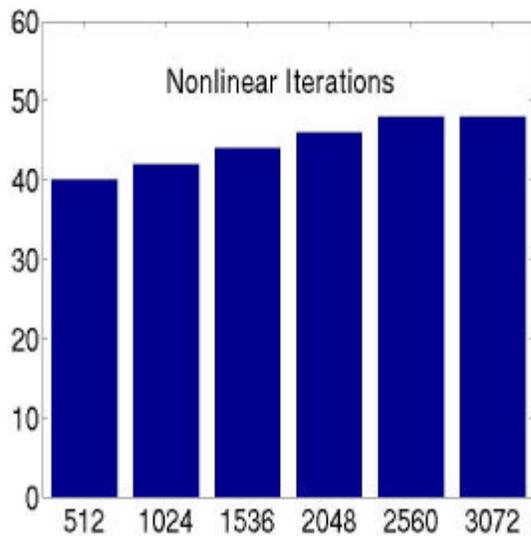
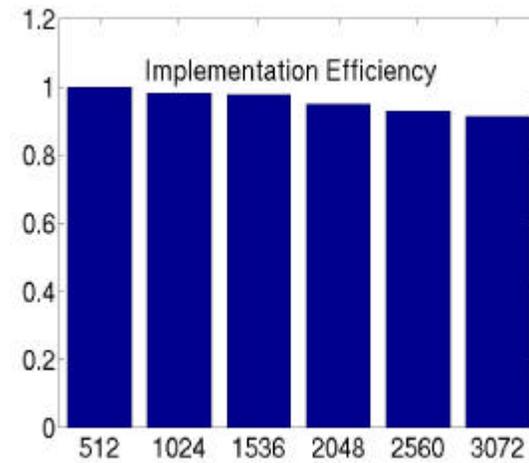
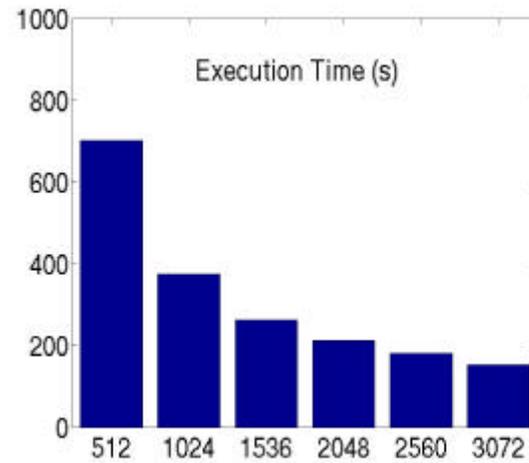
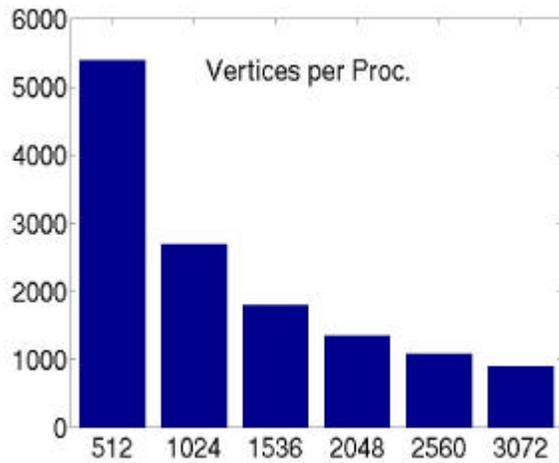


Fixed-size Parallel Scaling Results (Time in seconds)



Inside the Parallel Scaling Results on ASCI Red

ONERA M6 Wing Test Case, Tetrahedral grid of 2.8 million vertices (about 11 million unknowns) on up to 3072 ASCI Red Nodes (each with dual Pentium Pro 333 MHz processors)



MPI/OpenMP in PETSc-FUN3D

- Only in the flux evaluation phase as it is not memory bandwidth bound
- Gives the best execution time as the number of nodes increases because the subdomains are chunkier as compared to pure MPI case

Nodes	MPI/OpenMP		MPI	
	1 Thr	2 Thr	1 Proc	2 Proc
256	483s	261s	456s	258s
2560	76s	39s	72s	45s
3072	66s	33s	62s	40s

Lower Precision Storage:

A Way to Reduce the Required Memory Bandwidth

Execution times on a 250MHz Origin 2000 for 358K-vertex case with single or double precision versions of the preconditioner matrix.

Number of Processors	Computational Phase			
	Linear Solve		Overall	
	Double	Single	Double	Single
16	223s	136s	746s	657s
32	117s	67s	373s	331s
64	60s	34s	205s	181s
120	31s	16s	122s	106s

Conclusions

- Unstructured (static) grid codes can run well on distributed hierarchical memory machines, with attention to partitioning, vertex ordering, component ordering, blocking, and tuning
- Parallel solver libraries can give new life to the most valuable, discipline-specific modules of legacy PDE codes
- Parallel scalability is easy, but attaining high per-processor performance for sparse problems gets more challenging with each machine generation
- The NKS family of algorithms can be and must be tuned to an application-architecture combination; profiling is critical
- The hybrid MPI/OpenMP achieves good overall performance but should be used only in the phases that are not memory bandwidth limited

Acknowledgments

- Accelerated Strategic Computing Initiative, DOE
 - ⌚ *access to ASCI Red and Blue machines*
- National Energy Research Scientific Computing Center (NERSC), DOE
 - ⌚ *access to large T3E*
- SGI-Cray
 - ⌚ *access to large T3E*
- National Science Foundation
 - ⌚ *research sponsorship under Multidisciplinary Computing Challenges Program*
- U. S. Department of Education
 - ⌚ *graduate fellowship support for D. Kaushik*

Related URLs

- Follow-up on this talk

<http://www.mcs.anl.gov/petsc-fun3d>

- PETSc

<http://www.mcs.anl.gov/petsc>

- FUN3D

<http://fmad-www.larc.nasa.gov/~wanderso/Fun>

- ASCI platforms

<http://www.llnl.gov/asci/platforms>

- International Conferences on Domain Decomposition Methods

<http://www.ddm.org>

- International Conferences on Parallel CFD

<http://www.parcfd.org>