Parallel CFD Workshop, Experiences in Implementation

Efficient Parallelization of an Unstructured Grid Solver: A Memory-Centric Approach

Dinesh K. Kaushik
MCS Division, Argonne National Laboratory
& CS Deptartment, Old Dominion University
David. E. Keyes
CS Deptartment, Old Dominion University
& ICASE, NASA Langley Research Center

Acknowledgments
William D. Gropp
MCS Division, Argonne National Laboratory
Barry F. Smith
MCS Division, Argonne National Laboratory
W. K. Anderson
FMAD Division, NASA Langley Research Center

June 16, 1999

Organization of Presentation

- Issues for unstructured grid domain decomposition methods
- Implications of memory hierarchy
- Latency tolerance
- Background of FUN3D and PETSc
- Parallelization philosophy
- Illustrations of general performance issues
- Summary of serial and parallel performance
- Conclusions and future plans

Solving Unstructured Mesh Problems in Serial

Loss of **regularity** in unstructured mesh solvers

- makes them more **memory** intensive
- reduces the **locality in data refence** patterns (which is required to get good cache performance)
- results in very high **memory bandwidth** since cache lines might be loaded multiple times
- requires lot of **integer operations** that make these solvers more susceptible to run into **operation issue** limitations

Solving Unstructured Grid Problems in Parallel: Main Issues

- SPMD parallelization of unstructured grid solvers is complicated by the fact that no two interprocessor data dependency patterns are alike
- The user-provided global ordering may be incompatible with the subdomain-contiguous ordering required for high performance and convenient SPMD coding

The Widening Gap between Memory and CPU

- Memory latency improvement is about 7% per year.
- CPU improvement is 35% per year until 1986, and 55% per year thereafter.

Figure from Hennessy and Patterson, page 374.

Some Important Dimensionless Numbers

- Typical number of cycles lost on a cache miss:

$$10 - 100$$

- Typical number of cycles lost on a page fault:

$$10^6 - 10^7$$

- Typical transfer rate for eight bytes per floating point operation time:

$$10 - 100$$

- Typical message initiation latency per floating point operation time:

$$100 - 1,000$$

Implications of Memory Hierarchy

- Arrange for temporal locality
  - *Once an operand is cached on a processor, use it as many times as practical before sending it "down" or "out"*
- Arrange for spatial locality
  - *When an operand needs to be moved "up" or "across", fill up the slots in the same packet with other operands that will be required soon*
- Don't agonize over flops
  - *Flops are cheap compared with memory transfers, so do some extra work per data transfer-laden "cycle" if it reduces the number of cycles*
- Agonize over (low) bandwidth and (high) latency tolerance

Latency Tolerance — Architect's Perspective

In "latency" we include both the startup (size-independent) part of the data access (when remote data is ready) **and** the synchronization cost (when remote data is not ready). There are two classes of latency tolerance strategies (from D. Culler, et al., 1998, Chapter 11):

- Amortize the latency:
  - Block data transfers
- Cover the latency:
  - Precommunication
  - Proceeding past an outstanding communication in the same thread
  - Multithreading

The requirements are excess concurrency in the program (beyond the number of processors being used) and excess capacity in the memory and communication architecture.

Why Library Based Approach?

- compilers can not do all
- users should not do more than once
- advantges of modern programming practices should be harnessed

My Objective

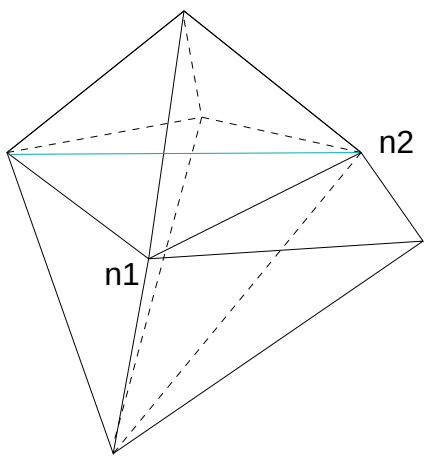
- To share with you our experiences of parallelizing an unstructured grid code using a library based approach
- To highlight the important issues (especially the memory centric code implementations) to get good performance

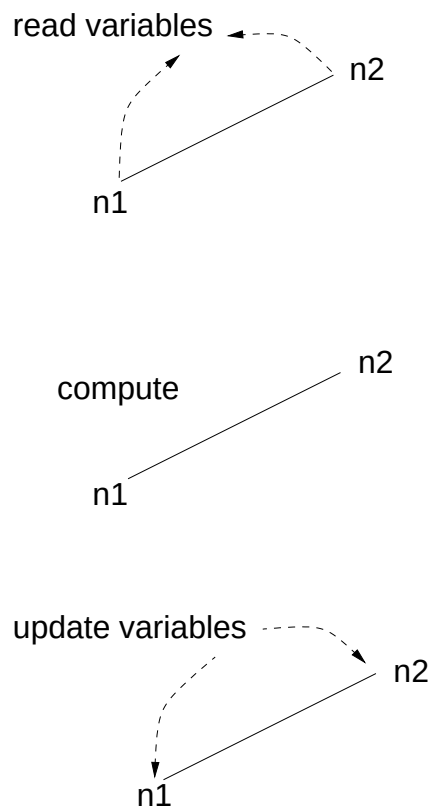
Description of the Legacy Code - FUN3D
(http://fmad-www.larc.nasa.gov/~wanderso/Fun/fun.html)

- FUN3D is a tetrahedral vertex-centered unstructured grid code developed by W. K. Anderson (LaRC) for compressible and incompressible Euler and Navier-Stokes equations
- Parallel experience is with Euler so far, but nothing in the solution algorithms or software changes when viscosity and turbulence are added; only convergence rate will vary with conditioning, as determined by Reynolds number (and mesh)
- FUN3D uses 1st- or 2nd-order Roe for convection and Galerkin for diffusion, and false timestepping with backwards Euler for nonlinear continuation towards steady state
- Solver is Newton-Krylov; timestep is advanced towards infinity by the switched evolution/relaxation (SER) heuristic of Van Leer & Mulder

Edge-based Loops for Flux Computation

- Used inside Newton loop in every residual evaluation
- Used inside Krylov loop in every matrix-vector product





Variables at each node:
density,
momentum (*x,y,z*),
energy,
pressure

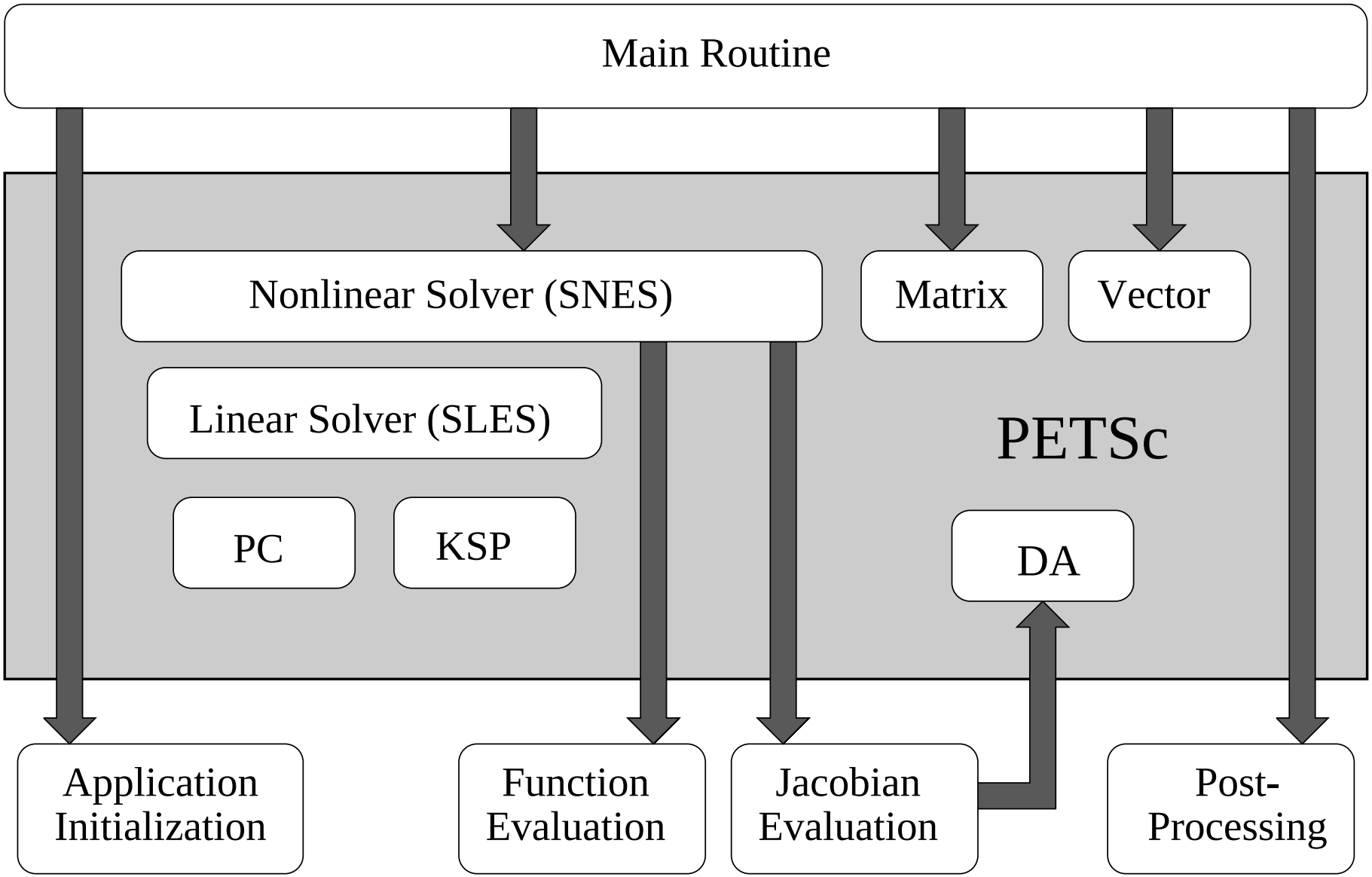
Variables at edge:
identity of nodes,
orientation(*x,y,z*)

PETSc —
a Portable Extensible Toolkit for Scientific Computing

- Gives relatively high-level expression to preconditioned iterative linear solvers, and Newton iterative methods
- Supports complex arithmetic
- Ports wherever MPI ports; committed to progressive MPI tuning
- Permits great flexibility (through object-oriented philosophy) for algorithmic innovation
- Freely available **(http://www.mcs.anl.gov/petsc)**
- Callable from FORTRAN77, C, and C++; written in C
- Includes diagnostic, monitoring, and visualization GUIs

Integration with the Library - PETSc

(http://www.mcs.anl.gov/petsc)



Object-Oriented Design of PETSc — Abstraction

- Abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects.
- Each PETSc object has an abstract structure defined in
  `$(PETSC_DIR)/src/<component>/<component>impl.h` .
  For example,

```
struct _p_Vec {
   PETSCHEADER
   struct _VeOps ops;
   void          *data;
};

typedef _p_Vec* Vec;
```

Object-Oriented Design of PETSc — Encapsulation

- Information hiding is very important for the development of representation free (generic) data structures.
- PETSc hides the implementation from interface. This is done by providing the pointer, `void *data` which is implementation specific.
- At the time of construction, each object provides a representation for its private context (i.e., the data pointer) and supplies the implementation of member functions (which are declared static to limit their visibility to the source code within a particular file only).

Object-Oriented Design of PETSc — Modularity

- Modularity is the division of a complex system into several tractable components.
- Higher level abstractions (data structures and routines) are built by using lower level abstractions, which may be built on even simpler modules.
- Nonlinear solver (SNES) context is built on the top of linear solver (SLES) context which is built on the sparse matrix class.

Object-Oriented Design of PETSc — Hierarchy

- Hierarchy is the ranking or ordering of abstractions.
- Inheritance (with polymorphism) is a very powerful mechanism for software reuse and future extensibility.
- PETSc has only a shallow inheritance tree.
- Each implementation of a particular abstract structure is an instance of a derived object, e.g., MatSeqAIJ, MatMPIAIJ, MatSeqBAIJ etc.

Solving a Nonlinear Problem with PETSc

```
struct _p_SNES {
  PETSCHEADER

  /*  ------------------------- User-provided stuff ----------------------
  void  *user;                              /* user-defined context */

  int   (*computefunction)(SNES,Vec,Vec,void*); /* function routine */

  int   (*computejacobian)(SNES,Vec,Mat*,Mat*,MatStructure*,void*);
  ..
  ..
  ..
 }
```

Conversion of Legacy FUN3D into PETSc/MPI version

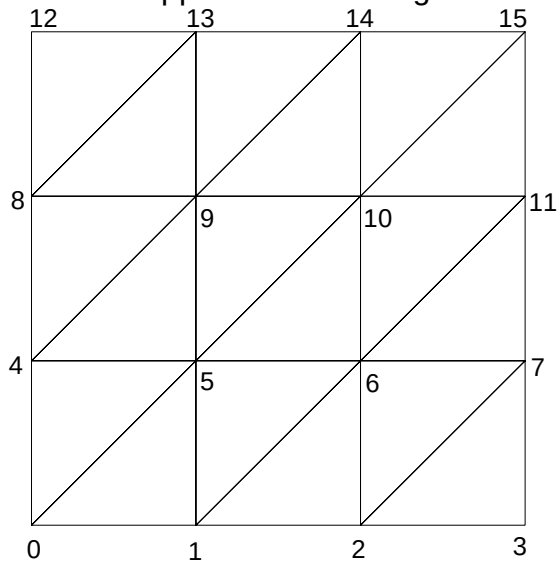
- Project begun 10/96, completed 3/97, undergoing continual enhancement
- Five-month (part-time) effort included:
  - learning FUN3D and the PUNS3D mesh preprocessor
  - learning the MeTiS partitioner
  - adding and testing new functionality in PETSc
  - restructuring FUN3D from vector to cache orientation
- Approximately 3,300 of 14,400 F77 lines of FUN3D retained (primarily as "node code" for flux and Jacobian evaluations); PETSc solvers used for the rest

Solving Unstructured Grid Problems in Parallel: Basic Outline of the Solution Strategy

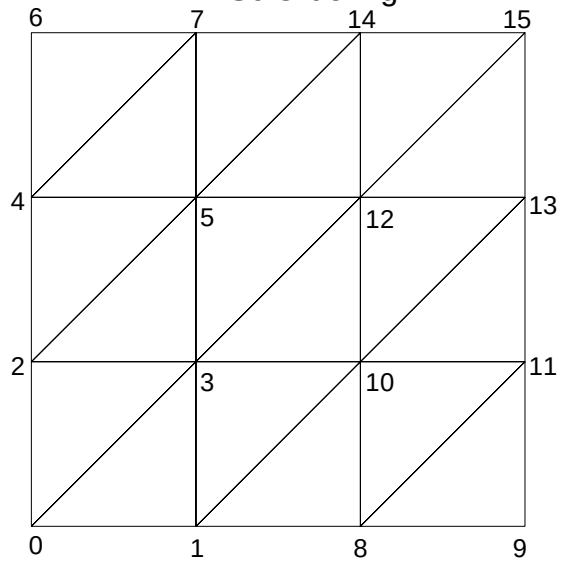
- Follow the "owner computes" rule under the dual constraints of minimizing the number of messages and overlapping communication with computation
- Each processor "ghosts" its stencil dependences in its neighbors
- Ghost nodes ordered after contiguous owned nodes
- Domain mapped from (user) global ordering 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 (nonblocking for overlap where hardware supports)

Three Different Orderings - In Focus

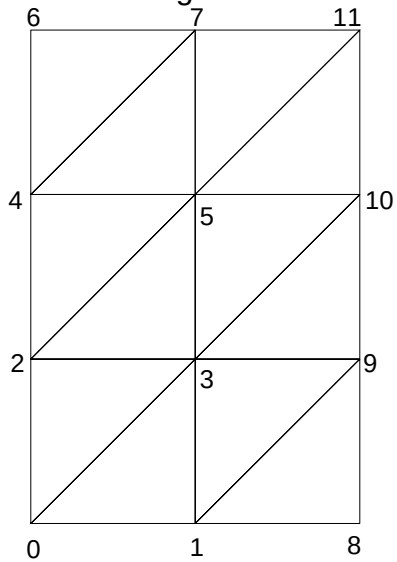
Application Ordering



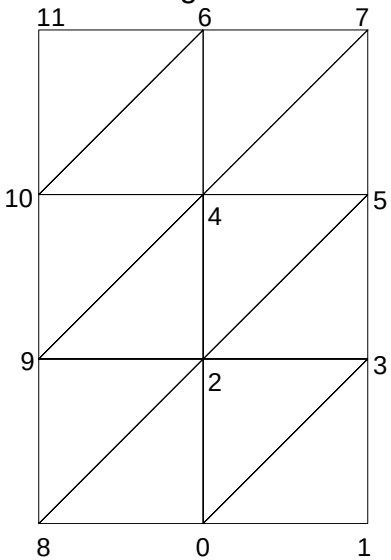
PETSc Ordering



Local Ordering for Processor 0



Local Ordering for Processor 1



Scattering Between the Orderings

- After establishing different orderings, establish the "scatter" between the global and local vectors in the following way :

```
ISCreateStride(MPI_COMM_SELF,bs*nvertices,0,1,&islocal);
ISCreateBlock(MPI_COMM_SELF,bs,nvertices,svertices,&isglobal);
VecScatterCreate(x,isglobal,user.localX,islocal,&user.scatter);
```

- Next, before using the local vector in any subroutine, carry out the scatter operation :

```
VecScatterBegin(X,localX,INSERT_VALUES,SCATTER_FORWARD,scatter);
VecScatterEnd(X,localX,INSERT_VALUES,SCATTER_FORWARD,scatter);
```

Pseudo-Transient Newton-Krylov-Schwarz Algorithm

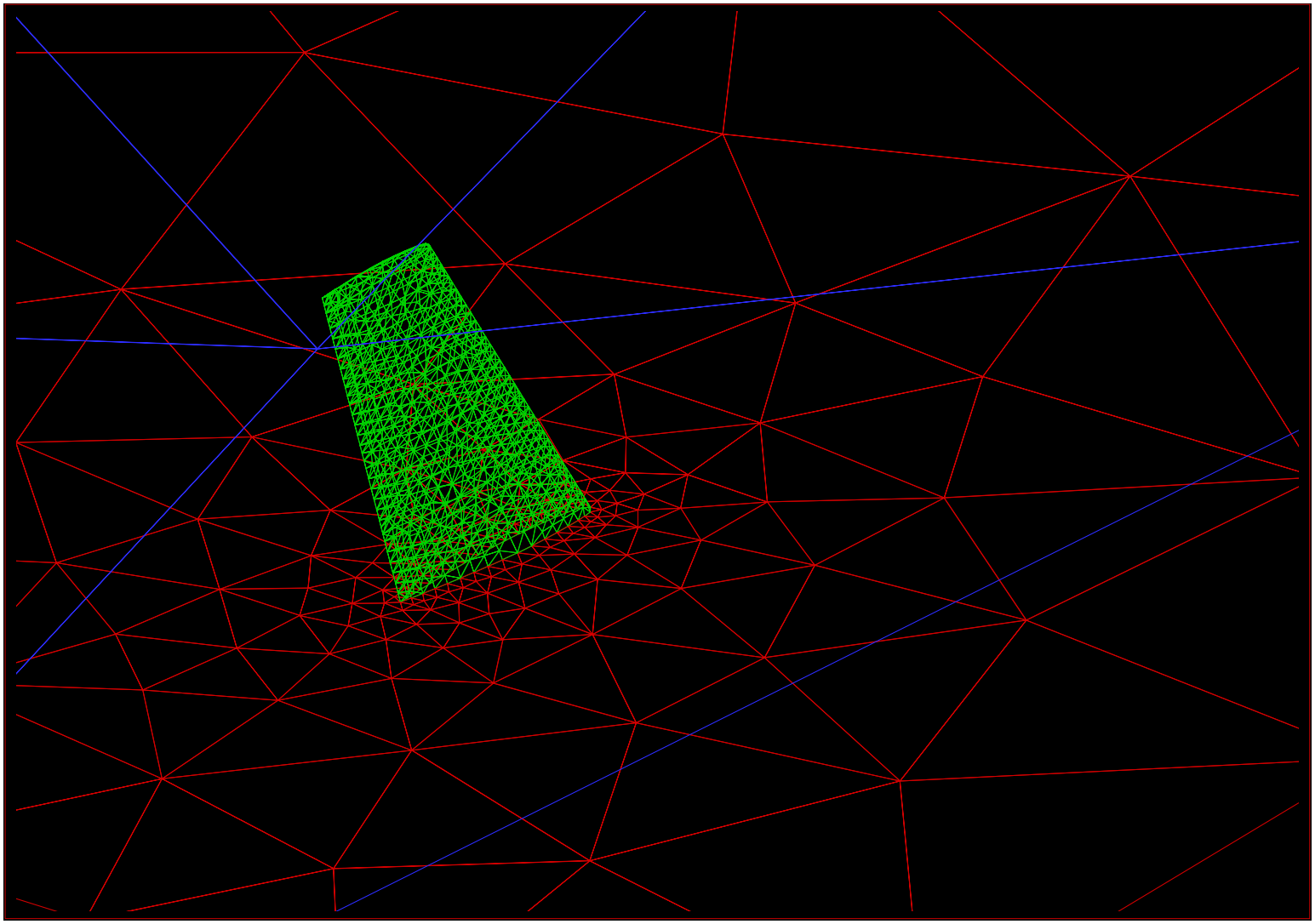
(after Cai, Gropp, Keyes, and Tidriri (1994))

```
for (l = 0; l < n_time; l++) {
  SELECT TIME-STEP
  for (k = 0; k < n_Newton; k++) {
    compute nonlinear residual and Jacobian
    for (j = 0; j < n_Krylov; j++) {
      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
```

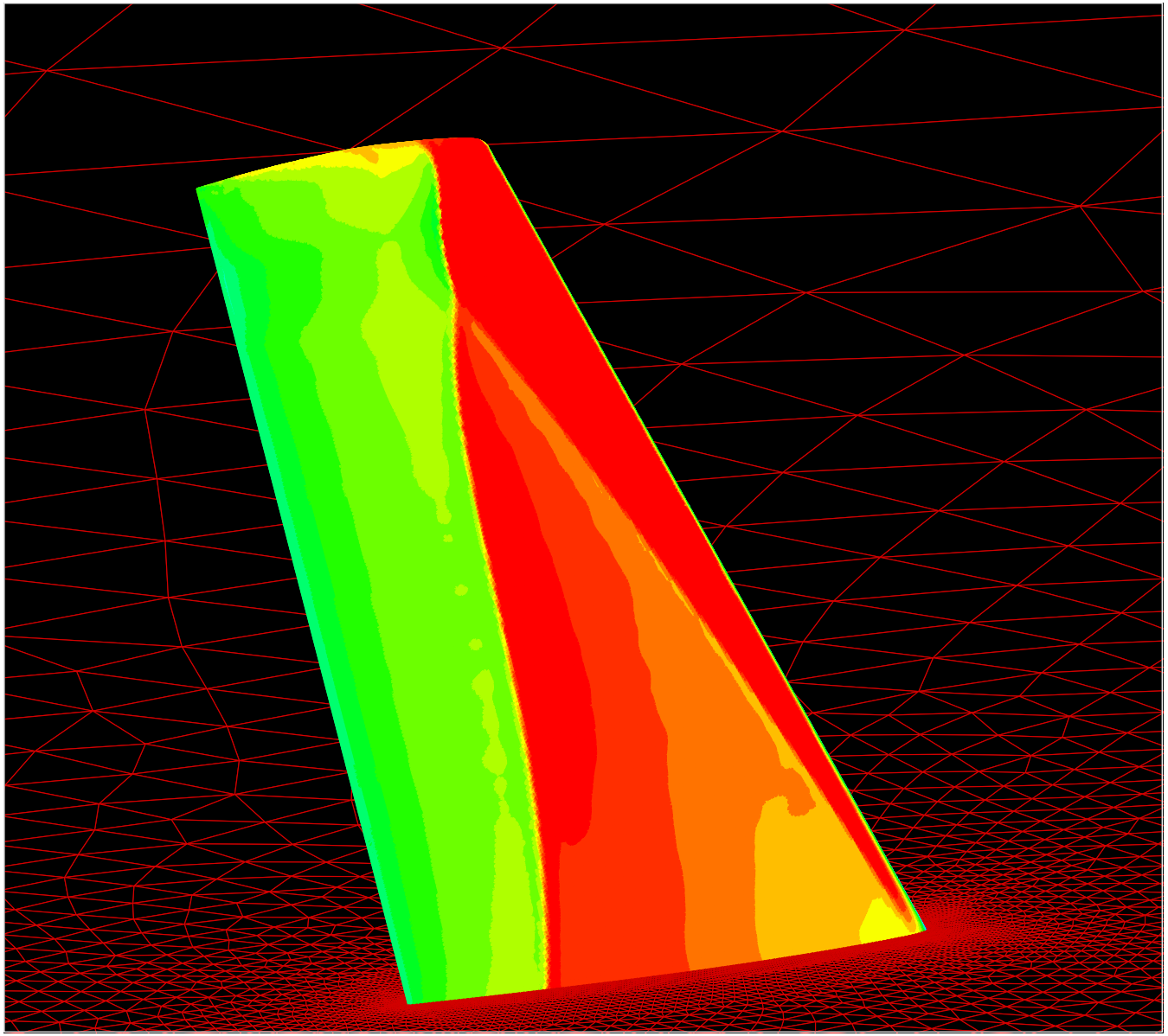
Problem Specific Settings for ΨNKS

- "its" represents the number of pseudo-transient Newton steps — one Newton step per timestep, with SER growth in timestep up to a CFL of 100,000, and with a maximum number (60) of Schwarz-preconditioned GMRES steps per Newton step with relative tolerance of 10^{-2}
- Convergence defined as a relative reduction in the norm of the steady-state nonlinear residual by a factor of 10^{-10}
- Jacobian matrix is never constructed explicitly; it is applied in matrix free way with second order of accuracy for the final solution
- The preconditioner (incomplete LU with zero fill) in each domain is derived from from 1st-order accurate jacobian

Surface Visualization of Test Domain (M6 wing)



Illustrative Solution of "Lambda Shock" Case



Performance Tuning – Three Fronts

- Algorithmic Tuning
  - Choose "optimal" compromise of large number of nonorthogonal parameters
- Compiler Transformations
  - Free the compiler to do what it does best
- Data Layouts
  - Stay in harmony with the memory hierarchy

Algorithmic Tuning for NKS Solver

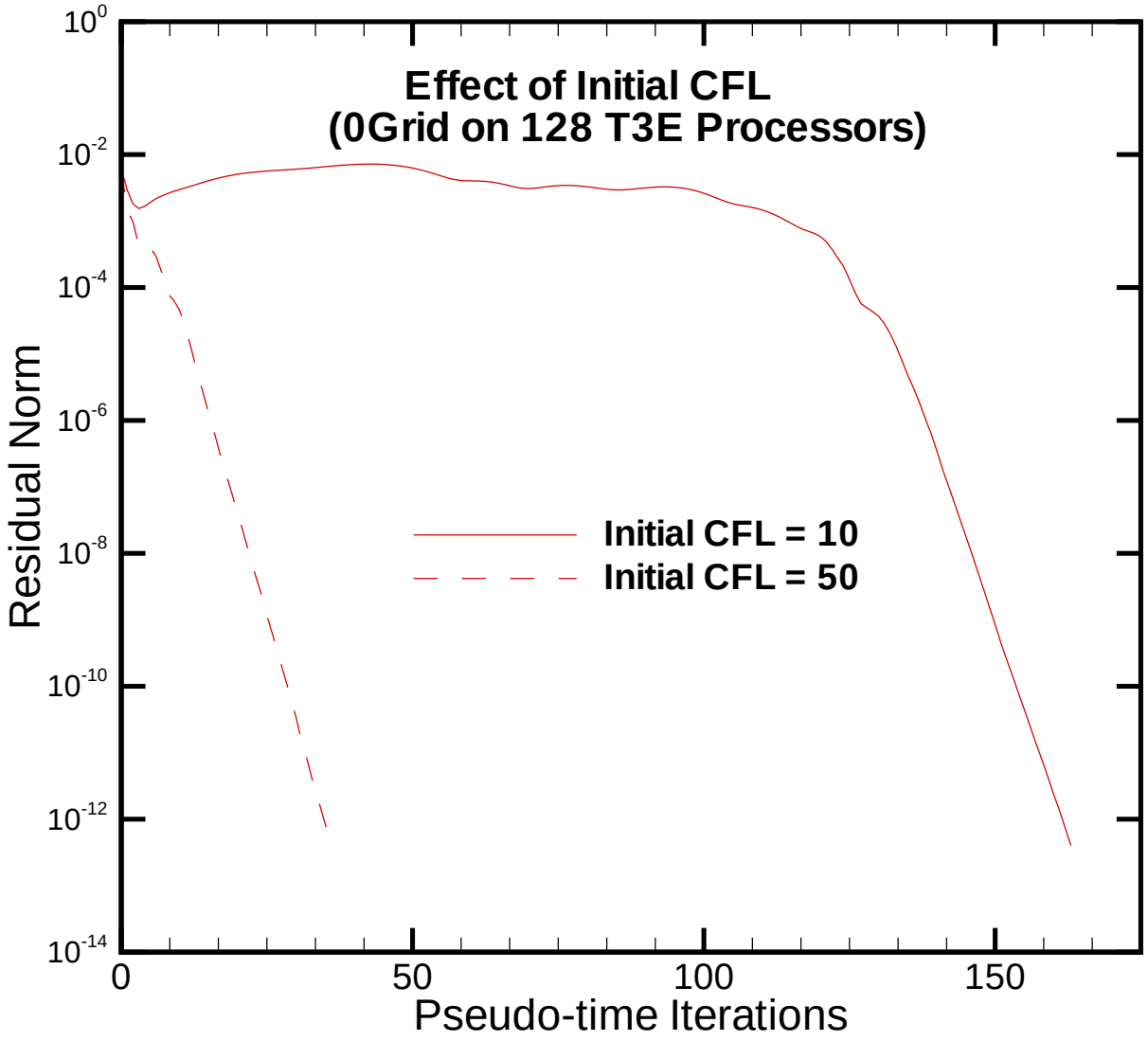
- Continuation parameters: discretization order, initial timestep, timestep evolution
- Newton parameters: convergence tolerance, globalization strategy, Jacobian refresh frequency
- Krylov parameters: convergence tolerance, subspace dimension, restart number, orthogonalization mechanism
- Schwarz parameters: subdomain number, subdomain solver, subdomain overlap, coarse grid usage
- Subproblem parameters: fill level, number of sweeps

Algorithmic Tuning – Continuation Parameters

- SER heuristic

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

- Parameters of Interest
  - Initial **CFL** number
  - Exponent in the Power Law
    > 1 for first-order discretization (1.5)
    < 1 at outset of second-order discretization (0.75)
    $= 1$ normally
  - Switch over Ratio between FO and SO


Effect of Initial CFL
(0Grid on 128 T3E Processors)
Residual Norm
Pseudo-time Iterations
Initial CFL = 10
Initial CFL = 50
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
0
50
100
150

Algorithmic Tuning – Krylov Parameters

- These parameters were chosen after lot of experiments
- Convergence Tolerance
  - a value of 0.01 works well for most of the cases run
- Subspace Dimension
  - depends on the problem dimension
  - typical values range from 10 (for smallest problem) to 60 for the largest problem
- Resrtart Number
  - dependent on the available memory
  - typical values are 15 to 30

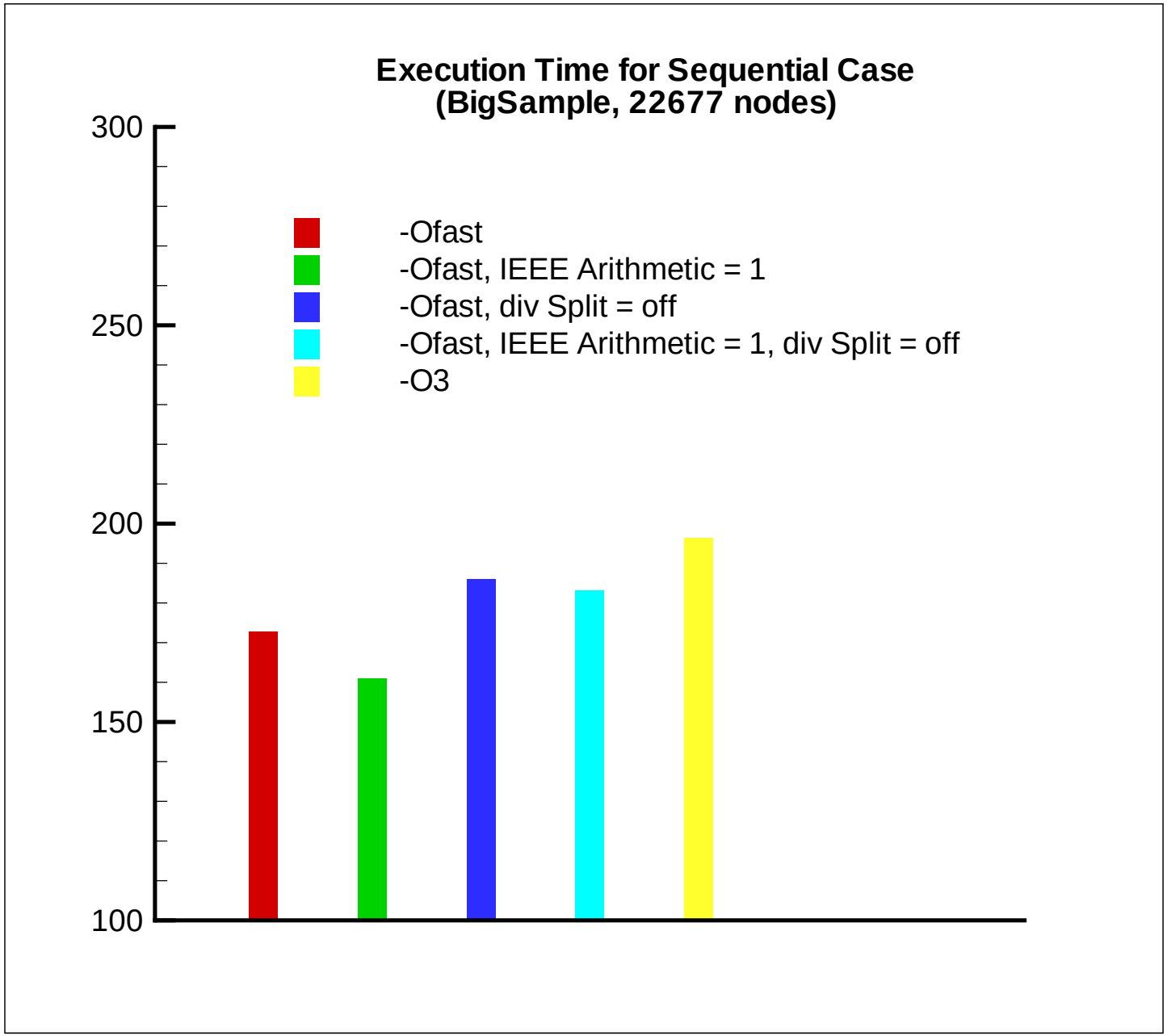
Optimal Granularity of Decomposition

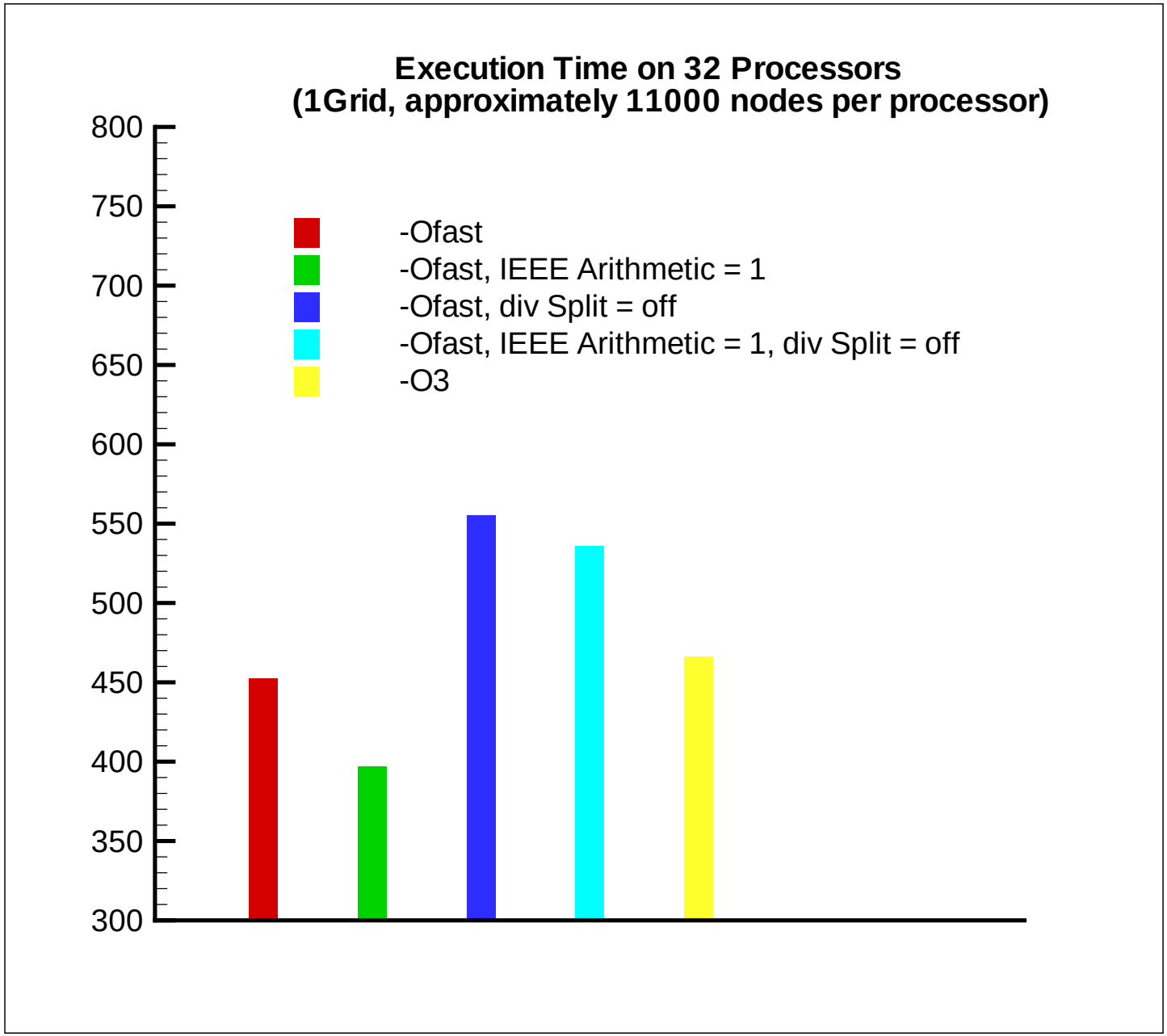
For **cache-based microprocessors**, granularity of domain decomposition iterative methods is determined by three forces:

- **Convergence Rate**
  usually deteriorates with increased granularity
- **Communication Volume**
  increases with increased granularity
- **Size of Local Working Set**
  fits better into successively smaller cache levels with increased granularity

Compiler Transformations

- Choose the highest level of optimization that give the right result
- Effect of different compiler flags (Origin 2000)
  - **-Ofast** : does aggressive optimization (including O3 optimizations)
  - **-OPT:IEEE_arithmetic=1** : inhibits optimizations that produce less accurate results than required by ANSI/IEEE 754-1985
  - **-OPT:div_split=off** : disables the calculation of x/y as x*(1.0/y)
  - **-O3** : level 3 optimization


Execution Time for Sequential Case
(BigSample, 22677 nodes)
300
250
200
150
100
-Ofast
-Ofast, IEEE Arithmetic = 1
-Ofast, div Split = off
-Ofast, IEEE Arithmetic = 1, div Split = off
-O3


Execution Time on 32 Processors
(1Grid, approximately 11000 nodes per processor)
800
750
700
650
600
550
500
450
400
350
300
-Ofast
-Ofast, IEEE Arithmetic = 1
-Ofast, div Split = off
-Ofast, IEEE Arithmetic = 1, div Split = off
-O3

Data Layouts

- Choose data layouts that enhance locality at every level of Memory hierarchy
- Storage/use patterns should follow memory hierarchy
  - **Blocks** for **Registers**
    block storage format for multicomponent systems – saves CPU cycles
  - **Interlaced Data Structures** for **Cache**
    choose

    $$u1, v1, w1, p1, u2, v2, w2, p2, \ldots$$

    in place of

    $$u1, u2, \ldots, v1, v2, \ldots, w1, w2, \ldots, p1, p2, \ldots$$
  - **Subdomains** for **Distributed Memory**
    "chunky" domain decomposition for optimal surface-to-volume (communication-to-computation) ratio
  - This hierarchy is concerned with different issues than the **algorithmic efficiency** issues associated with hierarchies of grids

Data Layouts(contd.) – Reorderings

- Edge Reordering
  - sort the nodes at either ends of the edges
  - this effectively transforms an edge based loop into a node based loop
  - enhances temporal locality
- Node Reordering
  - ∗ Reverse Cuthill Mckee (**RCM**)
    ∗ Fast Sloan

Locality Enhancing Strategies in Serial

- Flow over M6 wing with fixed-size grid of 22,677 vertices (90,708 DOFs incompressible; 113,385 DOFs compressible)
- Turn on each optimization one by one to isoalate the effect of each
- Five architectures considered: Cray T3E, IBM SP, Origin 2000, Intel Pentium, and Sun Ultra
- Impact of these techniques vary on different architecures — improvement ranges from **2.5 on Pentium** to **7.5 on SP**

Sequential Performance on IBM SP

IBM P2SC ("thin"), 120MHz, cache: 128KB data and 32 KB instr

Enhancements			Results			
Field Interlacing	Structural Blocking	Edge Reordering	Incompressible Time/Step	Incompressible Ratio	Compressible Time/Step	Compressible Ratio
			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

Sequential Performance on Intel Pentium

Intel Pentium II (NT), 400MHz, cache: 16KB data / 16KB instr / 512KB L2

Enhancements			Results			
			Incompressible		Compressible	
Field Interlacing	Structural Blocking	Edge Reordering	Time/Step	Ratio	Time/Step	Ratio
			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

Sequential Performance on SGI Origin

MIPS R10000, 250MHz, cache: 32KB data / 32KB instr / 4MB L2

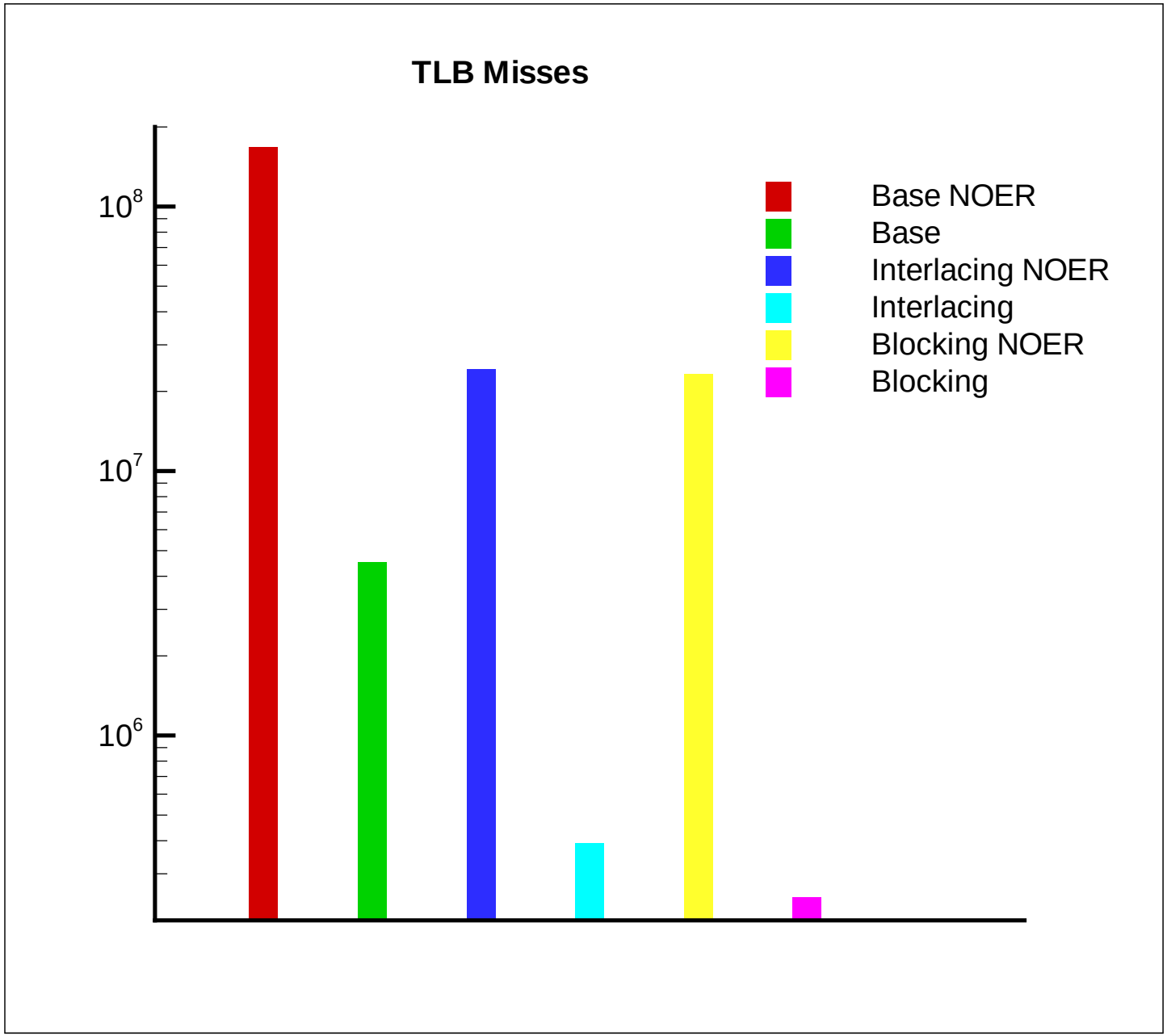
Enhancements			Results			
Field Interlacing	Structural Blocking	Edge Reordering	Incompressible Time/Step	Incompressible Ratio	Compressible Time/Step	Compressible Ratio
			83.6s	—	140.0s	—
×			36.1s	2.31	57.5s	2.44
×	×		29.0s	2.88	43.1s	3.25
		×	29.2s	2.86	59.1s	2.37
×		×	23.4s	3.57	35.7s	3.92
×	×	×	16.9s	4.96	24.5s	5.71

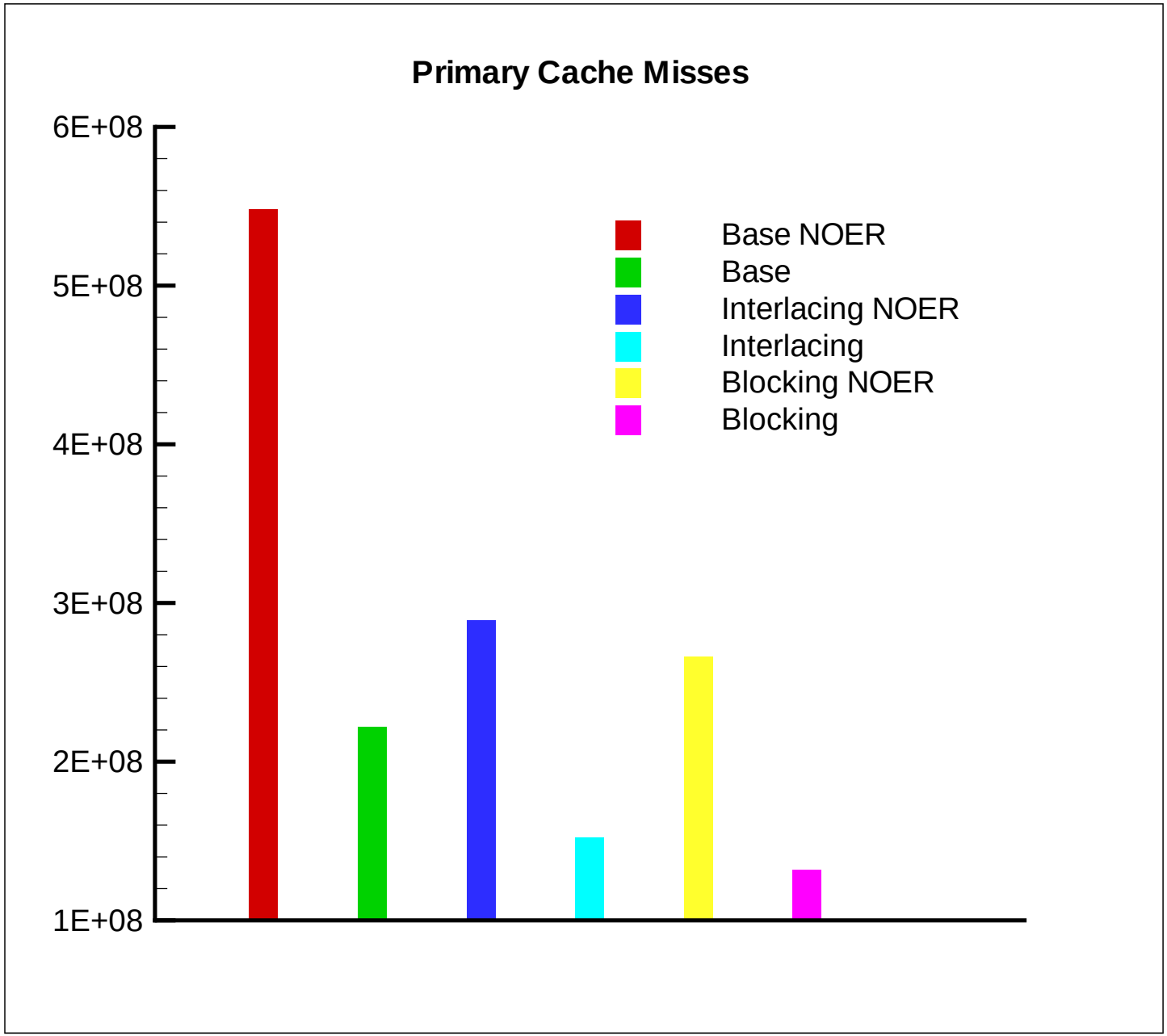
Performance Monitoring – Hardware Counters

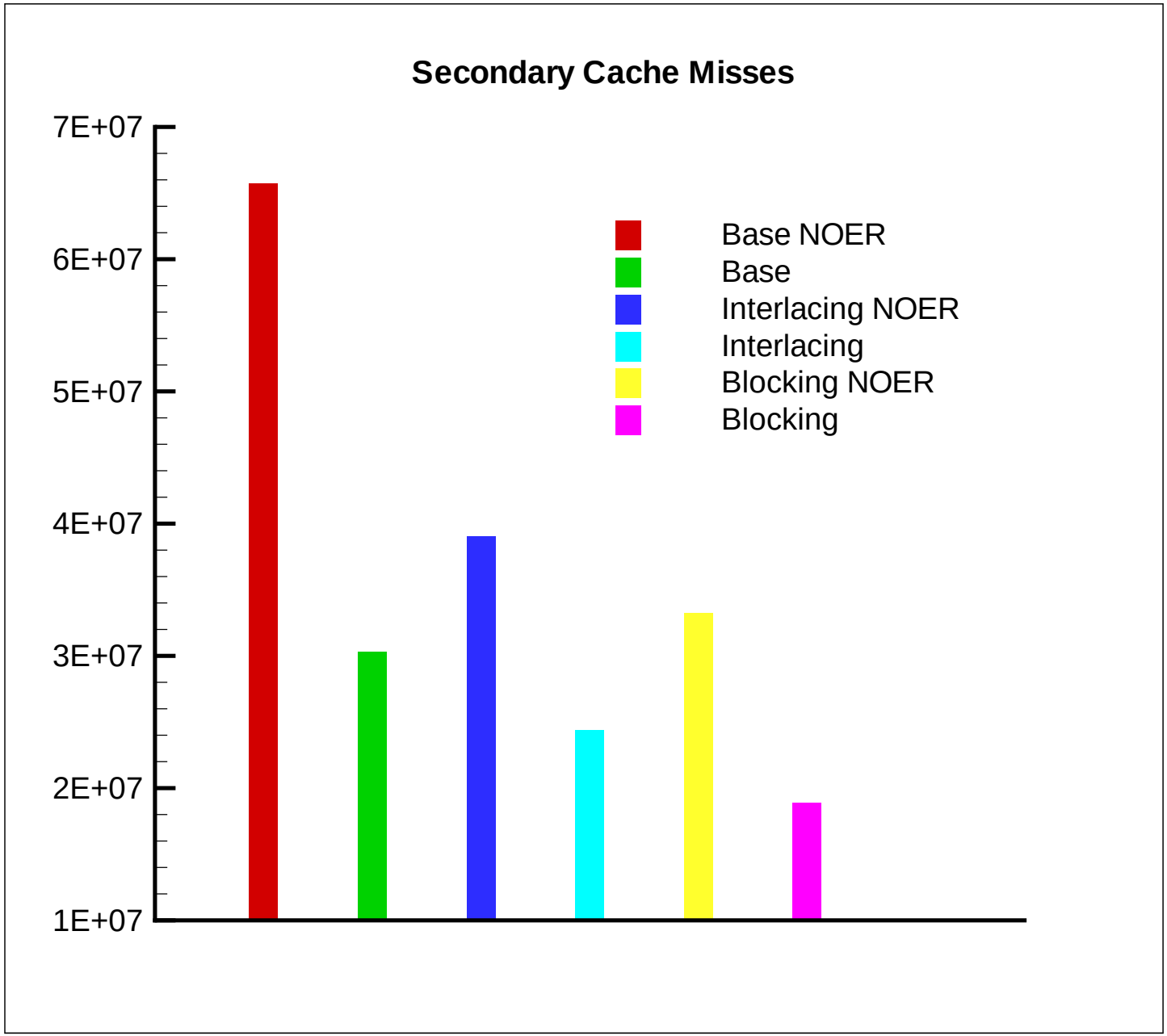
- Hardware counters available on almost all modern architectures
- Each vendor provides own interface performance monitoring
- At least two independent efforts to provide a portable user interface
  - PCL — The Performance Counter Library from Central Institute for Applied Mathematics, Research Centre Juelich, Germany
  - RABBIT — A Performance Counters Library for Intel Processors and Linux from Ames Laboratory
- PerfAPI — Performance Data Standard and API project is directed towards a possible standard

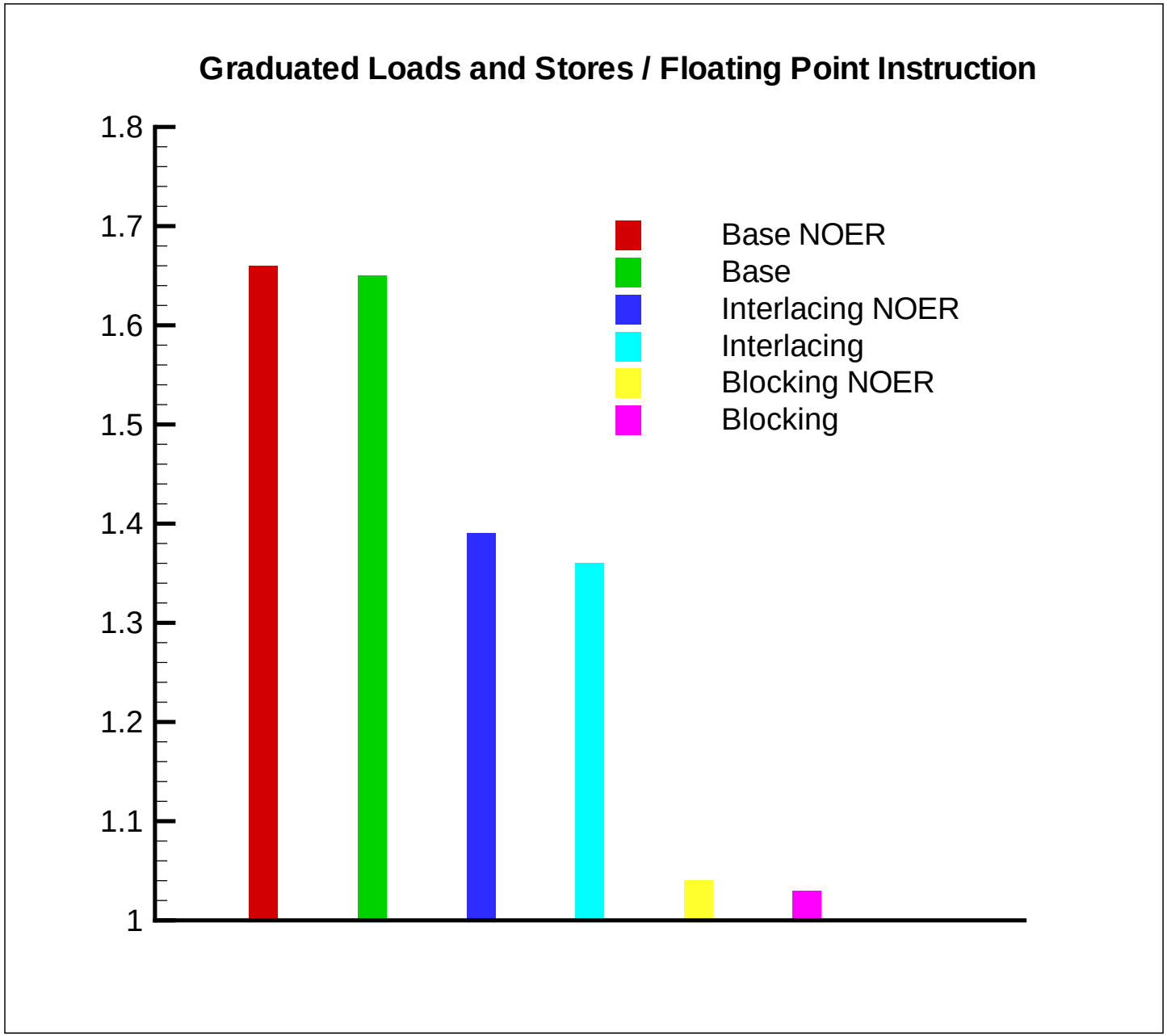
Hardware Profiling on SGI Origin

- TLB Misses
- Primary Cache Misses
- Secondary Cache Misses
- Graduated Loads and Stores Per Floating Point Instruction


TLB Misses
10^8
10^7
10^6
Base NOER
Base
Interlacing NOER
Interlacing
Blocking NOER
Blocking


Primary Cache Misses
6E+08
5E+08
4E+08
3E+08
2E+08
1E+08
Base NOER
Base
Interlacing NOER
Interlacing
Blocking NOER
Blocking


Secondary Cache Misses
7E+07
6E+07
5E+07
4E+07
3E+07
2E+07
1E+07
Base NOER
Base
Interlacing NOER
Interlacing
Blocking NOER
Blocking


Graduated Loads and Stores / Floating Point Instruction
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
Base NOER
Base
Interlacing NOER
Interlacing
Blocking NOER
Blocking

Notes on Parallel Efficiency

Conflicting definitions of parallel efficiency abound, depending upon two choices:

- What scaling is to be used as the number of processors is varied?
  - overall fixed-size problem
  - varying size problem with fixed memory per processor
  - varying size problem with fixed work per processor
- What form of the algorithm is to be used as number of processor is varied?
  - reproduce the sequential arithmetic exactly
  - adjust parameters to perform best on each given number of processors

Our charts include both overall fixed-size scaling and approximately fixed memory per processor (Gustafson) scaling

We always adjust the subdomain blocking parameter to match the number of processors, one subdomain per processor; this causes the number of iterations to vary

Notes on Parallel Efficiency, cont. (2)

Effect of changing-strength preconditioner and effect of parallel overhead are often separated into algorithmic and implementation factors

- Customary definition of overall efficiency in going from q to p processors ($p > q$):

$$\eta(p|q) = \frac{q \cdot T(q)}{p \cdot T(p)}$$

  where $T(p)$ is the overall execution time on p processors (measured)

- Factor $T(p)$ into $I(p)$, the number of iterations, and $C(p)$, the average cost per iteration.

- Algorithmic efficiency is measure of preconditioning quality (measured):

$$\eta_{alg}(p|q) = \frac{I(q)}{I(p)}$$

- Implementation efficiency is remaining (inferred, not directly measurable) factor:

$$\eta_{impl}(p|q) = \frac{q \cdot C(q)}{p \cdot C(p)}$$

Notes on Parallel Efficiency, cont. (3)

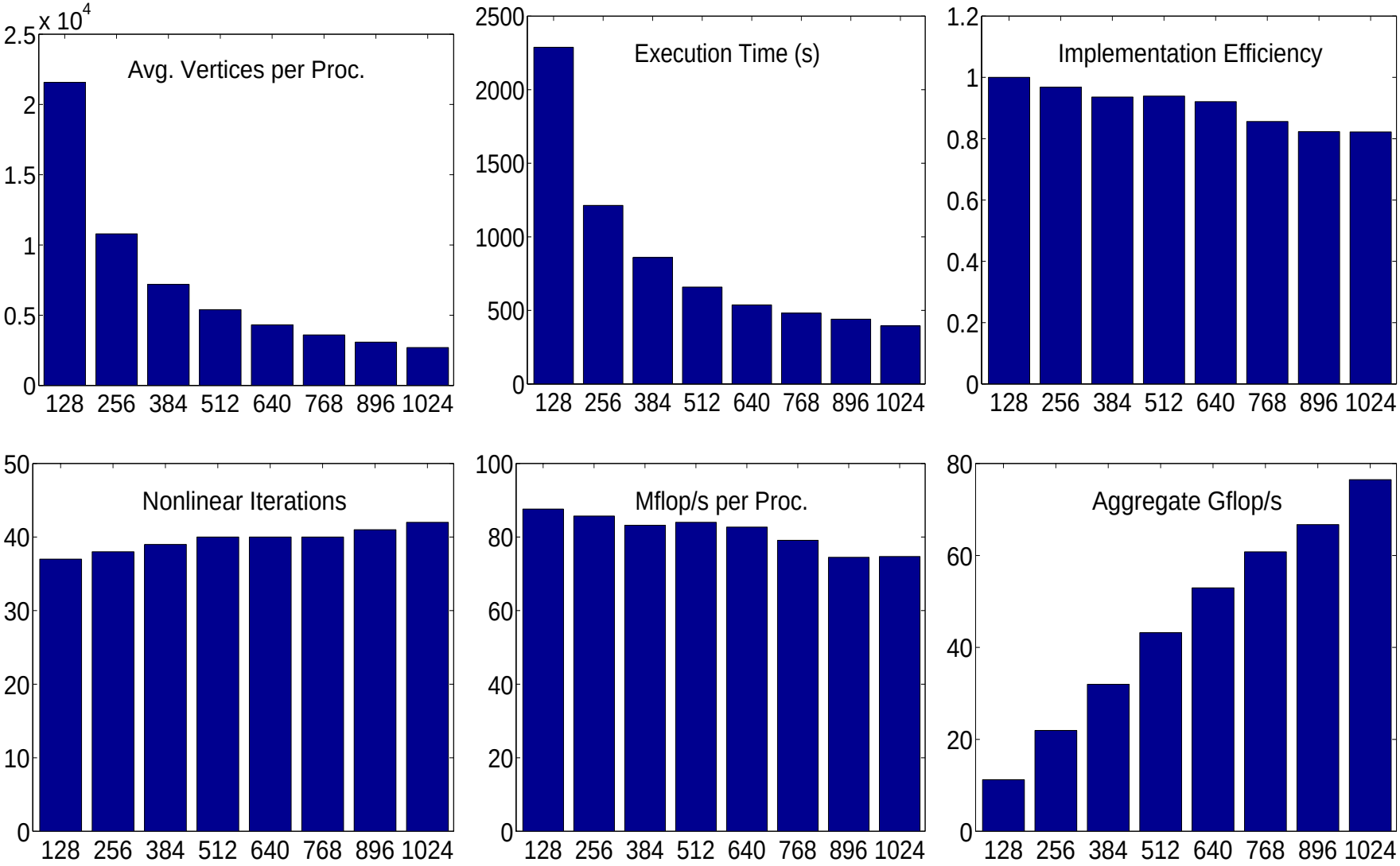
- Convergence rate typically degrades slightly as number of processors is increased, due to introduction of concurrency in preconditioner — highly partition-dependent
- Implementation efficiency may improve slightly as processors are added, due to smaller workingsets — better cache residency
- Implementation efficiency ultimately degrades as communication-to-computation ratio increases

Measuring the Parallel Performance

- We have used PETSc's profiling and logging features
- PETSc uses manual counting of flops, which are afterwards aggregated over all the processors for parallel performance statistics
- Time devoted to I/O and data partitioning
- Code is preloaded into memory by doing one nonlinear iteration to avoid the timing variations caused by disk paging
- Our base processor number (for scalability calculations) is such that the problem has just fit into the local memory

Parallel Performance of Incompressible Solver on Cray T3E

ONERA M6 Wing Test Case, Tetrahedral grid of 2.8 million vertices (about 11 million unknowns) on upto 1024 Cray T3E 600 MHz processors



Parallel Performance on Cray T3E, contd.

procs	its	time	speedup	Efficiency			Communication		sustained Mflop/s per proc.	sustained total Gflop/s
				η_{alg}	η_{impl}	$\eta_{overall}$	inner prod.	halo exch.		
128	37	2,288.5s	1.00	1.00	1.00	1.00	7%	3%	87.6	11.2
256	38	1,213.5s	1.89	0.97	0.97	0.94	8%	4%	85.7	21.9
512	41	658.6s	3.47	0.90	0.96	0.87	9%	4%	84.0	43.2
1024	42	394.6s	5.80	0.88	0.82	0.72	20%	6%	74.7	76.5

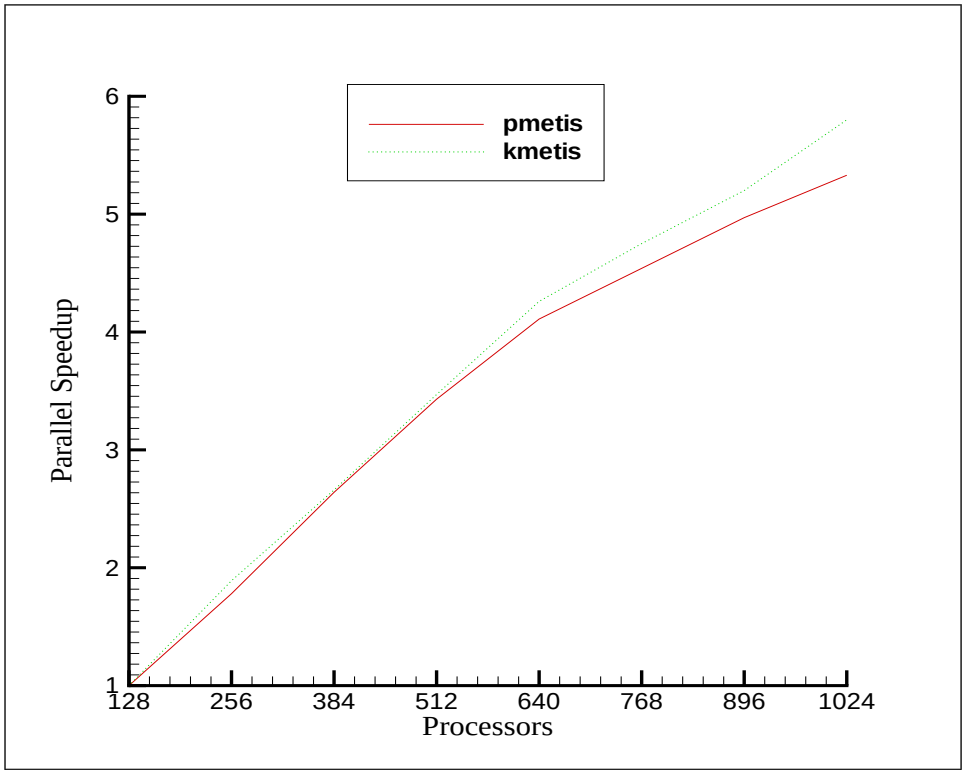
- Principal nonscaling feature is **global inner products** (growing to **20%** of execution time for 1024 nodes), due mostly to synchronization delays

Multithreading for Synchronization Relief

- Typical critical path in a nonlinear implicit method:
  `..., solve, bound_step, update, solve, bound_step, update, solve, ...`
- There are other useful operations to perform, off the critical path:
  - refresh Jacobian
  - test convergence
  - adapt continuation and algorithmic parameters
  - perform interprocessor communication and disk I/O
  - visualize, filter, compress, data mine, ...
- Some "off the path" tasks enjoy considerable locality
- Parallelizability of "naked" sparse linear problems may be unrealistically pessimistic, relative to full simulations

Effect of Data Partitioning Strategies:
pmetis vs. kmetis

- pmetis attempts to balance the number of nodes and edges on each partition
- kmetis tries to reduce the number of non-contiguous subdomains and connectivity of the subdomains
- We observe that kmetis gives slightly better scalability (especially when granularity is large)



Parallel Scalability of M6 Euler Flow with Fixed-Storage-Per-Processor

Vertices	No. Procs	Vert/Proc	Its	Time	Time/It
357,900	80	4474	78	559.93s	7.18s
53,961	12	4497	36	265.72s	7.38s
9,428	2	4714	19	131.07s	6.89s

- Convergence is strongly dependent on resolution
- Implementation efficiency is roughly flat (same execution time per iteration when same number of vertices per processor)

Parallel Performance of Compressible Solver on Cray T3E, IBM SP, and SGI Origin

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

- Number of vertices per processor ranges from 22,369 to 3,729
- Variations in iterations across machines may be attributable to different floating points and different orders of commutative operations (e.g., inner products)

Compressible versus Incompressible: Interesting Differences Beyond the Physics

- 5×5 blocks of compressible model versus 4×4 blocks of incompressible model change
  - Balance of work performed (volumetric) to data communicated (surfacial)
  - Balance of work performed on in-cache data to data loaded into cache from memory
  - Balance of work performed in different phases of the code, e.g., Jacobian evaluation and linear solution
- In addition, compressible and incompressible models exhibit differences in convergence rates and robustness

Comparison of Euler Flow Regimes over M6 Wing on SGI Origin 2000

Fixed Size Scaling: 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 (Mach 0) (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

Comparison of Euler Flow Regimes over M6 Wing on SGI Origin 2000

Fixed Size Scaling: 357,900 vertices
(1,789,500 DOFs compressible)

No. Procs.	Steps	Time per Step	Per-Step Speedup	Impl. Eff.	FcnEval Mflop/s	JacEval Mflop/s
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

Conclusions

- The near-scalable algorithms for general purpose PDE simulations that we use today can in theory be adapted to an architectural climate of diverging rates of computation and memory access, requiring increased concurrency with concentrated locality
- But, in practice, we must simultaneously improve algorithmic tolerance to the memory latency of the architecture

Future Directions

- Architecture-oriented
  - correlate hardware counter measurements with data structure organization and refine cache strategies in a quantitative way
- Programming model-oriented
  - examine appropriate role of multi-threading within a subdomain in a hybrid DSM/SMP programming style
- Application-oriented
  - examine the relative advantages of structured and unstructured grids from a performance perspective (partitioning and ordering flexibility versus representation efficiency)

References

- *Achieving High Sustained Performance in an Unstructured Mesh CFD Application* (with Anderson, Gropp, Keyes, and Smith), 1999, submitted to Gordon Bell Prize Committee.
  - right performance metrics
- *Newton-Krylov-Schwarz Methods for Aerodynamics Problems: Compressible and Incompressible Flows on Unstructured Grids* (with Keyes and Smith), 1998, submitted to "Proc. of the 11th Intl. Conf. on Domain Decomposition Methods", C.-H. Lai et al., eds.
  - multi-platform comparisons focus
- *Prospects for CFD on Petaflops Systems* (with Keyes and Smith), 1997, in "CFD Review 1998", M. Hafez and K. Oshima, eds., World Scientific, Singapore, pp. 1079-1096.
  - parallel scalability focus
- all these can be downloaded from
  - *http://www.mcs.anl.gov/petsc-fun3d*