

# Letting Physicists be Physicists, and Other Goals of the TOPS Scalable Solver Project

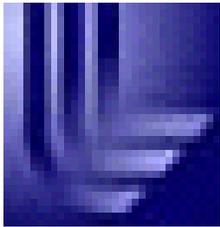


David E. Keyes

Department of Applied Physics & Applied Mathematics

Columbia University

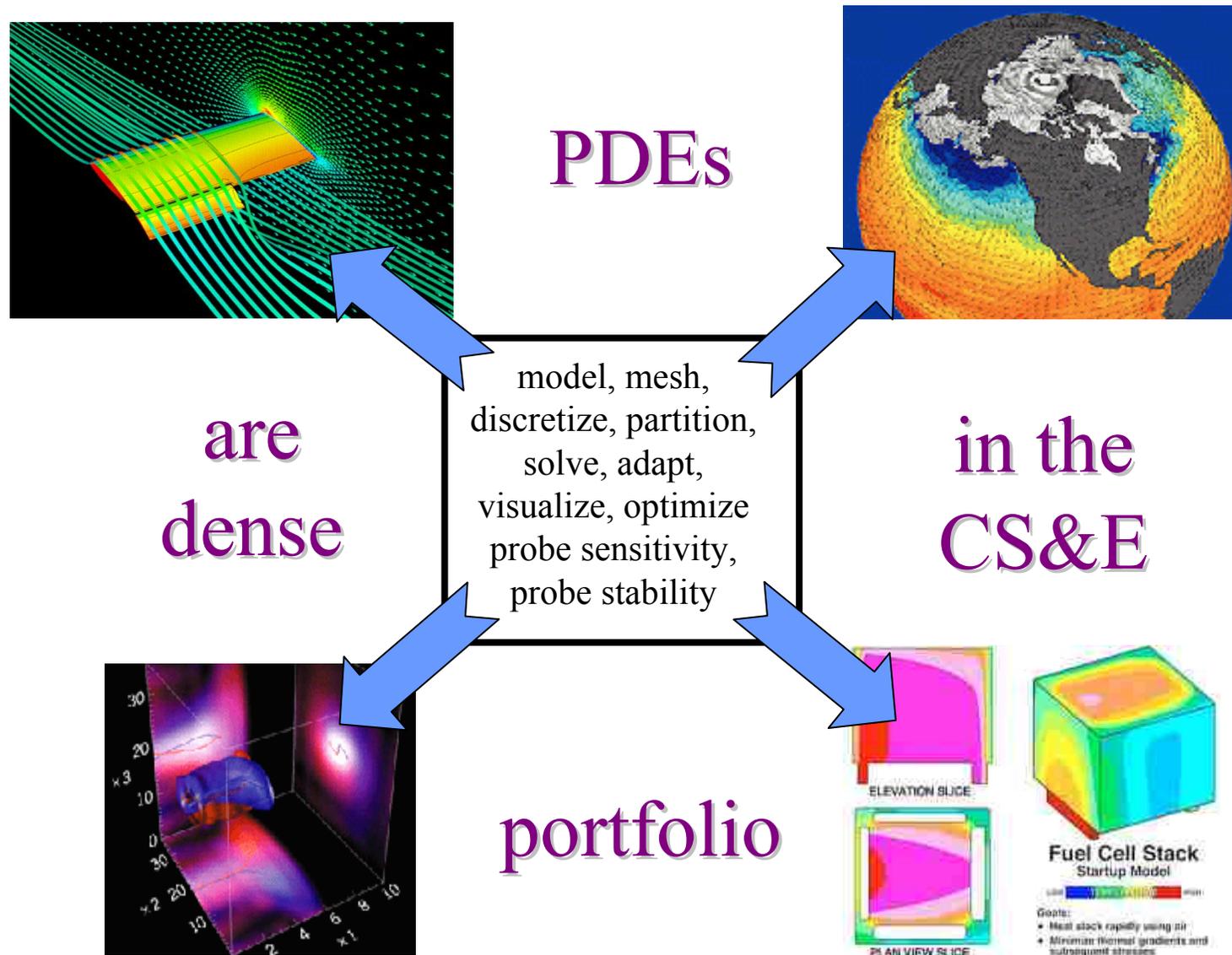
# TOPS Integrated Software Infrastructure Center (ISIC) spans 3 labs & 7 universities



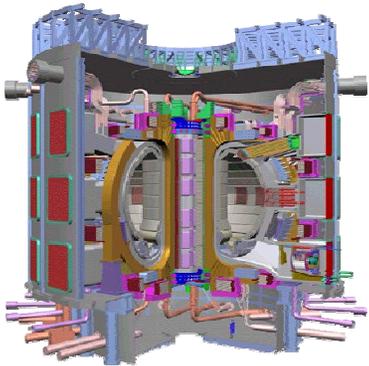
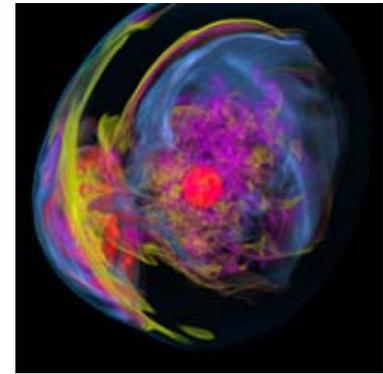
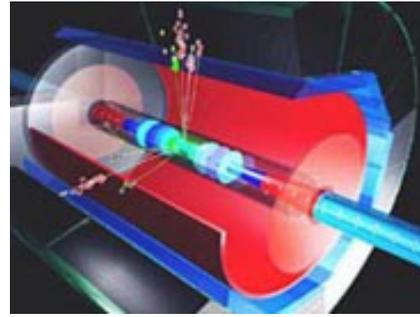
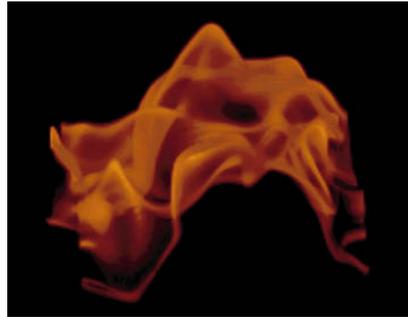
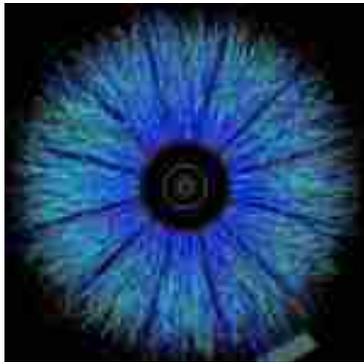
**Carnegie Mellon**



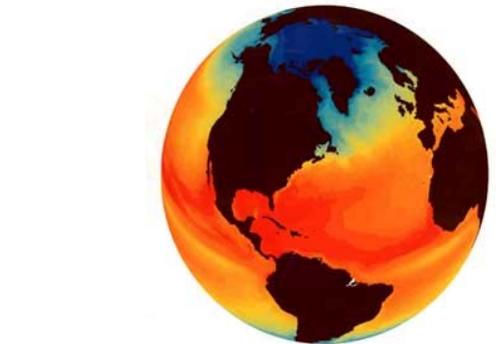
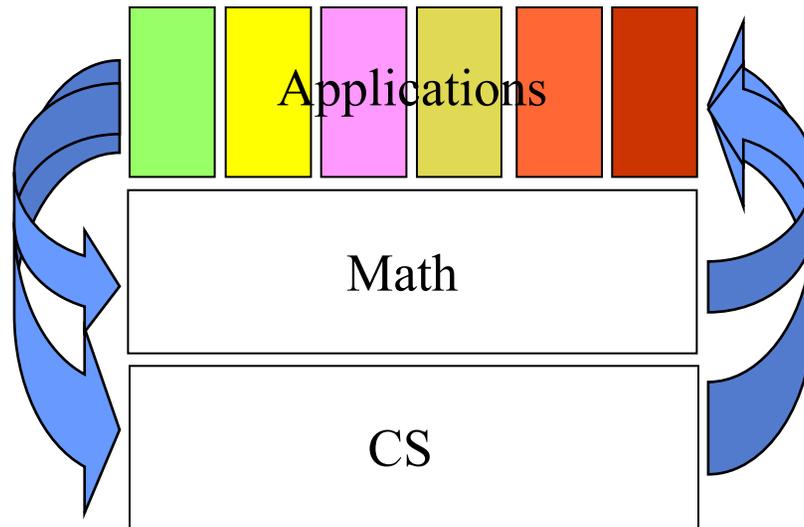
*“The partial differential equation entered theoretical physics as a handmaid, but has gradually become mistress.” – A. Einstein*



# SciDAC is not mainly about the math...



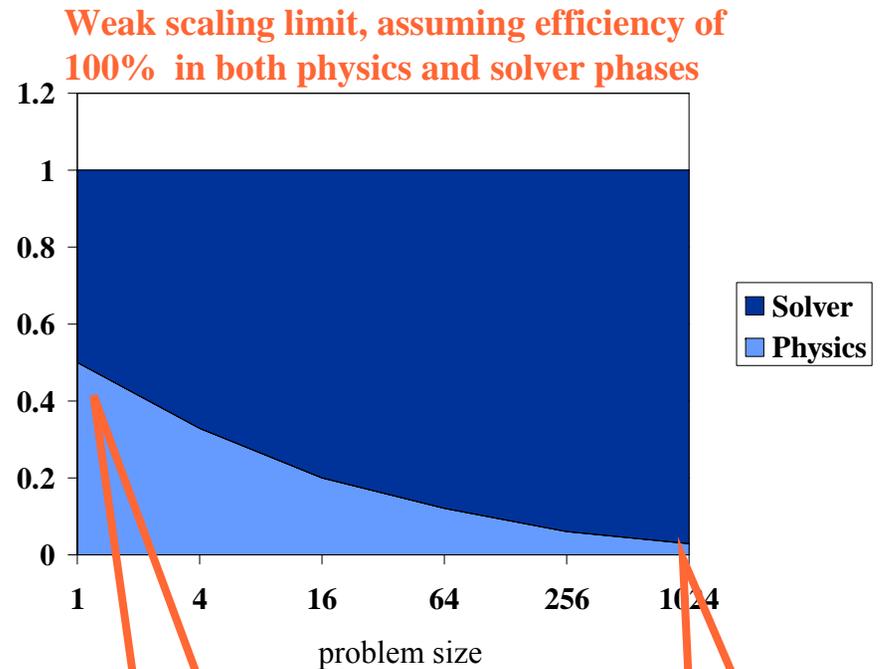
Applications  
drive



Enabling  
technologies  
respond

# ... It's *all* about the math

- **Given, for example:**
  - a “physics” phase that scales as  $O(N)$
  - a “solver” phase that scales as  $O(N^{3/2})$
  - computation is almost all solver after several doublings
- **Some application groups have not yet “felt” this curve in their gut**
  - BG/L will change this



Solver takes  
50% time  
on 64 procs

Solver takes  
97% time on  
64K procs

# Examples of codes that often profile at 90% solver time

- **Environment**

Porous media flow (*div-grad Darcy problems*)

- **Materials**

Quantum chemistry (*generalized eigenproblems*)

- **Fusion energy**

Magnetohydrodynamics (*Poisson problems*)

- **High energy and nuclear physics**

Quantum chromodynamics (*Dirac inversions*)

# Challenges for math software designers

- **The past challenge:**

**Increase functionality and capability for a small number of users who are expert in the domain of the software**

- **A present and future challenge:**

**Increase ease of use (for correctness *and* for efficiency) for a large number of users who are expert in something else**

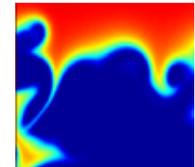
# Run-up to the era of simulation

(dates are symbolic)



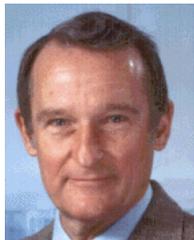
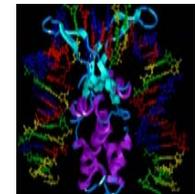
1686

scientific models



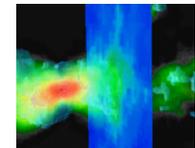
1947

numerical algorithms



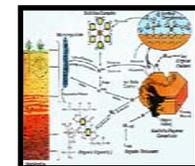
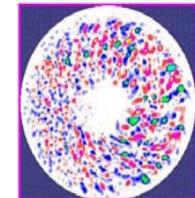
1976

computer architecture



1992

scientific software engineering



“Computational science is undergoing a phase transition.” – D. Hitchcock, DOE

# Design principle: multiple layers

- **Top layer (all users)**
  - Abstract interface featuring language of application domain, hiding details, with conservative parameter defaults
  - *Robustness, correctness, ease of use*
- **Middle layers (experienced users)**
  - Rich collection of state-of-the-art methods and data structures, exposed upon demand, highly configurable
  - *Capability, algorithmic efficiency, extensibility, composability, comprehensibility of performance and resource use*
- **Bottom layer (developers)**
  - Support for variety of execution environments
  - *Portability, implementation efficiency*

# What physicists want in math software

- **Develop code “without having to make bets”**
  - accomplish certain abstract mathematical tasks
  - stay agnostic about particular solution methods and codes
  - run everywhere: laptops (on travel), low-cost unmetered clusters (at work), and unique shared national resources
- **Ordered goals (need them all for production use)**
  - usability and robustness
  - portability
  - algorithmic efficiency (optimality) and implementation efficiency (within a processor and in parallel)
- **Algorithmic optimality and software stability**
  - scalable methods needed for multiscale problems
  - no “hand coding” for evanescent computing environments

# Foci for math ISICs

- **The past focus:**

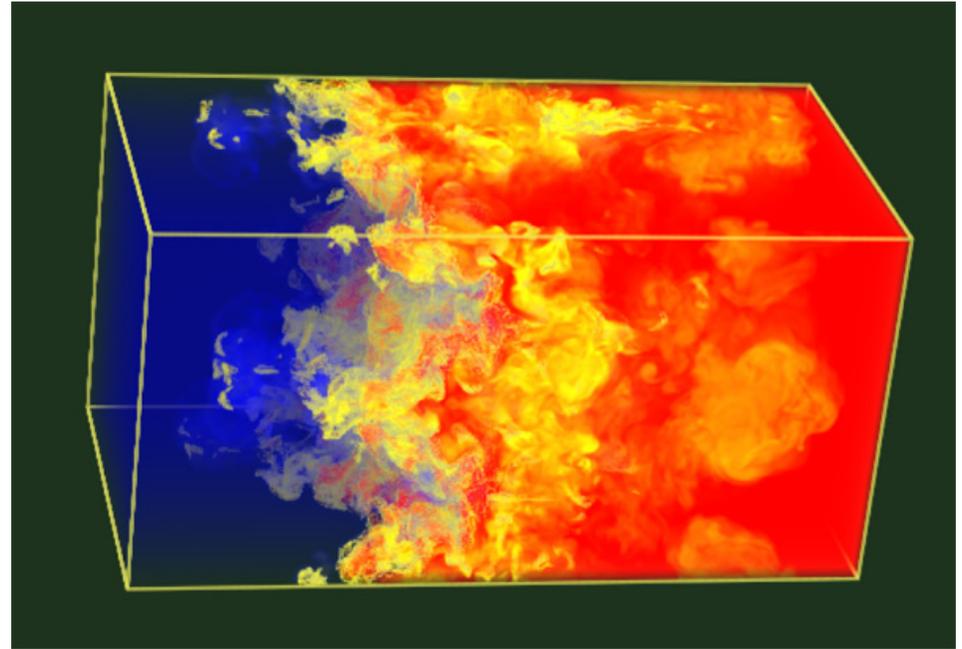
**Meet the understood and expressed needs of application groups, where they are, *typically across a conventional interface***

- **A present and future focus:**

**Lure application groups into new explorations, boldly going where no one has gone before, *typically blurring conventional interfaces***

# Multiscale apps demand scalable solvers

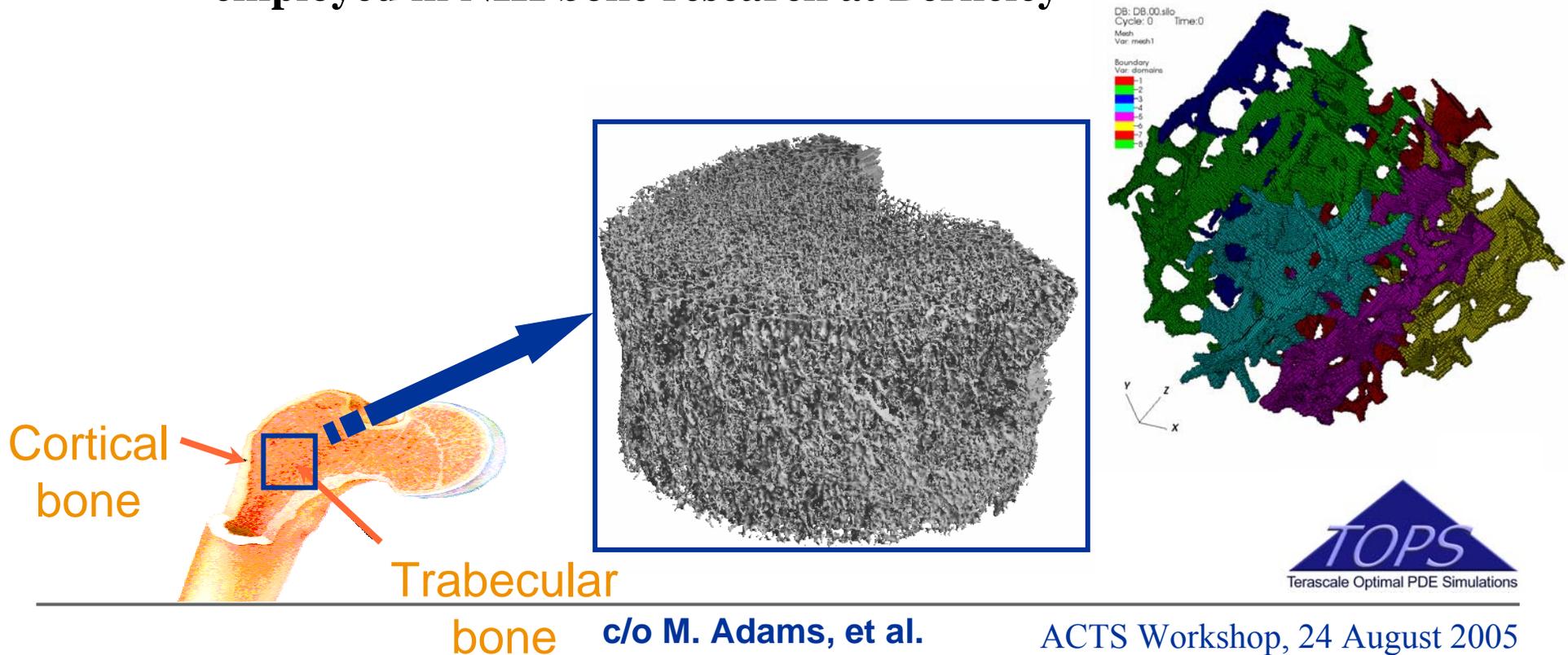
- **Multiple spatial scales**
  - interfaces, fronts, layers
  - thin relative to domain size,  $\delta \ll L$
- **Multiple temporal scales**
  - fast waves
  - small transit times relative to convection or diffusion,  $\tau \ll T$
- Analyst must *isolate dynamics of interest* and *model the rest* in a system that can be discretized over more modest range of scales
- Often involves filtering of high frequency modes, quasi-equilibrium assumptions, etc.
- May lead to infinitely “stiff” subsystem requiring implicit treatment
- Resulting implicit subsystem may be very ill-conditioned



Richtmyer-Meshkov instability, c/o A. Mirin, LLNL

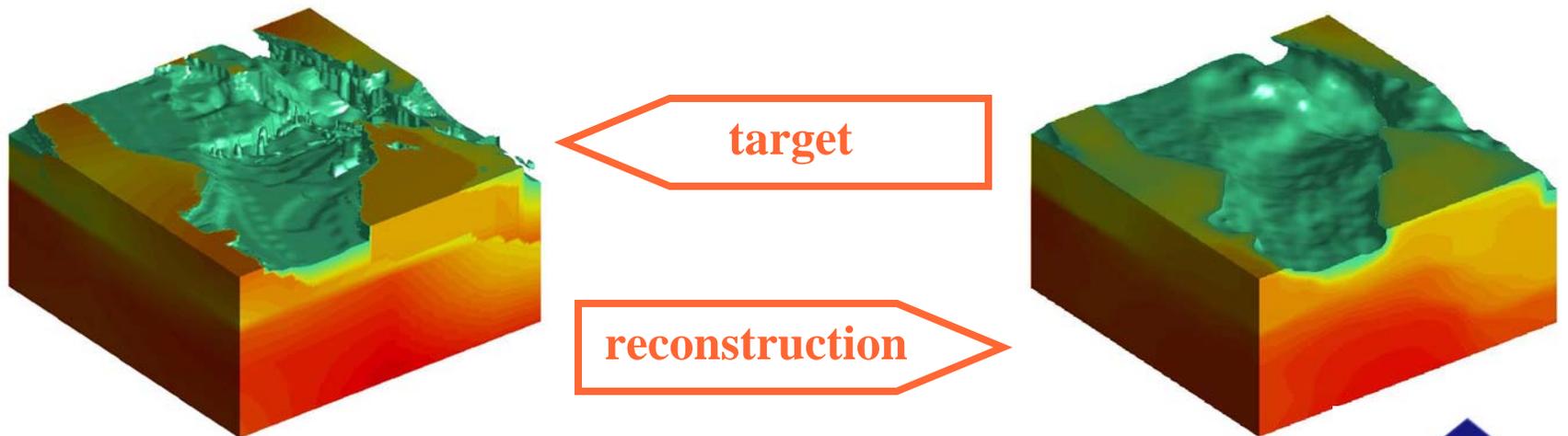
# 2004 Gordon Bell “special” prize

- 2004 Bell Prize in “special category” went to an implicit, unstructured grid bone mechanics simulation
  - 0.5 Tflop/s sustained on 4 thousand procs of ASCI White
  - 0.5 billion degrees of freedom
  - large-deformation analysis
  - employed in NIH bone research at Berkeley



# 2003 Gordon Bell “special” prize

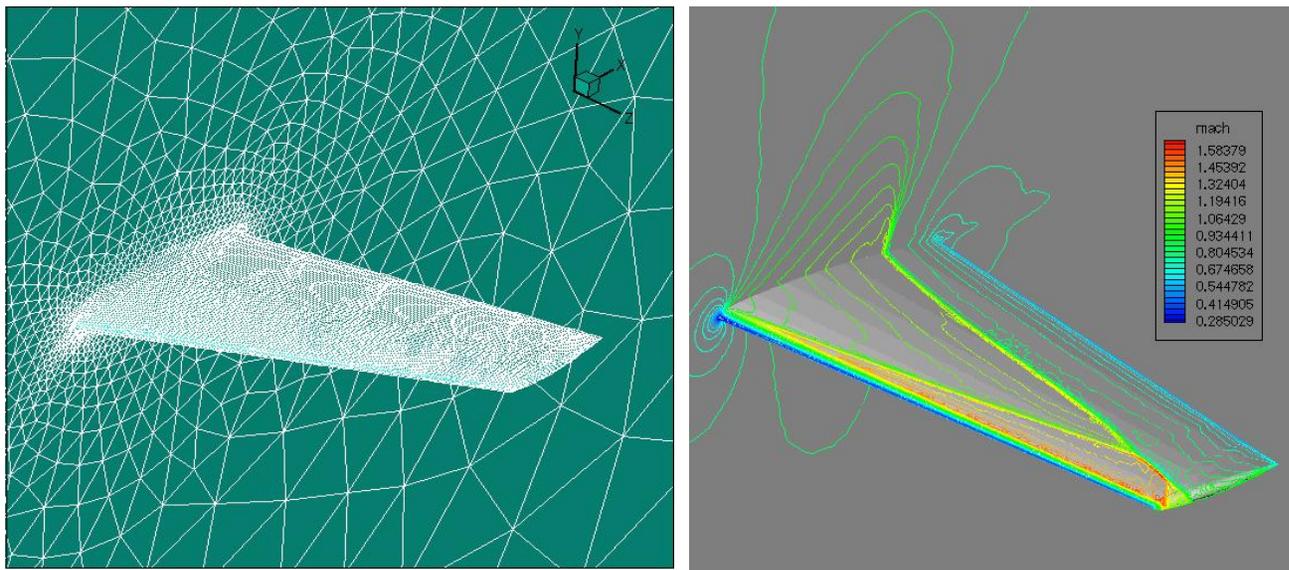
- 2003 Bell Prize in “special category” went to unstructured grid geological parameter estimation problem
  - 1 Tflop/s sustained on 2 thousand processors of HP’s “Lemieux
  - each explicit forward PDE solve: 17 million degrees of freedom
  - seismic inverse problem: 70 billion degrees of freedom
  - employed in NSF seismic research at CMU



# 1999 Gordon Bell “special” prize

- 1999 Bell Prize in “special category” went to implicit, unstructured grid aerodynamics problems
  - 0.23 Tflop/s sustained on 3 thousand processors of ASCI Red
  - 11 million degrees of freedom
  - incompressible and compressible Euler flow
  - Employed in NASA analysis/design missions

Transonic “Lambda” Shock, Mach contours on surfaces



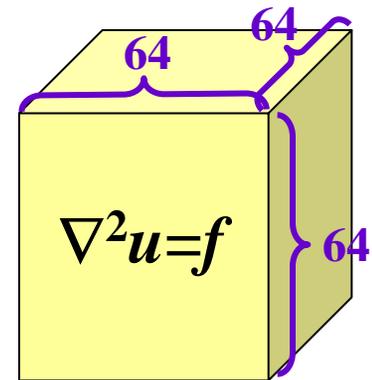
# Observations on Bell Prizes re: solvers

- **A single solver toolkit (PETSc) employed with three different types of functionality**
  - **as programmer interface for MPI-implemented distributed data structures in explicit time integration**
  - **as linear Krylov package to accelerate expert user-provided algebraic MG**
  - **as full nonlinear pseudo-transient Newton-Krylov-Schwarz solver**
- **We will see a fourth use later in a PDE-constrained optimization application**

# Solvers evolve underneath “ $Ax=b$ ”

- Advances in algorithmic efficiency rival advances in hardware architecture
- Consider Poisson’s equation on a cube of size  $N=n^3$

<i>Year</i>	<i>Method</i>	<i>Reference</i>	<i>Storage</i>	<i>Flops</i>
1947	GE (banded)	Von Neumann & Goldstine	$n^5$	$n^7$
1950	Optimal SOR	Young	$n^3$	$n^4 \log n$
1971	CG	Reid	$n^3$	$n^{3.5} \log n$
1984	Full MG	Brandt	$n^3$	$n^3$

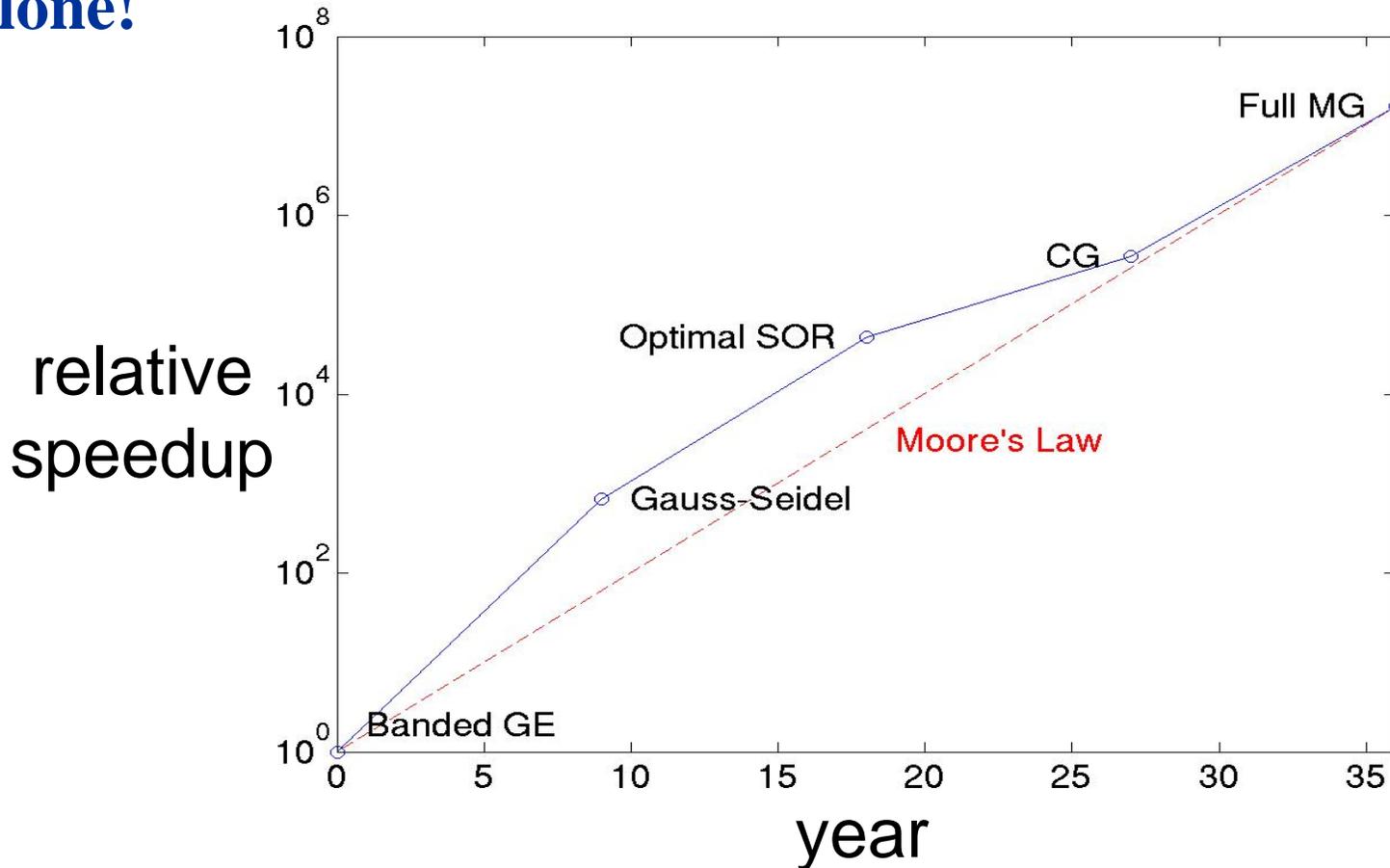


- If  $n=64$ , this implies an overall reduction in flops of ~16 million \*

\*Six months is reduced to 1 second

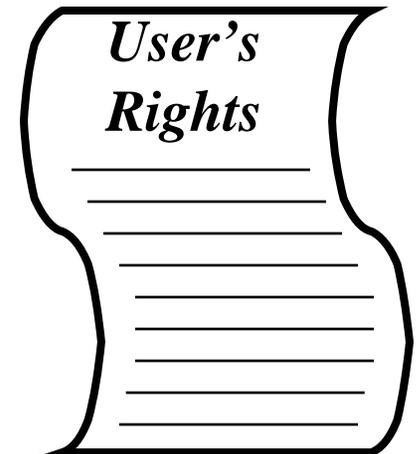
# Solver algorithms and Moore's Law

- This advance took place over a span of about 36 years, or 24 doubling times for Moore's Law
- $2^{24} \approx 16$  million  $\Rightarrow$  the same as the factor from algorithms alone!



# TOPS has a dream that users will...

- **Understand range of algorithmic options w/tradeoffs**  
*e.g., memory vs. time, comp. vs. comm., inner iteration work vs. outer*
- **Try all reasonable options “easily”**  
without recoding or extensive recompilation
- **Know how their solvers are performing**  
with access to detailed profiling information
- **Intelligently drive solver research**  
*e.g., publish joint papers with algorithm researchers*
- **Simulate *truly new physics* free from solver limits**  
*e.g., finer meshes, complex coupling, full nonlinearity*



# What we believe about *apps*

- **Solution of a system of PDEs is rarely a goal in itself**

- Actual goal is characterization of a response surface or a design or control strategy
- Solving the PDE is just one forward map in this process
- Together with analysis, sensitivities and stability are often desired

⇒ **Software tools for PDE solution should also support related follow-on desires**

- **No general purpose PDE solver can anticipate all needs**

- Why we have *national laboratories*, not *numerical libraries* for PDEs today
- A PDE solver improves with user interaction
- Pace of algorithmic development is very rapid

⇒ **Extensibility is important**

# What we believe about *users*

- **Solvers are used by people of varying numerical backgrounds**

- Some expect MATLAB-like defaults
- Others want to control everything, e.g., even varying the type of smoother and number of smoothings on different levels of a multigrid algorithm

⇒ **Multilayered software design is important**

- **Users' demand for resolution is virtually insatiable**

- Relieving resolution requirements with modeling (e.g., turbulence closures, homogenization) only defers the demand for resolution to the next level
- Validating such models requires high resolution

⇒ **Processor scalability and algorithmic scalability (optimality) are critical**



# What we believe about *legacy code*

- **Porting to a scalable framework does not mean starting from scratch**
  - **High-value physics routines in original languages can be substantially preserved**
  - **Partitioning, reordering and mapping onto distributed data structures (that solver may provide) adds code but little runtime**
- ⇒ **Solver distributions should include code samples exemplifying “separation of concerns”**
- **Legacy solvers may be limiting resolution, accuracy, and generality of modeling overall**
  - **Replacing the solver may “solve” several other issues**
  - **However, pieces of the legacy solver may have value as part of a preconditioner**
- ⇒ **Solver toolkits should include “shells” for callbacks to high-value legacy routines**

# What we believe about *solvers*

- **Solvers are employed as part of a larger code**
  - Solver library is not the only library to be linked
  - Solvers may be called in multiple, nested places
  - Solvers typically make callbacks
  - Solvers should be swappable

⇒ **Solver threads must not interfere with other component threads, including other active instances of themselves**

- **Solvers are employed in many ways over the life cycle of an applications code**
  - During development and upgrading, robustness (of the solver) and verbose diagnostics are important
  - During production, solvers are streamlined for performance

⇒ **Tunability is important**



# What we believe about *numerical software*

- A continuous operator may appear in a discrete code in many different instances
  - Optimal algorithms are nested iterative and hierarchical
  - Majority of progress towards desired *highly resolved, high fidelity result* occurs through *cost-effective low resolution, low fidelity parallel-efficient stages*
- ⇒ Operator and grid-function abstractions must let the user glide up and down the representation hierarchy
- Hardware changes many times over the life cycle of a software package
  - Processors, memory, and networks evolve annually
  - Machines are replaced every 3-5 years at major DOE centers
  - Codes persist for decades
- ⇒ Portability is critical

# A central concept: solver toolchain

- From *solutions* to *sensitivity*, *stability*, *optimization*
- Nested modules
- Leveraging of coding work in the distributed data structures
- Hiding of communication and performance-oriented details so users deal with mathematical objects throughout



# Solver software toolchain

- Design and implementation of “solvers”

- Linear solvers

$$Ax = b$$

- Eigensolvers

$$Ax = \lambda Bx$$

- Nonlinear solvers (w/ sens. anal.)

$$F(x, p) = 0$$

- Time integrators (w/ sens. anal.)

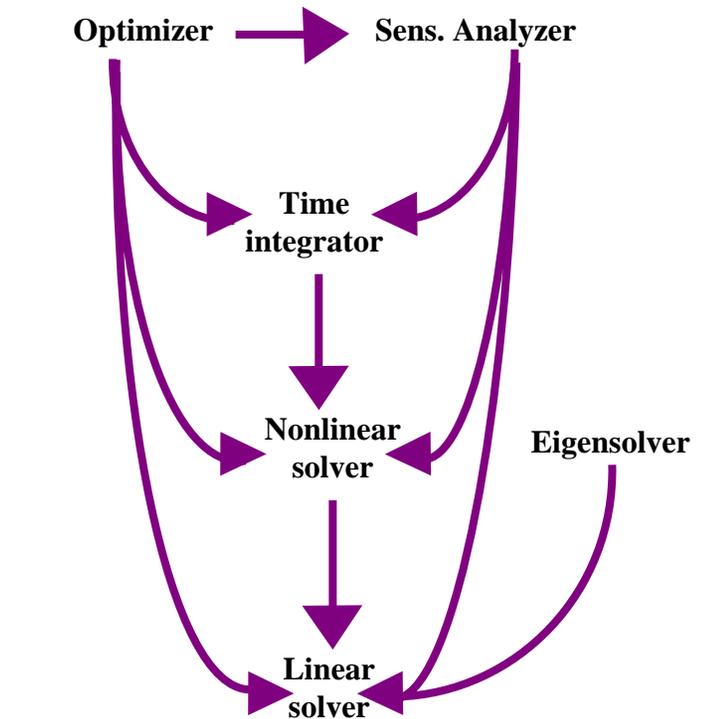
$$f(\dot{x}, x, t, p) = 0$$

- Optimizers

$$\min_u \phi(x, u) \text{ s.t. } F(x, u) = 0, u \geq 0$$

- Software integration

- Performance optimization



→ Indicates dependence



# SciDAC solver collaboration examples

- **Meeting physicists *at* a well-defined traditional interface**
  - **Magnetic fusion energy – swapping in new linear solvers**
- **Collaborating with physicists *across* traditional interfaces**
  - **Accelerator design – multidisciplinary design optimization**
  - **Quantum chromodynamics – research prototyping of new algorithm**

# Illustrations from computational MHD

- **M3D code (Princeton)**
  - multigrid replaces block Jacobi/ASM preconditioner for optimality
  - new algorithm callable across  $Ax=b$  interface
- **NIMROD code (General Atomics)**
  - direct elimination replaces PCG solver for robustness
  - scalable implementation of old algorithm for  $Ax=b$

The fusion community may use more cycles on unclassified U.S. DOE computers than any other (e.g., 32% of all cycles at NERSC in 2003). Well over 90% of these cycles are spent solving *linear systems* in M3D and NIMROD, which are prime U.S. code contributions to the designing of ITER.

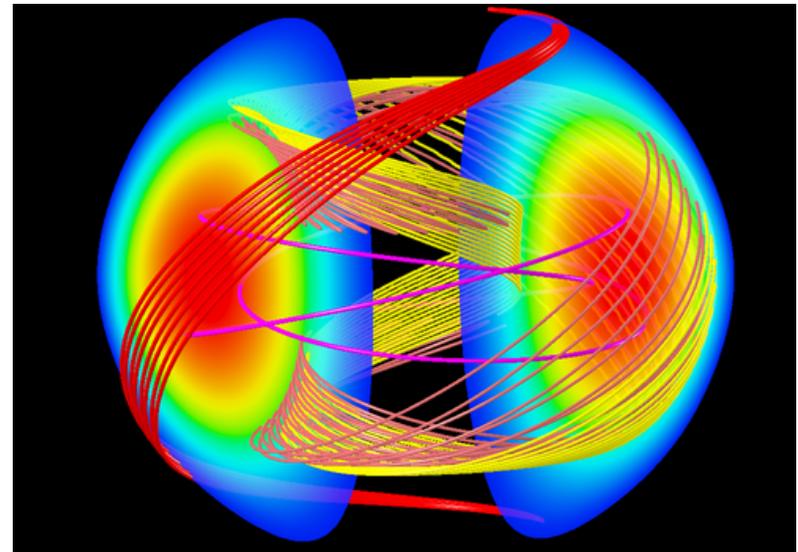
# NIMROD: direct elim. for robustness

- **NIMROD code**

- high-order finite elements
- complex, nonsymmetric linear systems with 10K-100K unknowns (**>90% exe. time**)

- **TOPS collaboration**

- replacement of diagonally scaled Krylov with SuperLU, a supernodal parallel sparse direct solver
- 2D tests run  $100 \times$  faster; 3D production runs are  $4\text{-}5 \times$  faster



*c/o D. Schnack, et al.*

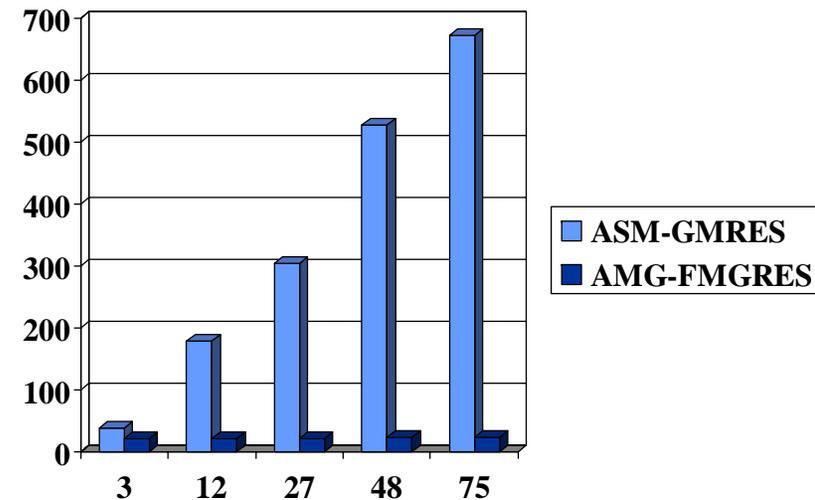
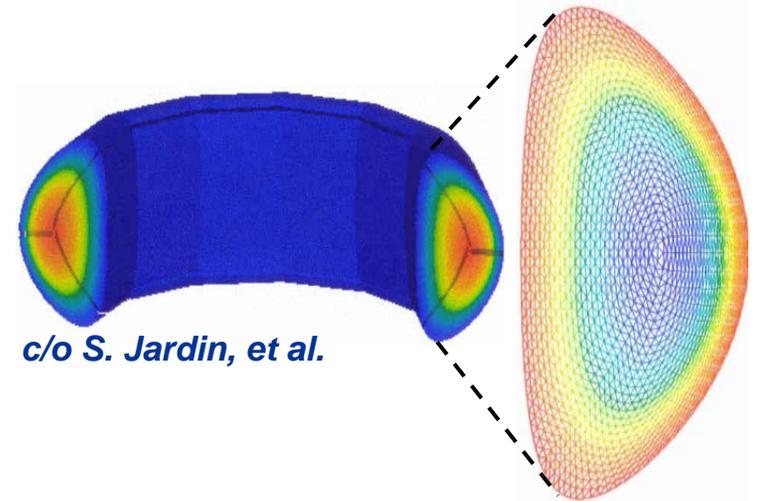
# M3D: multigrid for optimality

- **M3D code**

- unstructured mesh, hybrid FE/FD discretization with C0 elements
- Sequence of real scalar systems (**>90% exe. time**)

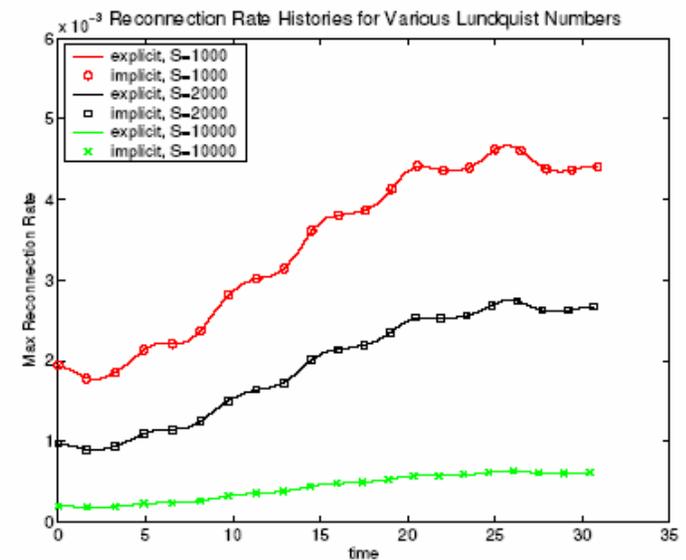
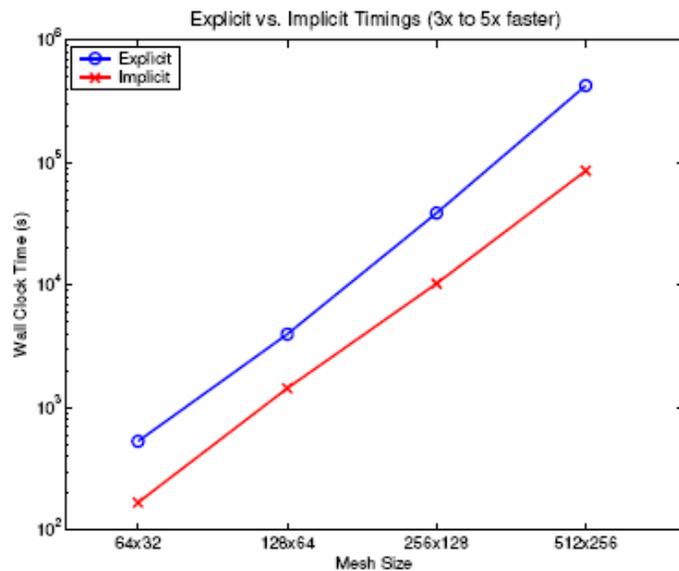
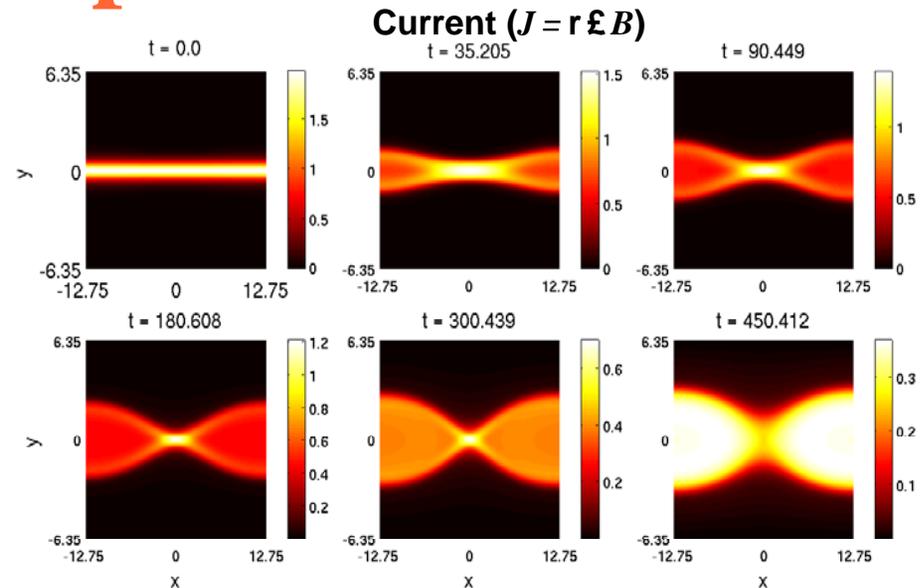
- **TOPS collaboration**

- replacement of additive Schwarz (ASM) preconditioner with algebraic multigrid (AMG)
- achieved mesh-independent convergence rate
- **4-5 × improvement in execution time**



# AMRMHD: implicit solver

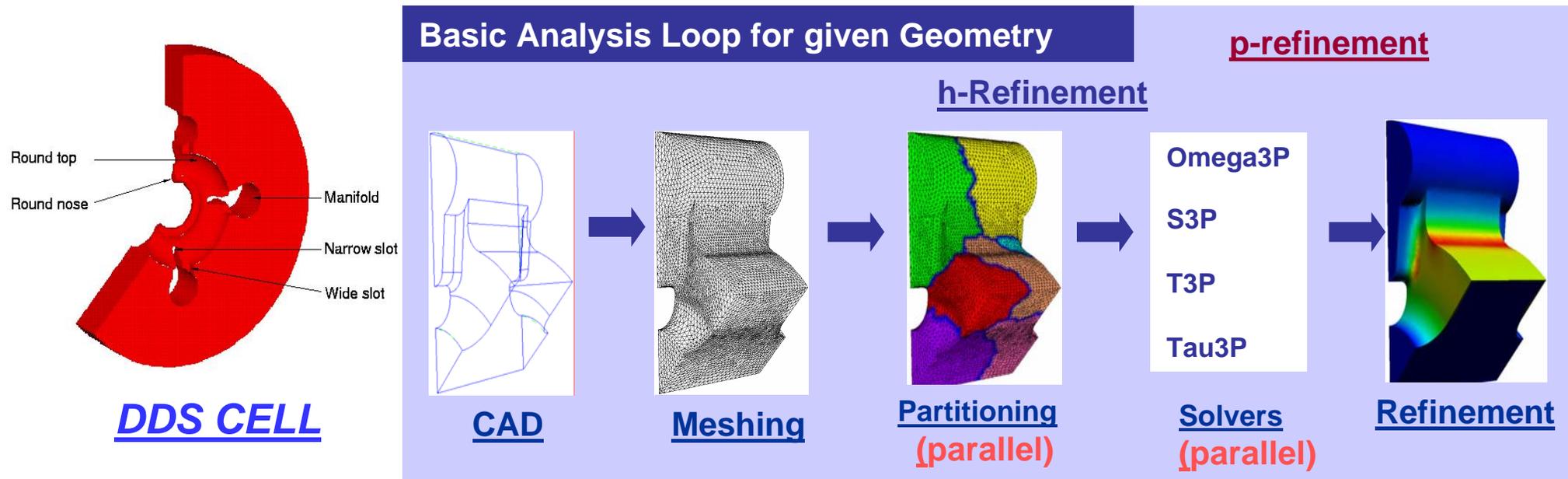
- *Magnetic reconnection: the breaking and reconnecting of oppositely directed magnetic field lines in a plasma*
- Large-scale current instabilities may replace hot plasma core with cool plasma, halting the fusion process



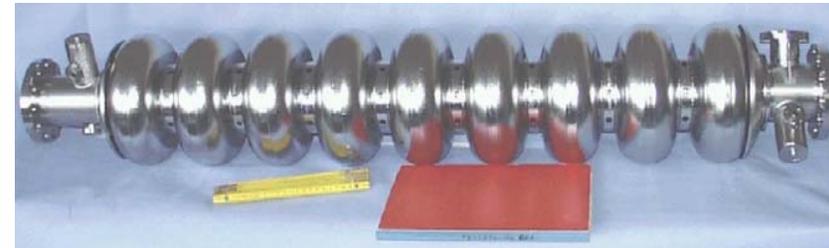
J. Brin et al., "Geospace Environmental Modeling (GEM) magnetic reconnection challenge," *J. Geophys. Res.* 106 (2001) 3715-3719.

# Shape optimization for accelerators

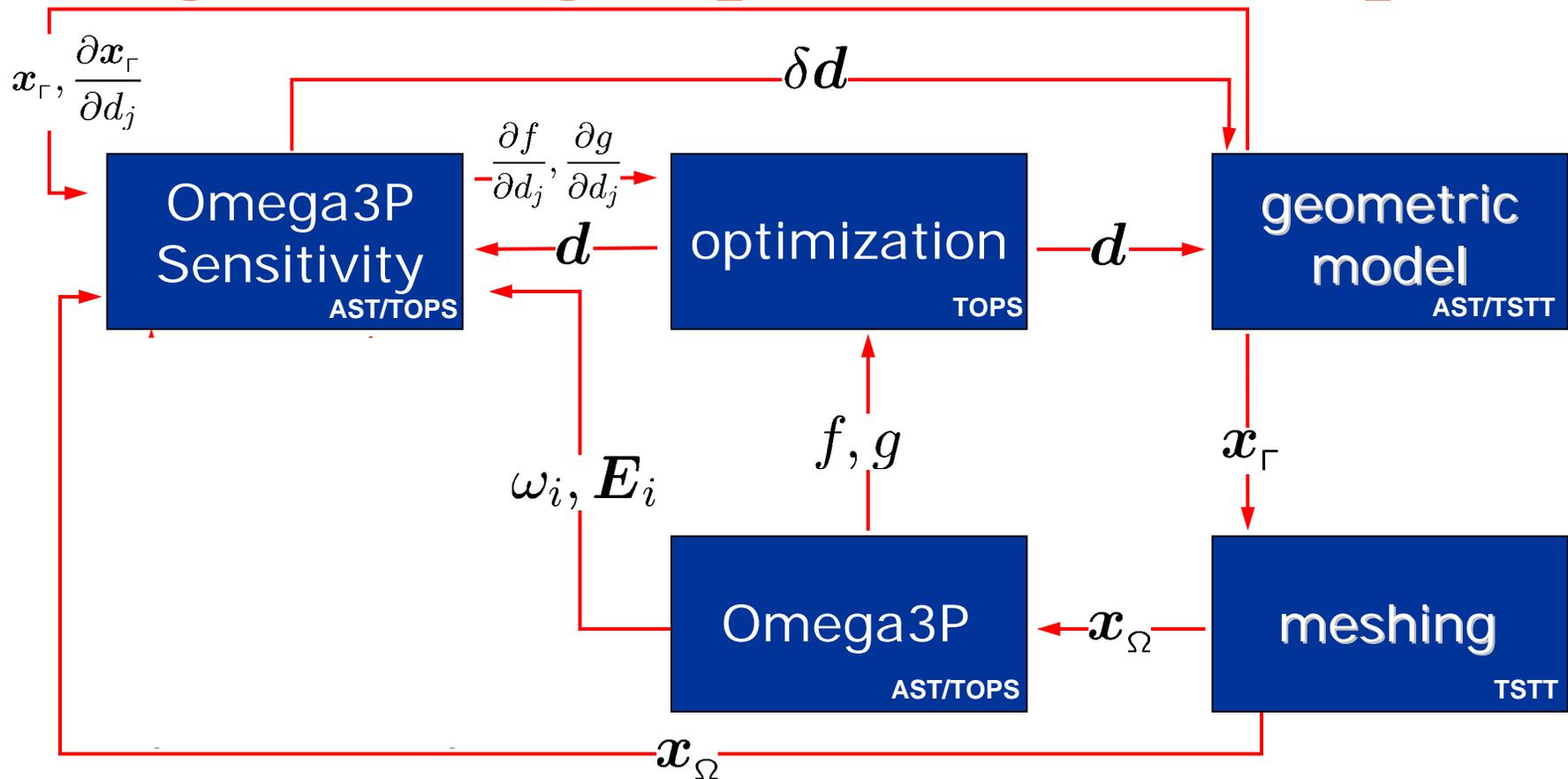
- Numerical modeling has replaced trial and error prototyping approach
- SciDAC adds advances that increase fidelity, speed, and accuracy:



- Next generation accelerators have complex cavities that require shape optimization for improved performance and reduced cost
- AST/TSTT/TOPS are collaborating to develop an automated capability to accelerate this otherwise manual process



# Omega3P design optimization components



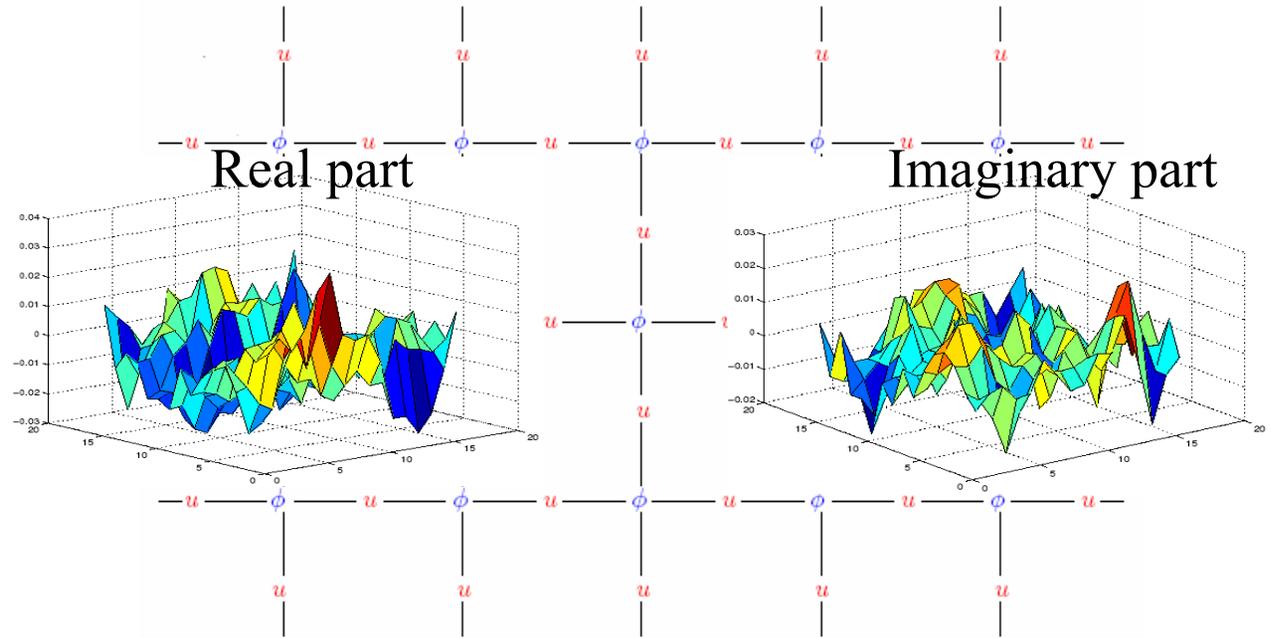
$d$  : design vector  
 $x_\Gamma$  : surface grid  
 $x_\Omega$  : volume grid  
 $f$  : objective  
 $g$  : constraint

$\omega_i, E_i$  : eigenpairs  
 $\delta d$  : perturbed design

# Multigrid for Lattice Gauge QCD

Fermion field:  $\phi(x,y)=(f_1,f_2)$

Gauge field:  $u(x,y)=e^{i\theta}$



Wilson-Fermion operator:

$$Hf = \sigma_3 \left[ \sum_{m=1}^2 \frac{1}{2} \sigma_m (\nabla_m^+ + \nabla_m^-) f + \frac{1}{2} \sum_{m=1}^2 (-\nabla_m^+ + \nabla_m^-) f + \rho f \right]$$

Difference operators:

$$(\nabla_m^+ f)(\vec{x}, s) = u(\vec{x}, m) f(\vec{x} + \hat{m}, s) - f(\vec{x}, s)$$

$$(\nabla_m^- f)(\vec{x}, s) = f(\vec{x}, s) - u(\vec{x} - \hat{m}, m)^* f(\vec{x} - \hat{m}, s)$$

Pauli spin matrices:

$$\sigma_1 := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \sigma_2 := \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \sigma_3 := -i\sigma_1\sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

# Algebraic Multigrid for QCD

iterations / per-iteration-reduction / condition number

## Diagonally scaled CG

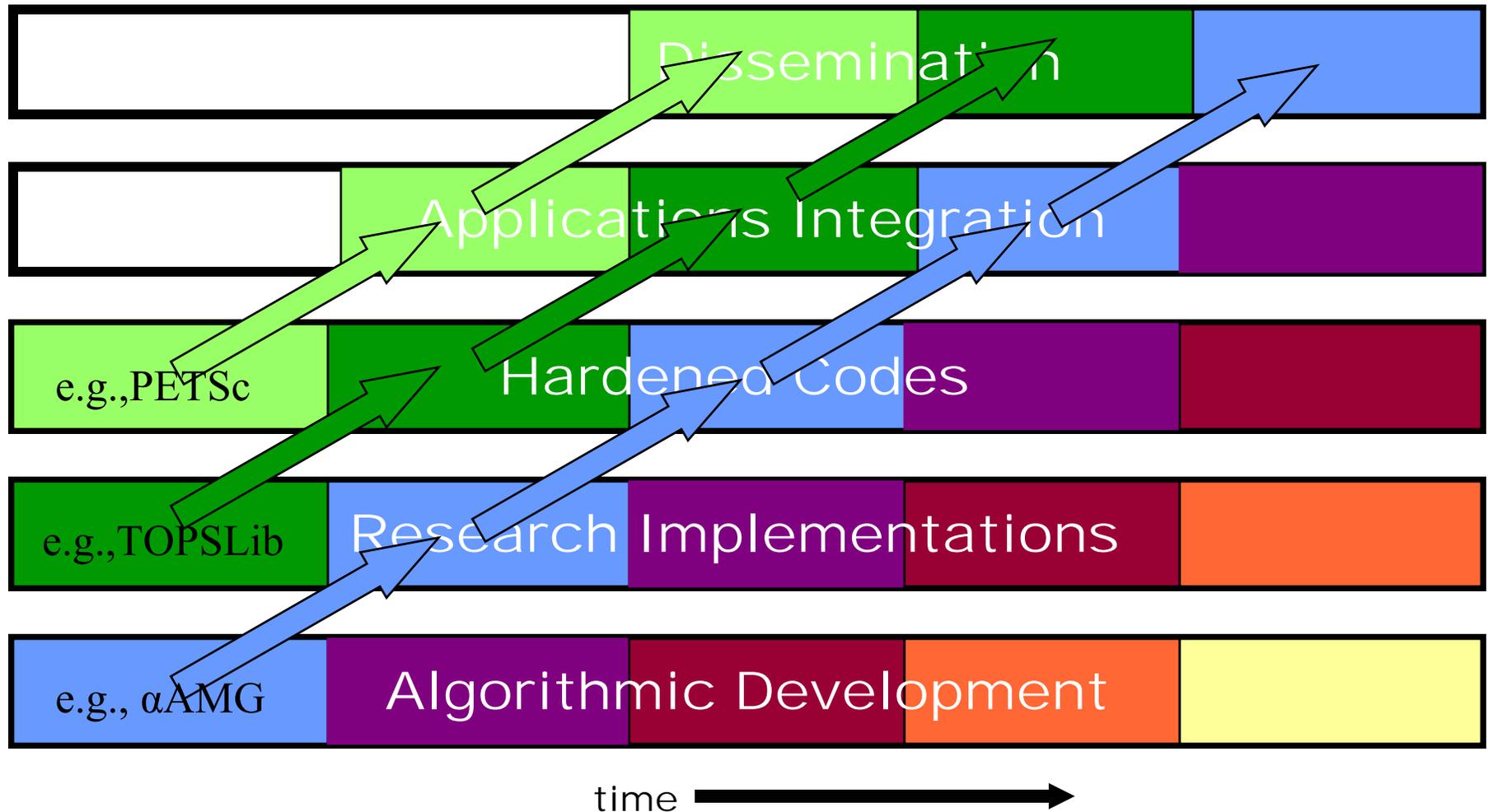
$N / \sigma$	0	.45	.65
32	102 / .77 / 416	271 / .90 / 822	185 / .87 / 267
64	207 / .88 / 166	316 / .91 / 750	196 / .87 / 290
128	415 / .94 / 664	328 / .92 / 786	205 / .88 / 296
256	817 / .96 / 2650	341 / .92 / 846	205 / .88 / 298

## Adaptive Smoothed Aggregation AMG CG

$N / \sigma$	0	.45	.65
32	11 / .11 / 1.7	24 / .38 / 6.54	20 / .29 / 3.85
64	11 / .12 / 1.7	25 / .38 / 6.30	20 / .29 / 3.82
128	13 / .16 / 3.1	26 / .40 / 7.39	19 / .29 / 3.88
256	15 / .20 / 3.44	27 / .41 / 7.35	22 / .34 / 5.22

# Abstract Gantt chart for solver development

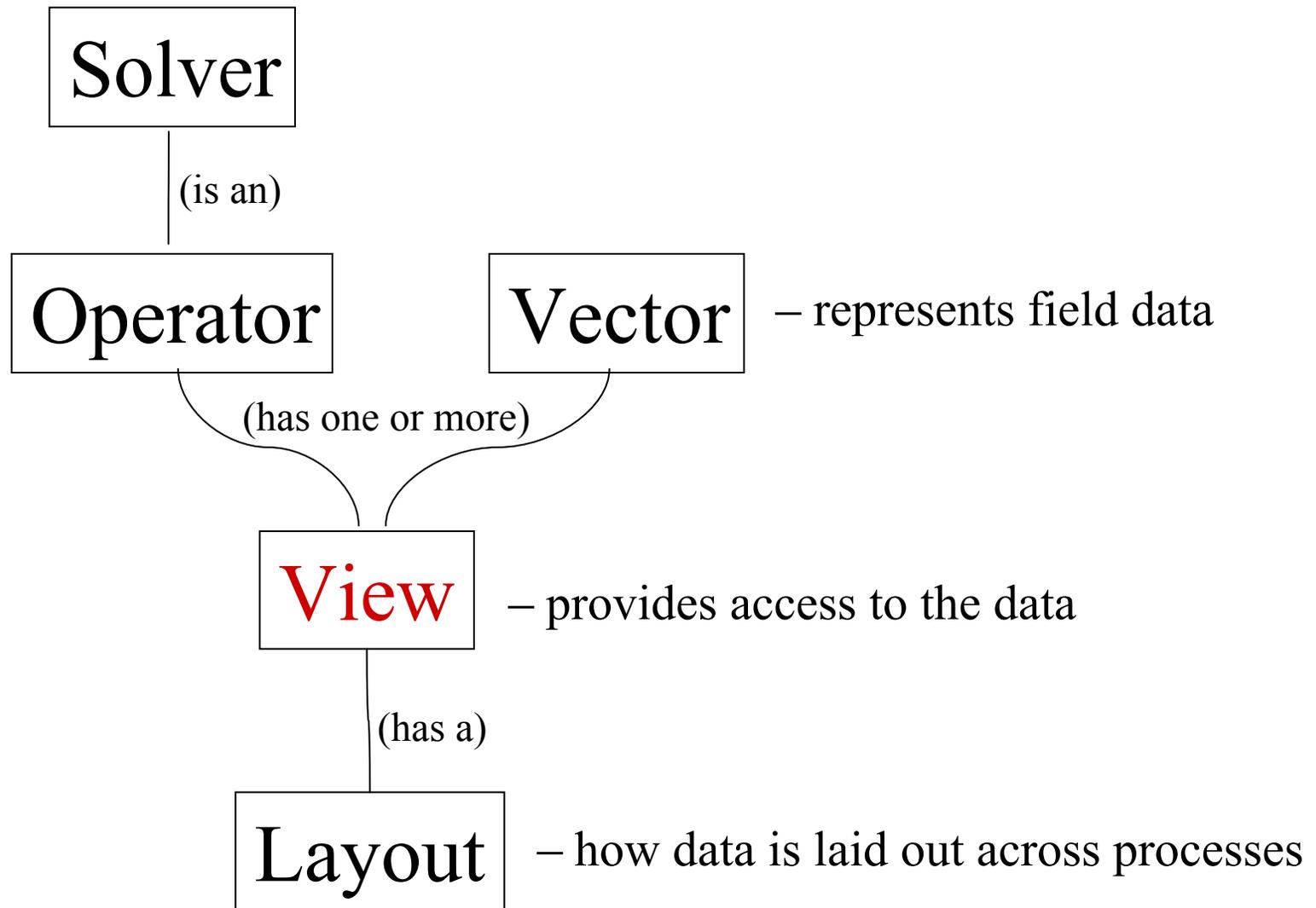
Each color module represents an algorithmic research idea on its way to becoming part of a supported community software tool. At any moment (vertical time slice), TOPS has work underway at multiple levels. While some codes are in applications already, they are being improved in functionality and performance as part of the TOPS research agenda.



# TOPS' linear solver interface informed by earlier interfaces

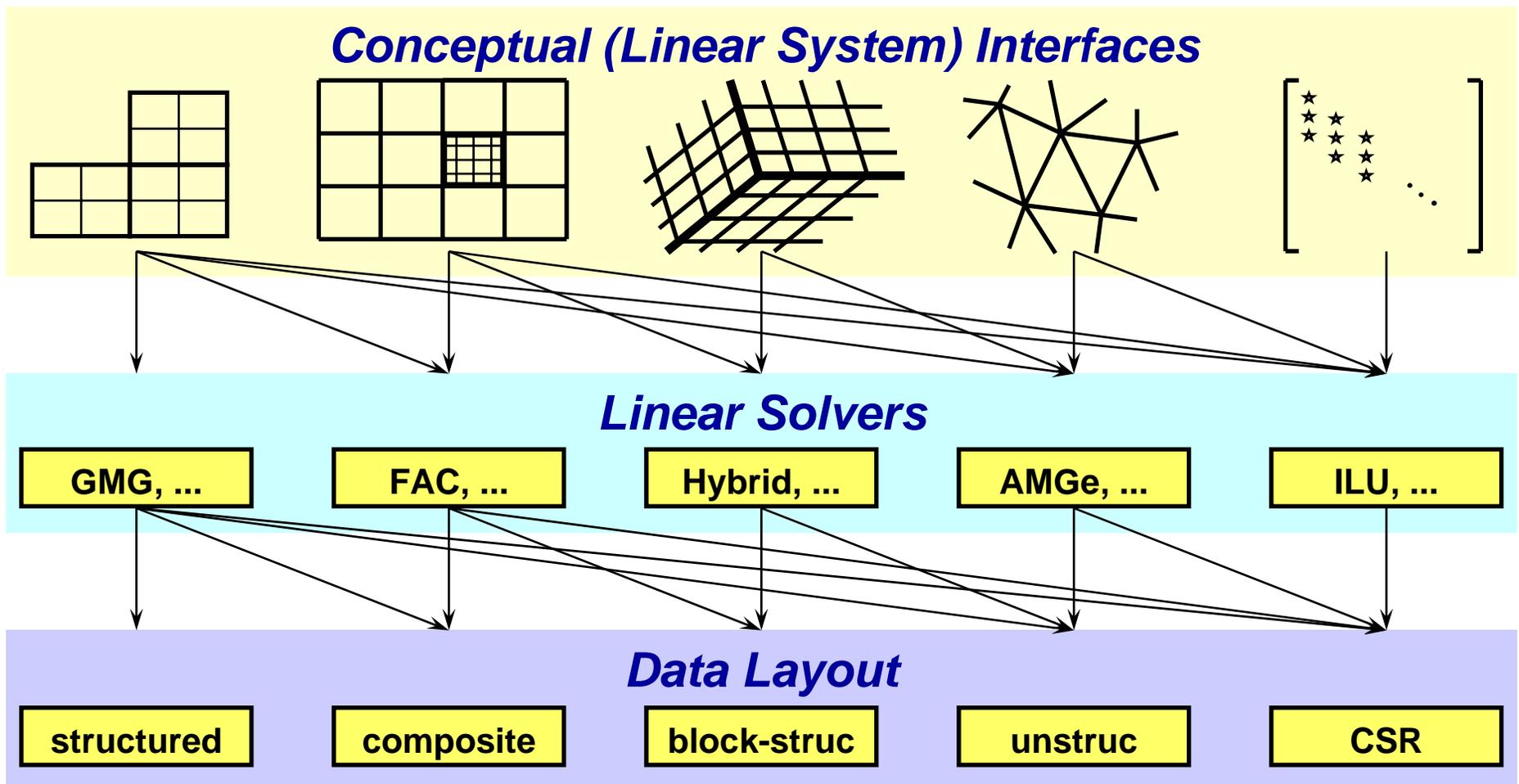
- **Some progenitors**
  - **FEI (finite element interface) / C++**
    - ◆ developed at SNL
  - **ESI (equation solver interface) / C++**
    - ◆ multi-lab effort
  - ***hypre* (conceptual interfaces) / C**
    - ◆ developed at LLNL
  - **PETSc / C**
    - ◆ developed at ANL
- **We support all of these in *some* of our packages, and will continue to do so**

# Object model concepts



# View allows users to access values in the “language of the application”

- Handles any data communication transparently
- Same idea as *hypr*'s conceptual interfaces



# Views differ primarily in the way they “set” and “get” data

- **Classical Linear Algebra View** – indices are scalars that represent locations in  $R^n$   
`array<double> getValues(array<int> indices);`
- **Structured Grid View** – indices are 3D triples that describe “boxes of data” (think 3D Fortran arrays)  
`array<double> getValues(<int,3> ilower, <int,3> iupper);`
- **Views / Layouts**
  - **Rn** – “classical linear algebra” access
  - **S** – single structured grid
  - **Fe** – finite element interface
  - **Ss** – semi-structured grids (structured grid “parts” with additional arbitrary connections)
  - ...

# What's coming in solvers

- **Greater interface standardization**
- **Greater solver interoperability**
- **Better integration upwards**  
w/ meshing and discretization systems
- **Better integration downwards**  
w/ performance monitoring and engineering systems
- **Better algorithms!**

# Tuning for performance

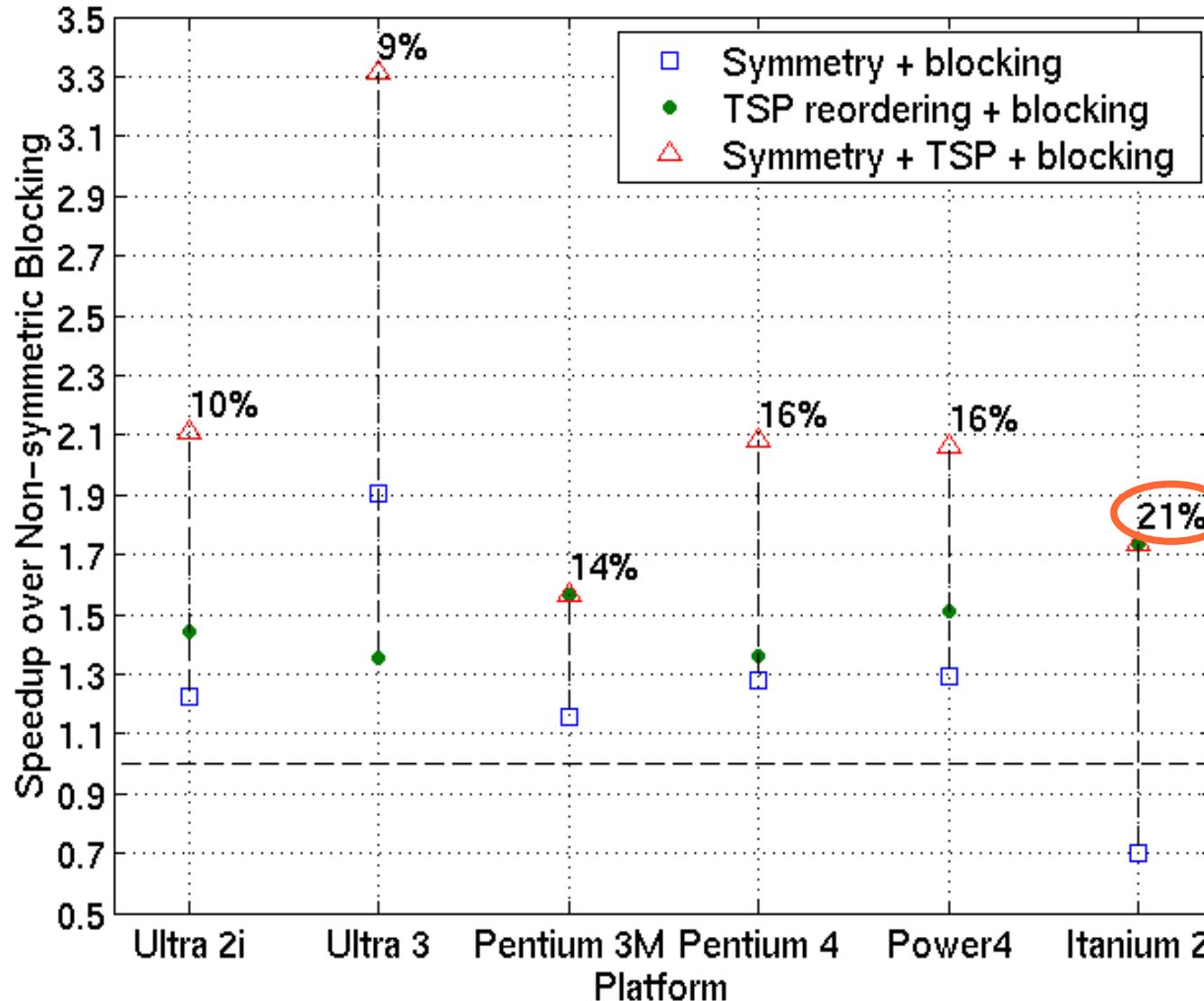
- **After functionality, efficiency is essential**
- **Combinatorially many implementations exist**
  - produce same results at very different rates and resource costs
  - little useful theory, few useful models to guide choices
- **We mention efforts aimed at**
  - implementation efficiency, in the real world of multilevel memory hierarchies
  - algorithm selection efficiency in the real world of and unsymmetric, unstructured, nonlinear problems

# Implementation efficiency

- **Ordering and blocking critical to floating point code performance**
  - **due to two-order-of-magnitude mismatch between FPU rate and memory BW**
  - **arithmetically neutral code transformations can lead to factors of up to  $\sim 7$  in performance improvement in common linear algebraic kernels**
- **Well exploited for dense kernels in, e.g., ATLAS**
- **Successes for sparse kernels now available in, e.g., OSKI**

# Combined gains for SpMV

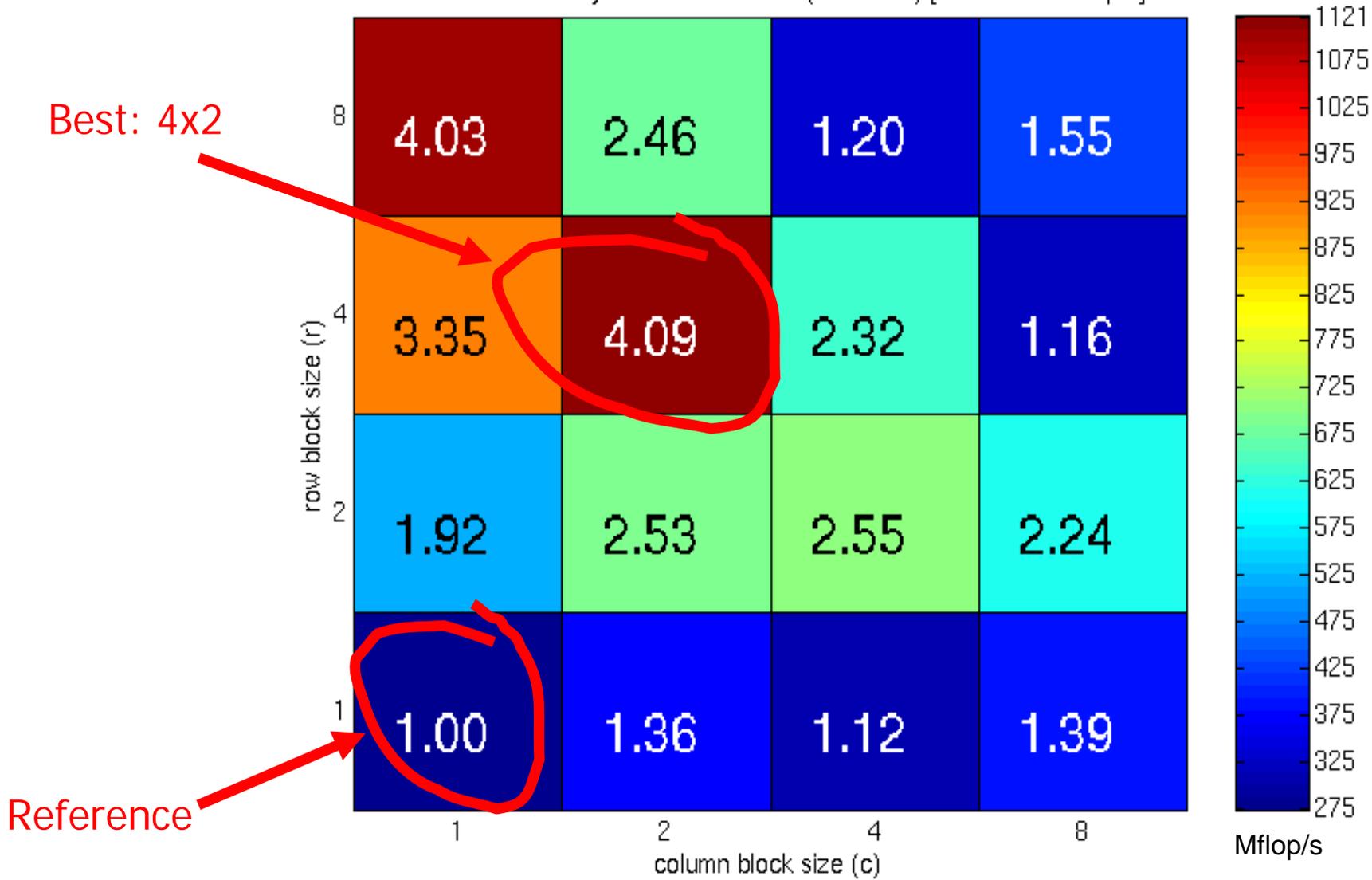
SpMV Speedups: SLAC Matrix (Omega3P)



These labels represent best percentage of theoretical peak flop/s achieved on each processor

# Speedups on Itanium 2 from blocking

Matrix #02-raefsky3.rua on Itanium 2 (900 MHz) [Ref=274.3 Mflop/s]

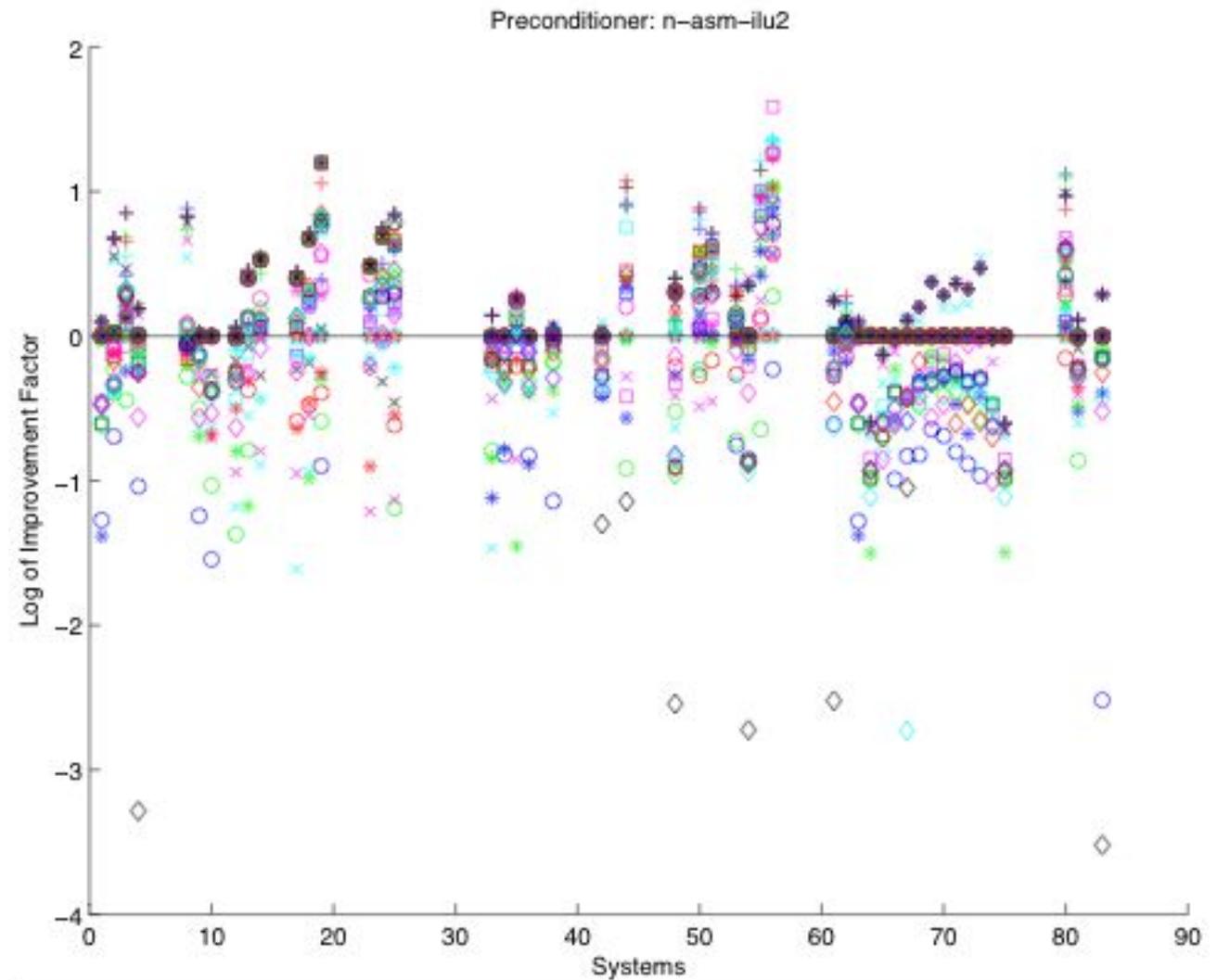


# Algorithm tuning efficiency

- **Parameter tuning critical to solver convergence performance**
  - **optimal scalable solvers are iterative, and hierarchical (employ multiple representations of the same continuous operator on different scales)**
  - **impedes their widespread adoption and leaves performance “on the table”**
- **Solver of choice is data- and machine-dependent**
- **Solvers used in inner loops of apps, w/related data**
- **Tremendous opportunity to apply machine learning methods (e.g., “boosting”)**

# Automated solver tuning

Example of performance variations of differently tuned Krylov solvers over a sequence of about 85 systems, vertical axis is log of runtime improvements



# On quality software



“Quality software  
*renormalizes* the difficulty  
of doing computation.”

*Peter Lax, 2005 Abel Prize  
Winner*

# What's coming in SciDAC-2 (2006 onward)

- **Commitment to maintenance of the many solver toolkits already in use**
- **Greater use made of “embedded mathematicians” (DOE “SAPP”-style funding)**
- **Greater software interface standardization**
- **Better vertical integration of separately developed SciDAC-1 components:**
  - **meshing and discretization systems w/ solvers**
  - **solvers w/ performance engineering**
- **More methods for users to choose from dynamically**
- **More automated selection of complex methods**

# Acknowledgments

- **DOE**
- **NSF**
- **PETSc software team**
- **Hypre software team**
- **SuperLU software team**
- **SUNDIALS software team**
- **M. Adams, S. Bhowmick, J. Brannick, J. Demmel, O. Ghattas, S. Jardin, K. Ko, A. Mirin, D. Schnack**

## Related URLs

- **TOPS project**

*<http://www.tops-scidac.org>*

- **SciDAC initiative**

*<http://www.science.doe.gov/scidac>*

- **SCaLeS report**

*<http://www.pnl.gov/scales>*

# Abstract

**Computational enabling technologies must offer both high-level abstractions in the language of their intended user community and detailed access to their powerful, composable innards for developers and prototypers. This can be achieved through a multilevel interface with robust default settings for a host of tuning parameters, the knobs for which can be exposed on demand. Drawing upon collaborations between DOE's Integrated Software Infrastructure Center for scalable solvers (TOPS, [www.tops-scidac.org](http://www.tops-scidac.org)) and projects in fusion energy, accelerator design, and QCD, we illustrate the necessity of a providing a rich set of linear solvers under a common high-level interface, in order to progress beyond mere feasibility to performance and portability. The TOPS project integrates the following elements presented at ACTS 2005: Hypre, PETSc, ScaLAPACK, SUNDIALS, and SuperLU.**

**While many customers can be adequately served with well defined multilevel software interfaces, we emphasize from this same set of SciDAC collaborations the importance of cross-disciplinary human interaction to discover altogether better abstractions (e.g., one fully coupled nonlinear problem, rather than a sequence of operator-split, linearized problems).**