The Portable Extensible Toolkit for Scientific computing

New developments, memory performance, and algorithmic experimentation

Jed Brown

Laboratory of Hydrology, Hydraulics, and Glaciology
ETH Zürich

NOTUR 2010-05-21, Bergen

Outline

Portable Extensible Toolkit for Scientific computing

- Architecture
  - tightly coupled (e.g. XT5, BG/P, Earth Simulator)
  - loosely coupled such as network of workstations
- Operating systems (Linux, Mac, Windows, BSD, proprietary Unix)
- Any compiler
- Real/complex, single/double precision, 32/64-bit int
- Usable from C, C++, Fortran 77/90, and Python
- Free to everyone (BSD-style license), open development
- 500B unknowns, 75% weak scalability on Jaguar (225k cores) and Jugene (295k cores)
- Same code runs performantly on a laptop
- ~~No~~ iPhone support

Portable Extensible Toolkit for Scientific computing

- Architecture
  - tightly coupled (e.g. XT5, BG/P, Earth Simulator)
  - loosely coupled such as network of workstations
- Operating systems (Linux, Mac, Windows, BSD, proprietary Unix)
- Any compiler
- Real/complex, single/double precision, 32/64-bit int
- Usable from C, C++, Fortran 77/90, and Python
- Free to everyone (BSD-style license), open development
- 500B unknowns, 75% weak scalability on Jaguar (225k cores) and Jugene (295k cores)
- Same code runs performantly on a laptop
- ~~No~~ iPhone support

Portable **Extensible** Toolkit for Scientific computing

Philosophy: Everything has a plugin architecture

- Vectors, Matrices, Coloring/ordering/partitioning algorithms
- Preconditioners, Krylov accelerators
- Nonlinear solvers, Time integrators
- Spatial discretizations/topology*

Example

Vendor supplies matrix format and associated preconditioner, distributes compiled shared library. Application user loads plugin at runtime, no source code in sight.

Portable Extensible **Toolkit** for Scientific computing

Algorithms, (parallel) debugging aids, low-overhead profiling

Composability

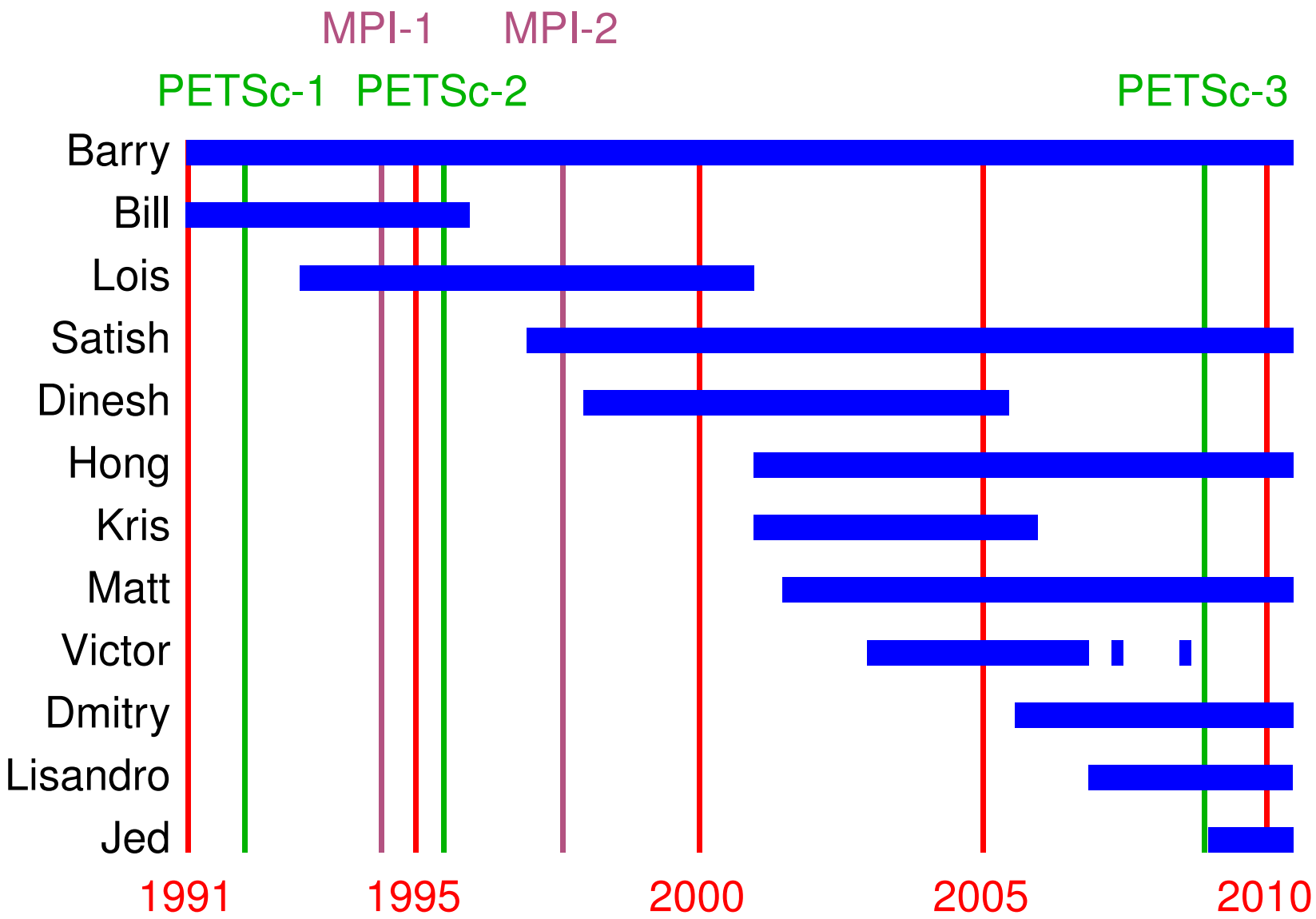
Try new algorithms by choosing from product space and composing existing algorithms (multilevel, domain decomposition, splitting).

Experimentation

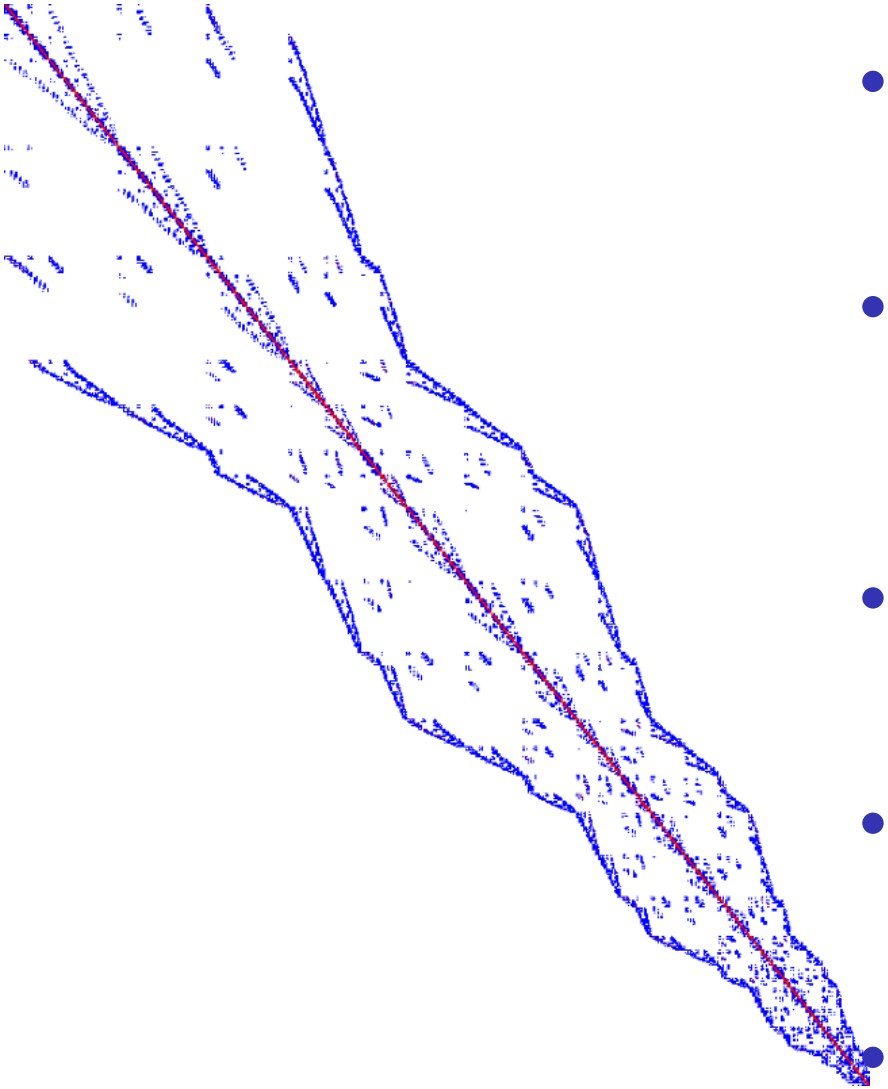
- It is not possible to pick the solver *a priori*.
  What will deliver best/competitive performance for a given physics, discretization, architecture, and problem size?
- PETSc's response: expose an algebra of composition so new solvers can be created at runtime.
- Important to keep solvers decoupled from physics and discretization because we also experiment with those.

Portable Extensible Toolkit for **Scientific computing**

- Computational Scientists
  - PyLith (CIG), Underworld (Monash), Magma Dynamics (LDEO, Columbia), PFLOTRAN (DOE), SHARP/UNIC (DOE)
- Algorithm Developers (iterative methods and preconditioning)
- Package Developers
  - SLEPc, TAO, Deal.II, Libmesh, FEniCS, PETSc-FEM, MagPar, OOFEM, FreeCFD, OpenFVM
- Funding
  - Department of Energy
    - SciDAC, ASCR ISICLES, MICS Program, INL Reactor Program
  - National Science Foundation
    - CIG, CISE, Multidisciplinary Challenge Program
- Hundreds of tutorial-style examples
- Hyperlinked manual, examples, and manual pages for all routines
- Support from `petsc-maint@mcs.anl.gov`

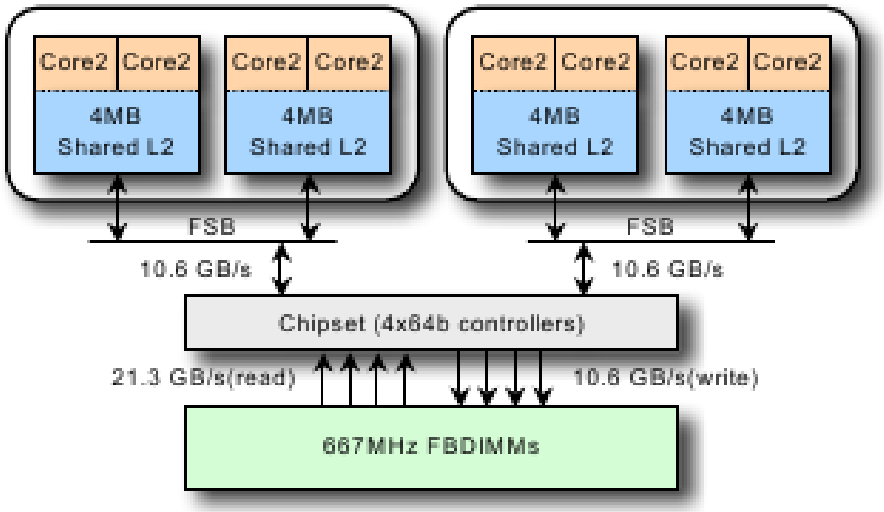

MPI-1
MPI-2
PETSc-1
PETSc-2
PETSc-3
Barry
Bill
Lois
Satish
Dinesh
Hong
Kris
Matt
Victor
Dmitry
Lisandro
Jed
1991
1995
2000
2005
2010

Bottlenecks of (Jacobian-free) Newton-Krylov



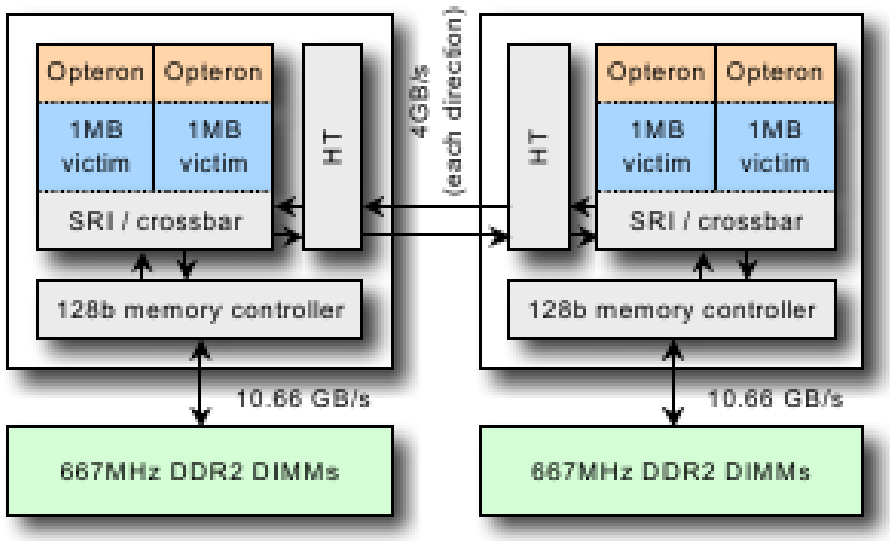
- Matrix assembly
  - integration/fluxes: FPU
  - insertion: memory/branching
- Preconditioner setup
  - coarse level operators
  - overlapping subdomains
  - (incomplete) factorization
- Preconditioner application
  - triangular solves/relaxation: memory
  - coarse levels: network latency
- Matrix multiplication
  - Sparse storage: memory
  - Matrix-free: FPU
- Globalization

Intel Clowertown



- 75 Gflop/s
- 21 GB/s bandwidth
- thread + instruction level parallelism
- vector instructions (SSE)

AMD Opteron



- 17 Gflop/s
- 21 GB/s bandwidth
- thread + instruction level parallelism
- vector instructions (SSE)

Hardware capabilities

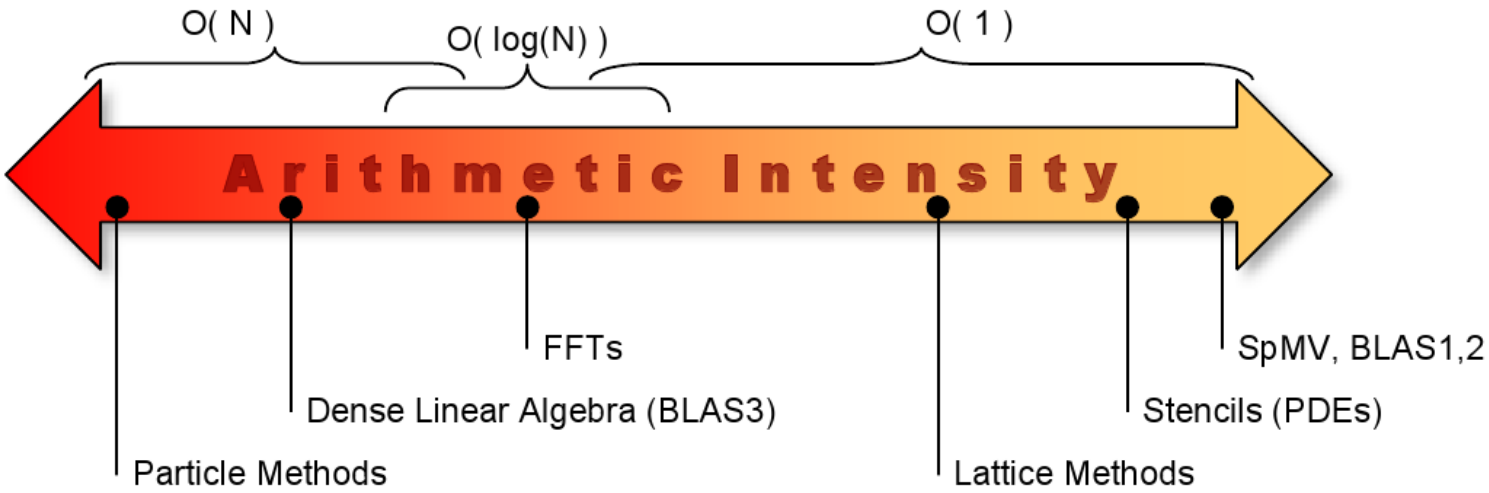
Floating point unit

Recent Intel: each core can issue

- 1 packed add (latency 3)
- 1 packed mult (latency 5)
- One can include a read
- Out of Order execution
- Peak: 10 Gflop/s (`double`)

Memory

- $\sim$ 250 cycle latency
- 5.3 GB/s bandwidth
- 1 `double` load / 3.7 cycles
- Pay by the cache line (32/64 B)
- L2 cache: $\sim$ 10 cycle latency



Sparse Mat-Vec performance model

Compressed Sparse Row format (AIJ)

For $m \times n$ matrix with N nonzeros

- ai row starts, length $m+1$
- aj column indices, length N, range $[0, n-1)$
- aa nonzero entries, length N, scalar values

$y \leftarrow y + Ax$

```
for (i=0; i<m; i++)
  for (j=ai[i]; j<ai[i+1]; j++)
    y[i] += aa[j] * x[aj[j]];
```

- One add and one multiply per inner loop
- Scalar `aa[j]` and integer `aj[j]` only used once
- Must load `aj[j]` to read from `x`, may not reuse cache well

Memory Bandwidth

- Stream Triad benchmark (GB/s): $w \leftarrow \alpha x + y$

Threads per Node	Cray XT5		BlueGene/P	
	Total	Per Core	Total	Per Core
1	8448	8448	2266	2266
2	10112	5056	4529	2264
4	10715	2679	8903	2226
6	10482	1747	-	-

- Sparse matrix-vector product: 6 bytes per flop

Machine	Peak MFlop/s per core	Bandwidth (GB/s)		Ideal MFlop/s
		Required	Measured	
Blue Gene/P	3,400	20.4	2.2	367
XT5	10,400	62.4	1.7	292

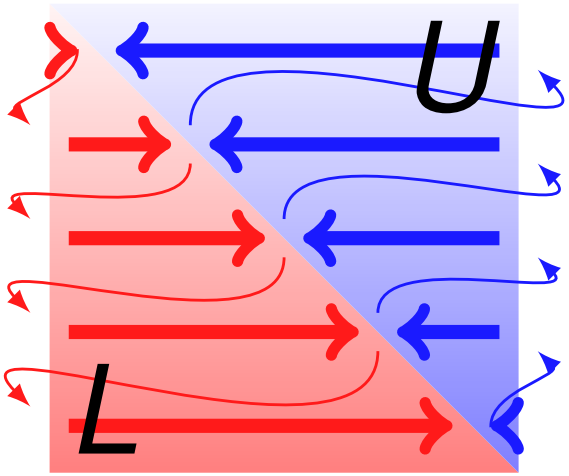
Optimizing Sparse Mat-Vec

- Order unknows so that vector reuses cache (Reverse Cuthill-McKee)
  - Optimal: $\frac{(2\text{ flops})(\text{bandwidth})}{\texttt{sizeof(Scalar)}+\texttt{sizeof(Int)}}$
  - Usually improves strength of ILU and SOR
- Coalesce indices for adjacent rows with same nonzero pattern (Inodes)
  - Optimal: $\frac{(2\text{ flops})(\text{bandwidth})}{\texttt{sizeof(Scalar)}+\texttt{sizeof(Int)}/i}$
  - Can do block SOR (much stronger than scalar SOR)
  - Default in PETSc, turn off with `-mat_no_inode`
  - Requires ordering unknowns so that fields are interlaced, this is (much) better for memory use anyway
- Use explicit blocking, hold one index per block (BAIJ format)
  - Optimal: $\frac{(2\text{ flops})(\text{bandwidth})}{\texttt{sizeof(Scalar)}+\texttt{sizeof(Int)}/b^2}$
  - Block SOR and factorization
  - Symbolic factorization works with blocks (much cheaper)
  - Very regular memory access, unrolled dense kernels
  - Faster insertion: `MatSetValuesBlocked()`

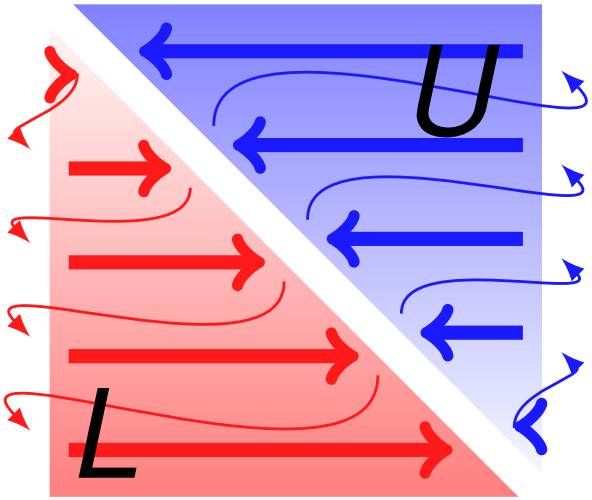
Optimizing unassembled Mat-Vec

- High order spatial discretizations do more work per node
  - Dense tensor product kernel (like small BLAS3)
  - Cubic (Q_3) elements in 3D can achieve $> 60\%$ of peak FPU (compare to $< 6\%$ for assembled operators on multicore)
  - Can store Jacobian information at quadrature points (usually pays off for Q_2 and higher in 3D)
  - Spectral methods
  - Often still need an assembled operator for preconditioning
- Boundary element methods
  - Dense kernels
  - Fast Multipole Method (FMM)

Storing Factors



- Forward and back solves skip over unused part of each row
  - Pollutes cache and bus with unused part, software prefetch helps some
- Back solves move backward through memory and so does vector
  - Core 2: hardware prefetch 16 forward-moving pointers and 4 backward-moving not across 4 KiB page boundaries



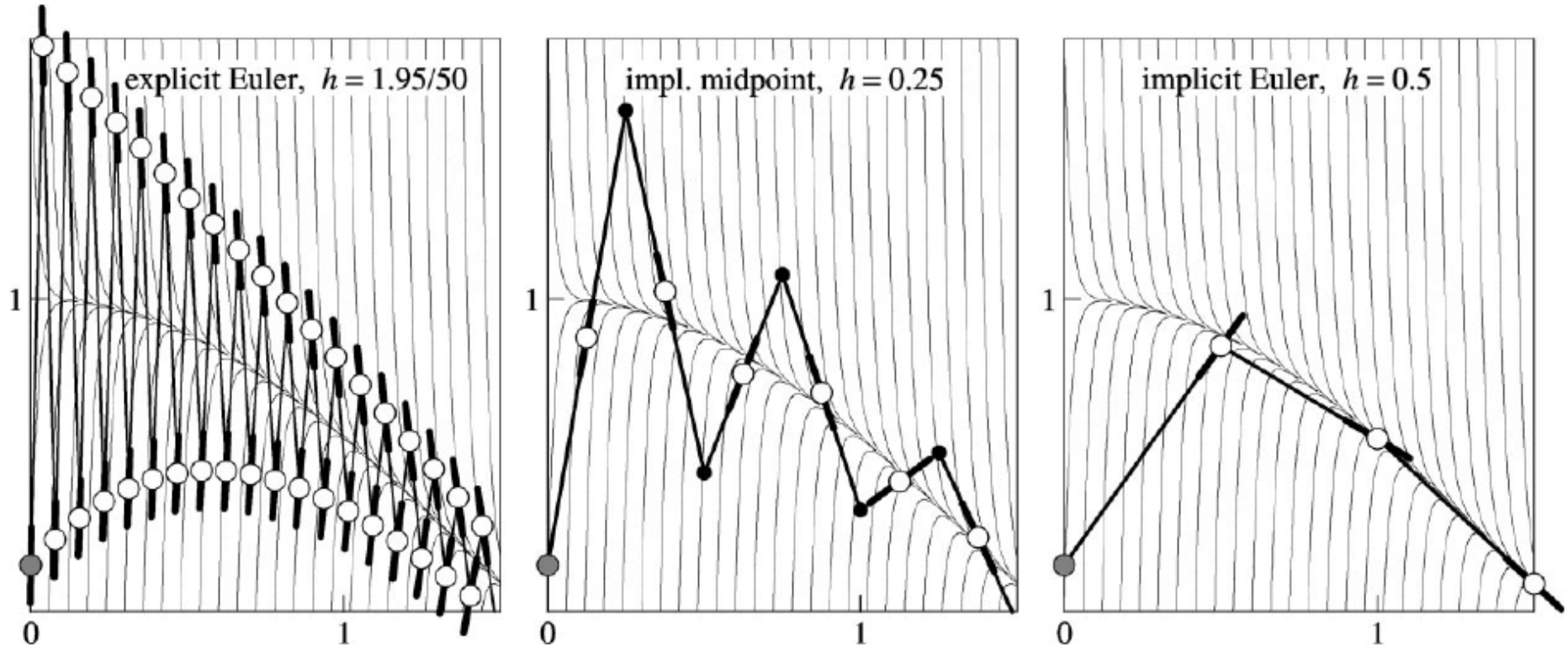
- Forward and back solves get contiguous memory
- Move forward through memory for matrix entries
  - Good for vector prefetch which necessarily tracks backward through memory

Improvement from better storage for factors

Processor	Matrix	SpMV (Mflop/s)	Triangular solves	
			Old format	New format
Core 2 Duo	7-point Laplace	537	261 (49%)	447 (83%)
	3D Euler AIJ	620	260 (42%)	660 (106%)
	3D Euler BAIJ	890	468 (53%)	758 (85%)
BlueGene/P	7-point Laplace	98	46 (47%)	62 (63%)
	3D Euler AIJ	148	99 (66%)	126 (85%)
	3D Euler BAIJ	303	198 (65%)	260 (86%)

Barry Smith and Hong Zhang, *Sparse triangular solves for ILU revisited: data layout crucial to better performance*, submitted to Intl. J. of High Performance Computing Applications.

Stiff integrators



(Hairer and Wanner, 1999)

$$\dot{x} + 50(x - \cos t) = 0$$

- $\dot{x} = \lambda x$
- $\mathscr{R}(h\lambda) = x^{n+1}/x^n$
- A-stable: $|\mathscr{R}(\{\Re[z] \leq 0\})| \leq 1$
- L-stable: $\lim_{z\to\infty} \mathscr{R}(z) = 0$

Barriers

Dahlquist's second barrier

An A-stable linear multistep method has order $p \leq 2$.

Diagonally implicit Runge-Kutta

A DIRK evaluates the first stage to order $q = 1$.

Circumvent with general linear methods

$$\begin{bmatrix} Y \\ X^{n+1} \end{bmatrix} = \begin{bmatrix} A & U \\ B & V \end{bmatrix} \begin{bmatrix} h\dot{Y} \\ X^{n} \end{bmatrix}$$

- stage values $Y = \{y_1, \ldots, y_s\}$
- Nordsieck vector passed between steps

$$X = \{x_1, \ldots, x_r\} = \{x, h\dot{x}, h^2\ddot{x}, \ldots\}$$

- A can be lower triangular (permits stages to be solved sequentially)

Special class: IRKS (inherent Runge-Kutta stability)

- A-stable
- L-stable
- order p, stage order q, $p = q = r - 1 = s - 1$
- diagonally implicit
- Asymptotically correct error estimates for present method *and* method of order $p+1$.
- Implemented in PETSc's TSGL (`-ts_type gl`)
  - implicit DAE form: $f(t, x, \dot{x}) = 0$
  - orders $p = 1, \ldots, 5$
  - adaptive-order, adaptive-step controller
  - plugin architecture for controllers
  - make new methods available to the controller by giving their tableau, error estimates computed automatically
  - solve $f(t, x, x_0 + \alpha x) = 0$ with SNES

Butcher, Jackiewicz, Wright, *On error propagation in general linear methods for ordinary differential equations*, 2007.

Strong stability preserving methods: `-ts_type ssp`

- Conservation laws $u_t + \operatorname{div} f(u) = 0$
- Discontinuous solutions, requires careful discretization to prevent spurious oscillations
- SSP property allows decoupling of spatial and time discretizations
- Choose spatial discretization that is stable with forward-Euler (e.g. TVD finite volume, ENO/WENO, Discontinuous Galerkin)
- Barriers for methods of order greater than 1 with s stages

$$c_{\text{eff}} = \frac{|\lambda_{\max}| \Delta t}{s \Delta x} < \begin{cases} 1 & \text{Explicit,} \\ 2 & \text{Implicit.} \end{cases}$$

-

Popular method	c_{eff}	Improved method	c_{eff}	Storage
SSPRK$(2,2)$	0.500	SSPRK$(m,2)$	$1-1/m$	$2N^*$
SSPRK$(3,3)$	0.333	SSPRK$(n^2,3)$	$1-1/n$	$2N$
SSPRK$(5,4)$	0.377	SSPRK$(10,4)$	0.6	$2N$

David Ketcheson, *Highly efficient strong stability-preserving Runge-Kutta methods with low-storage implementations*, 2008.

Splitting for Multiphysics

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}$$

- Relaxation: `-pc_fieldsplit_type [additive,multiplicative,symmetric_multiplicative]`

$$\begin{bmatrix} A & \\ & D \end{bmatrix}^{-1} \qquad \begin{bmatrix} A & \\ C & D \end{bmatrix}^{-1} \qquad \begin{bmatrix} A & \\ & 1 \end{bmatrix}^{-1} \left(1 - \begin{bmatrix} A & B \\ & 1 \end{bmatrix} \begin{bmatrix} A & \\ C & D \end{bmatrix}^{-1} \right)$$

  - Gauss-Seidel inspired, works when fields are loosely coupled
- Factorization: `-pc_fieldsplit_type schur`

$$\begin{bmatrix} 1 & \\ CA^{-1} & 1 \end{bmatrix} \begin{bmatrix} A & B \\ & S \end{bmatrix}, \qquad S = D - CA^{-1}B$$

  - robust (exact factorization), can often drop lower block
  - how to precondition S which is usually dense?
    - interpret as differential operators, use approximate commutators

Examples of splitting for strong coupling

- Incompressible flow

$$J(u)\begin{bmatrix} w \\ p \end{bmatrix} \sim \begin{bmatrix} u\cdot\nabla - \Delta & \nabla \\ -\operatorname{div} & \end{bmatrix}\begin{bmatrix} w \\ p \end{bmatrix}$$

$$S \sim \operatorname{div}(u\cdot\nabla - \Delta)^{-1}\nabla \approx \Delta(u\cdot\nabla - \Delta)^{-1}$$

  - S^{-1} requires solve with a Laplacian and application of an advection-diffusion operator defined in pressure space (Elman et al, 2008)

- Shallow water with stiff gravity wave, $\alpha = 1/\Delta t$, wave speed $\sqrt{gh}$

$$J(h,uh)\begin{bmatrix} h' \\ uh' \end{bmatrix} \sim \begin{bmatrix} \alpha & \operatorname{div} \\ gh\nabla & \alpha \end{bmatrix}\begin{bmatrix} h' \\ uh' \end{bmatrix} + (\text{non-stiff terms})$$

$$S \sim \alpha - \alpha^{-1} g \operatorname{div} h\nabla$$

  - Scalar parabolic operator, good for multigrid (Mousseau, Knoll, and Reisner, 2002)

Hydrostatic equations for ice sheet flow

- Valid in the limit $w_x \ll u_z$, independent of basal friction
- Eliminate p and w by incompressibility:

  3D elliptic system for $u = (u, v)$

$$-\nabla \cdot \left[\eta \begin{pmatrix} 4u_x + 2v_y & u_y + v_x & u_z \\ u_y + v_x & 2u_x + 4v_y & v_z \end{pmatrix} \right] + \rho g \nabla s = 0$$

$$\eta(\gamma) = \frac{B}{2}(\varepsilon^2 + \gamma)^{\frac{1-\mathfrak{n}}{2\mathfrak{n}}}, \qquad \mathfrak{n} \approx 3$$

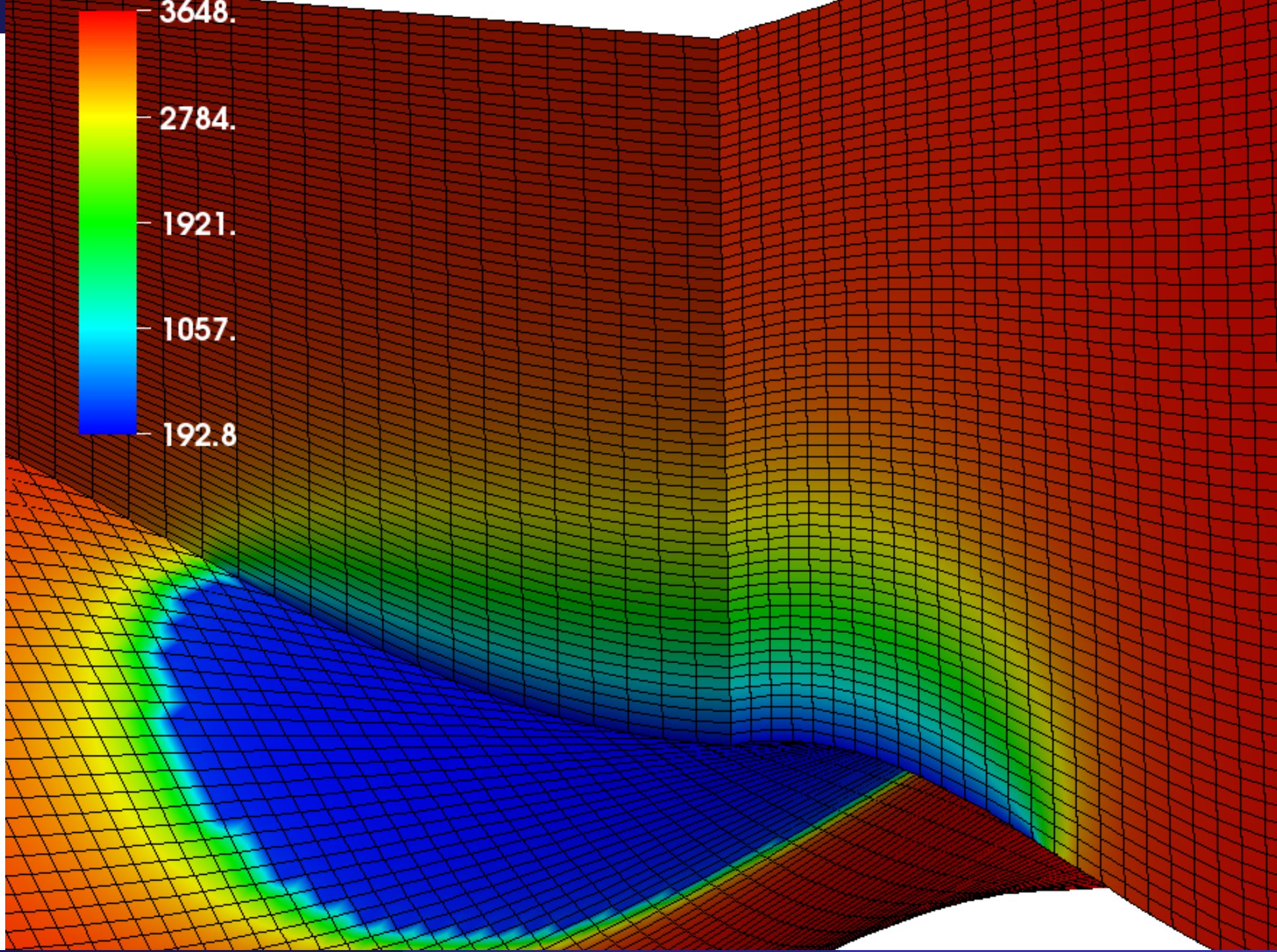
$$\gamma = u_x^2 + v_y^2 + u_x v_y + \frac{1}{4}(u_y + v_x)^2 + \frac{1}{4}u_z^2 + \frac{1}{4}v_z^2$$

  and slip boundary $\sigma \cdot n = \beta^2 u$ where

$$\beta^2(\gamma_b) = \beta_0^2(\varepsilon_b^2 + \gamma_b)^{\frac{\mathfrak{m}-1}{2}}, \qquad 0 < \mathfrak{m} \leq 1$$

$$\gamma_b = \frac{1}{2}(u^2 + v^2)$$

- Q_1 FEM: `src/snes/examples/tutorials/ex48.c`


3648.
2784.
1921.
1057.
192.8

What about splitting at the global level?

- Split (u, v) multiplicatively at global level: `-pc_type fieldsplit`
  - parallel direct solve in splits
    ```
    -fieldsplit_pc_type cholesky
    -fieldsplit_pc_factor_mat_solver_package mumps
    ```
  - Split additively instead
    ```
    -pc_fieldsplit_type additive
    ```
  - Parallel ML in splits, ASM(1)/ICC(1) on levels
    ```
    -fieldsplit_pc_type asm
        -fieldsplit_sub_pc_type icc
        -fieldsplit_sub_pc_factor_levels 1
    ```
  - Parallel BoomerAMG in splits
    ```
    -fieldsplit_pc_type hypre
    ```
  - ASM/Cholesky in splits
    ```
    -fieldsplit_pc_type asm
        -fieldsplit_sub_pc_type cholesky
    ```
  - ASM/ICC(0) in splits
    ```
    -fieldsplit_pc_type asm
        -fieldsplit_sub_pc_type icc
    ```

Split in subdomains?

- ASM on coupled system: `-pc_type asm`
  - Split in subdomains, Cholesky in splits
    `-sub_pc_type fieldsplit`
    `-sub_fieldsplit_pc_type cholesky`
  - Split in subdomainst, ML in splits
    `-sub_pc_type fieldsplit`
    `-sub_fieldsplit_pc_type ml`
  - Split in subdomains, BoomerAMG in splits
    `-sub_pc_type fieldsplit`
    `-sub_fieldsplit_pc_type hypre`
  - Split in subdomains, ICC(1) in splits
    `-sub_pc_type fieldsplit`
    `-sub_fieldsplit_pc_type icc`
  - Block ICC(0) on subdomains (no splitting), stored as block symmetric
    `-sub_pc_type icc -da_mat_type sbaij`

Coupled Multigrids

- Geometric multigrid with isotropic coarsening, ASM(1)/Cholesky and ASM(0)/ICC(0) on levels

  `-mg_levels_pc_type bjacobi -mg_levels_sub_pc_type icc -mg_levels_1_pc_type asm -mg_levels_1_sub_pc_type cholesky`
- . . . with Galerkin coarse operators

  `-pc_mg_galerkin`
- . . . with ML's aggregates

  `-pc_type ml -mg_levels_pc_type asm`
- Geometric multigrid with aggressive semi-coarsening, ASM(1)/Cholesky and ASM(0)/ICC(0) on levels

  `-da_refine_hierarchy_x 1,1,8,8 -da_refine_hierarchy_y 2,2,1,1 -da_refine_hierarachy_z 2,2,1,1`
- Simulate 1024 cores, interactively, on my laptop

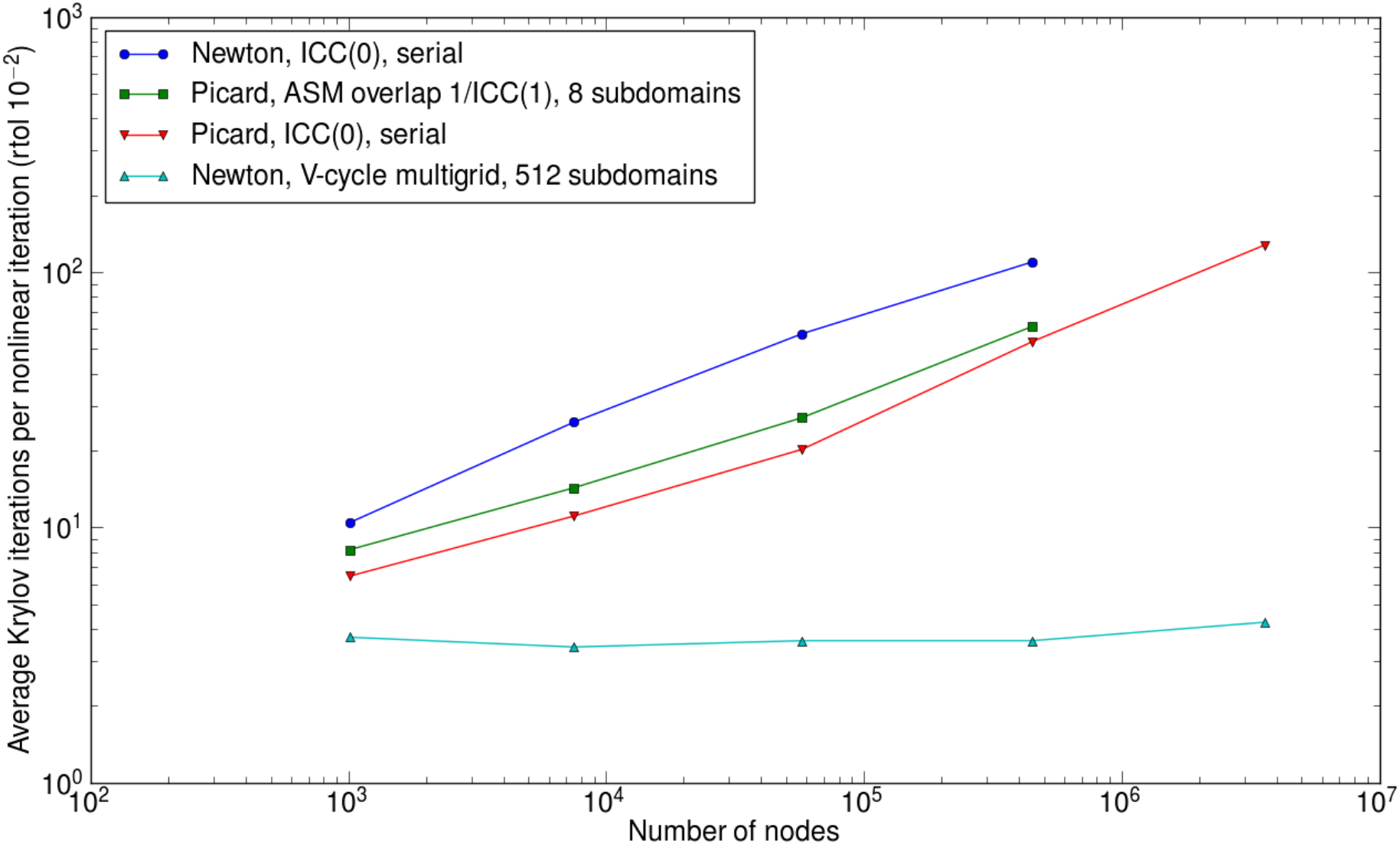
  `-mg_levels_pc_asm_blocks 1024`

Summary of solvers

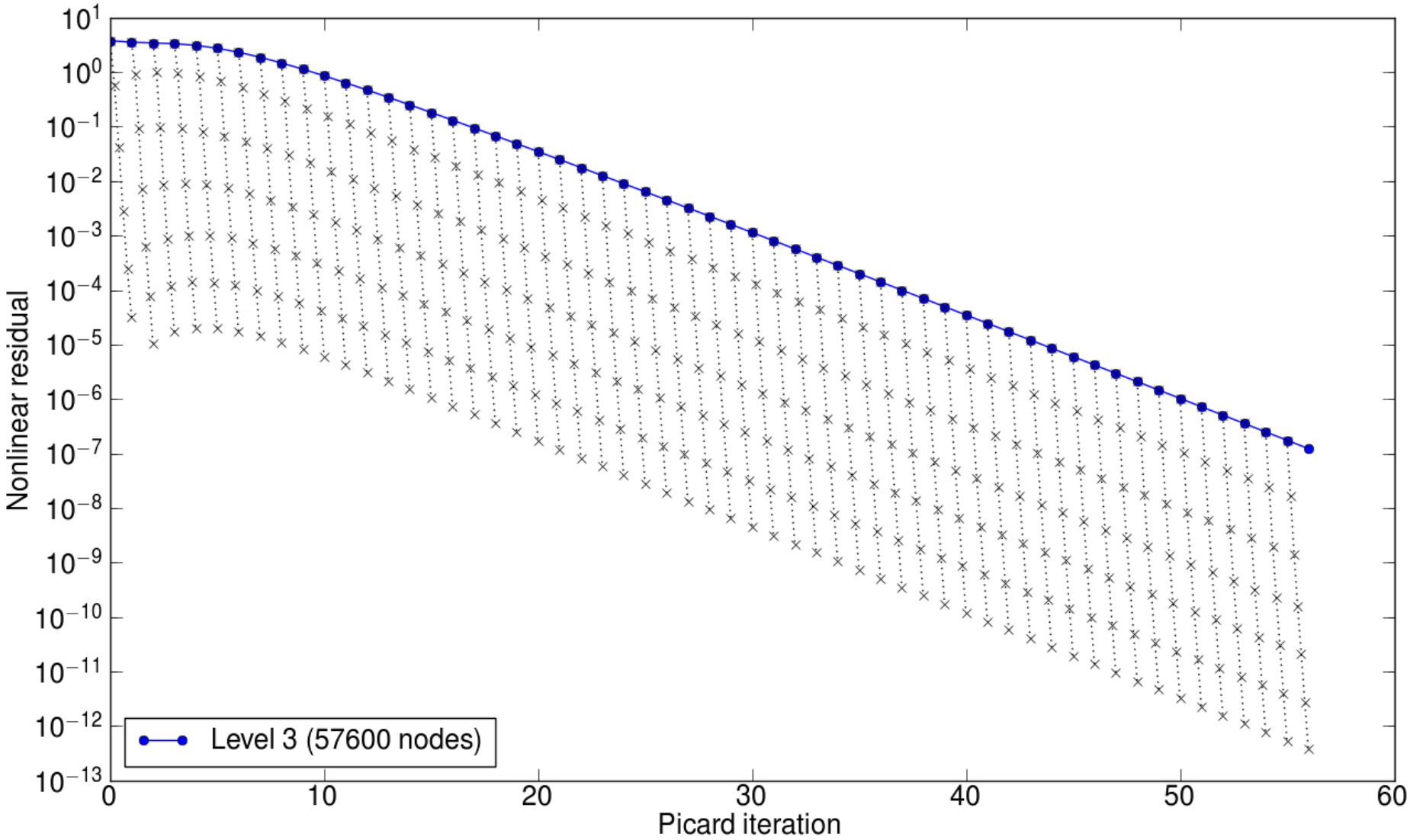
Method	Avg Krylov/Newton
Global multiplicative FieldSplit, ASM/LU	175
Global multiplicative FieldSplit, BoomerAMG	59
Global multiplicative FieldSplit, strong ML	71
Global additive FieldSplit, ASM/LU	197
Global ASM, FieldSplit/LU inside	215
Global ASM, LU inside	167
Coupled BoomerAMG	60
Coupled ML with strong smoothers	72
Geometric multigrid	11
Geometric multigrid, Galerkin coarse	122

Smallish size $40 \times 40 \times 10$, relatively difficult parameters

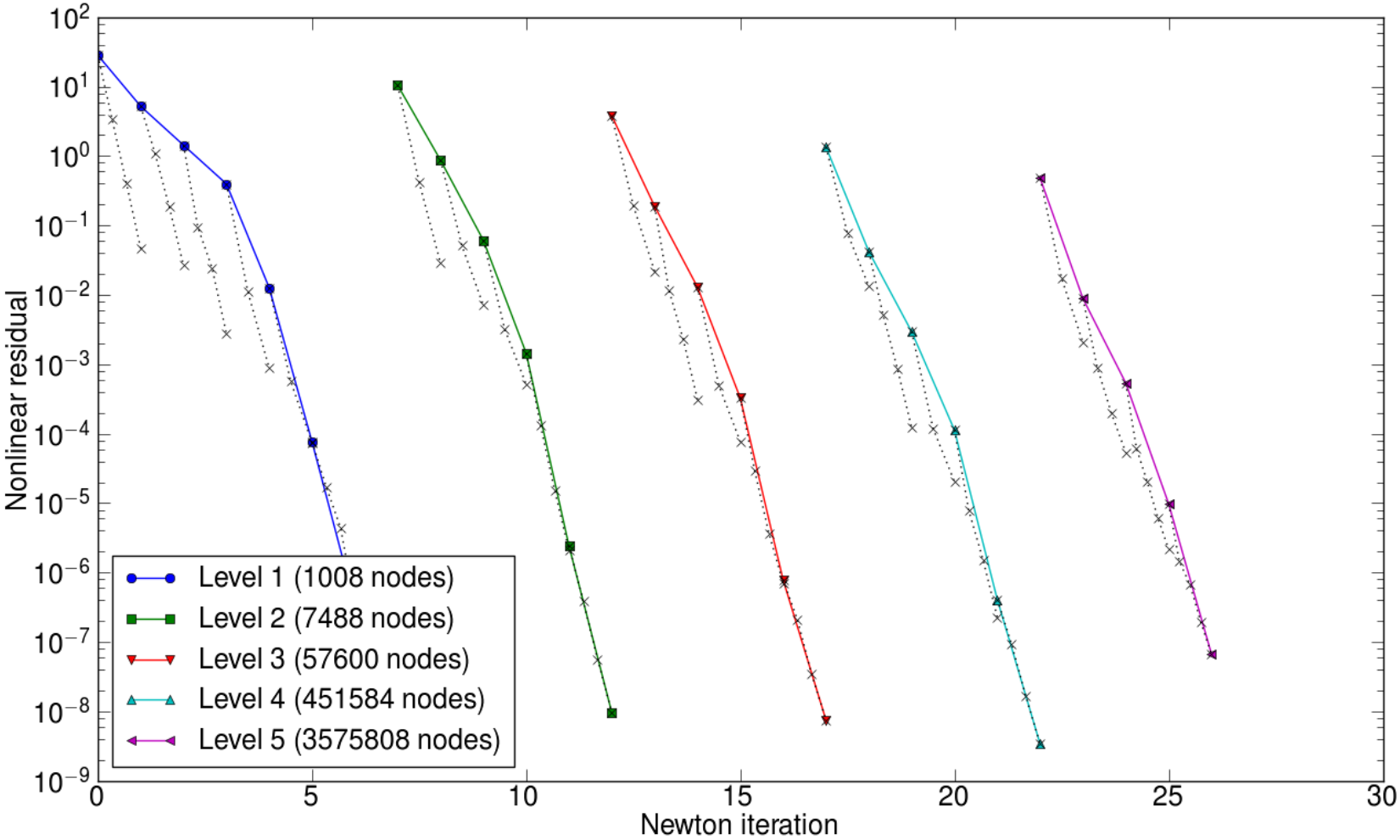
Linear solve performance



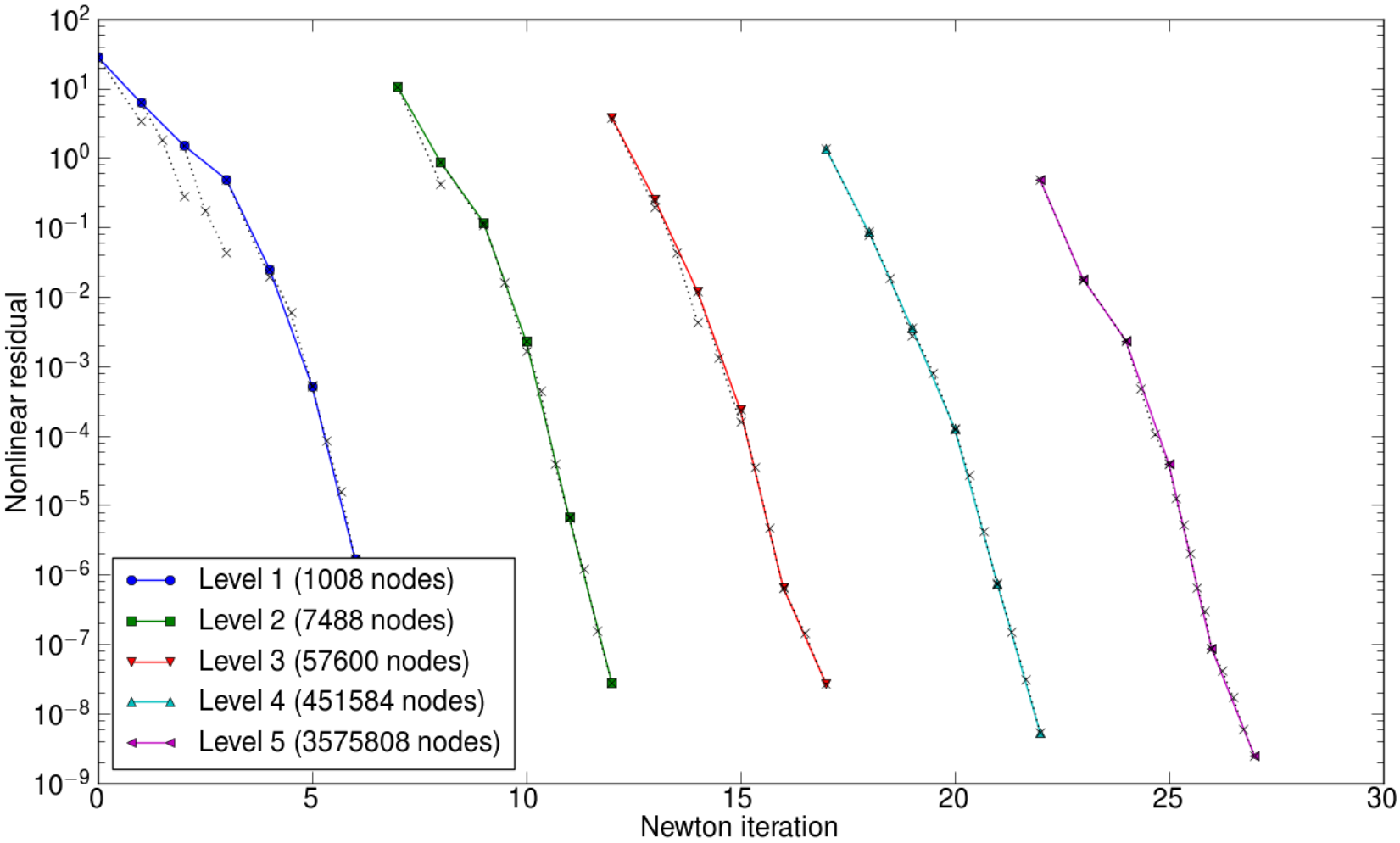
Status-quo Picard



Grid sequencing: `-dmmg_grid_sequence`



Avoid oversolving



Luis Chacon's variant of Eisenstat-Walker:

```
-snes_ksp_ew -snes_ksp_ew_rtolmax 0.5 -snes_ksp_ew_version 3
```

Thoughts on Multigrid

- Rapid coarsening is essential for weak scalability
  - Push the algorithm towards “multilevel domain decomposition”
- Energy minimizing interpolants (Wan, Chan, and Smith 2000)
  - Similar to exotic Schwarz methods, see Dohrmann and Widlund 2008, 2009
  - Closely related to FETI-DP/BDDC coarse spaces
- (Precondition with) first-order upwind for transport/waves
- Smooth all components together (block SOR, Vanka smoothers for indefinite problems)
- Interpolation operators must be compatible with physics (e.g. inf-sup conditions)
- Ordering of unknowns can make incomplete factorization behave similar to line smoothers
- Nonlinear multigrid (FAS) is worth trying if pointwise or block residuals are cheap, or globalization is especially challenging
- Monotone multigrid (Kornhuber) for variational inequalities
- Boundary conditions in subdomain problems (“optimized Schwarz”)

Wrap-up

- PETSc can help you
  - easily construct a code to experiment with ideas
  - scale an existing code base
  - incorporate more scalable or higher performance algorithms
  - attain high performance on a variety of architectures
  - debug and profile a parallel application (not discussed today)
  - package and distribute your code (e.g. graph algorithms, domain decomposition and multilevel solvers), `--download-xxx`
- I will be around most of today, find me to discuss
  - new and old features in PETSc
  - performance, scalability, and algorithms
  - design of new codes
  - integration with your existing application
- `http://mcs.anl.gov/petsc`
- `petsc-users@mcs.anl.gov` and `petsc-dev@mcs.anl.gov`
- `petsc-maint@mcs.anl.gov`