PETSc Solvers Tutorial

Jed Brown

CU Boulder and Argonne National Laboratory

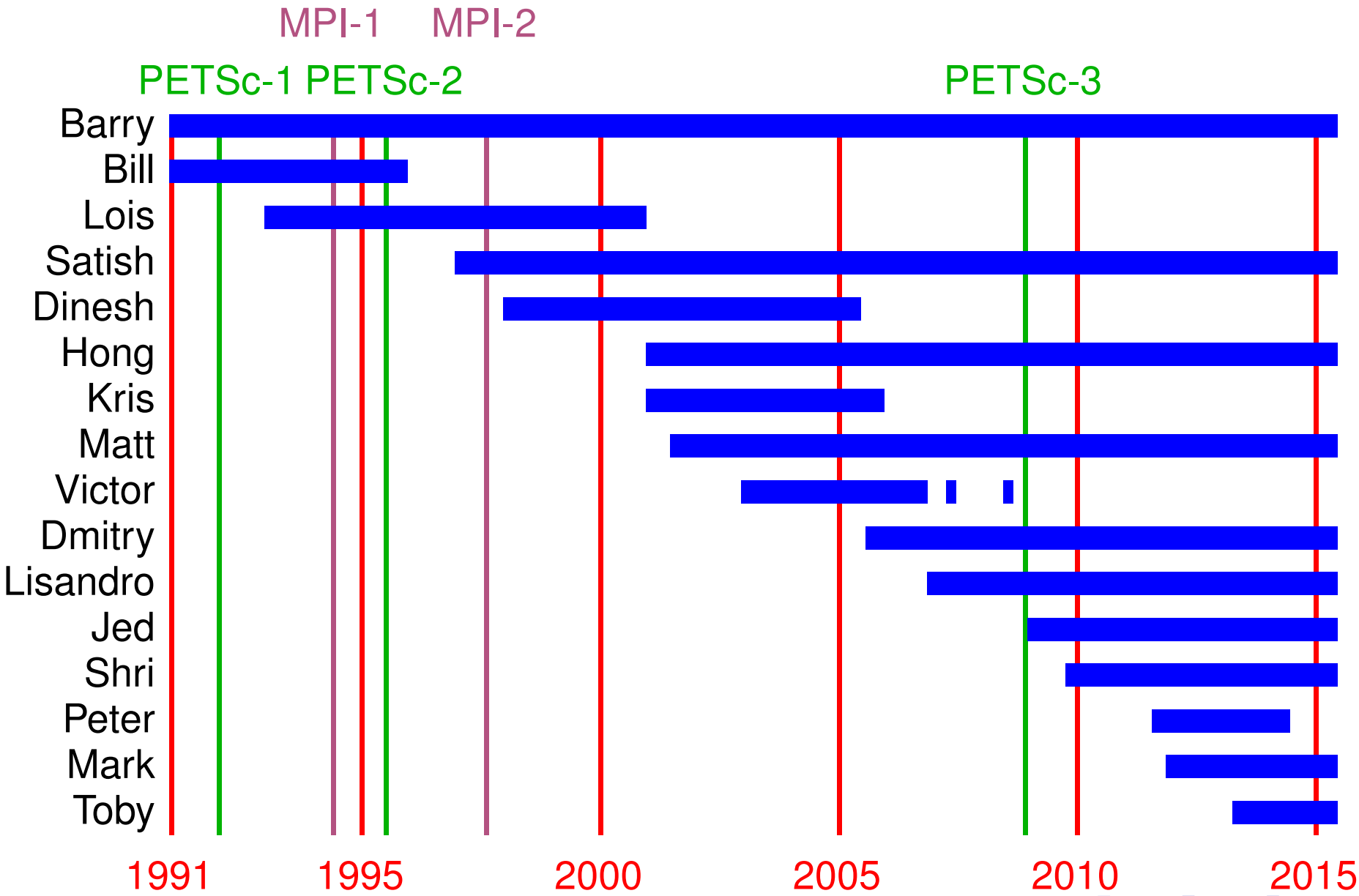
PETSc User Meeting, 2017-06-14, Boulder, CO

Outline

Follow Up; Getting Help

- `http://www.mcs.anl.gov/petsc`
- Public questions: `petsc-users@mcs.anl.gov`, archived
- Private questions: `petsc-maint@mcs.anl.gov`, not archived

Outline


MPI-1
MPI-2
PETSc-1
PETSc-2
PETSc-3
Barry
Bill
Lois
Satish
Dinesh
Hong
Kris
Matt
Victor
Dmitry
Lisandro
Jed
Shri
Peter
Mark
Toby
1991
1995
2000
2005
2010
2015

Portable Extensible Toolkit for Scientific computing

- Architecture
  - tightly coupled (e.g. Cray, Blue Gene)
  - loosely coupled such as network of workstations
  - GPU clusters (many vector and sparse matrix kernels)
- Operating systems (Linux, Mac, Windows, BSD, proprietary Unix)
- Any compiler
- Real/complex, single/double/quad precision, 32/64-bit int
- Usable from C, C++, Fortran 77/90, Python, and MATLAB
- Free to everyone (2-clause BSD license), open development
- 10^{12} unknowns, full-machine scalability on Top-10 systems
- Same code runs performantly on a laptop
- ~~No~~ iPhone support

Portable Extensible Toolkit for Scientific computing

- Architecture
  - tightly coupled (e.g. Cray, Blue Gene)
  - loosely coupled such as network of workstations
  - GPU clusters (many vector and sparse matrix kernels)
- Operating systems (Linux, Mac, Windows, BSD, proprietary Unix)
- Any compiler
- Real/complex, single/double/quad precision, 32/64-bit int
- Usable from C, C++, Fortran 77/90, Python, and MATLAB
- Free to everyone (2-clause BSD license), open development
- 10^{12} unknowns, full-machine scalability on Top-10 systems
- 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), Climate (ICL/UK Met), PFLOTRAN (DOE), MOOSE (DOE), Proteus (ERDC)
- 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`

Outline

Downloading

- `http://mcs.anl.gov/petsc`, download tarball
- We will use Git in this tutorial:
  - `http://git-scm.com`
  - Debian/Ubuntu: `$ apt install git`
  - Fedora: `$ yum install git`
- Get the most recent stable PETSc
  - `$ git clone https://bitbucket.org/petsc/petsc`
  - `$ cd petsc`
  - `$ git checkout maint`
- Get the latest bug fixes with `$ git pull`
- Can also `$ git checkout v3.7.6`

Configuration

Basic configuration

- `$ export PETSC_DIR=$PWD PETSC_ARCH=gnu-dbg`
- `$ ./configure`
- `$ make all test`

- Other common options
  - `--with-scalar-type=<real or complex>`
  - `--with-precision=<single,double,__float128>`
  - `--with-64-bit-indices`
  - `--download-{umfpack,mumps,scalapack,parmetis}`
- reconfigure at any time with

```
$ gnu-dbg/lib/petsc/conf/reconfigure-gnu-dbg.py \
      --new-options
```

An optimized build

- `$ gnu-dbg/lib/petsc/conf/reconfigure-gnu-dbg.py \`
  `PETSC_ARCH=gnu-opt \`
  `--with-debugging=0 && make PETSC_ARCH=gnu-opt`
- What does `--with-debugging=1` (default) do?
  - Keeps debugging symbols (of course)
  - Maintains a stack so that errors produce a full stack trace (even SEGV)
  - Does lots of integrity checking of user input
  - Places sentinels around allocated memory to detect memory errors
  - Allocates related memory chunks separately (to help find memory bugs)
  - Keeps track of and reports unused options
  - Keeps track of and reports allocated memory that is not freed
    `-malloc_dump`

Outline

MPI communicators

- Opaque object, defines process group and synchronization channel
- PETSc objects need an `MPI_Comm` in their constructor
  - `PETSC_COMM_SELF` for serial objects
  - `PETSC_COMM_WORLD` common, but not required
- Can split communicators, spawn processes on new communicators, etc
- Operations are one of
  - Not Collective: `VecGetLocalSize()`, `MatSetValues()`
  - Logically Collective: `KSPSetType()`, `PCMGSetCycleType()`
    - checked when running in debug mode
  - Neighbor-wise Collective: `VecScatterBegin()`, `MatMult()`
    - Point-to-point communication between two processes
    - Neighbor collectives in MPI-3
  - Collective: `VecNorm()`, `MatAssemblyBegin()`, `KSPCreate()`
    - Global communication, synchronous
    - Non-blocking collectives in MPI-3
- Deadlock if some process doesn't participate (e.g. wrong order)

Objects

```
Mat A;
PetscInt m,n,M,N;
MatCreate(comm,&A);
MatSetSizes(A,m,n,M,N);          /* or PETSC_DECIDE */
MatSetOptionsPrefix(A,"foo_");
MatSetFromOptions(A);
/* Use A */
MatView(A,PETSC_VIEWER_DRAW_WORLD);
MatDestroy(A);
```

- `Mat` is an opaque object (pointer to incomplete type)
  - Assignment, comparison, etc, are cheap
- What's up with this "Options" stuff?
  - Allows the type to be determined at runtime: `-foo_mat_type sbaij`
  - Inversion of Control similar to "service locator", related to "dependency injection"
  - Other options (performance and semantics) can be changed at runtime under `-foo_mat_`

Basic `PetscObject` Usage

Every object in PETSc supports a basic interface

Function	Operation
`Create()`	create the object
`Get/SetName()`	name the object
`Get/SetType()`	set the implementation type
`Get/SetOptionsPrefix()`	set the prefix for all options
`SetFromOptions()`	customize object from the command line
`SetUp()`	preform other initialization
`View()`	view the object
`Destroy()`	cleanup object allocation

Also, all objects support the `-help` option.

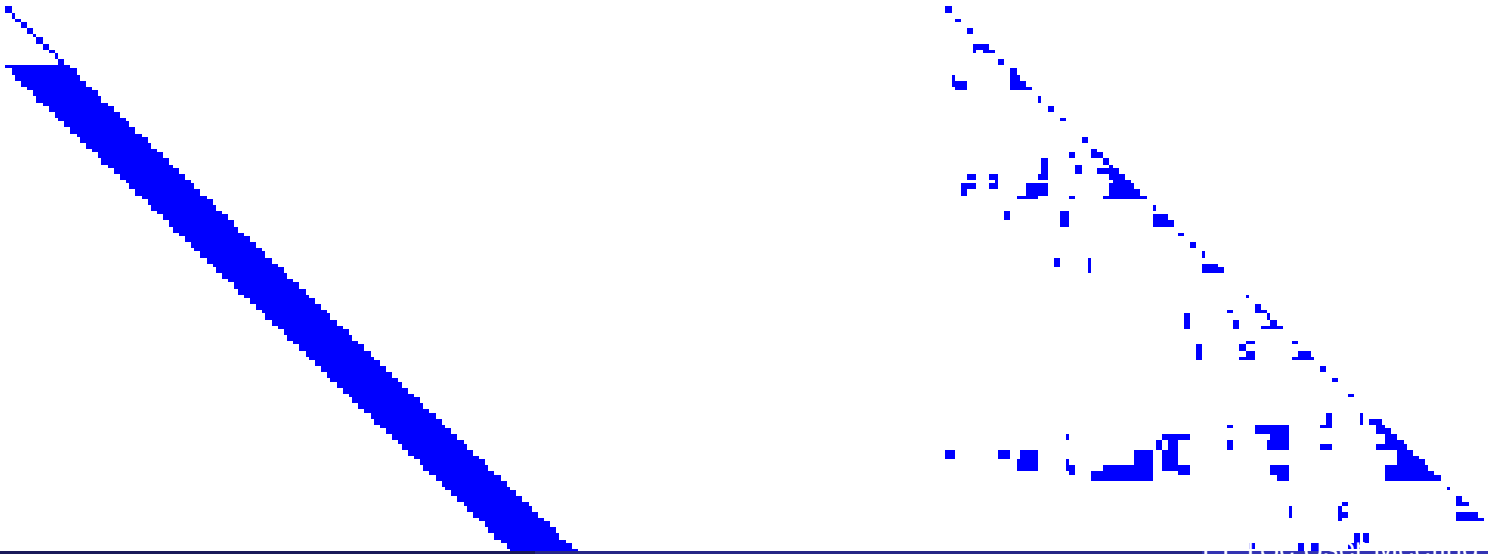
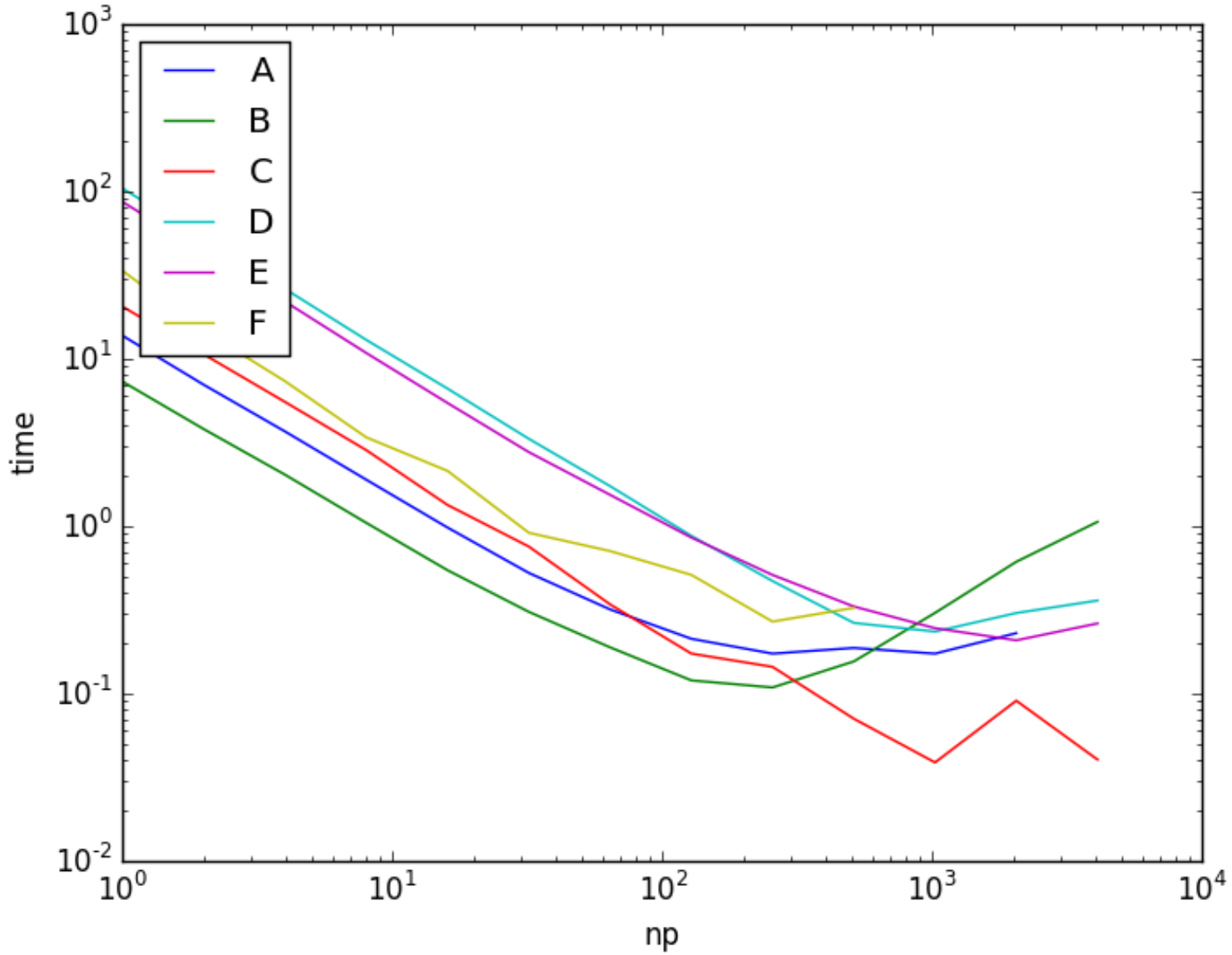
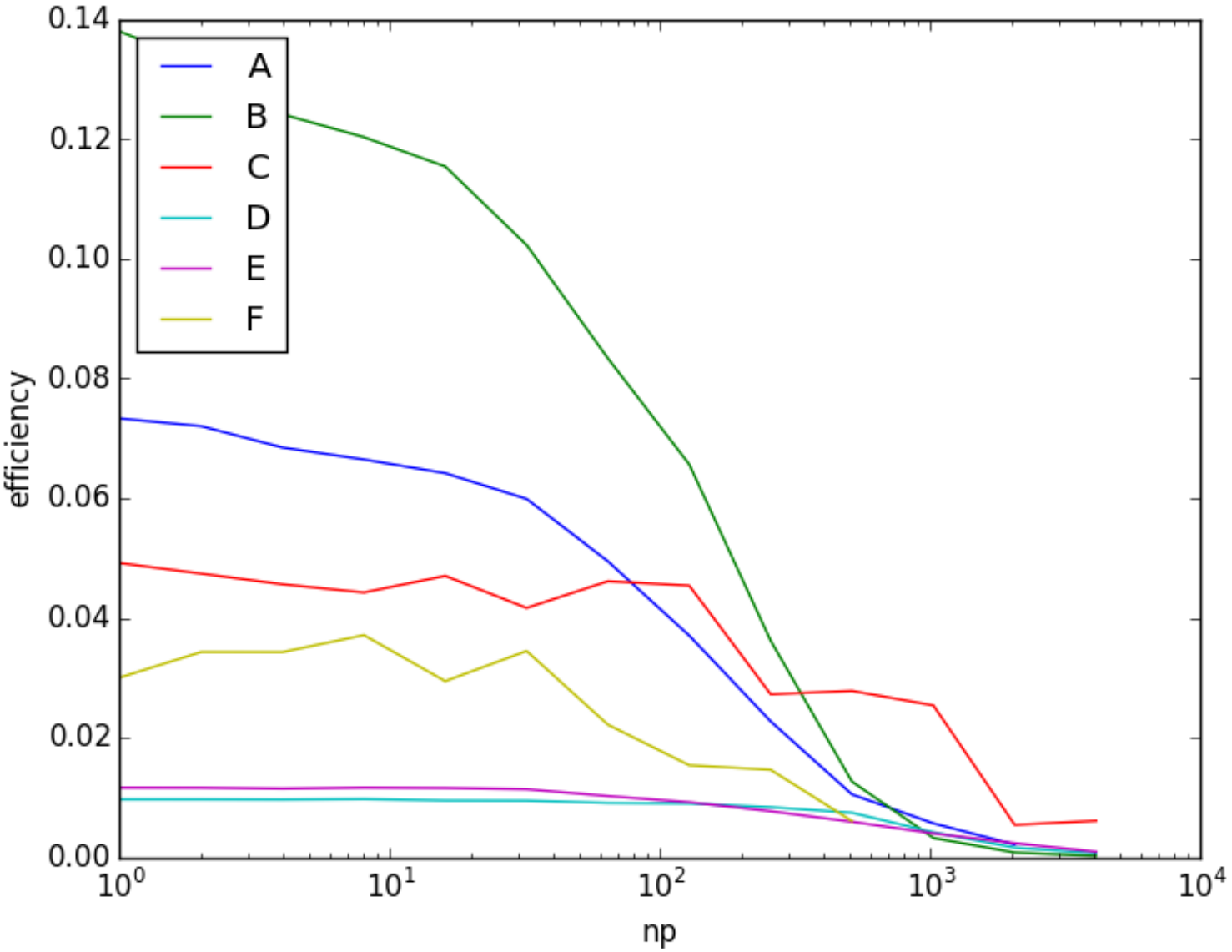
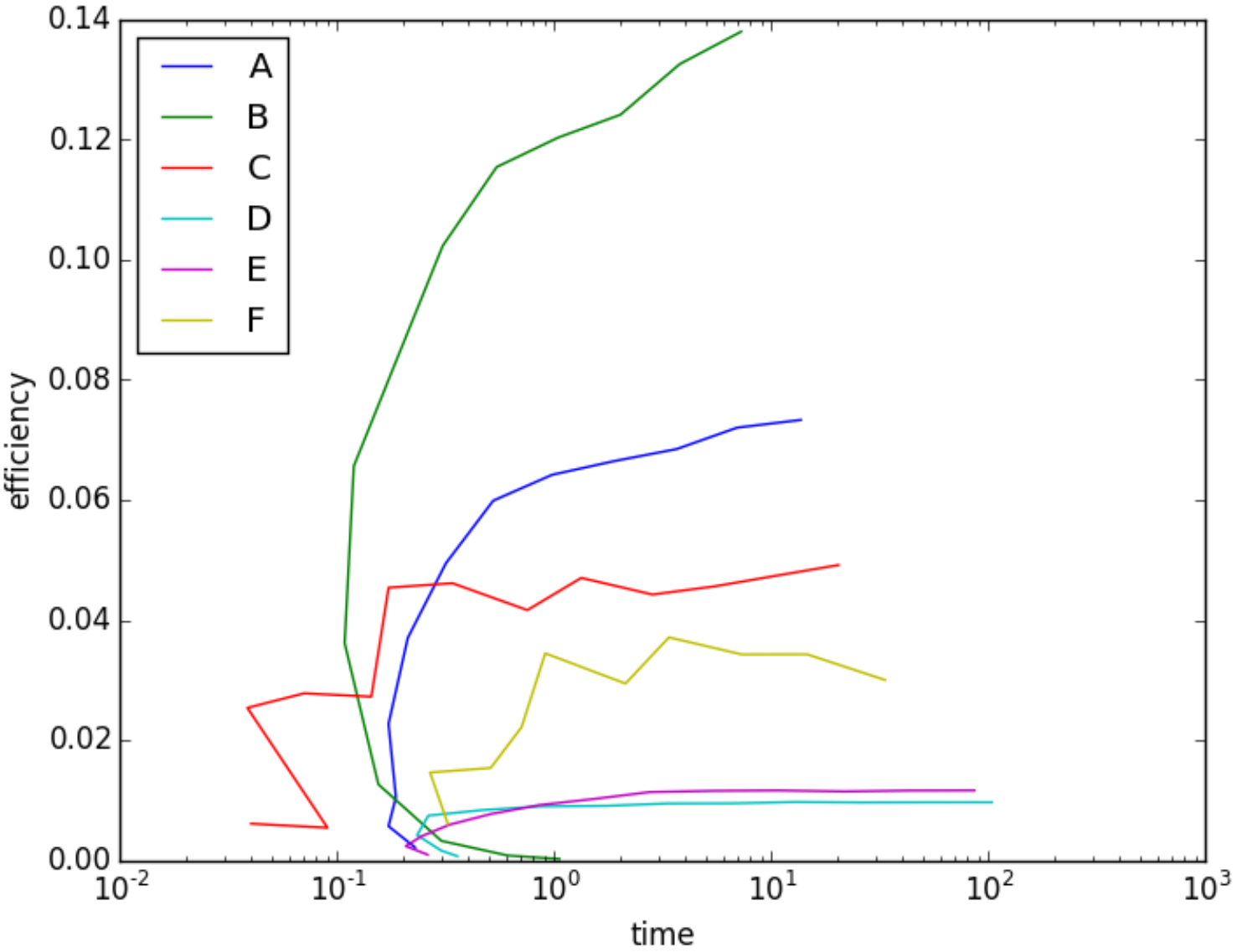
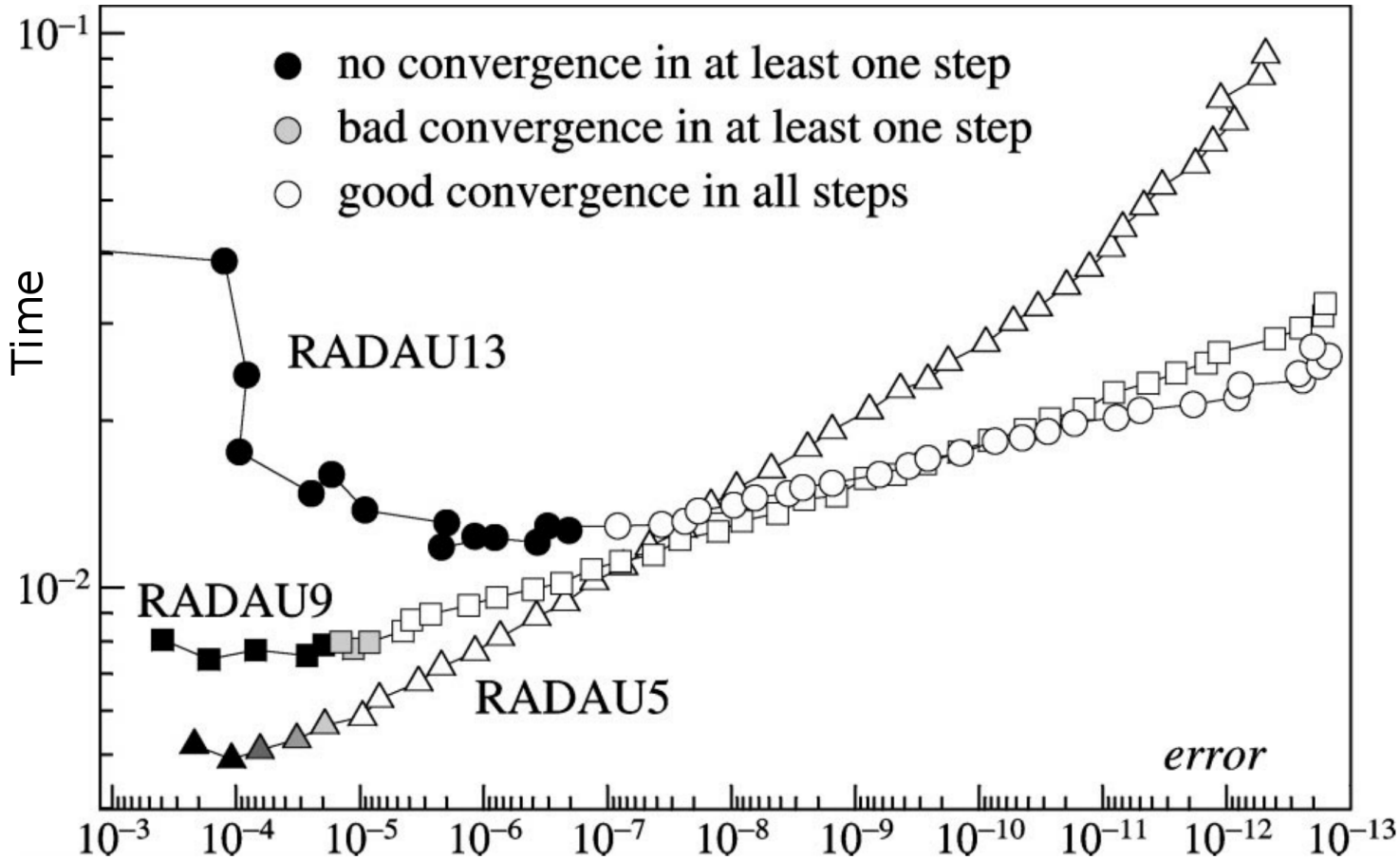
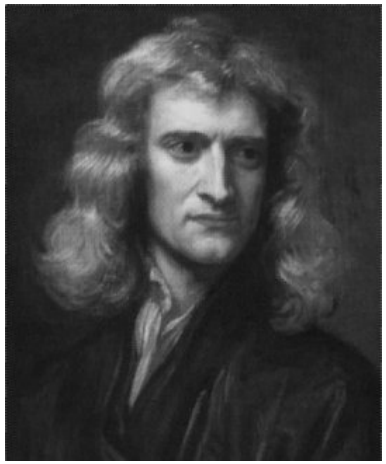
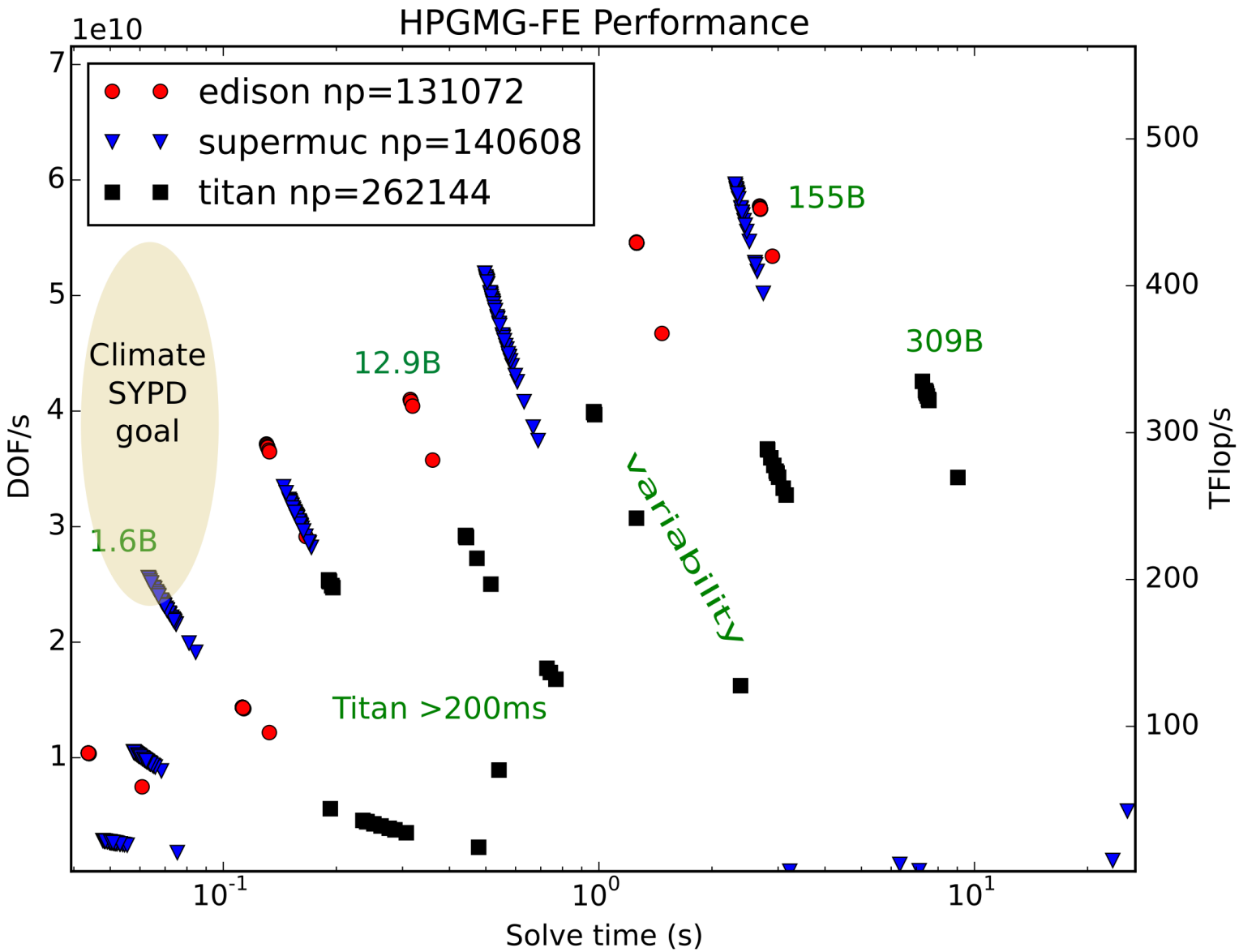
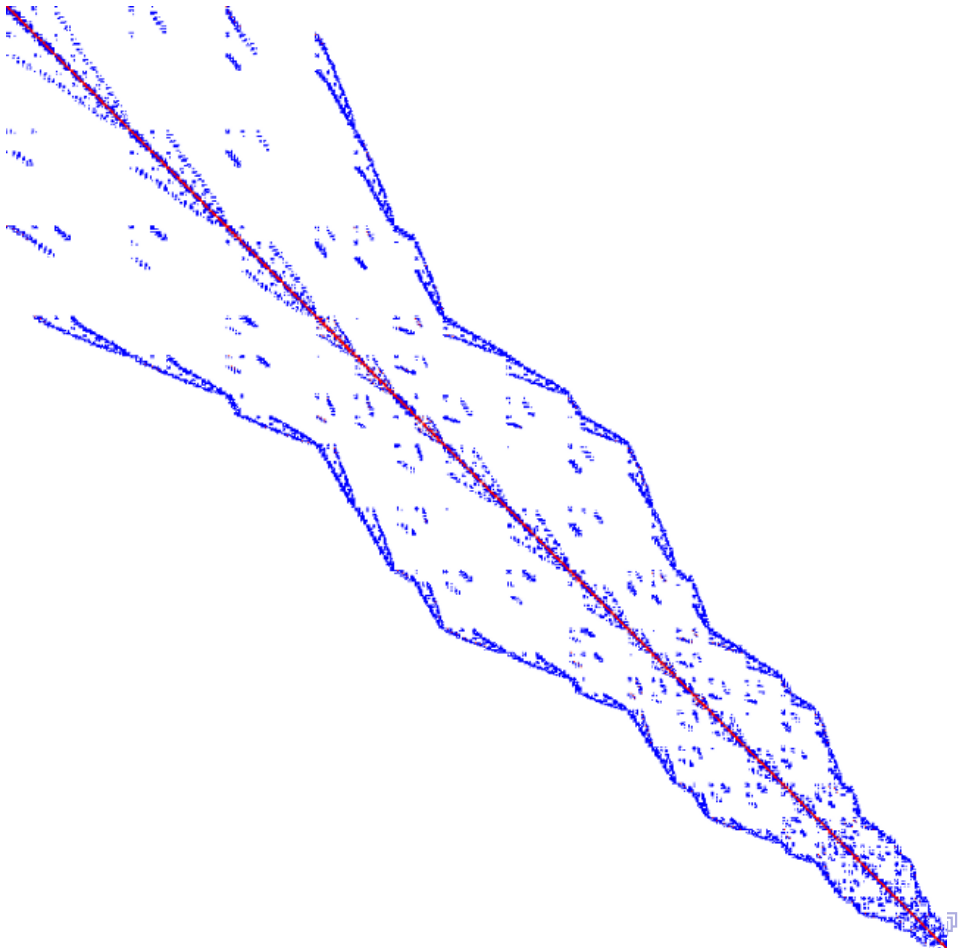
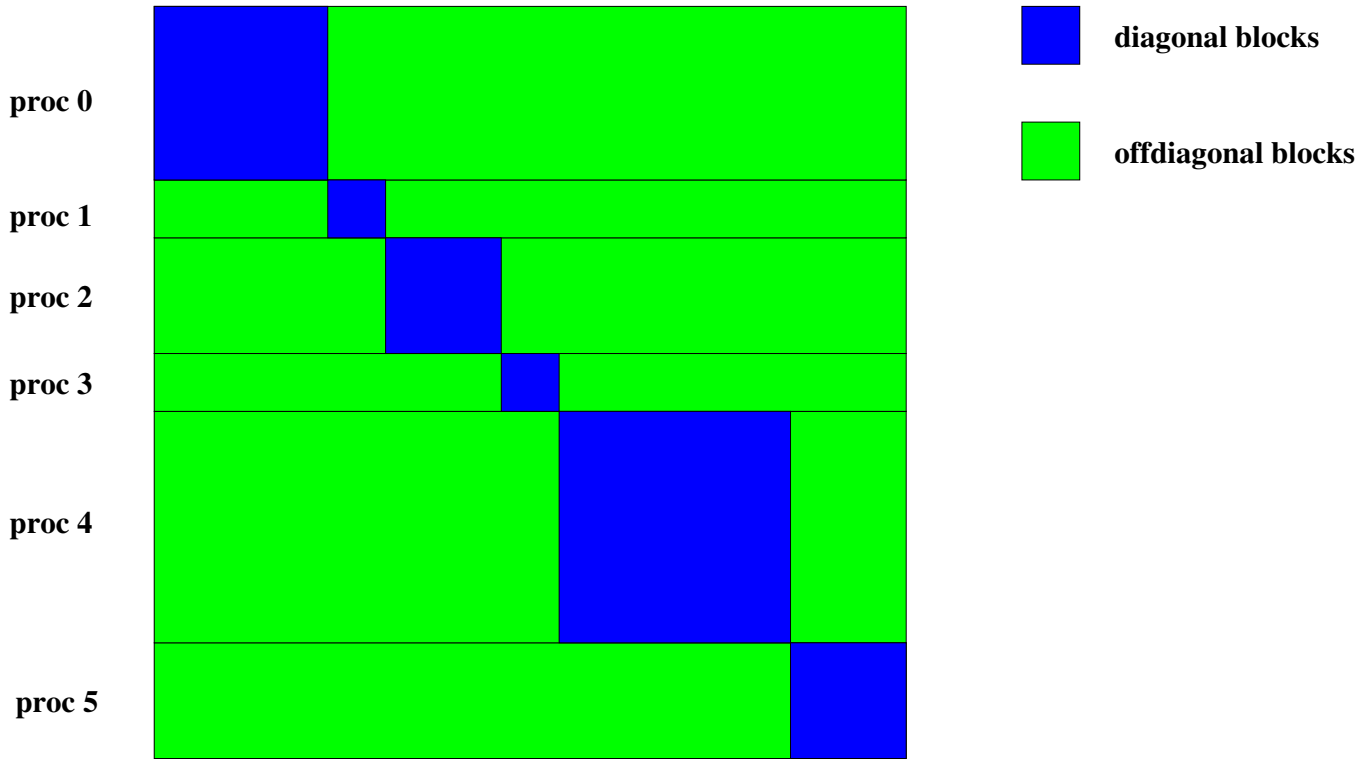
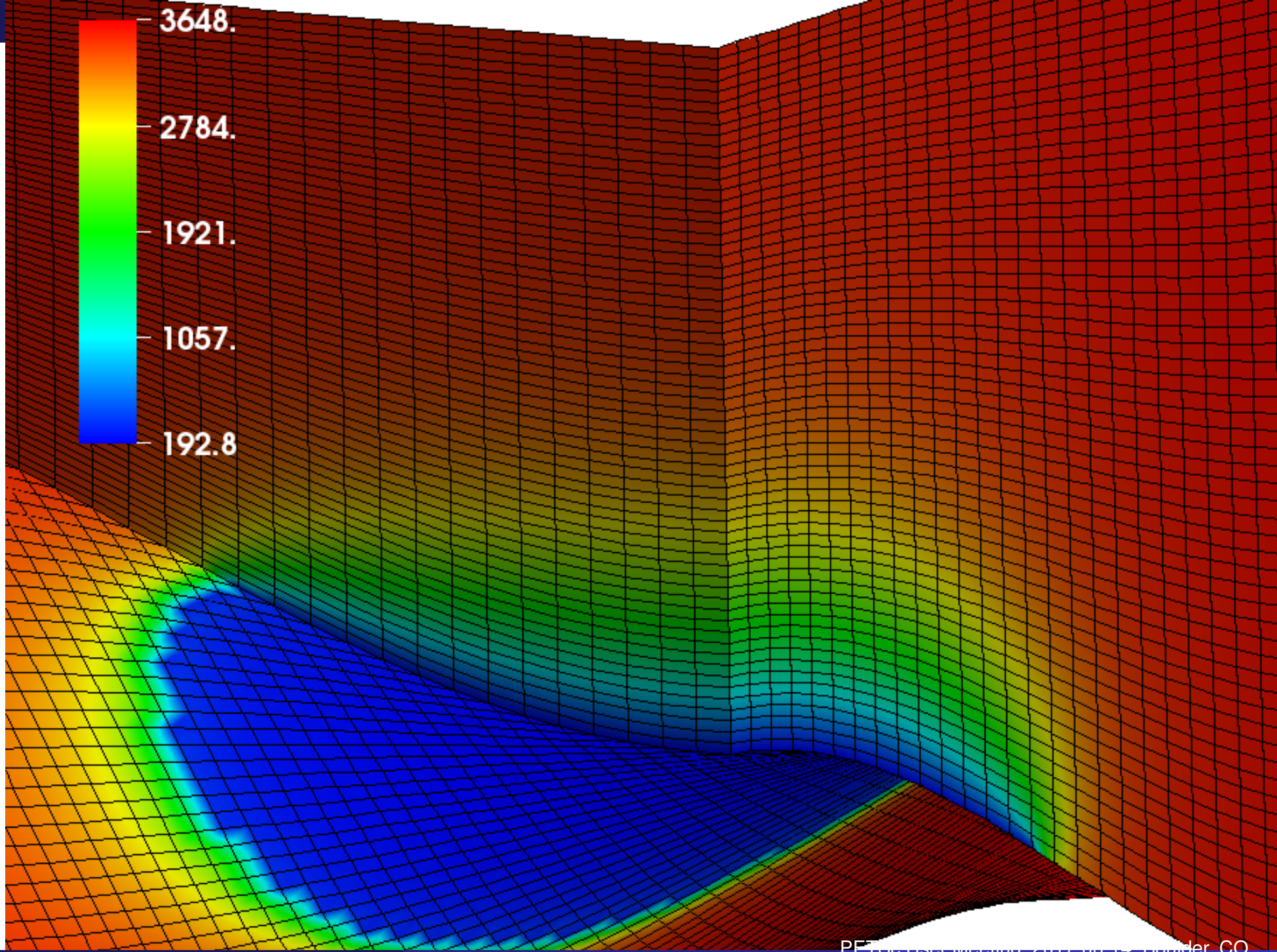
Outline

Ways to set options

- Command line
- Filename in the third argument of `PetscInitialize()`
- `~/.petscrc`
- `$PWD/.petscrc`
- `$PWD/petscrc`
- `PetscOptionsInsertFile()`
- `PetscOptionsInsertString()`
- `PETSC_OPTIONS` environment variable
- command line option `-options_file [file]`

Try it out

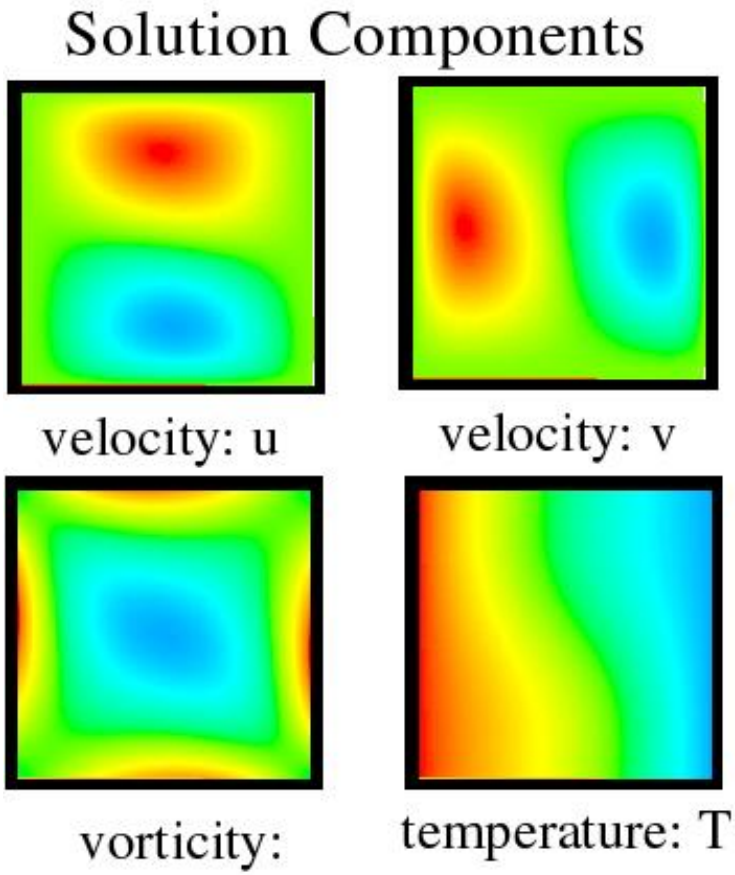
```
$ cd $PETSC_DIR/src/snes/examples/tutorials && make ex5
```

- ```
  $ ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7 \
  -snes_monitor -{ksp,snes}_converged_reason \
  -snes_view
  ```
- ```
  $ ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7 \
  -snes_monitor -{ksp,snes}_converged_reason \
  -snes_view -ksp_view_mat draw -draw_pause -1
  ```
- ```
  $ ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7 \
  -snes_monitor -{ksp,snes}_converged_reason \
  -snes_view -pc_type lu \
  -pc_factor_mat_ordering_type natural
  ```
- Use `-help` to find other ordering types
  ```

Sample output

```
0 SNES Function norm 1.139460779565e+00
Linear solve converged due to CONVERGED_RTOL iterations 1
1 SNES Function norm 4.144493702305e-02
Linear solve converged due to CONVERGED_RTOL iterations 1
2 SNES Function norm 6.309075568032e-03
Linear solve converged due to CONVERGED_RTOL iterations 1
3 SNES Function norm 3.359792279909e-04
Linear solve converged due to CONVERGED_RTOL iterations 1
4 SNES Function norm 1.198827244256e-06
Linear solve converged due to CONVERGED_RTOL iterations 1
```



Sample output (SNES and KSP)

```
SNES Object: 1 MPI processes
  type: ls
    line search variant: CUBIC
    alpha=1.000000000000e-04, maxstep=1.000000000000e+08, minlambd
    damping factor=1.000000000000e+00
  maximum iterations=50, maximum function evaluations=10000
  tolerances: relative=1e-08, absolute=1e-50, solution=1e-08
  total number of linear solver iterations=5
  total number of function evaluations=6
  KSP Object:   1 MPI processes
    type: gmres
      GMRES: restart=30, using Classical (unmodified) Gram-Schmidt
      GMRES: happy breakdown tolerance 1e-30
    maximum iterations=10000, initial guess is zero
    tolerances:  relative=1e-05, absolute=1e-50, divergence=10000
    left preconditioning
    using PRECONDITIONED norm type for convergence test
```

Sample output (PC and Mat)

```
PC Object:   1 MPI processes
  type: lu
    LU: out-of-place factorization
    tolerance for zero pivot 2.22045e-14
    matrix ordering: nd
    factor fill ratio given 5, needed 2.95217
      Factored matrix follows:
        Matrix Object:              1 MPI processes
          type: seqaij
          rows=100, cols=100
          package used to perform factorization: petsc
          total: nonzeros=1358, allocated nonzeros=1358
          total number of mallocs used during MatSetValues calls
            not using I-node routines
  linear system matrix = precond matrix:
  Matrix Object:     1 MPI processes
    type: seqaij
    rows=100, cols=100
    total: nonzeros=460, allocated nonzeros=460
    total number of mallocs used during MatSetValues calls =0
      not using I-node routines
```

In parallel

- `$ mpiexec -n 4 ./ex5 \`
  `-da_grid_x 10 -da_grid_y 10 -par 6.7 \`
  `-snes_monitor -{ksp,snes}_converged_reason \`
  `-snes_view -sub_pc_type lu`
- How does the performance change as you
  - vary the number of processes (up to 32 or 64)?
  - increase the problem size?
  - use an inexact subdomain solve?
  - try an overlapping method: `-pc_type asm -pc_asm_overlap 2`
  - simulate a big machine: `-pc_asm_blocks 512`
  - change the Krylov method: `-ksp_type ibcgs`
  - use algebraic multigrid: `-pc_type hypre`
  - use smoothed aggregation multigrid: `-pc_type gamg` or `-pc_type ml`

Outline

Why Parallel?

- Solve a fixed problem faster
- Obtain a more accurate solution in the same amount of time
- Solve a more complicated problem in the same amount of time
- Use more memory than available on one machine

Strong Scaling



Efficiency versus Number of Processes



Efficiency versus Time



Scaling Challenges

The easiest way to make software scalable
is to make it sequentially inefficient.
(Gropp 1999)

- Solver iteration count may increase from
  - increased resolution
  - model parameters (e.g., coefficient contrast/structure)
  - more realistic models (e.g., plasticity)
  - model coupling
- Algorithm may have suboptimal complexity (e.g., direct solver)
- Increasing spatial resolution requires more time steps (usually)
- Implementation/data structures may not scale
- Architectural effects – cache, memory

Accuracy-time tradeoffs: de rigueur in ODE community



[Hairer and Wanner (1999)]

- Tests discretization, adaptivity, algebraic solvers, implementation
- No reference to number of time steps, number of grid points, etc.

Outline

Outline

Newton iteration: workhorse of SNES



- Standard form of a nonlinear system

$$F(u) = 0$$

- Iteration

$$\begin{aligned} \text{Solve:} \quad & J(u)w = -F(u) \\ \text{Update:} \quad & u^+ \leftarrow u + w \end{aligned}$$

- Quadratically convergent near a root: $\left|u^{n+1} - u^*\right| \in \mathscr{O}\left(\left|u^n - u^*\right|^2\right)$
- Picard is the same operation with a different $J(u)$

Example (Nonlinear Poisson)

$$\begin{aligned} F(u) = 0 \quad &\sim \quad -\nabla\cdot\left[(1+u^2)\nabla u\right] - f = 0 \\ J(u)w \quad &\sim \quad -\nabla\cdot\left[(1+u^2)\nabla w + 2uw\nabla u\right] \end{aligned}$$

SNES Paradigm

The SNES interface is based upon callback functions

- `FormFunction()`, set by `SNESSetFunction()`
- `FormJacobian()`, set by `SNESSetJacobian()`

When PETSc needs to evaluate the nonlinear residual $F(x)$,

- Solver calls the **user's** function
- User function gets application state through the `ctx` variable
  - PETSc never sees application data

SNES Function

The user provided function which calculates the nonlinear residual has signature

```
PetscErrorCode (*func)(SNES snes,Vec x,Vec r,void *ctx)
```

- `x`: The current solution
- `r`: The residual
- `ctx`: The user context passed to `SNESSetFunction()`
  - Use this to pass application information, e.g. physical constants

SNES Jacobian

The user provided function that calculates the Jacobian has signature

```
PetscErrorCode (*func)(SNES snes,Vec x,Mat J,
                       Mat Jpre,void *ctx)
```

`x`: The current solution

`J`: The Jacobian

`Jpre`: The Jacobian preconditioning matrix (possibly J itself)

`ctx`: The user context passed to `SNESSetFunction()`

- Use this to pass application information, e.g. physical constants

Alternatively, you can use

- a builtin sparse finite difference approximation (“coloring”)

Outline

Matrices

Definition (Matrix)

A matrix is a linear transformation between finite dimensional vector spaces.

Definition (Forming a matrix)

Forming or assembling a matrix means defining it's action in terms of entries (usually stored in a sparse format).

Matrices

Definition (Matrix)

A matrix is a linear transformation between finite dimensional vector spaces.

Definition (Forming a matrix)

Forming or assembling a matrix means defining it's action in terms of entries (usually stored in a sparse format).

Important matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of anything interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\varepsilon \to 0} \frac{F(x+\varepsilon y) - F(x)}{\varepsilon}$
4. Fourier transform $\mathscr{F}, \mathscr{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

Important matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of anything interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\varepsilon \to 0} \frac{F(x+\varepsilon y) - F(x)}{\varepsilon}$
4. Fourier transform $\mathscr{F}, \mathscr{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

   - These matrices are dense. Never form them.

Important matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of anything interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\varepsilon \to 0} \frac{F(x+\varepsilon y) - F(x)}{\varepsilon}$
4. Fourier transform $\mathscr{F}, \mathscr{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

- These are not very sparse. Don't form them.

Important matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of anything interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\varepsilon \to 0} \frac{F(x+\varepsilon y) - F(x)}{\varepsilon}$
4. Fourier transform $\mathscr{F}, \mathscr{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

- None of these matrices "have entries"

What can we do with a matrix that doesn't have entries?

Krylov solvers for $Ax = b$

- Krylov subspace: $\{b, Ab, A^2b, A^3b, \dots\}$
- Convergence rate depends on the spectral properties of the matrix
  - Existance of small polynomials $p_n(A) < \varepsilon$ where $p_n(0) = 1$.
  - condition number $\kappa(A) = \|A\| \left\|A^{-1}\right\| = \sigma_{\max}/\sigma_{\min}$
  - distribution of singular values, spectrum Λ, pseudospectrum Λ_ε
- For any popular Krylov method $\mathscr{K}$, there is a matrix of size m, such that $\mathscr{K}$ outperforms all other methods by a factor at least $\mathscr{O}(\sqrt{m})$ [Nachtigal et. al., 1992]

Typically...

- The action $y \leftarrow Ax$ can be computed in $\mathscr{O}(m)$
- Aside from matrix multiply, the n^{th} iteration requires at most $\mathscr{O}(mn)$

GMRES

Brute force minimization of residual in $\{b, Ab, A^2b, \dots\}$

1. Use Arnoldi to orthogonalize the nth subspace, producing

$$AQ_n = Q_{n+1}H_n$$

2. Minimize residual in this space by solving the overdetermined system

$$H_n y_n = e_1^{(n+1)}$$

using QR-decomposition, updated cheaply at each iteration.

Properties

- Converges in n steps for all right hand sides if there exists a polynomial of degree n such that $\|p_n(A)\| < tol$ and $p_n(0) = 1$.
- Residual is monotonically decreasing, robust in practice
- Restarted variants are used to bound memory requirements

Preconditioning

Idea: improve the conditioning of the Krylov operator

- Left preconditioning

$$(P^{-1}A)x = P^{-1}b$$

$$\{P^{-1}b, (P^{-1}A)P^{-1}b, (P^{-1}A)^2P^{-1}b, \dots\}$$

- Right preconditioning

$$(AP^{-1})Px = b$$

$$\{b, (AP^{-1}b, (AP^{-1})^2b, \dots\}$$

- The product $P^{-1}A$ or AP^{-1} is not formed.

Definition (Preconditioner)

A preconditioner $\mathscr{P}$ is a method for constructing a matrix (just a linear function, not assembled!) $P^{-1} = \mathscr{P}(A, A_p)$ using a matrix A and extra information A_p, such that the spectrum of $P^{-1}A$ (or AP^{-1}) is well-behaved.

Preconditioning

Definition (Preconditioner)

A preconditioner $\mathscr{P}$ is a method for constructing a matrix $P^{-1} = \mathscr{P}(A, A_p)$ using a matrix A and extra information A_p, such that the spectrum of $P^{-1}A$ (or AP^{-1}) is well-behaved.

- P^{-1} is dense, P is often not available and is not needed
- A is rarely used by $\mathscr{P}$, but $A_p = A$ is common
- A_p is often a sparse matrix, the "preconditioning matrix"
- Matrix-based: Jacobi, Gauss-Seidel, SOR, ILU(k), LU
- Parallel: Block-Jacobi, Schwarz, Multigrid, FETI-DP, BDDC
- Indefinite: Schur-complement, Domain Decomposition, Multigrid

Questions to ask when you see a matrix

1. What do you want to do with it?
   - Multiply with a vector
   - Solve linear systems or eigen-problems
2. How is the conditioning/spectrum?
   - distinct/clustered eigen/singular values?
   - symmetric positive definite ($\sigma(A) \subset \mathbb{R}^+$)?
   - nonsymmetric definite ($\sigma(A) \subset \{z \in \mathbb{C} : \Re[z] > 0\}$)?
   - indefinite?
3. How dense is it?
   - block/banded diagonal?
   - sparse unstructured?
   - denser than we'd like?
4. Is there a better way to compute Ax?
5. Is there a different matrix with similar spectrum, but nicer properties?
6. How can we precondition A?

Questions to ask when you see a matrix

1. What do you want to do with it?
   - Multiply with a vector
   - Solve linear systems or eigen-problems
2. How is the conditioning/spectrum?
   - distinct/clustered eigen/singular values?
   - symmetric positive definite ($\sigma(A) \subset \mathbb{R}^+$)?
   - nonsymmetric definite ($\sigma(A) \subset \{z \in \mathbb{C} : \Re[z] > 0\}$)?
   - indefinite?
3. How dense is it?
   - block/banded diagonal?
   - sparse unstructured?
   - denser than we'd like?
4. Is there a better way to compute Ax?
5. Is there a different matrix with similar spectrum, but nicer properties?
6. How can we precondition A?

Relaxation

Split into lower, diagonal, upper parts: $A = L + D + U$

Jacobi `-pc_type jacobi`

Cheapest preconditioner: $P^{-1} = D^{-1}$

Successive over-relaxation (SOR) `-pc_type sor`

$$\left(L + \frac{1}{\omega} D\right) x_{n+1} = \left[\left(\frac{1}{\omega} - 1\right) D - U\right] x_n + \omega b$$

$$P^{-1} = k \text{ iterations starting with } x_0 = 0$$

- Implemented as a sweep
- $\omega = 1$ corresponds to Gauss-Seidel
- Very effective at removing high-frequency components of residual

Preconditioned Richardson convergence

The Richardson iteration for $Ax = b$ with preconditioner ωP^{-1} is

$$x_{n+1} = x_n + \omega P^{-1}(b - Ax_n)$$

- If X_* is a solution $Ax_* = b$ then

$$\underbrace{x_{n+1} - x_*}_{e_{n+1}} = \underbrace{x_n - x_*}_{e_n} - \omega P^{-1} A(\underbrace{x_n - x_*}_{e_n}) = (I - \omega P^{-1} A) e_n$$

$$e_n = (I - \omega P^{-1} A)^n e_0$$

- If $\left\| I - \omega P^{-1} A \right\| < 1$ then we get convergence with any initial guess x_0
- If an eigendecomposition $X \Lambda X^{-1} = I - \omega P^{-1} A$ exists, we would like to choose ω to minimize the maximum eigenvalue magnitude.

Factorization

Two phases

- symbolic factorization: find where fill occurs, only uses sparsity pattern
- numeric factorization: compute factors

LU decomposition

- Ultimate preconditioner
- Expensive, for $m \times m$ sparse matrix with bandwidth b, traditionally requires $\mathcal{O}(mb^2)$ time and $\mathcal{O}(mb)$ space.
  - Bandwidth scales as $m^{\frac{d-1}{d}}$ in d-dimensions
  - Optimal in 2D: $\mathcal{O}(m \cdot \log m)$ space, $\mathcal{O}(m^{3/2})$ time
  - Optimal in 3D: $\mathcal{O}(m^{4/3})$ space, $\mathcal{O}(m^2)$ time
- Symbolic factorization is problematic in parallel

Incomplete LU

- Allow a limited number of levels of fill: ILU(k)
- Only allow fill for entries that exceed threshold: ILUT
- Usually poor scaling in parallel
- No guarantees
- Hierarchical/low-rank representations have potential: STRUMPACK

1-level Domain decomposition

Domain size L, subdomain size H, element size h

Overlapping/Schwarz

- Solve Dirichlet problems on overlapping subdomains
- No overlap: $its \in \mathscr{O}\left(\frac{L}{\sqrt{Hh}}\right)$
- Overlap δ: $its \in \mathscr{O}\left(\frac{L}{\sqrt{H\delta}}\right)$

Neumann-Neumann

- Solve Neumann problems on non-overlapping subdomains
- $its \in \mathscr{O}\left(\frac{L}{H}\left(1+\log\frac{H}{h}\right)\right)$
- Tricky null space issues (floating subdomains)
- Need subdomain matrices, not globally assembled matrix.

- Multilevel variants knock off the leading $\frac{L}{H}$
- Both overlapping and nonoverlapping with this bound

Multigrid

Hierarchy: Interpolation and restriction operators

$$\mathscr{I}^{\uparrow} : X_{\text{coarse}} \to X_{\text{fine}} \qquad \mathscr{I}^{\downarrow} : X_{\text{fine}} \to X_{\text{coarse}}$$

- Geometric: define problem on multiple levels, use grid to compute hierarchy
- Algebraic: define problem only on finest level, use matrix structure to build hierarchy

Galerkin approximation

Assemble this matrix: $A_{\text{coarse}} = \mathscr{I}^{\downarrow} A_{\text{fine}} \mathscr{I}^{\uparrow}$

Application of multigrid preconditioner (V-cycle)

- Apply pre-smoother on fine level (any preconditioner)
- Restrict residual to coarse level with $\mathscr{I}^{\downarrow}$
- Solve on coarse level $A_{\text{coarse}} x = r$
- Interpolate result back to fine level with $\mathscr{I}^{\uparrow}$
- Apply post-smoother on fine level (any preconditioner)

Multigrid convergence properties

- Textbook: $P^{-1}A$ is spectrally equivalent to identity
  - Constant number of iterations to converge up to discretization error
- Most theory applies to SPD systems
  - variable coefficients (e.g. discontinuous): low energy interpolants
  - mesh- and/or physics-induced anisotropy: semi-coarsening/line smoothers
  - complex geometry: difficult to have meaningful coarse levels
- Deeper algorithmic difficulties
  - nonsymmetric (e.g. advection, shallow water, Euler)
  - indefinite (e.g. incompressible flow, Helmholtz)
- Performance considerations
  - Aggressive coarsening is critical in parallel
  - Most theory uses SOR smoothers, ILU often more robust
  - Coarsest level usually solved semi-redundantly with direct solver
- Multilevel Schwarz is essentially the same with different language
  - assume strong smoothers, emphasize aggressive coarsening

Norms

- Krylov subspace: $\{P^{-1}b, (P^{-1}A)P^{-1}b, (P^{-1}A)^2P^{-1}b, \dots\}$
- Subspace needs to contain the solution
  - Diameter of preconditioned connectivity graph
- Need to find the correct linear combination
  - Optimize unpreconditioned residual norm (usually right preconditioning)

$$\|Ax - b\|_2 = \|A(x - x^*)\|_2 = \|x - x^*\|_{A^TA}$$

  - Optimize preconditioned residual norm (usually left preconditioning)

$$\left\|P^{-1}(Ax - b)\right\|_2 = \left\|P^{-1}A(x - x^*)\right\|_2 = \|x - x^*\|_{A^TP^{-T}P^{-1}A}$$

  - Natural norm (conjugate gradients) minimizes $\|x - x^*\|_{P^{-1/2}AP^{-1/2}}$
- Evaluating convergence
  - Preconditioned, unpreconditioned, or natural norm
  - Which one to trust?
  - `-ksp_monitor_true_residual`, `-ksp_norm_type`

Outline

Profiling

- Use `-log_view` for a performance profile
  - Event timing
  - Event flops
  - Memory usage
  - MPI messages
- Call `PetscLogStagePush()` and `PetscLogStagePop()`
  - User can add new stages
- Call `PetscLogEventBegin()` and `PetscLogEventEnd()`
  - User can add new events
- Call `PetscLogFlops()` to include your flops

Reading `-log_view`

-

	Max	Max/Min	Avg	Total
Time (sec):	1.548e+02	1.00122	1.547e+02	
Objects:	1.028e+03	1.00000	1.028e+03	
Flops:	1.519e+10	1.01953	1.505e+10	1.204e+11
Flops/sec:	9.814e+07	1.01829	9.727e+07	7.782e+08
MPI Messages:	8.854e+03	1.00556	8.819e+03	7.055e+04
MPI Message Lengths:	1.936e+08	1.00950	2.185e+04	1.541e+09
MPI Reductions:	2.799e+03	1.00000		

- Also a summary per stage
- Memory usage per stage (based on when it was allocated)
- Time, messages, reductions, balance, flops per event per stage
- Always send `-log_view` when asking performance questions on mailing list

Reading `-log_view`

```
Event                Count      Time (sec)     Flops                             --- Global ---  -
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   Avg len Reduct  %T %F %M %L %R  %
------------------------------------------------------------------------------------------------
--- Event Stage 1: Full solve
VecDot                43 1.0 4.8879e-02 8.3 1.77e+06 1.0 0.0e+00 0.0e+00 4.3e+01  0  0  0  0  0
VecMDot             1747 1.0 1.3021e+00 4.6 8.16e+07 1.0 0.0e+00 0.0e+00 1.7e+03  0  1  0  0 14
VecNorm             3972 1.0 1.5460e+00 2.5 8.48e+07 1.0 0.0e+00 0.0e+00 4.0e+03  0  1  0  0 31
VecScale            3261 1.0 1.6703e-01 1.0 3.38e+07 1.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0
VecScatterBegin     4503 1.0 4.0440e-01 1.0 0.00e+00 0.0 6.1e+07 2.0e+03 0.0e+00  0  0 50 26  0
VecScatterEnd       4503 1.0 2.8207e+00 6.4 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0
MatMult             3001 1.0 3.2634e+01 1.1 3.68e+09 1.1 4.9e+07 2.3e+03 0.0e+00 11 22 40 24  0  2
MatMultAdd           604 1.0 6.0195e-01 1.0 5.66e+07 1.0 3.7e+06 1.3e+02 0.0e+00  0  0  3  0  0
MatMultTranspose     676 1.0 1.3220e+00 1.6 6.50e+07 1.0 4.2e+06 1.4e+02 0.0e+00  0  0  3  0  0
MatSolve            3020 1.0 2.5957e+01 1.0 3.25e+09 1.0 0.0e+00 0.0e+00 0.0e+00  9 21  0  0  0  1
MatCholFctrSym         3 1.0 2.8324e-04 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0
MatCholFctrNum        69 1.0 5.7241e+00 1.0 6.75e+08 1.0 0.0e+00 0.0e+00 0.0e+00  2  4  0  0  0
MatAssemblyBegin     119 1.0 2.8250e+00 1.5 0.00e+00 0.0 2.1e+06 5.4e+04 3.1e+02  1  0  2 24  2
MatAssemblyEnd       119 1.0 1.9689e+00 1.4 0.00e+00 0.0 2.8e+05 1.3e+03 6.8e+01  1  0  0  0  1
SNESSolve              4 1.0 1.4302e+02 1.0 8.11e+09 1.0 6.3e+07 3.8e+03 6.3e+03 51 50 52 50 50  9
SNESLineSearch        43 1.0 1.5116e+01 1.0 1.05e+08 1.1 2.4e+06 3.6e+03 1.8e+02  5  1  2  2  1  1
SNESFunctionEval      55 1.0 1.4930e+01 1.0 0.00e+00 0.0 1.8e+06 3.3e+03 8.0e+00  5  0  1  1  0  1
SNESJacobianEval      43 1.0 3.7077e+01 1.0 7.77e+06 1.0 4.3e+06 2.6e+04 3.0e+02 13  0  4 24  2  2
KSPGMRESOrthog      1747 1.0 1.5737e+00 2.9 1.63e+08 1.0 0.0e+00 0.0e+00 1.7e+03  1  1  0  0 14
KSPSetup             224 1.0 2.1040e-02 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 3.0e+01  0  0  0  0  0
KSPSolve              43 1.0 8.9988e+01 1.0 7.99e+09 1.0 5.6e+07 2.0e+03 5.8e+03 32 49 46 24 46  6
PCSetUp              112 1.0 1.7354e+01 1.0 6.75e+08 1.0 0.0e+00 0.0e+00 8.7e+01  6  4  0  0  1  1
PCSetUpOnBlocks     1208 1.0 5.8182e+00 1.0 6.75e+08 1.0 0.0e+00 0.0e+00 8.7e+01  2  4  0  0  1
PCApply              276 1.0 7.1497e+01 1.0 7.14e+09 1.0 5.2e+07 1.8e+03 5.1e+03 25 44 42 20 41  4
```

Communication Costs

- Reductions: usually part of Krylov method, latency limited
  - `VecDot`
  - `VecMDot`
  - `VecNorm`
  - `MatAssemblyBegin`
  - Change algorithm (e.g. IBCGS, PGMRES)
- Point-to-point (nearest neighbor), latency or bandwidth
  - `VecScatter`
  - `MatMult`
  - `PCApply`
  - `MatAssembly`
  - `SNESFunctionEval`
  - `SNESJacobianEval`
  - Compute subdomain boundary fluxes redundantly
  - Ghost exchange for all fields at once, or overlap
  - Better partition

HPGMG-FE https://hpgmg.org



Outline

Matrices, redux

What are PETSc matrices?

- Linear operators on finite dimensional vector spaces. (snarky)
- Fundamental objects for storing stiffness matrices and Jacobians
- Each process locally owns a contiguous set of rows
- Supports many data types
  - AIJ, Block AIJ, Symmetric AIJ, Block Diagonal, etc.
- Supports structures for many packages
  - MUMPS, SuperLU, UMFPack, Hypre, Elemental

Matrices, redux

What are PETSc matrices?

- Linear operators on finite dimensional vector spaces. (snarky)
- Fundamental objects for storing stiffness matrices and Jacobians
- Each process locally owns a contiguous set of rows
- Supports many data types
  - AIJ, Block AIJ, Symmetric AIJ, Block Diagonal, etc.
- Supports structures for many packages
  - MUMPS, SuperLU, UMFPack, Hypre, Elemental

How do I create matrices?

- `MatCreate(MPI_Comm, Mat *)`
- `MatSetSizes(Mat, int m, int n, int M, int N)`
- `MatSetType(Mat, MatType typeName)`
- `MatSetFromOptions(Mat)`
  - Can set the type at runtime
- `MatSetBlockSize(Mat, int bs)`
  - for vector problems
- `MatXAIJSetPreallocation(Mat,...)`
  - important for assembly performance
- `MatSetValues(Mat,...)`
  - **MUST** be used, but does automatic communication
  - `MatSetValuesLocal()`,`MatSetValuesStencil()`
  - `MatSetValuesBlocked()`

Matrix Polymorphism

The PETSc `Mat` has a single user interface,

- Matrix assembly
  - `MatSetValues()`
- Matrix-vector multiplication
  - `MatMult()`
- Matrix viewing
  - `MatView()`

but multiple underlying implementations.

- AIJ, Block AIJ, Symmetric Block AIJ,
- Dense, Elemental
- Matrix-Free
- etc.

A matrix is defined by its interface, not by its data structure.

Matrix Assembly

- A three step process
  - Each process sets or adds values
  - Begin communication to send values to the correct process
  - Complete the communication
- `MatSetValues(Mat A, m, rows[], n, cols[], values[], mode)`
  - `mode` is either INSERT_VALUES or ADD_VALUES
  - Logically dense block of values
- Two phase assembly allows overlap of communication and computation
  - `MatAssemblyBegin(Mat m, type)`
  - `MatAssemblyEnd(Mat m, type)`
  - `type` is either MAT_FLUSH_ASSEMBLY or MAT_FINAL_ASSEMBLY
- For vector problems

```
MatSetValuesBlocked(Mat A, m, rows[],
        n, cols[], values[], mode)
```

- The same assembly code can build matrices of different format
  - choose format at run-time.

Matrix Assembly

- A three step process
  - Each process sets or adds values
  - Begin communication to send values to the correct process
  - Complete the communication
- `MatSetValues(Mat A, m, rows[], n, cols[], values[], mode)`
  - `mode` is either INSERT_VALUES or ADD_VALUES
  - Logically dense block of values
- Two phase assembly allows overlap of communication and computation
  - `MatAssemblyBegin(Mat m, type)`
  - `MatAssemblyEnd(Mat m, type)`
  - `type` is either MAT_FLUSH_ASSEMBLY or MAT_FINAL_ASSEMBLY
- For vector problems

```
MatSetValuesBlocked(Mat A, m, rows[],
        n, cols[], values[], mode)
```

- The same assembly code can build matrices of different format
  - choose format at run-time.

A Scalable Way to Set the Elements of a Matrix

Simple 3-point stencil for 1D Laplacian

```
v[0] = -1.0; v[1] = 2.0; v[2] = -1.0;
for (row = start;  row < end; row++) {
  cols[0] = row-1; cols[1] = row; cols[2] = row+1;
  if (row == 0) {
    MatSetValues(A,1,&row,2,&cols[1],&v[1],INSERT_VALUES);
  } else if (row == N-1) {
    MatSetValues(A,1,&row,2,cols,v,INSERT_VALUES);
  } else {
    MatSetValues(A,1,&row,3,cols,v,INSERT_VALUES);
  }
}
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);
```

Why Are PETSc Matrices That Way?

- No one data structure is appropriate for all problems
  - Blocked and diagonal formats provide significant performance benefits
  - PETSc has many formats and makes it easy to add new data structures
- Assembly is difficult enough without worrying about partitioning
  - PETSc provides parallel assembly routines
  - Achieving high performance still requires making most operations local
  - However, programs can be incrementally developed.
  - `MatPartitioning` and `MatOrdering` can help
- Matrix decomposition in contiguous chunks is simple
  - Makes interoperation with other codes easier
  - For other ordering, PETSc provides “Application Orderings” (AO)

Approximating condition numbers

- Small matrices:
  ```
  -pc_type svd -pc_svd_monitor
  ```
- Large matrices (avoid restarts!):
  ```
  -pc_type none -ksp_type gmres \
      -ksp_monitor_singular_value \
      -ksp_gmres_restart 1000
  ```
- Condition of preconditioned operator:
  ```
  -pc_type some_pc -ksp_type gmres \
      -ksp_monitor_singular_value \
      -ksp_gmres_restart 1000
  ```

Try these:

- `$ cd $PETSC_DIR/src/ksp/ksp/examples/tutorials \ && make ex2`
- `$ ./ex2 -m 20 -n 20 <other_options>`

Preliminary Conclusions

PETSc can help you

- solve algebraic and DAE problems in your application area
- rapidly develop efficient parallel code, can start from examples
- develop new solution methods and data structures
- debug and analyze performance
- advice on software design, solution algorithms, and performance
  - Public questions: `petsc-users@mcs.anl.gov`, archived
  - Private questions: `petsc-maint@mcs.anl.gov`, not archived

You can help PETSc

- report bugs and inconsistencies, or if you think there is a better way
- tell us if the documentation is inconsistent or unclear
- consider developing new algebraic methods as plugins, contribute if your idea works

Outline

Application Integration

- Be willing to experiment with algorithms
  - No optimality without interplay between physics and algorithmics
- Adopt flexible, extensible programming
  - Algorithms and data structures not hardwired
- Be willing to play with the real code
  - Toy models have limited usefulness
  - But make test cases that run quickly
- If possible, profile before integration
  - Automatic in PETSc

Incorporating PETSc into existing codes

- PETSc does not seize `main()`, does not control output
- Propogates errors from underlying packages, flexible error handling
- Nothing special about `MPI_COMM_WORLD`
- Can wrap existing data structures/algorithms
  - `MatShell`, `PCShell`, full implementations
  - `VecCreateMPIWithArray()`
  - `MatCreateSeqAIJWithArrays()`
  - Use an existing semi-implicit solver as a preconditioner
  - Usually worthwhile to use native PETSc data structures unless you have a good reason not to
- Uniform interfaces across languages
  - C, C++, Fortran 77/90, Python, MATLAB
- Do not have to use high level interfaces (e.g. SNES, TS, DM)
  - but PETSc can offer more if you do, like MFFD and SNES Test

Integration Stages

- Version Control
  - It is impossible to overemphasize
- Initialization
  - Linking to PETSc
- Profiling
  - Profile before changing
  - Also incorporate command line processing
- Linear Algebra
  - First PETSc data structures
- Solvers
  - Very easy after linear algebra is integrated

Initialization

- Call `PetscInitialize()`
  - Setup static data and services
  - Setup MPI if it is not already
  - Can set `PETSC_COMM_WORLD` to use your communicator (can always use subcommunicators for each object)
- Call `PetscFinalize()`
  - Calculates logging summary
  - Can check for leaks/unused options
  - Shutdown and release resources
- Can only initialize PETSc once

Matrix Memory Preallocation

- PETSc sparse matrices are dynamic data structures
  - can add additional nonzeros freely
- Dynamically adding many nonzeros
  - requires additional memory allocations
  - requires copies
  - can kill performance
- Memory preallocation provides
  - the freedom of dynamic data structures
  - good performance
- Easiest solution is to replicate the assembly code
  - Remove computation, but preserve the indexing code
  - Store set of columns for each row
- Call preallocation routines for all datatypes
  - `MatSeqAIJSetPreallocation()`
  - `MatMPIBAIJSetPreallocation()`
  - Only the relevant data will be used. Or `MatXAIJSetPreallocation()`

Sequential Sparse Matrices

`MatSeqAIJSetPreallocation(Mat A, int nz, int nnz[])`

nz: expected number of nonzeros in any row

nnz[i]: expected number of nonzeros in row i



Parallel Sparse Matrix

- Each process locally owns a submatrix of contiguous global rows
- Each submatrix consists of diagonal and off-diagonal parts



- `MatGetOwnershipRange(Mat A,int *start,int *end)`

`start`: first locally owned row of global matrix

`end-1`: last locally owned row of global matrix

Parallel Sparse Matrices

```
MatMPIAIJSetPreallocation(Mat A, int dnz, int
dnnz[],
int onz, int onnz[])
```

dnz: expected number of nonzeros in any row in the diagonal block

dnnz[i]: expected number of nonzeros in row i in the diagonal block

onz: expected number of nonzeros in any row in the offdiagonal portion

onnz[i]: expected number of nonzeros in row i in the offdiagonal portion

Verifying Preallocation

- Use runtime options
  `-mat_new_nonzero_location_err`
  `-mat_new_nonzero_allocation_err`
- Use `-ksp_view` or `-snes_view` and look for
  `total number of mallocs used during MatSetValues calls =0`
- Use runtime option `-info`
- Output:
  `[proc #] Matrix size: %d X %d; storage space: %d unneeded, %d used`
  `[proc #] Number of mallocs during MatSetValues( ) is %d`

```
[merlin] mpirun ex2 -log_info
[0]MatAssemblyEnd_SeqAIJ:Matrix size: 56 X 56; storage space:
[0]   310 unneeded, 250 used
[0]MatAssemblyEnd_SeqAIJ:Number of mallocs during MatSetValues() is 0
```

Block and symmetric formats

- BAIJ
  - Like AIJ, but uses static block size
  - Preallocation is like AIJ, but just one index per block
- SBAIJ
  - Only stores upper triangular part
  - Preallocation needs number of nonzeros in upper triangular parts of on- and off-diagonal blocks
- `MatSetValuesBlocked()`
  - Better performance with blocked formats
  - Also works with scalar formats, if `MatSetBlockSize()` was called
  - Variants `MatSetValuesBlockedLocal()`, `MatSetValuesBlockedStencil()`
  - Change matrix format at runtime, don't need to touch assembly code

Linear Solvers

Krylov Methods

- Using PETSc linear algebra, just add:
  - `KSPSetOperators(KSP ksp, Mat A, Mat Pmat)`
  - `KSPSolve(KSP ksp, Vec b, Vec x)`
- Can access subobjects
  - `KSPGetPC(KSP ksp, PC *pc)`
- Preconditioners must obey PETSc interface
  - Basically just the KSP interface
- Can change solver dynamically from the command line, `-ksp_type`

Nonlinear Solvers

Newton and Picard Methods

- Using PETSc linear algebra, just add:
  - `SNESSetFunction(SNES snes, Vec r, residualFunc, void *ctx)`
  - `SNESSetJacobian(SNES snes, Mat A, Mat M, jacFunc, void *ctx)`
  - `SNESSolve(SNES snes, Vec b, Vec x)`
- Can access subobjects
  - `SNESGetKSP(SNES snes, KSP *ksp)`
- Can customize subobjects from the cmd line
  - Set the subdomain preconditioner to ILU with `-sub_pc_type ilu`

Outline

Outline

Hydrostatic equations for ice sheet flow

- Valid when $w_x \ll u_z$, independent of basal friction (Schoof&Hindmarsh 2010)
- Eliminate p and w from Stokes by incompressibility:

  3D elliptic system for $\boldsymbol{u} = (u, v)$

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

$$\eta(\theta, \gamma) = \frac{B(\theta)}{2} (\gamma_0 + \gamma)^{\frac{1-n}{2n}}, \qquad n \approx 3$$

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

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

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

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

- Q_1 FEM with Newton-Krylov-Multigrid solver in PETSc:
  `src/snes/examples/tutorials/ex48.c`


3648.
2784.
1921.
1057.
192.8

Some Multigrid Options

- `-snes_grid_sequence`: [0]
  Solve nonlinear problems on coarse grids to get initial guess
- `-pc_mg_galerkin`: [FALSE]
  Use Galerkin process to compute coarser operators
- `-pc_mg_type`: [FULL]
  (choose one of) MULTIPLICATIVE ADDITIVE FULL KASKADE
- `-mg_coarse_{ksp,pc}_*`
  control the coarse-level solver
- `-mg_levels_{ksp,pc}_*`
  control the smoothers on levels
- `-mg_levels_3_{ksp,pc}_*`
  control the smoother on specific level
- These also work with ML's algebraic multigrid.

What is this doing?

- ```
  mpiexec -n 4 ./ex48 -M 16 -P 2 \
  -da_refine_hierarchy_x 1,8,8 \
  -da_refine_hierarchy_y 2,1,1 -da_refine_hierarchy_z 2,1,1 \
  -snes_grid_sequence 1 -log_view \
  -ksp_converged_reason -ksp_gmres_modifiedgramschmidt \
  -ksp_monitor -ksp_rtol 1e-2 \
  -pc_mg_type multiplicative \
  -mg_coarse_pc_type lu -mg_levels_0_pc_type lu \
  -mg_coarse_pc_factor_mat_solver_package mumps \
  -mg_levels_0_pc_factor_mat_solver_package mumps \
  -mg_levels_1_sub_pc_type cholesky \
  -snes_converged_reason -snes_monitor -snes_stol 1e-12 \
  -thi_L 80e3 -thi_alpha 0.05 -thi_friction_m 0.3 \
  -thi_hom x -thi_nlevels 4
  ```
- What happens if you remove `-snes_grid_sequence`?
- What about solving with block Jacobi, ASM, or algebraic multigrid?
  ```

Outline

SNES Example

Driven Cavity



- Velocity-vorticity formulation
- Flow driven by lid and/or bouyancy
- Logically regular grid
  - Parallelized with `DMDA`
- Finite difference discretization
- Authored by David Keyes

`src/snes/examples/tutorials/ex19.c`

SNES Example

Driven Cavity Application Context

```
/* Collocated at each node */
typedef struct {
  PetscScalar u,v,omega,temp;
} Field;

typedef struct {
       /* physical parameters */
   PetscReal lidvelocity,prandtl,grashof;
       /* color plots of the solution */
   PetscBool  draw_contours;
} AppCtx;
```

SNES Example

```
DrivenCavityFunction(SNES snes, Vec X, Vec F, void *ptr) {
  AppCtx         *user = (AppCtx *) ptr;
  /* local starting and ending grid points */
  PetscInt       istart, iend, jstart, jend;
  PetscScalar    *f;                /* local vector data */
  PetscReal      grashof = user->grashof;
  PetscReal      prandtl = user->prandtl;
  PetscErrorCode ierr;

  /* Code to communicate nonlocal ghost point data */
  DMDAVecGetArray(da, F, &f);

  /* Loop over local part and assemble into f[idxloc] */
  /* .... */

  DMDAVecRestoreArray(da, F, &f);
  return 0;
}
```

SNES Example with local evaluation

```
PetscErrorCode DrivenCavityFuncLocal(DMDALocalInfo *info,
                     Field **x,Field **f,void *ctx) {
  /* Handle boundaries ... */
  /* Compute over the interior points */
  for(j = info->ys; j < info->ys+info->ym; j++) {
    for(i = info->xs; i < info->xs+info->xm; i++) {
      /* convective coefficients for upwinding ... */
      /* U velocity */
      u           = x[j][i].u;
      uxx         = (2.0*u - x[j][i-1].u - x[j][i+1].u)*hydhx;
      uyy         = (2.0*u - x[j-1][i].u - x[j+1][i].u)*hxdhy;
      f[j][i].u   = uxx + uyy - .5*(x[j+1][i].omega-x[j-1][i].omega
      /* V velocity, Omega ... */
      /* Temperature */
      u           = x[j][i].temp;
      uxx         = (2.0*u - x[j][i-1].temp - x[j][i+1].temp)*hy
      uyy         = (2.0*u - x[j-1][i].temp - x[j+1][i].temp)*hx
      f[j][i].temp =  uxx + uyy + prandtl
        * (  (vxp*(u - x[j][i-1].temp) + vxm*(x[j][i+1].temp - u))
           + (vyp*(u - x[j-1][i].temp) + vym*(x[j+1][i].temp - u))
```

Running the driven cavity

- `./ex19 -lidvelocity 100 -grashof 1e2 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2`
- `./ex19 -lidvelocity 100 -grashof 1e4 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2`
- `./ex19 -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2 -pc_type lu`
- Uh oh, we have convergence problems
- Does `-snes_grid_sequence` help?

Running the driven cavity

- `./ex19 -lidvelocity 100 -grashof 1e2 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2`

```
lid velocity = 100, prandtl # = 1, grashof # = 1000
0 SNES Function norm 7.682893957872e+02
1 SNES Function norm 6.574700998832e+02
2 SNES Function norm 5.285205210713e+02
3 SNES Function norm 3.770968117421e+02
4 SNES Function norm 3.030010490879e+02
5 SNES Function norm 2.655764576535e+00
6 SNES Function norm 6.208275817215e-03
7 SNES Function norm 1.191107243692e-07
Number of SNES iterations = 7
```

- `./ex19 -lidvelocity 100 -grashof 1e4 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2`
- `./ex19 -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2 -pc_type lu`
- Uh oh, we have convergence problems
- Does `-snes_grid_sequence` help?

Running the driven cavity

- `./ex19 -lidvelocity 100 -grashof 1e2 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2`
- `./ex19 -lidvelocity 100 -grashof 1e4 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2`

```
lid velocity = 100, prandtl # = 1, grashof # = 10000
0 SNES Function norm 7.854040793765e+02
1 SNES Function norm 6.630545177472e+02
2 SNES Function norm 5.195829874590e+02
3 SNES Function norm 3.608696664876e+02
4 SNES Function norm 2.458925075918e+02
5 SNES Function norm 1.811699413098e+00
6 SNES Function norm 4.688284580389e-03
7 SNES Function norm 4.417003604737e-08
Number of SNES iterations = 7
```

- `./ex19 -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2 -pc_type lu`
- Uh oh, we have convergence problems
- Does `-snes_grid_sequence` help?

Running the driven cavity

- `./ex19 -lidvelocity 100 -grashof 1e2 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2`
- `./ex19 -lidvelocity 100 -grashof 1e4 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2`
- `./ex19 -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2 -pc_type lu`

```
lid velocity = 100, prandtl # = 1, grashof # = 100000
0 SNES Function norm 1.809960438828e+03
1 SNES Function norm 1.678372489097e+03
2 SNES Function norm 1.643759853387e+03
3 SNES Function norm 1.559341161485e+03
4 SNES Function norm 1.557604282019e+03
5 SNES Function norm 1.510711246849e+03
6 SNES Function norm 1.500472491343e+03
7 SNES Function norm 1.498930951680e+03
8 SNES Function norm 1.498440256659e+03
...
```

- Uh oh, we have convergence problems
- Does `-snes_grid_sequence` help?

Running the driven cavity

- `./ex19 -lidvelocity 100 -grashof 1e2 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2`
- `./ex19 -lidvelocity 100 -grashof 1e4 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2`
- `./ex19 -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -snes_monitor -snes_view -da_refine 2 -pc_type lu`
- Uh oh, we have convergence problems
- Does `-snes_grid_sequence` help?

Why isn't SNES converging?

- The Jacobian is wrong (maybe only in parallel)
  - Check with `-snes_compare_explicit` and `-snes_mf_operator -pc_type lu`
- The linear system is not solved accurately enough
  - Check with `-pc_type lu`
  - Check `-ksp_monitor_true_residual`, try right preconditioning
- The Jacobian is singular with inconsistent right side
  - Use `MatNullSpace` to inform the `KSP` of a known null space
  - Use a different Krylov method or preconditioner
- The nonlinearity is just really strong
  - Run with `-snes_linesearch_monitor`
  - Try using trust region instead of line search `-snes_type newtontr`
  - Try grid sequencing if possible
  - Use a continuation

Globalizing the lid-driven cavity

- Pseudotransient continuation (Ψtc)
  - Do linearly implicit backward-Euler steps, driven by steady-state residual
  - Residual-based adaptive controller retains quadratic convergence in terminal phase
- Implemented in `src/ts/examples/tutorials/ex26.c`
- `$ ./ex26 -ts_type pseudo -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -ts_monitor`
- Make the method nonlinearly implicit: `-snes_type newtonls -snes_monitor`
  - Compare required number of linear iterations
- Try error-based adaptivity: `-ts_type rosw -ts_adapt_dt_min 1e-4`
- Try increasing `-lidvelocity`, `-grashof`, and problem size
- Coffey, Kelley, and Keyes, Pseudotransient continuation and differential algebraic equations, SIAM J. Sci. Comp, 2003.

Globalizing the lid-driven cavity

- Pseudotransient continuation (Ψtc)
  - Do linearly implicit backward-Euler steps, driven by steady-state residual
  - Residual-based adaptive controller retains quadratic convergence in terminal phase
- Implemented in `src/ts/examples/tutorials/ex26.c`
- `$ ./ex26 -ts_type pseudo -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -ts_monitor`

```
16x16 grid, lid velocity = 100, prandtl # = 1, grashof # = 100000
0 TS dt 0.03125 time 0
1 TS dt 0.034375 time 0.034375
2 TS dt 0.0398544 time 0.0742294
3 TS dt 0.0446815 time 0.118911
4 TS dt 0.0501182 time 0.169029
...
24 TS dt 3.30306 time 11.2182
25 TS dt 8.24513 time 19.4634
26 TS dt 28.1903 time 47.6537
27 TS dt 371.986 time 419.64
28 TS dt 379837 time 380257
29 TS dt 3.01247e+10 time 3.01251e+10
30 TS dt 6.80049e+14 time 6.80079e+14

CONVERGED_TIME at time 6.80079e+14 after 30 steps
```

- Make the method nonlinearly implicit: `-snes_type newtonls -snes_monitor`
  - Compare required number of linear iterations

Globalizing the lid-driven cavity

- Pseudotransient continuation (Ψtc)
  - Do linearly implicit backward-Euler steps, driven by steady-state residual
  - Residual-based adaptive controller retains quadratic convergence in terminal phase
- Implemented in `src/ts/examples/tutorials/ex26.c`
- `$ ./ex26 -ts_type pseudo -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -ts_monitor`
- Make the method nonlinearly implicit: `-snes_type newtonls -snes_monitor`
  - Compare required number of linear iterations
- Try error-based adaptivity: `-ts_type rosw -ts_adapt_dt_min 1e-4`
- Try increasing `-lidvelocity`, `-grashof`, and problem size
- Coffey, Kelley, and Keyes, Pseudotransient continuation and differential algebraic equations, SIAM J. Sci. Comp, 2003.

Nonlinear multigrid (full approximation scheme)

```
$ cd $PETSC_DIR/src/snes/examples/tutorials
```

- V-cycle structure, but use nonlinear relaxation and skip the matrices
- `./ex19 -da_refine 4 -snes_monitor -snes_type nrichardson -npc_fas_levels_snes_type ngs -npc_fas_levels_snes_gs_sweeps 3 -npc_snes_type fas -npc_snes_max_it 1 -npc_snes_fas_smoothup 6 -npc_snes_fas_smoothdown 6 -lidvelocity 100 -grashof 4e4`
- `./ex19 -da_refine 4 -snes_monitor -snes_type ngmres -npc_fas_levels_snes_type ngs -npc_fas_levels_snes_gs_sweeps 3 -npc_snes_type fas -npc_snes_max_it 1 -npc_snes_fas_smoothup 6 -npc_snes_fas_smoothdown 6 -lidvelocity 100 -grashof 4e4`

Nonlinear multigrid (full approximation scheme)

```
$ cd $PETSC_DIR/src/snes/examples/tutorials
```

- V-cycle structure, but use nonlinear relaxation and skip the matrices
- `./ex19 -da_refine 4 -snes_monitor -snes_type nrichardson -npc_fas_levels_snes_type ngs -npc_fas_levels_snes_gs_sweeps 3 -npc_snes_type fas -npc_snes_max_it 1 -npc_snes_fas_smoothup 6 -npc_snes_fas_smoothdown 6 -lidvelocity 100 -grashof 4e4`

```
lid velocity = 100, prandtl # = 1, grashof # = 40000
0 SNES Function norm 1.065744184802e+03
1 SNES Function norm 5.213040454436e+02
2 SNES Function norm 6.416412722900e+01
3 SNES Function norm 1.052500804577e+01
4 SNES Function norm 2.520004680363e+00
5 SNES Function norm 1.183548447702e+00
6 SNES Function norm 2.074605179017e-01
7 SNES Function norm 6.782387771395e-02
8 SNES Function norm 1.421602038667e-02
9 SNES Function norm 9.849816743803e-03
10 SNES Function norm 4.168854365044e-03
11 SNES Function norm 4.392925390996e-04
12 SNES Function norm 1.433224993633e-04
13 SNES Function norm 1.074357347213e-04
14 SNES Function norm 6.107933844115e-05
15 SNES Function norm 1.509756087413e-05
16 SNES Function norm 3.478180386598e-06

Number of SNES iterations = 16
```

- `./ex19 -da_refine 4 -snes_monitor -snes_type ngmres -npc_fas_levels_snes_type ngs`

Nonlinear multigrid (full approximation scheme)

```
$ cd $PETSC_DIR/src/snes/examples/tutorials
```

- V-cycle structure, but use nonlinear relaxation and skip the matrices
- `./ex19 -da_refine 4 -snes_monitor -snes_type nrichardson -npc_fas_levels_snes_type ngs -npc_fas_levels_snes_gs_sweeps 3 -npc_snes_type fas -npc_snes_max_it 1 -npc_snes_fas_smoothup 6 -npc_snes_fas_smoothdown 6 -lidvelocity 100 -grashof 4e4`
- `./ex19 -da_refine 4 -snes_monitor -snes_type ngmres -npc_fas_levels_snes_type ngs -npc_fas_levels_snes_gs_sweeps 3 -npc_snes_type fas -npc_snes_max_it 1 -npc_snes_fas_smoothup 6 -npc_snes_fas_smoothdown 6 -lidvelocity 100 -grashof 4e4`

```
lid velocity = 100, prandtl # = 1, grashof # = 40000
0 SNES Function norm 1.065744184802e+03
1 SNES Function norm 9.413549877567e+01
2 SNES Function norm 2.117533223215e+01
3 SNES Function norm 5.858983768704e+00
4 SNES Function norm 7.303010571089e-01
5 SNES Function norm 1.585498982242e-01
6 SNES Function norm 2.963278257962e-02
7 SNES Function norm 1.152790487670e-02
8 SNES Function norm 2.092161787185e-03
9 SNES Function norm 3.129419807458e-04
10 SNES Function norm 3.503421154426e-05
11 SNES Function norm 2.898344063176e-06

Number of SNES iterations = 11
```

Outline

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`

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

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

Coupled approach to multiphysics

- Smooth all components together
  - Block SOR is the most popular
  - Block ILU sometimes more robust (e.g. transport/anisotropy)
  - Vanka field-split smoothers or for saddle-point problems
  - Distributive relaxation
- Scaling between fields is critical
- Indefiniteness
  - Make smoothers and interpolants respect inf-sup condition
  - Difficult to handle anisotropy
  - Exotic interpolants for Helmholtz
- Transport
  - Define smoother in terms of first-order upwind discretization (*h*-ellipticity)
  - Evaluate residuals using high-order discretization
  - Use Schur field-split: “parabolize” at top level or for smoother on levels
- Multigrid inside field-split or field-split inside multigrid
- Open research area, hard to write modular software

Anisotropy, Heterogeneity

- Anisotropy
  - Semi-coarsening
  - Line smoothers
  - Order unknowns so that incomplete factorization “includes” a line smoother
- Heterogeneity
  - Make coarse grids align
  - Strong smoothers
  - Energy-minimizing interpolants
  - Galerkin coarse levels
  - Homogenization
- Mostly possible with generic software components

Algebraic Multigrid Tuning

- Smoothed Aggregation (GAMG, ML)
  - Graph/strength of connection – MatSetBlockSize()
  - Threshold (`-pc_gamg_threshold`)
  - Aggregate (MIS, HEM)
  - Tentative prolongation – MatSetNearNullSpace()
  - Eigenvalue estimate
  - Chebyshev smoothing bounds
- BoomerAMG (Hypre)
  - Strong threshold (`-pc_hypre_boomeramg_strong_threshold`)
  - Aggressive coarsening options