The Portable Extensible Toolkit for Scientific computing

This talk: http://59A2.org/files/20110817-ACTS.pdf

Jed Brown

Mathematics and Computer Science Division, Argonne National Laboratory

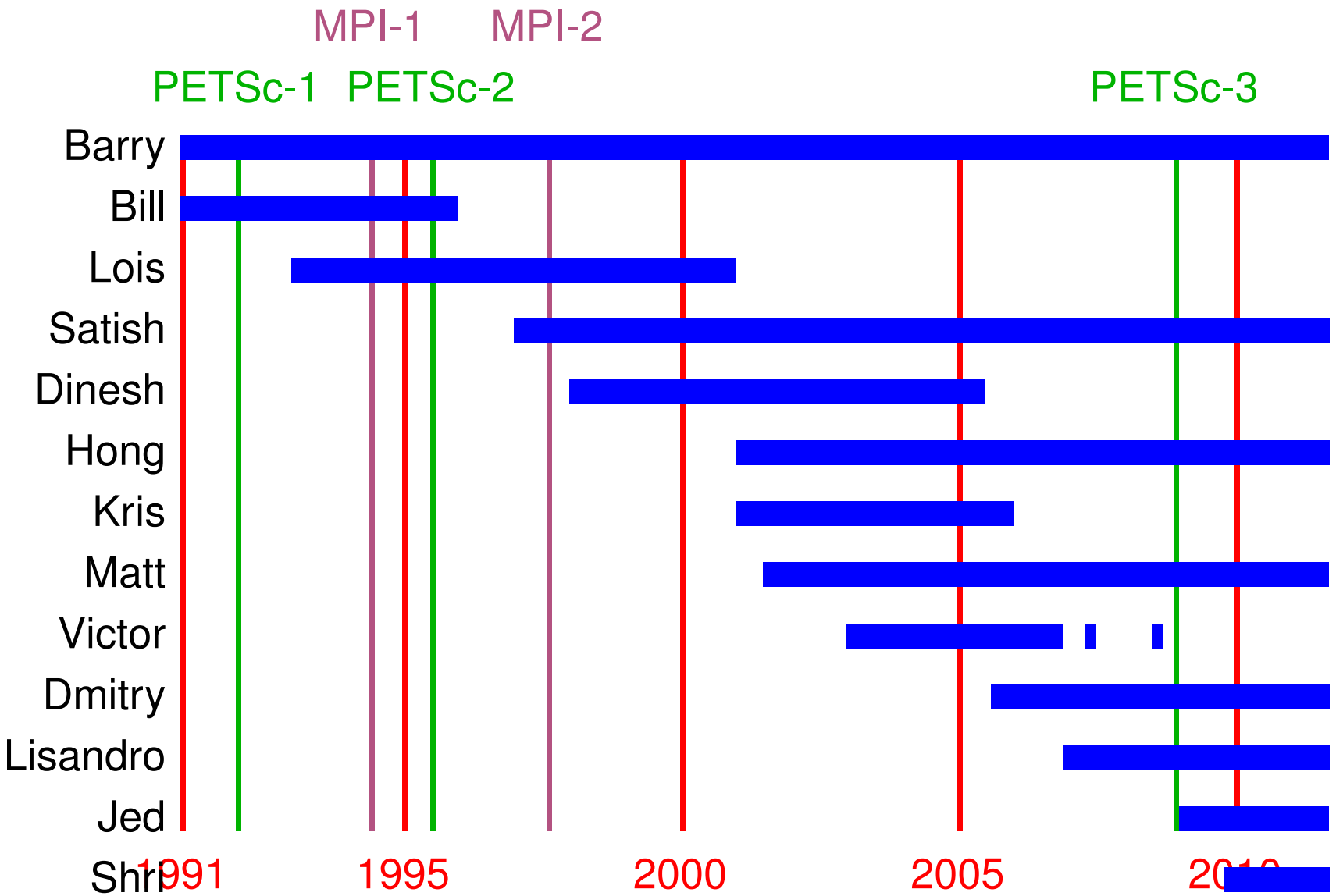
ACTS 2011-08-17

Outline

Requests

- Tell me if you do not understand
- Tell me if an example does not work
- Suggest better wording, figures, organization
- Follow up:
  - Configuration issues, private: `petsc-maint@mcs.anl.gov`
  - Public questions: `petsc-users@mcs.anl.gov`
  - Me: `jed@59A2.org`

Outline


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

Portable Extensible Toolkit for Scientific computing

- Architecture
  - tightly coupled (e.g. XT5, BG/P, Earth Simulator)
  - 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 (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
  - 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 (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`

The Role of PETSc

Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort.

PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver, nor a silver bullet.

— Barry Smith

Outline

Downloading

- `http://mcs.anl.gov/petsc`, download tarball
- We will use Mecurial in this tutorial:
  - `http://mercurial.selenic.com`
  - Debian/Ubuntu: `$ aptitude install mercurial`
  - Fedora: `$ yum install mercurial`
- Get the PETSc release
  - `$ hg clone` \
    `http://petsc.cs.iit.edu/petsc/releases/petsc-3.1`
  - `$ cd petsc-3.1`
  - `$ hg clone` \
    `http://petsc.cs.iit.edu/petsc/releases/BuildSystem-3.1` \
    `config/BuildSystem`
  - Get the latest bug fixes with `$ hg pull --update`

Configuration

Basic configuration

- `$ export PETSC_DIR=$PWD PETSC_ARCH=mpich-gcc-dbg`
- ```
  $ ./configure --with-shared                                    \
        --with-blas-lapack-dir=/usr                              \
        --download-{mpich,ml,hypre}
  ```
- `$ make all test`

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

  ```
  $ mpich-gcc-dbg/conf/reconfigure-mpich-gcc-dbg.py              \
        --new-options
  ```
  ```

Automatic Downloads

- Most packages can be automatically
  - Downloaded
  - Configured and Built (in `$PETSC_DIR/externalpackages`)
  - Installed with PETSc
- Currently works for
  - petsc4py
  - PETSc documentation utilities (Sowing, lgrind, c2html)
  - BLAS, LAPACK, BLACS, ScaLAPACK, PLAPACK
  - MPICH, MPE, Open MPI
  - ParMetis, Chaco, Jostle, Party, Scotch, Zoltan
  - MUMPS, Spooles, SuperLU, SuperLU_Dist, UMFPack, pARMS
  - PaStiX, BLOPEX, FFTW, SPRNG
  - Prometheus, HYPRE, ML, SPAI
  - Sundials
  - Triangle, TetGen, FIAT, FFC, Generator
  - HDF5, Boost

Can also use `--with-xxx=/path/to/your/install`

An optimized build

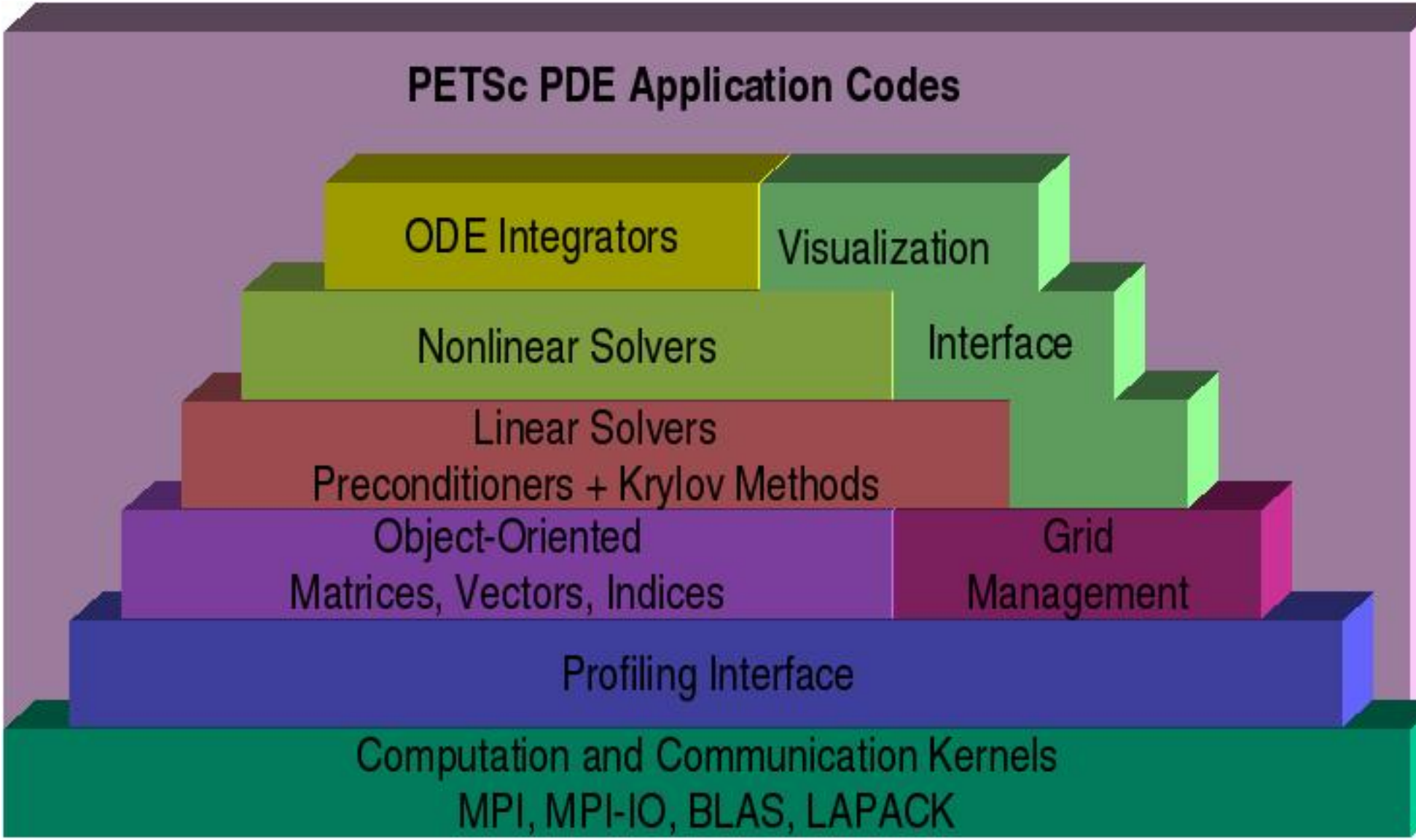
- `$ mpich-gcc-dbg/conf/reconfigure-mpich-gcc-dbg.py PETSC_ARCH=mpich-gcc-opt --with-debugging=0 && make PETSC_ARCH=mpich-gcc-opt`
- What does `--with-debugging=1` (default) do?
  - Keeps debugging symbols (of course)
  - Maintains a stack so that errors produce a full stack trace (even SEGV)
  - Does 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`

Compiling an example

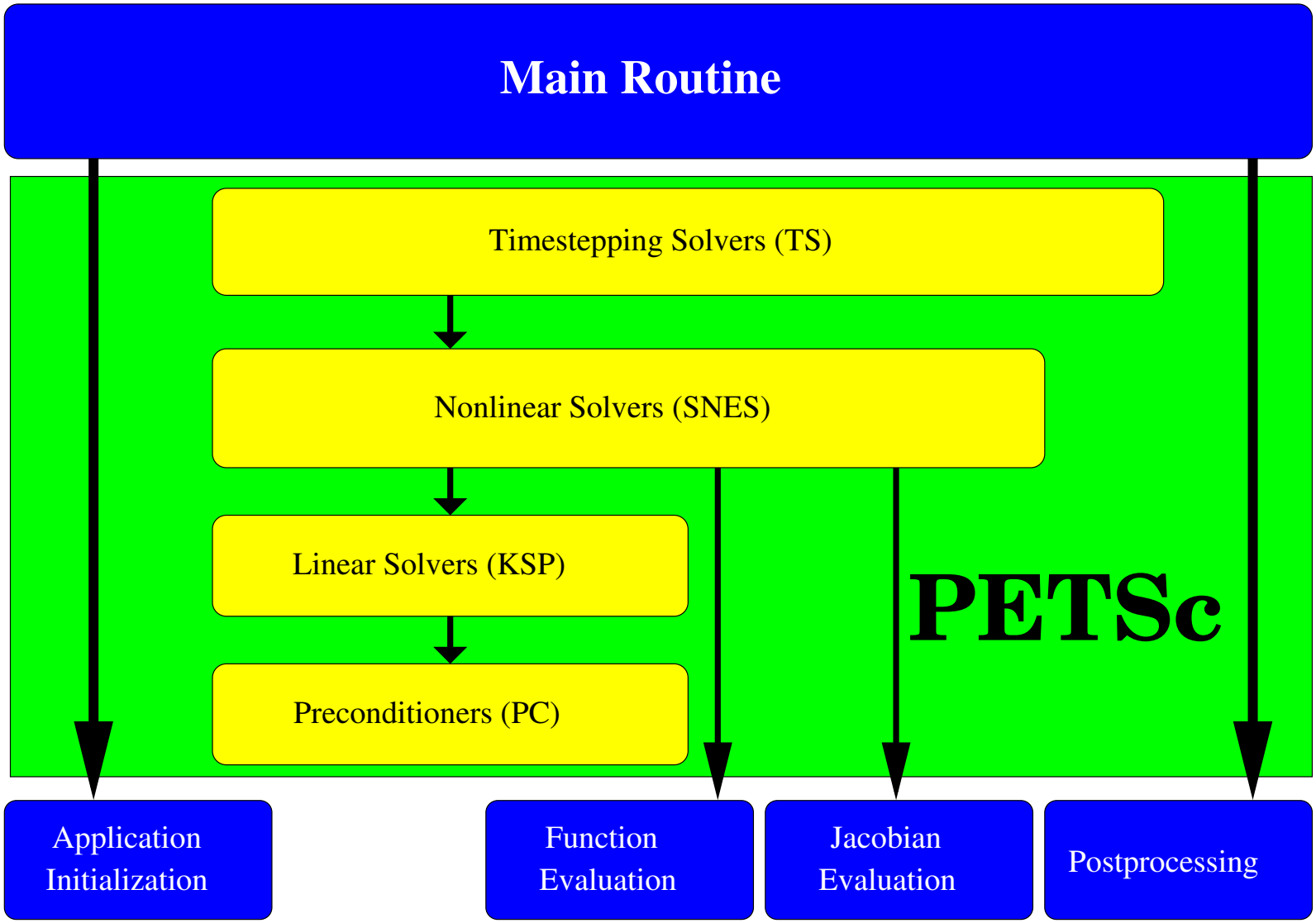
- `$ hg clone http://petsc.cs.iit.edu/petsc/tutorials/CSCS10`
- `$ cd CSCS10`
- `$ hg update 1`
- `$ make`
- `$ ./pbratu -help | less`

Outline

PETSc Structure



Flow Control for a PETSc Application



MPI

- Message Passing Interface
- Defines an interface, many implementations
- Can write and run multiprocess jobs without a multicore processor
- Highly optimized
  - Often bypasses kernel, IP stack
  - Network hardware can send/receive directly from user memory (no-copy)
  - Many nonblocking primitives, one-sided operations
  - Can use shared memory within a node
    - Sometimes faster to have network hardware do the copy
- Designed for library interoperability (e.g. attribute caching)
- Not very fault tolerant
  - Usually can't recover if one process disappears, deadlock possible
  - CAP Theorem
- `$ mpiexec -n 4 ./app -program -options`
- Highly configurable runtime options (Open MPI, MPICH2)
- Rarely have to call MPI directly when using PETSc

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 upcoming MPI-3
  - Collective: `VecNorm()`, `MatAssemblyBegin()`, `KSPCreate()`
    - Global communication, synchronous
    - Non-blocking collectives in upcoming MPI-3
- Deadlock if some process doesn't participate (e.g. wrong order)

Advice from Bill Gropp

You want to think about how you decompose your data structures, how you think about them globally. [...] If you were building a house, you'd start with a set of blueprints that give you a picture of what the whole house looks like. You wouldn't start with a bunch of tiles and say. "Well I'll put this tile down on the ground, and then I'll find a tile to go next to it." But all too many people try to build their parallel programs by creating the smallest possible tiles and then trying to have the structure of their code emerge from the chaos of all these little pieces. You have to have an organizing principle if you're going to survive making your code parallel.

(`http://www.rce-cast.com/Podcast/rce-28-mpich2.html`)

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.

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
```

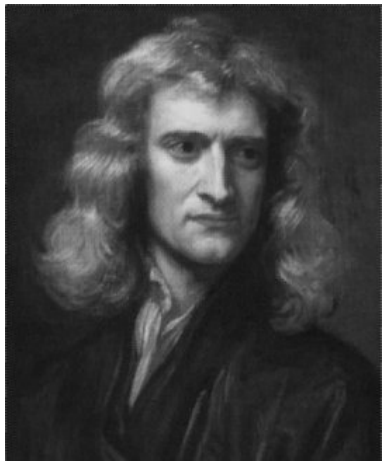
- `$ ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7 -snes_monitor -{ksp,snes}_converged_reason -snes_view`
- `$ ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7 -snes_monitor -{ksp,snes}_converged_reason -snes_view -mat_view_draw -draw_pause 0.5`
- `$ ./ex5 -da_grid_x 10 -da_grid_y 10 -par 6.7 -snes_monitor -{ksp,snes}_converged_reason -snes_view -mat_view_draw -draw_pause 0.5 -pc_type lu -pc_factor_mat_ordering_type natural`
- Use `-help` to find other ordering types

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 ml`

Outline

Newton iteration: foundation 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)

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

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

The $\mathfrak{p}$-Bratu equation

- 2-dimensional model problem

$$-\nabla\cdot\left(|\nabla u|^{\mathfrak{p}-2}\nabla u\right)-\lambda e^{u}-f=0, \qquad 1\leq\mathfrak{p}\leq\infty, \quad \lambda<\lambda_{\text{crit}}(\mathfrak{p})$$

  Singular or degenerate when $\nabla u = 0$, turning point at λ_{crit}.

- Regularized variant

$$-\nabla\cdot(\eta\nabla u)-\lambda e^{u}-f=0$$

$$\eta(\gamma)=(\varepsilon^{2}+\gamma)^{\frac{\mathfrak{p}-2}{2}} \qquad \gamma(u)=\frac{1}{2}|\nabla u|^{2}$$

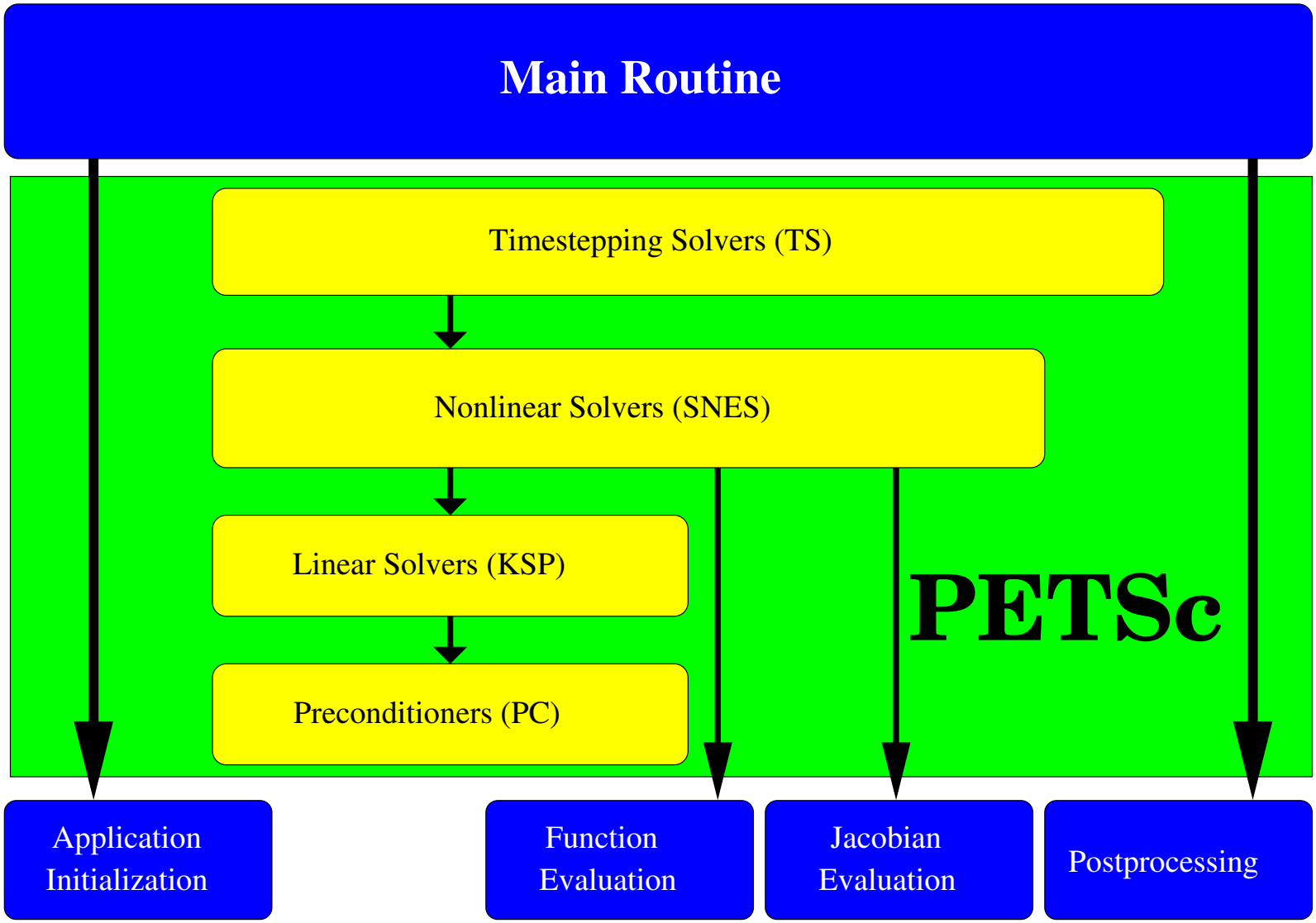
- Jacobian

$$J(u)w\sim-\nabla\cdot\left[(\eta 1+\eta'\nabla u\otimes\nabla u)\nabla w\right]-\lambda e^{u}w$$

$$\eta'=\frac{\mathfrak{p}-2}{2}\eta/(\varepsilon^{2}+\gamma)$$

  Physical interpretation: conductivity tensor flattened in direction ∇u

Flow Control for a PETSc Application



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 which calculates the Jacobian has signature

```
PetscErrorCode (*func)(SNES snes,Vec x,Mat *J,Mat *M,
                       MatStructure *flag,void *ctx)
```

- `x`: The current solution
- `J`: The Jacobian
- `M`: The Jacobian preconditioning matrix (possibly J itself)
- `ctx`: The user context passed to `SNESSetFunction()`
  - Use this to pass application information, e.g. physical constants
- Possible `MatStructure` values are:
  - SAME_NONZERO_PATTERN
  - DIFFERENT_NONZERO_PATTERN

Alternatively, you can use

- a builtin sparse finite difference approximation (“coloring”)
- automatic differentiation (ADIC/ADIFOR)

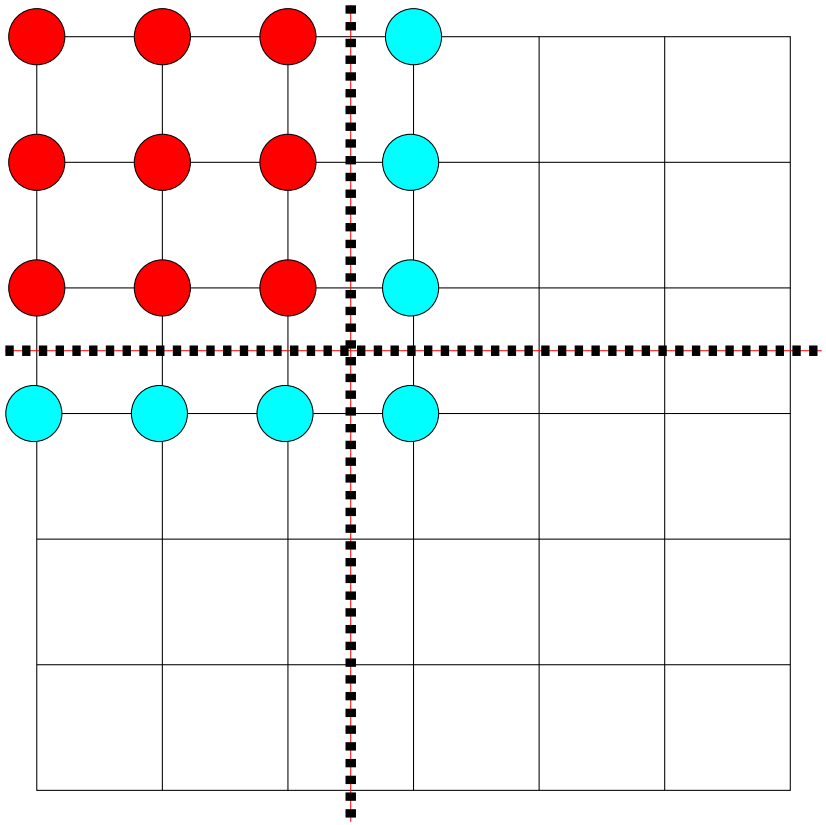
Distributed Array

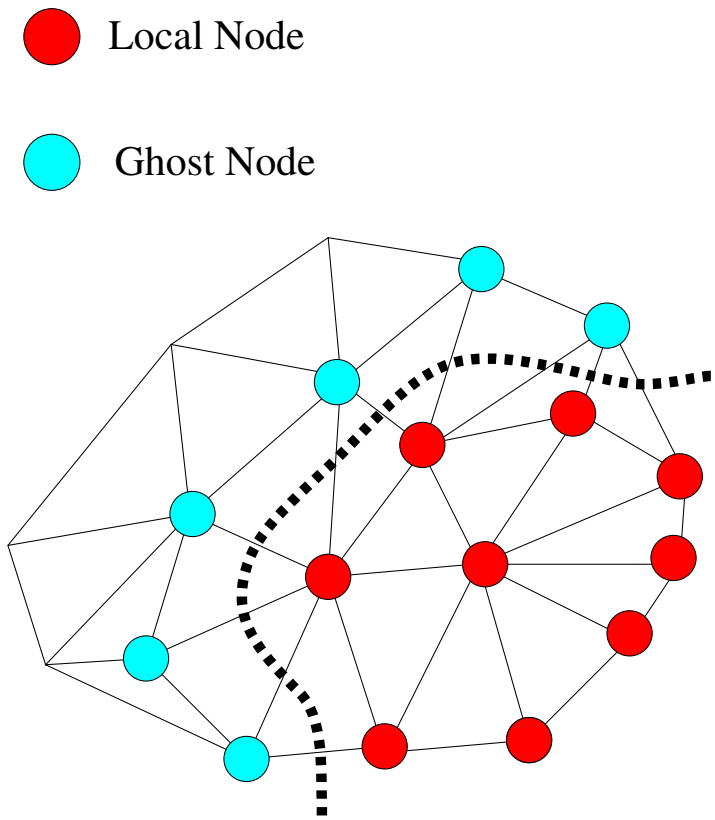
- Interface for topologically structured grids
- Defines (topological part of) a finite-dimensional function space
  - Get an element from this space: `DACreateGlobalVector()`
- Provides parallel layout
- Refinement and coarsening
  - `DARefineHierarchy()`
- Ghost value coherence
  - `DAGlobalToLocalBegin()`
- Matrix preallocation:
  - `DAGetMatrix()`

Ghost Values

To evaluate a local function $f(x)$, each process requires

- its local portion of the vector x
- its ghost values, bordering portions of x owned by neighboring processes





DA Global Numberings

Proc 2			Proc 3	
25	26	27	28	29
20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4
Proc 0			Proc 1	

Natural numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

PETSc numbering

DA Global vs. Local Numbering

- **Global**: Each vertex has a unique id belongs on a unique process
- **Local**: Numbering includes vertices from neighboring processes
  - These are called ghost vertices

Proc 2			Proc 3	
X	X	X	X	X
X	X	X	X	X
12	13	14	15	X
8	9	10	11	X
4	5	6	7	X
0	1	2	3	X
Proc 0			Proc 1	

Local numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

Global numbering

DA Vectors

- The DA object contains only layout (topology) information
  - All field data is contained in PETSc `Vec`s
- Global vectors are parallel
  - Each process stores a unique local portion
  - `DACreateGlobalVector(DA da, Vec *gvec)`
- Local vectors are sequential (and usually temporary)
  - Each process stores its local portion plus ghost values
  - `DACreateLocalVector(DA da, Vec *lvec)`
  - includes ghost values!
- Coordinate vectors store the mesh geometry
  - `DAGetCoordinates(DA,Vec *coords)`
  - Can be manipulated with their own DA
    `DAGetCoordinateDA(DA,DA *cda)`

Updating Ghosts

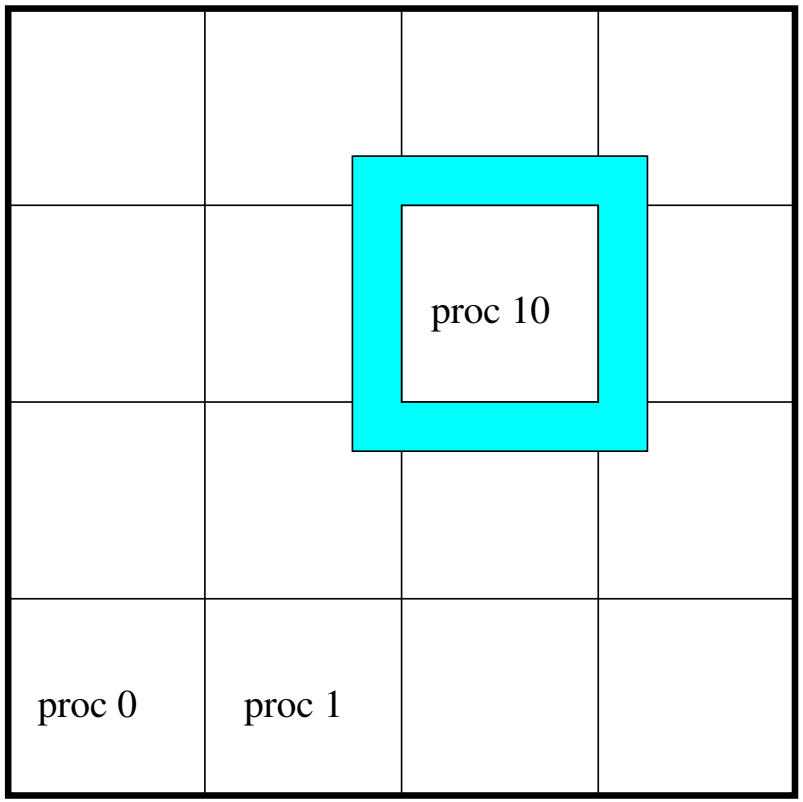
Two-step process enables overlapping computation and communication

- `DAGlobalToLocalBegin(da, gvec, mode, lvec)`
  - `gvec` provides the data
  - `mode` is either `INSERT_VALUES` or `ADD_VALUES`
  - `lvec` holds the local and ghost values
- `DAGlobalToLocalEnd(da, gvec, mode, lvec)`
  - Finishes the communication

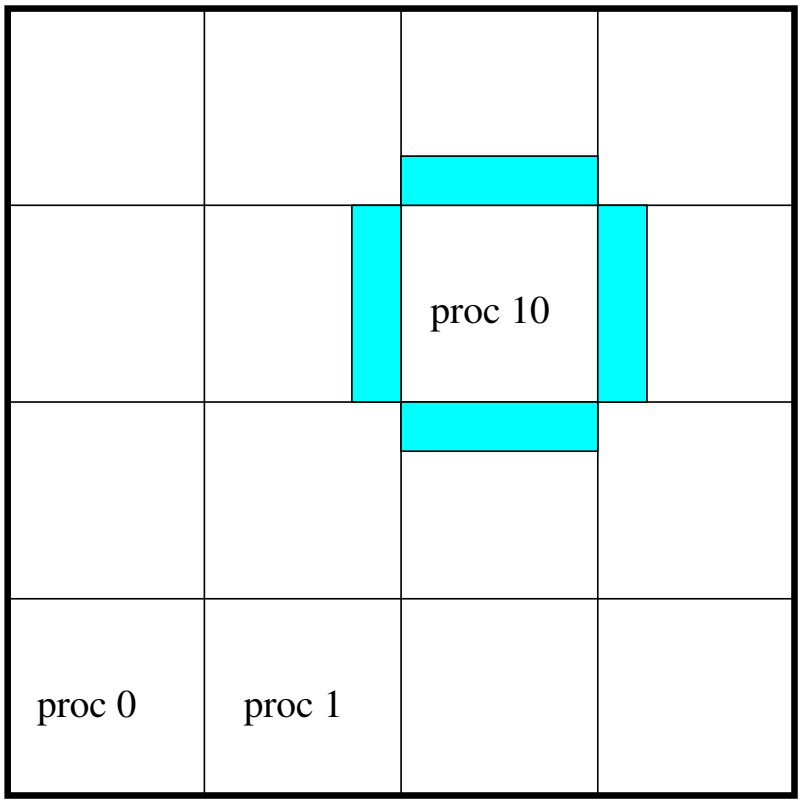
The process can be reversed with `DALocalToGlobal()`.

DA Stencils

Both the box stencil and star stencil are available.



Box Stencil



Star Stencil

Creating a DA

```
DACreate2d(comm,wrap, type, M, N, m, n,
           dof, s, lm[], ln[], DA *da)
```

`wrap`: Specifies periodicity

- `DA_NONPERIODIC`, `DA_XPERIODIC`, `DA_YPERIODIC`, or `DA_XYPERIODIC`

`type`: Specifies stencil

- `DA_STENCIL_BOX` or `DA_STENCIL_STAR`

`M,N`: Number of grid points in x/y-direction

`m,n`: Number of processes in x/y-direction

`dof`: Degrees of freedom per node

`s`: The stencil width

`lm,ln`: Alternative array of local sizes

- Use `PETSC_NULL` for the default

Working with the local form

Wouldn't it be nice if we could just write our code for the natural numbering?

- Yes, that's what `DAVecGetArray()` is for.
- Also, the DA offers local callback functions
  - `FormFunctionLocal()`, set by `DASetLocalFunction()`
  - `FormJacobianLocal()`, set by `DASetLocalJacobian()`
- When PETSc needs to evaluate the nonlinear residual $F(x)$,
  - Each process evaluates the local residual
  - PETSc assembles the global residual automatically
    - Uses `DALocalToGlobal()` method

DA Local Function

The user provided function which calculates the nonlinear residual in 2D has signature

```
PetscErrorCode (*lfunc)(DALocalInfo *info,
            Field **x, Field **r, void *ctx)
```

`info`: All layout and numbering information

`x`: The current solution

- Notice that it is a multidimensional array

`r`: The residual

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

The local DA function is activated by calling

```
SNESSetFunction(snes, r, SNESDAFormFunction, ctx)
```

Bratu Residual Evaluation

$$-\Delta u - \lambda e^u = 0$$

```
BratuResidualLocal(DALocalInfo *info,Field **x,Field **f,
                   UserCtx *user)
{
  /* Not Shown: 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++) {
      u         = x[j][i];
      u_xx      = (2.0*u - x[j][i-1] - x[j][i+1])*hydhx;
      u_yy      = (2.0*u - x[j-1][i] - x[j+1][i])*hxdhy;
      f[j][i] = u_xx + u_yy - hx*hy*lambda*exp(u);
    }
  }
}
```

$PETSC_DIR/src/snes/examples/tutorials/ex5.c

Start with 2-Laplacian plus Bratu nonlinearity

- Matrix-free Jacobians, no preconditioning `-snes_mf`
- `$ hg update -r3`
- `$ ./pbratu -da_grid_x 10 -da_grid_y 10 -lambda 6.7 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_grid_x 20 -da_grid_y 20 -lambda 6.7 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_grid_x 40 -da_grid_y 40 -lambda 6.7 -snes_mf -snes_monitor -ksp_converged_reason`
- Watch linear and nonlinear convergence

Add $\mathfrak{p}$ nonlinearity

- Matrix-free Jacobians, no preconditioning `-snes_mf`
- `$ hg update -r4`
- `$ ./pbratu -da_grid_x 10 -da_grid_y 10 -lambda 1 -p 1.3 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_grid_x 20 -da_grid_y 20 -lambda 1 -p 1.3 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_grid_x 40 -da_grid_y 40 -lambda 1 -p 1.3 -snes_mf -snes_monitor -ksp_converged_reason`
- Watch linear and nonlinear convergence

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

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

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

- 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

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

Successive over-relaxation (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

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
- Very poor scaling in parallel, don't bother beyond 8 PEs.
- No guarantees

1-level Domain decomposition

Domain size L, subdomain size H, element size h

Overlapping/Schwarz

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

Neumann-Neumann

- Solve Neumann problems on non-overlapping subdomains
- $its \in \mathcal{O}\left(\frac{L}{H}(1+\log\frac{H}{h})\right)$
- Tricky null space issues (floating subdomains)
- Need subdomain matrices, net 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

Finite Difference Jacobians

PETSc can compute and explicitly store a Jacobian via 1st-order FD

- Dense
  - Activated by `-snes_fd`
  - Computed by `SNESDefaultComputeJacobian()`
- Sparse via colorings
  - Coloring is created by `MatFDColoringCreate()`
  - Computed by `SNESDefaultComputeJacobianColor()`

Can also use Matrix-free Newton-Krylov via 1st-order FD

- Activated by `-snes_mf` without preconditioning
- Activated by `-snes_mf_operator` with user-defined preconditioning
  - Uses preconditioning matrix from `SNESSetJacobian()`

Add finite difference Jacobian by coloring

- `$ hg update -r5`
- `$ ./pbratu -da_grid_x 10 -da_grid_y 10 -lambda 1 -p 1.3 -snes_fd -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_grid_x 10 -da_grid_y 10 -lambda 1 -p 1.3 -fd_jacobian -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_grid_x 10 -da_grid_y 10 -lambda 1 -p 1.3 -fd_jacobian -snes_monitor -ksp_converged_reason`
- Try some different preconditioners (`jacobi`, `sor`, `asm`, `hypre`, `ml`)
- Try changing the physical parameters
- May need `-mat_fd_type ds`

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, Spooles, SuperLU, UMFPack, Hypre

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, Spooles, SuperLU, UMFPack, Hypre

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
- `MatMPIBAIJSetPreallocation(Mat,...)`
  - important for assembly performance, more tomorrow
- `MatSetBlockSize(Mat, int bs)`
  - for vector problems
- `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
- 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.

One 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;
if (rank == 0) {
  for(row = 0;  row < N; 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);
```

A Better 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)

$\mathfrak{p}$-Bratu assembly

- Use `DAGetMatrix()` (can skip matrix preallocation details)
- Start by just assembling Bratu nonlinearity
- `$ hg update -r6`
- Watch `-snes_converged_reason`, what happens for $p \neq 2$?
- Solve exactly with the preconditioner `-pc_type lu`
- Try `-snes_mf_operator`

$\mathfrak{p}$-Bratu assembly

- We need to assemble the $\mathfrak{p}$ part

$$J(u)w \quad \sim \quad -\nabla\cdot\left[(\eta 1 + \eta'\nabla u \otimes \nabla u)\nabla w\right]$$

- Second part is scary, but what about just using $-\nabla\cdot(\eta\nabla w)$?
- `$ hg update -r7`
- Solve exactly with the preconditioner `-pc_type lu`
- Try `-snes_mf_operator`
- Refine the grid, change $\mathfrak{p}$
- Try algebraic multigrid if available: `-pc_type [ml,hypre]`

Does the preconditioner need Newton linearization?

- The anisotropic part looks messy.
  Is it worth writing the code to assemble that part?
- Easy profiling: `-log_summary`
- Observation: the Picard linearization uses a “star” (5-point) stencil while Newton linearization needs a “box” (9-point) stencil.
- Add support for reduced preallocation with a command-line option
- `$ hg update -r8`
- Compare performance (time, memory, iteration count) of
  - 5-point Picard-linearization assembled by hand
  - 5-point Newton-linearized Jacobian computed by coloring
  - 9-point Newton-linearized Jacobian computed by coloring

Maybe it's not worth it, but let's assemble it anyway

- `$ hg update -r9`
- Crash!
- You were using the the debug PETSC_ARCH, right?
- Launch the debugger
  - `-start_in_debugger [gdb,dbx,noxterm]`
  - `-on_error_attach_debugger [gdb,dbx,noxterm]`
- Attach the debugger only to some parallel processes
  - `-debugger_nodes 0,1`
- Set the display (often necessary on a cluster)
  - `-display :0`

Debugging Tips

- Put a breakpoint in `PetscError()` to catch errors as they occur
- PETSc tracks memory overwrites at both ends of arrays
  - The `CHKMEMQ` macro causes a check of all allocated memory
  - Track memory overwrites by bracketing them with `CHKMEMQ`
- PETSc checks for leaked memory
  - Use `PetscMalloc()` and `PetscFree()` for all allocation
  - Print unfreed memory on `PetscFinalize()` with `-malloc_dump`
- Simply the best tool today is Valgrind
  - It checks memory access, cache performance, memory usage, etc.
  - http://www.valgrind.org
  - Pass `-malloc 0` to PETSc when running under Valgrind
  - Might need `--trace-children=yes` when running under MPI
  - `--track-origins=yes` handy for uninitialized memory

Memory error is gone now

- `$ hg update -r10`
- Run with -snes_mf_operator -pc_type lu
- Do you see quadratic convergence?
- Hmm, there must be a bug in that mess, where is it?

Memory error is gone now

- `$ hg update -r10`
- Run with -snes_mf_operator -pc_type lu
- Do you see quadratic convergence?
- Hmm, there must be a bug in that mess, where is it?

SNES Test

- PETSc can compute a finite difference Jacobian and compare it to yours
- `-snes_type test`
  - Is the difference significant?
- `-snes_type test -snes_test_display`
  - Are the entries in the star stencil correct?
- Find which line has the typo
- `$ hg update -r11`
- Check with `-snes_type test`
- and `-snes_mf_operator -pc_type lu`

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 are rarely helpful
- 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

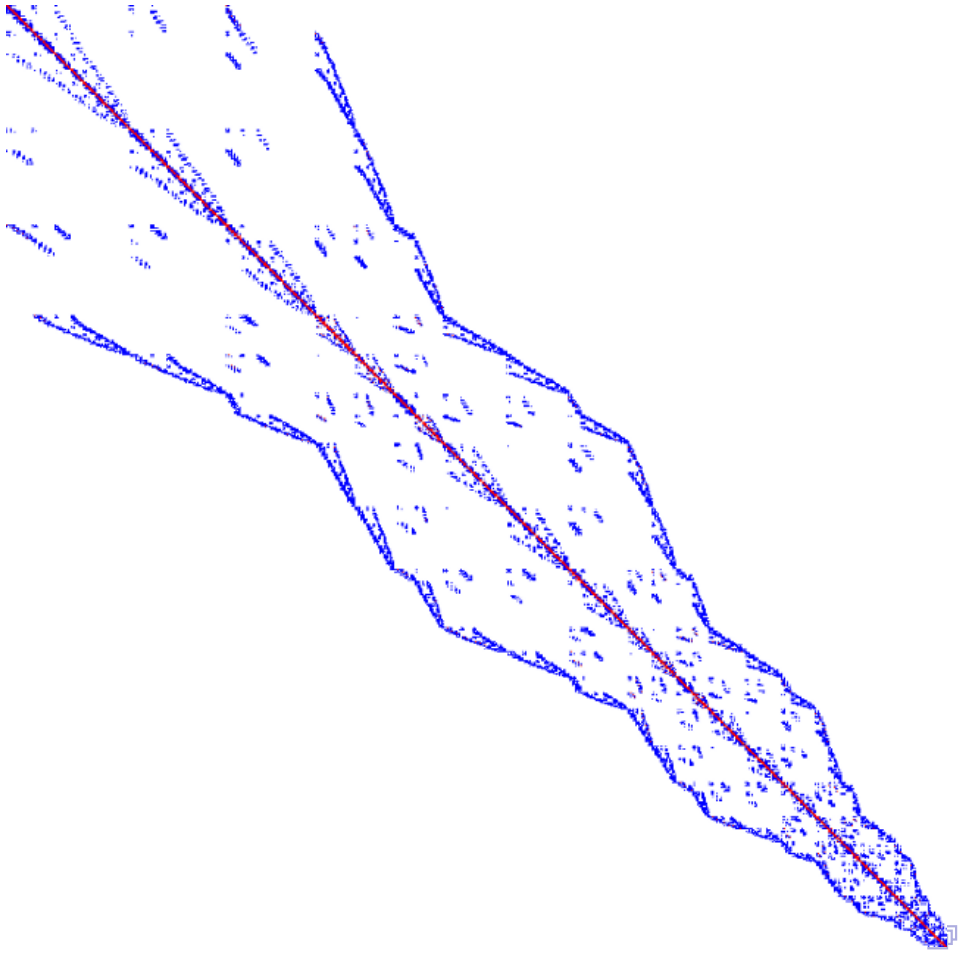
- PETSc sparse matrices are dynamic data structures
  - can add additional nonzeros freely
- Dynamically adding many nonzeros
  - requires additional memory allocations
  - requires copies
  - can kill performance
- Memory preallocation provides
  - the freedom of dynamic data structures
  - good performance
- 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

Sequential Sparse Matrices

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

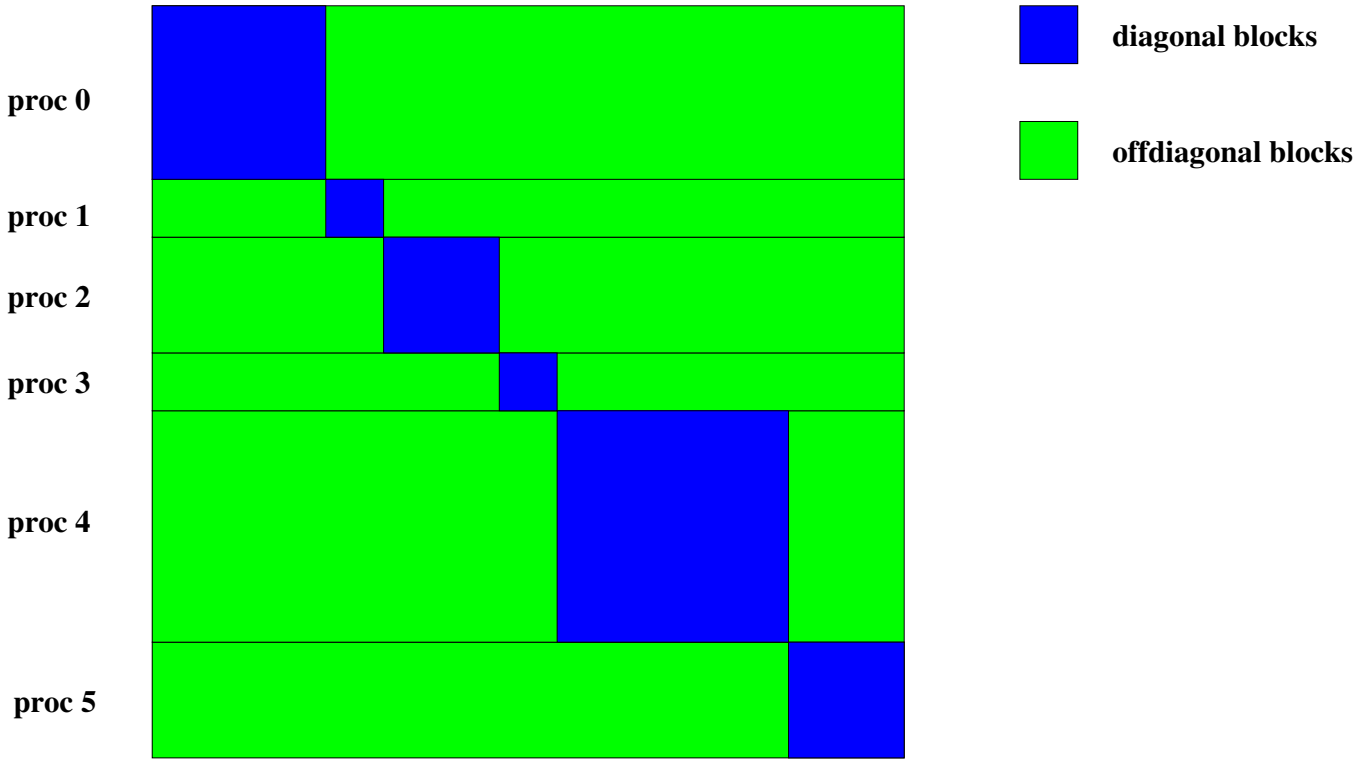
nz: expected number of nonzeros in any row

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

```
MatMPIAIJPreallocation(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 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
[0]MatAssemblyEnd_SeqAIJ:Most nonzeros in any row is 5
[0]Mat_AIJ_CheckInode: Found 56 nodes out of 56 rows. Not using Inode routine
[0]Mat_AIJ_CheckInode: Found 56 nodes out of 56 rows. Not using Inode routine
Norm of error 0.000156044 iterations 6
[0]PetscFinalize:PETSc successfully ended!
```

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 M, MatStructure flag)`
  - `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`

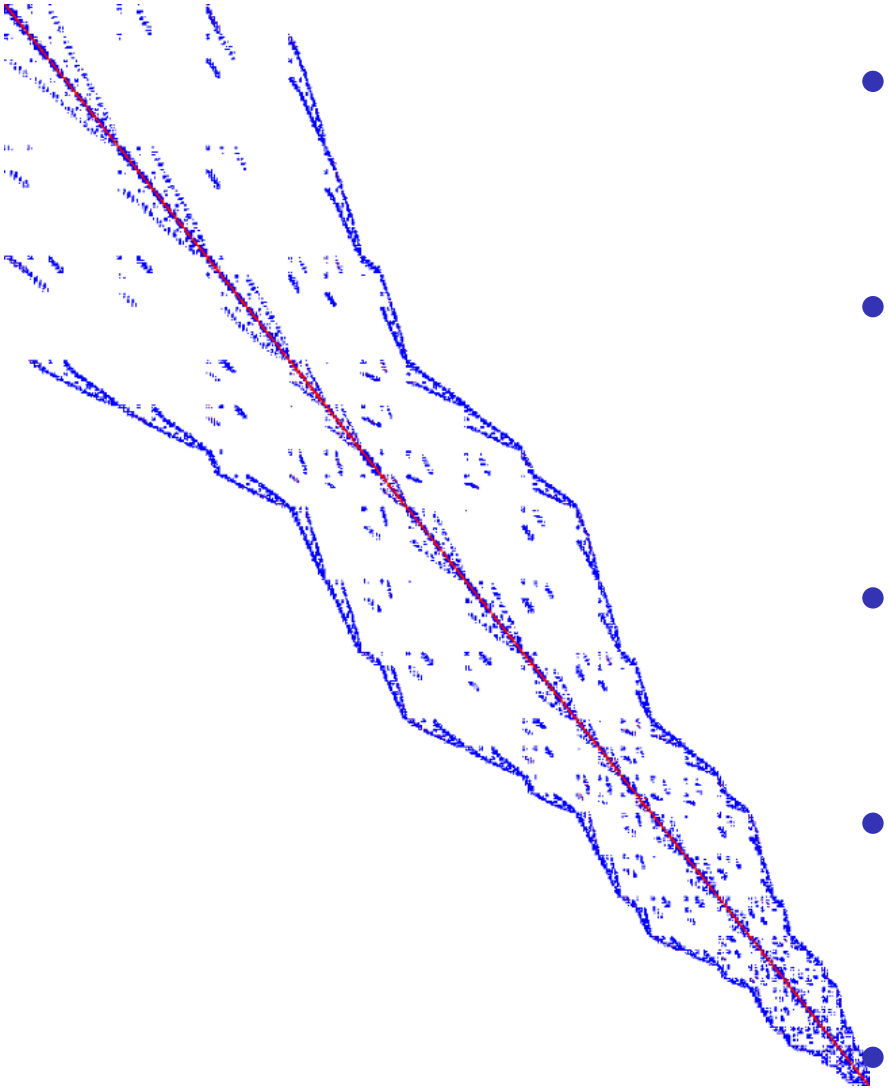
Nonlinear Solvers

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

Bottlenecks of (Jacobian-free) Newton-Krylov

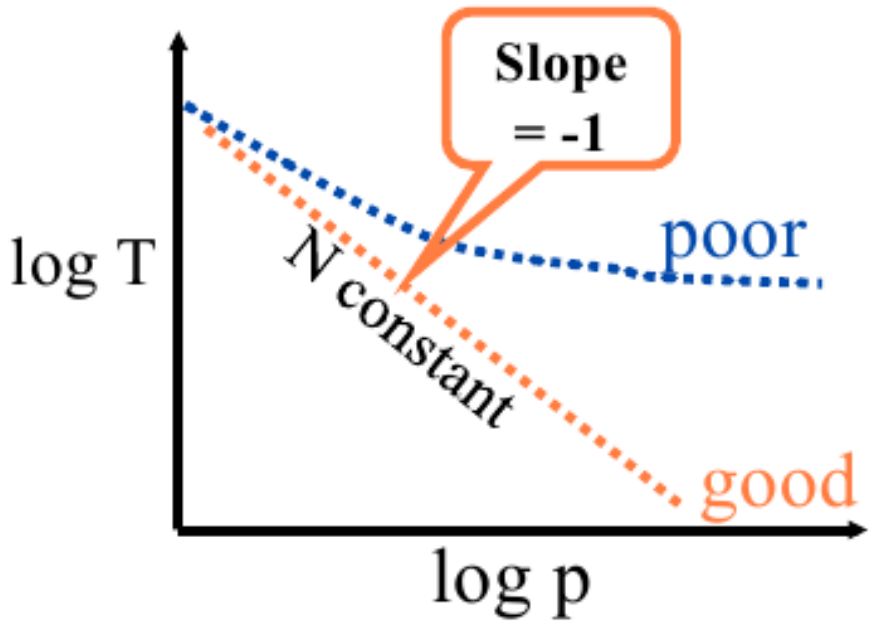


- Matrix assembly
  - integration/fluxes: FPU
  - insertion: memory/branching
- Preconditioner setup
  - coarse level operators
  - overlapping subdomains
  - (incomplete) factorization
- Preconditioner application
  - triangular solves/relaxation: memory
  - coarse levels: network latency
- Matrix multiplication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

Scalability definitions

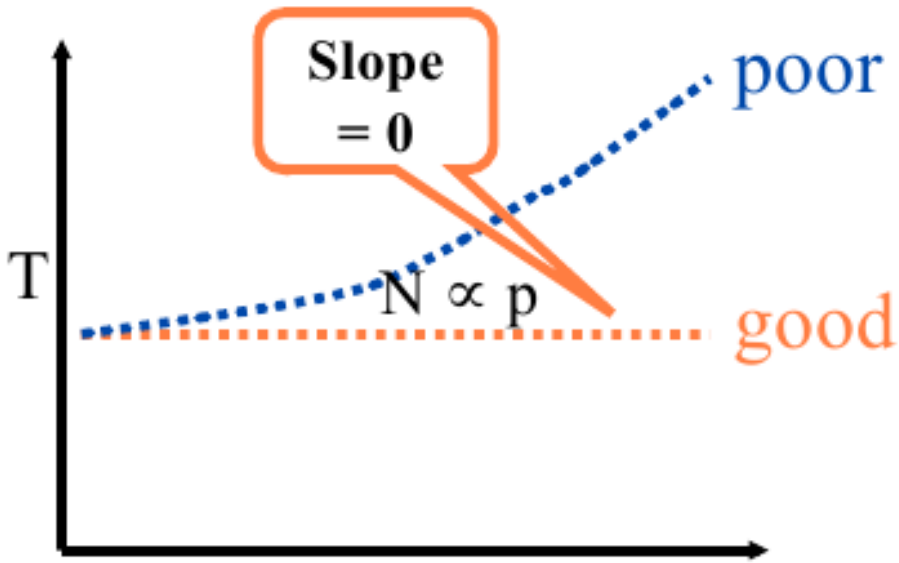
Strong scalability

- Fixed problem size
- execution time T inversely proportional to number of processors p



Weak scalability

- Fixed problem size per processor
- execution time constant as problem size increases

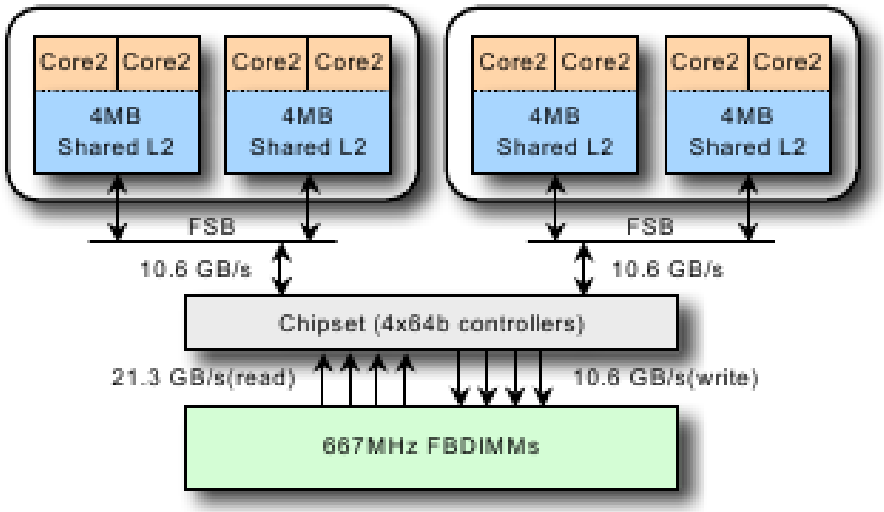


Scalability Warning

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

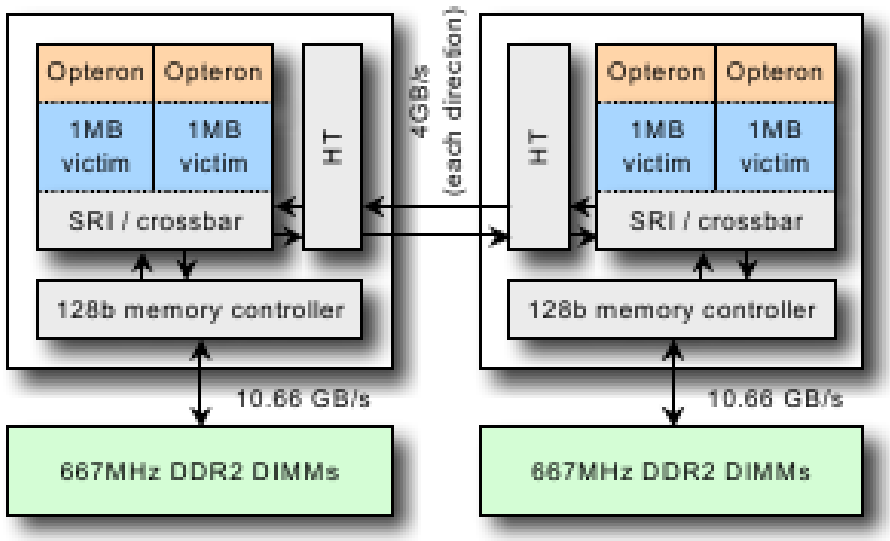
- We really want efficient software
- Need a performance model
  - memory bandwidth and latency
  - algorithmically critical operations (e.g. dot products, scatters)
  - floating point unit
- Scalability shows marginal benefit of adding more cores, nothing more
- Constants hidden in the choice of algorithm
- Constants hidden in implementation

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

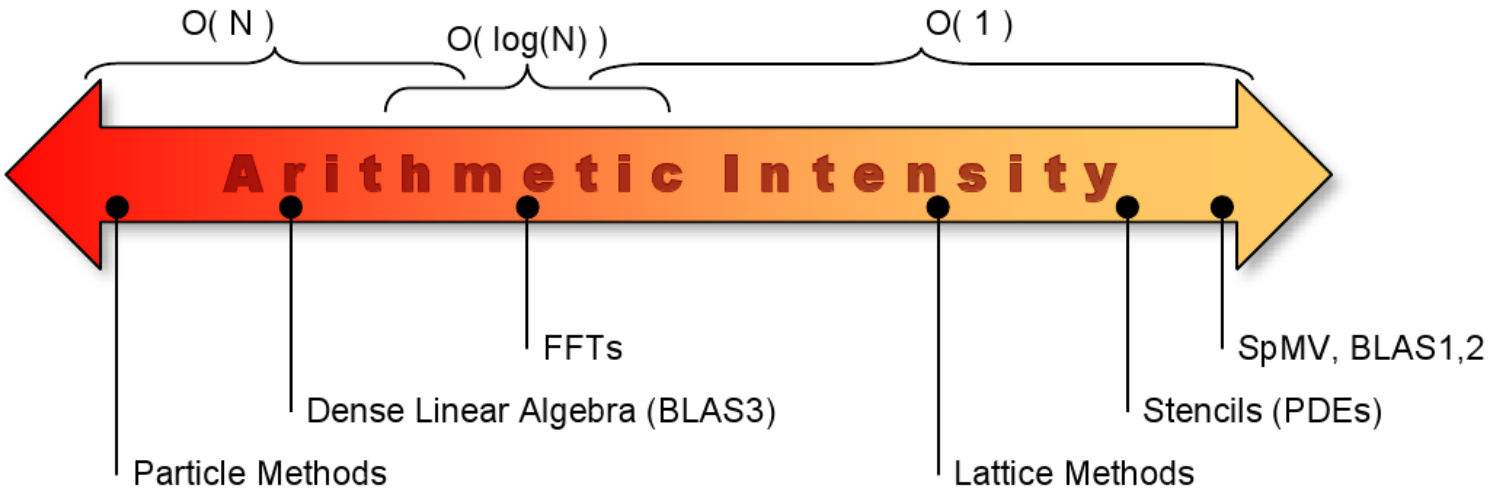
Floating point unit

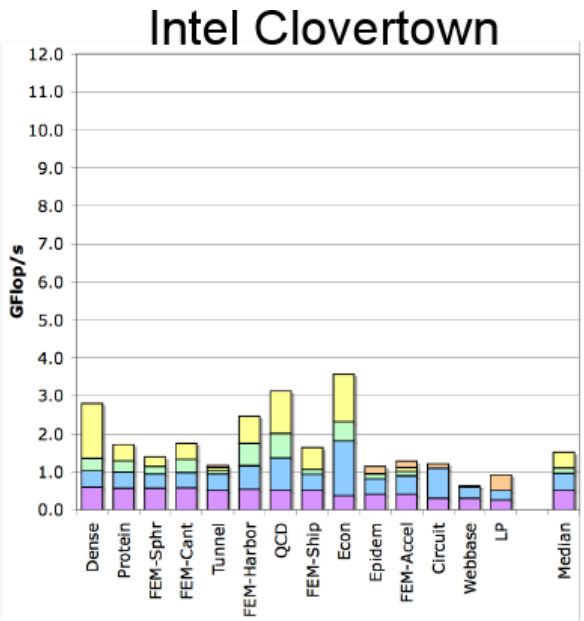
Recent Intel: each core can issue

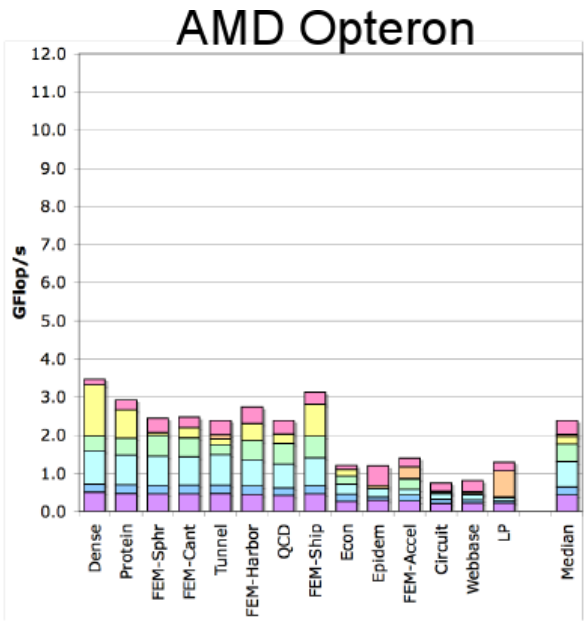
- 1 packed add (latency 3)
- 1 packed mult (latency 5)
- One can include an aligned read
- Out of Order execution
- Peak: 10 Gflop/s (`double`)

Memory

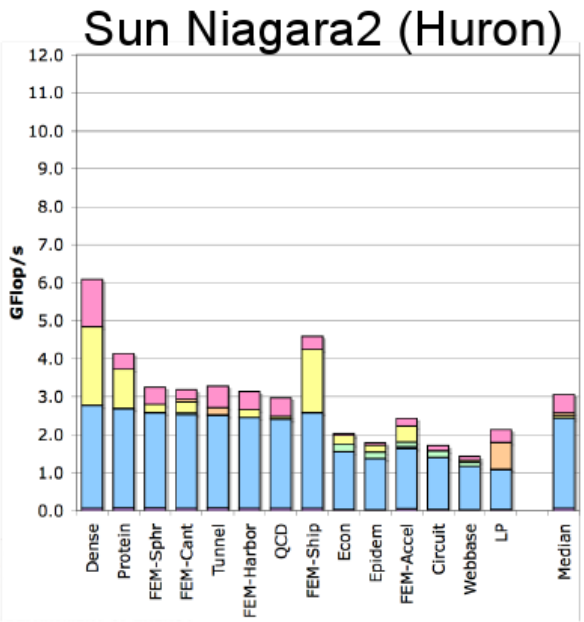
- $\sim$ 250 cycle latency
- 5.3 GB/s bandwidth
- 1 `double` load / 3.7 cycles
- Pay by the cache line (32/64 B)
- L2 cache: $\sim$ 10 cycle latency

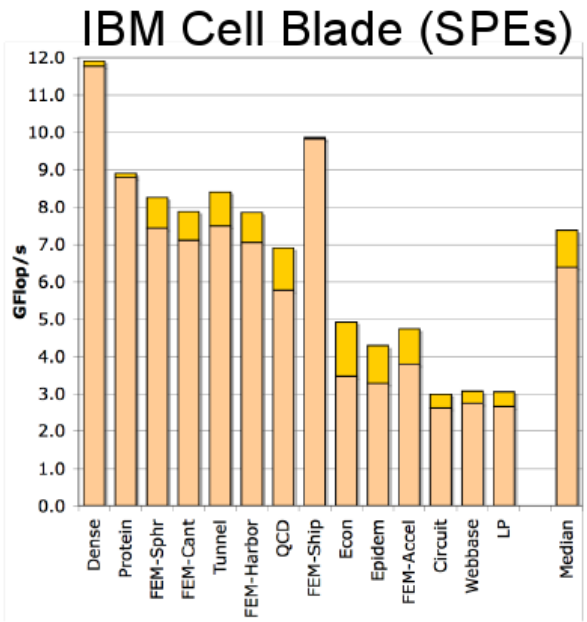


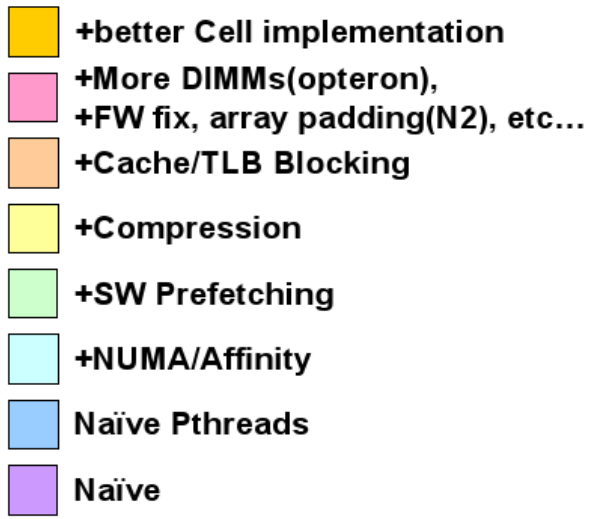




- Model faster cores by commenting out the inner kernel calls, but still performing all DMAs
- Enabled 1x1 BCOO
- ~16% improvement







(Oliker et al. Multi-core Optimization of Sparse Matrix Vector Multiplication, 2008)

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	-	-

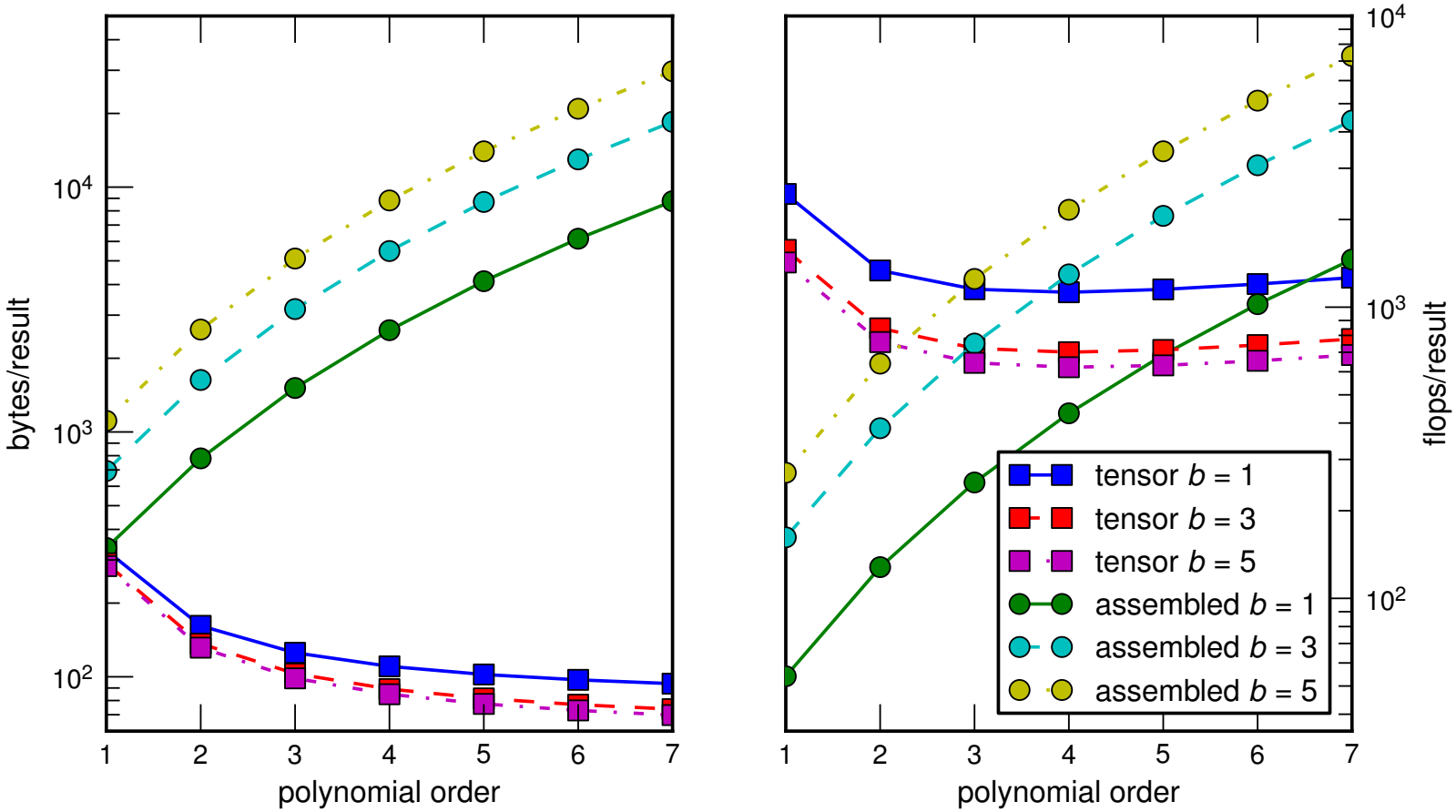
- Sparse matrix-vector product: 6 bytes per flop

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()`

Performance of assembled versus unassembled



- Arithmetic intensity for Q_p elements
  - $\leq \frac{1}{4}$ (assembled), ≈ 10 (unassembled), ≈ 4 (hardware)
- store Jacobian information at Quass quadrature points, can use AD

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 $> 70\%$ of peak FPU (compare to $< 5\%$ for assembled operators on multicore)
  - Can store Jacobian information at quadrature points (usually pays off for Q_2 and higher in 3D)
  - Spectral, WENO, DG, FD
  - Often still need an assembled operator for preconditioning
- Boundary element methods
  - Dense kernels
  - Fast Multipole Method (FMM)
- Preconditioning requires more effort
  - Useful have code to assemble matrices: try out new methods quickly

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 $> 70\%$ of peak FPU (compare to $< 5\%$ for assembled operators on multicore)
  - Can store Jacobian information at quadrature points (usually pays off for Q_2 and higher in 3D)
  - Spectral, WENO, DG, FD
  - Often still need an assembled operator for preconditioning
- Boundary element methods
  - Dense kernels
  - Fast Multipole Method (FMM)
- Preconditioning requires more effort
  - Useful have code to assemble matrices: try out new methods quickly

Profiling

- Use `-log_summary` 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_summary`

-

	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_summary` when asking performance questions on mailing list

Communication Costs

- Reductions: usually part of Krylov method, latency limited
  - `VecDot`
  - `VecMDot`
  - `VecNorm`
  - `MatAssemblyBegin`
  - Change algorithm (e.g. IBCGS)
- 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
  - Better partition

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 $u = (u, v)$

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

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

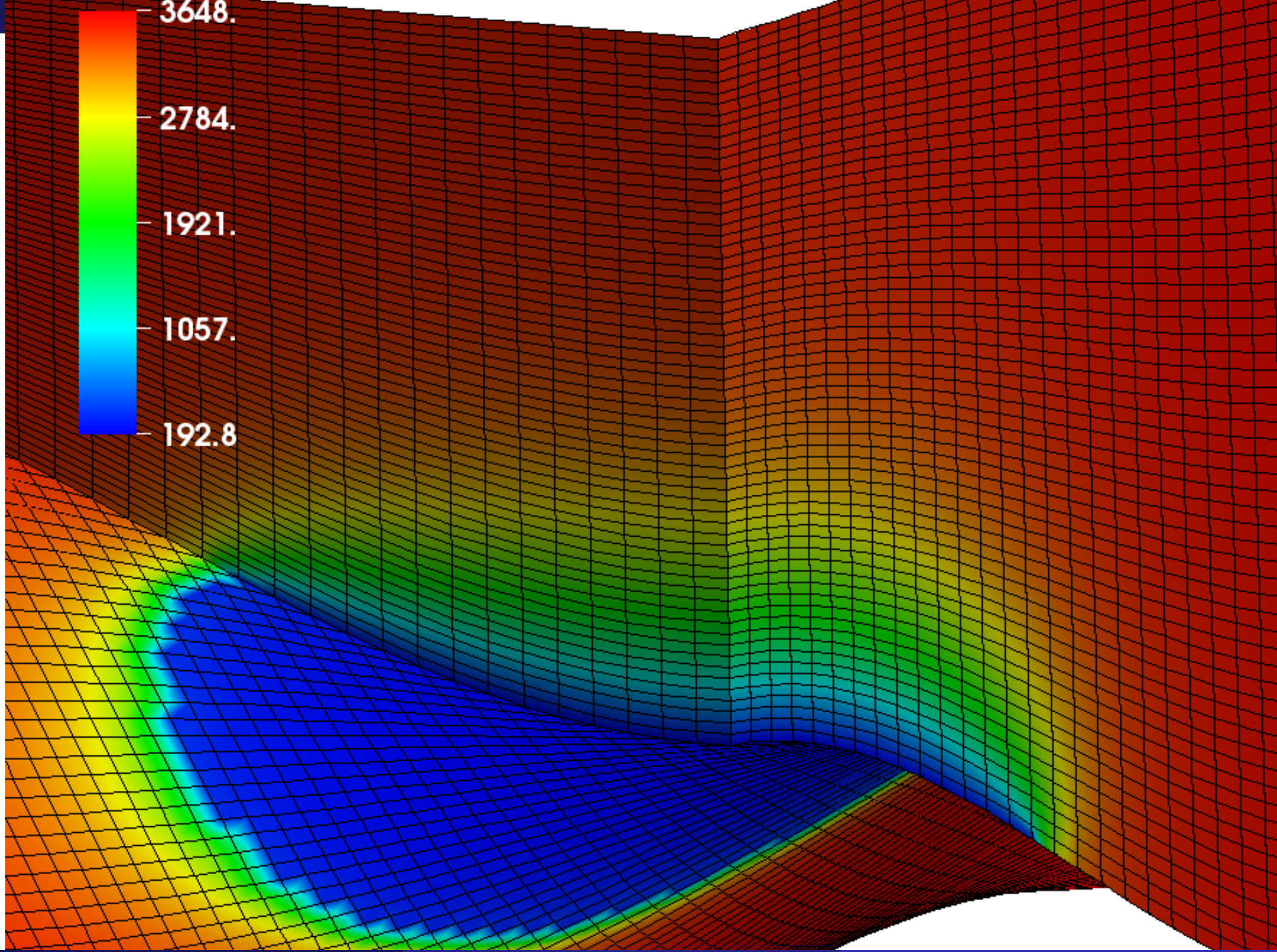
$$\gamma = u_x^2 + v_y^2 + u_x v_y + \frac{1}{4}(u_y + v_x)^2 + \frac{1}{4}u_z^2 + \frac{1}{4}v_z^2$$

  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 with Newton-Krylov-Multigrid solver in PETSc:


3648.
2784.
1921.
1057.
192.8

Some Multigrid Options

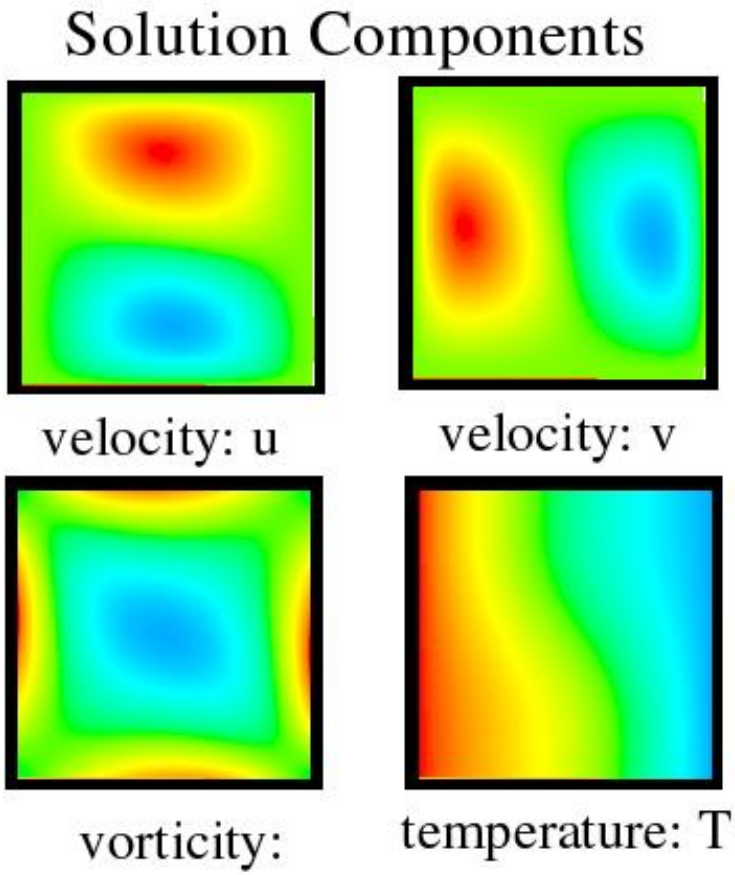
- `-dmmg_grid_sequencce`: [FALSE]
  Solve nonlinear problems on coarse grids to get initial guess
- `-pc_mg_galerkin`: [FALSE]
  Use Galerkin process to compute coarser operators
- `-pc_mg_type`: [FULL]
  (choose one of) MULTIPLICATIVE ADDITIVE FULL KASKADE
- `-mg_coarse_{ksp,pc}_*`
  control the coarse-level 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 -dmmg_grid_sequence 1 -dmmg_view -log_summary -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 `-dmmg_grid_sequence`?
- What about solving with block Jacobi, ASM, or algebraic multigrid?

SNES Example

Driven Cavity



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

SNES Example

Driven Cavity Application Context

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

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

SNES Example

Driven Cavity Residual Evaluation

```
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 */
  VecGetArray(F, &f);
  /* Code to compute local function components */
  VecRestoreArray(F, &f);
  return 0;
}
```

SNES Example

Better Driven Cavity Residual Evaluation

```
PetscErrorCode DrivenCavityFuncLocal(DALocalInfo *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;
      upw        = 0.5*(x[j+1][i].omega-x[j-1][i].omega)*hx
      f[j][i].u  = uxx + uyy - upw;
      /* V velocity, Omega, Temperature */
}}}
```

$PETSC_DIR/src/snes/examples/tutorials/ex19.c

Running the driven cavity

- `./ex19 -lidvelocity 100 -grashof 1e2 -da_grid_x 16 -da_grid_y 16 -snes_monitor -dmmg_view -nlevels 3`
- `./ex19 -lidvelocity 100 -grashof 1e4 -da_grid_x 16 -da_grid_y 16 -snes_monitor -dmmg_view -nlevels 3`
- `./ex19 -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -snes_monitor -dmmg_view -nlevels 3`
- Uh oh, we have convergence problems
- Run with `-snes_monitor_convergence`
- Does `-dmmg_grid_sequence` help?

Running the driven cavity

- `./ex19 -lidvelocity 100 -grashof 1e2 -da_grid_x 16 -da_grid_y 16 -snes_monitor -dmmg_view -nlevels 3`
- `./ex19 -lidvelocity 100 -grashof 1e4 -da_grid_x 16 -da_grid_y 16 -snes_monitor -dmmg_view -nlevels 3`
- `./ex19 -lidvelocity 100 -grashof 1e5 -da_grid_x 16 -da_grid_y 16 -snes_monitor -dmmg_view -nlevels 3`
- Uh oh, we have convergence problems
- Run with `-snes_monitor_convergence`
- Does `-dmmg_grid_sequence` help?

Why isn't SNES converging?

- The Jacobian is wrong (maybe only in parallel)
  - Check with `-snes_type test` 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 `-info` or `-snes_ls_monitor` (petsc-dev) to see line search
  - Try using trust region instead of line search `-snes_type tr`
  - Try grid sequencing if possible
  - Use a continuation

Globalizing the lid-driven cavity

Pseudotransient continuation continuation (Ψtc)

- Do linearly implicit backward-Euler steps, driven by steady-state residual
- Clever way to adjust step sizes to retain quadratic convergence in terminal phase

- Implemented in `src/snes/examples/tutorials/ex27.c`
- `$ make runex27`
- Make the method linearly implicit: `-snes_max_it 1`
  - Compare required number of linear iterations
- Try increasing `-lidvelocity`, `-grashof`, and problem size
- Coffey, Kelley, and Keyes, Pseudotransient continuation and differential algebraic equations, SIAM J. Sci. Comp, 2003.

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]`

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

  - 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 often more robust (e.g. transport/anisotropy)
  - Vanka field-split smoothers or for saddle-point problems
- 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

“Physics-based” preconditioners (semi-implicit method)

Shallow water with stiff gravity wave

h is hydrostatic pressure, u is velocity, $\sqrt{gh}$ is fast wave speed

$$h_t - (uh)_x = 0$$
$$(uh)_t + (u^2 h + \frac{1}{2} g h^2)_x = 0$$

Semi-implicit method

Suppress spatial discretization, discretize in time, implicitly for the terms contributing to the gravity wave

$$\frac{h^{n+1} - h^n}{\Delta t} + (uh)_x^{n+1} = 0$$
$$\frac{(uh)^{n+1} - (uh)^n}{\Delta t} + (u^2 h)_x^n + g(h^n h^{n+1})_x = 0$$

Rearrange, eliminating $(uh)^{n+1}$

$$\frac{h^{n+1} - h^n}{\Delta t} - \Delta t (g h^n h_x^{n+1})_x = -S_x^n$$

Delta form

- Preconditioner should work like the Newton step: $-F(x) \mapsto \delta x$
- Recast semi-implicit method in delta form

$$\frac{\delta h}{\Delta t} + (\delta uh)_x = -F_0, \quad \frac{\delta uh}{\Delta t} + gh^n(\delta h)_x = -F_1, \quad \widehat{J} = \begin{pmatrix} \frac{1}{\Delta t} & \nabla\cdot \\ gh^n\nabla & \frac{1}{\Delta t} \end{pmatrix}$$

- Eliminate δuh

$$\frac{\delta h}{\Delta t} - \Delta t(gh^n(\delta h)_x)_x = -F_0 + (\Delta t F_1)_x, \quad S \sim \frac{1}{\Delta t} - g\Delta t\nabla\cdot h^n\nabla$$

- Solve for δh, then evaluate

$$\delta uh = -\Delta t\left[gh^n(\delta h)_x - F_1\right]$$

- Fully implicit solver
  - Is nonlinearly consistent (no splitting error), can be high-order in time
  - Uses existing code when a semi-implicit method has been implemented
  - Allows efficient bifurcation analysis, steady-state analysis

Outline

Multiphysics problems

Examples

- Saddle-point problems (e.g. incompressibility, contact)
- Stiff waves (e.g. low-Mach combustion)
- Mixed type (e.g. radiation hydrodynamics, ALE free-surface flows)
- Multi-domain problems (e.g. fluid-structure interaction)
- Full space PDE-constrained optimization

Software/algorithmic considerations

- Separate groups develop different "physics" components
- Do not know a priori which methods will have good algorithmic properties
- Achieving high throughput is more complicated
- Multiple time and/or spatial scales
  - Splitting methods are delicate, often not in asymptotic regime
  - Strongest nonlinearities usually non-stiff: prefer explicit for TVD

The Great Solver Schism: Monolithic or Split?

Monolithic

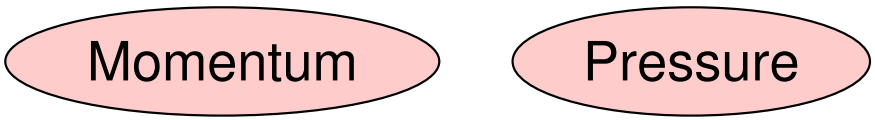
- Direct solvers
- Coupled Schwarz
- Coupled Neumann-Neumann (need unassembled matrices)
- Coupled multigrid
- X Need to understand local spectral and compatibility properties of the coupled system

Split

- Physics-split Schwarz (based on relaxation)
- Physics-split Schur (based on factorization)
  - approximate commutators SIMPLE, PCD, LSC
  - segregated smoothers
  - Augmented Lagrangian
  - "parabolization" for stiff waves
- X Need to understand global coupling strengths

- Preferred data structures depend on which method is used.
- Interplay with geometric multigrid.

Multi-physics coupling in PETSc

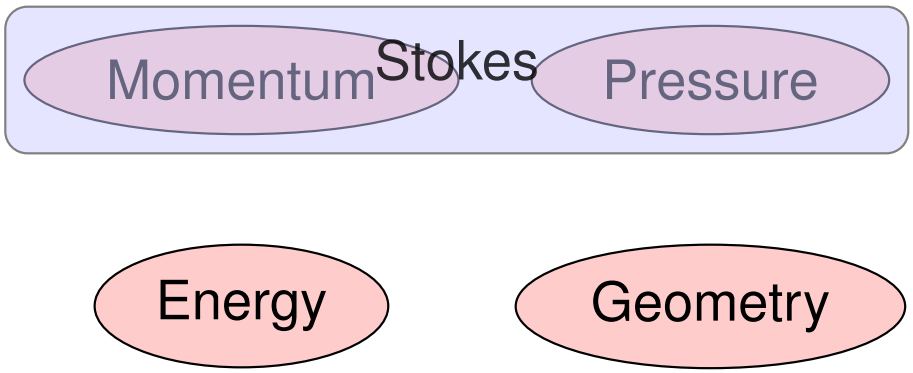


- package each "physics" independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format for each physics (e.g. symmetric block size 3)
- matrix-free anywhere
- multiple levels of nesting

Multi-physics coupling in PETSc

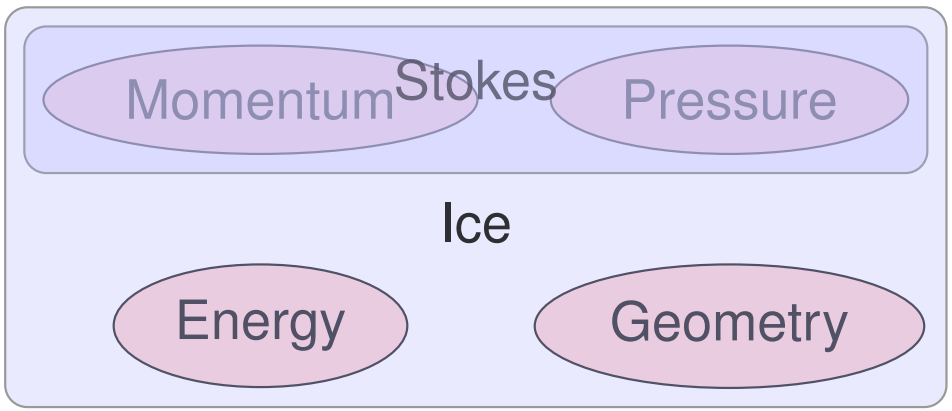
- package each "physics" independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format for each physics (e.g. symmetric block size 3)
- matrix-free anywhere
- multiple levels of nesting

Multi-physics coupling in PETSc



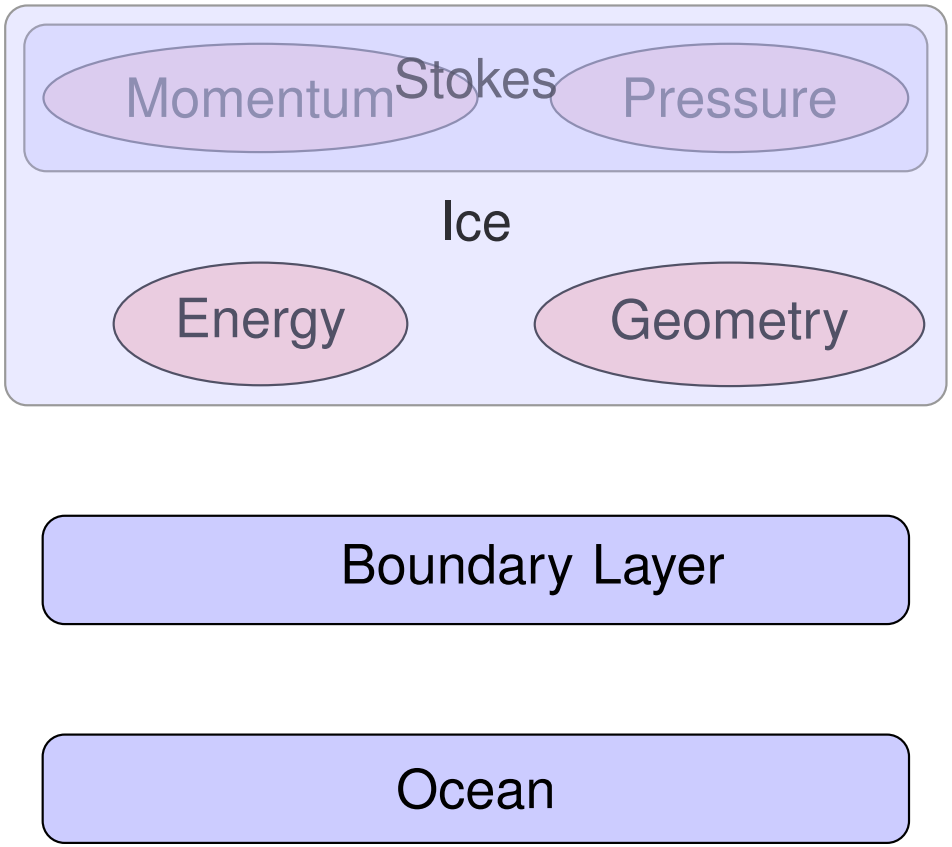
- package each "physics" independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format for each physics (e.g. symmetric block size 3)
- matrix-free anywhere
- multiple levels of nesting

Multi-physics coupling in PETSc

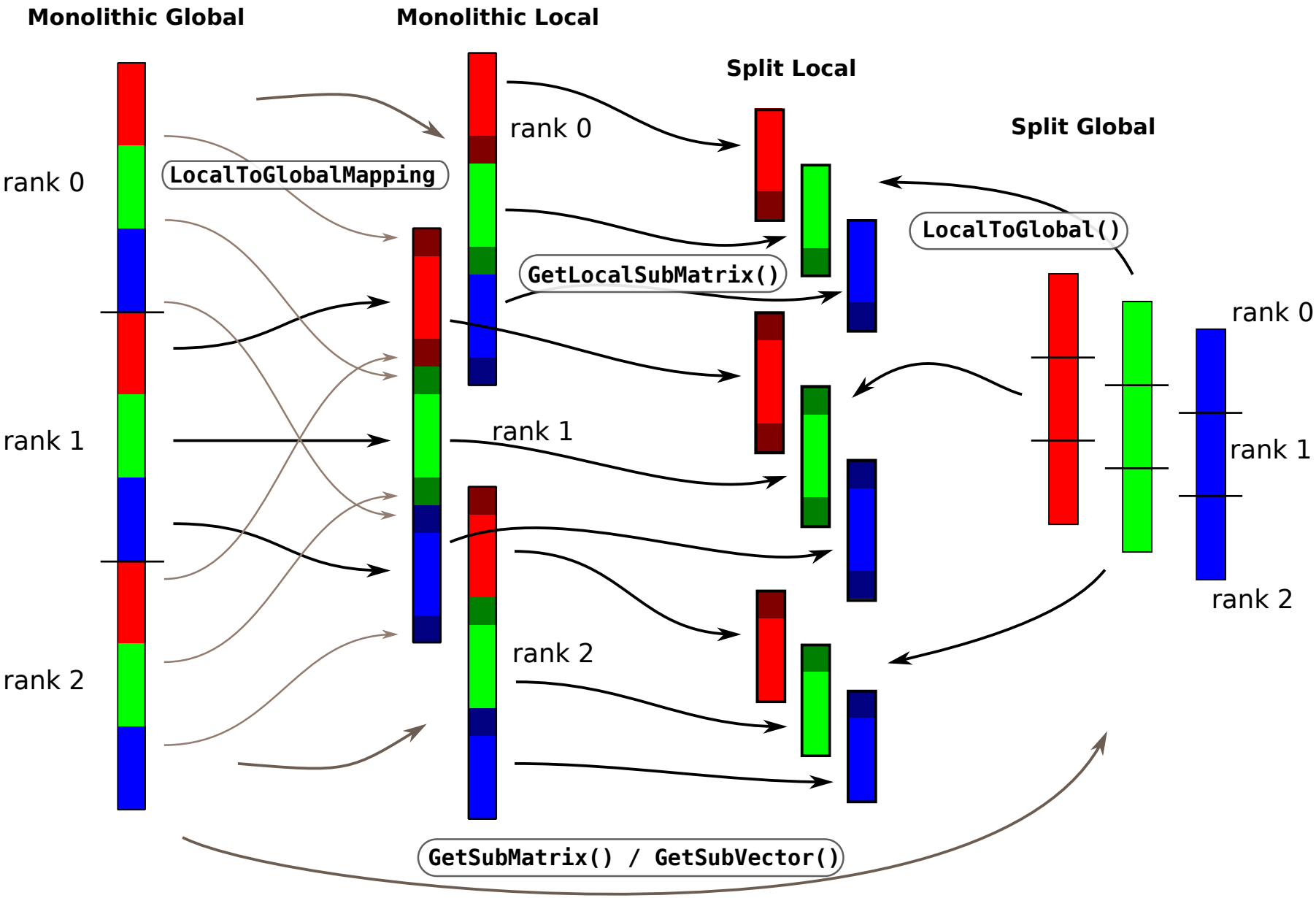


- package each "physics" independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format for each physics (e.g. symmetric block size 3)
- matrix-free anywhere
- multiple levels of nesting

Multi-physics coupling in PETSc



- package each "physics" independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format for each physics (e.g. symmetric block size 3)
- matrix-free anywhere
- multiple levels of nesting


Monolithic Global
Monolithic Local
Split Local
Split Global
rank 0
rank 1
rank 2
LocalToGlobalMapping
GetLocalSubMatrix()
LocalToGlobal()
GetSubMatrix() / GetSubVector()

```
MatGetLocalSubMatrix(Mat A,IS rows,IS cols,Mat *B);
```

- Primarily for assembly
  - `B` is not guaranteed to implement `MatMult`
  - The communicator for `B` is not specified,
    only safe to use non-collective ops (unless you check)
- `IS` represents an index set, includes a block size and communicator
- `MatSetValuesBlockedLocal()` is implemented
- MatNest returns nested submatrix, no-copy
- No-copy for Neumann-Neumann formats
  (unassembled across procs, e.g. BDDC, FETI-DP)
- Most other matrices return a lightweight proxy `Mat`
  - `COMM_SELF`
  - Values not copied, does not implement `MatMult`
  - Translates indices to the language of the parent matrix
  - Multiple levels of nesting are flattened

IMEX time integration in PETSc-3.2

- Additive Runge-Kutta IMEX methods

$$G(t, x, \dot{x}) = F(t, x)$$

$$J_\alpha = \alpha G_{\dot{x}} + G_x$$

  - User provides:
    - `FormRHSFunction(ts,t,x,F,void *ctx);`
    - `FormIFunction(ts,t,x,ẋ,G,void *ctx);`
    - `FormIJacobian(ts,t,x,ẋ,α,J,Jp,mstr,void *ctx);`
  - L-stable DIRK for stiff part G
  - Choice of explicit method, e.g. SSP
  - Orders 2 through 5, embedded error estimates
  - Dense output, hot starts for Newton
  - Can use more accurate methods if G is linear
  - Can use preconditioner from classical "semi-implicit" methods
- Eliminate many interface quirks
- Single step interface so user can have own time loop

Variational Inequalities

- Supports inequality and box constraints on solution variables.
- Solution methods
  - Semismooth Newton
    - reformulate problem as a non-smooth system, Newton on subdifferential
    - Newton step solves diagonally perturbed systems
  - Active set
    - similar linear algebra to solving PDE
    - sometimes slower convergence or "bouncing"
- composes with multigrid and field-split
- demonstrated optimality for phase-field problems with millions of degrees of freedom

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
  - `petsc-{users,dev,maint}@mcs.anl.gov`

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

References

- Knoll and Keyes, Jacobian-free Newton-Krylov methods: a survey of approaches and applications, JCP, 2004.
- Elman et. al., A Taxonomy and Comparison of Parallel Block Multi-Level Preconditioners for the Incompressible Navier-Stokes Equations, JCP, 2008.
- Wan, Chan, and Smith, An Energy-minimizing Interpolation for Robust Multigrid Methods, SIAM J. Sci. Comp, 2000.
- Gropp, Kaushik, Keyes, Smith, Performance Modeling and Tuning of an Unstructured Mesh CFD Application, Supercomputing, 2000.
- Gropp, Exploiting Existing Software in Libraries: Successes, Failures, and Reasons Why, OO methods for interoperable scientific and engineering computing, 1999.
- Upcoming report from ICiS Multiphysics workshop, July 31 – August 5