PETSc Tutorial

Satish Balay
Lois Curfman McInnes

Argonne National Laboratory

with thanks to Jed Brown, Matt Knepley, Karl Rupp, and Barry Smith for slides
additional tutorial material available via `https://www.mcs.anl.gov/petsc`

Argonne Training Program on Extreme-Scale Computing
August 5, 2016

Outline

PETSc Tutorial

- Philosophy
- Vectors and matrices (Vec, Mat)
- Linear solvers (KSP, PC)
- Nonlinear solvers (SNES)
- DAE/ODE integrators: Timestepping (TS)
- Optimization solvers (TAO)
- Topology abstractions: Distributed arrays (DMDA)
- Understanding performance

About PETSc

PETSc Origins

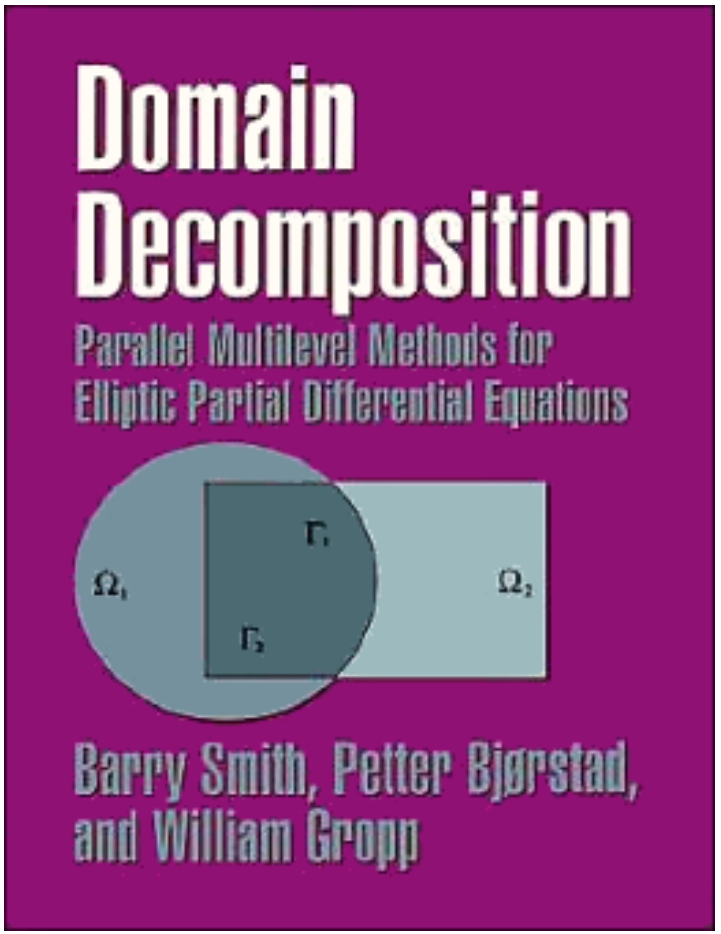
PETSc was developed as a Platform for

Experimentation

We want to experiment with different

- Models
- Discretizations
- Solvers
- Algorithms

These boundaries are often blurred...



PETSc

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)

Software Environment

Operating systems (Linux, Mac, Windows, BSD, proprietary Unix)

Any compiler

Usable from C, C++, Fortran 77/90, Python

Real/complex, single/double/quad precision, 32/64-bit int

System Size

500B unknowns, 75% weak scalability on Jaguar (225k cores) and Jugene (295k cores)

Same code runs performantly on a laptop

Free to everyone (BSD license), open development

PETSc

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.

PETSc

Portable Extensible **Toolkit** for Scientific Computing

Toolset

Algorithms

(Parallel) debugging aids

Low-overhead profiling

Composability

Try new algorithms by choosing from product space

Composing existing algorithms (multilevel, domain decomposition, splitting)

Experimentation

Impossible to pick the solver *a priori*

PETSc's response: expose an algebra of composition

Keep solvers decoupled from physics and discretization

PETSc

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

Documentation and Support

Hundreds of tutorial-style examples

Hyperlinked manual, examples, and manual pages for all routines

Support at `petsc-maint@mcs.anl.gov` `petsc-users@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

Obtaining PETSc

Linux Package Managers

Web: http://mcs.anl.gov/petsc, download tarball

Git: https://bitbucket.org/petsc/petsc

Installing PETSc

```
$> cd /path/to/petsc/workdir
$> git clone https://bitbucket.org/petsc/petsc -b maint
$> cd petsc
```

```
$> export PETSC_DIR=$PWD PETSC_ARCH=mpich-gcc-dbg
$> ./configure
     --with-cc=gcc --with-fc=gfortran --with-cxx=g++
     --download-fblaslapack --download-mpich
     --download-{ml,hypre,superlu}
```

PETSc External Packages

Most packages can be automatically

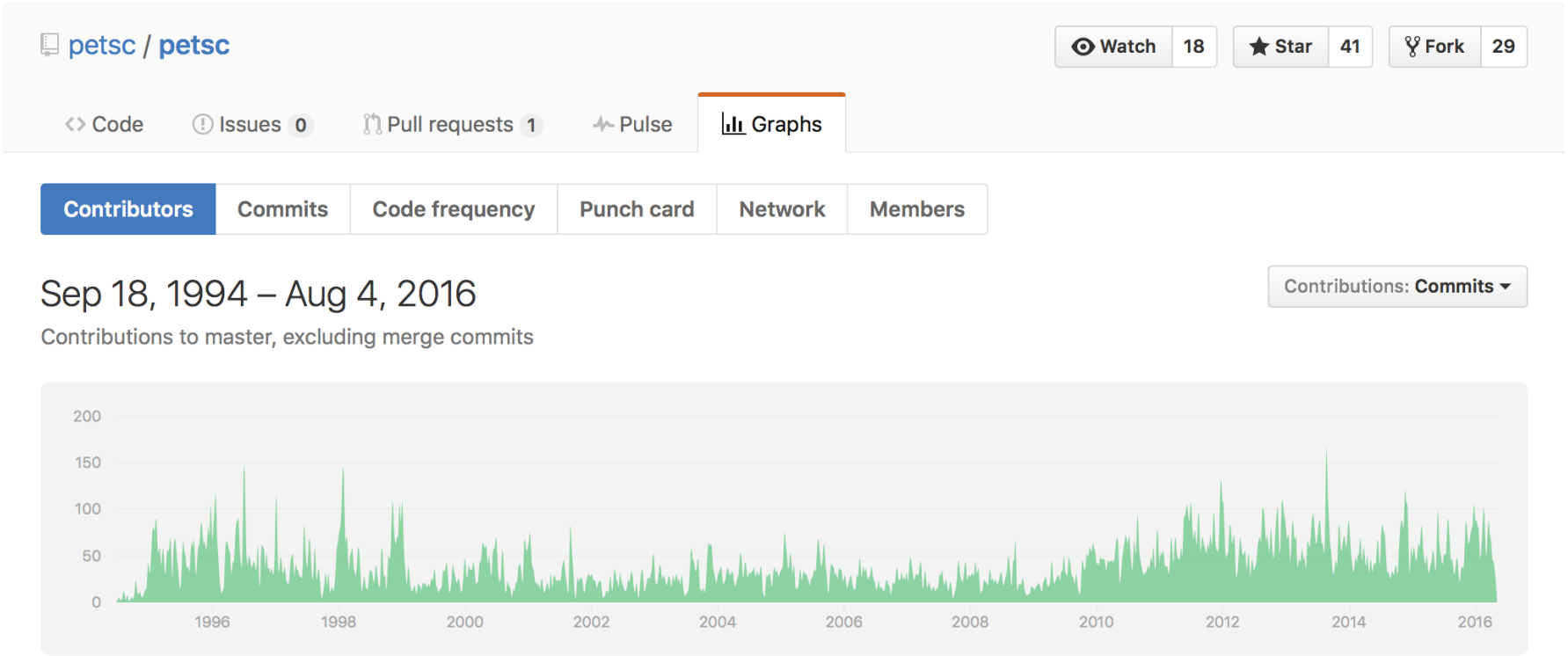
- Downloaded
- Configured and Built (in `$PETSC_DIR/$PETSC_ARCH/externalpackages`)
- Installed with PETSc

Works for (list incomplete)

- petsc4py
- PETSc documentation utilities (Sowing, lgrind, c2html)
- BLAS, LAPACK, BLACS, ScaLAPACK
- MPICH, MPE, OpenMPI
- ParMetis, Chaco, Party, Scotch, Zoltan
- MUMPS, Spooles, SuperLU, SuperLU_Dist, UMFPack, pARMS
- PaStiX, BLOPEX, FFTW, SPRNG
- HYPRE, ML, SPAI
- SUNDIALS
- Triangle, TetGen, FIAT
- HDF5, Boost

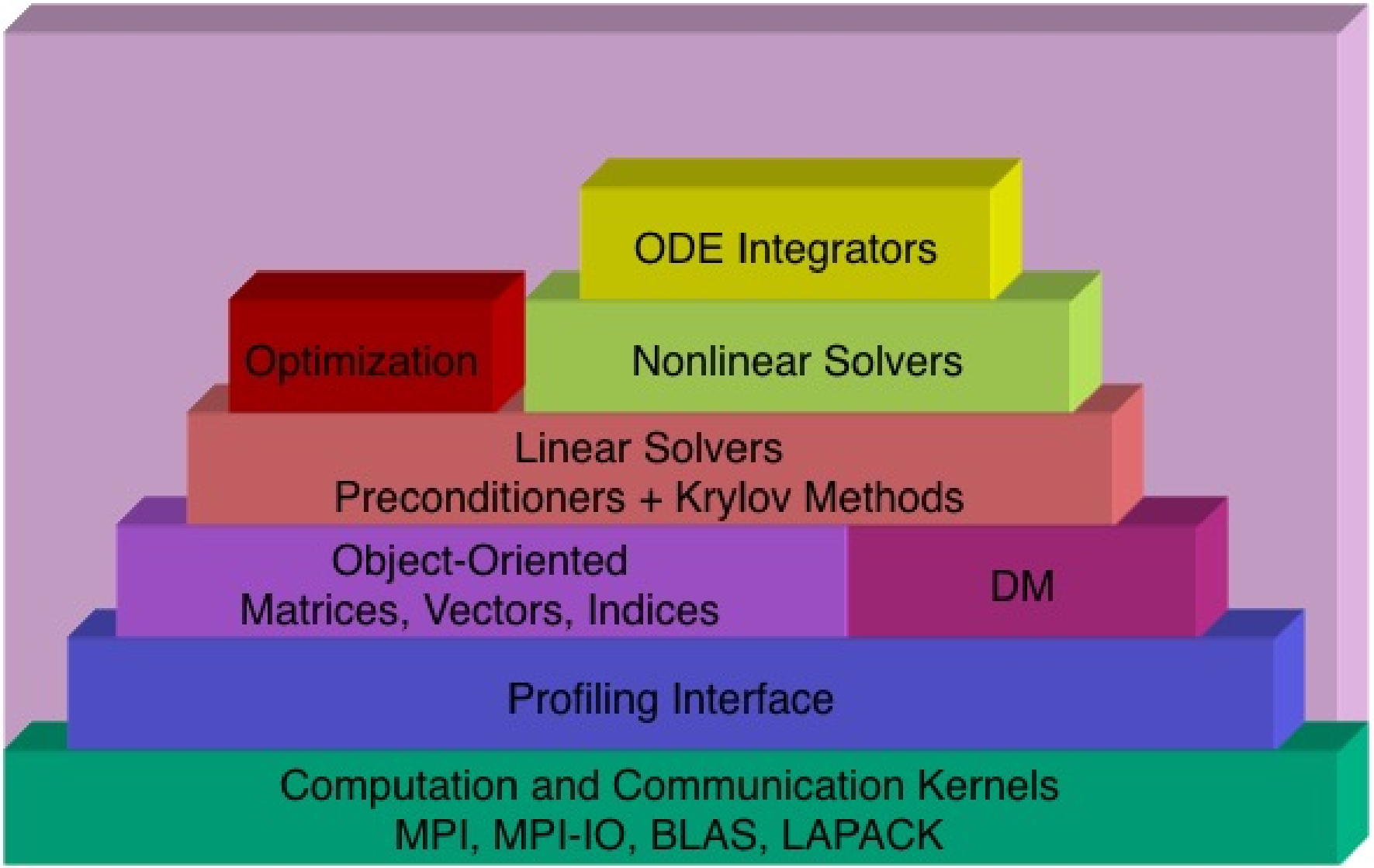
Commits to the PETSc Repository

Graph of commit histroy



PETSc Pyramid

PETSc Structure



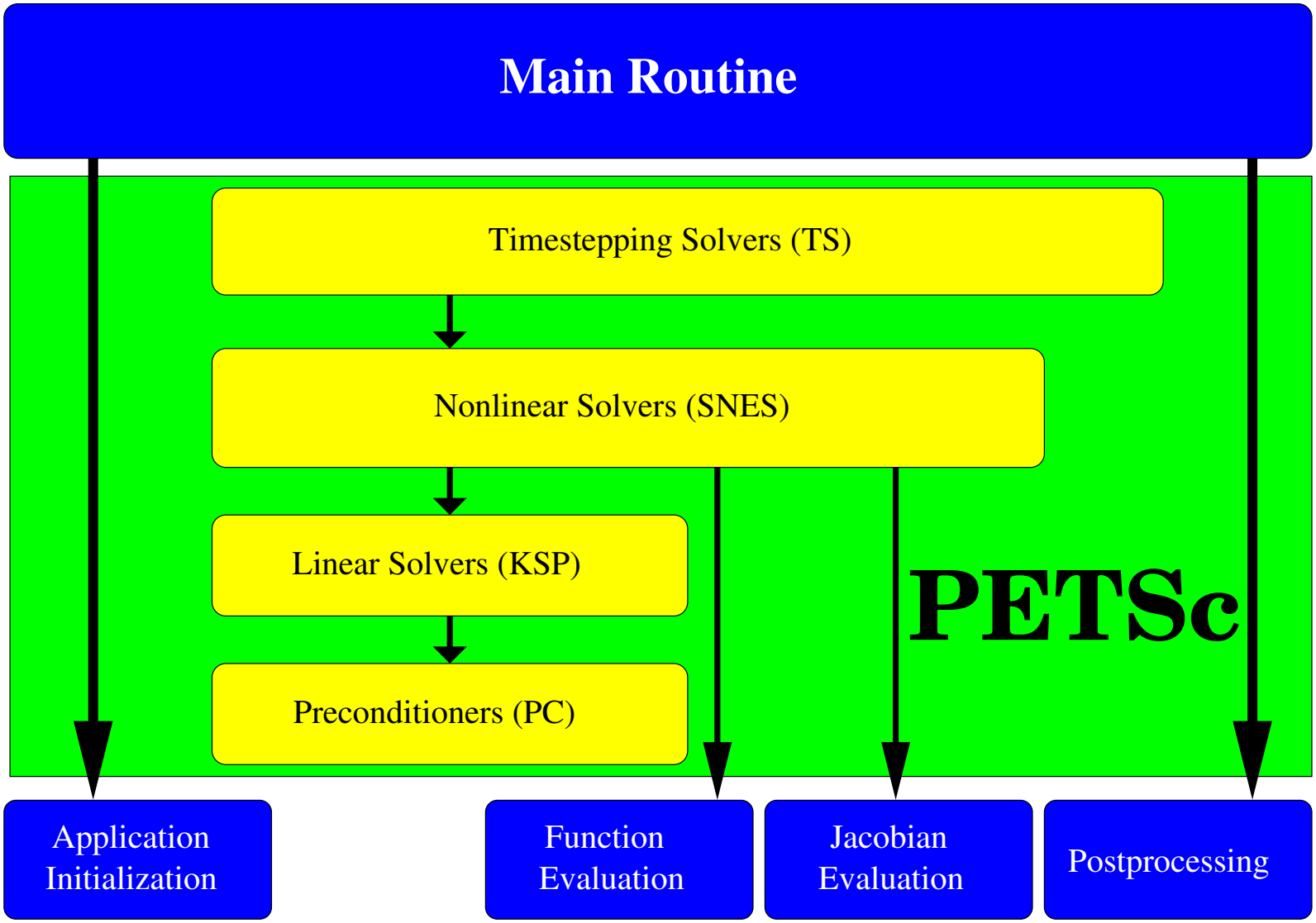
Library Design

Numerical libraries should interact at a higher level than MPI

- MPI coordinates data movement and synchronization for data parallel applications

- Numerical libraries should coordinate access to a given data structure
  - MPI can handle data parallelism and something else (runtime engine) handle task parallelism
  - Algorithm should be data structure neutral, but its main operation is still to structure access

Flow Control for a PETSc Application



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 memory leaks, unused options
- Shutdown and release resources

Recommend initializing PETSc only once

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.

Vectors and Matrices

Vector Algebra

What are PETSc vectors?

Fundamental objects representing field solutions, right-hand sides, etc.

Each process locally owns a subvector of contiguous global data

How do I create vectors?

```
VecCreate(MPI_Comm, Vec *)
VecSetSizes(Vec, int n, int N)
VecSetType(Vec, VecType typeName)
```

`VecSetFromOptions(Vec)` – Can set the type at runtime

A PETSc Vec

Has a direct interface to the values

Supports all vector space operations

`VecDot()`, `VecNorm()`, `VecScale()`

Has unusual operations, e.g. `VecSqrt()`, `VecWhichBetween()`

Communicates automatically during assembly

Has customizable communication (scatters)

Collectivity

MPI communicators (`MPI_Comm`) specify collectivity

- Processes involved in a computation

Constructors are collective over a communicator

- `VecCreate(MPI_Comm comm, Vec *x)`
- Use `PETSC_COMM_WORLD` for all processes and `PETSC_COMM_SELF` for one

Some operations are collective, while others are not

- collective: `VecNorm()`
- not collective: `VecGetLocalSize()`

Sequences of collective calls must be in the same order on each process

Parallel Assembly

Vectors and Matrices

- Processes may set an arbitrary entry
  - Must use proper interface
- Entries need not be generated locally
  - Local meaning the process on which they are stored
- PETSc automatically moves data if necessary
  - Happens during the assembly phase

Vector Assembly

A three step process

- Each process sets or adds values
- Begin communication to send values to the correct process
- Complete the communication

```
VecSetValues(Vec v, int n, int rows[], PetscScalar values[], mode)
```

- `mode` is either INSERT_VALUES or ADD_VALUES

Two phase assembly allows overlap of communication and computation

```
VecAssemblyBegin(Vec v)
VecAssemblyEnd(Vec v)
```

One Way to Set the Elements of a Vector

```
VecGetSize(x, &N);
MPI_Comm_rank(PETSC_COMM_WORLD, &rank);
if (rank == 0) {
  for(i = 0, val = 0.0; i < N; i++, val += 10.0) {
    VecSetValues(x, 1, &i, &val, INSERT_VALUES);
  }
}
/* These routines ensure that the data is distributed to the
other processes */
VecAssemblyBegin(x);
VecAssemblyEnd(x);
```

A Better Way to Set the Elements of a Vector

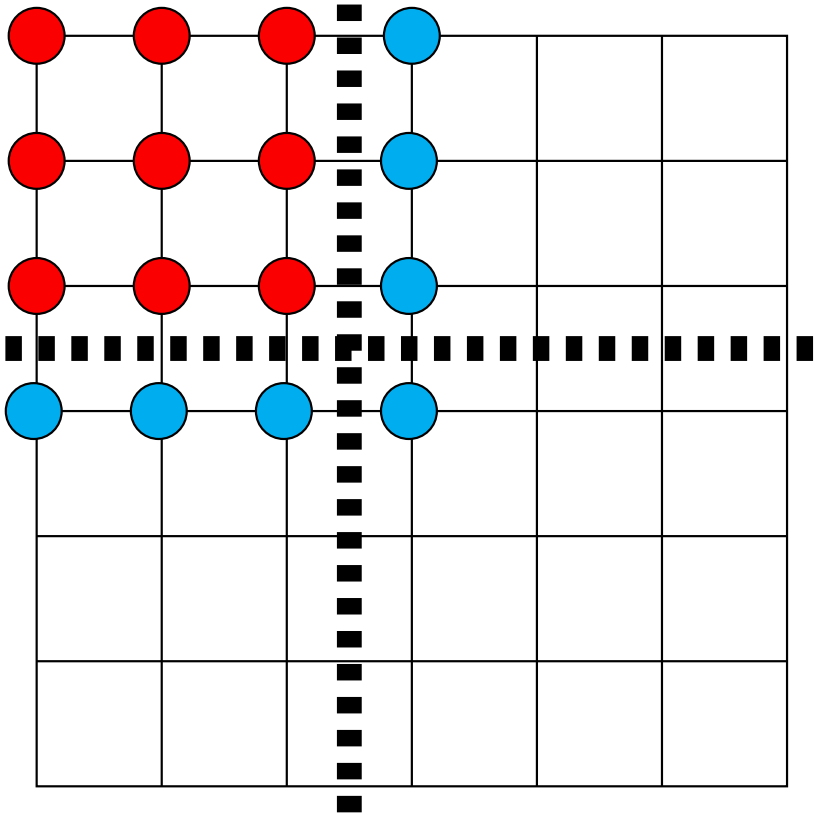
```
VecGetOwnershipRange(x, &low, &high);
for(i = low,val = low*10.0; i < high; i++,val += 10.0) {
  VecSetValues(x, 1, &i, &val, INSERT_VALUES);
}
/* These routines ensure that the data is distributed to the
other processes */
VecAssemblyBegin(x);
VecAssemblyEnd(x);
```

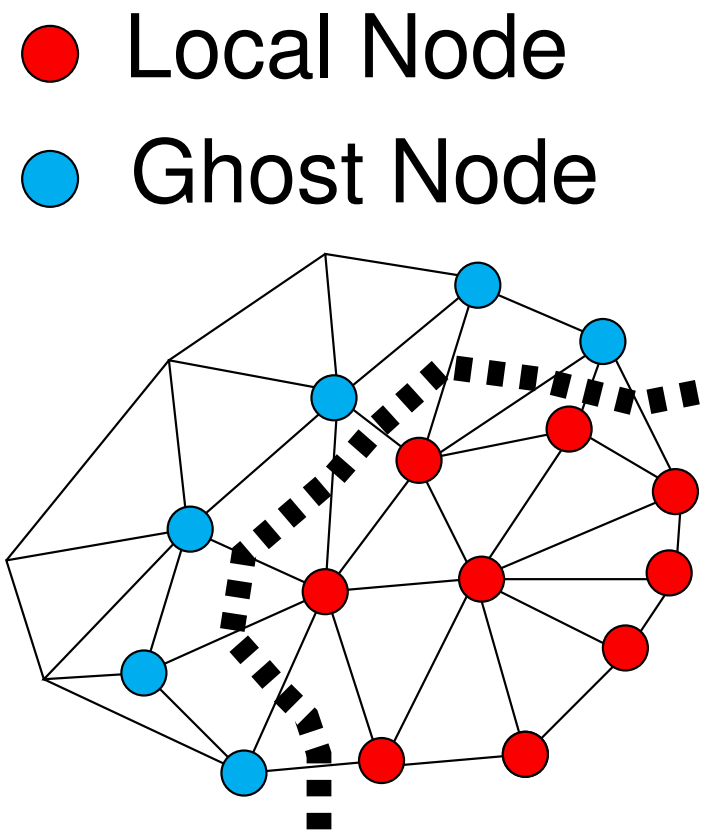
Ghost Values

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





Working With Local Vectors

It is sometimes more efficient to directly access local storage of a `Vec`.

- PETSc allows you to access the local storage with

  ```
  VecGetArray(Vec, double *[])
  ```

- You must return the array to PETSc when you finish

  ```
  VecRestoreArray(Vec, double *[])
  ```

- Allows PETSc to handle data structure conversions
  - Commonly, these routines are inexpensive and do not involve a copy

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 its action in terms of entries (usually stored in a sparse format).

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

`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.

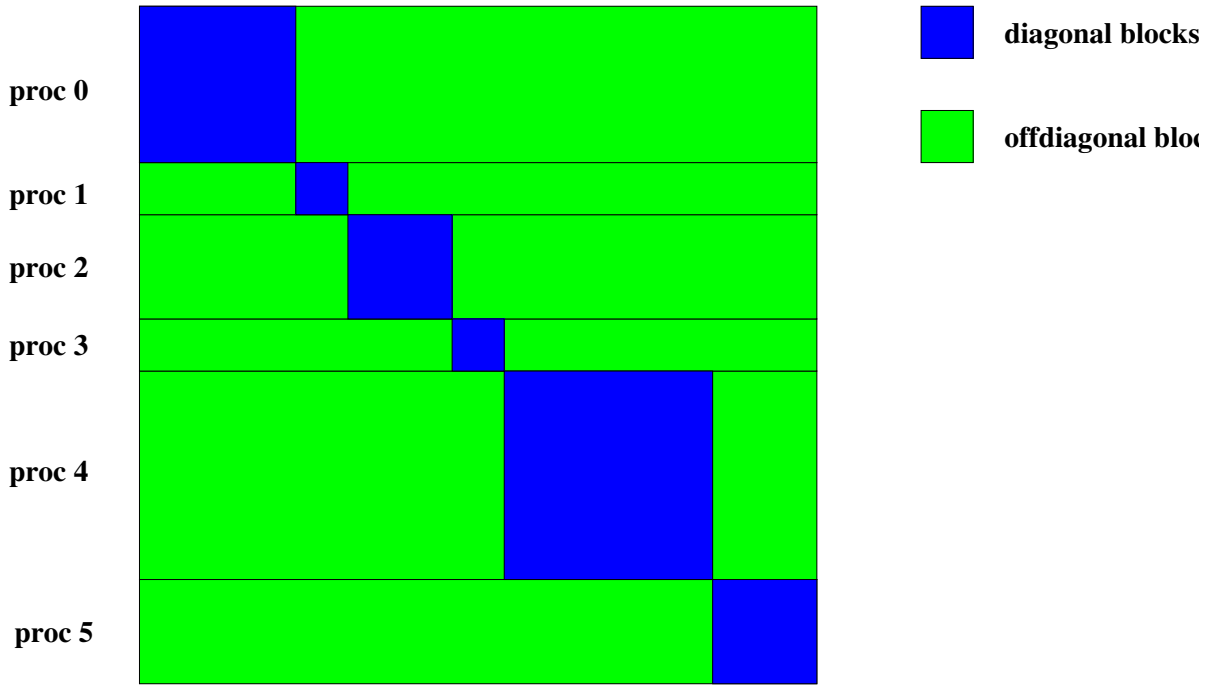
- AIJ, Block AIJ, Symmetric Block AIJ,
- Dense, Elemental
- Matrix-Free
- etc.

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

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

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

A More Efficient Way

```
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);
```

Advantages

All ranks busy: Scalable!

Amount of code essentially unchanged

Iterative Solvers

Matrices

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

For any popular Krylov method $\mathcal{K}$, there is a matrix of size m, such that $\mathcal{K}$ outperforms all other methods by a factor at least $\mathcal{O}(\sqrt{m})$ [Nachtigal et. al., 1992]

Typically...

The action $y \leftarrow Ax$ can be computed in $\mathcal{O}(m)$

Aside from matrix multiply, the n^{th} iteration requires at most $\mathcal{O}(mn)$

Linear Solvers - Krylov Methods

Using PETSc linear algebra, just add:

```
KSPSetOperators(KSP ksp, Mat A, Mat M)
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`

Linear Solvers in PETSc KSP

Linear Solvers in PETSc KSP (Excerpt)

- Richardson
- Chebychev
- Conjugate Gradient
- BiConjugate Gradient
- Generalized Minimum Residual Variants
- Transpose-Free Quasi-Minimum Residual
- Least Squares Method
- Conjugate Residual

Complete table of solvers

Preconditioners

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.

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

Preconditioners in PETSC PC

Preconditioners in PETSc PC (Excerpt)

- Jacobi
- block Jacobi
- SOR
- Additive Schwarz
- Incomplete factorizations (ILU(k), ICC(k))
- Multigrid (geometric, algebraic)
- Physics-based splitting
- Approximate inverses
- Substructuring
- Matrix-free (infrastructure for custom approaches provided by user)

Complete table of solvers

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]`

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

Gauss-Seidel inspired, works when fields are loosely coupled

Factorization: `-pc_fieldsplit_type schur`

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

robust (exact factorization), can often drop lower block
how to precondition S which is usually dense?

interpret as differential operators, use approximate commutators

Solver Composition

Unintrusive composition of multigrid and block preconditioning

We can build many preconditioners from the literature
on the command line

User code does not depend on matrix format, preconditioning method, nonlinear solution method, time integration method (implicit or IMEX), or size of coupled system (except for driver).

Nonlinear Solvers

Nonlinear Solvers in PETSc SNES

Nonlinear solvers in PETSc SNES

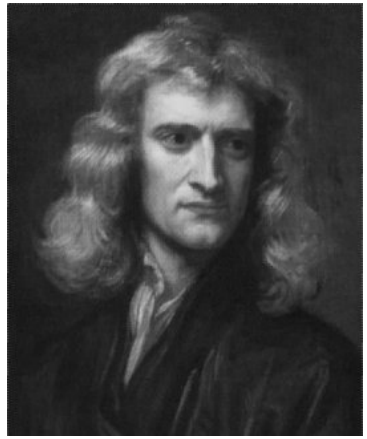
LS, TR	Newton-type with line search and trust region
NRichardson	Nonlinear Richardson, usually preconditioned
VIRS, VISS	reduced space and semi-smooth methods for variational inequalities
QN	Quasi-Newton methods like BFGS
NGMRES	Nonlinear GMRES
NCG	Nonlinear Conjugate Gradients
GS	Nonlinear Gauss-Seidel/multiplicative Schwarz sweeps
FAS	Full approximation scheme (nonlinear multigrid)
MS	Multi-stage smoothers, often used with FAS for hyperbolic problems
Shell	Your method, often used as a (nonlinear) preconditioner

Basic Solver Usage

We will illustrate basic solver usage with `SNES`.

- Use `SNESSetFromOptions()` so that everything is set dynamically
  - Use `-snes_type` to set the type or take the default
- Override the tolerances
  - Use `-snes_rtol` and `-snes_atol`
- View the solver to make sure you have the one you expect
  - Use `-snes_view`
- For debugging, monitor the residual decrease
  - Use `-snes_monitor`
  - Use `-ksp_monitor` to see the underlying linear solver

Newton iteration: Workhorse of SNES



Standard form of a nonlinear system

$$F(u) = 0$$

Iteration

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

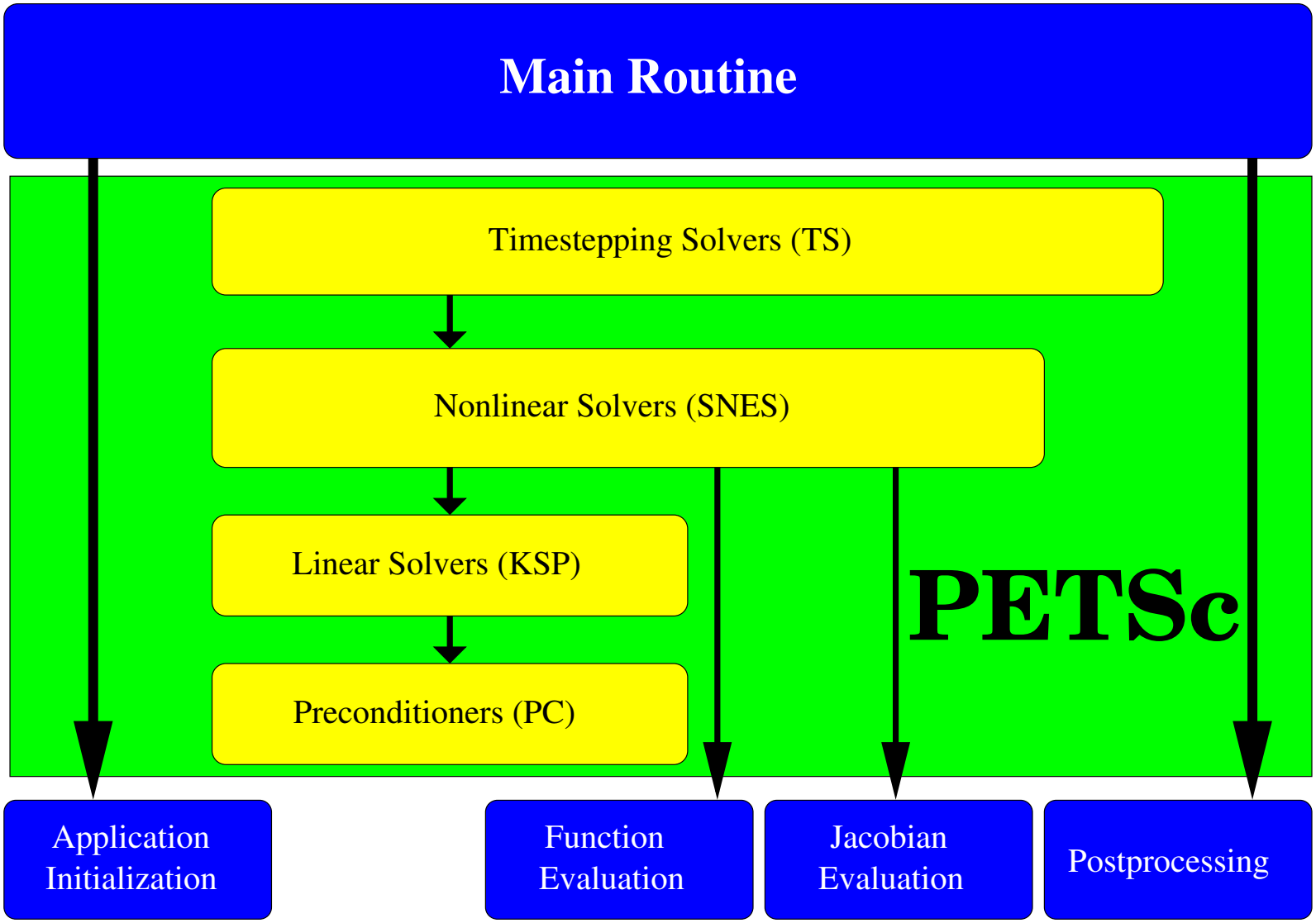
Quadratically convergent near a root: $\left|u^{n+1} - u^*\right| \in \mathcal{O}\left(\left|u^n - u^*\right|^2\right)$

Picard is the same operation with a different $J(u)$

Example (Nonlinear Poisson)

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

Flow Control for a PETSc Application



SNES Paradigm

SNES Interface based upon Callback Functions

`FormFunction()`, set by `SNESSetFunction()`

`FormJacobian()`, set by `SNESSetJacobian()`

Evaluating 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

$$F(u) = 0$$

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

User-provided function calculating the Jacobian Matrix

```
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`

Alternatives

a builtin sparse finite difference approximation (“coloring”)

automatic differentiation (ADIC/ADIFOR)

Time Integration

Time Integration in PETSc

ODE forms supported

$$G(t,x,\dot{x}) = F(t,x)$$
$$J_\alpha = \alpha G_{\dot{x}} + G_x \text{ or}$$
$$M(t)\dot{x} = F(t,x)$$
$$J_\alpha = \alpha M \text{ or}$$
$$\dot{x} = F(t,x)$$

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,void *ctx);
```

Motivation for IMEX Time Integration

Explicit methods are easy and accurate, but must resolve all time scales

- Reactions, acoustics, incompressibility

Implicit methods are robust

- Mathematically good for stiff systems
- Harder to implement, need efficient solvers

Implicit-explicit methods blend the benefits of both

- Benefits of explicit for parts of problems that are not stiff
- Benefits of implicit where needed
- Good approach for multiphysics applications

Some TS Methods

TSSSPRK104 10-stage, fourth order, low-storage, optimal explicit SSP Runge-Kutta $c_{\text{eff}} = 0.6$ (Ketcheson 2008)

TSARKIMEX2E second order, one explicit and two implicit stages, L-stable, optimal (Constantinescu)

TSARKIMEX3 (and 4 and 5), L-stable (Kennedy and Carpenter, 2003)

TSROSWRA3PW three stage, third order, for index-1 PDAE, A-stable, $R(\infty) = 0.73$, second order strongly A-stable embedded method (Rang and Angermann, 2005)

TSROSWRA34PW2 four stage, third order, L-stable, for index 1 PDAE, second order strongly A-stable embedded method (Rang and Angermann, 2005)

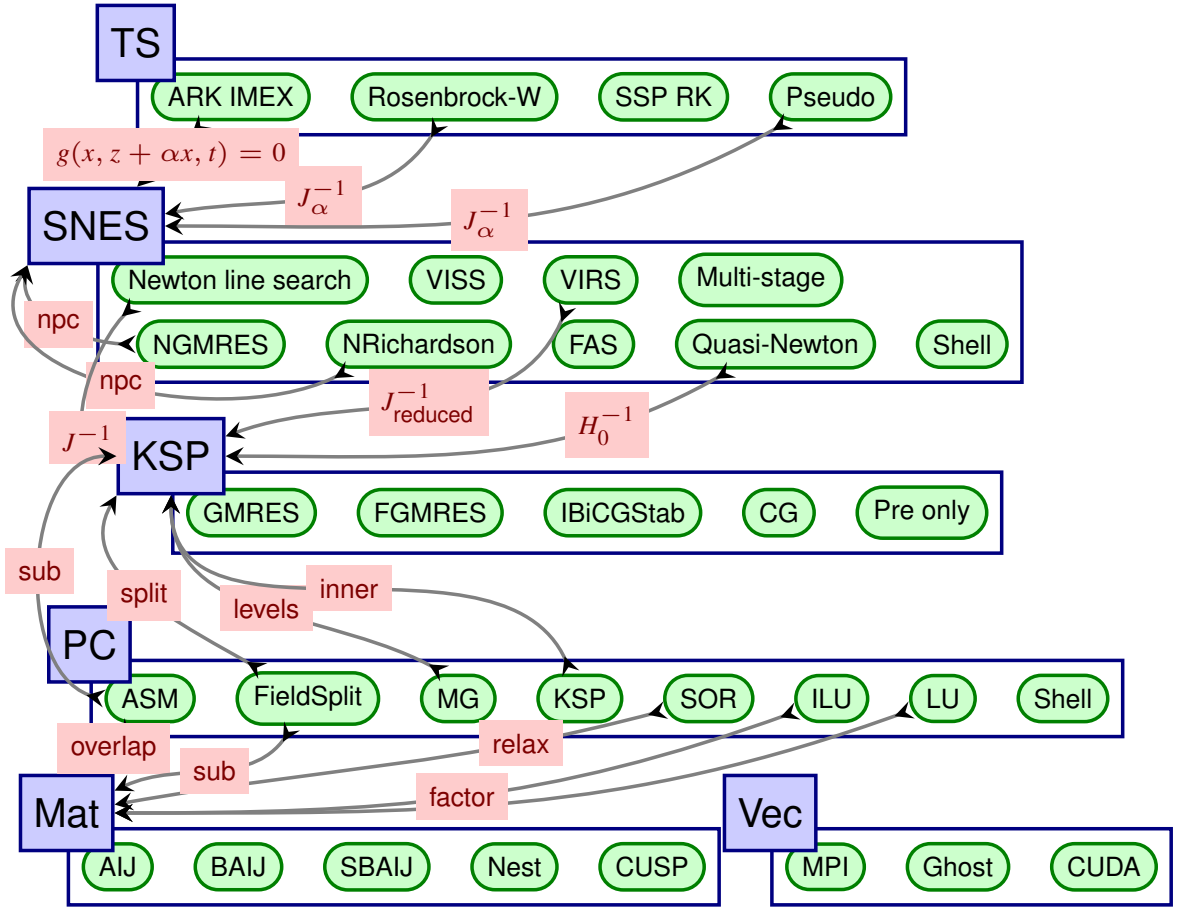
TSROSWLLSSP3P4S2C four stage, third order, L-stable implicit, SSP explicit, L-stable embedded method (Constantinescu)

TS solver options

Interactions among Composable Solvers

Interface to highest level that makes sense for your problem: TS preferable

Customize algorithmic layers as needed: SNES, KSP, PC
Can make choices at runtime for algorithms and data structures



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

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

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

Better To Use than PETSc

Use the package with the highest level of abstraction that uses PETSc

- Eigenvalues - SLEPc
- Finite Elements - Deal.II, Libmesh, FEniCS, PETSc-FEM, OOFEM,
- Finite Elements and Multiphysics - MOOSE
- Finite Volumes - FreeCFD, OpenFVM
- Wave Propagation - PyClaw
- Micromagnetics - MagPar
- Numerical Optimization - Toolkit for Advanced Optimization (TAO)
  - TAO is now part of PETSc

Nonlinear Optimization: TAO

Developers: Todd Munson, Jason Sarich, Stefan Wild

Nonlinear Optimization

Solves Nonlinear Optimization Problems:

$$f : \mathbb{R}^N \mapsto \mathbb{R}$$
$$\min_{x \in \mathbb{R}^N} f(x)$$

With optional variable bounds:

$$\text{subject to} \quad x_l \leq x \leq x_u \qquad \text{(bounds)}$$

Or complementarity constraints:

$$\begin{aligned} F_i(x^*) &\geq 0 \qquad \text{if } x_i^* = \ell_i \\ F_i(x^*) &= 0 \quad \text{if } \ell_i < x_i^* < u_i \\ F_i(x^*) &\leq 0 \qquad \text{if } x_i^* = u_i. \end{aligned}$$

Also some support for PDE-constrained applications and general contraints

TAO Algorithms

TAO provides a suite of (iterative) nonlinear optimization algorithms. Typically, each iteration involves calculating a *search direction* d_k, then function values and gradients along that direction are calculated until desired conditions are met.

- Newton's Method

  Calculate the direction d_{k+1} by solving the system:

  $$\nabla^2 f(x_k) d_{k+1} = -\nabla f(x_k)$$

- Quasi-Newton Methods

  Use approximate Hessian $B_k \approx \nabla^2 f(x_k)$. Choose a formula for B_k so that B_k relies on first derivative information only, can be easily stored and $B_k d_{k+1} = -\nabla f(x_k)$ can be easily solved.

- Conjugate Gradient

- Derivative Free

Solvers available in TAO

	handles constraints	requires gradient	requires Hessian
Quasi-Newton (lmvm)	no	yes	no
Newton Line Search (nls)	no	yes	yes
Newton Trust Region (ntr)	no	yes	yes
Newton Trust with Line Search (ntl)	no	yes	yes
Conjugate Gradient (cg)	no	yes	no
Nelder-Mead (nm)	no	no	no
Quasi-Newton (blmvm)	bounds	yes	no
Newton Trust Region (tron)	bounds	yes	yes
Conjugate Gradient (gpcg) (Quadratic objective only)	bounds	yes	no
Model-based derivative free nonlinear least-squares (pounders)	yes	no	no
Semismooth – Feasibility-enforced (SSFLS)	complementarity	yes	yes
Semismooth – Feasibility not enforced (SSILS)	complementarity	yes	yes
Active-Set Semismooth – Feasibility-enforced (ASFLS)	complementarity	yes	yes
Active-Set Semismooth – Feasibility not enforced (ASILS)	complementarity	yes	yes
Linearly Constrained Lagrangian	pde		
Interior Point Method (ipm)	general	yes	yes

Manual pages for TAO

Topology Abstractions: Distributed Arrays

What is a DM?

Interface for linear algebra to talk to grids

Defines (topological part of) a finite-dimensional function space

- Get an element from this space: `DMCreateGlobalVector()`

Provides parallel layout

Refinement and coarsening

- `DMRefine()`, `DMCoarsen()`

Ghost value coherence

- `DMGlobalToLocalBegin()`

Matrix preallocation:

- `DMCreateMatrix()` (formerly `DMGetMatrix()`)

Topology Abstractions

`DMDA`

- Abstracts Cartesian grids in 1, 2, or 3 dimension
- Supports stencils, communication, reordering
- Nice for simple finite differences

`DMPLEX`

- Abstracts general topology in any dimension
- Also supports partitioning, distribution, and global orders
- Allows aribtrary element shapes and discretizations

`DMCOMPOSITE`

- Composition of two or more DMs

`DMNetwork` - for discrete networks like power grids and circuits

`DMMoab` - interface to the MOAB unstructured mesh library

Manual pages for DM

Distributed Array

Interface for topologically structured grids

Defines (topological part of) a finite-dimensional function space

Get an element from this space: `DMCreateGlobalVector()`

Provides parallel layout

Refinement and coarsening

`DMRefineHierarchy()`

Ghost value coherence

`DMGlobalToLocalBegin()`

Matrix preallocation

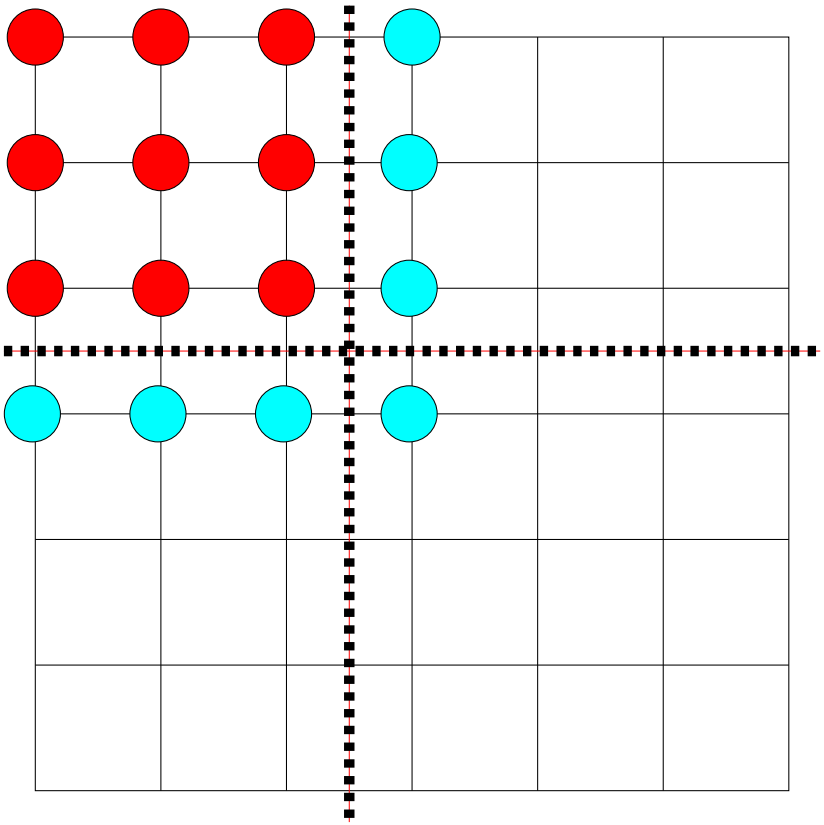
`DMCreateMatrix()` (formerly `DMGetMatrix()`)

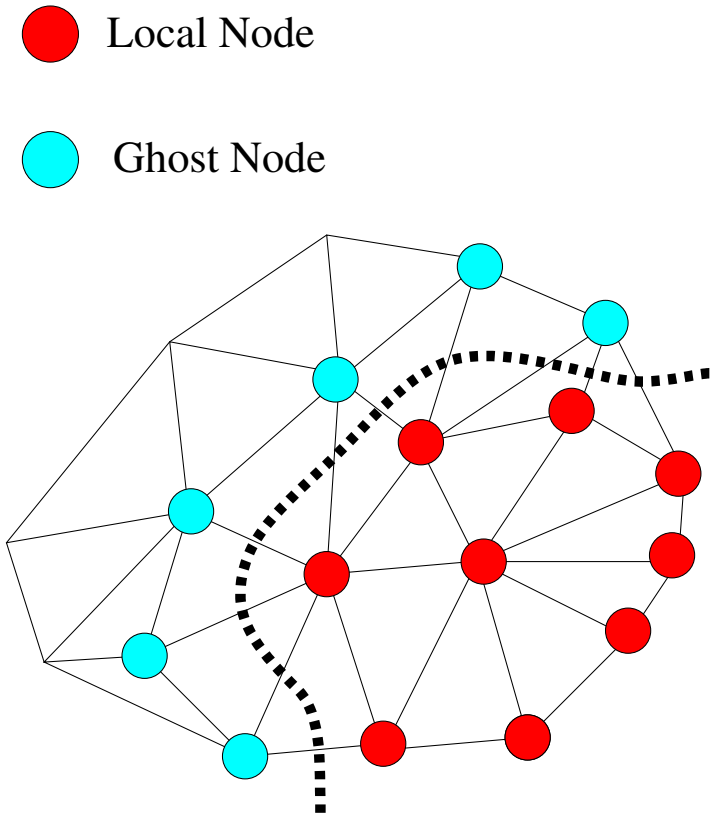
Ghost Values

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





DMDA 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

DMDA 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

DM Vectors

The DM 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

```
DMCreateGlobalVector(DM dm, Vec *gvec)
```

Local vectors are sequential (and usually temporary)

Each process stores its local portion plus ghost values

```
DMCreateLocalVector(DM dm, Vec *lvec)
```

includes ghost values!

Coordinate vectors store the mesh geometry

```
DMDAGetCoordinates(DM dm, Vec *coords)
```

Can be manipulated with their own DMDA

```
DMDAGetCoordinateDA(DM dm,DM *cda)
```

Updating Ghosts

Two-step Process for Updating Ghosts

enables overlapping computation and communication

`DMGlobalToLocalBegin(dm, gvec, mode, lvec)`

`gvec` provides the data

`mode` is either `INSERT_VALUES` or `ADD_VALUES`

`lvec` holds the local and ghost values

`DMGlobalToLocalEnd(dm, gvec, mode, lvec)`

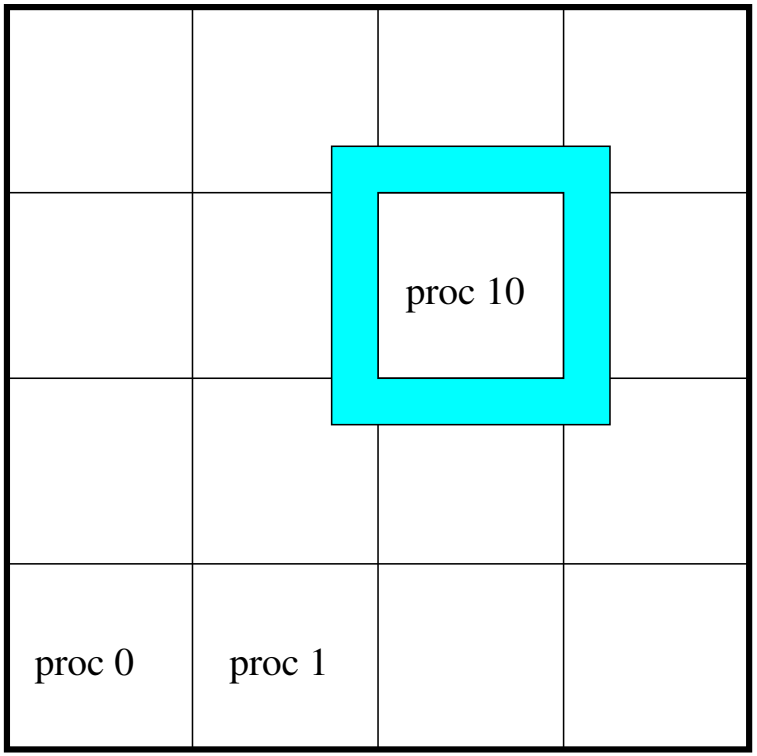
Finishes the communication

Reverse Process

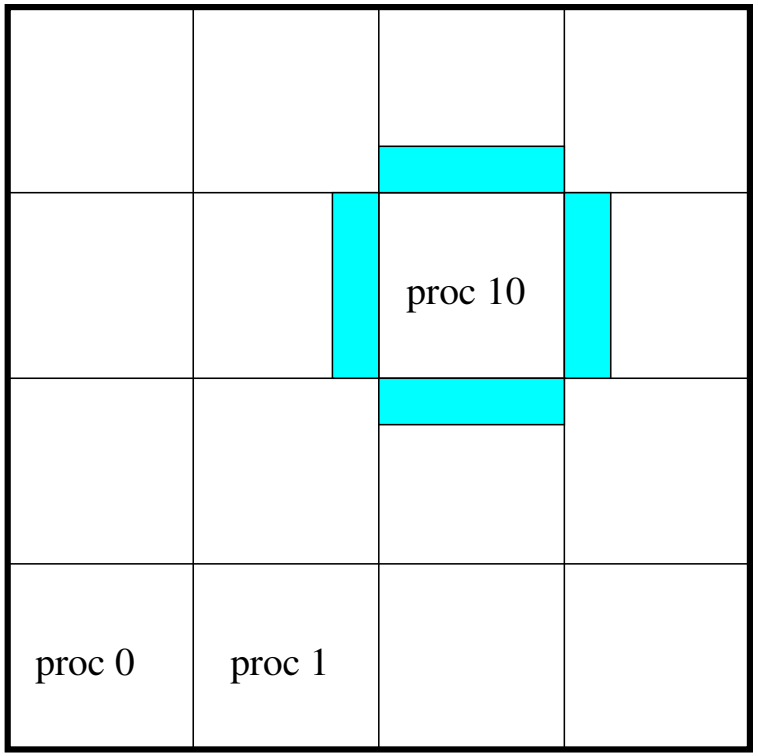
Via `DMLocalToGlobalBegin()` and `DMLocalToGlobalEnd()`.

DMDA Stencils

Available Stencils



Box Stencil



Star Stencil

Creating a DMDA

```
DMDACreate2d(comm, xbdy, ybdy, type, M, N, m, n,
             dof, s, lm[], ln[], DA *da)
```

`xbdy,ybdy`: Specifies periodicity or ghost cells

`DM_BOUNDARY_NONE`, `DM_BOUNDARY_GHOSTED`, `DM_BOUNDARY_MIRROR`, `DM_BOUNDARY_PERIODIC`

`type`

Specifies stencil: `DMDA_STENCIL_BOX` or `DMDA_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

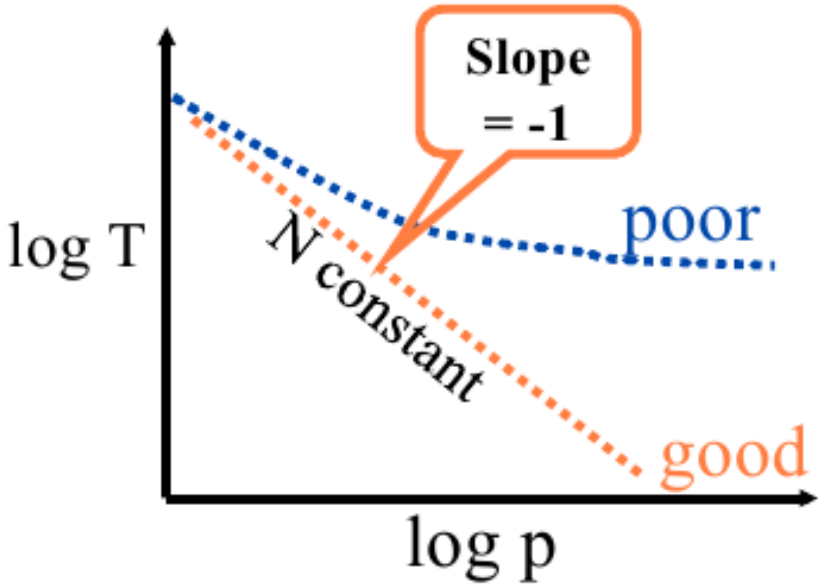
Use `NULL` for the default

Debugging and Profiling

Scalability Definitions

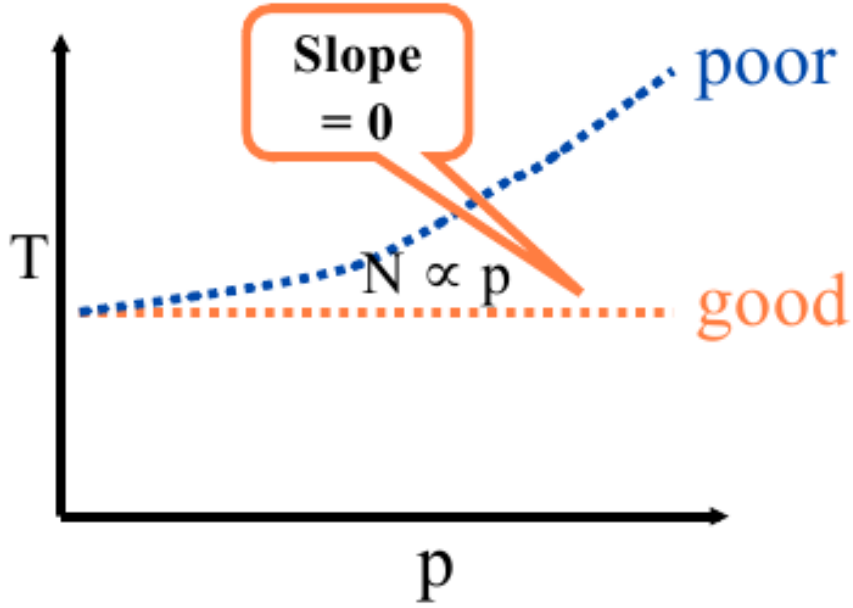
Strong scalability

- 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

PETSc Debugging

By default, a debug build is provided

Launch the debugger

```
-start_in_debugger [gdb,lldb,dbx,noxterm]
-on_error_attach_debugger [gdb,lldb,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

Profiling

- Use `-log_view` [previously `-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_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

PETSc Profiling

```
Event                Count      Time (sec)     Flops                             --- Global ---  --- Stage ---   Total
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   Avg len Reduct  %T %F %M %L %R  %T %F %M %L %R Mflop/s
------------------------------------------------------------------------------------------------------------------------
--- 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   0  0  0  0  1 73954
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   1  1  0  0 27 128346
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   1  1  0  0 61 112366
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   0  0  0  0  0 414021
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   0  0 96 53  0      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   0  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  22 44 78 49  0 220314
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   0  1  6  0  0 192658
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   1  1  7  0  0 100638
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  18 41  0  0  0 256792
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   0  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   4  9  0  0  0 241671
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   2  0  3 47  5      0
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   1  0  0  0  1      0
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  99100 99100 97 113626
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  10  1  4  4  3 13592
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  10  0  3  3  0      0
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  26  0  7 48  5    429
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   1  2  0  0 27 212399
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   0  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  62 99 88 48 88 178078
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  12  9  0  0  1 79715
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   4  9  0  0  1 237761
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  49 88 81 39 79 200691
```

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

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

Follow Up; Getting Help

`http://www.mcs.anl.gov/petsc`

Public questions: `petsc-users@mcs.anl.gov`, with searchable archives

Private questions: `petsc-maint@mcs.anl.gov`

Hands On Exercises

Instructions

`http://www.mcs.anl.gov/petsc/petsc-3.7-atpesc2016/tutorials/HandsOnExercise.html`

Examples

Linear Poisson equation on a 2D grid

src/ksp/ksp/examples/tutorials/ex50.c

Nonlinear ODE arising from a time-dependent one-dimensional PDE

src/ts/examples/tutorials/ex2.c

Nonlinear PDE on a structured grid

src/snes/examples/tutorials/ex19.c

Linear Stokes-type PDE on a structured grid

src/ksp/ksp/examples/tutorials/ex42.c

Nonlinear time-dependent PDE on Unstructured Grid

src/ts/examples/tutorials/ex11.c