

PETSc: Platform for Scientific Computing

Matthew Knepley

Computation Institute
University of Chicago

ME 964: High Performance Computing
for Engineering Applications
University of Wisconsin – Madison
April 21, 2011



Outline

1 Introduction

- Who uses and develops PETSc?
- Stuff for Windows
- How can I get PETSc?
- How do I Configure PETSc?
- How do I Build PETSc?
- How do I run an example?
- How do I get more help?

2 Version Control

3 Vector Algebra

4 Matrix Algebra

What I Need From You

- Tell me if you do not understand
- Tell me if an example does not work
- Suggest better wording or **figures**
- Followup problems at petsc-maint@mcs.anl.gov

Ask Questions!!!

- Helps **me** understand what you are missing
- Helps **you** clarify misunderstandings
- Helps **others** with the same question

How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions
- Help install
- Guide design of large scale codes
- Answer email at petsc-maint@mcs.anl.gov

How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions
- Help install
- Guide design of large scale codes
- Answer email at petsc-maint@mcs.anl.gov

How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions
- Help install
- Guide design of large scale codes
- Answer email at petsc-maint@mcs.anl.gov

How We Can Help at the Tutorial

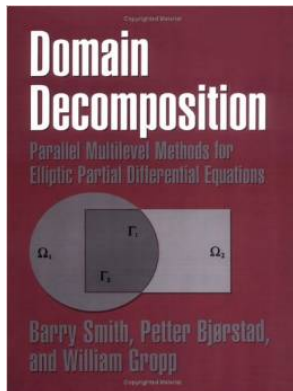
- Point out relevant documentation
- Quickly answer questions
- Help install
- Guide design of large scale codes
- Answer email at petsc-maint@mcs.anl.gov

How did PETSc Originate?

PETSc was developed as a Platform for Experimentation

We want to experiment with different

- Models
- Discretizations
- Solvers
- Algorithms
 - which blur these boundaries



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

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>)

What is PETSc?

A freely available and supported research code for the parallel solution of nonlinear algebraic equations

Free

- Download from <http://www.mcs.anl.gov/petsc>
- Free for everyone, including industrial users

Supported

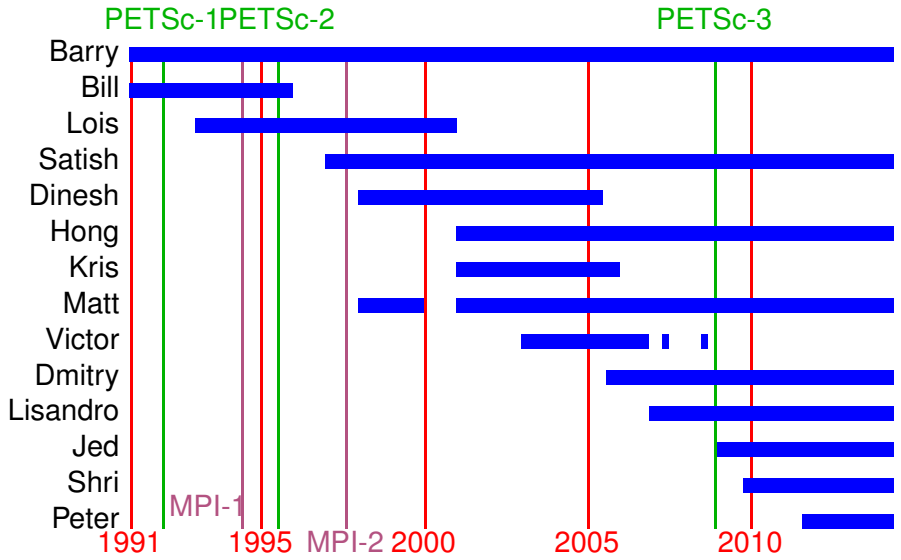
- Hyperlinked manual, examples, and manual pages for all routines
- Hundreds of tutorial-style examples
- Support via email: petsc-maint@mcs.anl.gov

Usable from C, C++, Fortran 77/90, Matlab, Julia, and Python

What is PETSc?

- Portable to any parallel system supporting MPI, including:
 - Tightly coupled systems
 - Cray XT6, BG/Q, NVIDIA Fermi, K Computer
 - Loosely coupled systems, such as networks of workstations
 - IBM, Mac, iPad/iPhone, PCs running Linux or Windows
- PETSc History
 - Begun September 1991
 - Over 60,000 downloads since 1995 (version 2)
 - Currently 400 per month
- PETSc Funding and Support
 - Department of Energy
 - SciDAC, MICS Program, AMR Program, INL Reactor Program
 - National Science Foundation
 - CIG, CISE, Multidisciplinary Challenge Program

Timeline



What Can We Handle?

- PETSc has run implicit problems with over **500 billion** unknowns
 - UNIC on BG/P and XT5
 - PFLOTRAN for flow in porous media
- PETSc has run on over **290,000** cores efficiently
 - UNIC on the IBM BG/P Jugene at Jülich
 - PFLOTRAN on the Cray XT5 Jaguar at ORNL
- PETSc applications have run at 23% of peak (**600 Teraflops**)
 - Jed Brown on NERSC Edison
 - HPGMG code

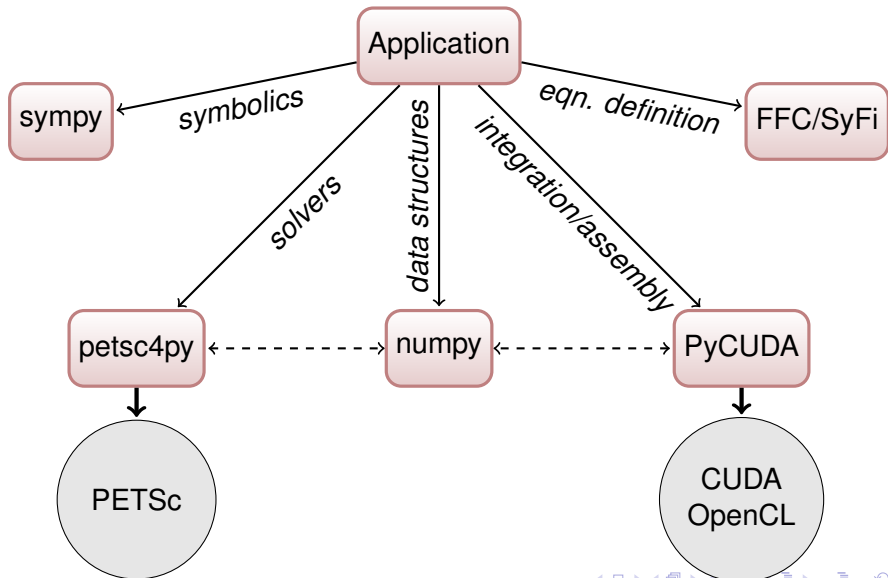
What Can We Handle?

- PETSc has run implicit problems with over **500 billion** unknowns
 - UNIC on BG/P and XT5
 - PFLOTRAN for flow in porous media
- PETSc has run on over **290,000** cores efficiently
 - UNIC on the IBM BG/P Jugene at Jülich
 - PFLOTRAN on the Cray XT5 Jaguar at ORNL
- PETSc applications have run at 23% of peak (**600 Teraflops**)
 - Jed Brown on NERSC Edison
 - HPGMG code

What Can We Handle?

- PETSc has run implicit problems with over **500 billion** unknowns
 - UNIC on BG/P and XT5
 - PFLOTRAN for flow in porous media
- PETSc has run on over **290,000** cores efficiently
 - UNIC on the IBM BG/P Jugene at Jülich
 - PFLOTRAN on the Cray XT5 Jaguar at ORNL
- PETSc applications have run at 23% of peak (**600 Teraflops**)
 - Jed Brown on NERSC Edison
 - HPGMG code

New Model for Scientific Software



Outline

1 Introduction

- **Who uses and develops PETSc?**
- Stuff for Windows
- How can I get PETSc?
- How do I Configure PETSc?
- How do I Build PETSc?
- How do I run an example?
- How do I get more help?

Who Uses PETSc?

Computational Scientists

- Earth Science
 - PyLith (CIG)
 - Underworld (Monash)
 - Magma Dynamics (LDEO, Columbia, Oxford)
- Subsurface Flow and Porous Media
 - STOMP (DOE)
 - PFLOTRAN (DOE)

Who Uses PETSc?

Computational Scientists

- CFD
 - Firedrake
 - Fluidity
 - OpenFOAM
 - freeCFD
 - OpenFVM
- MicroMagnetics
 - MagPar
- Fusion
 - XGC
 - BOUT++
 - NIMROD

Who Uses PETSc?

Algorithm Developers

- Iterative methods
 - Deflated GMRES
 - LGMRES
 - QCG
 - SpecEst
- Preconditioning researchers
 - Prometheus (Adams)
 - ParPre (Eijkhout)
 - FETI-DP (Klawonn and Rheinbach)

Who Uses PETSc?

Algorithm Developers

- Finite Elements

- libMesh
- MOOSE
- PETSc-FEM
- Deal II
- OOFEM

- Other Solvers

- Fast Multipole Method (PetFMM)
- Radial Basis Function Interpolation (PetRBF)
- Eigensolvers (SLEPc)
- Optimization (TAO)

The PETSc Team



Bill Gropp



Barry Smith



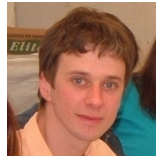
Satish Balay



Jed Brown



Matt Knepley



Lisandro Dalcin



Hong Zhang



Mark Adams



Toby Issac

Outline

1 Introduction

- Who uses and develops PETSc?
- **Stuff for Windows**
- How can I get PETSc?
- How do I Configure PETSc?
- How do I Build PETSc?
- How do I run an example?
- How do I get more help?

Questions for Windows Users

- Have you installed cygwin?
 - Need python, make, and build-utils packages
- Will you use the GNU compilers?
 - If not, remove `link.exe`
 - If MS, check compilers from `cmd` window and use `win32fe`
- Which MPI will you use?
 - You can use `-with-mpi=0`
 - If MS, need to install MPICH2
 - If GNU, can use `-download-mpich`

Outline

1 Introduction

- Who uses and develops PETSc?
- Stuff for Windows
- **How can I get PETSc?**
- How do I Configure PETSc?
- How do I Build PETSc?
- How do I run an example?
- How do I get more help?

Downloading PETSc

- The latest tarball is on the PETSc site:
<http://www.mcs.anl.gov/petsc/download>
- There is a **Debian package** (`aptitude install petsc-dev`)
- There is a **Git development repository**

Cloning PETSc

- The full development repository is open to the public
 - <https://bitbucket.org/petsc/petsc/>
- Why is this better?
 - You can clone to any release (or any specific ChangeSet)
 - You can easily rollback changes (or releases)
 - You can get fixes from us the same day
- All releases are just tags:
 - [Source at tag v3.4.4](#)

Unpacking PETSc

- Just clone development repository

- `git clone http://bitbucket.org/petsc/petsc.git`
- `git clone -rv3.4.4 petsc petsc-3.4.4`

or

- Unpack the tarball

- `tar xzf petsc.tar.gz`

Exercise 1

Download and Unpack PETSc!

Outline

1 Introduction

- Who uses and develops PETSc?
- Stuff for Windows
- How can I get PETSc?
- **How do I Configure PETSc?**
- How do I Build PETSc?
- How do I run an example?
- How do I get more help?

Configuring PETSc

- Set `$PETSC_DIR` to the installation root directory
- Run the configuration utility
 - `$PETSC_DIR/configure`
 - `$PETSC_DIR/configure -help`
 - `$PETSC_DIR/configure -download-mpich`
 - `$PETSC_DIR/configure -prefix=/usr`
- There are many examples on the installation page
- Configuration files are in `$PETSC_DIR/$PETSC_ARCH/conf`
 - Configure header is in `$PETSC_DIR/$PETSC_ARCH/include`
 - `$PETSC_ARCH` has a default if not specified

Configuring PETSc

- You can easily reconfigure with the same options
 - `./$PETSC_ARCH/conf/reconfigure-$PETSC_ARCH.py`
- Can maintain several different configurations
 - `./configure -PETSC_ARCH=linux-fast`
`-with-debugging=0`
- All configuration information is in the logfile
 - `./$PETSC_ARCH/conf/configure.log`
 - **ALWAYS** send this file with bug reports

Automatic Downloads

- Starting in 2.2.1, some packages are 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, OpenMPI
 - ParMetis, Chaco, Jostle, Party, Scotch, Zoltan
 - MUMPS, Spooles, SuperLU, SuperLU_Dist, UMFPack, pARMS
 - BLOPEX, FFTW, SPRNG
 - Prometheus, HYPRE, ML, SPAI
 - Sundials
 - Triangle, TetGen
 - FIAT, FFC, Generator
 - Boost

Exercise 2

Configure your downloaded PETSc.

Outline

1 Introduction

- Who uses and develops PETSc?
- Stuff for Windows
- How can I get PETSc?
- How do I Configure PETSc?
- **How do I Build PETSc?**
- How do I run an example?
- How do I get more help?

Building PETSc

- There is now One True Way to build PETSc:
 - `make`
 - `make install` if you configured with `--prefix`
 - Check build when done with `make test`
- Can build multiple configurations
 - `PETSC_ARCH=linux-fast make`
 - Libraries are in `$PETSC_DIR/$PETSC_ARCH/lib/`
- Complete log for each build is in logfile
 - `./$PETSC_ARCH/conf/make.log`
 - ALWAYS send this with bug reports

Exercise 3

Build your configured PETSc.

Exercise 4

Reconfigure PETSc to use ParMetis.

- 1 `linux-c-debug/conf/reconfigure-linux-c-debug.py`
 - `-PETSC_ARCH=linux-parmetis`
 - `-download-metis -download-parmetis`
- 2 `PETSC_ARCH=linux-parmetis make`
- 3 `PETSC_ARCH=linux-parmetis make test`

Outline

1 Introduction

- Who uses and develops PETSc?
- Stuff for Windows
- How can I get PETSc?
- How do I Configure PETSc?
- How do I Build PETSc?
- **How do I run an example?**
- How do I get more help?

Running PETSc

- Try running PETSc examples first
 - `cd $PETSC_DIR/src/snes/examples/tutorials`
- Build examples using make targets
 - `make ex5`
- Run examples using the make target
 - `make runex5`
- Can also run using MPI directly
 - `mpirun ./ex5 -snes_max_it 5`
 - `mpiexec ./ex5 -snes_monitor`

Using MPI

- The **M**essage **P**assing **I**nterface is:
 - a library for parallel communication
 - a system for launching parallel jobs (mpirun/mpiexec)
 - a community standard
- Launching jobs is easy
 - `mpiexec -n 4 ./ex5`
- You should never have to make MPI calls when using PETSc
 - Almost never

Common Viewing Options

- Gives a text representation
 - `-vec_view`
- Generally views subobjects too
 - `-snes_view`
- Can visualize some objects
 - `-mat_view draw::`
- Alternative formats
 - `-vec_view binary:sol.bin:, -vec_view ::matlab,`
`-vec_view socket`
- Sometimes provides extra information
 - `-mat_view ::ascii_info, -mat_view`
`::ascii_info_detailed`
- Use `-help` to see all options

Common Monitoring Options

- Display the residual
 - `-ksp_monitor`, graphically `-ksp_monitor_draw`
- Can disable dynamically
 - `-ksp_monitors_cancel`
- Does not display subsolvers
 - `-snes_monitor`
- Can use the true residual
 - `-ksp_monitor_true_residual`
- Can display different subobjects
 - `-snes_monitor_residual`, `-snes_monitor_solution`,
`-snes_monitor_solution_update`
 - `-snes_monitor_range`
 - `-ksp_gmres_krylov_monitor`
- Can display the spectrum
 - `-ksp_monitor_singular_value`

Exercise 5

Run SNES Example 5 using some custom options.

- 1 `cd $PETSC_DIR/src/snes/examples/tutorials`
- 2 `make ex5`
- 3 `mpiexec ./ex5 -snes_monitor -snes_view`
- 4 `mpiexec ./ex5 -snes_type tr -snes_monitor
-snes_view`
- 5 `mpiexec ./ex5 -ksp_monitor -snes_monitor
-snes_view`
- 6 `mpiexec ./ex5 -pc_type jacobi -ksp_monitor
-snes_monitor -snes_view`
- 7 `mpiexec ./ex5 -ksp_type bicg -ksp_monitor
-snes_monitor -snes_view`

Exercise 6

Create a new code based upon SNES Example 5.

1 Create a new directory

- `mkdir -p /home/knepley/proj/newsim/src`

2 Copy the source

- `cp ex5.c /home/knepley/proj/newsim/src`
- **Add `myStuff.c` and `myStuff2.F`**

3 Create a PETSc makefile

- `bin/ex5: src/ex5.o src/myStuff.o src/myStuff2.o`
- `${CLINKER} -o $@ $^ ${PETSC_SNES_LIB}`
- `include ${PETSC_DIR}/conf/variables`
- `include ${PETSC_DIR}/conf/rules`

To get the project ready-made

```
hg clone http://petsc.cs.iit.edu/petsc/tutorials/SimpleTutorial newsim
```

Outline

1 Introduction

- Who uses and develops PETSc?
- Stuff for Windows
- How can I get PETSc?
- How do I Configure PETSc?
- How do I Build PETSc?
- How do I run an example?
- **How do I get more help?**

Getting More Help

- <http://www.mcs.anl.gov/petsc>
- Hyperlinked documentation
 - Manual
 - Manual pages for every method
 - HTML of all example code (linked to manual pages)
- FAQ
- Full support at petsc-maint@mcs.anl.gov
- High profile users
 - David Keyes
 - Marc Spiegelman
 - Richard Katz
 - Brad Aagaard
 - Aron Ahmadi

Outline

- 1 Introduction
- 2 Version Control**
- 3 Vector Algebra
- 4 Matrix Algebra
- 5 Algebraic Solvers
- 6 SNES
- 7 DA

Location and Retrieval

“Where’s the Tarball”

- Version Control
 - Mercurial, Git, Subversion
- Hosting
 - BitBucket, GitHub, Launchpad
- Community involvement
 - arXiv, PubMed

Distributed Version Control

- CVS/SVN manage a single repository
 - Versioned data
 - Local copy for modification and checkin
- Mercurial manages many repositories
 - Identified by URLs
 - No one *Master*
- Repositories communicate by **ChangeSets**
 - Use `push` and `pull` to move changesets
 - Can move arbitrary changes with *patch queues*

Project Workflow



Figure: Single Repository

Project Workflow

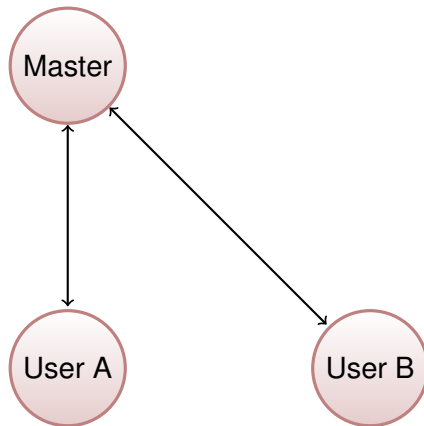


Figure: Master Repository with User Clones

Project Workflow

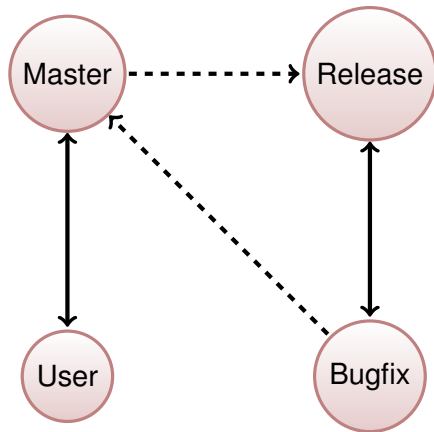


Figure: Project with Release and Bugfix Repositories

Outline

- 1 Introduction
- 2 Version Control
- 3 Vector Algebra**
- 4 Matrix Algebra
- 5 Algebraic Solvers
- 6 SNES
- 7 DA

Vector Algebra

What are PETSc vectors?

- Fundamental objects representing
 - solutions
 - right-hand sides
 - coefficients
- Each process locally owns a subvector of contiguous global data

Vector Algebra

How do I create vectors?

- `VecCreate(MPI_Comm, Vec*)`
- `VecSetSizes(Vec, PetscIntn, PetscInt N)`
- `VecSetType(Vec, VecType typeName)`
- `VecSetFromOptions(Vec)`
 - Can set the type at runtime

Vector Algebra

A PETSc Vec

- Supports all vector space operations
 - `VecDot()`, `VecNorm()`, `VecScale()`
- Has a direct interface to the values
 - `VecGetArray()`, `VecGetArrayF90()`
- Has unusual operations
 - `VecSqrtAbs()`, `VecStrideGather()`
- Communicates automatically during assembly
- Has customizable communication (**PetscSF**, **VecScatter**)

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, PetscInt n, PetscInt rows[],  
            PetscScalar values[], InsertMode mode)
```

- Mode is either `INSERT_VALUES` or `ADD_VALUES`
- Two phases allow overlap of communication and computation
 - `VecAssemblyBegin(Vecv)`
 - `VecAssemblyEnd(Vecv)`

One Way to Set the Elements of a Vector

```
VecGetSize(x, &N);
MPI_Comm_rank(PETSC_COMM_WORLD, &rank);
if (rank == 0) {
    val = 0.0;
    for(i = 0; i < N; ++i) {
        VecSetValues(x, 1, &i, &val, INSERT_VALUES);
        val += 10.0;
    }
}
/* 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);  
val = low*10.0;  
for(i = low; i < high; ++i) {  
    VecSetValues(x, 1, &i, &val, INSERT_VALUES);  
    val += 10.0;  
}  
/* No data will be communicated here */  
VecAssemblyBegin(x);  
VecAssemblyEnd(x);
```

Selected Vector Operations

Function Name	Operation
VecAXPY(Vec y, PetscScalar a, Vec x)	$y = y + a * x$
VecAYPX(Vec y, PetscScalar a, Vec x)	$y = x + a * y$
VecWAYPX(Vec w, PetscScalar a, Vec x, Vec y)	$w = y + a * x$
VecScale(Vec x, PetscScalar a)	$x = a * x$
VecCopy(Vec y, Vec x)	$y = x$
VecPointwiseMult(Vec w, Vec x, Vec y)	$w_i = x_i * y_i$
VecMax(Vec x, PetscInt *idx, PetscScalar *r)	$r = \max r_i$
VecShift(Vec x, PetscScalar r)	$x_i = x_i + r$
VecAbs(Vec x)	$x_i = x_i $
VecNorm(Vec x, NormType type, PetscReal *r)	$r = x $

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 fast and do not involve a copy

VecGetArray in C

```
Vec          v;  
PetscScalar *array;  
PetscInt    n, i;  
  
VecGetArray(v, &array);  
VecGetLocalSize(v, &n);  
PetscSynchronizedPrintf(PETSC_COMM_WORLD,  
    "First element of local array is %f\n", array[0]);  
PetscSynchronizedFlush(PETSC_COMM_WORLD);  
for(i = 0; i < n; ++i) {  
    array[i] += (PetscScalar) rank;  
}  
VecRestoreArray(v, &array);
```

VecGetArray in F77

```
#include "finclude/petsc.h"

Vec          v;
PetscScalar  array(1)
PetscOffset  offset
PetscInt     n, i
PetscErrorCode ierr

call VecGetArray(v, array, offset, ierr)
call VecGetLocalSize(v, n, ierr)
do i=1,n
  array(i+offset) = array(i+offset) + rank
end do
call VecRestoreArray(v, array, offset, ierr)
```

VecGetArray in F90

```
#include "finclude/petsc.h90"  
  
Vec          v;  
PetscScalar  pointer :: array (:)  
PetscInt     n, i  
PetscErrorCode ierr  
  
call VecGetArrayF90(v, array, ierr)  
call VecGetLocalSize(v, n, ierr)  
do i=1,n  
  array(i) = array(i) + rank  
end do  
call VecRestoreArrayF90(v, array, ierr)
```

VecGetArray in Python

```
with v as a:  
    for i in range(len(a)):  
        a[i] = 5.0*i
```

DMDAVecGetArray in C

```
DM          da;
Vec         v;
DMDALocalInfo *info;
PetscScalar **array;

DMDAVecGetArray(da, v, &array);
for(j = info->ys; j < info->ys+info->ym; ++j) {
  for(i = info->xs; i < info->xs+info->xm; ++i) {
    u      = x[j][i];
    uxx    = (2.0*u - x[j][i-1] - x[j][i+1])*hydhx;
    uyy    = (2.0*u - x[j-1][i] - x[j+1][i])*hxdhy;
    f[j][i] = uxx + uyy;
  }
}
DMDAVecRestoreArray(da, v, &array);
```

Outline

- 1 Introduction
- 2 Version Control
- 3 Vector Algebra
- 4 Matrix Algebra**
- 5 Algebraic Solvers
- 6 SNES
- 7 DA

Matrix Algebra

What are PETSc matrices?

- 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 Matrix, etc.
- Supports structures for many packages
 - MUMPS, Spooles, SuperLU, UMFPack, DSCPack

How do I create matrices?

- `MatCreate(MPI_Comm, Mat*)`
- `MatSetSizes(Mat, PetscIntm, PetscInt n, M, N)`
- `MatSetType(Mat, MatType typeName)`
- `MatSetFromOptions(Mat)`
 - Can set the type at runtime
- `MatSeqAIJPreallocation(Mat, PetscIntnz, const PetscInt nnz[])`
- `MatXAIJPreallocation(Mat, bs, dnz[], onz[], dnzu[], onzu[])`
- `MatSetValues(Mat, m, rows[], n, cols [], values [], InsertMode)`
 - **MUST** be used, but does automatic communication

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(Matm, 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(Matm, MatAssemblyType type)`
 - `MatAssemblyEnd(Matm, MatAssemblyType type)`
 - `type` is either `MAT_FLUSH_ASSEMBLY` or `MAT_FINAL_ASSEMBLY`

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);
```

Matrix Storage Layout

- 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

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;
MatGetOwnershipRange(A,&start,&end);
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 performance benefits
 - PETSc has many formats
 - Makes it easy to add new data structures
- Assembly is difficult enough without worrying about partitioning
 - PETSc provides parallel assembly routines
 - 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)

Outline

- 1 Introduction
- 2 Version Control
- 3 Vector Algebra
- 4 Matrix Algebra
- 5 Algebraic Solvers**
- 6 SNES
- 7 DA

Solver Types

- **Explicit:**
 - Field variables are updated using local neighbor information
- **Semi-implicit:**
 - Some subsets of variables are updated with global solves
 - Others with direct local updates
- **Implicit:**
 - Most or all variables are updated in a single global solve

Linear Solvers

Krylov Methods

- Using PETSc linear algebra, just add:
 - `KSPSetOperators(KSPksp, MatA, MatM, MatStructure flag)`
 - `KSPSolve(KSPksp, Vecb, Vecx)`
- Can access subobjects
 - `KSPGetPC(KSPksp, PC*pc)`
- Preconditioners must obey PETSc interface
 - Basically just the KSP interface
- Can change solver dynamically from the command line
 - `-ksp_type bicgstab`

Nonlinear Solvers

- Using PETSc linear algebra, just add:
 - `SNESSetFunction(SNESsnes, Vecr, residualFunc, void *ctx)`
 - `SNESSetJacobian(SNESsnes, MatA, MatM, jacFunc, void *ctx)`
 - `SNESsolve(SNESsnes, Vecb, Vecx)`
- Can access subobjects
 - `SNESGetKSP(SNESsnes, KSP*ksp)`
- Can customize subobjects from the cmd line
 - Set the subdomain preconditioner to ILU with `-sub_pc_type ilu`

Basic Solver Usage

Use `SNESSetFromOptions()` so that everything is set dynamically

- Set the type
 - Use `-snes_type` (or take the default)
- Set the preconditioner
 - Use `-npc_snes_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

3rd Party Solvers in PETSc

Complete table of solvers

1 Sequential LU

- ILUDT (SPARSEKIT2, Yousef Saad, U of MN)
- EUCLID & PILUT (Hypre, David Hysom, LLNL)
- ESSL (IBM)
- SuperLU (Jim Demmel and Sherry Li, LBNL)
- Matlab
- UMFPACK (Tim Davis, U. of Florida)
- LUSOL (MINOS, Michael Saunders, Stanford)

2 Parallel LU

- MUMPS (Patrick Amestoy, IRIT)
- SPOOLES (Cleve Ashcroft, Boeing)
- SuperLU_Dist (Jim Demmel and Sherry Li, LBNL)

3 Parallel Cholesky

- DSCPACK (Padma Raghavan, Penn. State)
- MUMPS (Patrick Amestoy, Toulouse)
- CHOLMOD (Tim Davis, Florida)

4 XYTLlib - parallel direct solver (Paul Fischer and Henry Tufo, ANL)

3rd Party Preconditioners in PETSc

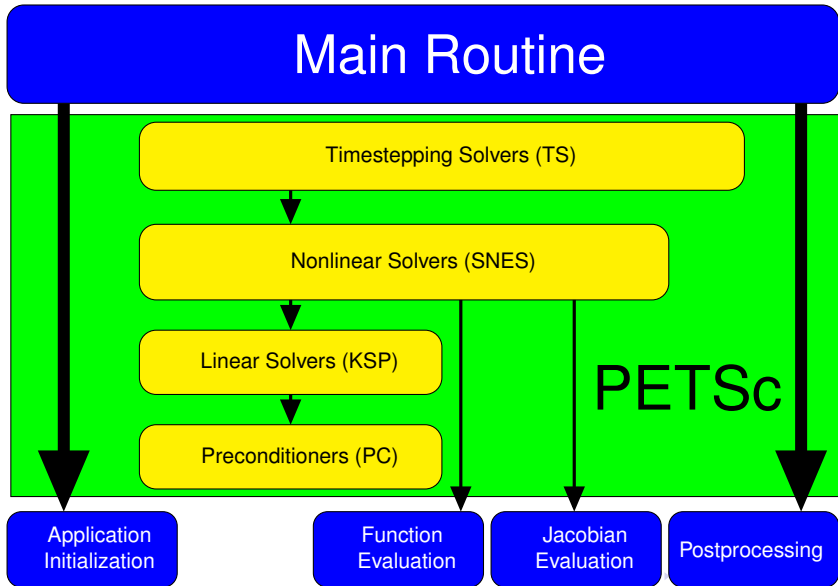
Complete table of solvers

- 1 Parallel ICC
 - BlockSolve95 (Mark Jones and Paul Plassman, ANL)
- 2 Parallel ILU
 - PaStiX (Faverge Mathieu, INRIA)
- 3 Parallel Sparse Approximate Inverse
 - Parasails (Hypre, Edmund Chow, LLNL)
 - SPAI 3.0 (Marcus Grote and Barnard, NYU)
- 4 Sequential Algebraic Multigrid
 - RAMG (John Ruge and Klaus Steuben, GMD)
 - SAMG (Klaus Steuben, GMD)
- 5 Parallel Algebraic Multigrid
 - Prometheus (Mark Adams, PPPL)
 - BoomerAMG (Hypre, LLNL)
 - ML (Trilinos, Ray Tuminaro and Jonathan Hu, SNL)

Outline

- 1 Introduction
- 2 Version Control
- 3 Vector Algebra
- 4 Matrix Algebra
- 5 Algebraic Solvers
- 6 SNES**
- 7 DA

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

Topology Abstractions

- DMDA
 - Abstracts Cartesian grids in any dimension
 - Supports stencils, communication, reordering
 - Nice for simple finite differences
- DMMesh
 - Abstracts general topology in any dimension
 - Also supports partitioning, distribution, and global orders
 - Allows arbitrary element shapes and discretizations

Assembly Abstractions

- `DM`
 - Abstracts the logic of multilevel (multiphysics) methods
 - Manages allocation and assembly of local and global structures
 - Interfaces to `PCMG` solver

- `PetscSection`
 - Abstracts functions over a topology
 - Manages allocation and assembly of local and global structures
 - Will merge with `DM` somehow

SNES Function

User provided function calculates the nonlinear residual:

```
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 calculates the Jacobian:

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

`x`: The current solution

`J`: The Jacobian

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

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

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

Alternatively, you can use

- matrix-free finite difference approximation, `-snes_mf`
- finite difference approximation with coloring, `-snes_fd`

SNES Variants

- Picard iteration
- Line search/Trust region strategies
- Quasi-Newton
- Nonlinear CG/GMRES
- Nonlinear GS/ASM
- Nonlinear Multigrid (FAS)
- Variational inequality approaches

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 (default)
 - Coloring is created by `MatFDColoringCreate()`
 - Computed by `SNESDefaultComputeJacobianColor()`

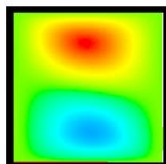
Can also use Matrix-free Newton-Krylov via 1st-order FD

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

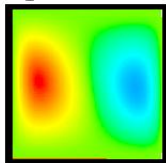
SNES Example

Driven Cavity

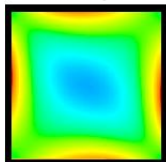
Solution Components



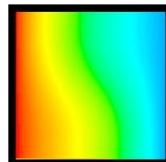
velocity: u



velocity: v



vorticity:



temperature: T

- Velocity-vorticity formulation
- Flow driven by lid and/or buoyancy
- Logically regular grid
 - Parallelized with DM2D
- Finite difference discretization
- Authored by David Keyes

[\\$PETSC_DIR/src/snes/examples/tutorials/ex19.c](#)

Driven Cavity Application Context

```
typedef struct {  
    /*———— basic application data ————*/  
    PetscReal lid_velocity;  
    PetscReal prandtl;  
    PetscReal grashof;  
    PetscBool draw_contours;  
} AppCtx;
```

[\\$PETSC_DIR/src/snes/examples/tutorials/ex19.c](#)

Driven Cavity Residual Evaluation

```

Residual(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;
}

```

[\\$PETSC_DIR/src/snes/examples/tutorials/ex19.c](#)

Better Driven Cavity Residual Evaluation

```

ResLocal(DMDALocalInfo *info ,
        PetscScalar **x, PetscScalar **f, void *ctx)
{
    for(j = info->ys; j < info->ys+info->ym; ++j) {
        for(i = info->xs; i < info->xs+info->xm; ++i) {
            u      = x[j][i];
            uxx = (2.0*u - x[j][i-1] - x[j][i+1])*hydhx;
            uyy = (2.0*u - x[j-1][i] - x[j+1][i])*hxdhy;
            f[j][i].u = uxx + uyy - .5*(x[j+1][i].omega-x[j-1][i].omega)*hx;
            f[j][i].v = uxx + uyy + .5*(x[j][i+1].omega-x[j][i-1].omega)*hy;
            f[j][i].omega = uxx + uyy +
                (vxp*(u - x[j][i-1].omega) + vxm*(x[j][i+1].omega - u))*hy +
                (vyp*(u - x[j-1][i].omega) + vym*(x[j+1][i].omega - u))*hx -
                0.5*grashof*(x[j][i+1].temp - x[j][i-1].temp)*hy;
            f[j][i].temp = uxx + uyy + prandtl*
                ((vxp*(u - x[j][i-1].temp) + vxm*(x[j][i+1].temp - u))*hy +
                 (vyp*(u - x[j-1][i].temp) + vym*(x[j+1][i].temp - u))*hx);
        }}

```

[\\$PETSC_DIR/src/snes/examples/tutorials/ex19.c](#)

Outline

- 1 Introduction
- 2 Version Control
- 3 Vector Algebra
- 4 Matrix Algebra
- 5 Algebraic Solvers
- 6 SNES
- 7 DA**

What is a DMDA?

DMDA is a topology interface on structured grids

- Handles parallel data layout
- Handles local and global indices
 - `DMDAGetGlobalIndices()` and `DMDAGetAO()`
- Provides local and global vectors
 - `DMGetGlobalVector()` and `DMGetLocalVector()`
- Handles ghost values coherence
 - `DMGlobalToLocalBegin/End()` and `DMLocalToGlobalBegin/End()`

Residual Evaluation

The **DM** interface is based upon *local* callback functions

- FormFunctionLocal()
- FormJacobianLocal()

Callbacks are registered using

- SNESSetDM(), TSSetDM()
- DMSNESSetFunctionLocal(), DMTSSetJacobianLocal()

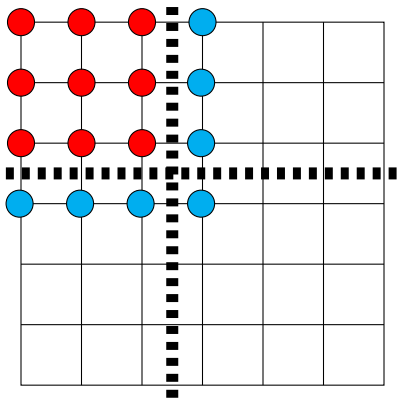
When PETSc needs to evaluate the nonlinear residual **F(x)**,

- Each process evaluates the local residual
- PETSc assembles the global residual automatically
 - Uses DMLocalToGlobal() method

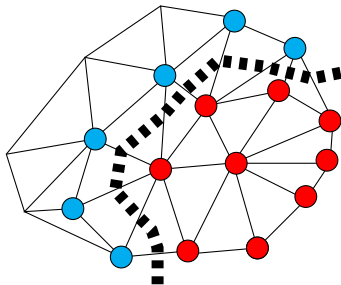
Ghost Values

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



- Local Node
- Ghost Node



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

DMDA Local Function

User provided function calculates the nonlinear residual (in 2D)

```
(* If)(DMDALocalInfo *info, PetscScalar**x, PetscScalar**r, void *ctx)
```

`info`: All layout and numbering information

`x`: The current solution (a multidimensional array)

`r`: The residual

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

The local DMDA function is activated by calling

```
DMDASNESSetFunctionLocal(dm, INSERT_VALUES, lfunc, &ctx)
```

Bratu Residual Evaluation

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

```

ResLocal(DMDALocalInfo *info, PetscScalar **x, PetscScalar **f, void *ctx)
for(j = info->ys; j < info->ys+info->ym; ++j) {
  for(i = info->xs; i < info->xs+info->xm; ++i) {
    u = x[j][i];
    if (i==0 || j==0 || i == M || j == N) {
      f[j][i] = 2.0*(hydhx+hxddy)*u; continue;
    }
    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])*hxddy;
    f[j][i] = u_xx + u_yy - hx*hy*lambda*exp(u);
  }}

```

[\\$PETSC_DIR/src/snes/examples/tutorials/ex5.c](#)

DMDA Local Jacobian

User provided function calculates the Jacobian (in 2D)

```
(* ljac )(DMDALocalInfo *info, PetscScalar**x, Mat J, void *ctx)
```

`info`: All layout and numbering information

`x`: The current solution

`J`: The Jacobian

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

The local DMDA function is activated by calling

```
DMDASNESSetJacobianLocal(dm, ljac, &ctx)
```

Bratu Jacobian Evaluation

```

JacLocal(DMDALocalInfo *info, PetscScalar **x, Mat jac, void *ctx) {
for(j = info->ys; j < info->ys + info->ym; j++) {
    for(i = info->xs; i < info->xs + info->xm; i++) {
        row.j = j; row.i = i;
        if (i == 0 || j == 0 || i == mx-1 || j == my-1) {
            v[0] = 1.0;
            MatSetValuesStencil(jac, 1, &row, 1, &row, v, INSERT_VALUES);
        } else {
            v[0] = -(hx/hy); col[0].j = j-1; col[0].i = i;
            v[1] = -(hy/hx); col[1].j = j; col[1].i = i-1;
            v[2] = 2.0*(hy/hx+hx/hy)
                - hx*hy*lambda*PetscExpScalar(x[j][i]);
            v[3] = -(hy/hx); col[3].j = j; col[3].i = i+1;
            v[4] = -(hx/hy); col[4].j = j+1; col[4].i = i;
            MatSetValuesStencil(jac, 1, &row, 5, col, v, INSERT_VALUES);
        }
    }
}
}
}
}

```

[\\$PETSC_DIR/src/snes/examples/tutorials/ex5.c](#)

DMDA Vectors

- The **DMDA** object contains only layout (topology) information
 - All field data is contained in PETSc **Vecs**
- Global vectors are parallel
 - Each process stores a unique local portion
 - `DMCreateGlobalVector(DM da, Vec *gvec)`
- Local vectors are sequential (and usually temporary)
 - Each process stores its local portion plus ghost values
 - `DMCreateLocalVector(DM da, Vec *lvec)`
 - includes ghost and boundary values!

Updating Ghosts

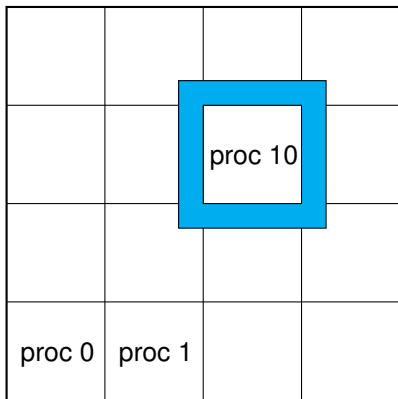
Two-step process enables overlapping computation and communication

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

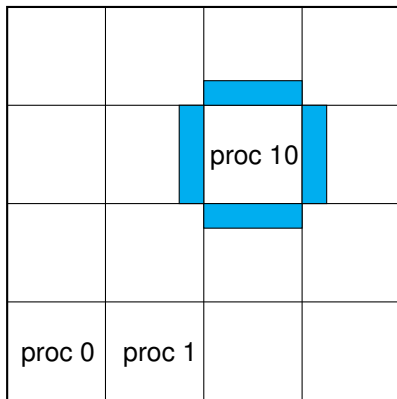
The process can be reversed with `DALocalToGlobalBegin/End()`.

DMDA Stencils

Both the **box** stencil and **star** stencil are available.



Box Stencil



Star Stencil

Setting Values on Regular Grids

PETSc provides

```
MatSetValuesStencil(Mat A, m, MatStencil idxm[], n, MatStencil idxn[],  
                  PetscScalar values[], InsertMode mode)
```

- Each row or column is actually a **MatStencil**
 - This specifies grid coordinates and a component if necessary
 - Can imagine for unstructured grids, they are *vertices*
- The values are the same logically dense block in row/col

Creating a DMDA

`DMDACreate2d(comm, bdX, bdY, type, M, N, m, n, dof, s, lm[], ln[], DMDA *da)`

`bd`: Specifies boundary behavior

- `DM_BOUNDARY_NONE`, `DM_BOUNDARY_GHOSTED`, or `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/n`: Alternative array of local sizes

- Use `NULL` for the default

Outline

- 1 Introduction
- 2 Version Control
- 3 Vector Algebra
- 4 Matrix Algebra
- 5 Algebraic Solvers
- 6 SNES
- 7 DA

MultiPhysics Paradigm

The **PCFieldSplit** interface

- extracts functions/operators corresponding to each physics
 - **VecScatter** and `MatGetSubMatrix()` for efficiency
- assemble functions/operators over all physics
 - Generalizes `LocalToGlobal()` mapping
- is composable with **ANY** PETSc solver and preconditioner
 - This can be done recursively

MultiPhysics Paradigm

The **PCFieldSplit** interface

- extracts functions/operators corresponding to each physics
 - **VecScatter** and `MatGetSubMatrix()` for efficiency
- assemble functions/operators over all physics
 - Generalizes `LocalToGlobal()` mapping
- is composable with **ANY** PETSc solver and preconditioner
 - This can be done recursively

FieldSplit provides the **building blocks** for multiphysics preconditioning.

MultiPhysics Paradigm

The **PCFieldSplit** interface

- extracts functions/operators corresponding to each physics
 - **VecScatter** and `MatGetSubMatrix()` for efficiency
- assemble functions/operators over all physics
 - Generalizes `LocalToGlobal()` mapping
- is composable with **ANY** PETSc solver and preconditioner
 - This can be done recursively

Notice that this works in exactly the same manner as

- multiple resolutions (MG, FMM, Wavelets)
- multiple domains (Domain Decomposition)
- multiple dimensions (ADI)

Preconditioning

Several varieties of preconditioners can be supported:

- Block Jacobi or Block Gauss-Siedel
- Schur complement
- Block ILU (approximate coupling and Schur complement)
- Dave May's implementation of Elman-Wathen type PCs

which only require actions of individual operator blocks

Notice also that we may have any combination of

- “canned” PCs (ILU, AMG)
- PCs needing special information (MG, FMM)
- custom PCs (physics-based preconditioning, Born approximation)

since we have access to an algebraic interface

Outline

- 1 Introduction
- 2 Version Control
- 3 Vector Algebra
- 4 Matrix Algebra
- 5 Algebraic Solvers
- 6 SNES
- 7 DA

Thrust

Thrust is a CUDA library of parallel algorithms

- Interface similar to C++ Standard Template Library
- Containers (`vector`) on both host and device
- Algorithms: `sort`, `reduce`, `scan`
- Freely available, part of PETSc configure (`-with-thrust-dir`)
- Included as part of CUDA 4.0 installation

Cusp

Cusp is a CUDA library for sparse linear algebra and graph computations

- Builds on data structures in Thrust
- Provides sparse matrices in several formats (CSR, Hybrid)
- Includes some preliminary preconditioners (Jacobi, SA-AMG)
- Freely available, part of PETSc configure (`-with-cusp-dir`)

Strategy: Define a new **Vec** implementation

- Uses **Thrust** for data storage and operations on GPU
- Supports full PETSc **Vec** interface
- Inherits PETSc scalar type
- Can be activated at runtime, `-vec_type cuda`
- PETSc provides memory coherence mechanism

Memory Coherence

PETSc Objects now hold a coherence flag

PETSC_CUDA_UNALLOCATED	No allocation on the GPU
PETSC_CUDA_GPU	Values on GPU are current
PETSC_CUDA_CPU	Values on CPU are current
PETSC_CUDA_BOTH	Values on both are current

Table: Flags used to indicate the memory state of a PETSc CUDA **Vec** object.

Also define new **Mat** implementations

- Uses **Cusp** for data storage and operations on GPU
- Supports full PETSc **Mat** interface, some ops on CPU
- Can be activated at runtime, `-mat_type aijcuda`
- Notice that parallel matvec necessitates off-GPU data transfer

Solvers come for Free

Preliminary Implementation of PETSc Using GPU,
Minden, Smith, Knepley, 2010

- All linear algebra types work with solvers
- Entire solve can take place on the GPU
 - Only communicate scalars back to CPU
- GPU communication cost could be amortized over several solves
- Preconditioners are a problem
 - Cusp has a promising AMG

Installation

PETSc only needs

```
# Turn on CUDA
--with-cuda
# Specify the CUDA compiler
--with-cudac='nvcc -m64'
# Indicate the location of packages
# --download-* will also work soon
--with-thrust-dir=/PETSc3/multicore/thrust
--with-cusp-dir=/PETSc3/multicore/cusp
# Can also use double precision
--with-precision=single
```

Example

Driven Cavity Velocity-Vorticity with Multigrid

```
ex50 -da_vec_type seqcusp
     -da_mat_type aijcusp -mat_no_inode # Setup types
     -da_grid_x 100 -da_grid_y 100     # Set grid size
     -pc_type none -pc_mg_levels 1     # Setup solver
     -preload off -cuda_synchronize   # Setup run
     -log_summary
```

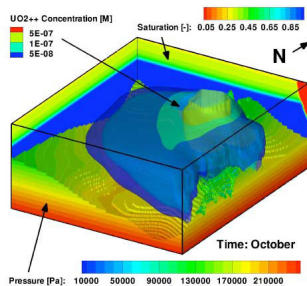

Example

PFLOTRAN

Flow Solver

32 × 32 × 32 grid

Routine	Time (s)	MFlops	MFlops/s
CPU			
KSPSolve	8.3167	4370	526
MatMult	1.5031	769	512
GPU			
KSPSolve	1.6382	4500	2745
MatMult	0.3554	830	2337



P. Lichtner, G. Hammond,
R. Mills, B. Phillip