The
Portable **E**xtensible **T**oolkit for **S**cientific **C**omputing

Matthew Knepley

Mathematics and Computer Science Division
Argonne National Laboratory

Computation Institute
University of Chicago

Scientific Software Days
TACC, Austin, TX May 21-22, 2009

Outline

Unit Objectives

- Introduce PETSc
- Download, Configure, Build, and Run an Example
- Empower students to learn more about PETSc

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

How We Can Help at the Tutorial

- Point out relevant documentation

- Answer email at petsc-maint@mcs.anl.gov

How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions

- Answer email at petsc-maint@mcs.anl.gov

How We Can Help at the Tutorial

- Point out relevant documentation
- Quickly answer questions
- Help install

- 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

What is PETSc?

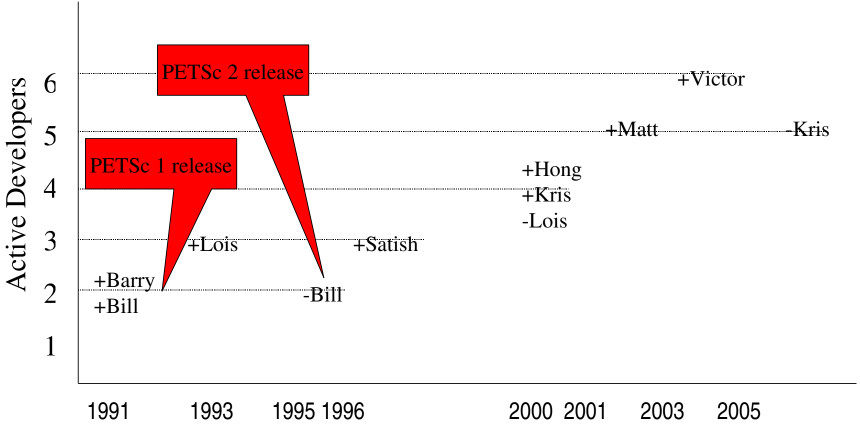
A freely available and supported research code

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

What is PETSc?

- Portable to any parallel system supporting MPI, including:
    - Tightly coupled systems
        - Cray T3E, SGI Origin, IBM SP, HP 9000, Sub Enterprise
    - Loosely coupled systems, such as networks of workstations
        - Compaq,HP, IBM, SGI, Sun, PCs running Linux or Windows
- PETSc History
    - Begun September 1991
    - Over 20,000 downloads since 1995 (version 2)
    - Currently 400 per month
- PETSc Funding and Support
    - Department of Energy
        - SciDAC, MICS Program, INL Reactor Program
    - National Science Foundation
        - CIG, CISE, Multidisciplinary Challenge Program

Timeline


Active Developers
6
5
4
3
2
1
PETSc 2 release
PETSc 1 release
+Victor
+Matt
-Kris
+Hong
+Kris
-Lois
+Lois
+Satish
+Barry
+Bill
-Bill
1991
1993
1995
1996
2000
2001
2003
2005

What Can We Handle?

- PETSc has run implicit problems with over 1 billion unknowns
  - PFLOTRAN for flow in porous media

What Can We Handle?

- PETSc has run implicit problems with over 1 billion unknowns
  - PFLOTRAN for flow in porous media
- PETSc has run on over 130,000 cores efficiently
  - UNIC on the IBM BG/P Intrepid at ANL
  - PFLOTRAN on the Cray XT5 Jaguar at ORNL

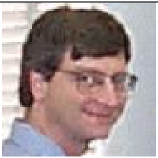
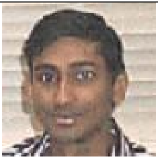
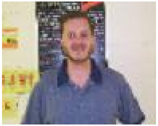
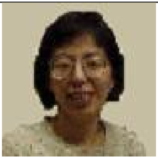
What Can We Handle?

- PETSc has run implicit problems with over 1 billion unknowns
  - PFLOTRAN for flow in porous media
- PETSc has run on over 130,000 cores efficiently
  - UNIC on the IBM BG/P Intrepid at ANL
  - PFLOTRAN on the Cray XT5 Jaguar at ORNL
- PETSc applications have run at 3 Teraflops
  - LANL PFLOTRAN code

Who Uses PETSc?

- Computational Scientists
  - PyLith (TECTON), Underworld, Columbia group, PFLOTRAN
- Algorithm Developers
  - Iterative methods and Preconditioning researchers
- Package Developers
  - SLEPc, TAO, PETSc-FEM, MagPar, StGermain, DealII

The PETSc Team

 Bill Gropp	 Barry Smith	 Satish Balay
 Dinesh Kaushik	 Kris Buschelman	 Matt Knepley
 Hong Zhang	 Victor Eijkhout	 Lois McInnes

Downloading PETSc

- The latest tarball is on the PETSc site
  - ftp://ftp.mcs.anl.gov/pub/petsc/petsc.tar.gz
  - We no longer distribute patches (everything is in the distribution)
- There is a Debian package
- There is a FreeBSD Port
- There is a Mercurial development repository

Cloning PETSc

- The full development repository is open to the public
  - http://petsc.cs.iit.edu/petsc/petsc-dev
  - http://petsc.cs.iit.edu/petsc/BuildSystem
- 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
- We also make release repositories available
  - http://petsc.cs.iit.edu/petsc/petsc-release-3.0.0

Cloning PETSc

- Just clone development repository
  - `hg clone http://petsc.cs.iit.edu/petsc/petsc-dev petsc-dev`
  - `hg clone -rRelease-3.0.0 petsc-dev petsc-3.0.0`

or

- Unpack the tarball
  - `tar xzf petsc.tar.gz`

Exercise 1

Download and Unpack PETSc!

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, LAM
  - 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.

Building PETSc

- Uses recursive make starting in `cd $PETSC_DIR`
  - `make`
  - `make install` if you configured with `--prefix`
  - Check build when done with `make test`
- Complete log for each build is in logfile
  - `./$PETSC_ARCH/conf/make.log`
  - ALWAYS send this with bug reports
- Can build multiple configurations
  - `PETSC_ARCH=linux-fast make`
  - Libraries are in `$PETSC_DIR/$PETSC_ARCH/lib/`
- Can also build a subtree
  - `cd src/snes; make`
  - `cd src/snes; make ACTION=libfast tree`

Exercise 3

Build your configured PETSc.

Exercise 4

Reconfigure PETSc to use ParMetis.

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

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 Message Passing Interface 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

MPI Concepts

- Communicator
  - A context (or scope) for parallel communication (“Who can I talk to”)
  - There are two defaults:
    - yourself (PETSC_COMM_SELF),
    - and everyone launched (PETSC_COMM_WORLD)
  - Can create new communicators by splitting existing ones
  - Every PETSc object has a communicator
  - Set PETSC_COMM_WORLD to put all of PETSc in a subcomm
- Point-to-point communication
  - Happens between two processes (like in `MatMult()`)
- Reduction or scan operations
  - Happens among all processes (like in `VecDot()`)

Alternative Memory Models

- Single process (address space) model
  - OpenMP and threads in general
  - Fortran 90/95 and compiler-discovered parallelism
  - System manages memory and (usually) thread scheduling
  - Named variables refer to the same storage
- Single name space model
  - HPF, UPC
  - Global Arrays
  - Titanium
  - Variables refer to the coherent values (distribution is automatic)
- Distributed memory (shared nothing)
  - Message passing
  - Names variables in different processes are unrelated

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`, `-vec_view_matlab`, `-vec_view_socket`
- Sometimes provides extra information
  - `-mat_view_info`, `-mat_view_info_detailed`

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 come 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.

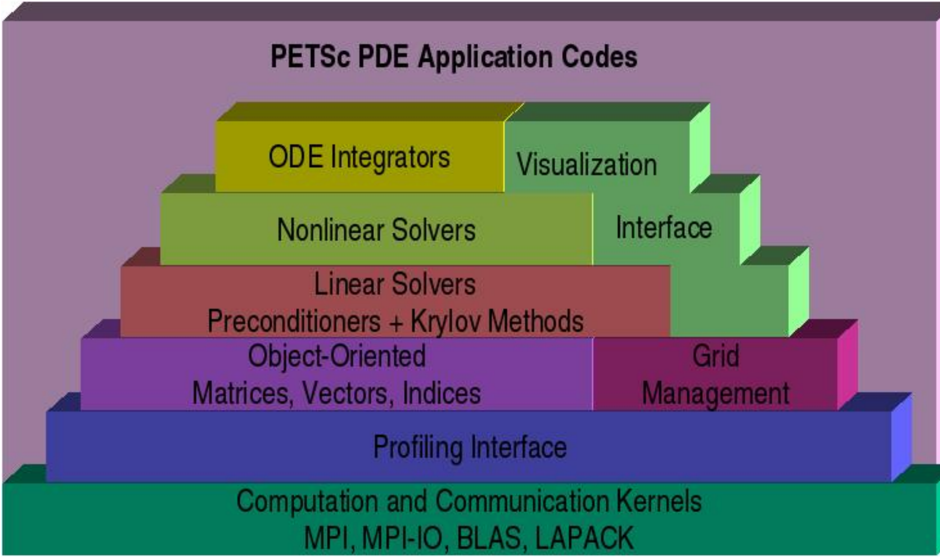
1. Create a new directory
   - `mkdir -p /home/knepley/proj/newsim/src`
2. Copy the source
   - `cp ex5.c /home/knepley/proj/newsim/src`
   - Add `myStuff.c` and `myStuff2.F`
3. Create a PETSc makefile
   - `ex5: ex5.o myStuff.o myStuff2.o`
   - `  ${CLINKER} -o $@ $^ ${PETSC_SNES_LIB}`
   - `include ${PETSC_DIR}/bmake/common/base`

Getting More Help

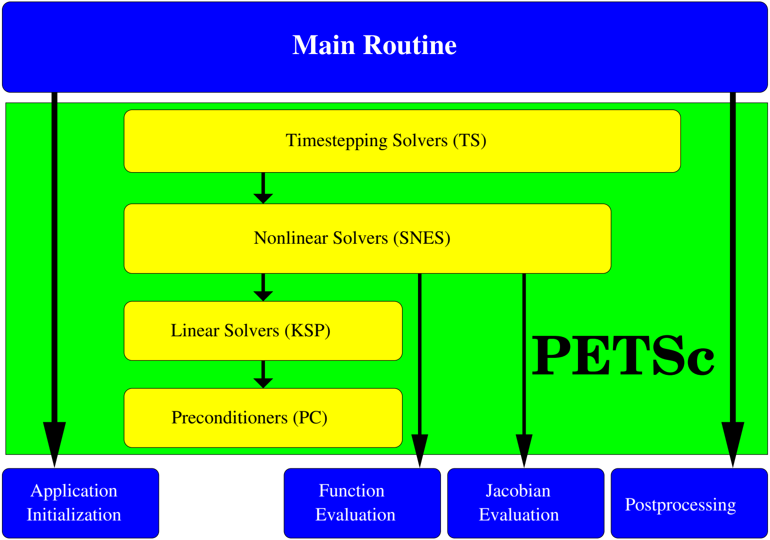
- http://www.mcs.anl.gov/petsc
- Hyperlinked documentation
    - Manual
    - Manual pages for evey method
    - HTML of all example code (linked to manual pages)
- FAQ
- Full support at petsc-maint@mcs.anl.gov
- High profile users
    - David Keyes
    - Marc Spiegelman
    - Richard Katz
    - Brad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

Outline

PETSc Structure



Flow Control for a PETSc Application



Levels of Abstraction
In Mathematical Software

- Application-specific interface
  - Programmer manipulates objects associated with the application
- High-level mathematics interface
  - Programmer manipulates mathematical objects
    - Weak forms, boundary conditions, meshes
- Algorithmic and discrete mathematics interface
  - Programmer manipulates mathematical objects
    - Sparse matrices, nonlinear equations
  - Programmer manipulates algorithmic objects
    - Solvers
- Low-level computational kernels
  - BLAS-type operations, FFT

Object-Oriented Design

- Design based on operations you perform,
  - rather than the data in the object
- Example: A vector is
  - **not** a 1d array of numbers
  - an object allowing addition and scalar multiplication
- The efficient use of the computer is an added difficulty
  - which often leads to code generation

The PETSc Programming Model

- Goals
  - Portable, runs everywhere
  - High performance
  - Scalable parallelism
- Approach
  - Distributed memory (“shared-nothing”)
  - No special compiler
  - Access to data on remote machines through MPI
  - Hide within objects the details of the communication
  - User orchestrates communication at a higher abstract level

Symmetry Principle

Interfaces to mutable data must be symmetric.

- Creation and query interfaces are paired
  - "No get without a set"
- Fairness
  - "If you can do it, your users will want to do it"
- Openness
  - "If you can do it, your users will want to undo it"

Empiricism Principle

Interfaces must allow easy testing and comparison.

- Swapping different implementations
  - “You will not be smart enough to pick the solver”
- Commonly violated in FE code
  - Elements are hard coded
- Also avoid assuming structure outside of the interface
  - Making continuous fields have discrete structure
  - Temptation to put metadata in a different places

Experimentation is Essential!

Proof is not currently enough to examine solvers

- N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen, How fast are nonsymmetric matrix iterations?, SIAM J. Matrix Anal. Appl., **13**, pp.778–795, 1992.
- Anne Greenbaum, Vlastimil Ptak, and Zdenek Strakos, Any Nonincreasing Convergence Curve is Possible for GMRES, SIAM J. Matrix Anal. Appl., **17** (3), pp.465–469, 1996.

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

What is not in PETSc?

- ~~Unstructured mesh generation and manipulation~~
  - In 3.0, we have `Mesh` objects
- Discretizations
  - DealII
  - In 3.0, we have an interface to FIAT
- Higher level representations of PDEs
  - FEniCS (FFC/Syfi) and Sundance
- Load balancing
  - Interface to Zoltan
- Sophisticated visualization capabilities
  - Interface to MayaVi2 through VTK
- Eigenvalues
  - SLEPc and SIP
- Optimization and sensitivity
  - TAO and Veltisto

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 lin
SetUp()	preform other initialization
View()	view the object
Destroy()	cleanup object allocation

Also, all objects support the -help option.

Correctness Debugging

- Automatic generation of tracebacks
- Detecting memory corruption and leaks
- Optional user-defined error handlers

Interacting with the Debugger

- 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 khan.mcs.anl.gov:0.0`

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
  - Need `-trace-children=yes` when running under MPI

Exercise 7

Use the debugger to find a SEGV
Locate a memory overwrite using CHKMEMQ.

- Get the example
  - `hg clone -r1 http://petsc.cs.iit.edu/petsc/TutorialExercises`
- Build the example `make`
- Run it and watch the fireworks
  - `mpiexec -n 2 ./bin/ex5 -use_coords`
- Run it under the debugger and correct the error
  - `mpiexec -n 2 ./bin/ex5 -use_coords -start_in_debugger -display :0.0`
  - `hg update -r2`
- Build it and run again smoothly

Performance Debugging

- PETSc has integrated profiling
  - Option `-log_summary` prints a report on `PetscFinalize()`
- PETSc allows user-defined events
  - Events report time, calls, flops, communication, etc.
  - Memory usage is tracked by object
- Profiling is separated into stages
  - Event statistics are aggregated by stage

Using Stages and Events

- Use `PetscLogStageRegister()` to create a new stage
  - Stages are identifier by an integer handle
- Use `PetscLogStagePush/Pop()` to manage stages
  - Stages may be nested, but will not aggregate in a nested fashion
- Use `PetscLogEventRegister()` to create a new stage
  - Events also have an associated class
- Use `PetscLogEventBegin/End()` to manage events
  - Events may also be nested and will aggregate in a nested fashion
  - Can use `PetscLogFlops()` to log user flops

Adding A Logging Stage

```
int stageNum;

PetscLogStageRegister(&stageNum, "name");
PetscLogStagePush(stageNum);

Code to Monitor

PetscLogStagePop();
```

Adding A Logging Event

```
static int USER_EVENT;

PetscLogEventRegister(&USER_EVENT, "name", CLS_COOKIE)

PetscLogEventBegin(USER_EVENT,0,0,0,0);

Code to Monitor

PetscLogFlops(user_event_flops);
PetscLogEventEnd(USER_EVENT,0,0,0,0);
```

Adding A Logging Class

```
static int CLASS_COOKIE;

PetscLogClassRegister(&CLASS_COOKIE,"name");
```

- Cookie identifies a class uniquely
- Must initialize before creating any objects of this type

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 rourines for all datatypes
  - `MatSeqAIJSetPreallocation()`
  - `MatMPIAIJSetPreallocation()`
  - Only the relevant data will be used

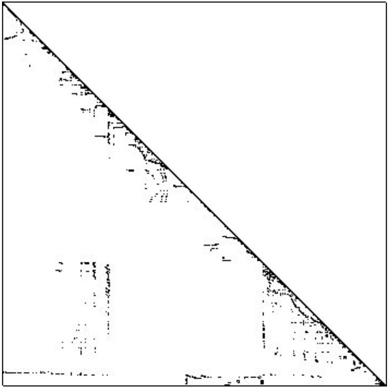
Matrix Memory Preallocation

Sequential Sparse Matrices

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

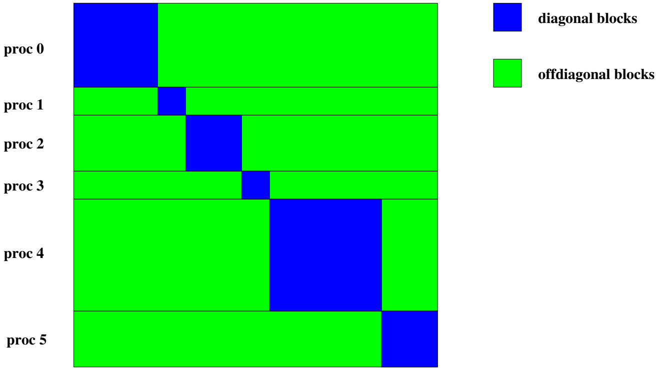
nz: expected number of nonzeros in any row

nnz(i): expected number of nonzeros in row i



Matrix Memory Preallocation
ParallelSparseMatrix

- 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

Matrix Memory Preallocation

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

nnz(i): expected number of nonzeros in row i in the diagonal block

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

nnz(i): expected number of nonzeros in row i in the offdiagonal portion

Matrix Memory Preallocation
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!
```

Exercise 8

Return to Execise 7 and add more profiling.

- Update to the next revision
  - `hg update -r3`
- Build, run, and look at the profiling report
  - `make ex5`
  - `./bin/ex5 -use_coords -log_summary`
- Add a new stage for setup
- Add a new event for `FormInitialGuess()` and log the flops
- Build it again and look at the profiling report

STREAM Benchmark

Simple benchmark program measuring sustainable memory bandwidth

- Protoypical operation is Triad (WAXPY): $\mathbf{w} = \mathbf{y} + \alpha\mathbf{x}$
- Measures the memory bandwidth bottleneck (much below peak)
- Datasets outstrip cache

Machine	Peak (MF/s)	Triad (MB/s)	MF/MW	Eq. MF/s
Matt's Laptop	1700	1122.4	12.1	93.5 (5.5%)
Intel Core2 Quad	38400	5312.0	57.8	442.7 (1.2%)

Table: Bandwidth limited machine performance

http://www.cs.virginia.edu/stream

Analysis of Sparse Matvec (SpMV)

Assumptions

- No cache misses
- No waits on memory references

Notation

m Number of matrix rows

nz Number of nonzero matrix elements

V Number of vectors to multiply

We can look at bandwidth needed for peak performance

$$\left(8 + \frac{2}{V}\right)\frac{m}{nz} + \frac{6}{V} \text{ byte/flop} \tag{1}$$

or achieveable performance given a bandwith BW

$$\frac{Vnz}{(8V + 2)m + 6nz} BW \text{ Mflop/s} \tag{2}$$

Towards Realistic Performance Bounds for Implicit CFD Codes, Gropp, Kaushik, Keyes, and Smith.

Improving Serial Performance

For a single matvec with 3D FD Poisson, Matt's laptop can achieve at most

$$\frac{1}{(8+2)\frac{1}{7}+6} \text{ bytes/flop}(1122.4 \text{ MB/s}) = 151 \text{ MFlops/s}, \tag{3}$$

which is a dismal 8.8% of peak.

Can improve performance by

- Blocking
- Multiple vectors

but operation issue limitations take over.

Improving Serial Performance

For a single matvec with 3D FD Poisson, Matt's laptop can achieve at most

$$\frac{1}{(8+2)\frac{1}{7}+6} \text{ bytes/flop}(1122.4 \text{ MB/s}) = 151 \text{ MFlops/s}, \tag{3}$$

which is a dismal 8.8% of peak.

Better approaches:

- Unassembled operator application (Spectral elements)
  - N data, N^2 computation
- Nonlinear evaluation (Picard, FAS, Exact Polynomial Solvers)
  - N data, N^k computation

Performance Tradeoffs

We must balance storage, bandwidth, and cycles

- Assembled Operator Action
  - Trades cycles and storage for bandwidth in application
- Unassembled Operator Action
  - Trades bandwidth and storage for cycles in application
  - For high orders, storage is impossible
  - Can make use of FErari decomposition to save calculation
  - Could storage element matrices to save cycles
- Partial assembly gives even finer control over tradeoffs
  - Also allows introduction of parallel costs (load balance, . . .)

Importance of Computational Modeling

Without a model,
performance measurements are meaningless!

Before a code is written, we should have a model of

- computation
- memory usage
- communication
- bandwidth
- achievable concurrency

This allows us to

- verify the implementation
- predict scaling behavior

Outline

1. Getting Started with PETSc
2. Common PETSc Usage
3. PETSc Integration
   - Initial Operations
   - Vector Algebra
   - Matrix Algebra
   - Algebraic Solvers
   - More Abstractions
4. Advanced PETSc
5. Future Plans

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

PETSc Integration

PETSc is a set a library interfaces

- We do not seize `main()`
- We do not control output
- We propagate errors from underlying packages
- We present the same interfaces in:
  - C
  - C++
  - F77
  - F90
  - Python

See Gropp in SIAM, OO Methods for Interop SciEng, '99

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
- Call `PetscFinalize()`
  - Calculates logging summary
  - Shutdown and release resources
- Checks compile and link

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

Command Line Processing

- Check for an option
  - `PetscOptionsHasName()`
- Retrieve a value
  - `PetscOptionsGetInt()`, `PetscOptionsGetIntArray()`
- Set a value
  - `PetscOptionsSetValue()`
- Check for unused options
  - `-options_left`
- Clear, alias, reject, etc.
- Modern form uses
  - `PetscOptionsBegin()`, `PetscOptionsEnd()`
  - `PetscOptionsInt()`,`PetscOptionsReal()`
  - Integrates with `-help`

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

Vector Algebra

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)

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);
```

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 inexpensive and do not involve a copy

VecGetArray in C

```
Vec v;
PetscScalar *array;
PetscInt n, i;
PetscErrorCode ierr;

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"
#include "finclude/petscvec.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.h"
#include "finclude/petscvec.h"
#include "finclude/petscvec.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)
```

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

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
- `MatSetValues(Mat,...)`
  - **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(Mat m, 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

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)

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(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`

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

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)
4. XYTlib - 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
   - BlockSolve95 (Mark Jones and Paul Plassman, ANL)
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)

Higher Level Abstractions

The PETSc `DA` class is a topology and discretization interface.

- Structured grid interface
  - Fixed simple topology
- Supports stencils, communication, reordering
  - Limited idea of operators
- Nice for simple finite differences

The PETSc `Mesh` class is a topology interface.

- Unstructured grid interface
  - Arbitrary topology and element shape
- Supports partitioning, distribution, and global orders

Higher Level Abstractions

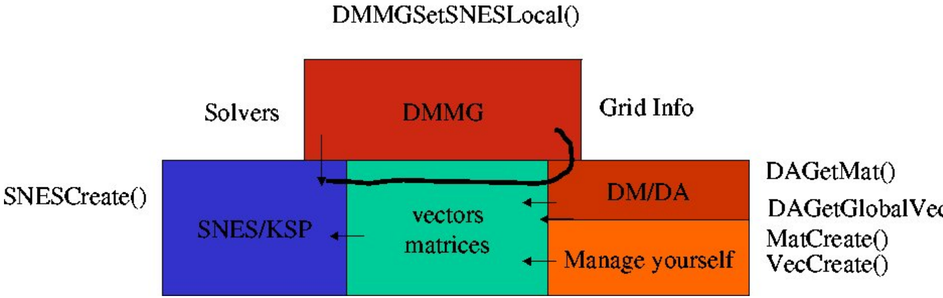
The PETSc `DM` class is a hierarchy interface.

- Supports multigrid
  - `DMMG` combines it with the `MG` preconditioner
- Abstracts the logic of multilevel methods

The PETSc `Section` class is a function interface.

- Functions over unstructured grids
  - Arbitrary layout of degrees of freedom
- Support distribution and assembly

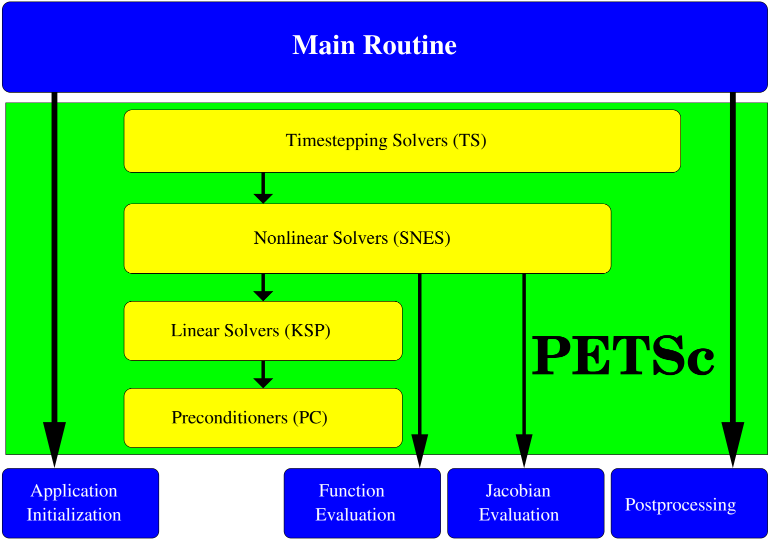
3 Ways To Use PETSc



- User manages all topology (just use Vec and Mat)
- PETSc manages single topology (use DA)
- PETSc manages a hierarchy (use DM)

Outline

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

- DA
  - Abstracts Cartesian grids in any dimension
  - Supports stencils, communication, reordering
  - Nice for simple finite differences
- Mesh
  - Abstracts general topology in any dimension
  - Also supports partitioning, distribution, and global orders
  - Allows aribtrary 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 `DMMG` solver
- `Section`
  - Abstracts functions over a topology
  - Manages allocation and assembly of local and global structures
  - Will merge with `DM` somehow

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
- automatic differentiation (ADIC/ADIFOR)

SNES Variants

- Line search strategies
- Trust region approaches
- Pseudo-transient continuation
- Matrix-free variants

Finite Difference Jacobians

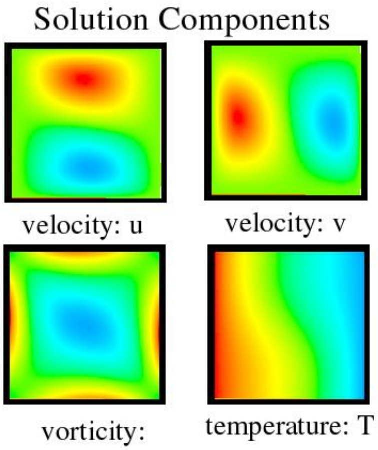
PETSc can compute and explicitly store a Jacobian via 1st-order FD

- 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()`

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

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

SNES Example

Driven Cavity Application Context

```
typedef struct {
  /*--- basic application data ---*/
  double lid_velocity;
  double prandtl, grashof;
  int mx, my;
  int mc;
  PetscTruth draw_contours;
  /*--- parallel data ---*/
  MPI_Comm comm;
  DA da;
  /* Local ghosted solution and residual */
  Vec localX, localF;
} AppCtx;
```

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

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 */
  int 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;
}
```

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

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->xs+info->xm; j++) {
    for(i = info->xs; i < info->ys+info->ym; 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 */
}}}
```

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

What is a DA?

`DA` is a topology interface handling parallel data layout on structured grids

- Handles local and global indices
  - `DAGetGlobalIndices()` and `DAGetAO()`
- Provides local and global vectors
  - `DAGetGlobalVector()` and `DAGetLocalVector()`
- Handles ghost values coherence
  - `DAGetGlobalToLocal()` and `DAGetLocalToGlobal()`

DA Paradigm

The DA interface is based upon local callback functions

- `FormFunctionLocal()`, set by `DASetLocalFunction()`
- `FormJacobianLocal()`, set by `DASetLocalJacobian()`

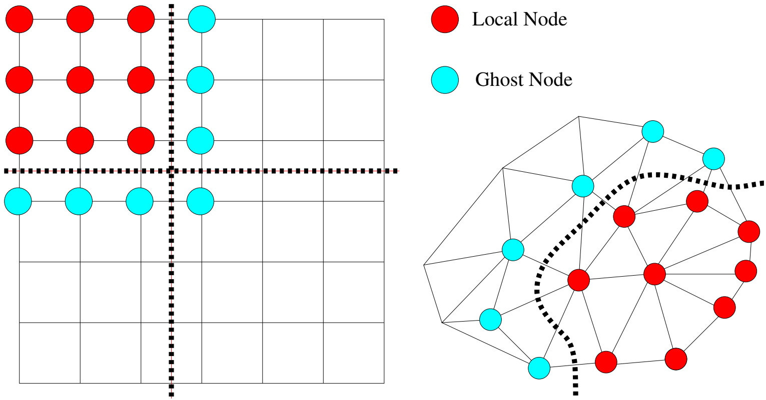
When PETSc needs to evaluate the nonlinear residual $F(x)$,

- Each process evaluates the local residual
- PETSc assembles the global residual automatically

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 Local Function

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

```
PetscErrorCode (*lfunc)(DALocalInfo *info,
PetscScalar **x, PetscScalar **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)
{
  /* Not Shown: Handle boundaries */
  /* Compute over the interior points */
  for(j = info->ys; j < info->xs+info->ym; j++) {
    for(i = info->xs; i < info->ys+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);
    }
  }
}
```

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

DA Local Jacobian

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

```
PetscErrorCode (*lfunc)(DALocalInfo *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 `DASetLocalFunction()`

The local DA function is activated by calling

```
SNESSetJacobian(snes, J, J, SNESDAComputeJacobian,
    ctx)
```

Bratu Jacobian Evaluation

```
BratuJacobianLocal(DALocalInfo *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)
} } } }
```

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

A DA is more than a Mesh

A DA contains topology, geometry, and an implicit Q1 discretization.

It is used as a template to create

- Vectors (functions)
- Matrices (linear operators)

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!

Updating Ghosts

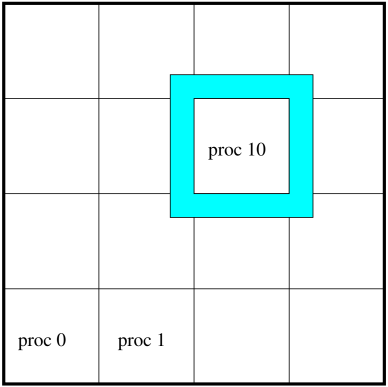
Two-step process enables overlapping computation and communication

- `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

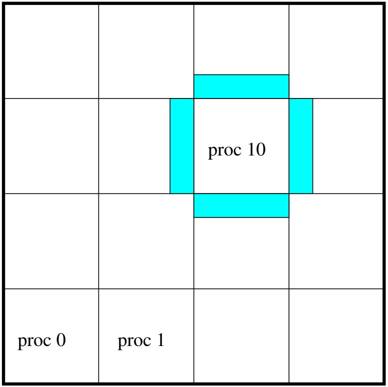
The process can be reversed with `DALocalToGlobal()`.

DA Stencils

Both the box stencil and star stencil are available.



Box Stencil



Star Stencil

Setting Values on Regular Grids

PETSc provides

```
MatSetValuesStencil(Mat A, m, MatStencil idxm[], n,
        MatStencil idxn[], values[], 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 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/n: Alternative array of local sizes
- Use PETSC_NULL for the default

Outline

Things To Check Out

- PCFieldSplit for multiphysics
- DMMG for multilevel solvers
- Sieve for topology automation
- DealII and FEniCS for FEM automation
- PetFMM for particle methods

MultiPhysics Paradigm

The PCFieldSplit interface uses the `VecScatter` objects to

- extract functions/operators corresponding to each physics
- assemble functions/operators over all physics

Notice that this works in exactly the same manner for

- multiple resolutions (MG, 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

DMMG Paradigm

The DMMG interface uses the local DA/Mesh callback functions to

- assemble global functions/operators from local pieces
- assemble functions/operators on coarse grids

DMMG relies upon DM to organize the

- assembly
- coarsening/refinement

while it organizes the control flow for the multilevel solve.

DMMG Integration with SNES

- DMMG supplies global residual and Jacobian to SNES
  - User supplies local version to DMMG
  - The `Rhs_*()` and `Jac_*()` functions in the example
- Allows automatic parallelism
- Allows grid hierarchy
  - Enables multigrid once interpolation/restriction is defined
- Paradigm is developed in unstructured work
  - Notice we have to scatter into contiguous global vectors (initial guess)
- Handle Neumann BC using `DMMGSetNullSpace()`

Structured Meshes

The `DMMG` allows multigrid which some simple options

- `-dmmg_nlevels`, `-dmmg_view`
- `-pc_mg_type`, `-pc_mg_cycle_type`
- `-mg_levels_1_ksp_type`, `-dmmg_levels_1_pc_type`
- `-mg_coarse_ksp_type`, `-mg_coarse_pc_type`

Mesh Paradigm

The Mesh interface also uses local callback functions

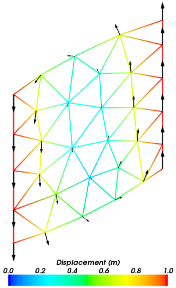
- maps between global `Vec` and local `Vec` (`Section`)
- provides `Complete()` which generalizes `LocalToGlobal()`

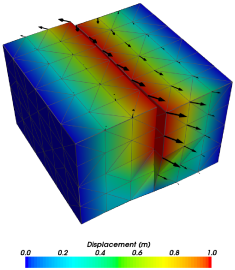
When PETSc needs to evaluate the nonlinear residual $F(x)$,

- Each process evaluates the local residual for each element
- PETSc assembles the global residual automatically

Multiple Mesh Types

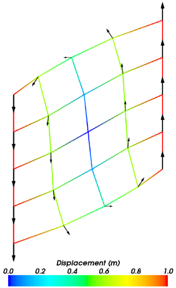
Triangular

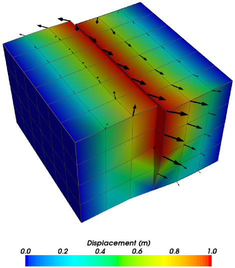




Tetrahedral

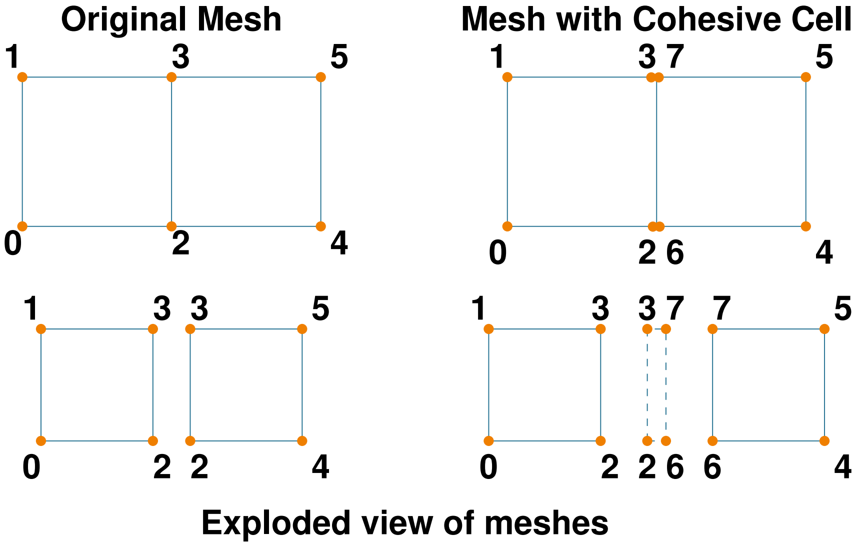
Rectangular





Hexahedral

Cohesive Cells



Cohesive Cells

Cohesive cells are used to enforce slip conditions on a fault

- Demand complex mesh manipulation
  - We allow specification of only fault vertices
  - Must “sew” together on output
- Use Lagrange multipliers to enforce constraints
  - Forces illuminate physics
- Allow different fault constitutive models
  - Simplest is enforced slip
  - Can write a general relation

FIAT

Finite Element Integrator And Tabulator by Rob Kirby

`http://www.fenics.org/fiat`

FIAT understands

- Reference element shapes (line, triangle, tetrahedron)
- Quadrature rules
- Polynomial spaces
- Functionals over polynomials (dual spaces)
- Derivatives

User can build arbitrary elements specifying the Ciarlet triple (K, P, P')

FIAT is part of the FEniCS project, as is the PETSc Sieve module

FFC

FFC is a compiler for variational forms by Anders Logg.

Here is a mixed-form Poisson equation:

$$a((\tau, w), (\sigma, u)) = L((\tau, w)) \qquad \forall (\tau, w) \in V$$

where

$$\begin{aligned} a((\tau, w), (\sigma, u)) &= \int_\Omega \tau\sigma - \nabla \cdot \tau u + w \nabla \cdot u \, dx \\ L((\tau, w)) &= \int_\Omega wf \, dx \end{aligned}$$

FFC

FFC is a compiler for variational forms by Anders Logg.

```
shape = "triangle"
BDM1 = FiniteElement("Brezzi-Douglas-Marini",shape,1)
DG0 = FiniteElement("Discontinuous Lagrange",shape,0)

element = BDM1 + DG0
(tau, w) = TestFunctions(element)
(sigma, u) = TrialFunctions(element)

f = Function(DG0)

a = (dot(tau, sigma) - div(tau)*u + w*div(sigma))*dx
L = w*f*dx
```

FFC

FFC is a compiler for variational forms by Anders Logg.

Here is a discontinuous Galerkin formulation of the Poisson equation:

$$a(v,u) = L(v) \qquad \forall v \in V$$

where

$$\begin{aligned}
a(v,u) &= \int_\Omega \nabla u \cdot \nabla v \, dx \\
&+ \sum_S \int_S - < \nabla v > \cdot [[u]]_n - [[v]]_n \cdot < \nabla u > - (\alpha/h) vu \, dS \\
&+ \int_{\partial\Omega} -\nabla v \cdot [[u]]_n - [[v]]_n \cdot \nabla u - (\gamma/h) vu \, ds \\
L(v) &= \int_\Omega vf \, dx
\end{aligned}$$

FFC

FFC is a compiler for variational forms by Anders Logg.

```
DG1 = FiniteElement("Discontinuous Lagrange",shape,1)
v = TestFunctions(DG1)
u = TrialFunctions(DG1)
f = Function(DG1)
g = Function(DG1)
n = FacetNormal("triangle")
h = MeshSize("triangle")
a = dot(grad(v), grad(u))*dx
  - dot(avg(grad(v)), jump(u, n))*dS
  - dot(jump(v, n), avg(grad(u)))*dS
  + alpha/h*dot(jump(v, n) + jump(u, n))*dS
  - dot(grad(v), jump(u, n))*ds
  - dot(jump(v, n), grad(u))*ds
  + gamma/h*v*u*ds
L = v*f*dx + v*g*ds
```

PetFMM

PetFMM is an freely available implementation of the
Fast Multipole Method

- Leverages PETSc
  - Same open source license
  - Uses Sieve for parallelism
- Extensible design in C++
  - Templated over the kernel
  - Templated over traversal for evaluation
- MPI implementation
  - Novel parallel strategy for anisotropic/sparse particle distributions
  - PetFMM–A dynamically load-balancing parallel fast multipole library
  - 86% efficient strong scaling on 64 procs
- Example application using the Vortex Method for fluids
- (coming soon) GPU implementation

Outline

Conclusions

PETSc can help you

- easily construct a code to test your ideas
- scale an existing code to large or distributed machines
- incorporate more scalable or higher performance algorithms
- tune your code to new architectures

Conclusions

PETSc can help you

- easily construct a code to test your ideas
    - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
- incorporate more scalable or higher performance algorithms
- tune your code to new architectures

Conclusions

PETSc can help you

- easily construct a code to test your ideas
    - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
    - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
- tune your code to new architectures

Conclusions

PETSc can help you

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition or multigrid
- tune your code to new architectures

Conclusions

PETSc can help you

- easily construct a code to test your ideas
  - Lots of code construction, management, and debugging tools
- scale an existing code to large or distributed machines
  - Using `FormFunctionLocal()` and scalable linear algebra
- incorporate more scalable or higher performance algorithms
  - Such as domain decomposition or multigrid
- tune your code to new architectures
  - Using profiling tools and specialized implementations