The Portable Extensible Toolkit for Scientific Computing

Matthew Knepley

Department of Computational and Applied Mathematics
Rice University

Lecture 7: Linear Algebra in PETSc
CAAM 519 Computational Science I
Houston, TX September 9, 2015
Never believe *anything*,

unless you can run it.
Never believe *anything*,

unless you can run it.
The PETSc Team

Bill Gropp
Barry Smith
Satish Balay
Jed Brown
Matt Knepley
Lisandro Dalcin
Hong Zhang
Mark Adams
Toby Issac
1 Getting Started with PETSc

- Who uses 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 PETSc Integration
PETSc was developed as a Platform for Experimentation

We want to experiment with different

- Models
- Discretizations
- Solvers
- Algorithms

which blur these boundaries
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
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
Outline

1. Getting Started with PETSc
   • Who uses 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?
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**)
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
PyLith

- Multiple problems
  - Dynamic rupture
  - Quasi-static relaxation
- Multiple models
  - Nonlinear visco-plastic
  - Finite deformation
  - Fault constitutive models
- Multiple meshes
  - 1D, 2D, 3D
  - Hex and tet meshes
- Parallel
  - PETSc solvers
  - DMPlax mesh management

\(^a\) Aagaard, Knepley, Williams
Multiple Mesh Types

- Triangular
- Tetrahedral
- Rectangular
- Hexahedral
Magma Dynamics

- Couples scales
  - Subduction
  - Magma Migration
- Physics
  - Incompressible fluid
  - Porous solid
  - Variable porosity
- Deforming matrix
  - Compaction pressure
- Code generation
  - FEniCS
- Multiphysics Preconditioning
  - PETSc FieldSplit

\(^a\text{Katz, Speigelman}\)
Magma Dynamics

- Couples scales
  - Subduction
  - Magma Migration
- Physics
  - Incompressible fluid
  - Porous solid
  - Variable porosity
- Deforming matrix
  - Compaction pressure
- Code generation
  - FEniCS
- Multiphysics Preconditioning
  - PETSc FieldSplit

\(^a^Katz, Speigelman
Fracture Mechanics

- Full variational formulation
  - Phase field
  - Linear or Quadratic penalty

- Uses TAO optimization
  - Necessary for linear penalty
  - Backtacking

- No prescribed cracks (movie)
  - Arbitrary crack geometry
  - Arbitrary intersections

- Multiple materials
  - Composite toughness

---

* Bourdin

---

*a* Bourdin

---
Fracture Mechanics

1 Bourdin
Vortex Method

\( t = 000 \)

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re

- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- Parallelism
  - MPI
  - GPU

\(^a\text{Cruz, Yokota, Barba, Knepley} \)
Vortex Method

\( t = 100 \)

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

\(^a\text{Cruz, Yokota, Barba, Knepley}\)
Vortex Method

\(t = 200\)

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re
- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates
- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner
- Parallelism
  - MPI
  - GPU

\(^a\)Cruz, Yokota, Barba, Knepley
Vortex Method

t = 300

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re

- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- Parallelism
  - MPI
  - GPU

\(^a\text{Cruz, Yokota, Barba, Knepley}\)
Vortex Method

Incompressible Flow
- Gaussian vortex blobs
- High Re

PetFMM
- 2D/3D domains
- Automatic load balancing
- Variety of kernels
- Optimized with templates

PetRBF
- Variety of RBFs
- Uses PETSc solvers
- Scalable preconditioner

Parallelism
- MPI
- GPU

\[ t = 400 \]

\[ ^a \text{Cruz, Yokota, Barba, Knepley} \]
Vortex Method

$t = 500$

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re

- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- Parallelism
  - MPI
  - GPU

\(^a\)Cruz, Yokota, Barba, Knepley
Getting Started with PETSc

Vortex Method

\[ t = 600 \]

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re

- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- Parallelism
  - MPI
  - GPU

\[ ^a\text{Cruz, Yokota, Barba, Knepley} \]
Vortex Method
\( t = 700 \)

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re

- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- Parallelism
  - MPI
  - GPU

---

\(^a\)Cruz, Yokota, Barba, Knepley
Vortex Method

\( t = 800 \)

- Incompressible Flow
  - Gaussian vortex blobs
  - High Re

- PetFMM
  - 2D/3D domains
  - Automatic load balancing
  - Variety of kernels
  - Optimized with templates

- PetRBF
  - Variety of RBFs
  - Uses PETSc solvers
  - Scalable preconditioner

- Parallelism
  - MPI
  - GPU

\(^a\text{Cruz, Yokota, Barba, Knepley} \)
Gravity Anomaly Modeling

- Potential Solution
  - Kernel of inverse problem
  - Needs optimal algorithm

- Implementations
  - Direct Summation
  - FEM
  - FMM

- Parallelism
  - MPI
  - 4000+ cores
  - All methods scalable

---

\(^a\text{May, Knepley}\)
**Rheologies**
- Maxwell
- Grade 2
- Oldroyd-B

**Stabilization**
- DG
- SUPG
- EVSS
- DEVSS
- Macroelement

**Automation**
- FIAT (elements)
- FFC (weak forms)

\(^{a}\text{Terrel}\)
FEniCS-Apps
Rheagen

- Rheologies
  - Maxwell
  - Grade 2
  - Oldroyd-B

- Stabilization
  - DG
  - SUPG
  - EVSS
  - DEVSS
  - Macroelement

- Automation
  - FIAT (elements)
  - FFC (weak forms)
Real-time Surgery

- Brain Surgery
  - Elastic deformation
  - Overlaid on MRI
  - Guides surgeon

- Laser Thermal Therapy
  - PDE constrained optimization
  - Per-patient calibration
  - Thermal inverse problem

\(^a\text{Warfield, Ferrant, et.al.}\)
Real-time Surgery

- Brain Surgery
  - Elastic deformation
  - Overlaid on MRI
  - Guides surgeon

- Laser Thermal Therapy
  - PDE constrained optimization
  - Per-patient calibration
  - Thermal inverse problem

\[^{a}\text{Fuentes, Oden, et.al.}\]
Outline

1. Getting Started with PETSc
   - Who uses 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`
Getting Started with PETSc

- Who uses 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?
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 v3.6.1
Unpacking PETSc

- Just clone development repository
  - `git clone http://bitbucket.org/petsc/petsc.git`
  - `git clone -rv3.6.1 petsc petsc-3.6.1`

  or

- Unpack the tarball
  - `tar xzf petsc.tar.gz`
Exercise 1

Download and Unpack PETSc!
1 Getting Started with PETSc

- Who uses 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
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
$PETSC_DIR/configure

--download-triangle --download-ctetgen
--download-chaco --download-parmetis
--download-hdf5 --download-netcdf --download-exodusii
--with-cuda
--with-cudac='nvcc -m64' --with-cuda-arch=sm_10
--with-cuda-only
--with-opencl
--with-opencl-include=/System/Library/Frameworks/OpenCL.framework/Headers/
--with-opencl-lib=/System/Library/Frameworks/OpenCL.framework/OpenCL
--with-precision=single
Configuring PETSc for FEM

$PETSC_DIR/configure
   --download-triangle --download-ctetgen
   --download-chaco --download-parmetis
   --download-hdf5 --download-netcdf --download-exodusii
   --with-cuda
      --with-cudac='nvcc -m64' --with-cuda-arch=sm_10
   --with-cuda-only
   --with-opencl
      --with-opencl-include=/System/Library/Frameworks/OpenCL.framework/Headers/
   --with-opencl-lib=/System/Library/Frameworks/OpenCL.framework/OpenCL
   --with-precision=single
Preparing PETSc for FEM

```bash
$PETSC_DIR/configure
    --download-triangle --download-ctetgen
    --download-chaco --download-parmetis
    --download-hdf5 --download-netcdf --download-exodusii
    --with-cuda
    --with-cudac='nvcc -m64' --with-cuda-arch=sm_10
    --with-cuda-only
    --with-opencl
    --with-opencl-include=/System/Library/Frameworks/OpenCL.framework/Headers/
    --with-opencl-lib=/System/Library/Frameworks/OpenCL.framework/OpenCL
    --with-precision=single
```
Configuring PETSc for FEM

```
$PETSC_DIR/configure
  --download-triangle --download-ctetgen
  --download-chaco --download-parmetis
  --download-hdf5 --download-netcdf --download-exodusii
  --with-cuda
  --with-cudac='nvcc -m64' --with-cuda-arch=sm_10
  --with-cuda-only
  --with-opencl
  --with-opencl-include=/System/Library/Frameworks/OpenCL.framework/Headers/
  --with-opencl-lib=/System/Library/Frameworks/OpenCL.framework/OpenCL
  --with-precision=single
```
$PETSC_DIR/configure
  --download-triangle  --download-ctetgen
  --download-chaco   --download-parmetis
  --download-hdf5   --download-netcdf --download-exodusii
  --with-cuda
  --with-cudac='nvcc -m64'  --with-cuda-arch=sm_10
  --with-cuda-only
  --with-opencl
  --with-opencl-include=/System/Library/Frameworks/OpenCL.framework/Headers/
  --with-opencl-lib=/System/Library/Frameworks/OpenCL.framework/OpenCL
  --with-precision=single
$PETSC_DIR/configure
  --download-triangle --download-ctetgen
  --download-chaco --download-parmetis
  --download-hdf5 --download-netcdf --download-exodusii
  --with-cuda
  --with-cudac='nvcc -m64' --with-cuda-arch=sm_10
  --with-cuda-only
  --with-opencl
  --with-opencl-include=/System/Library/Frameworks/OpenCL.framework/Headers/
  --with-opencl-lib=/System/Library/Frameworks/OpenCL.framework/OpenCL
  --with-precision=single
Configuring PETSc for FEM

$PETSC_DIR/configure
   --download-triangle --download-ctetgen
   --download-chaco --download-parmetis
   --download-hdf5 --download-netcdf --download-exodusii
   --with-cuda
   --with-cudac='nvcc -m64' --with-cuda-arch=sm_10
   --with-cuda-only
   --with-opencl
   --with-opencl-include=/System/Library/Frameworks/OpenCL.framework-Headers/
   --with-opencl-lib=/System/Library/Frameworks/OpenCL.framework/OpenCL
   --with-precision=single
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. Getting Started with PETSc
   - Who uses 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
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. Getting Started with PETSc
   - Who uses 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?
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
Can run any PETSc example
- `python ./config/builder2.py check
  $PETSC_DIR/src/snes/examples/tutorials/ex5.c`

Checks against test output
- Ignores if no output is present

Can specify multiple files
- `python ./config/builder2.py check
  [$PETSC_DIR/src/snes/examples/tutorials/ex5.c,
  extraFile.c]`

Can also run using MPI directly
- Use `-retain` to keep executable
- `mpiexec ./$PETSC_ARCH/lib/lib-ex5/ex5
  -snes_monitor`
Using MPI

- **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
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())
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`
Outline

1. Getting Started with PETSc
   - Who uses 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 Ahmadia
1. Getting Started with PETSc

2. PETSc Integration
   - Initial Operations
   - Vector Algebra
   - Matrix Algebra
2. PETSc Integration
   - Initial Operations
   - Vector Algebra
   - Matrix Algebra
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 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
  - We use Git
- 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
Outline

2 PETSc Integration
   - Initial Operations
   - Vector Algebra
   - Matrix Algebra
What are PETSc vectors?

- Fundamental objects representing
  - solutions
  - right-hand sides
  - coefficients

- Each process locally owns a subvector of contiguous global data
How do I create vectors?

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

\[ \text{VecSetValues}(\text{Vec } v, \text{PetscInt } n, \text{PetscInt} \text{ rows}[], \text{PetscScalar values}[], \text{InsertMode} \text{ mode}) \]

- Mode is either \text{INSERT\_VALUES} or \text{ADD\_VALUES}
- Two phases allow overlap of communication and computation
  - \text{VecAssemblyBegin}(\text{Vecv})
  - \text{VecAssemblyEnd}(\text{Vecv})
One Way to Set the Elements of a Vector

```c
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);
```
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

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VecAXPY(Vec y, PetscScalar a, Vec x)</td>
<td>$y = y + a \cdot x$</td>
</tr>
<tr>
<td>VecAYPX(Vec y, PetscScalar a, Vec x)</td>
<td>$y = x + a \cdot y$</td>
</tr>
<tr>
<td>VecWAYPX(Vec w, PetscScalar a, Vec x, Vec y)</td>
<td>$w = y + a \cdot x$</td>
</tr>
<tr>
<td>VecScale(Vec x, PetscScalar a)</td>
<td>$x = a \cdot x$</td>
</tr>
<tr>
<td>VecCopy(Vec y, Vec x)</td>
<td>$y = x$</td>
</tr>
<tr>
<td>VecPointwiseMult(Vec w, Vec x, Vec y)</td>
<td>$w_i = x_i \cdot y_i$</td>
</tr>
<tr>
<td>VecMax(Vec x, PetscInt *idx, PetscScalar *r)</td>
<td>$r = \max r_i$</td>
</tr>
<tr>
<td>VecShift(Vec x, PetscScalar r)</td>
<td>$x_i = x_i + r$</td>
</tr>
<tr>
<td>VecAbs(Vec x)</td>
<td>$x_i =</td>
</tr>
<tr>
<td>VecNorm(Vec x, NormType type, PetscReal *r)</td>
<td>$r =</td>
</tr>
</tbody>
</table>
It is sometimes more efficient to directly access local storage of a $\text{Vec}$.  
- **PETSc** allows you to access the local storage with $\text{VecGetArray}($ $\text{Vec}$, $\text{double } *[])$
- You must return the array to **PETSc** when you finish $\text{VecRestoreArray}($ $\text{Vec}$, $\text{double } *[])$
- 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

```c
#include "finlude/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)
```
```c
#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)
```
with v as a:
    for i in range(len(a)):
        a[i] = 5.0*i
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

2. PETSc Integration
   - Initial Operations
   - Vector Algebra
   - 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, PetscInt m, PetscInt n, M, N)`
- `MatSetType(Mat, MatTypetypeName)`
- `MatSetFromOptions(Mat)`
  - Can set the type at runtime
- `MatSeqAIJPreallocation(Mat, PetscInt nz, 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
The PETSc \textbf{Mat} has a single user interface,

- Matrix assembly
  - \texttt{MatSetValues()}
- Matrix-vector multiplication
  - \texttt{MatMult()}
- Matrix viewing
  - \texttt{MatView()}

but multiple underlying implementations.

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

A matrix is defined by its \textit{interface}, not by its \textit{data structure}. 

---

**MGK (Rice)**

**PETSc**

**CAAM 519** 76 / 81
Matrix Assembly

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

\texttt{MatSetValues(Matm, m, rows[], n, cols [], values [], mode)}
- \texttt{mode} is either \texttt{INSERT\_VALUES} or \texttt{ADD\_VALUES}
- Logically dense block of values

Two phase assembly allows overlap of communication and computation

\texttt{MatAssemblyBegin(Matm, MatAssemblyType type)}
\texttt{MatAssemblyEnd(Matm, MatAssemblyType type)}
- \texttt{type} is either \texttt{MAT\_FLUSH\_ASSEMBLY} or \texttt{MAT\_FINAL\_ASSEMBLY}
One Way to Set the Elements of a Matrix
Simple 3-point stencil for 1D Laplacian

```c
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);
}
```
Parallel Sparse Matrix Layout

- proc 0
- proc 1
- proc 2
- proc 3
- proc 4
- proc 5

- diagonal blocks
- offdiagonal blocks
A Better Way to Set the Elements of a Matrix
Simple 3-point stencil for 1D Laplacian

\[
v[0] = -1.0; \ v[1] = 2.0; \ v[2] = -1.0;
\]

\[
\text{MatGetOwnershipRange}(A, \& \text{start}, \& \text{end});
\]

\[
\text{for} (\text{row} = \text{start}; \ \ \text{row} < \text{end}; \ \ \text{row}++) {
\]
\[
\text{cols}[0] = \text{row} - 1; \ \ \text{cols}[1] = \text{row}; \ \ \text{cols}[2] = \text{row} + 1;
\]
\[
\text{if} (\text{row} == 0) {
\]
\[
\text{MatSetValues}(A, 1, \& \text{row}, 2, \& \text{cols}[1], \& v[1], \text{INSERT_VALUES});
\]
\[
} \ \ \text{else if} (\text{row} == \text{N} - 1) {
\]
\[
\text{MatSetValues}(A, 1, \& \text{row}, 2, \text{cols}, v, \text{INSERT_VALUES});
\]
\[
} \ \ \text{else} {
\]
\[
\text{MatSetValues}(A, 1, \& \text{row}, 3, \text{cols}, v, \text{INSERT_VALUES});
\]
\[
}
\]

\[
\text{MatAssemblyBegin}(A, \text{MAT_FINAL_ASSEMBLY});
\]

\[
\text{MatAssemblyEnd}(A, \text{MAT_FINAL_ASSEMBLY});
\]
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)