UNIVERSITY OF SASKATCHEWAN Numerical Simulation Laboratory

Introduction to using PETSc for scientific computing

Kevin R. Green, Raymond J. Spiteri

Department of Computer Science
University of Saskatchewan

Westgrid HPC Summer School
Saskatoon, SK
July 24-27, 2017

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Outline

1. Introduction

2. PETSc data structures and classes

3. Solving problems

4. Scaling of solutions

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Outline

1. Introduction

2. PETSc data structures and classes

3. Solving problems

4. Scaling of solutions

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Source code for workshop examples

Exercise 1: Get example codes

Available in the git repository:
https://github.com/krg86/201707_petsc_workshop

Clone to the machine where you want to work:

```
$ ssh username@graham.computecanada.ca
$ git clone https://github.com/kevinrichardgreen/201707_petsc_workshop.git
```

The pdf of these slides is in there too.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

During the workshop

- Stop me to ask for clarifications.
    - I'll do my best to help out / point you in the right direction.
    - I don't know everything $\in \Omega_{\texttt{PETSc}}$, but
    - We can all learn from this!
- Take some notes. I say things that aren't necessarily written down here.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

PETSc

https://www.mcs.anl.gov/petsc/

PETSc (|' petsi:|) stands for the Portable, Extensible Toolkit for Scientific computing.

It provides a suite of libraries for the numerical solution of partial differential equations (PDEs) and related problems.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Portable Extensible Toolkit for Scientific computing

- Architecture
  - tightly coupled (e.g. Cray, Blue Gene)
  - loosely coupled such as network of workstations
  - GPU clusters (many vector and sparse matrix kernels)
- Operating systems (Linux, Mac, Windows, BSD, proprietary Unix)
- Any compiler
- Real/complex, single/double/quad precision, 32/64-bit int
- Usable from C, C++, Fortran 77/90, Python, and MATLAB
- Free to everyone (2-clause BSD license), open development
- 10^{12} unknowns, full-machine scalability on Top-10 systems
- Same code runs performantly on a laptop

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Portable **Extensible** Toolkit for Scientific computing

Philosophy: Everything has a plugin architecture

- Vectors, Matrices, Coloring/ordering/partitioning algorithms
- Preconditioners, Krylov accelerators
- Nonlinear solvers, Time integrators
- Spatial discretizations/topology

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Portable Extensible **Toolkit** for Scientific computing

Algorithms, (parallel) debugging aids, low-overhead profiling.
Composability

- Try new algorithms by choosing from product space and composing existing algorithms (multilevel, domain decomposition, splitting).

Experimentation

- It is **not** possible to pick the best solver *a priori*.
  What will deliver best/competitive performance for a given physics, discretization, architecture, and problem size?
- PETSc's response: expose an algebra of composition so new solvers can be created at runtime.
- Important to keep solvers decoupled from physics and discretization because we also experiment with those.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Portable Extensible Toolkit for **Scientific computing**

- Computational Scientists
  - PyLith, Underworld, PFLOTRAN, MOOSE, Proteus, PyClaw, CHASTE
- Algorithm Developers (iterative methods and preconditioning)
- Package Developers
  - SLEPc, TAO, Deal.II, Libmesh, FEniCS, PETSc-FEM, MagPar, OOFEM, FreeCFD, OpenFVM
- Hundreds of tutorial-style examples
- Hyperlinked manual, examples, and manual pages for all routines
- Support from petsc-maint@mcs.anl.gov, petsc-users@mcs.anl.gov

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

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

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

PETSc installation

Installing on your own machine (see Appendix)

Installing on a cluster:

- Don't do it.
- Use pre-built install.
- Consult with computing staff if not available.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Enabling PETSc

PETSc requires a couple of environment variables to be set:

- PETSC_DIR - the top level directory of PETSc
- PETSC_ARCH - the *architecture* for which PETSc has been built

Allows multiple ARCH types built on a given system, which can easily be swapped for your specific application code.
ie., having a version configured with debugging flags enabled, and a version without.

Generally on a cluster, PETSC_ARCH is left empty, and only an optimized build is maintained.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

On graham.computecanada.ca

PETSc is pre-built and has a module:

$ module load petsc/3.7.5

This sets up the required environment for compiling PETSc application programs.

The programs can then be run as any other MPI programs.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Testing your PETSc environment

Exercise 2: Testing PETSc env

```
$ module load petsc/3.7.5
$ cd 201707_petsc_workshop/examples/test
$ make MPIVersion
```

- Submit the executable as an MPI job using 8 cores, what kind of memory bandwidth does it show?
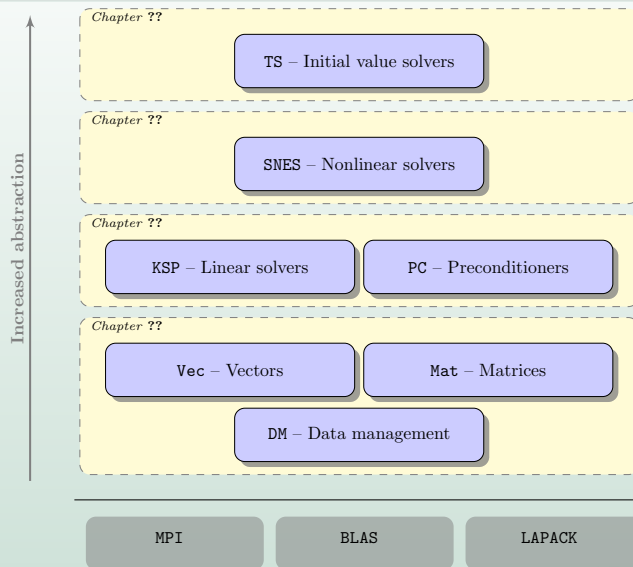- Try submissions with 1, 8, 16, 32 cores.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Outline

1 Introduction

2 PETSc data structures and classes

3 Solving problems

4 Scaling of solutions

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

PETSc data types

- `PetscInt`
  - defaults to 32bit integer
  - can be 64bit int if you're dealing with many unknowns (`--with-64-bit-indices`)
- `PetscScalar`
  - Primary type for your unknowns, depending on configuration can be:
    - Double precision real (default) or complex
    - Single precision
    - Quad precision
- `PetscReal`
  - Same precision as `PetscScalar`, but always real

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

PETSc hierarchy

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Message passing interface (MPI)

Typically, user does not call MPI functions directly.

At most, `MPI_Comm_size()` and `MPI_Comm_rank()` can be used. All other MPI communication functions are wrapped within PETSc functions.

However, some degree of understanding how MPI works is needed to create PETSc code that can scale well.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

MPI communicators in PETSc

- PETSc objects need an `MPI_Comm` in their constructor
  - `PETSC_COMM_SELF` for serial objects
  - `PETSC_COMM_WORLD` common, but *not* required
- Can split communicators, spawn processes on new communicators, etc
- Operations are one of
  - Not Collective: *ie.,* `VecGetLocalSize()`, `MatSetValues()`
  - Logically Collective: *ie.,* `KSPSetType()`
    - checked when running in debug mode
  - Neighbor-wise Collective: `VecScatterBegin()`, `MatMult()`
    - Point-to-point communication between two processes
  - Collective: `VecNorm()`, `MatAssemblyBegin()`, `KSPCreate()`
    - Global communication, synchronous
- Deadlock if some process doesn't participate (e.g. wrong order)

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

PetscObject

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 command line
SetUp()	perform other initialization
View()	view the object
Destroy()	cleanup object allocation

Also, all objects support the –help option.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Options

Ways to set options:

- Command line
- Filename in the third argument of `PetscInitialize()`
- $\sim$/.petscrc
- $PWD/.petscrc
- $PWD/petscrc
- `PetscOptionsInsertFile()`
- `PetscOptionsInsertString()`
- `PETSC_OPTIONS` environment variable
- command line option `-options_file [file]`

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Vectors (Vec)

- Extension of arrays.
  - Parallel (global) representations
  - Serial (local) representations
- Think of them just like vectors in linear algebra.
- Used most often to store:
  - solution to your problem
  - residuals - nonlinear and linear
- Can be used to hold location dependent field data for the problem as well.
- Can extract data as arrays for manipulation.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Distributed Vec

Parallel vector stored on 3 processors, as an example

$$\mathbf{u}_{\text{global}} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} \quad \begin{cases} \mathbf{u}_{\text{local}}^{(0)} = \begin{bmatrix} u_0 & u_1 & u_2 & \mathbf{u}_{\text{ghost}}^{(0)T} \end{bmatrix}^T \\ \\ \mathbf{u}_{\text{local}}^{(1)} = \begin{bmatrix} u_3 & u_4 & u_5 & \mathbf{u}_{\text{ghost}}^{(1)T} \end{bmatrix}^T \\ \\ \mathbf{u}_{\text{local}}^{(2)} = \begin{bmatrix} u_6 & u_7 & \mathbf{u}_{\text{ghost}}^{(2)T} \end{bmatrix}^T \end{cases}$$

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Vec operations

Man pages:
http:
//www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/Vec/index.html

- Things you expect with vectors:
    - axpy, norms, dot products, etc.
- Explicit access to entries:
    - VecGetArray(), VecRestoreArray()
    - and their *read only* variants
- Parallel communication:
    - assembly, scatter, gather
- Examples: 201707_petsc_workshop/examples/vec/

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Vec exercise

Exercise 3: Basic vector routines

- Investigate the source code `examples/vec/ex1.c`, and the structure of its makefile.
- Set the vector size to 1000 using a command-line option.
- Investigate other options available to this program with the `-help` option.
- Can you get the program to output the line:
  `Total flops over all processors 32991.` ?

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Matrices (Mat)

What are PETSc matrices?

- Linear operators on finite dimensional vector spaces.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Matrices (`Mat`)

What are `PETSc` matrices?

- Linear operators on finite dimensional vector spaces.
- 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, SuperLU, UMFPack, Hypre, Elemental

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Matrices (`Mat`)

What are `PETSc` matrices?

- Linear operators on finite dimensional vector spaces.
- 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, SuperLU, UMFPack, Hypre, Elemental

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Distributed Mat

Nonzero entries of a matrix stored on 3 processors.

$$
\left[
\begin{array}{ccc|ccc|cc}
1 & 2 & 0 & 3 & 0 & 0 & 4 & 0 \\
5 & 6 & 7 & 0 & 0 & 8 & 9 & 0 \\
0 & 10 & 11 & 12 & 0 & 0 & 13 & 14 \\
\hline
0 & 0 & 15 & 16 & 17 & 0 & 0 & 18 \\
19 & 0 & 0 & 20 & 21 & 22 & 23 & 0 \\
0 & 24 & 0 & 0 & 25 & 26 & 27 & 28 \\
\hline
0 & 29 & 0 & 0 & 0 & 30 & 31 & 32 \\
33 & 0 & 0 & 34 & 35 & 0 & 36 & 37
\end{array}
\right]
$$

Proc 0

Proc 1

Proc 2

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Matrix Polymorphism

The PETSc Mat has a single user interface,

- Matrix assembly: MatSetValues()
- Matrix-vector multiplication: MatMult()
- Matrix viewing: MatView()

but multiple underlying implementations.

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

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

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Important matrices

- Sparse (e.g. discretization of a PDE operator)
- Inverse of *anything* interesting $B = A^{-1}$
- Jacobian of a nonlinear function $Jy = \lim_{\epsilon \to 0} \frac{F(x+\epsilon y) - F(x)}{\epsilon}$
- Fourier transform $\mathcal{F}, \mathcal{F}^{-1}$
- Other fast transforms, e.g. Fast Multipole Method
- Low rank correction $B = A + uv^T$
- Schur complement $S = D - CA^{-1}B$
- Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
- Linearization of a few steps of a time integration

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Important matrices

- Sparse (e.g. discretization of a PDE operator)
- Inverse of *anything* interesting $B = A^{-1}$
- Jacobian of a nonlinear function $Jy = \lim_{\epsilon \to 0} \frac{F(x+\epsilon y) - F(x)}{\epsilon}$
- Fourier transform $\mathcal{F}, \mathcal{F}^{-1}$
- Other fast transforms, e.g. Fast Multipole Method
- Low rank correction $B = A + uv^T$
- Schur complement $S = D - CA^{-1}B$
- Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
- Linearization of a few steps of a time integration

  - These matrices are dense. Never form them.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Important matrices

- Sparse (e.g. discretization of a PDE operator)
- Inverse of *anything* interesting $B = A^{-1}$
- Jacobian of a nonlinear function $Jy = \lim_{\epsilon \to 0} \frac{F(x+\epsilon y) - F(x)}{\epsilon}$
- Fourier transform $\mathcal{F}, \mathcal{F}^{-1}$
- Other fast transforms, e.g. Fast Multipole Method
- Low rank correction $B = A + uv^T$
- Schur complement $S = D - CA^{-1}B$
- Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
- Linearization of a few steps of a time integration

  - These are not very sparse. Don't form them.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Important matrices

- Sparse (e.g. discretization of a PDE operator)
- Inverse of *anything* interesting $B = A^{-1}$
- Jacobian of a nonlinear function $Jy = \lim_{\epsilon \to 0} \frac{F(x+\epsilon y) - F(x)}{\epsilon}$
- Fourier transform $\mathcal{F}, \mathcal{F}^{-1}$
- Other fast transforms, e.g. Fast Multipole Method
- Low rank correction $B = A + uv^T$
- Schur complement $S = D - CA^{-1}B$
- Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
- Linearization of a few steps of a time integration

  - None of these matrices "have entries"

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

How to 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
- `MatSetBlockSize(Mat, int bs)`
- `MatXAIJSetPreallocation(Mat,...)`
- `MatSetValues(Mat,...)`
  - `MatSetValuesLocal, MatSetValuesStencil`
  - `MatSetValuesBlocked`
- `MatAssemblyBegin/End`

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

What can we do with a matrix that doesn't have entries?

Krylov solvers for $Au = b$

- Finding solutions in the Krylov subspace:
  $\{b, Ab, A^2b, A^3b, \dots\}$
- Convergence rate depends on the spectral properties of the matrix
  - Existence of small polynomials $p_n(A) < \epsilon$ where $p_n(0) = 1$.
  - condition number $\kappa(A) = \|A\|\|A^{-1}\| = \sigma_{\max}/\sigma_{\min}$
  - distribution of singular values, spectrum Λ, pseudospectrum Λ_ϵ

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Krylov subspace problem (KSP)

- Solving linear systems:

$$Au = b$$

- Various *Krylov* methods:
    - GMRES (`gmres`)
    - Conjugate gradient (`cg`)
    - Biconjugate gradient - stabilized (`bcgs`)
    - etc
- Direct methods (external packages recommended):
    - LU factorization
    - Cholesky factorization

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Quintessential Krylov example: GMRES

Brute force minimization of residual in $\{b, Ab, A^2b, \dots\}$

1. Use Arnoldi to orthogonalize the nth subspace, producing

$$AQ_n = Q_{n+1}H_n$$

2. Minimize residual in this space by solving the overdetermined system

$$H_ny_n = e_1^{(n+1)}$$

using QR-decomposition, updated cheaply at each iteration.

Properties

- Converges in n steps for all right hand sides if there exists a polynomial of degree n such that $\|p_n(A)\| < tol$ and $p_n(0) = 1$.

- Residual is monotonically decreasing, robust in practice

- Restarted variants are used to bound memory requirements

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

How to set up a linear solver

- `KSPCreate(MPI_Comm, KSP *)`
- `KSPSetOperators(KSP, Mat A, Mat A_p)`
- `KSPSetFromOptions(KSP)`
    - Can set all aspects of solver at runtime
- `KSPGetPC(KSP,PC *)`
    - Explicitly get the preconditioner to manipulate
    - can be set from command line if `KSPSetFromOptions` was used
- `KSPSolve(KSP,Vec b, Vec u)`

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Preconditioning?

Definition (Preconditioner)

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

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

PC idea

Idea: improve the conditioning of the Krylov operator

- Left preconditioning

$$(P^{-1}A)u = P^{-1}b$$

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

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

PC idea

Idea: improve the conditioning of the Krylov operator

- Left preconditioning

$$(P^{-1}A)u = P^{-1}b$$

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

- Right preconditioning

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

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

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

PC idea

Idea: improve the conditioning of the Krylov operator

- Left preconditioning

$$(P^{-1}A)u = P^{-1}b$$

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

- Right preconditioning

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

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

- The product $P^{-1}A$ or AP^{-1} is *not* formed.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

KSP exercise

Exercise 4: Simple linear solve

- Looking at the code in `examples/ksp/ex2.c`, determine:
  - Where the matrix values are set.
  - Where the solver type is set.
  - How the error is computed.
- What are the default KSP type and PC type?
  - *Don't forget about* `-help`
- Do defaults change for serial and parallel execution?
- Can you solve the system with LU factorization?

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

LU factorization

- Handled as a preconditioner.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

LU factorization

- Handled as a preconditioner.
- Logic: LU factorization with back/forward substitution is a *perfect* left preconditioner.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

LU factorization

- Handled as a preconditioner.
- Logic: LU factorization with back/forward substitution is a *perfect* left preconditioner.
- PETSc only provides a serial LU preconditioner...

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

LU factorization

- Handled as a preconditioner.
- Logic: LU factorization with back/forward substitution is a *perfect* left preconditioner.
- PETSc only provides a serial LU preconditioner...
- But we can use external packages for parallel solves: MUMPS, SuperLU_dist

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

LU example

Exercise 5: Solve with LU factorization

- Use the options:
  -pc_type lu -pc_factor_mat_solver_package mumps
  to solve using LU factorization in parallel.

- Look at -ksp_view to see what info this gives you.

An alternative is to use
-pc_factor_mat_solver_package superlu_dist.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Structured nonlinear equation solver (SNES)

- Standard form of a nonlinear system

$$F(u) = 0$$

- Newton iteration

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

- Quadratically convergent near a root: $|u^{n+1} - u^*| \in \mathcal{O}\left(|u^n - u^*|^2\right)$
- *Quasi-Newton* methods can use a different $J(u)$

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Structured nonlinear equation solver (SNES)

- Standard form of a nonlinear system

$$F(u) = 0$$

- Newton iteration

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

- Quadratically convergent near a root: $|u^{n+1} - u^*| \in \mathcal{O}\left(|u^n - u^*|^2\right)$
- *Quasi-Newton* methods can use a different $J(u)$

Example: Nonlinear Poisson equation

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

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

How to set up a SNES I

- SNESCreate(SNES *)
- The SNES interface is based upon callback functions
    - FormFunction(...), set by SNESSetFunction()
    - FormJacobian(...), set by SNESSetJacobian()
- When PETSc needs to evaluate F and J,
    - Solver calls the **user's** function
    - User function gets application state through the ctx variable
        - cast to/from a void pointer (see upcoming exercise)
        - PETSc *never* sees application data
- SNESSetFromOptions(SNES)
  SNESSolve(SNES,Vec b, Vec u)

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

How to set up a SNES II

The user provided function which calculates the **nonlinear residual** has signature

```
PetscErrorCode (*func)(SNES snes,Vec u,Vec r,void *ctx)
```

u: The current solution

r: The residual

ctx: The user context passed to SNESSetFunction()

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

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

How to set up a SNES III

Similar story for the function to evaluate Jacobian:

```
PetscErrorCode (*func)(SNES snes,Vec u,Mat J,
                       Mat Jpre,void *ctx)
```

   u: The current solution

   J: The Jacobian

Jpre: The Jacobian preconditioning matrix (possibly J itself)

 ctx: The user context passed to SNESSetFunction()

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

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

How to set up a SNES III

Similar story for the function to evaluate Jacobian:

```
PetscErrorCode (*func)(SNES snes,Vec u,Mat J,
                       Mat Jpre,void *ctx)
```

   u: The current solution
   J: The Jacobian
Jpre: The Jacobian preconditioning matrix (possibly J itself)
 ctx: The user context passed to SNESSetFunction()
   - Use this to pass application information, e.g. physical constants

Alternatively, you can use

- a built-in sparse finite difference approximation ("coloring")

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

SNES global strategies

Newton's method can diverge if not close to a solution.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

SNES global strategies

Newton's method can diverge if not close to a solution.

Line search (`newtonls`):

1. Update **direction** determined by linear solve in Newton method
2. Update **size** chosen by maximizing decrease in nonlinear residual in the update direction

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

SNES global strategies

Newton's method can diverge if not close to a solution.

Line search (`newtonls`):

1. Update **direction** determined by linear solve in Newton method
2. Update **size** chosen by maximizing decrease in nonlinear residual in the update direction

Trust region (`newtontr`):

1. Trust region **size** chosen based on how well a local approximation matches F
2. Update **direction** chosen by maximizing the decrease in nonlinear residual within the trust region

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

SNES exercise

Exercise 6: Simple nonlinear solve

- Investigate the code in examples/snes/ex2.c
  (note this is a serial example!), determine:
    - How the residual is evaluated.
    - How the Jacobian is evaluated.
    - What is the monitor doing?
- What are the default methods employed?
- Can you get the program to output the reason for convergence?
- Can you monitor the linear system solves as well?

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Timestepping (TS)

Typically applied to *method of lines* discretizations of PDEs.

- Additive Runge-Kutta IMEX methods

$$G(t, u, \dot{u}) = F(t, u)$$
$$J_\alpha = \alpha G_{\dot{u}} + G_u$$

- Stiff part in G.
- Orders 2 through 5, embedded error estimates
- Dense output, hot starts for Newton
- Extensible adaptive controllers
- Easy to register new methods: TSARKIMEXRegister()

- Other more simple methods available

- Single step interface so user can control their own time loop

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

Some TS methods

TSTHETA Theta methods

TSALPHA Alpha methods

TSRK Runge-Kutta methods

TSARKIMEX 2-Additive Runge-Kutta methods

TSBDF Backward differentiation methods

TSROSW Rosenbrock-W methods

TSSSP Strong stability methods (Total variation diminishing)

TSGL General linear methods - multi-stage/multi-step

TSSUNDIALS SUNDIALS (external) ode integrators

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

How to set up a TS I

- TSCreate(TS *)
- User provides functions for:
  - Explicit rhs - TSSetRHSFunction()
  - Implicit lhs - TSSetIFunction(), TSSetIJacobian()
- TSSetFromOptions(TS)
- TSSolve(TS,Vec u)

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

How to set up a TS II

Function signatures:

- `FormRHSFunction(ts,t,u,F,void *ctx);`
- `FormIFunction(ts,t,u,udot,G,void *ctx);`
- `FormIJacobian(ts,t,u,udot,alpha,J,J_p,void *ctx);`

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

How to set up a TS II

Function signatures:

- `FormRHSFunction(ts,t,u,F,void *ctx);`
- `FormIFunction(ts,t,u,udot,G,void *ctx);`
- `FormIJacobian(ts,t,u,udot,alpha,J,J_p,void *ctx);`

Can also supply:

- `FormRHSJacobian(ts,t,u,F,J,J_p,void *ctx);`

to use fully implicit methods.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

MPI
Objects
Solvers

TS exercise

Exercise 7: Simple time integration

- Investigate the source code in `examples/ts/ex3.c`, determine:
  - What problem is being solved, and how is it discretized?
  - Is its formulation slightly different from TS as outlined in this document?
  - What is special about linear time integration problems?
- What types of time integration methods can you use, what is the default?
- Run the program with the backward Euler method, solving the linear system with LU factorization.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

Outline

1. Introduction

2. PETSc data structures and classes

3. **Solving problems**

4. Scaling of solutions

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

Spatial discretization
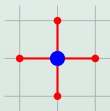
To evaluate a local function, say $F(u)$, each process requires

- its local portion of the vector u
- its *ghost values*, bordering portions of u that are local to other processes

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

Spatial discretization

To evaluate a local function, say $F(u)$, each process requires

- its local portion of the vector u
- its *ghost values*, bordering portions of u that are local to other processes

Example: logically rectangular grids

star-type stencil of width one   box-type stencil of width one

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems
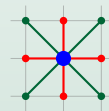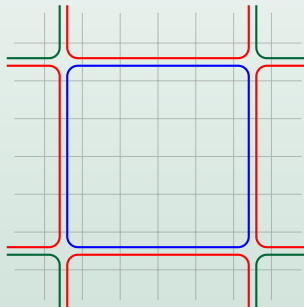
DMDA overview

Distribution **M**anager - **D**istributed **A**rrays

- handle allocation and communication requirements for logically rectangular grids.
- A subset of DM, which is generally more complicated to set up.
- Useful for finite difference and finite volume discretizations.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

DMDA 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

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 (for Proc 0)

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

How to work with a `DMDA` I

Creation:

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

xbdy,ybdy: Specifies periodicity or ghost cells

- DMDA_BOUNDARY_NONE, DMDA_BOUNDARY_GHOSTED, DMDA_BOUNDARY_MIRROR, DMDA_BOUNDARY_PERIODIC

type: Specifies stencil

- DMDA_STENCIL_BOX or DMDA_STENCIL_STAR

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

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

dof: Degrees of freedom per node

s: The stencil width

lm,ln: Alternative array of local sizes

- Use PETSC_NULL for the default

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

How to work with a DMDA II

Vectors and Matrices:

- Global vectors are parallel
    - Each process stores a unique local portion
    - DMCreateGlobalVector(DM dm, Vec *gvec)
- Local vectors are sequential (and usually temporary)
    - Each process stores its local portion plus ghost values
    - DMCreateLocalVector(DM dm, Vec *lvec)
    - includes ghost values!
- Can get arrays from either global or local vectors
    - DMDAVecGetArray(DM,Vec,void *)
    - DMDAVecRestoreArray(DM,Vec,void *)
- Matrices can be preallocated from DMDAs
    - DMCreateMatrix(DM dm,Mat *mat)

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

How to work with a DMDA III

Updating ghost values:

Two-step process, as with matrix assembly

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

The process can be reversed with DMLocalToGlobalBegin() and
DMLocalToGlobalEnd().

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

SNES and Jacobians

- May want to start off your problem solving by using a Finite differenced Jacobian approximation.
    - -snes_fd
- For data in a DMDA, coloring can be used to vastly speed up the finite differencing.
    - -snes_fd_color
- Using an analytic (coded) Jacobian will likely be the fastest.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

FD Jacobian efficiency

Exercise 8: FD Jacobians

- Consider the code in `examples/snes/ex5.c` (This uses an analytical Jacobian)
- Run the code with the option `-da_refine 6`
  - What is this option doing?
- Compare run times when `-snes_fd` vs `-snes_fd_color` are used.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

Convergence problems?

Why isn't SNES converging?

- Analytic Jacobians can be tricky to implement correctly for parallel code, your Jacobian may be wrong.
  - Check with -snes_compare_explicit and -snes_mf_operator -pc_type lu
- Linear system not solved accurately enough?
  - Try LU factorization
  - Try right preconditioning
  - Try -ksp_true_residual
- Singular Jacobian with inconsistent RHS
  - MatNullSpace can be used to set a known null space
- Really strong nonlinearity
  - -snes_linesearch_monitor to view behaviour of line search
  - Try newtontr

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

Debugging a Jacobian

Exercise 9: Output of -snes_compare_explicit

- Consider exercises/snes/ex3.c, in the FormJacobian function, investigate how the field arrays (xx) are obtained from the DMDA.
- Build and execute the code as is.
- Remove "*xx[i]" from line 467.
- Rebuild and execute the code. What happens to the solution?
- Use the command-line options:
  -snes_compare_explicit -snes_max_it 1
  - Which part indicates that the Jacobian is wrong?

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

Example codes

- PETSc has a wealth of example codes - exercises in this workshop have been taken directly from PETSc source.
- Generally located in src/<classname>/example/tutorials directories (see Appendix for obtaining source code).
- Hyperlinked in man pages, for example
  - http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/TS/TSSolve.html
  - http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/DM/DMDACreate3d.html

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

Nonlinear example: 3D Bratu problem

Bratu equation in 3D (examples/snes/ex14.c):

$$-\nabla^2 u - \lambda e^u = 0$$

$$(x, y, z) \in [0, 1]^3$$

$$u = 0 \text{ on boundary}$$

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Spatial discretization
Jacobians!
Further example problems

PDE TS example problem

Reaction diffusion problem in 2D (examples/ts/ex5.c):

$$u_t = D_1 \nabla^2 u - uv^2 + \gamma(1 - u)$$
$$v_t = D_2 \nabla^2 + uv^2 - (\gamma + \kappa)v$$

$$(x, y) \in [0, 1]^2$$

Periodic boundary conditions in all directions.

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Profiling
Predicting

Outline

1 Introduction

2 PETSc data structures and classes

3 Solving problems

4 Scaling of solutions

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Profiling
Predicting

PETSc logging features

- Use -log_view for a performance profile
    - Event timing
    - Event flops
    - Memory usage
    - MPI messages
- Call PetscLogStagePush() and PetscLogStagePop()
    - User can add new stages
- Call PetscLogEventBegin() and PetscLogEventEnd()
    - User can add new events
- Call PetscLogFlops() to include *your* flops
    - *ie.*, in Function or Jacobian assembly
    - flops within PETSc recorded automatically

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Profiling
Predicting

Reading -log_view

- Look out for this:

```
##############################################################
#                                                            #
#                        WARNING!!!                          #
#                                                            #
#    This code was compiled with a debugging option,         #
#    To get timing results run ./configure                   #
#    using --with-debugging=no, the performance will         #
#    be generally two or three times faster.                 #
#                                                            #
##############################################################
```

- Get a summary per stage
- Memory usage per stage (based on when it was allocated)
- Time, messages, reductions, balance, flops per event per stage
- Always send -log_view when asking performance questions on the mailing lists

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Profiling
Predicting

Reading `-log_view` II

Event	Count Max Ratio	Time (sec) Max Ratio	Flops Max Ratio	Mess	Avg len	Reduct	--- Global --- %T %F %M %L %R	--- Stage --- %T %F %M %L %R	Total Mflop/s

--- Event Stage 0: Main Stage

Event	Count Max Ratio	Time (sec) Max Ratio	Flops Max Ratio	Mess	Avg len	Reduct	%T %F %M %L %R	%T %F %M %L %R	Mflop/s
SNESSolve	1 1.0	3.6435e+01 1.0	6.75e+10 1.0	0.0e+00	0.0e+00	0.0e+00	100 100 0 0 0	100 100 0 0 0	1852
SNESFunctionEval	4 1.0	1.4375e-02 1.0	6.52e+06 1.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	454
SNESJacobianEval	3 1.0	7.3466e-02 1.0	0.00e+00 0.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	0
SNESLineSearch	3 1.0	1.7947e-02 1.0	1.38e+07 1.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	768
VecDot	3 1.0	4.9686e-04 1.0	8.89e+05 1.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	1790
VecMDot	5319 1.0	7.9103e+00 1.0	2.44e+10 1.0	0.0e+00	0.0e+00	0.0e+00	22 36 0 0 0	22 36 0 0 0	3079
VecNorm	5505 1.0	8.8808e-01 1.0	1.63e+09 1.0	0.0e+00	0.0e+00	0.0e+00	2 2 0 0 0	2 2 0 0 0	1838
VecScale	5498 1.0	5.7987e-01 1.0	8.15e+08 1.0	0.0e+00	0.0e+00	0.0e+00	2 1 0 0 0	2 1 0 0 0	1405
VecCopy	185 1.0	3.4122e-02 1.0	0.00e+00 0.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	0
VecSet	223 1.0	3.0877e-02 1.0	0.00e+00 0.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	0
VecAXPY	355 1.0	6.0618e-02 1.0	1.05e+08 1.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	1736
VecWAXPY	3 1.0	6.5303e-04 1.0	4.45e+05 1.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	681
VecMAXPY	5498 1.0	9.9785e+00 1.0	2.59e+10 1.0	0.0e+00	0.0e+00	0.0e+00	27 38 0 0 0	27 38 0 0 0	2599
VecScatterBegin	7 1.0	1.0543e-03 1.0	0.00e+00 0.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	0
VecReduceArith	6 1.0	9.5391e-04 1.0	1.78e+06 1.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	1865
VecReduceComm	3 1.0	2.8610e-06 1.0	0.00e+00 0.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	0
VecNormalize	5498 1.0	1.4746e+00 1.0	2.44e+09 1.0	0.0e+00	0.0e+00	0.0e+00	4 4 0 0 0	4 4 0 0 0	1658
MatMult	5498 1.0	6.2141e+00 1.0	7.32e+09 1.0	0.0e+00	0.0e+00	0.0e+00	17 11 0 0 0	17 11 0 0 0	1178
MatSolve	5498 1.0	1.0525e+01 1.0	7.32e+09 1.0	0.0e+00	0.0e+00	0.0e+00	29 11 0 0 0	29 11 0 0 0	695
MatLUFactorNum	3 1.0	2.2453e-02 1.0	4.83e+06 1.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	215
MatILUFactorSym	1 1.0	5.4770e-03 1.0	0.00e+00 0.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	0
MatAssemblyBegin	4 1.0	1.9073e-06 1.0	0.00e+00 0.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	0
MatAssemblyEnd	4 1.0	7.3540e-03 1.0	0.00e+00 0.0	0.0e+00	0.0e+00	0.0e+00	0 0 0 0 0	0 0 0 0 0	0

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Profiling
Predicting

Predicting resource usage - time

Made a little easier thanks to PETSc's extensive, minimally intrusive logging and profiling.

However, there are still many things that need to be observed to predict scaling in time

- Number of iterations may increase when problem is refined, which can lead to
  - Restarts (slows convergence rate greatly)
  - Loss of orthogonality (breaks convergence completely)
- Extra communication required for more stable algorithms:
  - Classical vs modified Gram–Schmidt
  - Increased iterations when only changing the number of processors

Introduction
PETSc data structures and classes
Solving problems
Scaling of solutions

Profiling
Predicting

Time scaling

Exercise 10: 2D Bratu scaling

- Source code: examples/snes/ex5.c.
- Run with -snes_monitor -ksp_monitor.
- Observe number of KSP iterations for da_refine factors of 3,4,5.
- Estimate KSP iteration count for a refinement factor of 6.
- How close were you?

Summary

- PETSc can give you a fully scriptable process for experimenting with the large scale solution of PDE models.
- Plenty of existing examples to use as a starting point for what you want to accomplish.
- Don't be afraid to ask for help/clarification: petsc-users@mcs.anl.gov
  - But at least show that you've made some effort to understand the manual and/or example codes!

Appendix

5 References

6 Personal installation

For Further Reading I

- PETSc website: http://www.mcs.anl.gov/petsc/
  - pdf manual
  - man pages
  - talks/workshop pdfs
- Jed Brown: http://jedbrown.org
  - talks/workshop pdfs
  - PETSc and general scientific computing topics

Summary
References
Personal installation

Obtaining
Unpacking
Configuring
Building

Appendix

Summary
References
Personal installation

Obtaining
Unpacking
Configuring
Building

Obtaining PETSc source code

- Specific releases, zipped tarballs
  http://www.mcs.anl.gov/petsc/download/
- Git repository [preferred]
  https://bitbucket.org/petsc/petsc/

Why is Git preferred?

- Public development repository, you can get *any* development version.
- All releases are just tags: eg. v3.7.6
- Easily rollback changes and releases.

Summary
References
Personal installation

Obtaining
Unpacking
Configuring
Building

Unpacking PETSc

Clone from repository:

```
git clone https://bitbucket.org/petsc/petsc.git
git checkout v3.7.6
```

OR

Unpack the archive:

```
tar xzf petsc.tar.gz
```

Summary
References
Personal installation

Obtaining
Unpacking
Configuring
Building

Configuring PETSc

- Set environment variable $PETSC_DIR to the installation root directory
- Choose a value for PETSC_ARCH (perhaps c-debug for a standard debug build)
- Run the configuration utility
  - $PETSC_DIR/configure [--help]
- May require additional dependencies
  - Read the output message if the configure fails, should tell you why, and suggest configure options to help

Summary
References
Personal installation

Obtaining
Unpacking
Configuring
Building

External libraries

External libraries can be downloaded during the configuration phase.

For example, MUMPS and SuperLU_dist can be obtained with

```
--download-mumps --download-scalapack
```

and

```
--download-superlu_dist --download-metis --download-parmetis
```

respectively.

Summary
References
Personal installation

Obtaining
Unpacking
Configuring
Building

Buidling PETSc

After successful configure, run

```
$ make
$ make test
$ make streams
```