# An Introduction to PETSc: The Portable, Extensible Toolkit for Scientific Computation

## A User's Overview

Jeff Borggaard

Virginia Tech

17 January 2006

# Disclaimer

PETSc was developed at Argonne National Laboratory by Satish Balay, Kris Buschelman, Victor Eijkhout, William Gropp, Dinesh Kaushik, Matthew Knepley, Lois Curfman McInnes, Barry Smith and Hong Zhang.
tt http://www.mcs.anl.gov/petsc

Along with Traian Iliescu and Alexey Miroshnikov, I have *used* PETSc to build a parallel 3D finite element flow solver for evaluating LES models of realistic flows.

# What is PETSc?

PETSc, the Portable, Extensible Toolkit for Scientific Computation:

- Data structures
- Functions

Built upon blas, linpack and MPI.

Built specifically to shorten the development time of parallel scientific application software. Especially on distributed memory architectures requiring message passing.

# Why PETSc?

Why PETSc?

# Functional

- Data structures
  - parallel vectors (sequential, MPI)
  - sparse matrices (compressed sparse row, etc.)
  - distributed arrays
  - index sets
- Functions
  - Krylov subspace methods (GMRES, CG, CGS, etc.)
  - preconditioners
  - nonlinear solvers (line search, trust-region)
  - time steppers (forward/backward Euler, pseudo-time stepping)
- Profiling
  - `-log_summary`

# Portable

- Freely available (`www.mcs.anl.gov/petsc`)
- Very responsive developers (`petsc-maint@mcs.anl.gov`)
- Avaliable for C, C++, FORTRAN 77/90
- Ported to a wide variety of platforms

Distributed Memory

- HP
- IBM
- SGI
- Sun
- Mac OS X
- PCs: Linux and Wintel

Shared Memory

- Cray T3E
- HP 9000
- IBM SP
- SGI Origin
- Sun Enterprise

# Extensible

Has been coupled numerous packages, including

- ADIC: Automatic Differentiation in C
- Chaco, Jostle, MeTiS: Graph partitioning packages
- Mathematica
- MATLAB
- Trilinos: Multilevel preconditioning package

# But . . .

It has a (reasonbly) steep learning curve

# How to use PETSc

Set up environment variables. On `phoenix.scs.fsu.edu`:

- `setenv PETSC_DIR /usr/local/petsc`
- `setenv PETSC_ARCH linux-gnu`

Add the following line to your `makefile`:

- include `${PETSC_DIR}/bmake/common/base`

then link with eg. `${PETSC_SYS_LIB}` or `${PETSC_KSP_LIB}`

Run just as any MPI job

- `mpirun -np 4 petsc-exec (options)`

inside of a condor script.

# Hello World! from MPI and PETSc

```
// MPI hello world
#include <stdio.h>
#include "mpi.h"

int main( int argc,
          char *argv[] ) {
  MPI_Init( &argc,
            &argv);
  printf( "Hello World\n");
  MPI_Finalize();
  return 0;
}
```

```
// PETSc hello world
#include "petsc.h"

int main( int argc,
          char *argv[] ) {
  PetscInitialize( &argc,
                   &argv,
                   PETSC_NULL,
                   PETSC_NULL);
  PetscPrintf( PETSC_COMM_WORLD,
               "Hello World\n");
  PetscFinalize();
  return 0;
}
```

# Hello World! from MPI and PETSc (2)

```
node001% mpirun -np 4 hello_mpi
Hello World
Hello World
Hello World
Hello World
node001%
```

```
node001% mpirun -np 4 hello_petsc
Hello World
node001%
```

# Hello World! from MPI and PETSc (3)

```
// The MPI hello world program (modified to print once)
#include <stdio.h>
#include "mpi.h"

int main( int argc, char *argv[] ) {
  int rank;

  MPI_Init( &argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if (rank == 0) {
    printf( "Hello World\n");
  }
  MPI_Finalize();
  return 0;
}
```

# First Example

We are going to limit our discussion to four components from PETSc,

- Vectors (`Vec*`)
- Matrices (`Mat*`)
- Krylov Subspace Methods (`KSP*`)
- Preconditioners (`PC*`)

and use them to solve a linear PDE.

These are the essential building blocks of PETSc.

# PETSc Vectors

There are essentially two types of PETSc Vectors:

- Sequential (local copy on each processor)
  `VecCreateSeq(PETSC_COMM_SELF, int m, Vec *x);`

and

- Parallel (large vector stored over all processors)
  `VecCreateMPI(PETSC_COMM_WORLD, int m, int M, Vec *x);`

a third, `VecCreateShared()`, could be used on a shared memory machine.

Either `m` or `M` above could be set to `PETSC_DECIDE` to have PETSc determine it.

# PETSc Vectors (2)

Vectors can be filled using a number of functions. These include

- Setting the entire vector to one value (often 0)
  `VecSet(Vec x, PetscScalar value);`

or to set individual components

- either *insert* values
  `VecSetValues(Vec x, int n, int *indices, PetscScalar *values, INSERT_VALUES);`
- or *add* values
  `VecSetValues(Vec x, int n, int *indices, PetscScalar *values, ADD_VALUES);`

Once all components have been set with `VecSetValues()`, the following functions must be called

- `VecAssemblyBegin(Vec x);`
- `VecAssemblyEnd(Vec x);`

# PETSc Vectors (3)

A partial list of functions on PETSc Vectors follows:

| | |
|---|---|
| `VecAXPY(Vec y, PetscScalar a, Vec x);` | `y = y + a*x` |
| `VecScale(Vec x, PetscScalar a);` | `x = a*x` |
| `VecDot(Vec x, Vec y, PetscScalar *r);` | `r = x̄'*y` |
| `VecNorm(Vec x, NormType type, double *r);` | $r = \|x\|_{type}$ |
| `VecSum(Vec x, PetscScalar *r);` | $r = \sum x_i$ |
| `VecCopy(Vec x, Vec y);` | `y = x` |
| `VecPointwiseMult(Vec w, Vec x, Vec y);` | $w_i = x_i * y_i$ |
| `VecMax(Vec x, int *index, double *r);` | $r = \max x_i$ |
| `VecSet(Vec x, PetscScalar a);` | $x_i$ = a |

Some operations are *collective* while others are not.

# PETSc Matrices

As with PETSc Vectors, there are many options for matrices:

- Those that are associated with individual processors
  - `MatCreateSeqAIJ`
  - `MatCreateSeqSBAIJ`
  - `MatCreateSeqDense`
  - `MatCreateSeqBDiag`

and

- those associated with an MPI communicator
  - `MatCreateMPIAIJ`
  - `MatCreateMPISBAIJ`
  - `MatCreateMPIDense`
  - `MatCreateMPIBDiag`

# PETSc Matrices (2)

The preallocation of storage for sparse matrix data structures is essential for good performance.

Thus, we want a good estimate of the number of "diagonal" and "off-diagonal" nonzero entries for each row.

One useful tool is

```
MatSetOption(Mat A, MAT_NO_NEW_NONZERO_LOCATIONS);
```

when matrices keep the same nonzero pattern (in many time-dependent problems, nonlinear problems).

# PETSc Matrices (3)

As with Vectors, `MatSetValues` is one tool for creating matrix entries.

- Again, either *insert* values
  ```
  MatSetValues(Mat A, int m, int *rowidx, int n, int
  *colidx, PetscScalar *values, INSERT_VALUES);
  ```
- or *add* values
  ```
  MatSetValues(Mat A, int m, int *rowidx, int n, int
  *colidx, PetscScalar *values, ADD_VALUES);
  ```

Once all components have been set with `MatSetValues()`, the following functions must be called

- `MatAssemblyBegin(Mat A, MAT_FINAL_ASSEMBLY);`
- `MatAssemblyEnd(Mat A, MAT_FINAL_ASSEMBLY);`

# PETSc Matrices (4)

A partial list of functions on PETSc Matrices follows:

| | |
|---|---|
| `MatAXPY(Mat Y, Mat X, a, MatStructure);` | `Y = Y + a*X` |
| `MatConvert(Mat A, MatType type, Mat B);` | `B = A` |
| `MatMult(Mat A, Vec x, Vec y);` | `y = A*x` |
| `MatNorm(Mat A, NormType type, double *r);` | `r =` $\|\mathtt{A}\|_{type}$ |
| `MatScale(Mat X, PetscScalar a);` | `X = a*X` |
| `MatTranspose(Mat A, Mat *B);` | `B =` $\mathtt{A}^T$ |
| `MatGetDiagonal(Mat A, Vec x);` | `x = diag(A)` |
| `MatZeroEntries(Mat A);` | `A = 0` |
| `MatShift(Mat X, PetscScalar a);` | `X = X + a*I` |

# PETSc Matrices (5)

Functions known as PETSc Viewers can be used to inspect matrices

- `MatView(Mat M, PETSC_VIEWER_STDOUT_WORLD);`

and

- `MatView(Mat M, PETSC_VIEWER_DRAW_WORLD);`

# PETSc KSP: Linear System Solvers

Krylov subspace methods and preconditioners are the essential tool for solving *sparse* linear systems of the form

$$Ax = b.$$

This is setup with the commands

- `KSPCreate(PETSC_COMM_WORLD, KSP *ksp);`
- `KSPSetOperators(KSP ksp, Mat Amat, Mat Pmat, MatStructure flag);`

`Pmat` is the matrix used to create the preconditioner (usually `Amat`).

# PETSc KSP: Linear System Solvers (2)

Many built-in Krylov subspace methods are available, including

| | |
|---|---|
| Richardson | `KSPRICHARDSON` |
| Chebychev | `KSPCHEBYCHEV` |
| Conjugate Gradient | `KSPCG` |
| BiConjugate Gradient | `KSPBCG` |
| Generalized Minimal Residual | `KSPGMRES` |
| BiCGSTAB | `KSPBCGS` |
| Conjugate Gradient Squared | `KSPCGS` |
| Transpose-Free Quasi-Minimal Residual | `KSPTFQMR` or `KSPTCQMR` |

# PETSc KSP: Linear System Solvers (3)

Either left or right preconditioning is possible (all of the built in preconditioners are left preconditioners by default).

$$\left(M_L^{-1} A M_R^{-1}\right)\left(M_R x\right) = \left(M_L^{-1} b\right)$$

The preconditioner is created using commands

- `KSPGetPC(KSP ksp, PC *pc);`
- `PCSetType(PC pc, PCtype option);`

where built in options include

| | |
|---|---|
| Jacobi | `PCJACOBI` |
| Block Jacobi | `PCBJACOBI` |
| SOR | `PCSOR` |
| SOR with Eisenstat trick | `PCEISENSTAT` |
| Incomplete Cholesky | `PCICC` |
| Incomplete LU | `PCILU` |

# PETSc KSP: Linear System Solvers (4)

Specify Krylov subspace tolerances:

```
KSPSetTolerances(KSP ksp, double rtol, double atol,
                 double dtol, int maxiter);
```

where

$$\|r_k\|_2 < \max(\texttt{rtol}\|b\|_2, \texttt{atol})$$

$k <$ `maxiter` corresponds to convergence, and

$$\|r_k\|_2 > \texttt{dtol}\|b\|_2$$

or $k =$ `maxiter` is divergence.

`PETSC_DEFAULT` can be used for any of these and can be overwritten at the command line with

```
KSPSetFromOptions(ksp);
```

# First Example ex1.c

Solve

$$-u_{xx} = 0$$

on $(0,1)$ with $u(0) = 1$ and $u(1) = 1$ using finite differences.
Then, partition $[0,1]$ with points $x_{-1}, \ldots, x_n$ with

$$x_{i+1} - x_i = \Delta x = \frac{1}{n+1}$$

Thus, we set

$$-u_x x(x_i) \approx \frac{-u_{i-1} + 2u_i - u_{i+1}}{\Delta x} = 0$$

for $i = 0, \ldots, n-1$ (knowing $u_{i-1} = 1 = u_n$).

# First Example `ex1.c`

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ & & \ddots & \ddots & \ddots & & & \\ & & & & \ddots & \ddots & \ddots & \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} u_0 \\ u_2 \\ u_3 \\ \vdots \\ \vdots \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ 1 \end{bmatrix}.$$

# First Example ex1.c

```
#include "petscksp.h"

int main(int argc,char **args)
{
  Vec            u, b, exact;   // approx sol., RHS
  Mat            A;             // linear system matrix
  KSP            ksp;           // linear solver context
  PC             pc;            // preconditioner context
  PetscErrorCode ierr;
  PetscInt       i, n = 10, row[2], col[3], its;
  PetscReal      norm;
  PetscScalar    neg_one = -1.0, one = 1.0, value[3];

  PetscInitialize(&argc,&args,PETSC_NULL,PETSC_NULL);
  PetscOptionsGetInt(PETSC_NULL,"-n",&n,PETSC_NULL);
```

# First Example ex1.c

```
VecCreateSeq(PETSC_COMM_SELF, n, &u);
VecDuplicate(u, &b);
VecDuplicate(u, &exact);

MatCreateSeq(PETSC_COMM_SELF, n, n, 0, PETSC_NULL, &A);

i = 0; col[0] = 0; col[1] = 1;
value[0] = 2.0; value[1] = -1.0;

MatSetValues(A,1,&i,2,col,value,INSERT_VALUES);
value[0] = -1.0; value[1] = 2.0; value[2] = -1.0;
for (i=1; i<n-1; i++) {
  col[0] = i-1; col[1] = i; col[2] = i+1;
  MatSetValues(A,1,&i,3,col,value,INSERT_VALUES);
}
```

# First Example ex1.c

```
i = n - 1; col[0] = n - 2; col[1] = n - 1;
value[0] = -1.0; value[1] = 2.0;
MatSetValues(A,1,&i,2,col,value,INSERT_VALUES);

ierr = MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr
ierr = MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);CHKERRQ(ierr);

row[0] = 0; value[0] = 1;
row[1] = n-1; value[1] = 1;
VecSetValues(b,2,row,value,INSERT_VALUES);

VecAssemblyBegin(b);
VecAssemblyEnd(b);

VecSet(exact,one);
```

# First Example ex1.c

```
KSPCreate(PETSC_COMM_WORLD,&ksp);
KSPSetOperators(ksp,A,A,DIFFERENT_NONZERO_PATTERN);
KSPGetPC(ksp,&pc);
PCSetType(pc,PCJACOBI);
KSPSetTolerances(ksp,1.e-7,PETSC_DEFAULT,
                 PETSC_DEFAULT,PETSC_DEFAULT);
KSPSolve(ksp,b,u);

VecAXPY(u,neg_one,exact);
VecNorm(u,NORM_2,&norm);

KSPGetIterationNumber(ksp,&its);
PetscPrintf(PETSC_COMM_WORLD,
            "Norm of error %A, Iterations %D\n",
            norm,its);
```

# First Example ex1.c

```
  VecDestroy(exact);
  VecDestroy(b);
  VecDestroy(u);

  MatDestroy(A);

  KSPDestroy(ksp);

  PetscFinalize();
  return 0;
}
```

Typing `mpirun -np 1 ex1` leads to the output:

```
Norm of error < 1.e-12, Iterations 5
```

# Second Example ex23.c

```
#include "petscksp.h"

int main(int argc,char **args)
{
  Vec            u, b, exact;   // approx sol., RHS, exact
  Mat            A;             // linear system matrix
  KSP            ksp;           // linear solver context
  PC             pc;            // preconditioner context
  PetscInt       i, n = 10000, row[1], col[3], its,
                 rstart, rend, nlocal;
  PetscReal      norm;          // norm of solution error
  PetscScalar    neg_one = -1.0, one = 1.0, value[3];

  PetscInitialize(&argc,&args,PETSC_NULL,PETSC_NULL);
  PetscOptionsGetInt(PETSC_NULL,"-n",&n,PETSC_NULL);
```

# Second Example ex23.c

```
PreLoadBegin(PETSC_TRUE,"beginning of code");

VecCreate(PETSC_COMM_WORLD,&u);
VecSetSizes(u,PETSC_DECIDE,n);
VecSetFromOptions(u);
VecDuplicate(u,&b);
VecDuplicate(u,&exact);

VecGetOwnershipRange(u,&rstart,&rend);
VecGetLocalSize(u,&nlocal);

MatCreate(PETSC_COMM_WORLD,&A);
MatSetSizes(A,nlocal,nlocal,n,n);
MatSetFromOptions(A);
```

# Second Example ex23.c

```
if (rstart == 0) {
  rstart = 1;
  i = 0; col[0] = 0; col[1] = 1; value[0] = 2.0; value[1]
  MatSetValues(A,1,&i,2,col,value,INSERT_VALUES);

  row[0] = 0;  value[0] = 1.0;
  VecSetValues(b,1,row,value,INSERT_VALUES);
}
if (rend == n) {
  rend = n-1;
  i = n-1; col[0] = n-2; col[1] = n-1; value[0] = -1.0; va
  MatSetValues(A,1,&i,2,col,value,INSERT_VALUES);

  row[0] = n-1;  value[0] = 1.0;
  VecSetValues(b,1,row,value,INSERT_VALUES);
}
```

# Second Example ex23.c

```
value[0] = -1.0; value[1] = 2.0; value[2] = -1.0;
for (i=rstart; i<rend; i++) {
  col[0] = i-1; col[1] = i; col[2] = i+1;
  MatSetValues(A,1,&i,3,col,value,INSERT_VALUES);
}

MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);

VecAssemblyBegin(b);
VecAssemblyEnd(b);

VecSet(exact,one);
```

# Second Example ex23.c

```
KSPCreate(PETSC_COMM_WORLD,&ksp);
KSPSetOperators(ksp,A,A,DIFFERENT_NONZERO_PATTERN);

KSPGetPC(ksp,&pc);
PCSetType(pc,PCJACOBI);
KSPSetTolerances(ksp,1.e-7,PETSC_DEFAULT,PETSC_DEFAULT,PE
KSPSetFromOptions(ksp);
KSPSolve(ksp,b,u);
KSPView(ksp,PETSC_VIEWER_STDOUT_WORLD);
VecAXPY(u,neg_one,exact);
VecNorm(u,NORM_2,&norm);
KSPGetIterationNumber(ksp,&its);

PetscPrintf(PETSC_COMM_WORLD,
            "Norm of error %A, Iterations %D\n",
            norm,its);
```

# Second Example ex23.c

```
    VecDestroy(exact);
    VecDestroy(b);
    VecDestroy(u);
    MatDestroy(A);
    KSPDestroy(ksp);

    PreLoadEnd();
    PetscFinalize();
    return 0;
}
```

Typing `mpirun -np 1 ex23` or `mpirun -np 4 ex23` each lead to the same output:

```
Norm of error < 1.e-12, Iterations 5
```

# ViTLES - Virginia Tech Large-Eddy Simulator

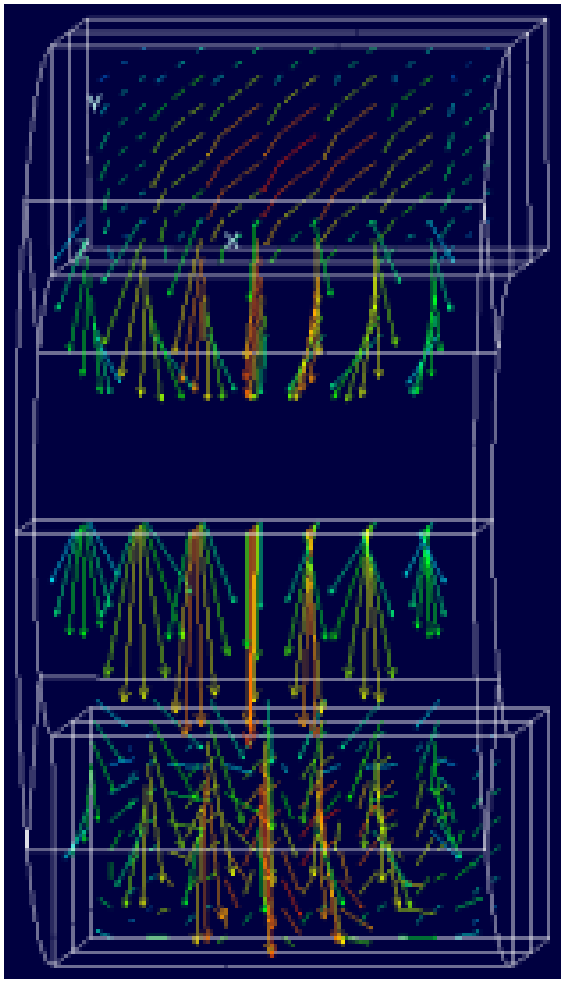
Developed with Traian Iliescu, uses PETSc to solve 2D/3D LES model equations.

Runs on System X: `www.tcf.vt.edu`

# ViTLES - Virginia Tech Large-Eddy Simulator


- Subgrid scale models
  - dynamic
  - deconvolution
  - differential filters
- Boundary condition models
  - Approximate deconvolution boundary conditions
- Reduced-order modeling
- Control

# ViTLES - Virginia Tech Large-Eddy Simulator



Finite elements: requires different PETSc functions

```
VecScatterCreate(g_ctx->crt_N_iter,
        l_ctx->from_is,
        l_ctx->g_vec,
        l_ctx->to_is, &scatter);
VecScatterBegin(g_ctx->crt_N_iter,
        l_ctx->g_vec,
        INSERT_VALUES,
        SCATTER_FORWARD,
        scatter);
VecScatterEnd(g_ctx->crt_N_iter,
        l_ctx->g_vec,
        INSERT_VALUES,
        SCATTER_FORWARD,
        scatter);
```