

# Introduction to PETSc

Mohamad M. Nasr-Azadani

Mechanical Engineering Department,  
University of California at Santa Barbara



UC SANTA BARBARA  
**engineering**

# What is PETSc?

- Definition: Portable Extendable Toolkit for Scientific Computations
- It essentially contains: Parallel PDE linear/nonlinear algebraic tools and **solvers**
- Hides **99%** of the need to run MPI commands (parallelism)
- Interfaces to many many other numerical packages
- It is extremely well-supported (an ongoing project)

# What is PETSc?

The screenshot shows a Gmail interface. On the left is a sidebar with navigation links: COMPOSE, Inbox (7), Important, Sent Mail, Circles, and [imap]/Trash. Below these is a search bar and a list of contacts: Diana, Edward, Saiph, and Scott. The main area displays an email from Mohamad M. Nasr-Azadani, dated 12/18/10. The email body contains a message from Barry Smith, dated 12/19/10, which is the focus of the slide. The email text discusses a code change in PETSc, specifically related to the compressed row multiple (PETSC\_TRUE vs PETSC\_FALSE) and a reassembly error. It includes code snippets for editing a file and running 'make'.

**Gmail**

COMPOSE

Inbox (7)  
Important  
Sent Mail  
Circles  
[imap]/Trash

Resume anyone? <http://i>

Search people...

Diana  
The Strokes - Aut...

Edward

Saiph

Scott  
it is me

**Mohamad M. Nasr-Azadani** 12/18/10

Ok, Sometimes it is hard to move on Barry! I was perfectly fine with the olde...

**Barry Smith** <bsmith@mcs.anl.gov> 12/19/10

to petsc-maint, me

Please edit `src/mat/impls/aij/mpi/mpiaij.c` and locate the line

```
((Mat_SeqAIJ *)aij->B->data)->compressedrow.use = PETSC_TRUE;
```

replace it with

```
((Mat_SeqAIJ *)aij->B->data)->compressedrow.use = PETSC_FALSE;
```

then run `make` in that directory.

There is an error with the compressed row multiple if the matrix gets "reassembled" which is what happens in your case when you added that extra value later. Sorry it took us so long to determine the problem.

This is also fixed in `petsc-dev`

Barry

> <DAMat.c>

# What is PETSc?

Gmail

COMPOSE

Inbox (7)  
Important  
Sent Mail  
Circles  
[imap]/Trash  
Resume anyone? <http://i>  
Search people...

Diana  
The Strokes - Aut...  
Edward  
Saiph  
Scott  
it is me

Move to Inbox

Mohamad M. Nasr-Azadani  
Ok, Sometimes it is hard to move on Barry! I was perfectly fine with the olde...

12/18/10

Barry Smith <bsmith@mcs.anl.gov>  
to petsc-maint, me

Please edit `src/mat/impls/aij/mpi/mpiaij.c` and locate the line

```
((Mat_SeqAIJ *)aij->B->data)->compressedrow.use = PETSC_TRUE;
```

replace it with

```
((Mat_SeqAIJ *)aij->B->data)->compressedrow.use = PETSC_FALSE;
```

then run make in that directory.

There is an error with the compressed row multiple if the matrix gets "reassembled" which is what happens in your case when you added that extra value later. Sorry it took us so long to determine the problem.

This is also fixed in `petsc-dev`

Barry  
> <DAMat.c>

# What does PETSc target?

- Serial and Parallel
- Linear and nonlinear
- Finite difference and finite element
- Structured and unstructured

# What does PETSc NOT target?

- Sophisticated visualization
- Discretize PDE's or ODE's
- Load balancing
- Grid generation, etc

# PETSc can be used with:

- C/C++ (originally written in C)
- Fortran
- Some: Python and MATLAB interface

# PETSc's basic objects

- Mat (matrix)
- Vec (vector)
- KSP (solver), [PC (preconditioners)]
- SNES (nonlinear solver)
- TS (time stepping)

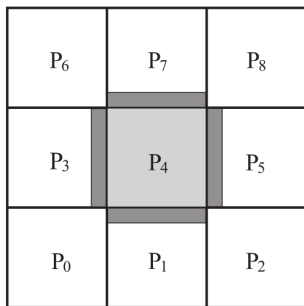


# Variable declarations, C

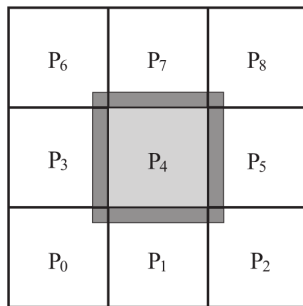
```
int main(int argc, char **argv) {  
    KSP solver;  
    Mat A;  
    Vec x, y;  
    PetscInt size = 10;  
    PetscScalar value = 1.0;  
    double extra = value*2.0;  
    .  
    .  
}
```

# Distributed arrays (DMDA)

- Very powerful tool: Structured orthogonal grids (1D, 2D, and 3D)
- Allows you to use  $(i,j,k)$  index in data arrays
- Hides the complexity associated with updating ghost nodes



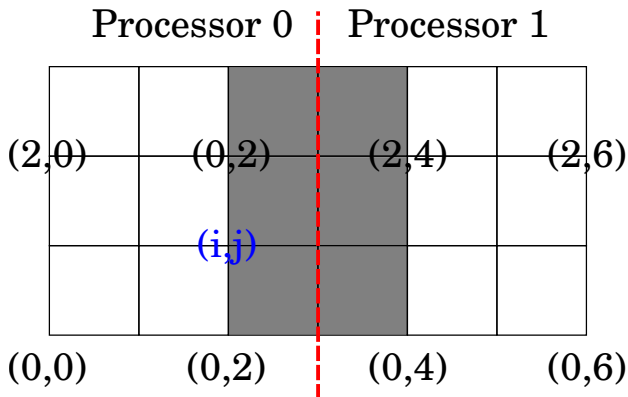
(a) Star stencil



(b) Box stencil

# Distributed arrays (DMDA)

- Major benefit: User can still employ data arrays in  $(i,j,k)$  format using **global indexing** rules
- Hides the complexity of ghost nodes and communication between processors



# Distributed arrays (DMDA)

- Tell PETSc about your data array size, boundary condition, ...

```
DMDACreate2d(some arguments, DM *dm);
```

- Automatically defines the matrix structure

```
DMCreateMatrix(DM dm, const MatType mtype, Mat *mat);
```

- Automatically creates vectors (data arrays) associated with the parallel layout

```
DMCreateGlobalVector(DM dm, Vec *vec);  
DMCreateLocalVector(DM dm, Vec *vec);
```

# DMDA Matrix

- Use this command to insert values at any given  $(i,j,k)$

```
MatSetValuesStencil(Mat mat, some arguments);
```

# DMDA Vector

- Use this command to access (read/write) vector data in (i,j,k) format

```
DMDAVecGetArray(DM da, Vec vec, void *array)
```

- Once operations are done, CALL this function to restore the vector

```
DMDAVecRestoreArray(DM da, Vec vec, void *array)
```

# Example: Poisson equation

- The Poisson equation of the form

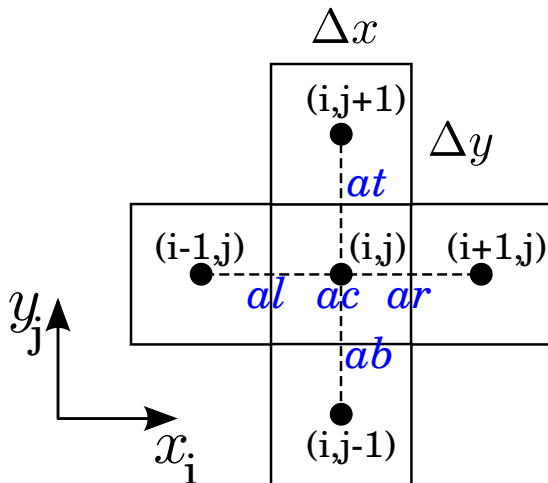
$$\nabla^2 q = f(x, y) \quad 0 \leq x \leq 1, \quad 0 \leq y \leq 1.$$

- Discretization on a uniform grid via  $2^{nd}$ -order finite difference scheme

$$\frac{q_{i+1,j} - 2q_{i,j} + q_{i-1,j}}{\Delta x^2} + \frac{q_{i,j+1} - 2q_{i,j} + q_{i,j-1}}{\Delta y^2} = f_{i,j}$$

# Example: Discretization

- Connectivity and 5-point numerical stencil





## Example: Matrix setup

- Connectivity and 5-point numerical stencil

```
MatStencils row_index, col_indices[5];
```

```
PetscScalar coefficients[5];
```

```
/* center node index (i,j) and coefficient, ac */
```

```
col_indices[0].i = i;
```

```
col_indices[0].j = j;
```

```
coefficients[0] = ac;
```

```
/* right node index (i+1,j) and coefficient, ar */
```

```
col_indices[1].i = i+1;
```

```
col_indices[1].j = j;
```

```
coefficients[1] = ar;
```

## Example: Matrix setup

```
/* left node index (i-1,j) and coefficient, a1 */
```

```
col_indices[2].i = i-1;
```

```
col_indices[2].j = j;
```

```
coefficients[2] = a1;
```

```
.
```

```
.
```

```
/* And of course the row index is also needed */
```

```
row_index.i = i; row_index.j = j;
```

```
/* Now, insert the coefficients */
```

```
MatSetValuesStencil(A, 1, &row_index, 5, col_indices, coefficients,  
INSERT_VALUES);
```

# Example: Poisson equation

- The Poisson equation of the form

$$\nabla^2 q = f(x, y) \quad 0 \leq x \leq 1, \quad 0 \leq y \leq 1.$$

- Discretization on a uniform grid via  $2^{nd}$ -order finite difference scheme

$$\frac{q_{i+1,j} - 2q_{i,j} + q_{i-1,j}}{\Delta x^2} + \frac{q_{i,j+1} - 2q_{i,j} + q_{i,j-1}}{\Delta y^2} = f_{i,j}$$

## Example: Right-hand-side setup

```
PetscScalar **f;  
Vec b;  
DM da;  
.  
.  
DMDAVecGetArray(da, b, &f);  
/* Insert the rhs_value for current index at (i,j) */  
f[j][i] = rhs_func(x,y);  
.  
.  
DMDAVecGetArray(da, b, &f);
```

## Example: KSP solver

- PETSc provides ksp (iterative) solvers with various preconditioners.
- Black box solvers!
- Pass matrix and right-hand-side vectors

```
KSP solver;  
Mat M;  
Vec b, x;  
    . /* Setup LHS matrix (M) and RHS (b) vector */  
    .  
  
KSPCreate(PETSC_COMM_WORLD, &solver);  
KSPSetOperators(solver, M, M, SAME_PRECONDITIONER);  
KSPSetFromOptions(solver);  
KSPSolve(solver, b, x);
```

# PETSc pseudo code

```
static char help[] = "Hello world PETSc!";
#include "petsc.h"
int main(int argc, char **argv) {

    PetscErrorCode ierr;

    /* Always start your program with this command */
    ierr = PetscInitialize(&argc, &argv, (char*)0, help);
    CHKERRQ(ierr); /* Error checking for PETSc functions */

    .

    /*PETSc variables and function calls */.

    .

    ierr = PetscFinalize();
    return 0;
}
```

# Example: Hello world

- Using **PetscPrintf()** and **PetscSynchronizedPrintf()**
- Let's run code ex0.c

```
make ex0  
./ex0
```

- Now, let's run it on 4 processors

```
mpiexec -np 4 ./ex0
```

## Example: 2-D Poisson equation

- After discretization, the resulting linear system is to be solved by an iterative solver.

$$[A] \{q\} = \{b\}$$

[A]: Penta-diagonal symmetric matrix

- We need:

Distributed array context **DMDA DA2d;**

LHS matrix **Mat LHS\_mat;**

RHS vector **Vec RHS\_vec\_g;**

Solution vector **Vec Sol\_vec;**

- Pass the setup matrix and RHS vector to solver

KSP iterative solver **KSP solver;**



# Example: 2-D Poisson equation

- Pass input parameters ( $N_x, N_y$ ) and boundary conditions (bc: 1 for **Neumann**, 2 for **Dirichlet**) via command line arguments

```
make ex1  
./ex1 -Nx 10 -Ny 10 -bc 1
```

- Now, let's run it on 4 processors

```
mpiexec -np 4 ./ex1 -Nx 10 -Ny 10 -bc 1
```

- Compare:
  - Number of iterations used for convergence
  - Errors (E1, and E2)

# Find more information

- Official website: `http://www.mcs.anl.gov/petsc/`
- Email lists:
  - `petsc-users@mcs.anl.gov`
  - `petsc-maint@mcs.anl.gov`
- Or email me: `mmnasr@engineering.ucsb.edu`

