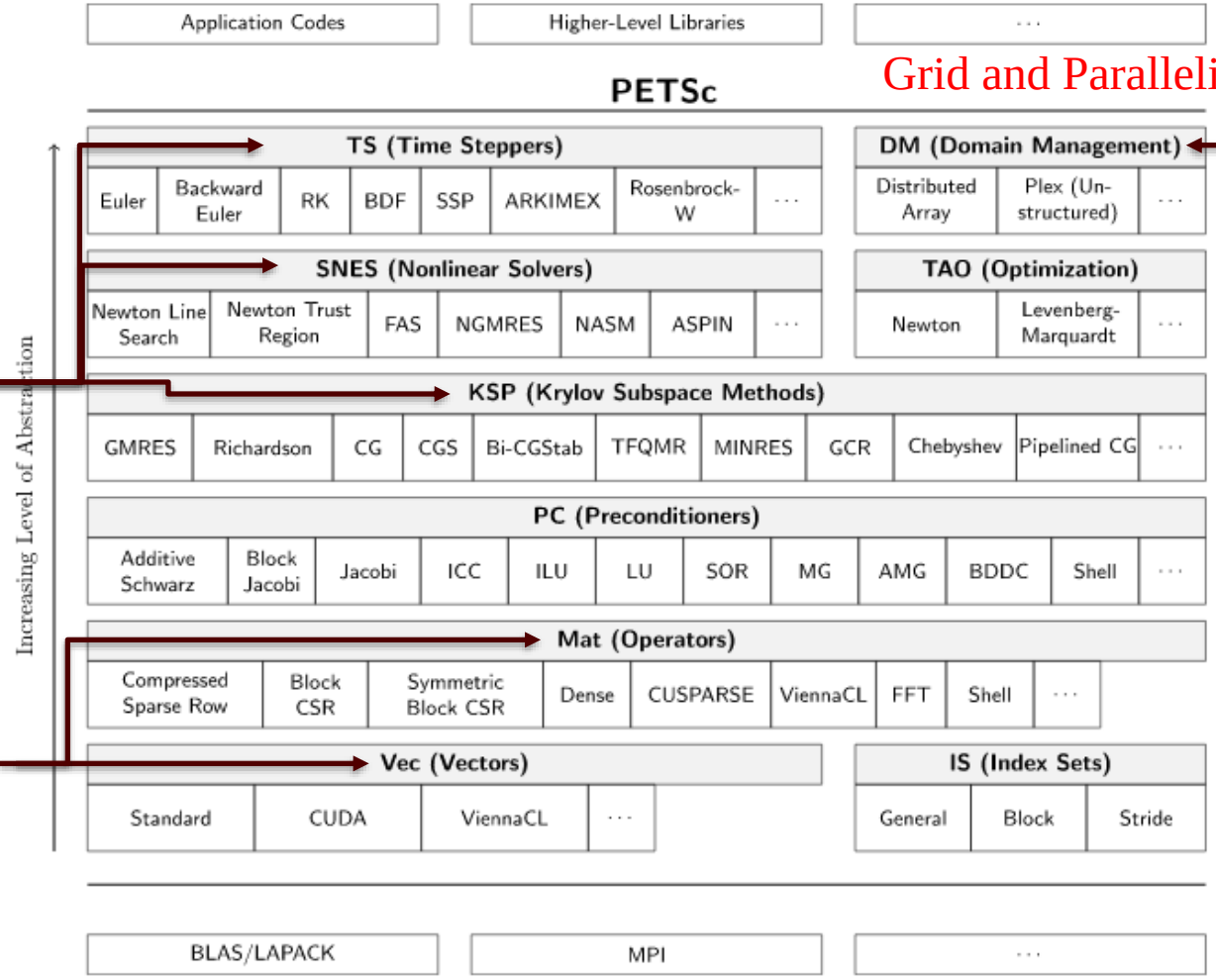
Introduction to PETSc Hands-on workshop

Iman Borazjani
Mechanical Engineering Department
TEXAS A&M UNIVERSITY

Introduction: PETSc in a nutshell

https://petsc.org/release/overview/nutshell/



PETSc on HPRC

- Note: you need to have a basic account at HPRC.
- Login to hprc portal: https://portal.hprc.tamu.edu/
- Click on Clusters tab >_grace Shell Access
- Alternatively, you can use ssh apps such as Putty or MobaXterm

- After successful login:
- At the prompt $ type: module spider petsc
- The available version will be shown.
- Load this one: module load PETSc/3.12.4-Python-3.8.2
- Note: some other modules might need to be loaded before PETSc – see: module spider PETSc/3.12.4-Python-3.8.2
- Specifically, for PETSc/3.12.4-Python-3.8.2, load only the following:

```
module load iccifort/2020.1.217
module load impi/2019.7.217
module load PETSc/3.12.4-Python-3.8.2
```

- Now that PETSc is loaded, you can write your first code and compile it.
- Note you need to have a makefile to easily compile PETSc codes (similar to C/C++)

PETSc makefile

```
FLAGS            =
FFLAGS           =
CPPFLAGS         =
FPPFLAGS         =
LOCDIR           =
DIRS             = network
EXAMPLESC        = ex1.c ex2.c ex3.c ex5.c ex6.c ex7.c ex8.c ex9.c \
                   ex10.c ex11.c ex12.c ex13.c ex15.c ex16.c ex18.c ex23.c \
                   ex25.c ex27.c ex28.c ex29.c ex30.c ex32.c ex34.c \
                   ex41.c ex42.c ex43.c \
                   ex45.c ex46.c  ex49.c ex50.c ex51.c ex52.c ex53.c \
                   ex54.c ex55.c ex56.c ex58.c ex62.c ex63.cxx ex64.c ex65.c ex66.c ex67.c ex68.c ex69.c ex70.c ex72.c ex100.c
EXAMPLESF        = ex1f.F90 ex2f.F90 ex6f.F90 ex11f.F90 ex13f90.F90 ex14f.F90 ex15f.F90 ex21f.F90 ex22f.F90 ex44f.F90 ex45f.F90 \
                   ex52f.F90 ex54f.F90 ex61f.F90 ex100f.F90
MANSEC           = KSP
CLEANFILES       = rhs.vtk solution.vtk
NP               = 1
LIBBASE = libpetscmat

include ${PETSC_DIR}/lib/petsc/conf/variables
include ${PETSC_DIR}/lib/petsc/conf/rules

ex50: ex50.o
                -${CLINKER} -o ex50 ex50.o ${PETSC_LIB} ${LIBS}

# testex100: ex100.PETSc
#                -@if [ "${PETSC_WITH_BATCH}" != "" ]; then \
#           echo "Running with batch filesystem; to test run src/ksp/ksp/examples/tutorials/ex100 with" ; \
#           echo "your systems batch system"; \
#         elif [ "${MPIEXEC}" = "/bin/false" ]; then \
#           echo "*mpiexec not found*. Please run src/ksp/ksp/examples/tutorials/ex100 manually"; \
#              elif [ -f ex100 ]; then \
#           ${MPIEXEC} -n 1 ./ex100 -test  > ex100_1.tmp 2>&1; \
#                if (${DIFF} output/ex100_1.testout ex100_1.tmp > /dev/null 2>&1) then \
#           echo "C/C++ example src/ksp/ksp/examples/tutorials/ex100 run successfully with 1 MPI process"; \
#                else echo "Possible error running C/C++ src/ksp/ksp/examples/tutorials/ex100 with 1 MPI process"; \
#           echo "See http://www.mcs.anl.gov/petsc/documentation/faq.html";\
#           cat ex100_1.tmp; fi; \
#         ${RM} -f ex100_1.tmp; fi

include ${PETSC_DIR}/lib/petsc/conf/test
```

Running PETSc Examples

- https://petsc.org/release/tutorials/handson/#example-1-linear-poisson-equation-on-a-2d-grid

- Copy examples and makefile from PETSc installation directory:

```
cp $PETSC_DIR/share/petsc/examples/src/ksp/ksp/examples/tutorials/ex50.c .
cp $PETSC_DIR/share/petsc/examples/src/ksp/ksp/examples/tutorials/makefile .
```

- Compile ex50: make ex50

- Run the compiled ex50 on a 3x3 mesh:

```
mpiexec -n 1 ./ex50  -da_grid_x 4 -da_grid_y 4 -mat_view
```

Writing the first PETSc code

- Open an editor: vi or emacs helloworld.c
- Type:

```
#include  <math.h>
#include  <petscvec.h>

static char help[] = “Hello World PETSc\n";

int main(int argc, char **argv) {
  PetscInitialize(&argc, &argv, (char*)0, help);
  PetscPrintf(PETSC_COMM_SELF, "Hello World!!\n");
  PetscFinalize();
  return(0);
}
```

Running the Hello World

- Add this to your makefile:

```
hello: helloworld.o
	-${CLINKER} -o hello helloworld.o ${PETSC_SNES_LIB}
${PETSC_TS_LIB} ${LIBS}
```

- Compile:

  >make hello

- Run by:

  >mpirun –np 2 ./hello

Writing your 2nd PETSc code

- Pressure Poisson eqn:

$$\nabla^2 p = -\nabla.(\vec{u}.\nabla\vec{u})$$

or

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = -f$$

where

$$f = \nabla.(\vec{u}.\nabla\vec{u}) = \frac{\partial}{\partial x}\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) + \frac{\partial}{\partial y}\left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right)$$

- For Green-Taylor vortex problem:

$$f = \cos 2x + \cos 2y$$

- Analytic solution:

$$p = -\frac{1}{4}(\cos 2x + \cos 2y)$$

- Discretize with 5-point formula:

$$\beta^2 p(i,j-1) + p(i-1,j) - 2\left(1+\beta^2\right)p(i,j) + p(i+1,j) + \beta^2 p(i,j+1) = -f(i,j)$$

- Solve with Dirichlet and Neumann BC.

Problem Setup

- With the 5-point formula, 5 variables and 1 eqn: need to solve all the nodes together

$$A\vec{p} = \vec{f}$$

  where A is the matrix (operator), and $\vec{p}$ and $\vec{f}$ are the 2D arrays put into a vector.

- PETSc has parallel vectors (Vec), matrices (Mat), and solvers (KSP) to solve this equation.

- Parallel data structure – use DM: "DM objects are used to manage communication between the algebraic structures in PETSc (Vec and Mat) and mesh data structures in PDE-based (or other) simulation."

Define User Context

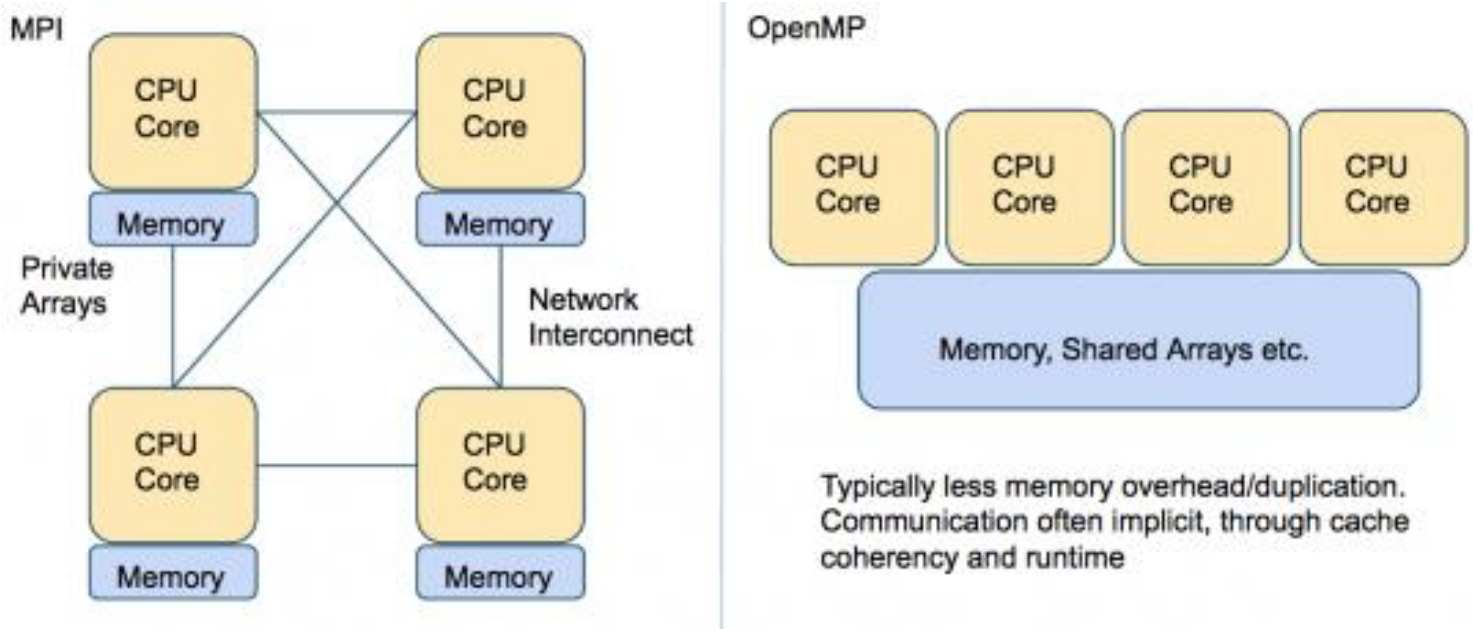
- Not necessary but good programming practice
- Put all the variable in variable for easy passage to subroutines (functions)
- Add this to your helloworld program:

```
#include  <petscsnes.h>
typedef struct {
  DM       da2D;  // Distributed array
  PetscInt Nx, Ny; //Grid size
  Vec      P, lP, P_Analytical, f; //pressure (global p and local p)
and rhs (f) vectors
  Mat      A;    // matrix(operator) A to solve Ap=f
} UserContext;

int main(int argc, char **argv) {
  UserContext *user; // Define the user context variable
  PetscInitialize(&argc, &argv, (char*)0, help);
 ….
```

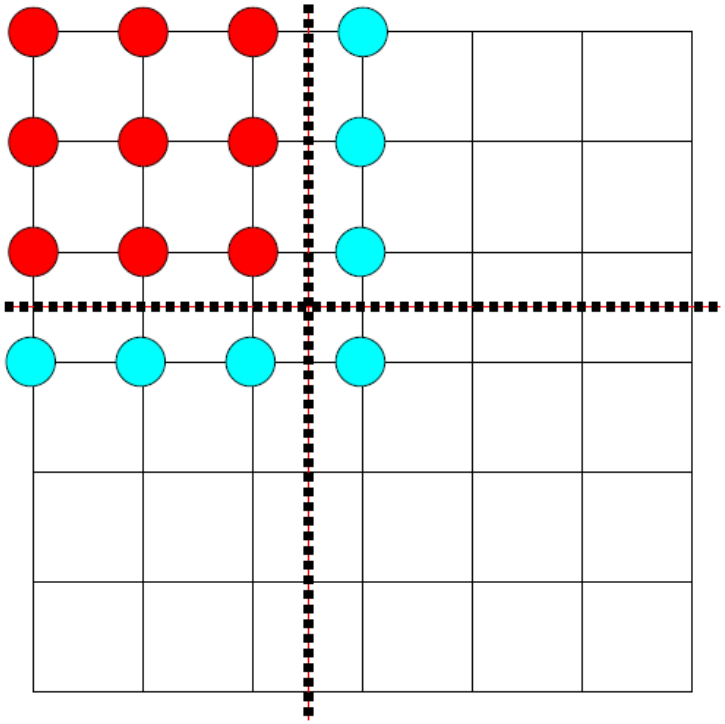
Types of Parallelism

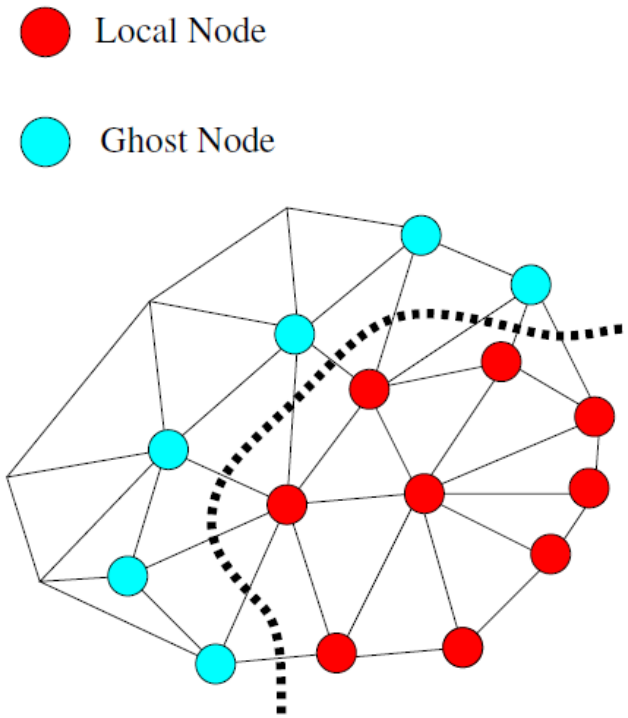
- Distributed Memory
  (MPI: Message Passing Interface)
- Shared Memory
  (OpenMP: Open Multi Processing)



DM: Distributed Memory – Ghost nodes

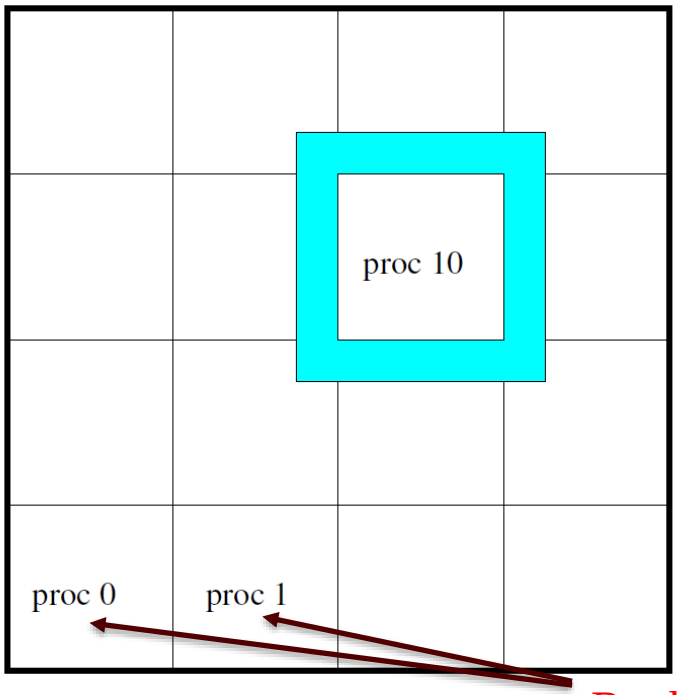
- To evaluate a local function x, each process requires:
  - Its local portion of the vector x
  - Its ghost values, bordering portions of x owned by neighboring processes



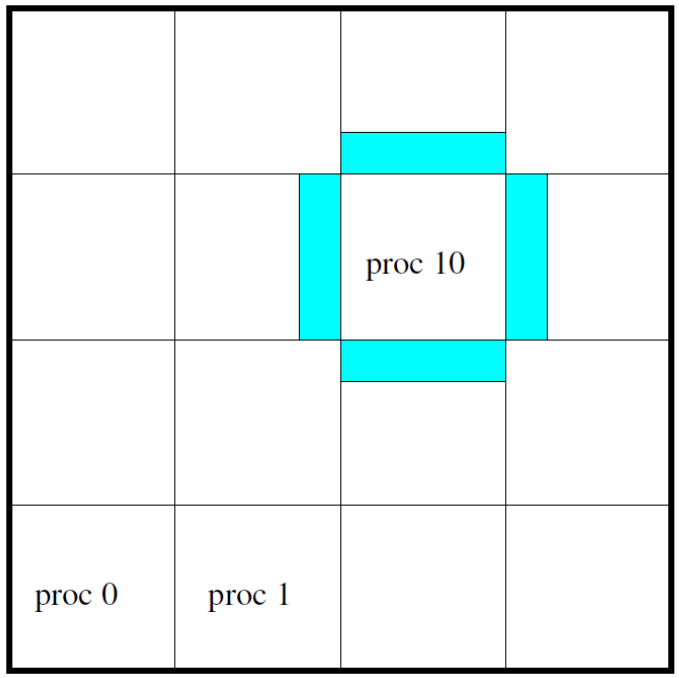


DMDA Stencils

- Both the box stencil and star stencil are available.



Box Stencil



Star Stencil

Rank and size of MPI comm

- Add this to your program, compile, and run.
- Compare PETSC_COMM_WORLD and PETSC_COMM_SELF

```
…
int main(int argc, char **argv) {

  UserContext *user; // Define the user context
variable
  PetscInitialize(&argc, &argv, (char*)0, help);

  PetscInt rank, size;
  MPI_Comm_size(PETSC_COMM_WORLD, &size);
  MPI_Comm_rank(PETSC_COMM_WORLD, &rank);
  PetscPrintf(PETSC_COMM_SELF, "Hello World!!
rank %d size %d\n", rank, size);
…
```

Create DA, vectors, and Mats

- In the main call the subroutine and pass the user variable:

```
CreateCtx(&user);
```

- The CreateCtx subroutine:

```
PetscErrorCode CreateCtx(UserContext *user)
{
  // Create the DA
  //grid size
  user->Nx=40; user->Ny=40;
  // also get from options from command lineg
  PetscOptionsGetInt(NULL,PETSC_NULL, "-Nx", &user->Nx,
PETSC_NULL);
  PetscOptionsGetInt(NULL,PETSC_NULL, "-Ny", &user->Ny,
PETSC_NULL);
```

Creating a DMDA

```
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

- In CreateCtx subroutine add:

```
DMDACreate2d(PETSC_COMM_WORLD, DM_BOUNDARY_NONE,
  DM_BOUNDARY_NONE, DMDA_STENCIL_STAR, user->Nx,
  user->Ny, PETSC_DECIDE, PETSC_DECIDE, 1, 1, NULL,
NULL,  &user->da2D);
DMSetFromOptions(user->da2D);
DMSetUp(user->da2D);
```

DM Vectors

- The DM object contains only layout (topology) information
  - All field data is contained in PETSc Vecs
- 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!
- Create vectros p,lp,f in CreateCtx:

```
/* Create Vectors */
 PetscErrorCode ierr;
 ierr = DMCreateGlobalVector(user->da2D, &user->p); CHKERRQ(ierr);
 VecDuplicate(user->P, user->f); /* VecDuplicate() DOES NOT COPY the vector entries,
                                    but rather allocates storage for the new vector. */
 VecDuplicate(user->P, &user->P_Analytical);

 ierr = DMCreateLocalVector(user->da2D, &user->lP); CHKERRQ(ierr);
```

DMDA Global Numbering

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

DMDA Global vs. Local Numbering

- **Global**: Each vertex has a unique id belongs on a unique process
- **Local**: Numbering includes vertices from neighboring processes
  - These are called ghost vertices

Proc 2			Proc 3	
X	X	X	X	X
X	X	X	X	X
12	13	14	15	X
8	9	10	11	X
4	5	6	7	X
0	1	2	3	X
Proc 0			Proc 1	

Local numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

Global numbering

Set Grid Coordinate for DA

- Define Cmp2D

```
typedef struct {
      PetscReal    x, y;
} Cmp2D;
```

- Coordinate vectors store the mesh geometry

```
/* Set the mesh coordinates for the DM
   For uniform mesh you can use this*/
  DMDASetUniformCoordinates(user->da2D, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0);
```

-

```
/* For more complex mesh you should use this:  */
  DM  cda;
  ierr = DMGetCoordinateDM(user->da2D, &cda); CHKERRQ(ierr);
  PetscPrintf(PETSC_COMM_SELF, "DM create cda!\n");
   Vec     gCoor;
   Cmp2D **coor;
   ierr = DMGetCoordinates(user->da2D, &gCoor);      CHKERRQ(ierr);
 PetscPrintf(PETSC_COMM_SELF, "DM create cda get gcoor!\n");
```

- Get the components of the vector on each rank into an array to manipulate

```
   PetscPrintf(PETSC_COMM_SELF, "DM set gcoor o!\n");
   ierr = DMDAVecGetArray(cda, gCoor, &coor); CHKERRQ(ierr);
   PetscPrintf(PETSC_COMM_SELF, "DM get gcoor array!\n");
```

Set the array values

```
// Get DM info
  DMDALocalInfo   info;
  DMDAGetLocalInfo(user->da2D, &info);
  PetscInt xs = info.xs, xe = info.xs + info.xm;
  PetscInt     ys = info.ys, ye = info.ys + info.ym;
  PetscInt zs = info.zs, ze = info.zs + info.zm;
  PetscInt      i,j,k;
  PetscReal Lx=1., Ly=1., dx, dy;
  user->Nx=info.mx; user->Ny=info.my;
  PetscPrintf(PETSC_COMM_SELF, "DM get info! xs %d xe %d ys
%d ye %d\n", xs, xe, ys, ye);
  dx = Lx/(user->Nx-1.);
  dy = Ly/(user->Ny-1.);
  for (j=ys; j<ye; j++) {
    for (i=xs; i<xe; i++) {
      coor[j][i].x = dx*i;
      coor[j][i].y = dy*j;
    }
  }
  user->beta=dx/dy; //Lx/Ly*(user->Ny-1.)/(user->Nx-1.);
```

Set Analytical Solution and RHS

```
PetscPrintf(PETSC_COMM_SELF, "DM create coor! beta
%f\n", user->beta);
  //set analytical p and RHS f
  PetscReal **p, **f;
  DMDAVecGetArray(user->da2D, user->P_Analytical, &p);
  DMDAVecGetArray(user->da2D, user->f, &f);
  for (j=ys; j<ye; j++) {
    for (i=xs; i<xe; i++) {
      p[j][i]=-
0.25*(cos(2*coor[j][i].x)+cos(2*coor[j][i].y));
      if (i==0 || j==0 || i==(user->Nx-1) || j==(user-
>Ny-1))
    // set f=p on the bounday for Drichlet BC
    f[j][i] = p[j][i];
      else
    f[j][i]=dx*dx*(cos(2*coor[j][i].x)+cos(2*coor[j][i].
y));
    }
  }
```

Restore Arrays

- Remember to restore arrays. If not, there will be memory leaks

```
 PetscPrintf(PETSC_COMM_SELF, "DM set P
analtyical!\n");
  // make sure to restore arrays --- else there will
be memory leak
  DMDAVecRestoreArray(user->da2D, user->P_Analytical,
&p);
  DMDAVecRestoreArray(user->da2D, user->f, &f);
  DMDAVecRestoreArray(cda, gCoor, &coor);

  // ierr = VecDestroy(&gCoor);CHKERRQ(ierr);
  // ierr = DMDestroy(&cda);CHKERRQ(ierr);
  return(0);
}
```

- This concludes the CreateCtx subroutine.

Create the Solver (KSP)

- In the main after calling the CreateCtx subroutine create the linear (KSP) solver

```
// Create KSP
  KSP             ksp;
  ierr = KSPCreate(PETSC_COMM_WORLD,&ksp);CHKERRQ(ierr);
  ierr = KSPSetDM(ksp,(DM)user.da2D);CHKERRQ(ierr);
  ierr =
DMSetApplicationContext(user.da2D,&user);CHKERRQ(ierr);

ierr =
KSPSetComputeOperators(ksp,ComputeLHS,&user);CHKERRQ(ier
r);
  ierr = KSPSetFromOptions(ksp);CHKERRQ(ierr);
  ierr = KSPSolve(ksp,user.f,user.P);CHKERRQ(ierr);
```

Set the LHS

- KSPSetComputeOperators(ksp,ComputeLHS,&user) sets the LHS, i.e., the matrix A, by introducing a function.
- Need to write the ComputeLHS subroutine.
- KSPSetComputeRHS(ksp,ComputeRHS,&user) can be used to set a function to compute the RHS.
- Not necessary here since we have a fix RHS and it was set in CreateCtx subroutine.

LHS subroutine

```
PetscErrorCode ComputeLHS(KSP ksp,Mat A, Mat B,void
*ctx)
{
  PetscErrorCode ierr;
  UserContext    *user = (UserContext*)ctx;
  DMDALocalInfo  info;
  DMDAGetLocalInfo(user->da2D, &info);
  PetscInt  xs = info.xs, mx = info.mx,
xe=info.xs+info.xm;
  PetscInt       ys = info.ys, my = info.my,
ye=info.ys+info.ym;
  //  PetscIntzs = info.zs, zm = info.zs + info.zm;
  PetscInt       i,j,k;
  MatStencil     row, col[5];
  PetscScalar    v[5];
  MatNullSpace   nullspace;
```

Setting values into the Matrix

```
// set matrix A coefficients
 for (j=ys; j<ye; j++) {
   for (i=xs; i<xe; i++) {
     row.i = i; row.j = j;

     if (i==0 || j==0 || i==mx-1 || j==my-1) {
    // Drichlet BC: on the boundary p=p_bc so only a diagonal coeff of 1
    v[0]= 1.; col[0].i = i; col[0].j=j;
    ierr = MatSetValuesStencil(A,1,&row,1,col,v,INSERT_VALUES);CHKERRQ(ierr);
     } else {
    // set the A based on the 5 point formula
    // beta^2*p(i,j-1)+p(i-1,j)-2(1+beta^2)p(i,j)+p(i+1,j)+beta^2p(i,j+1)=f
    v[0]= user->beta*user->beta; col[0].i = i  ; col[0].j=j-1; //i, j-1
    v[1]= 1.                   ; col[1].i = i-1; col[1].j=j  ; //i-1, j
    v[2]=-2*(1.+user->beta*user->beta); col[2].i = i  ; col[2].j=j; //i, j
    v[3]= 1.                   ; col[3].i = i+1; col[3].j=j  ; //i+1, j
    v[4]= user->beta*user->beta; col[4].i = i  ; col[4].j=j+1; //i, j+1
    ierr = MatSetValuesStencil(A,1,&row,5,col,v,INSERT_VALUES);CHKERRQ(ierr);
     }
   }
 }
```

Assemble the matrix

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

  //ierr = MatCopy(A, B,
SAME_NONZERO_PATTERN);CHKERRQ(ierr);

  /* ierr =
MatNullSpaceCreate(PETSC_COMM_WORLD,PETSC_TRUE,0,0,&nullspac
e);CHKERRQ(ierr); */
  /* ierr = MatSetNullSpace(A,nullspace);CHKERRQ(ierr); */
  /* ierr = MatNullSpaceDestroy(&nullspace);CHKERRQ(ierr);
*/

  return(0);
}
```

- No need for null-space with Dirichlet conditions

Output Solution

- Back in the main use viewer to output the solution

```
// writeout the solution vtk format
  PetscViewer viewer;
  PetscObjectSetName((PetscObject) user.P, "p");
  PetscObjectSetName((PetscObject) user.P_Analytical,
"p_analytical");
  PetscViewerCreate(PETSC_COMM_WORLD, &viewer);
  PetscViewerSetType(viewer, PETSCVIEWERVTK);
  PetscViewerFileSetMode(viewer, FILE_MODE_WRITE);
  PetscViewerFileSetName(viewer, "pressure.vts");
  VecView(user.P, viewer);
  VecView(user.P_Analytical, viewer);
  PetscViewerDestroy(&viewer);
```

Compute Error

```
// test the solution and compute error
  // Error = P-P_analytical
  VecAXPY(user.P, -1., user.P_Analytical);
  PetscReal err, err_inf;
  VecNorm(user.P, NORM_2, &err);
  err= err/ (user.Nx*user.Ny);
  VecNorm(user.P, NORM_INFINITY, &err_inf);
  PetscPrintf(PETSC_COMM_WORLD, "Error 2 norm %f,
infinity norm %f \n", err, err_inf);
```

Destroy what you created

- Free memory

```
  DestroyCtx(&user);

  ierr = KSPDestroy(&ksp);CHKERRQ(ierr);
  ierr = DMDestroy(&user.da2D);CHKERRQ(ierr);
  ierr = PetscFinalize();

  return(0);
} // end main


PetscErrorCode DestroyCtx(UserContext *user)
{
  PetscErrorCode ierr;
  ierr = VecDestroy(&user->P_Analytical);CHKERRQ(ierr);
  ierr = VecDestroy(&user->P);CHKERRQ(ierr);
  ierr = VecDestroy(&user->lP);CHKERRQ(ierr);
  ierr = VecDestroy(&user->f);CHKERRQ(ierr);

  return(0);
}
```

Run the code

- make poisson
- mpirun –np 4 ./poisson
- Use -log_view options to ensure balance create/destroy

- Practice at home:
- Compare your solver against ex50.c
- What is similar and what is different?
- Run with different solver/pc by using different options at run time, e.g., -ksp_…
- Check the parallel efficiency of your code

Contact:

Email: iman@tamu.edu