Introduction to PETSc

Bhupender Thakur

Outline

- **Installation on LONI and HPC**
- **Brief overview**
- **Examples on Vec, Mat, KSP, SNES**

Download petsc source to better follow the discussion
wget ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.2-p6.tar.gz
http://cct.lsu.edu/~bthakur/petsc_tutorial.pdf
http://cct.lsu.edu/~bthakur/petsc_tutorial.pptx

PETSc

http://www.mcs.anl.gov/petsc/petsc-current

- Aimed at parallel non-trivial PDE solvers
- Portable to any parallel system supporting MPI
- Offers robust scaling
- Supports many languages: C, Fortran, Python

Installing PETSc

- **Download current release 3.2-7 or 3.2-6**
  - wget ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.2-p6.tar.gz
- **Installation instructions**
  - www.mcs.anl.gov/petsc/documentation/installation.html
- **You may follow all examples from**
  - http://www.mcs.anl.gov/petsc/petsc-current/

Installing PETSc
on LONI

- **Compilers :**
  - intel_fc_11.1, intel_cc_11.1,
  - gcc-4.3.2
  - pgi-7.0-4
- **MPI :**
  - mvapich-2.1.4,
  - openmpi-1.3.4
- **Download**
  - --download-mpich=1
  - --download-openmpi=1

Source : http://docs.loni.org

Installing PETSc
on LONI

Configure: Check all options with ./configure –help

```
./configure  --prefix=/work/bthakur/soft/petsc-fd \
--with-fc=/usr/local/packages/mvapich2/1.6rc1/intel-11.1/bin/mpif90 \
 --with-cc=/usr/local/packages/mvapich2/1.6rc1/intel-11.1/bin/mpicc \
 --with-cxx=/usr/local/packages/mvapich2/1.6rc1/intel-11.1/bin/mpicxx \
--with-blas-lib=/usr/local/packages/lapack/3.2/intel-11.1/lib/libblas.a \
--with-lapack-lib=/usr/local/packages/lapack/3.2/intel-11.1/lib/liblapack.a \
--with-netcdf-dir=/usr/local/packages/netcdf/4.0/intel-11.1
```

Installing PETSc
on LONI

Other useful options

```
--COPTFLAGS=-O2
--with-pic=1
--with-fortran-datatypes=1
--with-netcdf-dir=/usr/local/packages/netcdf/4.0/intel-11.1
--with-cuda-dir=/usr/local/packages/cuda  (only on philip)
```

Installing PETSc
on LONI

```
Completed building libraries
=========================================
Now to install the libraries do:
make PETSC_DIR=/work/bthakur/soft/petsc-3.2-p6
PETSC_ARCH=arch-linux2-c-debug install
=========================================
$ make PETSC_DIR=/work/bthakur/soft/petsc-3.2-p6 PETSC_ARCH=arch-linux2-c-debug
install
*** using PETSC_DIR=/work/bthakur/soft/petsc-3.2-p6
PETSC_ARCH=arch-linux2-c-debug ***
*** Installing PETSc at /work/bthakur/soft/petsc-fd  ***
====================================
Install complete. It is useable with PETSC_DIR=/work/bthakur/soft/petsc-fd
[and no more PETSC_ARCH].
Now to check if the libraries are working do (in current directory):
make PETSC_DIR=/work/bthakur/soft/petsc-fd test
```

The PETSc Programming Model

- **Goals**
  - Portable, runs everywhere
  - Performance
  - Scalable parallelism
- **Approach**
  - Distributed memory, "shared-nothing"
    - Requires only a compiler (single node or processor)
    - Access to data on remote machines through MPI
  - Can still exploit "compiler discovered" parallelism on each node (e.g., SMP)
  - Hide within parallel objects the details of the communication
  - User orchestrates communication at a higher abstract level than message passing

PETSc Numerical Components

Nonlinear Solvers		
Newton-based Methods		Other
Line Search	Trust Region	

Time Steppers			
Euler	Backward Euler	Pseudo Time Stepping	Other

Krylov Subspace Methods							
GMRES	CG	CGS	Bi-CG-STAB	TFQMR	Richardson	Chebychev	Other

Preconditioners						
Additive Schwartz	Block Jacobi	Jacobi	ILU	ICC	LU (Sequential only)	Others

Matrices				
Compressed Sparse Row (AIJ)	Blocked Compressed Sparse Row (BAIJ)	Block Diagonal (BDIAG)	Dense	Other

Vectors

Index Sets			
Indices	Block Indices	Stride	Other

PETSc

Components

- **PETSc components :**
  - **Vectors**
  - **Matrices**
  - **Data and Grid management**
  - **Linear Solvers**
  - **Nonlinear solvers**
  - **Time stepping ODE solvers**

Source : http://

PETSc

Components

Vec:
Provides the vector operations required for setting up and solving large-scale linear and nonlinear problems. Includes easy-to-use parallel scatter and gather operations, as well as special-purpose code for handling ghost points for regular data structures.

Mat:
A large suite of data structures and code for the manipulation of parallel sparse matrices. Includes four different parallel matrix data structures, each appropriate for a different class of problems.

PC:
A collection of sequential and parallel preconditioners, including (sequential) ILU(k), LU, and (both sequential and parallel) block Jacobi, overlapping additive Schwarz methods and structured MG using DMMG.

PETSc
Components

KSP:
Parallel implementations of many popular Krylov subspace iterative methods, including GMRES, CG, CGS, Bi-CG-Stab, two variants of TFQMR, CR, and LSQR, immediately usable with any preconditioners and any matrix data structures, including matrix-free methods.

SNES:
Data-structure-neutral implementations of Newton-like methods for nonlinear systems. Includes both line search and trust region techniques with a single interface. Users can set custom monitoring routines, convergence criteria, etc.

TS:
Code for the time evolution of solutions of PDEs. In addition, provides pseudo-transient continuation techniques for computing steady-state solutions.

PETSc

- **Control over PETSc objects and types**
  - Allows control at command line of choice of methods abd objects -pc_type, -ksp_type, -ksp_monitor_draw -ksp_rtol

PETSc

Program structure

- Structure of a PETSc program

  - Initialize: Initializes the PETSc database and MPI
  - Create PETSc Objects, Solvers
  - Run Solvers
  - Destroy PETSc objects and Finalize

PETSc

Program structure

- **Initialize a PETSc program**
  - PetscInitialize(int *argc,char ***argv,char *file,char *help);
  - call PetscInitialize(character(*) file,integer ierr)

    PetscInitialize() automatically calls MPI_Init()
    sets PETSc "world" communicator, given by PETSC COMM WORLD

  - All PETSc programs should call PetscFinalize()

PETSc
Program structure

- **Initialize a PETSc program (include headers)**

petscsys.h	- base PETSc routines
petscvec.h	- vectors
petscmat.h	- matrices
petscis.h	- index sets
petscksp.h	- Krylov subspace methods
petscviewer.h	- viewers
petscpc.h	- preconditioners

PETSc
Program structure

- **PETSc Options database**
  - The user can input control data at run time using the options database. In this example the command

    PetscOptionsGetInt(PETSC NULL,”-n”,&n,&flg);

    checks whether the user has provided a command line option to set the value of n, the problem dimension.

  - To print a list of available options for a given program
    mpiexec -n 1 ./ex1 -help

PETSc

Program structure

- **PETSc Options database**
  - Krylov subspace technique BiCGStab by calling KSPSetType(ksp,KSPBCGS);

    One could then override this choice at runtime with the option -ksp type tfqmr

PETSc

Program structure

- **Structure of a PETSc program**

```
#include "petsc.h”
int main( int argc, char *argv[] )
{
        int rank;
        PetscInitialize( &argc, &argv, 0, 0 );
        MPI_Comm_rank( PETSC_COMM_WORLD, &rank );

        PetscSynchronizedPrintf( PETSC_COMM_WORLD,
        "Hello World from rank %d\n", rank );

        PetscSynchronizedFlush( PETSC_COMM_WORLD );
        PetscFinalize( );
        return 0;
}
```

PETSc

Vec: PETSc vectors (Vec objects) are used to store the field variables

- **PETSc vectors :** docs/manualpages/Vec/index.html
  - **Beginner - Basic usage**
    VecCreate, VecCopy, VecDestroy, VecLog, VecType, VecSet, VecView, VecDuplicate, VecScatter, VecPermute, Norm_
  - **Intermediate – Options for algorithms and data structures**
    VecCreateMPI, VecAbs, VecMax, VecDot, VecShift, VecNormalize, VecMax
  - **Advanced**
    VecPointwiseMin, VecStrideNorm, VecCreateGhost

PETSc

Vectors- Example 1

- ex1.c

```
int main(int argc,char **argv)
{
  Vec              x,y,w;         /* vectors */
  Vec              *z;            /* array of vectors */
  PetscReal        norm,v,v1,v2,maxval;
  PetscInt         n = 20,maxind;
  PetscErrorCode   ierr;
  PetscScalar      one = 1.0,two = 2.0,three = 3.0,dots[3],dot;

  ierr = PetscInitialize(&argc,&argv,(char*)0,help);                    CHKERRQ(ierr);
  ierr = PetscOptionsGetInt(PETSC_NULL,"-n",&n,PETSC_NULL); CHKERRQ(ierr);
```

Source: http://www.mcs.anl.gov/petsc/petsc-current/src/vec/vec/examples/tutorials

PETSc

Vectors- Example 1

- ex1.c - contd

```
ierr = VecCreate(PETSC_COMM_WORLD,&x);CHKERRQ(ierr);
ierr = VecSetSizes(x,PETSC_DECIDE,n);CHKERRQ(ierr);

ierr = VecDuplicate(x,&y);CHKERRQ(ierr);
ierr = VecDuplicate(x,&w);CHKERRQ(ierr);
ierr = VecNorm(w,NORM_2,&norm);CHKERRQ(ierr);
ierr = VecDuplicateVecs(x,3,&z);CHKERRQ(ierr);

ierr = VecSet(x,one);CHKERRQ(ierr);
ierr = VecSet(y,two);CHKERRQ(ierr);
ierr = VecSet(z[0],one);CHKERRQ(ierr);
ierr = VecSet(z[1],two);CHKERRQ(ierr);
ierr = VecSet(z[2],three);CHKERRQ(ierr);
```

PETSc
Vectors- Example 2

- ex1.c - contd

```
  ierr = VecScale(x,two);CHKERRQ(ierr);
  ierr = VecNorm(x,NORM_2,&norm);CHKERRQ(ierr);
  ierr = VecCopy(x,w);CHKERRQ(ierr);
  ierr = VecPointwiseMult(w,y,x);CHKERRQ(ierr);
  ierr = VecNorm(w,NORM_2,&norm);CHKERRQ(ierr);

  ierr = VecDestroy(&x);CHKERRQ(ierr);
  ierr = VecDestroy(&y);CHKERRQ(ierr);
  ierr = VecDestroy(&w);CHKERRQ(ierr);
  ierr = VecDestroyVecs(3,&z);CHKERRQ(ierr);
  ierr = PetscFinalize();
  return 0;
}
```

PETSc

Example

# Makefile petsc3.0

```
PETSC_DIR = /usr/local/packages/petsc/3.0.0.p3/intel-11.1-mpich-1.2.7p1
include ${PETSC_DIR}/conf/base
ex1: ex1.o  chkopts
            -${FLINKER} -o ex1 ex1.o ${PETSC_VEC_LIB}
             ${RM} -f ex1.o
```

# Makefile petsc3.2

```
PETSC_DIR = /work/bthakur/soft/petsc
include ${PETSC_DIR}/conf/variables
include ${PETSC_DIR}/conf/rules

ex1: ex1.o  chkopts
            -${CLINKER} -o ex1 ex1.o ${PETSC_VEC_LIB}
             ${RM} -f ex1.o
```

PETSc

Example- Fortran

```
program main

#include "finclude/petsc.h"
#include "finclude/petscvec.h"

!  Variables:
!     x, y, w - vectors
!     z       - array of vectors

      Vec             x,y,w,z(5)
      PetscReal       norm,v,v1,v2
      PetscInt        n,ithree
      PetscTruth      flg
      PetscErrorCode ierr
      PetscMPIInt  rank
      PetscScalar  one,two,three

call PetscInitialize &
(PETSC_NULL_CHARACTER,ierr)
…
call PetscOptionsGetInt &
(PETSC_NULL_CHARACTER,'n',n,flg,ierr)
…
call MPI_Comm_rank &
(PETSC_COMM_WORLD,rank,ierr)
…
call VecCreate &
(PETSC_COMM_WORLD,x,ierr)
…
call VecDot(x,x,dot,ierr)
call VecNorm(x,NORM_2,norm,ierr)
call VecDestroy(x,ierr)
…
call PetscFinalize(ierr)

end
```

www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/Sys/UsingFortran.html

PETSc

Vectors- Example 2

- Demo vec/vec/examples/tutorials/ex20f90.F90

PETSc

Mat: Abstract PETSc matrix object

- **PETSc matrices** : docs/manualpages/Mat/index.html
  - **Beginner - Basic usage**
    MatCreate, MatCreateMPIAIJ, MatSetValues,
    -mat_type aij -sets the type to "aij" with call to MatSetFromOptions
  - **Intermediate – Options for algorithms and data structures**
    MatConvert, MatCopy, MatTranspose
  - **Advanced**
    MatGetColumnVector, MatGetDiagonalBlock

Source : www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/Mat/index.html

PETSc
Matrices- Example

- Demo:
  mat/examples/tutorials/ex10.c

Source : http://www.mcs.anl.gov/petsc/petsc-current/src/mat/examples/tutorials/

PETSc

KSP : Abstract PETSc object that manages all Krylov methods

- **PETSc vectors :** docs/manualpages/Vec/index.html
  - **Beginner - Basic usage**
    KSPCreate, KSPDestroy, KSPReset
  - **Intermediate – Options for algorithms and data structures**
  - **Advanced**

Source : http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/KSP/

PETSc

KSP : Abstract PETSc object that manages all Krylov methods

- **PETSc KSP :** The Louisville-Bratu-Gelfand Problem

$-\Delta \mathbf{u}=\mathbf{1}$, Δ=laplacian, $0 < x,y,z < 1$

- Simplification of the Solid-Fuel Ignition Problem
- Also a nonlinear eigenproblem
- Dirichlet conditions : $u=0$ for $x=0$, $x=1$, $y=0$, $y=1$, $z=0$, $z=1$

PETSc

KSP- Example : ksp/ksp/examples/tutorials/ex45.c

```
int main(int argc,char **argv)
{
 PetscErrorCode          ierr;
 KSP                     ksp;
 PetscReal               norm;
 DM                      da;
 Vec                     x,b,r;
 Mat                     A;
 PetscInitialize(&argc,&argv,(char *)0,help);

 ierr = KSPCreate(PETSC_COMM_WORLD,&ksp);CHKERRQ(ierr);
 ierr = DMDACreate3d(PETSC_COMM_WORLD,DMDA_BOUNDARY_NONE,DMDA_BOUNDARY_NONE,DMDA_BOUNDARY_NONE,DMDA
_STENCIL_STAR,-7,-7,-7,PETSC_DECIDE,PETSC_DECIDE,PETSC_DECIDE,1,1,0,0,0,&da);CHKERRQ(ierr);
 ierr = DMSetInitialGuess(da,ComputeInitialGuess);CHKERRQ(ierr);
 ierr = DMSetFunction(da,ComputeRHS);CHKERRQ(ierr);
 ierr = DMSetJacobian(da,ComputeMatrix);CHKERRQ(ierr);
 ierr = KSPSetDM(ksp,da);CHKERRQ(ierr);
 /*  ierr = KSPSetDMActive(ksp,PETSC_FALSE);CHKERRQ(ierr);*/
 ierr = DMDestroy(&da);CHKERRQ(ierr);

 ierr = KSPSetFromOptions(ksp);                                              CHKERRQ(ierr);
 ierr = KSPSetUp(ksp);                                                       CHKERRQ(ierr);
 ierr = KSPSolve(ksp,PETSC_NULL,PETSC_NULL);                   CHKERRQ(ierr);
 ierr = KSPGetSolution(ksp,&x);                                CHKERRQ(ierr);
 ierr = KSPGetRhs(ksp,&b);                                     CHKERRQ(ierr);
 ierr = VecDuplicate(b,&r);                                                  CHKERRQ(ierr);
 ierr = KSPGetOperators(ksp,&A,PETSC_NULL,PETSC_NULL);         CHKERRQ(ierr);

 ierr = MatMult(A,x,r);CHKERRQ(ierr);
 ierr = VecAXPY(r,-1.0,b);CHKERRQ(ierr);
 ierr = VecNorm(r,NORM_2,&norm);CHKERRQ(ierr);
 ierr = PetscPrintf(PETSC_COMM_WORLD,"Residual norm %G\n",norm);CHKERRQ(ierr);

 ierr = VecDestroy(&r);CHKERRQ(ierr);
 ierr = KSPDestroy(&ksp);CHKERRQ(ierr);
 ierr = PetscFinalize();

 return 0;
}
```

Louisville-Bratu-Gelfand Problem

$-\Delta \mathbf{u}=\mathbf{1},$

Δ=laplacian, $0 < x,y,z < 1$

PETSc

KSP- Example : ksp/ksp/examples/tutorials/ex45.c

```
int main(int argc,char **argv)
{
  PetscErrorCode          ierr;
  KSP                     ksp;
  PetscReal               norm;
  DM                      da;
  Vec                     x,b,r;
  Mat                     A;
  PetscInitialize(&argc,&argv,(char *)0,help);

  ierr = KSPCreate(PETSC_COMM_WORLD,&ksp);CHKERRQ(ierr);
  ierr = DMDACreate3d(PETSC_COMM_WORLD,DMDA_BOUNDARY_NONE,DMDA_BOUNDARY_NONE,DMDA_BOUNDARY_NONE,DMDA
_STENCIL_STAR,-7,-7,-7,PETSC_DECIDE,PETSC_DECIDE,PETSC_DECIDE,1,1,0,0,0,&da);CHKERRQ(ierr);
  ierr = DMSetInitialGuess(da,ComputeInitialGuess);CHKERRQ(ierr);
  ierr = DMSetFunction(da,ComputeRHS);CHKERRQ(ierr);
  ierr = DMSetJacobian(da,ComputeMatrix);CHKERRQ(ierr);
  ierr = KSPSetDM(ksp,da);CHKERRQ(ierr);
  /*  ierr = KSPSetDMActive(ksp,PETSC_FALSE);CHKERRQ(ierr);*/
  ierr = DMDestroy(&da);CHKERRQ(ierr);

  ierr = KSPSetFromOptions(ksp);CHKERRQ(ierr);
  ierr = KSPSetUp(ksp);CHKERRQ(ierr);
  ierr = KSPSolve(ksp,PETSC_NULL,PETSC_NULL);CHKERRQ(ierr);
  ierr = KSPGetSolution(ksp,&x);CHKERRQ(ierr);
  ierr = KSPGetRhs(ksp,&b);CHKERRQ(ierr);
  ierr = VecDuplicate(b,&r);CHKERRQ(ierr);
 ierr = KSPGetOperators(ksp,&A,PETSC_NULL,PETSC_NULL);CHKERRQ(ierr);

  ierr = MatMult(A,x,r);CHKERRQ(ierr);
  ierr = VecAXPY(r,-1.0,b);CHKERRQ(ierr);
  ierr = VecNorm(r,NORM_2,&norm);CHKERRQ(ierr);
  ierr = PetscPrintf(PETSC_COMM_WORLD,"Residual norm %G\n",norm);CHKERRQ(ierr);

  ierr = VecDestroy(&r);CHKERRQ(ierr);
  ierr = KSPDestroy(&ksp);CHKERRQ(ierr);
  ierr = PetscFinalize();

  return 0;
}
```

PETSc

KSP- Example : ksp/ksp/examples/tutorials/ex45.c

```
int main(int argc,char **argv)
{
  PetscErrorCode                ierr;
  KSP                           ksp;
  PetscReal                     norm;
  DM                            da;
  Vec                           x,b,r;
  Mat                           A;
  PetscInitialize(&argc,&argv,(char *)0,help);

  ierr = KSPCreate(        PETSC_COMM_WORLD,&ksp);                          CHKERRQ(ierr);

  ierr = DMDACreate3d(     PETSC_COMM_WORLD, DMDA_BOUNDARY_NONE, DMDA_BOUNDARY_NONE,
                           DMDA_BOUNDARY_NONE, DMDA_STENCIL_STAR,-7,-7,-7,PETSC_DECIDE,
                           PETSC_DECIDE, PETSC_DECIDE,1,1,0,0,0,&da;         CHKERRQ(ierr);
  ierr = DMSetInitialGuess(da,ComputeInitialGuess);                         CHKERRQ(ierr);
  ierr = DMSetFunction(da,ComputeRHS);                                      CHKERRQ(ierr);
  ierr = DMSetJacobian(da,ComputeMatrix);                                   CHKERRQ(ierr);
  ierr = KSPSetDM(ksp,da);                                                  CHKERRQ(ierr);
  ierr = DMDestroy(&da);                                                    CHKERRQ(ierr);

  ierr = KSPSetFromOptions(ksp);CHKERRQ(ierr);
  ierr = KSPSetUp(ksp);CHKERRQ(ierr);
  ierr = KSPSolve(ksp,PETSC_NULL,PETSC_NULL);CHKERRQ(ierr);
  ierr = KSPGetSolution(ksp,&x);CHKERRQ(ierr);
  ierr = KSPGetRhs(ksp,&b);CHKERRQ(ierr);
  ierr = VecDuplicate(b,&r);CHKERRQ(ierr);
 ierr = KSPGetOperators(ksp,&A,PETSC_NULL,PETSC_NULL);CHKERRQ(ierr);

  ierr = MatMult(A,x,r);CHKERRQ(ierr);
  ierr = VecAXPY(r,-1.0,b);CHKERRQ(ierr);
  ierr = VecNorm(r,NORM_2,&norm);CHKERRQ(ierr);
  ierr = PetscPrintf(PETSC_COMM_WORLD,"Residual norm %G\n",norm);CHKERRQ(ierr);

  ierr = VecDestroy(&r);CHKERRQ(ierr);
  ierr = KSPDestroy(&ksp);CHKERRQ(ierr);
  ierr = PetscFinalize();

  return 0;
}
```

PETSc

KSP- Example : ksp/ksp/examples/tutorials/ex45.c

```
int main(int argc,char **argv)
{
  PetscErrorCode              ierr;
  KSP                         ksp;
  PetscReal                   norm;
  DM                          da;
  Vec                         x,b,r;
  Mat                         A;
  PetscInitialize(&argc,&argv,(char *)0,help);

  ierr = KSPCreate(PETSC_COMM_WORLD,&ksp);CHKERRQ(ierr);
  ierr = DMDACreate3d(PETSC_COMM_WORLD,DMDA_BOUNDARY_NONE,DMDA_BOUNDARY_NONE,DMDA_BOUNDARY_NONE,DMDA
_STENCIL_STAR,-7,-7,-7,PETSC_DECIDE,PETSC_DECIDE,PETSC_DECIDE,1,1,0,0,0,&da);CHKERRQ(ierr);
  ierr = DMSetInitialGuess(da,ComputeInitialGuess);CHKERRQ(ierr);
  ierr = DMSetFunction(da,ComputeRHS);CHKERRQ(ierr);
  ierr = DMSetJacobian(da,ComputeMatrix);CHKERRQ(ierr);
  ierr = KSPSetDM(ksp,da);CHKERRQ(ierr);
  /*  ierr = KSPSetDMActive(ksp,PETSC_FALSE);CHKERRQ(ierr);*/
  ierr = DMDestroy(&da);CHKERRQ(ierr);

  ierr = KSPSetFromOptions(ksp);                                CHKERRQ(ierr);
  ierr = KSPSetUp(ksp);                                         CHKERRQ(ierr);
  ierr = KSPSolve(ksp,PETSC_NULL,PETSC_NULL);                   CHKERRQ(ierr);
  ierr = KSPGetSolution(ksp,&x);                                CHKERRQ(ierr);
  ierr = KSPGetRhs(ksp,&b);                                     CHKERRQ(ierr);
  ierr = VecDuplicate(b,&r);                                    CHKERRQ(ierr);
  ierr = KSPGetOperators(ksp,&A,PETSC_NULL,PETSC_NULL);         CHKERRQ(ierr);

  ierr = MatMult(A,x,r);CHKERRQ(ierr);
  ierr = VecAXPY(r,-1.0,b);CHKERRQ(ierr);
  ierr = VecNorm(r,NORM_2,&norm);CHKERRQ(ierr);
  ierr = PetscPrintf(PETSC_COMM_WORLD,"Residual norm %G\n",norm);CHKERRQ(ierr);

  ierr = VecDestroy(&r);CHKERRQ(ierr);
  ierr = KSPDestroy(&ksp);CHKERRQ(ierr);
  ierr = PetscFinalize();

  return 0;
```

PETSc

KSP- Example : ksp/ksp/examples/tutorials/ex45.c

```
int main(int argc,char **argv)
{
  PetscErrorCode          ierr;
  KSP                     ksp;
  PetscReal               norm;
  DM                      da;
  Vec                     x,b,r;
  Mat                     A;
  PetscInitialize(&argc,&argv,(char *)0,help);

  ierr = KSPCreate(PETSC_COMM_WORLD,&ksp);CHKERRQ(ierr);
  ierr = DMDACreate3d(PETSC_COMM_WORLD,DMDA_BOUNDARY_NONE,DMDA_BOUNDARY_NONE,DMDA_BOUNDARY_NONE,DMDA
_STENCIL_STAR,-7,-7,-7,PETSC_DECIDE,PETSC_DECIDE,PETSC_DECIDE,1,1,0,0,0,&da);CHKERRQ(ierr);
  ierr = DMSetInitialGuess(da,ComputeInitialGuess);CHKERRQ(ierr);
  ierr = DMSetFunction(da,ComputeRHS);CHKERRQ(ierr);
  ierr = DMSetJacobian(da,ComputeMatrix);CHKERRQ(ierr);
  ierr = KSPSetDM(ksp,da);CHKERRQ(ierr);
  /*  ierr = KSPSetDMActive(ksp,PETSC_FALSE);CHKERRQ(ierr);*/
  ierr = DMDestroy(&da);CHKERRQ(ierr);

  ierr = KSPSetFromOptions(ksp);                                          CHKERRQ(ierr);
  ierr = KSPSetUp(ksp);                                                   CHKERRQ(ierr);
  ierr = KSPSolve(ksp,PETSC_NULL,PETSC_NULL);                   CHKERRQ(ierr);
  ierr = KSPGetSolution(ksp,&x);                                CHKERRQ(ierr);
  ierr = KSPGetRhs(ksp,&b);                                     CHKERRQ(ierr);
  ierr = VecDuplicate(b,&r);                                              CHKERRQ(ierr);
  ierr = KSPGetOperators(ksp,&A,PETSC_NULL,PETSC_NULL);         CHKERRQ(ierr);

  ierr = MatMult(A,x,r);                                                  CHKERRQ(ierr);
  ierr = VecAXPY(r,-1.0,b);                                               CHKERRQ(ierr);
  ierr = VecNorm(r,NORM_2,&norm);                                         CHKERRQ(ierr);
  ierr = PetscPrintf(PETSC_COMM_WORLD,"Residual norm %G\n",norm);         CHKERRQ(ierr);

  ierr = VecDestroy(&r);CHKERRQ(ierr);
  ierr = KSPDestroy(&ksp);CHKERRQ(ierr);
  ierr = PetscFinalize();

  return 0;
}
```

PETSc

KSP- Example : ksp/ksp/examples/tutorials/ex45.c

```
int main(int argc,char **argv)
{
  PetscErrorCode          ierr;
  KSP                     ksp;
  PetscReal               norm;
  DM                      da;
  Vec                     x,b,r;
  Mat                     A;
  PetscInitialize(&argc,&argv,(char *)0,help);

  ierr = KSPCreate(PETSC_COMM_WORLD,&ksp);CHKERRQ(ierr);
  ierr = DMDACreate3d(PETSC_COMM_WORLD,DMDA_BOUNDARY_NONE,DMDA_BOUNDARY_NONE,DMDA_BOUNDARY_NONE,DMDA
_STENCIL_STAR,-7,-7,-7,PETSC_DECIDE,PETSC_DECIDE,PETSC_DECIDE,1,1,0,0,0,&da);CHKERRQ(ierr);
  ierr = DMSetInitialGuess(da,ComputeInitialGuess);CHKERRQ(ierr);
  ierr = DMSetFunction(da,ComputeRHS);CHKERRQ(ierr);
  ierr = DMSetJacobian(da,ComputeMatrix);CHKERRQ(ierr);
  ierr = KSPSetDM(ksp,da);CHKERRQ(ierr);
  /*  ierr = KSPSetDMActive(ksp,PETSC_FALSE);CHKERRQ(ierr);*/
  ierr = DMDestroy(&da);CHKERRQ(ierr);

  ierr = KSPSetFromOptions(ksp);                                              CHKERRQ(ierr);
  ierr = KSPSetUp(ksp);                                                       CHKERRQ(ierr);
  ierr = KSPSolve(ksp,PETSC_NULL,PETSC_NULL);                    CHKERRQ(ierr);
  ierr = KSPGetSolution(ksp,&x);                                 CHKERRQ(ierr);
  ierr = KSPGetRhs(ksp,&b);                                      CHKERRQ(ierr);
  ierr = VecDuplicate(b,&r);                                                  CHKERRQ(ierr);
  ierr = KSPGetOperators(ksp,&A,PETSC_NULL,PETSC_NULL);          CHKERRQ(ierr);

  ierr = MatMult(A,x,r);CHKERRQ(ierr);
  ierr = VecAXPY(r,-1.0,b);CHKERRQ(ierr);
  ierr = VecNorm(r,NORM_2,&norm);CHKERRQ(ierr);
  ierr = PetscPrintf(PETSC_COMM_WORLD,"Residual norm %G\n",norm);CHKERRQ(ierr);

  ierr = VecDestroy(&r);          CHKERRQ(ierr);
  ierr = KSPDestroy(&ksp);        CHKERRQ(ierr);
  ierr = PetscFinalize();

  return 0;
}
```

PETSc

SNES : The Scalable Nonlinear Equations Solvers

- **PETSc SNES :** /docs/manualpages/SNES/index.html
  Abstract PETSc object that manages all nonlinear solves

  Beginner - Basic usage
  SNESCreate, SNESDestroy

  Intermediate – Options for algorithms and data structures
  SNESDefaultComputeJacobian, SNESGetType, SNESGetSolution

  Advanced
  SNESSetUp, SNESGetJacobian, SNESMatrixFreeCreate2

PETSc
SNES : The Scalable Nonlinear Equations Solvers

- **PETSc SNES :** The Louisville-Bratu-Gelfand Problem

$$-\Delta u - \lambda e^{u} = f, \qquad \Delta = \text{laplacian}$$

- Simplification of the Solid-Fuel Ignition Problem
- Also a nonlinear eigenproblem
- Dirichlet conditions : u=0 for x=0, x=1, y=0, y=1, z=0, z=1

PETSc

SNES- Example : ex14.c

- Demo: snes/examples/tutorials/ex14.c

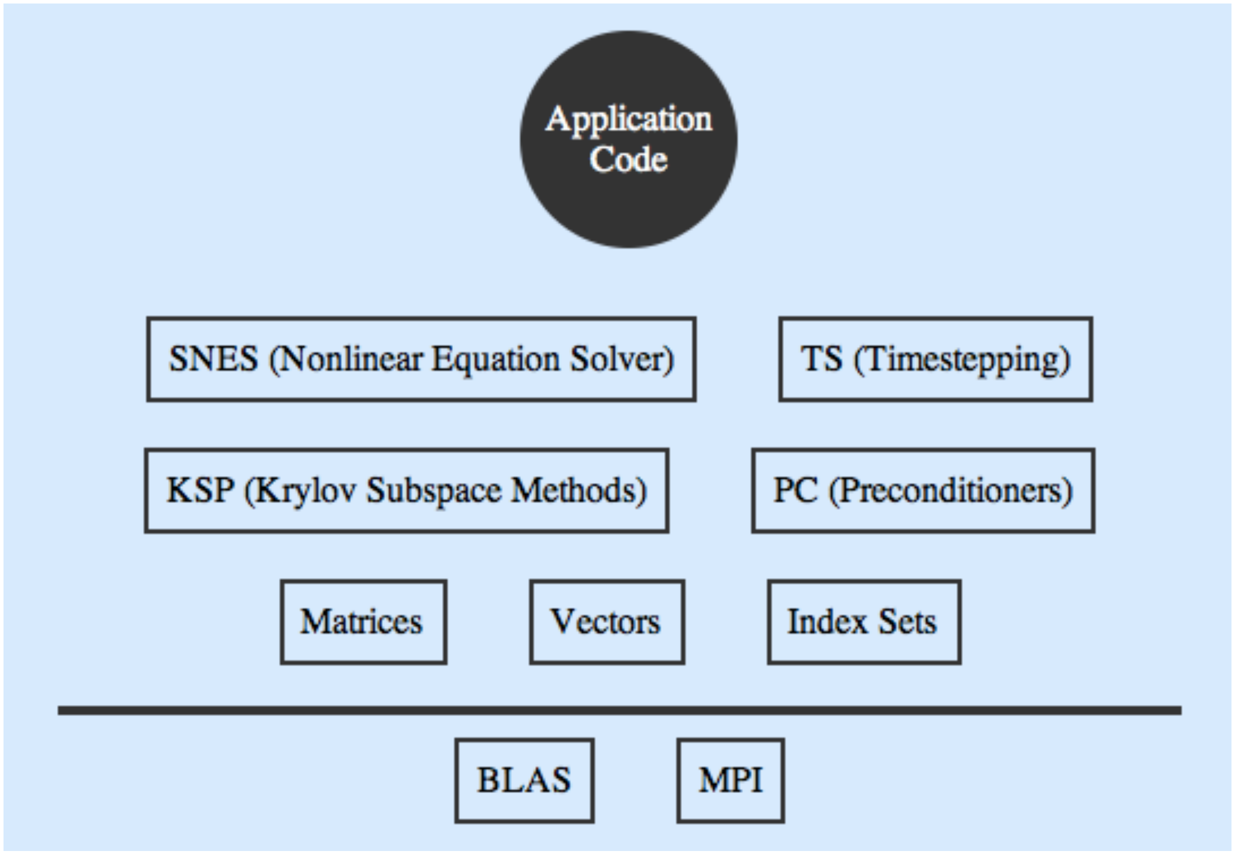
DA

Three pillars of computing

- ***Debug***
  Get rid of any bugs you may have.
- ***Optimize***
  Improve code performance. Often, you may need radical design changes to get large speedups.
- ***Profile***
  Profile and benchmark it to confirm the algorithm scales as expected

PETSc

Recap


Application Code
SNES (Nonlinear Equation Solver)
TS (Timestepping)
KSP (Krylov Subspace Methods)
PC (Preconditioners)
Matrices
Vectors
Index Sets
BLAS
MPI

PETSc

Advantages

- PETSc objects and procedures
- Single user interface but multiple underlying implementations
- Higher abstractions
- Integrated profiling and debugging

Conclusion

- Many advantages to using PETSc
- Allow easy structuring of non-trivial problems
- Try it yourself to find ways to enhance productivity

PETSc

Conclusion

- Many advantages to using PETSc
- Allow easy structuring of non-trivial problems
- Try it yourself to find ways to enhance productivity