PETSc: Portable, Extensible Toolkit for Scientific Computation

Zhengyi Zhang
zhang701@purdue.edu

Purdue University
Department of Computer Science

Match 28 , 2016

1

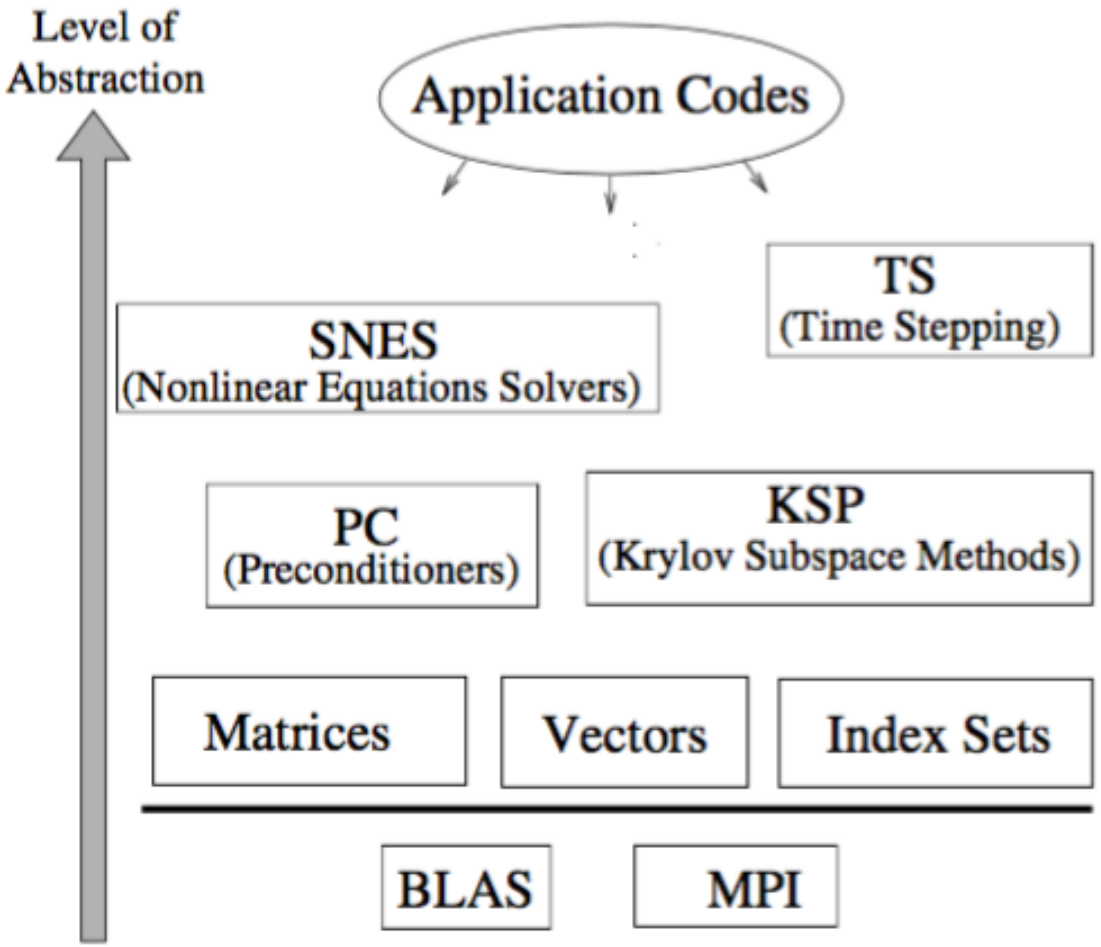
[1] **PETSc Users Manual 3.6**.

Outline

PETSc Version

- The current version is 3.6; released June 9, 2015.
- PETSc Users Manual Revision 3.6.

Organization of the PETSc Libraries



Numerical Libraries of PETSc

Parallel Numerical Components of PETSc

Nonlinear Solvers		
Newton–based Methods		Other
Line Search	Trust Region	

Time Steppers			
General Linear	IMEX	Pseudo–Time Stepping	Runge–Kutt

Krylov Subspace Methods							
GMRES	CG	CGS	Bi–CG–Stab	TFQMR	Richardson	Chebychev	Other

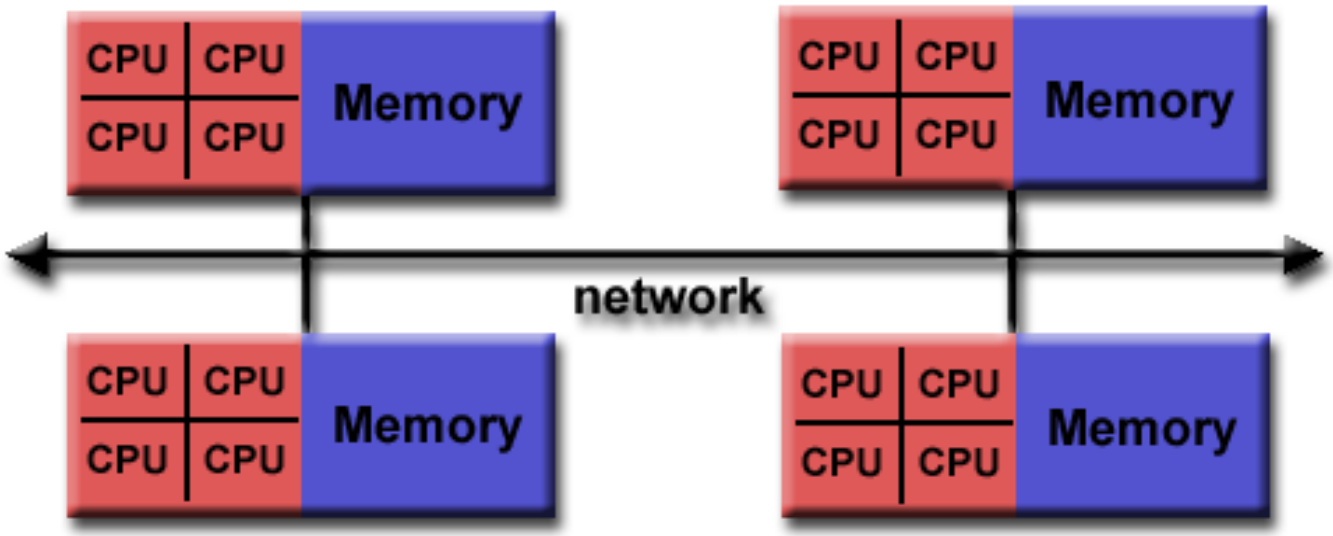
Preconditioners						
Additive Schwarz	Block Jacobi	Jacobi	ILU	ICC	LU (sequential only)	Other

Matrices				
Compressed Sparse Row (AIJ)	Block Compressed Sparse Row (BAIJ)	Symmetric Block Compressed Row (SBAIJ)	Dense	Other

Vectors

Index Sets			
Indices	Block Indices	Stride	Other

Message Passing Interface (MPI)



2

[2] **https://computing.llnl.gov/tutorials/mpi/**.

Threads

- PETSc: pure MPI, no OpenMP.
- Active OpenMP with ./configure only when one has many small systems (or sets of ODEs).

GPU

https://www.mcs.anl.gov/petsc/features/gpus.html

- CUDA (NVIDIA)
- OpenCL (NVIDIA, AMD, Intel MIC)

Supported languages

- PETSc: written in C
- C++
- Fortran
- Python
- MATLAB (limited)

Installation

- Linux with default installation options
  - ./configure –with-cc=gcc –with-cxx=g++ –with-fc=gfortran –download-fblaslapack –download-mpich
  - make all test
- Different architecture, external packages
  http://www.mcs.anl.gov/petsc/documentation/installation.html

Compiling: Makefile

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

ex1: ex1.o  chkopts
        -${CLINKER} -o ex1 ex1.o  ${PETSC_KSP_LIB}
        ${RM} ex1.o

ex2: ex2.o  chkopts
        -${CLINKER} -o ex2 ex2.o  ${PETSC_KSP_LIB}
        ${RM} ex2.o
```

Execution

mpiexec -np 8 ./petsc-program-name petsc-options

petsc-options:

- -log_summary
- -malloc_dump
- -info

Objects

- IS: index sets;
- Vec: vectors;
- Mat: matrices;
- DM: managing interactions between mesh data structures and vectors and matrices;
- KSP: Krylov subspace mthdos;
- PC: preconditioners;
- SNES: onlinear solvers;
- TS: timesteppers for solving time-dependent PDEs.

Vectors

- Sequential:
  VecCreateSeq(PETSC_COMM_SELF,int m,Vec *x);
- Parallel:
  VecCreateMPI(MPI_Comm comm,int m,int M,Vec *x);
- Or,
  VecCreate(MPI_Comm comm,Vec *v);
  VecSetSizes(Vec v, int m, int M);
  VecSetFromOptions(Vec v);

Vectors

- Assigning a single value to all components:
  VecSet(Vec x,PetscScalar value);
- Assigning a set of components:
  call
  VecSetValues(Vec x,int n,int *indices,PetscScalar *values,
  INSERT VALUES);
  any number of times, then call
  VecAssemblyBegin(Vec x);
  VecAssemblyEnd(Vec x);

Basic vector operations

Function Name	Operation
VecAXPY(Vec y,PetscScalar a,Vec x);	$y = y + a * x$
VecAYPX(Vec y,PetscScalar a,Vec x);	$y = x + a * y$
VecWAXPY(Vec w,PetscScalar a,Vec x,Vec y);	$w = a * x + y$
VecAXPBY(Vec y,PetscScalar a,PetscScalar b,Vec x);	$y = a * x + b * y$
VecScale(Vec x, PetscScalar a);	$x = a * x$
VecDot(Vec x, Vec y, PetscScalar *r);	$r = \bar{x}' * y$
VecTDot(Vec x, Vec y, PetscScalar *r);	$r = x' * y$
VecNorm(Vec x,NormType type, PetscReal *r);	$r = \|\|x\|\|_{type}$
VecSum(Vec x, PetscScalar *r);	$r = \sum x_i$
VecCopy(Vec x, Vec y);	$y = x$
VecSwap(Vec x, Vec y);	$y = x$ while $x = y$
VecPointwiseMult(Vec w,Vec x,Vec y);	$w_i = x_i * y_i$
VecPointwiseDivide(Vec w,Vec x,Vec y);	$w_i = x_i / y_i$
VecMDot(Vec x,int n,Vec y[],PetscScalar *r);	$r[i] = \bar{x}' * y[i]$
VecMTDot(Vec x,int n,Vec y[],PetscScalar *r);	$r[i] = x' * y[i]$
VecMAXPY(Vec y,int n, PetscScalar *a, Vec x[]);	$y = y + \sum_i a_i * x[i]$
VecMax(Vec x, int *idx, PetscReal *r);	$r = \max x_i$
VecMin(Vec x, int *idx, PetscReal *r);	$r = \min x_i$
VecAbs(Vec x);	$x_i = \|x_i\|$
VecReciprocal(Vec x);	$x_i = 1/x_i$
VecShift(Vec x,PetscScalar s);	$x_i = s + x_i$
VecSet(Vec x,PetscScalar alpha);	$x_i = \alpha$

Supported matrix formats

- Dense matrices
- Sparse matrices
- Block matrices
- Matrix-Free matrices (function handle in MATLAB)

Creating and assembling matrices

```
MatCreate(MPI_Comm comm,Mat *A)
MatSetSizes(Mat A,int m,int n,int M,int N)
MatSetValues(Mat A,int m,const int idxm[],int n,const int idxn[],
      const PetscScalar values[], INSERT VALUES);
MatAssemblyBegin(Mat A,MAT FINAL ASSEMBLY);
MatAssemblyEnd(Mat A,MAT FINAL ASSEMBLY);
```

Dense matrices

- Column major order, different from C/C++
- Sequential:
  MatCreateSeqDense(PETSC_COMM_SELF,int m,int n,
  PetscScalar *data,Mat *A);
- Parallel:
  MatCreateDense(MPI_Comm comm,int m,int n,int M,int N,
  PetscScalar *data,Mat *A);

Sparse matrices

The general sparse AIJ format (CSR: compressed sparse row)

$$\boldsymbol{A} = \begin{pmatrix} 10 & 20 & 0 & 0 \\ 0 & 30 & 0 & 0 \\ 0 & 0 & 40 & 50 \\ 70 & 0 & 0 & 80 \end{pmatrix}$$

$$VA = [10, 20, 30, 40, 50, 70, 80]$$
$$JA = [0, 1, 1, 2, 3, 0, 3]$$
$$IA = [0, 2, 3, 5]$$

Sequential AIJ sparse matrices

MatCreateSeqAIJ(PETSC_COMM_SELF,int m,int n,
int nz, int *nnz,Mat *A);

nz: the expected number of nonzeros in a given row.

Eg: the tridiagonal matrix

nnz: the array of length m, which indicate the exact number of elements for each row.

int nnz[m];

nnz[0]= nonzeros in row 0;

nnz[1]=nonzeros in row 1;

...

nnz[m-1]=nonzeros in row m-1;

Parallel AIJ sparse matrices

MatCreateMPIAIJ(MPI_Comm comm,int m,int n,int M,int N,int d_nz, int *d_nnz, int o_nz,int *o_nnz,Mat *A);

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

Basic matrix operations

Function Name	Operation
MatAXPY(Mat Y, PetscScalar a,Mat X,MatStructure);	$Y = Y + a * X$
MatMult(Mat A,Vec x, Vec y);	$y = A * x$
MatMultAdd(Mat A,Vec x, Vec y,Vec z);	$z = y + A * x$
MatMultTranspose(Mat A,Vec x, Vec y);	$y = A^T * x$
MatMultTransposeAdd(Mat A,Vec x, Vec y,Vec z);	$z = y + A^T * x$
MatNorm(Mat A,NormType type, double *r);	$r = \|\|A\|\|_{type}$
MatDiagonalScale(Mat A,Vec l,Vec r);	$A = \text{diag}(l) * A * \text{diag}(r)$
MatScale(Mat A,PetscScalar a);	$A = a * A$
MatConvert(Mat A,MatType type,Mat *B);	$B = A$
MatCopy(Mat A,Mat B,MatStructure);	$B = A$
MatGetDiagonal(Mat A,Vec x);	$x = \text{diag}(A)$
MatTranspose(Mat A,MatReuse,Mat* B);	$B = A^T$
MatZeroEntries(Mat A);	$A = 0$
MatShift(Mat Y,PetscScalar a);	$Y = Y + a * I$

The nonsingular systems

Solve

$$\boldsymbol{A}\mathbf{x} = \mathbf{b}$$

where $\boldsymbol{A}$ is nonsingular.

Using KSP

```
KSPCreate(MPI_Comm comm,KSP *ksp);
KSPSetOperators(KSP ksp,Mat Amat,Mat Pmat);
KSPSetFromOptions(KSP ksp);
KSPSolve(KSP ksp,Vec b,Vec x);
KSPDestroy(KSP *ksp);
```

KSP objects

Method	KSPType	Options Database Name
Richardson	KSPRICHARDSON	richardson
Chebyshev	KSPCHEBYSHEV	chebyshev
Conjugate Gradient [17]	KSPCG	cg
BiConjugate Gradient	KSPBICG	bicg
Generalized Minimal Residual [26]	KSPGMRES	gmres
Flexible Generalized Minimal Residual	KSPFGMRES	fgmres
Deflated Generalized Minimal Residual	KSPDGMRES	dgmres
Generalized Conjugate Residual	KSPGCR	gcr
BiCGSTAB [30]	KSPBCGS	bcgs
Conjugate Gradient Squared [29]	KSPCGS	cgs
Transpose-Free Quasi-Minimal Residual (1) [12]	KSPTFQMR	tfqmr
Transpose-Free Quasi-Minimal Residual (2)	KSPTCQMR	tcqmr
Conjugate Residual	KSPCR	cr
Least Squares Method	KSPLSQR	lsqr
Shell for no KSP method	KSPPREONLY	preonly
Shell for no KSP method	KSPPREONLY	preonly

KSP preconditioners

Method	PCType	Options Database Name
Jacobi	PCJACOBI	jacobi
Block Jacobi	PCBJACOBI	bjacobi
SOR (and SSOR)	PCSOR	sor
SOR with Eisenstat trick	PCEISENSTAT	eisenstat
Incomplete Cholesky	PCICC	icc
Incomplete LU	PCILU	ilu
Additive Schwarz	PCASM	asm
Generalized Additive Schwarz	PCGASM	gasm
Algebraic Multigrid	PCGAMG	gamg
Balancing Domain Decomposition by Constraints	PCBDDC	bddc
Linear solver	PCKSP	ksp
Combination of preconditioners	PCCOMPOSITE	composite
LU	PCLU	lu
Cholesky	PCCHOLESKY	cholesky
No preconditioning	PCNONE	none
Shell for user-defined PC	PCSHELL	shell

Examples

cp -r /u/u24/zhang701/cse/example .

- Direct solver:
  make ex0.c
  make run0
- Krylov subspace method with different preconditioners
  make ex1.c
  make run1
  make ex2.c
  make run2

When to use PETSc

PETSc is not intended for the classes of problems for which effective MATLAB code can be written

How to learn PETSc

- PETSc manual
- References on the web page (with examples for each functions)
- Examples in the src/.

How to write a new application program using PETSc,

1. Install and test PETSc according to the instructions at the PETSc web site.
2. Copy one of the many PETSc examples in the directory that corresponds to the class of problem of interest.
3. Copy the corresponding makefile within the example directory; compile and run the example program.
4. Use the example program as a starting point for developing a custom code.

Questions?