LSU INFORMATION TECHNOLOGY SERVICES

# Introduction to Numerical Libraries

Shaohao Chen

High performance computing @ Louisiana State University

# Outline

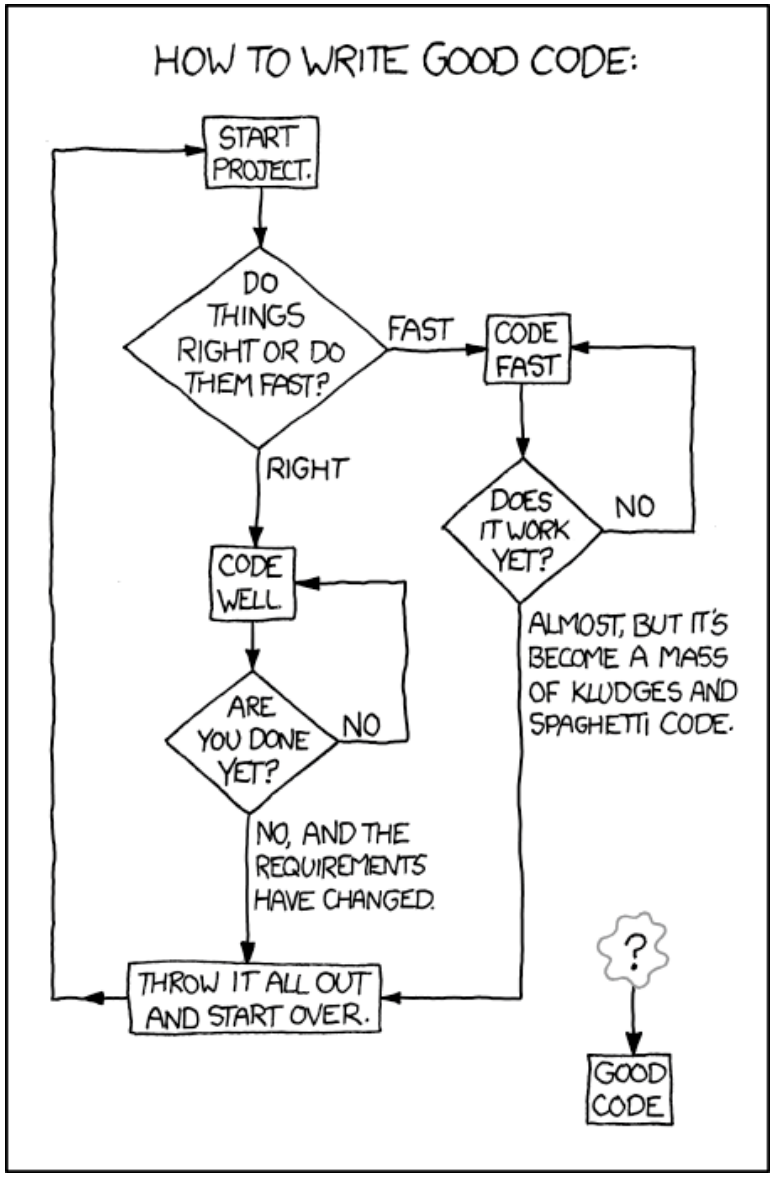
1. Introduction: why numerical libraries?
2. Fast Fourier transform: FFTw
3. Linear algebra libraries: LAPACK
4. Krylov subspace solver: PETSc
5. GNU scientific libraries: GSL

# 1. Introduction

It is very easy to write bad code!

- Loops : branching, dependencies
- I/O : resource conflicts,
- Memory : long fetch-time
- Portability : code and data portable?
- Readability : Can you still understand it?

A solution: look for existing libraries!



# Why numerical libraries?

- Many functions or subroutines you need may have already been coded by others. Just use them.
- Not necessary to code every line by yourself.
- Check available libs before starting to write your program.
- Save your time and efforts!

Advantages of using numerical libraries:

- Computing optimizations
- Portability
- Easy to debug
- Easy to read

# What you will learn in this training

- General knowledge of numerical libs.
- How to check available libs on HPC systems and the web.
- How to use numerical libs on HPC systems.
- Basic programming with numerical libs.

# List of notable numerical libraries

- Linear Algebra Package (LAPACK): computes matrix and vector operations. [Fortran]
- Fastest Fourier Transform in the West (FFTW): computes Fourier and related transforms. [C/C++]
- GNU Scientific Library (GSL): provides a wide range of mathematical routines. [C/C++]
- Portable, Extensible Toolkit for Scientific Computation (PETSc): a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. [C/C++]
- NumPy: adds support for the manipulation of large, multi-dimensional arrays and matrices; also includes a large collection of high-level mathematical functions. [Python]

- For more: http://en.wikipedia.org/wiki/List_of_numerical_libraries

# Prerequisite knowledge

## Compilers: compile source codes

- Intel: icc, icpc, ifort
- GNU: gcc, g++, gfortran
- PGI: pgcc, pgc++, pgf90

## MPI implementations: enable parallel computation with MPI standard

- mpich :
- mvapich2 :
- openmpi :
- impi :

# Installing libs

A typical three-step installation:

- configure : configure machine-dependent environments
- make : compiles source codes based on settings in the makefile
- make install : copy installed libs and binaries to a destination

Other types of installation:

- manually modify makefile, then make
- cmake : machine-dependent make

✓Most HPC users dot not have to install numerical libs.

# Available libs for LSU HPC and LONI users

- LSU HPC website

  http://www.hpc.lsu.edu/docs/guides/index.php

- module: for SuperMIC and QB2

  module av

  module whatis

  module list

- softenv: for SuperMike, phlip and eric

  softenv -a

  softenv -k

# How to use libs

## In your source codes:

- Call functions or subroutines provided by the libs.
- Include head files if necessary.

## In a Linux shell:

- Set up environmental valuables related to the libs (using module or softenv, or set up manually).
- Link your own codes to the precompiled libs.

## Notes

- In most cases, the compilers and the MPI implementations should be the same for your own codes and for the libs.

# Static libs

- libname.a files: an archive of a bunch of name.o files.

- Set up environmental valuables

  For bash: export LIBRARY_PATH=/path/to/lib

  For csh/tcsh: setenv LIBRARY_PATH /path/to/lib

  Use module: module load

  Use softenv: add keys to the ~/.soft file, then resoft, or add soft

- Compile and link

```
${compiler} –c –I/path/to/include name.c (or name.f)      # compile your own source code
${compiler} name.o –l${libname} -o name                   # link to the lib
${compiler} name.o –L/path/to/lib –l${libname} -o name    # if LIBRARY_PATH is not set
```

# Dynamic libs

- libname.so or libname.so.* files

- Set up environmental valuables for run-time access

  For bash: export LD_LIBRARY_PATH=/path/to/lib

  For csh/tcsh: setenv LD_LIBRARY_PATH /path/to/lib

  Use module: module load

  Use softenv: add keys to the ~/.soft file, then resoft, or add soft

Notes

- Make sure the executable binary can “see” the dynamic libs.
- If a parallel job runs on multi nodes, LD_LIBRARY_PATH has to be set for every node. Set it in the PBS batch script.

# 2. Fast Fourier Transform: FFTw

Main features:

- C subroutine library for computing the discrete Fourier transform (DFT)
- Both C and Fortran interfaces
- One or more dimensions
- Arbitrary input size
- Both real and complex data
- Even/odd data, i.e. the discrete cosine/sine transforms
- Efficient handling of multiple, strided transforms
- Parallel transforms: parallelized code for platforms with SMP machines with some flavor of threads (e.g. POSIX) or OpenMP. An MPI version for distributed-memory transforms is also available in FFTW 3.3.

# FFTw basics


Data type

- fftw_complex
- fftw_plan

Allocate and deallocate data

- fftw_malloc
- fftw_free

FFT plan and execution

- FFT plan fuctions (see next pages)
- fftw_execute // execute FFT plan
- fftw_destroy_plan

# FFTw plan functions I

❑ Complex DFT: $$X_k \stackrel{\text{def}}{=} \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}, \quad k \in \mathbb{Z}$$

❑ Inverse Complex DFT: $$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i2\pi kn/N}, \quad n \in \mathbb{Z}$$

One dimensional

- fftw_plan_dft_1d(int n, fftw_complex *in, fftw_complex *out, int sign, unsigned flags);
  sign: either FFTW_FORWARD (-1) or FFTW_BACKWARD (+1).
  flags: either FFTW_MEASURE or FFTW_ESTIMATE

Multi dimensional

- fftw_plan_dft_2d // two dimensions
- fftw_plan_dft_3d // three dimensions
- fftw_plan_dft // arbitrary dimensions

# FFTw plan functions II

Real DFTs

- fftw_plan_r2r_1d(int n, double *in, double *out, fftw_r2r_kind kind, unsigned flags)

  kind: FFTW_REDFT00, FFTW_RODFT00, etc. For different types of even or odd transforms.
- fftw_plan_r2r_2d, fftw_plan_r2r_3d, fftw_plan_r2r

Real input, complex output, always FFTW_FORWARD

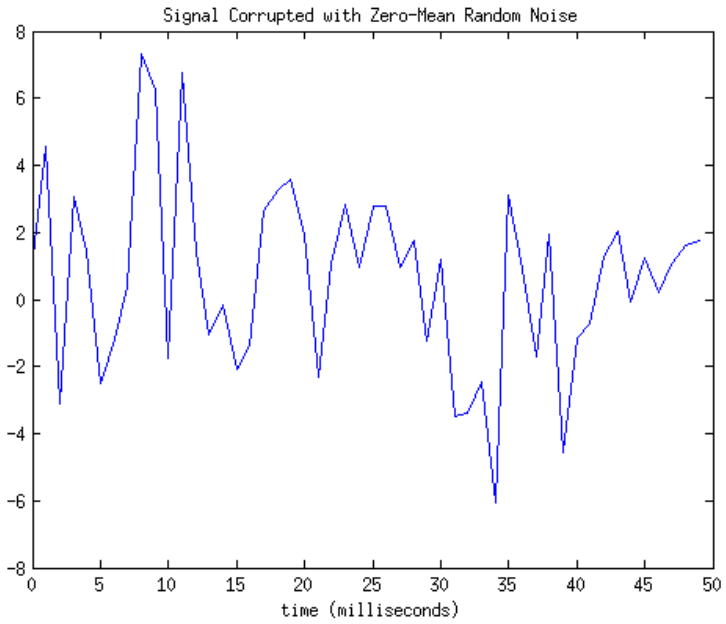
- fftw_plan_dft_r2c_1d, fftw_plan_dft_r2c_2d
- fftw_plan_dft_r2c_3d, fftw_plan_dft_r2c

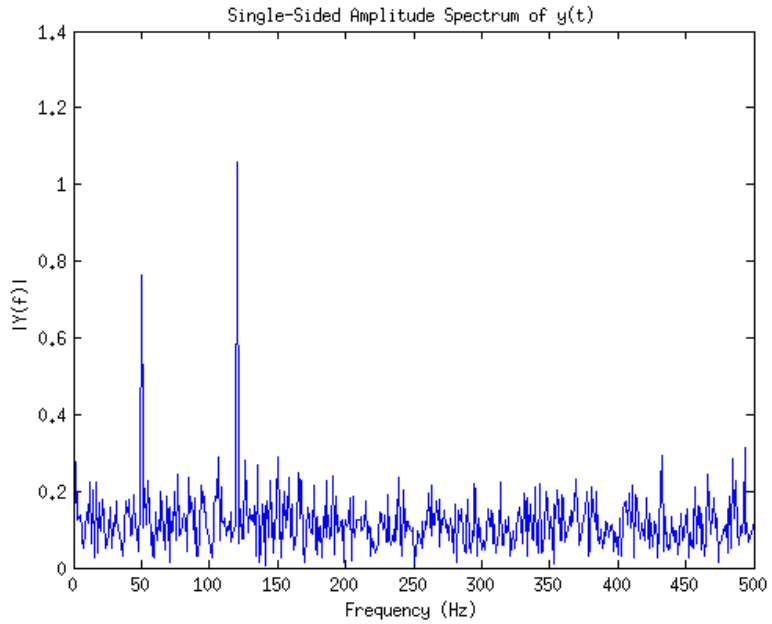
Complex input, real output, always FFTW_BACKWARD

- fftw_plan_dft_c2r_1d, fftw_plan_dft_c2r_2d
- fftw_plan_dft_c2r_3d, fftw_plan_dft_c2r

# Exercise 1: Fourier transform with FFTw

- Task: Compute the Fourier transform of a one-dimensional complex array, and compute the inverse Fourier transform of the output, which should be the same as the original input data.



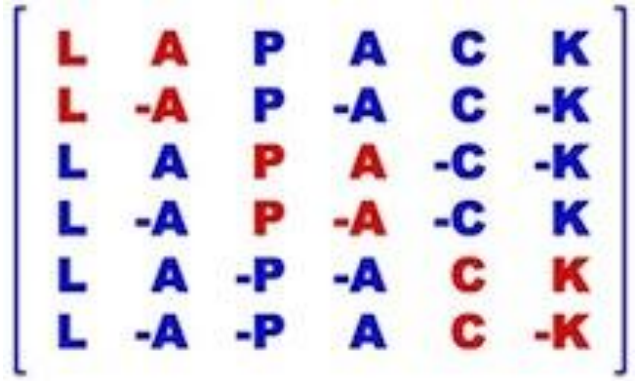
# Solution for Exercise 1

- Source code at /home/shaohao/hpc_train/numlibs/fftw/fftw3_prb.c
- Include fftw head file: # include <fftw3.h>
- Call fftw functions: fftw_plan_dft_1d, fftw_execute, etc.

- Compile and run

```
module load fftw/3.3.3/INTEL-140-MVAPICH2-2.0      # load fftw by moudle
module show fftw/3.3.3/INTEL-140-MVAPICH2-2.0      # show fftw-related environments
icc -c -I${LHPC_ROOTFFTW}/include fftw3_prb.c      # compile
icc fftw3_prb.o -lfftw3 -o fftw3_prb               # link
./fftw3_prb                                        # run
```

# 3. Linear Algebra libs




History:

- LINPACK (LINear algebra PACKage): since 1974

  based on level-1 BLAS
- LAPACK (Linear Algebra PACKage): since 1989

  based on level-3 BLAS, vectorized and threaded in Intel MKL
- ScaLAPACK (Scalable LAPACK): since 1995

  parallel with MPI, for distributed memory, only a subset of LAPACK functions
- DPLASMA (Distributed Parallel Linear Algebra Software for Multicore Architectures): 2000's

  parallel for shared memory
- MAGMA (Matrix Algebra for GPUs and Multicore Architectures): 2000's

  parallel for GPU

- Matlab: a commercial software developed from LINPACK.

# LAPACK

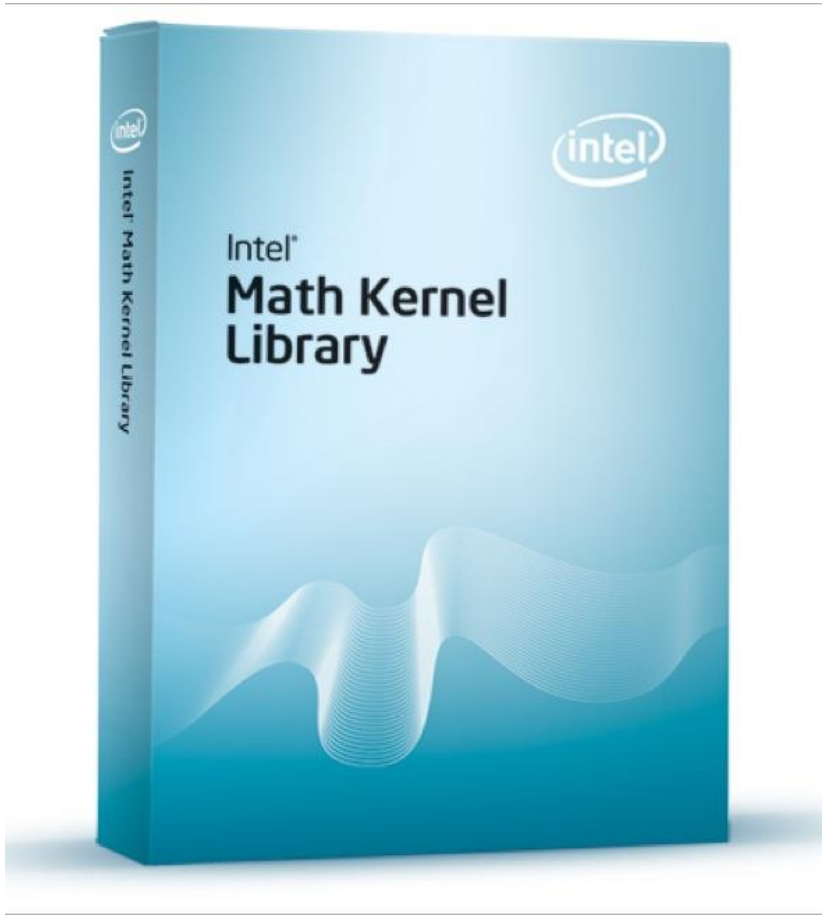
- provides routines for solving systems of linear equations, linear least squares, eigenvalue problems, and singular value decomposition.
- also includes routines to implement the associated matrix factorizations such as LU, QR, Cholesky and Schur decomposition.
- was originally written in FORTRAN 77, but moved to Fortran 90 in version 3.2 (2008).
- can be seen as the successor to the linear equations and linear least-squares routines of LINPACK and the eigenvalue routines of EISPACK.

## ❑ Usage of LAPACK

```
module load lapack/3.4.2/INTEL-140-MVAPICH2-2.0
icc -llapack name.c -o name
ifort –llapack name.f -o name
```

# Intel Math Kernel Library (MKL)

- Optimization for intel processors.
- accelerates math processing routines that increase application performance and reduce development time.
- includes highly vectorized and threaded Lapack, FFT, Vector Math and Statistics functions.
- Xeon-phi enabled.



# MKL LAPACK subroutines I

| Routine | Description |
|---|---|
| ?geev | Computes the eigenvalues and, optionally, the left and/or right eigenvectors of a general matrix. |
| ?gels | Uses *QR* or *LQ* factorization to solve an overdetermined or underdetermined linear system with a full rank matrix. |
| ?gelsd | Computes the minimum norm solution to a linear least squares problem using the singular value decomposition of *A* and a divide and conquer method. |
| ?gesdd | Computes the singular value decomposition of a general rectangular matrix using a divide and conquer algorithm. |
| ?gesv | Computes the solution to the system of linear equations with a square matrix $A$ and multiple right-hand sides. |
| ?gesvd | Computes the singular value decomposition of a general rectangular matrix. |
| ?heev | Computes all the eigenvalues and, optionally, the eigenvectors of a Hermitian matrix. |
| ?heevd | Computes all the eigenvalues and, optionally, all the eigenvectors of a complex Hermitian matrix using a divide and conquer algorithm. |

? could be: s – single precision;  d – double precision; c – single-precision complex; z – double-precision complex.

# MKL LAPACK subroutines II

| | |
|---|---|
| ?heevr | Computes the selected eigenvalues and, optionally, the eigenvectors of a Hermitian matrix using the Relatively Robust Representations. |
| ?heevx | Computes the selected eigenvalues and, optionally, the eigenvectors of a Hermitian matrix. |
| ?hesv | Computes the solution to the system of linear equations with a Hermitian matrix $A$ and multiple right-hand sides. |
| ?posv | Computes the solution to the system of linear equations with a symmetric or Hermitian positive definite matrix $A$ and multiple right-hand sides. |
| ?syev | Computes all the eigenvalues and, optionally, the eigenvectors of a real symmetric matrix. |
| ?syevd | Computes all the eigenvalues and, optionally, all the eigenvectors of a real symmetric matrix using a divide and conquer algorithm. |
| ?syevr | Computes the selected eigenvalues and, optionally, the eigenvectors of a real symmetric matrix using the Relatively Robust Representations. |
| ?syevx | Computes the selected eigenvalues and, optionally, the eigenvectors of a symmetric matrix. |
| ?sysv | Computes the solution to the system of linear equations with a real or complex symmetric matrix $A$and multiple right-hand sides. |

# Exercise 2: Solve a linear system with LAPACK subroutines in MKL

❑ Task: Compute the solution to the system of linear equations AX=B with a square matrix A and multiple right-hand sides B.

❑ Solution: source code at /home/shaohao/hpc_train/numlibs/mkl/dgesv_ex.c

- Call the MKL LAPACK function: dgesv

❑ Compile and run

```
icc -mkl dgesv_ex.c -o dgesv_ex
./dgesv
```

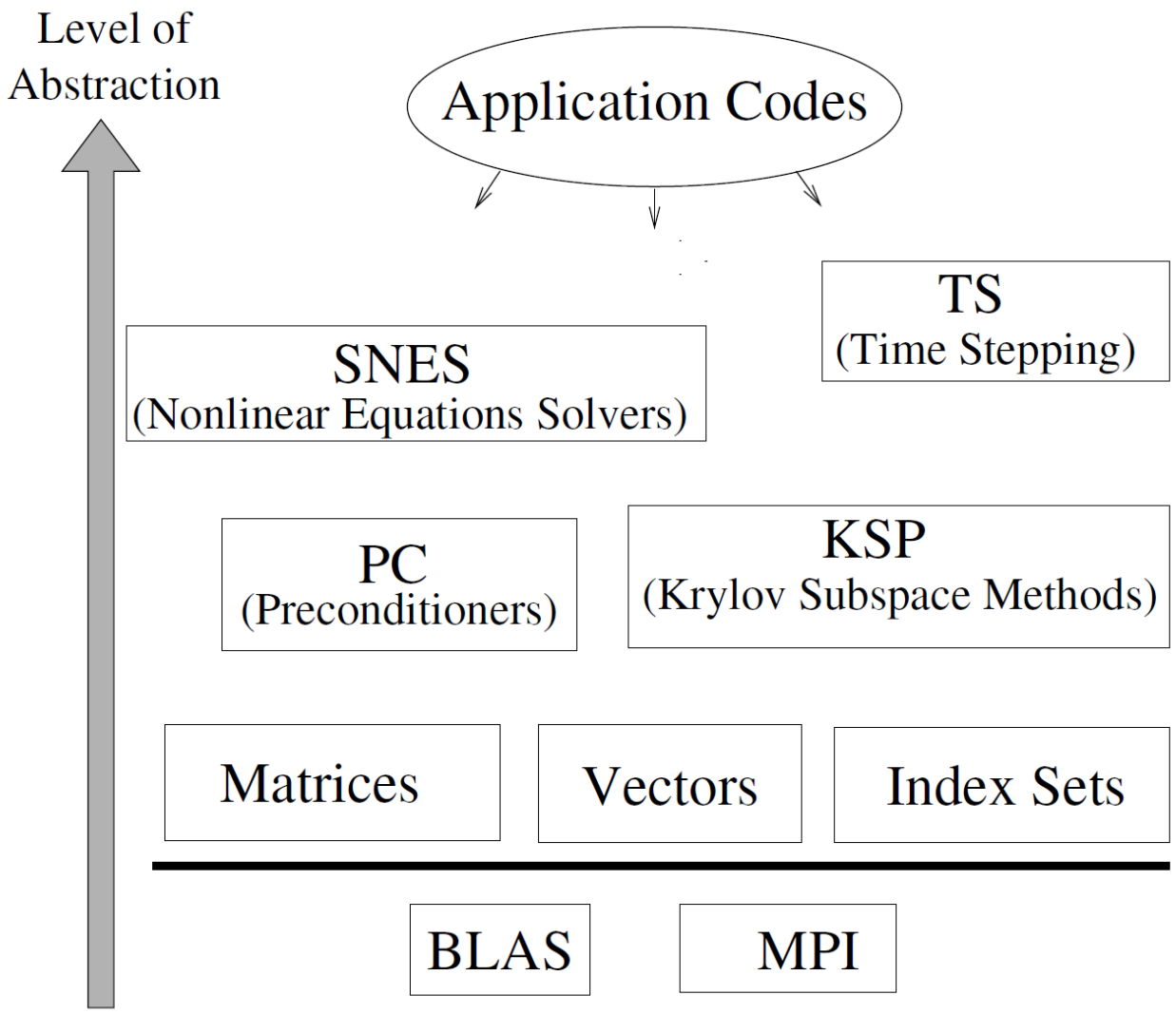
# 4. PETSc

- PETSc, pronounced PET-see (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations.
- It supports MPI, shared memory pthreads, and GPUs through CUDA or OpenCL, as well as hybrid MPI-shared memory pthreads or MPI-GPU parallelism.
- Efficient for sparse-matrix problems

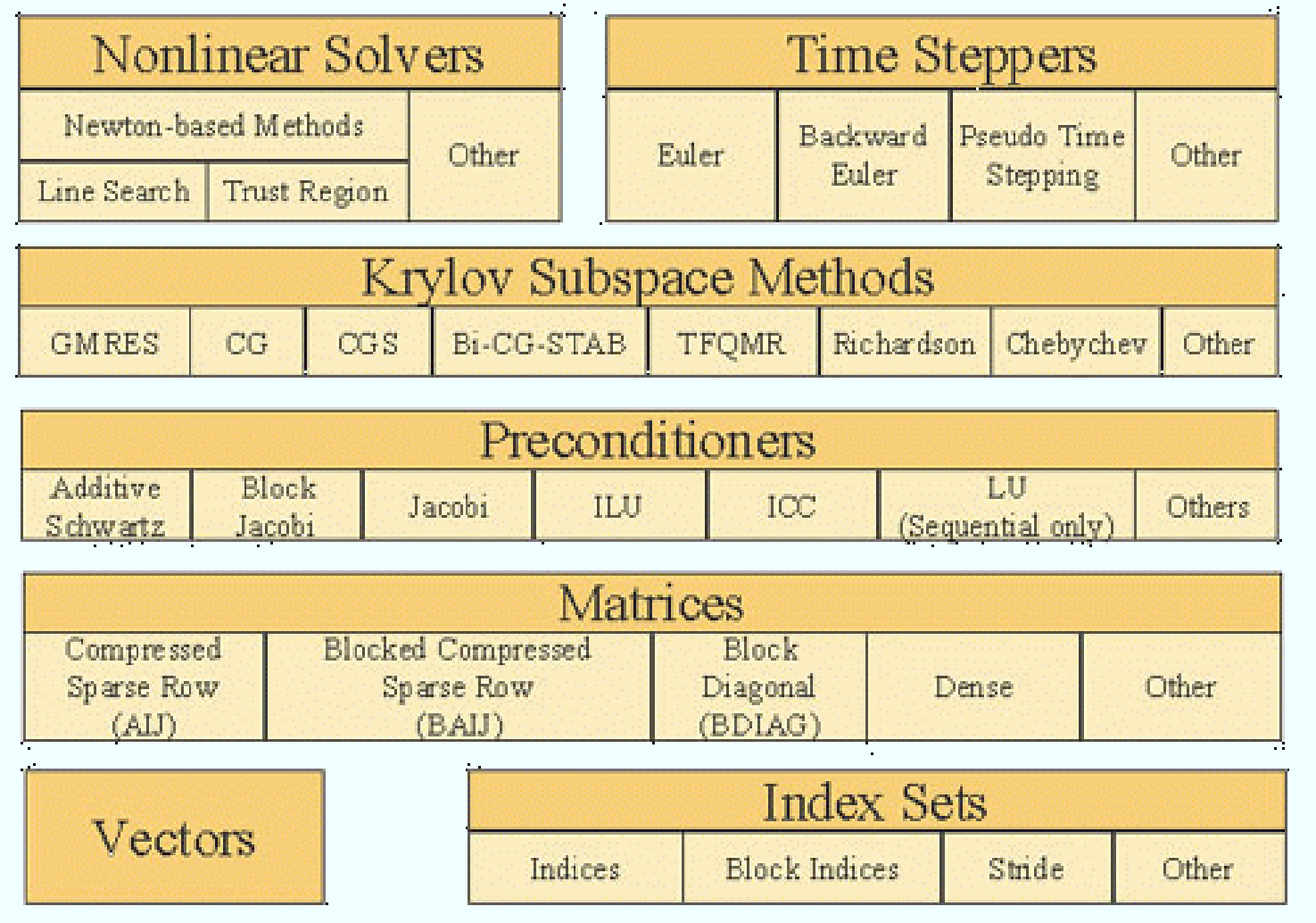
Main contents:

- distributed data structure: vectors, matrices, index sets
- vector operations, matrix operations
- Krylov subspace solver (KSP): solve linear systems, Laplacian solver, etc.
- SNES: Nonlinear Solvers
- TS: Scalable ODE and DAE Solvers

# Organization of the PETSc Libraries



# Parallel Numerical Components of PETSc



# PETSc Basics I

- PetscInitialize // call MPI_Initialized
- PetscFinalize // call MPI_Finalize

- Data types:

  PetscInt, PetscScalar, Vec, Mat

- Create objects:

  VecCreate(MPI_Comm comm, Vec *vec)

  MatCreate(MPI_Comm comm, Mat *mat)

- Destroy objects

  VecDestroy(Vec *vec)

  MatDestroy(Mat *mat)

# PETSc Basics II

- Set sizes of objects

  VecSetSizes(Vec v, PetscInt n, PetscInt N)  //  local size n, global size N

  MatSetSizes(Mat A, PetscInt m, PetscInt n, PetscInt M, PetscInt N)  // local size m, n, global size M, N

- Set values of objects

  VecSetValues(Vec x, PetscInt ni, const PetscInt ix[], const PetscScalar y[], InsertMode mode)

  MatSetValues(Mat mat, PetscInt m, const PetscInt idxm[], PetscInt n, const PetscInt idxn[], const PetscScalar v[], InsertMode mode)  // Set values of a block. Unset blocks are filled with zero.

  mode: either INSERT_VALUES or ADD_VALUES

# PETSc Basics III

- Assembly

  VecAssemblyBegin(Vec vec)

  VecAssemblyEnd(Vec vec)

  MatAssemblyBegin(Mat mat, MatAssemblyType type)

  MatAssemblyEnd(Mat mat, MatAssemblyType type)

  type: either MAT_FLUSH_ASSEMBLY or MAT_FINAL_ASSEMBLY

  Vector and matrix are ready to use only after the assembly functions have been called.

- Vector operations (see next slides)
- Matrix operations (see next slides)

- PETSc documentation: http://www.mcs.anl.gov/petsc/documentation/index.html

| 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); | $x_i = \alpha$ |

# PETSc vector operations

# PETSc 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); | $Y = Y + a * I$ |

# PETSc Krylov subspace solver

- KSP: Krylov subspace solver
- PC: preconditioner

Basic KSP functions:

- KSPCreate(MPI_Comm comm, KSP *ksp)
- KSPSetOperators(KSP ksp, Mat Amat, Mat Pmat) // assign the linear system to a KSP solver
- KSPSetType(KSP ksp, KSPType type) // KSP type: see next slides
- KSPGetPC(KSP ksp, PC *pc)
- PCSetType(PC pc, PCType type) // PC type: see next slides
- KSPSetTolerances(KSP ksp, PetscReal rtol, PetscReal abstol, PetscReal dtol, PetscInt maxits)
- KSPDestroy(KSP *ksp)

# PETSc KSP types

| Method | KSPType | Options Database Name |
|---|---|---|
| Richardson | KSPRICHARDSON | richardson |
| Chebyshev | KSPCHEBYSHEV | chebyshev |
| Conjugate Gradient [12] | KSPCG | cg |
| BiConjugate Gradient | KSPBICG | bicg |
| Generalized Minimal Residual [16] | KSPGMRES | gmres |
| Flexible Generalized Minimal Residual | KSPFGMRES | fgmres |
| Deflated Generalized Minimal Residual | KSPDGMRES | dgmres |
| Generalized Conjugate Residual | KSPGCR | gcr |
| BiCGSTAB [19] | KSPBCGS | bcgs |
| Conjugate Gradient Squared [18] | KSPCGS | cgs |
| Transpose-Free Quasi-Minimal Residual (1) [8] | 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 |

# PETSc PC types

| 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 |
| Algebraic Multigrid | PCGAMG | gamg |
| 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 |

# Exercise 3: Solves a linear system in parallel with PETSc

❑ Task: Compute the solution of a sparse-matrix linear system Ax=b, using a KSP solver (e.g. MINRES).

❑ Solution: C source code at /home/shaohao/hpc_train/numlibs/petsc/ex42.c

- Include petsc head file: #include <petscksp.h>
- Call petsc functions: KSPSetOperators, KSPSolve, KSPSetType, etc.

❑ Compile and run

- module load mvapich2/2.0/INTEL-14.0.2 # set up MPI
- module load petsc/3.5.0-real/INTEL-140-MVAPICH2-2.0 # set up PETSc
- make ex42 # compile and link
- mpirun -n 20 ./ex42 -m 120 # run job on a single node
- mpirun_rsh -n $NPROC -hostfile $PBS_NODEFILE ./ex42 -m 120 # run job on multi nodes

# PETSc-dependent pakcages

- SLEPc:

  Scalable Library for Eigenvalue Problems

- MOOSE:

  Multiphysics Object-Oriented Simulation Environment finite element framework, built on top of libMesh and PETSc

For more:

http://www.mcs.anl.gov/petsc/index.html

http://www.mcs.anl.gov/petsc/publications/index.html

# 5. GNU Scientific Libs: GSL

Main features:

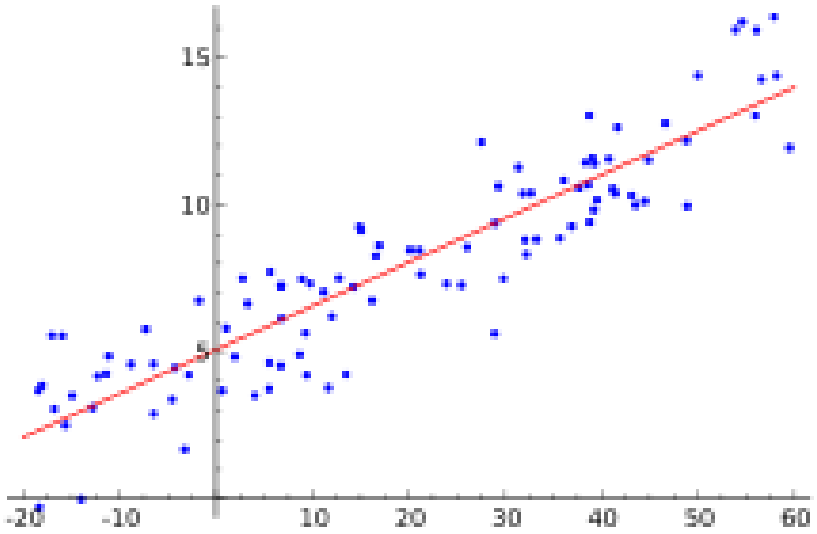
- A numerical library for C and C++ programmers
- Provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting
- Uses an object-oriented design. Different algorithms can be plugged-in easily or changed at run-time without recompiling the program.
- It is intended for ordinary scientific users. Anyone who knows some C programming will be able to start using the library straight-away.
- Serial

# Complete GSL subjects

- Mathematical Functions
- Complex Numbers
- Polynomials
- Special Functions
- Vectors and Matrices
- Permutations
- Combinations
- Multisets
- Sorting
- BLAS Support
- Linear Algebra
- Eigensystems
- Fast Fourier Transforms
- Numerical Integration
- Random Number Generation
- Quasi-Random Sequences
- Random Number Distributions
- Statistics
- Histograms
- N-tuples
- Monte Carlo Integration
- Simulated Annealing
- Ordinary Differential Equations
- Interpolation
- Numerical Differentiation
- Chebyshev Approximations
- Series Acceleration
- Wavelet Transforms
- Discrete Hankel Transforms
- One dimensional Root-Finding
- One dimensional Minimization
- Multidimensional Root-Finding
- Multidimensional Minimization
- Least-Squares Fitting
- Nonlinear Least-Squares Fitting
- Basis Splines
- Physical Constants

# Exercise 4: Computes linear fit with GSL

- Task: computes a least squares straight-line fit to a simple dataset, and outputs the best-fit line and its associated one standard-deviation error bars.



# Solution for Exercise 4

❑ C source code at /home/shaohao/hpc_train/numlibs/gsl/linear_fit.c

- Include gsl head file: #include <gsl/gsl_fit.h>
- Call gsl function: gsl_fit_linear_est

❑ Compile and run

```
module load gsl/1.16/INTEL-14.0.2      # set up gsl environmens
module show gsl/1.16/INTEL-14.0.2      # show gsl environments
icc -c -I${LHPC_ROOTGSL}/include linear_fit.c    # compile
icc linear_fit.o -lgsl -o linear_fit           # link
./linear_fit                                   # run
```

LSU INFORMATION TECHNOLOGY SERVICES

# Next training: Introduction to R

## March 18, 2015

## 9:30 AM - 11:30 AM

- R is a free software environment for statistical computing and graphics.