Compiler optimization and libraries

Bhupender Thakur, Jim Lupo, Le Yan, Alex Pacheco

Fortran Compilers

Compiler	PGI	Intel	GNU	Cray	IBM
command	pgif90, pgfortran	ifort	g77 gfortran,	ftn	xlf90, xlf2003
options	pgf90 -help	ifort –help	gfortran –help	ftn -help	xlf90 –help
Available on	LONI, HPC	LONI, HPC	LONI, HPC	--	Pandora

Optimization flags

- Generate debug info
  - -g, -O0, -traceback
  - check array bounds
  - info and warnings
- Fast optimizations,
  - may relax precision, compliance
  - –O1, –O2, –O3
  - –fast
  - loop unroll
- Instrumentation and profiling
  - few compilers support it

Libraries

What are they useful for?

- Numerics
- Visualization
- I/O
- Profiling

Where to find them?

- Netlib
- DOE ACTS collection
- ORNL, CERN, NIST, JPL
- Vendor libraries

Broswse the list of libraries on any supercomputer/cluster

: softenv, module avail

Libraries to be covered

- **BLAS/LAPACK**: Linear algebra library
- **ARPACK**: Large eigen value problem
- **PETSc**: Portable, Extensible Toolkit for Scientific Computation
- **NetCDF**: Machine-independent data library

Libraries
General features

Know the library location

- Lib_DIR=\root\some\where

Look for {lib}, {include}, {bin} subdirectories

- Static : $(Lib_DIR)\lib\libsome.a
- Dynamic: $(Lib_DIR)\lib\libsomeso
- Include files: $(Lib_DIR)\include

Link your executable when compiling

- $(F90) program.f90 \
  –I $(Lib_DIR)\include -L$(Lib_DIR)\lib -lsome

BLAS

Basic Linear Algebra Subprograms

- Available from: www.netlib.org/blas/
- Three levels of BLAS:
  - Vector operations, scalar products and norms
  - Matrix-vector operations: y=A*x + c
  - Matrix-matrix operations: A=B*C + D*y
- Usually ships with LAPACK

LAPACK
Linear Algebra PACKage

- Routines for
  - Solving systems of linear equations and linear least squares,
  - Eigenvalue problems,
  - Singular value decomposition.
- Important implementations:
  - Netlib.org/lapack, Intel MKL, ATLAS, LACAML, CLAPACK, SciPy, PLASMA, MAGMA,…

LAPACK

Naming convention: **X-YY-ZZZ**

X

- S: Single
- D: Double
- C: Complex

YY

- DI: Diagonal
- SY: Symmetric
- GE: General

ZZZ

- LQF:
  LQ factorization
- EGR:
  Few eigenvalues
- TRD:
  Tridiagonal reduction

Example

Calculate the eigenvalue and eigenvectors of A(n,n):

Call DSYEVD('V', 'U' ,N, A, N, w, work, lwork, iwork, liwork, info)

'V'	Compute eigenvalue and eigenvectors;
'U'	Upper triangle of mat is stored;
A	N*N matrix on (input) and eigenvectors on (output);
w	Array of (output) eigenvalue;
work	Real (workspace/output) of dimension lwork;
iwork	Integer (workspace/output) of dimension liwork;
info	(Output) flag for successful run.

Example (cont.)

```
program ex_dsyev
  use global
…
  allocate( w (n), &
            work (1000), &
             iwork(1000) )
…
Call DSYEV('V', 'U', n, mat, n, w, &
 work, 1000, iwork,1000,info)
…
```

Link to lapack and blas:

On LONI add following to the soft file

```
+lapack-3.2-intel-11.1

DIR=/usr/local/packages/lapack/3.2/
intel-11.1
LIBS= -L$(DIR)/lib -llapack  -lblas

ifort global.f90 ex_dsyev.f90 $(LIBS)
```

Intel MKL

Fast LAPACK by Intel

LDIR = /usr/local/compilers/Intel/mkl-10.2.2.025/lib/em64t

Dynamic linking

LIBS=-shared-intel -Wl, -rpath,$(LDIR) -L$(LDIR) -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -lguide –lpthread

Static linking

LIBS=-shared-intel -Wl,--start group $(LDIR)/libmkl_intel_lp64.a $(LDIR)/libmkl_intel_thread.a $(LDIR)/em64t/libmkl_core.a -Wl,--end-group -lguide -lpthread

ARPACK

Large sparse eigenvalue problem

- LAPACK requires as input A(N, N)
- Consider N=1,000,000

  Storage = $(10^{12} * 4)/(1024)^3$ = 3725 GB

- Fortunately,
  - Matrices are usually sparse
  - Iterative algorithms need not store full matrix
  - +arpack-96-intel-11.1 on LONI

Sparse matrices

Sparse matrix formats store non-zero elements and arrays referencing them

- *Coordinate format*

  idx(k), jdx(k), r(k)

1	3	0
2	4	6
0	5	7

- *Matvec operation (v=Au)*

  v(idx(k)) = v(idx(k))
  + u(jdx(k))*r(k)

idx= [1 2 1 2 3 2 3]
jdx= [1 1 2 2 2 3 3]
r = [1 2 3 4 5 6 7]

ARPACK

Large sparse eigenvalue problem

```
progran test_lanczos

use modarpack
use sparpack

…

    which_eig          =  "LM"
    n_eigvalues        =  2
    tolerance          = .01d0
    get_vectors        = .true.
    i_guesspivot       = 0
…
```

Code is a modified example from arpack
examples/SYM/dsdrv1.f

What spectrum of eigenvalues is requested? Largest magnitude

Number of eigenvalues

Accuracy of eigenvalues

Need eigenvectors as well?

ARPACK

Large sparse eigenvalue problem

Reverse interface for user matvec

```
call dsaupd ( ido, bmat, n, which, nev, tol, resid, &
              ncv, v, ldv, iparam, ipntr, workd, workl, &
              lworkl, info )

if (ido .eq. -1 .or. ido .eq. 1) then

   %-----------------------------------%
   | Perform matrix vector multiplication |
   |              y <--- OP*x          |
   | The user should supply his/her own |
   | matrix vector multiplication routine |
   | here that takes workd(ipntr(1)) as |
   | the input, and return the result to |
   | workd(ipntr(2)).                  |
   %-----------------------------------%

   call av (nx, workd(ipntr(1)), workd(ipntr(2)))
```

User supplied matrix-vector multiplication

LSU

ARPACK

Large sparse eigenvalue problem

```
#!/bin/bash

arpack_dir=/usr/local/packages/arpack/96/intel-11.1/lib
lapack_dir=/usr/local/packages/lapack/3.2/intel-11.1/lib

ifort -o cpu.exe \
      \
        sparpack.f90 modarpack.f90 test_lanczos.f90 \
      \
       -L${arpack_dir} -larpack_LINUX \
      \
       -L${lapack_dir} -llapack -lblas
```

ARPACK

Large sparse eigenvalue problem

```
progran test_lanczos

use modarpack
use sparpack

…

   which_eig          =  "LM"    ! Which ?
   n_eigvalues        =  2       ! Eigenvalues
   tolerance          = .01d0    ! Tolerance
   get_vectors        = .true.   ! Eigenvectors
   i_guesspivot       = 0        ! Guess
…
```

```
[bthakur@eric2]$ ./cpu.exe

 E-vals  -6.38745949107957
 E-vals   6.07252812124572

 _SDRV1
 ======

 Size of the matrix is                51537
 Ritz values requested is             2
 Arnoldi vecs generated(NCV)          3
 Portion of the spectrum:             LM
 Number of converged values           2
 Implicit Arnoldi iterations          135
 The number of OP*x is                137
 The convergence criterion            0.01
```

More libraries

- **Parallel libraries beyond LAPACK/ARPACK are available**
  - **SLEPSc** Scalable Library for Eigenvalue Problem Computations
  - **Hypre** Solves large, sparse linear systems of equations on massivelly parallel computers
  - **Blopex** parallel preconditioned eigenvalue solvers
  - **Plasma** Parallel Linear Algebra Software for Multicore Architectures
  - **Scalapack** Scalable LAPACK

PETSc

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

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.

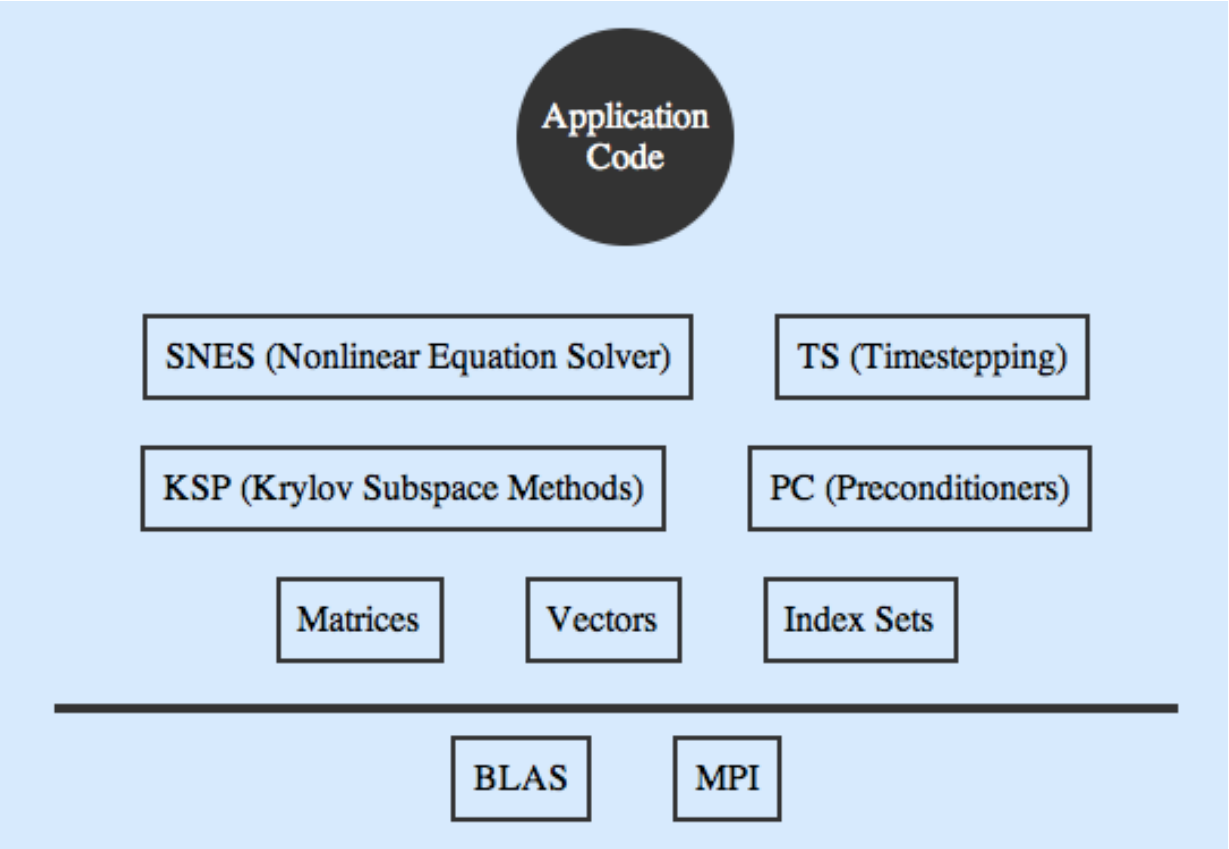
PETSc
Components

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
Components



LSU

PETSc

- **Initialize PETSc**

```
#include "finclude/petsc.h90”
#include "finclude/petscvec.h90”

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

Call PetscInitialize()
Call MPI_Comm_rank(PETSC_COMM_WORLD,rank,ierr)
```

PETSc

PETSc objects and procedures

VecCreateSeq()	VecScale(), VecNorm()
VecCreateMPI()	VecAXPY(), VecDot()

Example

```
Vec                          v
PetscScalar pointer ::       array(:,:)
PetscInt                     n, i
PetscErrorCode               ierr
call VecGetArrayF90(v, array, ierr)
call VecGetLocalSize(v, n, ierr)
do i=1,n
  array(i) = array(i) + rank
end do
```

PETSc

PETSc objects and procedures

MatCreate(MPI_Comm, Mat *)

MatSetSizes(Mat, PetscInt m, PetscInt n, M, N)

MatSetType(Mat, MatType typeName)

MatSetFromOptions(Mat)

Single user interface but multiple underlying implementations

PETSc

Higher abstractions

The PETSc DA class is a topology and discretization interface.

The PETSc Mesh class is a topology interface.

The PETSc DM class is a hierarchy interface.

The PetscSection class is a helper class for data layout.

Profiling and debugging

PETSc has integrated profiling, logging events.

Higher level of error detection

PETSc
Example

```
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
```

PETSc
Example

 # Makefile

```
PETSC_DIR = /usr/local/packages/petsc/3.0.0.p3/intel-11.1-mpich-1.2.7p1

include ${PETSC_DIR}/conf/base

ex1f: ex1f.o  chkopts
          -${FLINKER} -o ex1f ex1f.o ${PETSC_VEC_LIB}
           ${RM} -f ex1f.o
```

NetCDF

- **What is NetCDF?**
  - NetCDF is a set of data formats, programming interfaces, and software libraries that help read and write scientific data files.
- **The Classic NetCDF Data Model**
  - The classic netCDF data model consists of variables, dimensions, and attributes. This way of thinking about data was introduced with the very first netCDF release, and is still the core of all netCDF files.

NetCDF

- **The Classic NetCDF Data Model**
  - ***Variables***: N-dimensional arrays of data. Variables in netCDF files can be one of six types (char, byte, short, int, float, double)
  - ***Dimensions*** describe the axes of the data arrays. A dimension has a name and a length. An unlimited dimension has a length that can be expanded at any time. NetCDF files can contain at most one unlimited dimension.
  - ***Attributes*** annotate variables or files with small notes or supplementary metadata. Attributes are always scalar values or 1D arrays, which can be associated with either a variable or the file.

NetCDF
Example

```
program simple_xy_wr
  use netcdf
  implicit none
! Name of the data file
  character (len = *), parameter ::
FILE_NAME = "simple_xy.nc”
! We are writing 2D data, a 6 x 12 grid.

integer, parameter :: NDIMS = 2
integer, parameter :: NX = 6, NY = 12
integer :: ncid, varid, dimids(NDIMS)
integer :: x_dimid, y_dimid

! This is the data array
 integer :: data_out(NY, NX)
```

NetCDF

```
! Create the netCDF file.
  call check( nf90_create(FILE_NAME,
                NF90_CLOBBER, ncid) )

! Define dimensions. It hands ID for each.
  call check( nf90_def_dim(ncid, "x", NX,
                x_dimid) )

  call check( nf90_def_dim(ncid, "y", NY,
                y_dimid) )
  dimids =  (/ y_dimid, x_dimid /)

! Define the variable ttype: NF90_INT
  call check( nf90_def_var(ncid, "data",
                NF90_INT, dimids, varid) )

  call check( nf90_put_var(ncid, varid,
  data_out) )

  call check( nf90_close(ncid) )

  end
```

Summary

- We reviewed some basic libraries, which form the core of many computational algorithms.
- Hopefully, through this tutorial, you have learnt how to use libraries to write more efficient programs.