UNO Computer Science

Introduction to LAPACK

Zhiyu Zhao (sylvia@cs.uno.edu)
The LONI Institute
&
Department of Computer Science
College of Sciences
University of New Orleans
03/16/2009

Outline

- What is LAPACK
  - Linear Algebra PACKage
  - Problems Solved by LAPACK
  - Matrices Handled by LAPACK
- Structure of LAPACK
  - Driver Routines
  - Computational Routines
  - Auxiliary Routines
  - LAPACK Naming Scheme

Outline (continued)

- Use LAPACK with Your Program
  - Availability of LAPACK on LONI Clusters
  - Get Information about a Routine on a Cluster
  - Use LAPACK Routines in Your Fortran Program
  - Use LAPACK Routines in Your C Program
  - Use LAPACK Routines in Your C++ Program
- Parallel and Distributed Programming with LAPACK
  - Multithreaded LAPACK
  - ScaLAPACK: Scalable LAPACK
  - Software Hierarchy
  - Intel MKL

What is LAPACK

Linear Algebra PACKage

- A free package of linear algebra subroutines written in Fortran
- Latest version: 3.2 (Nov. 18, 2008)
- Website: http://www.netlib.org/lapack/

What is LAPACK

Problems Solved by LAPACK

- Systems of linear equations
- Linear least squares problems
- Eigenvalue problems
- Singular value problems
- Associated computations
  - Matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur)
  - Reordering of the Schur factorizations
  - Estimating condition numbers
  - …

What is LAPACK

Matrices Handeled by LAPACK

- Dense and band matrices (not general sparse matrices)
- Real and complex matrices
- Single and double precision matrices

Structure of LAPACK

Driver Routines

- Each solves a complete problem and calls a sequence of computational routines
- Problems solved
  - Linear Equations
  - Linear Least Squares (LLS) Problems
  - Generalized Linear Least Squares (LSE and GLM) Problems
  - Standard Eigenvalue and Singular Value Problems
  - Generalized Eigenvalue and Singular Value Problems

For a complete list of driver routines, visit http://www.netlib.org/lapack/lug/node25.html .

Structure of LAPACK

Computational Routines

- Each performs a distinct computational task
- Use them when driver routines are not the best choice
- Problems solved
  - Linear Equations
  - Orthogonal Factorizations and Linear Least Squares Problems
  - Generalized Orthogonal Factorizations and Linear Least Squares Problems
  - Symmetric Eigenproblems

Structure of LAPACK

Computational Routines

- Problems solved (continued)
  - Nonsymmetric Eigenproblems
  - Singular Value Decomposition
  - Generalized Symmetric Definite Eigenproblems
  - Generalized Nonsymmetric Eigenproblems
  - Generalized (or Quotient) Singular Value Decomposition

For a complete list of computational routines, visit http://www.netlib.org/lapack/lug/node37.html .

Structure of LAPACK

Auxiliary Routines

- Routines that subtasks of block algorithms
- Routines that perform some commonly required low-level computations
- A few extensions to the BLAS (Basic Linear Algebra Subprograms)

For a complete list of auxiliary routines, visit http://www.netlib.org/lapack/lug/node144.html .

Structure of LAPACK

LAPACK Naming Scheme

- Each routine has a 6-character name
  - Some driver routines have 5 only (the 6th is blank)
- All driver and computational routines have names of the form **XYYZZZ**
  - X: Data type (S – single real, D – double real, C – single complex, Z – double complex)
  - YY: Matrix type (e.g. BD – bidiagonal, DI – diagonal)

    See http://www.netlib.org/lapack/lug/node24.html for a complete list of matrix types.
  - ZZZ: Computation performed (e.g. SVX – an expert driver which solves AX = B, QRF – QR factorization)

Use LAPACK with Your Program

Availability of LAPACK on LONI Clusters

- Software version: 3.1.1
- Installed on: Queen Bee, Louie, Eric, Poseidon, Oliver, and Painter
- To make sure LAPACK is installed on a cluster, logon that cluster and run the following command:

  $ *softenv* | *grep* lapack

  You should see one or more keys for LAPACK.

Use LAPACK with Your Program

Get Information about a Routine on a Cluster

- Logon a LONI cluster
- Run the following command

$ *man* routine_name # routine_name is the name of a LAPACK routine

e.g.

$ *man* dgesvd

Use LAPACK with Your Program

Use LAPACK Routines in Your Fortran Program

- Call routines as Fortran built-in functions

  e.g. CALL DGESV(N, NRHS, A, LDA, IPIV, B, LDB, INFO)

- Compile with the library **lapack**

  e.g. $ifort –llapack –o filename filename.f

Use LAPACK with Your Program

Use LAPACK Routines in Your C Program

- Routine must be declared with **extern**

  e.g. extern void dgetrf_(int*, int*, double*, int*, int*, int*);
- Arguments must be passed by reference
  - Pointers to variables instead of variable values
- Matrices must be transposed
  - In C matrices are stored in row major order
  - In Fortran matrices are stored in column major order
- Routine name is in lower case and followed by an '_'

  e.g. dgetrf_(&m, &n, (double*)A, &lda, IPIV, &info);
- Compile with the library **lapack**

  e.g. $icc –o filename filename.c –llapack

Use LAPACK with Your Program

Use LAPACK Routines in Your C++ Program

- All rules for C apply (see the previous slide) except that routine must be declared with **extern "C "**

  e.g.

  extern "C" dgetrf_(int*, int*, double*, int*, int*, int*);

- Compile with the library **lapack**

  e.g. $icpc –o filename filename.c –llapack

Use LAPACK with Your Program

Lab 1: Using LAPACK in Your Code

- Write a Fortran/C/C++ program which uses the LAPACK routine DGESV to solve a system of linear equations AX = B, where

$$A = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{matrix} \quad \text{and} \quad B = \begin{matrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{matrix}$$

  Hint: $*man* dgesv to get more information about this routine.

- Compile your code with **–llapack** .

Use LAPACK with Your Program

Answer to Lab 1 (Fortran)

```
      PROGRAM LapackLab1
c     ifort –o Lab1f Lab1.f -llapack
      INTEGER IPIV(3), info
      DOUBLE PRECISION A(3,3), B(3,2)
      A(1,1)=1
      A(1,2)=2
      …
      A(3,3)=10
c     Continued on next slide
```

Use LAPACK with Your Program

Answer to Lab 1 (Fortran continued)

```
    B(1,1)=1
    B(2,1)=0
    ...
    B(3,2)=0
    CALL DGESV(3, 2, A, 3, IPIV, B, 3, info)
c If DGESV is called successfully info should be 0.
    IF (info .EQ. 0) THEN
          DO i=1,3
                WRITE(*,'(2F8.3)') (B(i,j), j=1,2)
          ENDDO
    ENDIF
    END
```

Use LAPACK with Your Program

Answer to Lab 1 (C)

```
/* icc –o Lab1c Lab1.c –llapack*/
/* Routine must be declared with extern.*/
extern void dgesv_(int*, int*, double*, int*, int*, double*, int*,
    int*);
int main () {
    int n, nrhs, lda, ldb, IPIV[3], info
    double A[3][3], B[2][3];        /* Matrices must be transposed.*/
    A[0][0]=1; A[1][0]=2; … A[2][2]=10;
    B[0][0]=1; B[0][1]=0; … B[1][2]=0;
```

Use LAPACK with Your Program

Answer to Lab 1 (C continued)

```
/* Arguments must be passed by reference.*/
n=3; nrhs=2; lda=3; ldb=3;
/* Routine name is in lower case and followed by an
underscore    '_'   .*/
dgesv_(&n, &nrhs, A, &lda, IPIV, B, &ldb, &info);
/* Print the result. B should be transposed back.*/
/* If DGESV is called successfully info should be 0.*/
if (info==0)
        for (i=0; i<3; i++)
                printf("%8.3f %8.3f\n", B[0][i], B[1][i]);
}
```

Use LAPACK with Your Program

Answer to Lab 1 (C++)

```
// icpc –o Lab1cpp Lab1.cpp –llapack
#include <iostream>
#include <iomanip>
using namespace std;
// Routine must be declared with extern "C".
extern "C" void dgesv_(int*, int*, double*, int*, int*, double*, int*,
    int*);
int main () {
    int n, nrhs, lda, ldb, IPIV[3], info;
    double A[3][3], B[2][3];        // Matrices must be transposed.
    A[0][0]=1; A[1][0]=2; … A[2][2]=10;
    B[0][0]=1; B[0][1]=0; … B[1][2]=0;
```

Use LAPACK with Your Program

Answer to Lab 1 (C++ continued)

```
    // Arguments must be passed by reference.
    n=3; nrhs=2; lda=3; ldb=3;
// Routine name is in lower case and followed by an underscore
      ' _ '  .
    dgesv_(&n, &nrhs, (double*)A, &lda, IPIV, (double*)B, &ldb,
    &info);
    // Print the result. B should be transposed back.
    // If DGESV is called successfully info should be 0.
    if (info==0)
          for (i=0; i<3; i++)
                    cout << setprecision(3) << fixed << setw(8) <<
    B[0][i] << ' ' << setw(8) << B[1][i] << endl;
}
```

Use LAPACK with Your Program

Solution to the Linear Equations in Lab 1

$$X = \begin{matrix} -0.667 & -1.333 \\ -0.667 & 3.667 \\ 1.000 & -2.000 \end{matrix}$$

Parallel and Distributed Programming with LAPACK

Multithreaded LAPACK

- Reference LAPACK does not support multithreading
- Some vendor versions of LAPACK do
  - Intel Math Kernel Library (Intel MKL)
    http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm
  - AMD Core Math Library (ACML)
    http://developer.amd.com/cpu/Libraries/acml/Pages/default.aspx
  - …

Parallel and Distributed Programming with LAPACK

ScaLAPACK: Scalable LAPACK

- A free package of linear algebra subroutines written in Fortran
- Designed for distributed-memory message-passing MIMD computers and networks of workstations supporting PVM and/or MPI
- Website: http://www.netlib.org/scalapack/
- Latest version: 1.8.0 (Apr 5, 2007)

Note: Intel MKL provides ScaLAPACK subroutines.

Parallel and Distributed Programming with LAPACK

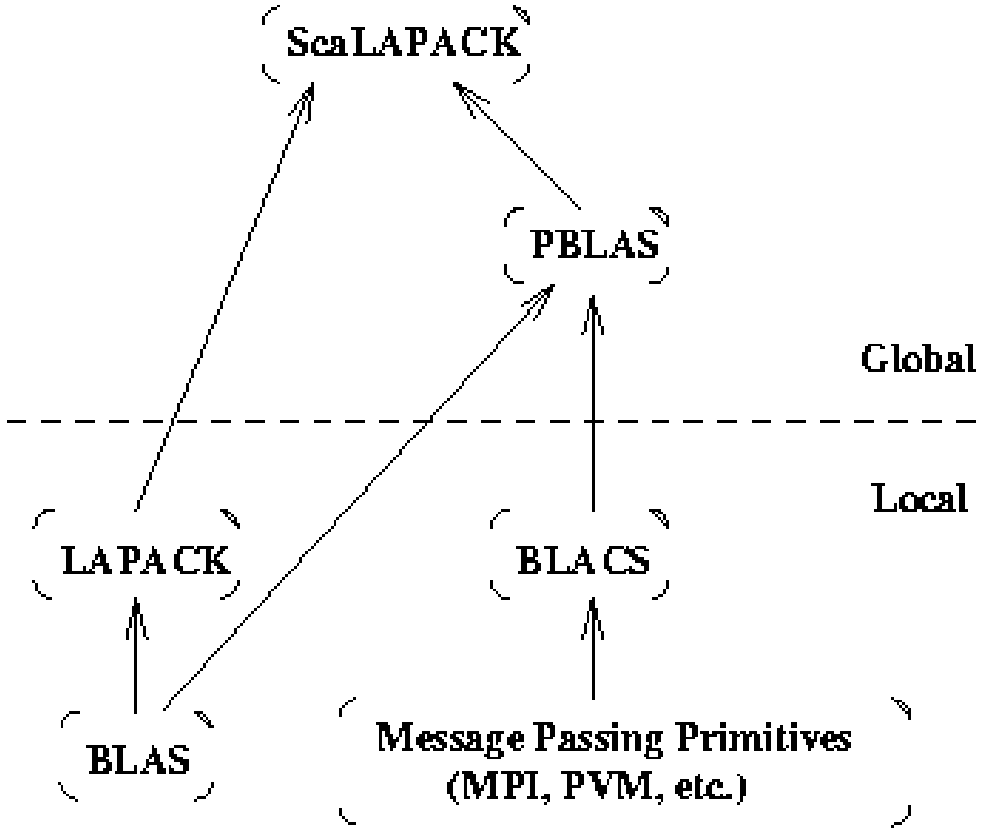
ScaLAPACK: Scalable LAPACK

- Each ScaLAPACK routine has a LAPACK equivalent
- Naming scheme: LAPACK name preceded by a ‘P’
- 4 basic steps required to call a ScaLAPACK routine
  - Initialize the process grid
  - Distribute matrices on the process grid
  - Call the ScaLAPACK routine
  - Release the process grid

For more information, view ScaLAPACK user’s guide at http://www.netlib.org/scalapack/slug/index.html .

Parallel and Distributed Programming with LAPACK

Software Hierarchy



Cited from http://www.netlib.org/scalapack/slug/node11.html .

Parallel and Distributed Programming with LAPACK

Software Hierarchy

- BLAS: Basic Linear Algebra Subprograms
  - Subroutines that provide standard building blocks for performing basic vector and matrix operations.
  - Used by LAPACK and PBLAS
  - Reference BLAS: a Fortran77 implementation
  - Website: http://www.netlib.org/blas/
  - Optimized BLAS libraries

    See http://www.netlib.org/blas/faq.html#5

Note: Intel MKL provides optimized BLAS subroutines.

Parallel and Distributed Programming with LAPACK

Software Hierarchy

- BLACS: Basic Linear Algebra Communication Subprograms
  - A linear algebra oriented message passing interface
  - Uses message passing primitives such as MPI and PVM
  - Used by PBLAS
  - Website: http://www.netlib.org/blacs/
- PBLAS: Parallel BLAS
  - Uses BLAS and BLACS
  - Used by ScaLAPACK

    See http://www.netlib.org/scalapack/slug/node14.html

Parallel and Distributed Programming with LAPACK

Intel MKL

- Intel ® Math Kernel Library
- Contains the complete set of functions from
  - BLAS / Sparse BLAS / CBLAS
  - LAPACK
  - ScaLAPACK
  - FFT
  - …
- Latest version: 10.1

  Website: http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm

Parallel and Distributed Programming with LAPACK

Intel MKL

- Multithreading implemented with OpenMP
  - Providing multithreaded BLAS and LAPACK routines
- Message passing implemented with MPI
  - Providing MPI based ScaLAPACK routines
- Availability on LONI clusters: Queen Bee, Eric, Louie, Poseidon, Oliver

For more information, view Intel MKL user's guide at http://www.intel.com/cd/software/products/asmo-na/eng/345631.htm .

Parallel and Distributed Programming with LAPACK

Intel MKL

- How to compile code with MKL on LONI clusters
  - Use MKL for multithreaded routines with OpenMP

$ ***compiler*** –openmp filename –L path_of_mkl_lib -lmkl

Note: ***compiler*** *is a Fortran/C/C++ compiler .*

e.g. on Queen Bee with Intel MKL 10.0 installed:

$ compiler –openmp filename -L /usr/local/compilers/Intel/mkl-10.0/lib/em64t -lmkl

  - Use MKL for ScaLAPACK routines with MPI & OpenMP

$ ***mpi_compiler*** –openmp filename –L path_of_mkl_lib -lmkl_scalapack_lp64 -lmkl_blacs_lp64 -lmkl_lapack –lmkl

Note: ***mpi_compiler*** *is a MPI Fortran/C/C++ compiler .*

Parallel and Distributed Programming with LAPACK

Lab 2: Using Intel MKL with Multithreaded LAPACK

- Write a Fortran/C/C++ program which uses the LAPACK routine DGETRF to compute the LU factorization of a matrix of2000*2000 random entries.
- Record the execution time of the DGETRF routine only (not including the time of generating random entries), and display that time.
- Compile your code on Queen Bee with –llapack and –openmp –lmkl, respectively, and observe the difference between the execution times.

Parallel and Distributed Programming with LAPACK

Answer to Lab 2 (Fortran)

```
      PROGRAM LapackLab2
c     ifort -o Lab2f Lab2.f -llapack
c     ifort -o Lab2f Lab2.f -L /usr/local/compilers/Intel/mkl-10.0/lib/em64t
     -lmkl -openmp
      USE IFPORT
      INTEGER IPIV(2000), info, i, j, start, end, rate, max, elapsed
      DOUBLE PRECISION A(2000,2000)
      DO i=1, 2000
            DO j=1, 2000
                  A(i,j)=RAND()
            ENDDO
      ENDDO
```

Parallel and Distributed Programming with LAPACK

Answer to Lab 2 (Fortran continued)

```
    CALL SYSTEM_CLOCK(start, rate, max)
    CALL DGETRF(2000, 2000, A, 2000, IPIV, info)
    CALL SYSTEM_CLOCK(end, rate, max)
    elapsed=(end-start)*1000/rate
c If DGETRF is called successfully info should be 0.
    IF (info .EQ. 0) THEN
          WRITE (*, '(A, I, A)') 'DGETRF is done. : ', elapsed, ' ms.'
    ENDIF
    END
```

Parallel and Distributed Programming with LAPACK

Answer to Lab 2 (C)

```
/* icc -o Lab2c Lab2.c –llapack*/
/* icc -o Lab2c Lab2.c -L /usr/local/compilers/Intel/mkl-10.0/lib/em64t -
    lmkl -openmp*/
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define M                    2000
#define N                    2000
extern void dgetrf_(int*, int*, double*, int*, int*, int*);
int main () {
    int m = M, n = N, lda = M, IPIV[N], info, i, j;
    double A[N][M];
    clock_t start, end;
```

Parallel and Distributed Programming with LAPACK

Answer to Lab 2 (C continued)

```
    for (i=0; i<n; i++)
            for (j=0; j<m; j++)
                    A[i][j]=(double)rand();
    start=clock();
    dgetrf_(&m, &n, (double*)A, &lda, IPIV, &info);
    end=clock();
/* If DGETRF is called successfully info should be 0.*/
    if (info==0)
            printf("dgetrf_ is done: %d ms.\n", (int)((end-
    start)*1000/CLOCKS_PER_SEC));
}
```

Parallel and Distributed Programming with LAPACK

Answer to Lab 2 (C++)

```
/* icpc -o Lab2cpp Lab2.cpp –llapack*/
/* icpc -o Lab2cpp Lab2.cpp -L /usr/local/compilers/Intel/mkl-
    10.0/lib/em64t -lmkl -openmp*/
#include <iostream>
#include <ctime>
using namespace std;
#define M                    2000
#define N                    2000
extern "C" void dgetrf_(int*, int*, double*, int*, int*, int*);
int main () {
    int m = M, n = N, lda = M, IPIV[N], info, i, j;
    double A[N][M];
    clock_t start, end;
```

Parallel and Distributed Programming with LAPACK

Answer to Lab 2 (C++ continued)

```
    for (i=0; i<n; i++)
            for (j=0; j<m; j++)
                    A[i][j]=(double)rand();
    start=clock();
    dgetrf_(&m, &n, (double*)A, &lda, IPIV, &info);
    end=clock();
/* If DGETRF is called successfully info should be 0.*/
    if (info==0)
            cout << "dgetrf_ is done: " << ((end-
    start)*1000/CLOCKS_PER_SEC) << " ms." << endl;
}
```

Parallel and Distributed Programming with LAPACK

Lab 3: Using Intel MKL with ScaLAPACK

- On Queen Bee, go to your work directory and download an example Fortran program from the official website of ScaLAPACK.

  $ wget http://www.netlib.org/scalapack/examples/example1.f

- Compile the example program

  $ mpif77 -openmp -o example1 example1.f -L /usr/local/compilers/Intel/mkl-10.0/lib/em64t -lmkl_scalapack_lp64 -lmkl_blacs_lp64 -lmkl_lapack –lmkl

Parallel and Distributed Programming with LAPACK

Lab 3: Using Intel MKL with ScaLAPACK

- Write a job submission script file and save it as example1.pbs

```
#!/bin/bash
#PBS -A your_allocation_name
#PBS -q checkpt
#PBS -l nodes=6:ppn=8
#PBS -l walltime=00:10:00
#PBS -o example1_output
#PBS -j oe
#PBS -N example1
mpirun -np 6 example1
```

Parallel and Distributed Programming with LAPACK

Lab 3: Using Intel MKL with ScaLAPACK

- Submit the job
- Wait for the job to be completed and check its output in the file example1_output
- You should see

```
ScaLAPACK Example Program #1 -- May 1, 1997

Solving Ax=b where A is a    9 by    9 matrix with a block size of    2
Running on    6 processes, where the process grid is    2 by    3

INFO code returned by PDGESV =    0

According to the normalized residual the solution is correct.

||A*x - b|| / ( ||x||*||A||*eps*N ) =    0.00000000E+00
```

Thank you!

Questions / Comments?