AMath 483/583 — Lecture 22

Outline:

- MPI Master–Worker paradigm
- Linear algebra
- LAPACK and the BLAS

References:

- `$UWHPSC/codes/mpi`
- class notes: MPI section
- class notes: Linear algebra

Another Send/Receive example

Computing the 1-norm of a matrix,

$$\|A\|_1 = \max_j \sum_i |a_{ij}| = \text{max of 1-norm of column vectors}$$

$$\implies \|Ax\|_1 \leq \|A\|_1 \|x\|_1 \text{ for all } x.$$

Sample codes are now in...

`$UWHPSC/codes/mpi/matrix1norm1.f90`:
Same number of processes as columns,

`$UWHPSC/codes/mpi/matrix1norm2.f90`:
Possibly more (or fewer) columns than processes.

The latter case shows the more typical situation where master process must send out new work as processes finish.

Can also view in class notes: MPI section

Another Send/Receive example

Master (Processor 0) sends jth column to Worker Processor j,
gets back 1-norm to store in `anorm(j)`, $j = 1, \ldots,$ `ncols`

```
! code for Master (Processor 0):
if (proc_num == 0) then

  do j=1,ncols
    call MPI_SEND(a(1,j), nrows, MPI_DOUBLE_PRECISION,&
                   j, j, MPI_COMM_WORLD, ierr)
    enddo

  do j=1,ncols
    call MPI_RECV(colnorm, 1, MPI_DOUBLE_PRECISION, &
                   MPI_ANY_SOURCE, MPI_ANY_TAG, &
                   MPI_COMM_WORLD, status, ierr)
    jj = status(MPI_TAG)
    anorm(jj) = colnorm
    enddo
  endif
```

Note: Master may receive back in any order!

`MPI_ANY_SOURCE` will match first to arrive.

The tag is used to tell which column's norm has arrived (`jj`).

Send and Receive example — worker code

Master (Processor 0) sends jth column to Worker Processor j,
gets back 1-norm to store in `anorm(j)`, $j = 1, \ldots,$ `ncols`

```
! code for Workers (Processors 1, 2, ...):

if (proc_num /= 0) then

  call MPI_RECV(colvect, nrows, MPI_DOUBLE_PRECISION,&
                0, MPI_ANY_TAG, &
                MPI_COMM_WORLD, status, ierr)

  j = status(MPI_TAG)    ! this is the column number
                         ! (should agree with proc_num)

  colnorm = 0.d0
  do i=1,nrows
      colnorm = colnorm + abs(colvect(i))
      enddo

  call MPI_SEND(colnorm, 1, MPI_DOUBLE_PRECISION, &
               0, j, MPI_COMM_WORLD, ierr)

  endif
```

Note: Sends back to Process 0 with tag j.

Mathematical Software

It is best to use high-quality software as much as possible, for several reasons:

- It will take less time to figure out how to use the software than to write your own version. (Assuming it's well documented!)
- Good general software has been extensively tested on a wide variety of problems.
- Often general software is much more sophisticated that what you might write yourself, for example it may provide error estimates automatically, or it may be optimized to run fast.

Software sources

- Netlib: **`http://www.netlib.org`**
- NIST Guide to Available Mathematical Software: **`http://gams.nist.gov/`**
- Trilinos: **`http://trilinos.sandia.gov/`**
- DOE ACTS: **`http://acts.nersc.gov/`**
- PETSc nonlinear solvers: **`http://www.mcs.anl.gov/petsc/petsc-as/`**
- Many others!

Function `zeroin` from Netlib

The code in `$UWHPSC/codes/fortran/zeroin` illustrate how to use the function zeroin obtained from the Golden Oldies (go) directory of Netlib.

See: **http://www.netlib.org/go/index.html**

Estimates a zero of the function $f(x)$ between `ax` and `bx` within some tolerance.

```
c     =============================================
      function zeroin(ax,bx,f,tol)
c     =============================================
      implicit double precision (a-h,o-z)
      external f
```

Note: Fortran 77 style!

LAPACK — www.netlib.org/lapack/

Many routines for linear algebra.

Typical name: XYYZZZ

X is precision

YY is type of matrix, e.g. GE (general), BD (bidiagonal),

ZZZ is type of operation, e.g. SV (solve system),
EV (eigenvalues, vectors), SVD (singular values, vectors)

LAPACK — www.netlib.org/lapack/

Many routines for linear algebra.

Typical name: XYYZZZ

X is precision

YY is type of matrix, e.g. GE (general), BD (bidiagonal),

ZZZ is type of operation, e.g. SV (solve system),
EV (eigenvalues, vectors), SVD (singular values, vectors)

Examples:

`DGESV` can be used to solve a general $n \times n$ linear system in double precision.

`DGTSV` can be used to solve a general $n \times n$ tridiagonal linear system in double precision.

Installing LAPACK

On Virtual Machine or other Debian or Ubuntu Linux:

```
  $ sudo apt-get install liblapack-dev
```

This will include BLAS (but not optimized for your system).

Alternatively can download tar files and compile.

See complete documentation at
http://www.netlib.org/lapack/

The BLAS

Basic Linear Algebra Subroutines

Core routines used by LAPACK (Linear Algebra Package)
and elsewhere.

Generally optimized for particular machine architectures, cache hierarchy.

Can create optimized BLAS using
ATLAS (Automatically Tuned Linear Algebra Software)

See notes and `http://www.netlib.org/blas/faq.html`

- Level 1: Scalar and vector operations
- Level 2: Matrix-vector operations
- Level 3: Matrix-matrix operations

The BLAS

Subroutine names start with:

- S: single precision
- D: double precision
- C: single precision complex
- Z: double precision complex

Examples:

- SAXPY: single precision replacement of y by $\alpha x + y$.
- DDOT: dot product of two vectors
- DGEMV: matrix-vector multiply, general matrices
- DGEMM: matrix-matrix multiply, general matrices
- DSYMM: matrix-matrix multiply, symmetric matrices

Using libraries

If `program.f90` uses BLAS routines...

```
$ gfortran -c program.f90
$ gfortran program.o -lblas
```

or can combine as

```
$ gfortran program.f90 -lblas
```

When linking together `.o` files, will look for a file called `libblas.a` (probably in `/usr/lib`).

This is a archived static library.

Using libraries

If `program.f90` uses BLAS routines...

```
$ gfortran -c program.f90
$ gfortran program.o -lblas
```

or can combine as

```
$ gfortran program.f90 -lblas
```

When linking together `.o` files, will look for a file called `libblas.a` (probably in `/usr/lib`).

This is a archived static library.

Can specify different library location using `-L/path/to/library`.

Making blas library

Download **http://www.netlib.org/blas/blas.tgz**.

Put this in desired location, e.g. `$HOME/lapack/blas.tgz`

```
$ cd $HOME/lapack
$ tar -zxf blas.tgz    # creates BLAS subdirect
$ cd BLAS
$ gfortran -O3 -c *.f
$ ar cr libblas.a *.o  # creates libblas.a
```

To use this library:

```
$ gfortran program.f90 -lblas \
           -L$HOME/lapack/BLAS
```

Note: Non-optimized Fortran 77 versions.

Better approach would be to use ATLAS.

Creating LAPACK library

Can be done from source at

`http://www.netlib.org/lapack/`

but somewhat more difficult.

Individual routines and dependencies can be obtained from netlib, e.g. the double precision versions from:

`http://www.netlib.org/lapack/double`

Download `.tgz` file and untar into directory where you want to use them, or make a library of just these files.

Memory management for arrays

Often a program needs to be written to handle arrays whose size is not known until the program is running.

Fortran 77 approaches:

- Allocate arrays large enough for any application,
- Use “work arrays” that are partitioned into pieces.

We will look at some examples from LAPACK since you will probably see this in other software!

Memory management for arrays

Often a program needs to be written to handle arrays whose size is not known until the program is running.

Fortran 77 approaches:

- Allocate arrays large enough for any application,
- Use “work arrays” that are partitioned into pieces.

We will look at some examples from LAPACK since you will probably see this in other software!

The good news:

Fortran 90 allows dynamic memory allocation.

DGESV — Solves a general linear system

http://www.netlib.org/lapack/double/dgesv.f

```
      SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV,
     &                  B, LDB, INFO )
```

N = size of system (square $N \times N$)

A = matrix on input, L,U factors on output,
 dimension(LDA,K) with LDA, K >= N

LDA = leading dimension of A
 (number of rows in declaration of A)

Example:

```
real(kind=8) dimension(100,500) :: a
! fill a(1:20, 1:20) with 20x20 matrix
n = 20
lda = 100
```

DGESV — Solves a general linear system

Example:

```
real(kind=8), dimension(100,500) :: a
real(kind=8), dimension(200,400) :: b
integer, dimension(600) :: ipiv
! fill a(1:20, 1:20) with 20x20 matrix
!      b(1:20, 1:3) with 3 right hand sides

n = 20;  nrhs = 3;  lda = 100;  ldb = 200

call dgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
```

What is passed to `dgesv` is `start_address`, the address of first element of `a`. (Matrix is stored by columns)

Whenever `a(i,j)` appears in code, address is:

```
address = start_address + (j-1)*lda + (i-1)
```

DGESV — Solves a general linear system

```
      SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV,
     &                  B, LDB, INFO )
```

`NRHS` = number of right hand sides

`B` = matrix whose columns are right hand side(s) on input
solution vector(s) on output.

`LDB` = leading dimension of `B`.

`INFO` = integer returning 0 if successful.

`A` = matrix on input, L,U factors on output,

`IPIV` = Returns pivot vector (permutation of rows)
`integer, dimension(N)`
Row `I` was interchanged with row `IPIV(I)`.

Gaussian elimination as factorization

If A is nonsingular it can be factored as

$$PA = LU$$

where

P is a permutation matrix (rows of identity permuted),

L is lower triangular with 1's on diagonal,

U is upper triangular.

After returning from `dgesv`:
`A` contains L and U (without the diagonal of L),
`IPIV` gives ordering of rows in P.

Gaussian elimination as factorization

Example:

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 4 & 3 & 6 \\ 2 & 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 1 & 3 \\ 4 & 3 & 6 \\ 2 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/2 & -1/3 & 1 \end{bmatrix} \begin{bmatrix} 4 & 3 & 6 \\ 0 & 1.5 & 1 \\ 0 & 0 & 1/3 \end{bmatrix}$$

`IPIV` = (2,3,1)

and `A` ends up as

$$\begin{bmatrix} 4 & 3 & 6 \\ 1/2 & 1.5 & 1 \\ 1/2 & -1/3 & 1/3 \end{bmatrix}$$

`dgesv` examples

See `$UWHPSC/codes/lapack/random`.

Sample codes that solve the linear system $Ax = b$ with a random $n \times n$ matrix A, where the value n is run-time input.

`randomsys1.f90` is with static array allocation.

`randomsys2.f90` is with dynamic array allocation.

`dgesv` examples

See `$UWHPSC/codes/lapack/random`.

Sample codes that solve the linear system $Ax = b$ with a random $n \times n$ matrix A, where the value n is run-time input.

`randomsys1.f90` is with static array allocation.

`randomsys2.f90` is with dynamic array allocation.

`randomsys3.f90` also estimates condition number of A.

$$\kappa(A) = \|A\| \, \|A^{-1}\|$$

Can bound relative error in solution in terms of relative error in data using this:

$$Ax^* = b^* \text{ and } A\tilde{x} = \tilde{b} \implies \frac{\|\tilde{x} - x^*\|}{\|x^*\|} \le \kappa(A) \frac{\|\tilde{b} - b^*\|}{\|b^*\|}$$