Parallel Libraries

Using Libraries: Pro and Con

Pro:
- Libraries can provide high quality, high performance code
- User does not have to write the parallel program, which is difficult
- MPI has good support for parallel libraries
- Use of communicators allows library to isolate its communications universe from rest of program, avoiding conflicts

Con:
- Writing and documenting a library is difficult and time-consuming, hence some libraries may be deficient in some respects and difficult to use
- Examples of well-designed and well-documented libraries: ScaLAPACK, PETSc

Parallel Libraries: FFTW with MPI

FFTW - Fastest Fourier Transform in the West

Most widely used implementation of FFT (Fast Fourier Transform)

Forward discrete Fourier transform of 1d complex array X of size n computes array of Y of size n via:

$$Y_k = \sum_{j=0}^{n-1} X_j e^{-2\pi jk\sqrt{-1}/n}$$

Algorithm scales as NlogN

Higher dimensional transforms are straightforward extensions of 1D transform

Installing FFTW with MPI support

Newest version 3.x of this not available on SHARCNET, so do it yourself

Download source code (fftw-3.3.1.tar.gz) from http://www.fftw.org/

The following will do the compile in /tmp (fastest way)

```
mkdir /tmp/mycompile
cp fftw-3.3.1.tar.gz /tmp/mycompile/
cd /tmp/mycompile
tar xvfz fftw-3.3.1.tar.gz
cd fftw-3.3.1
./configure --prefix=/tmp/mycompile/fftw_exec --enable-mpi MPICC=mpicc
make
make install
cd
mv /tmp/mycompile .

mpicc -I/home/$USER/mycompile/fftw_exec/include  -L/home/$USER/
mycompile/fftw_exec/lib -lfftw3_mpi -lfftw3 test.c
```

Example: 2D complex forward FFT   in parallel

Data distribution: 1D block along first dimension

FFTW will decide how to distribute the data, and this information must be extracted from it and used in the program

Example code from:
http://www.fftw.org/fftw3_doc/2d-MPI-example.html#g_t2d-MPI-example

```
#include <complex.h>
#include <fftw3-mpi.h>

fftw_complex my_function(ptrdiff_t i_in, ptrdiff_t j_in){
 return 3*i_in+4*j_in*I;  // simple example function
}
```

```
int main(int argc, char **argv)
{
    const ptrdiff_t N0 = 200, N1 = 200;
    fftw_plan plan;
    fftw_complex *data;
    ptrdiff_t alloc_local, local_n0, local_0_start, i, j;

    MPI_Init(&argc, &argv);
    fftw_mpi_init();

    /* get local data size and allocate */
    alloc_local = fftw_mpi_local_size_2d(N0, N1, MPI_COMM_WORLD,
                                         &local_n0, &local_0_start);
    data = fftw_alloc_complex(alloc_local);

    /* create plan for in-place forward DFT */
    plan = fftw_mpi_plan_dft_2d(N0, N1, data, data, MPI_COMM_WORLD,
                                FFTW_FORWARD, FFTW_ESTIMATE);
```

```
    /* initialize data to some function my_function(x,y) */

    for (i = 0; i < local_n0; ++i)
         {
           for (j = 0; j < N1; ++j)
           {
                 data[i*N1 + j] = my_function(local_0_start + i, j);
           }
         }

    /* compute transforms, in-place, as many times as desired */
    fftw_execute(plan);

    fftw_destroy_plan(plan);

    MPI_Finalize();
}
```

Parallel Libraries: ScaLAPACK

LAPACK and ScaLAPACK

LAPACK, stands for “Linear Algebra Package”

Large library of functions for solving problems in linear algebra
Relies on lower level library called BLAS, which stands for “Basic Linear Algebra Subprogram” that performs common operations like dot product, matrix-vector multiply and matrix-matrix multiply

BLAS is usually highly optimized for a particular system

ScaLAPACK, stands for “Scalable LAPACK”

Parallel extension (for distributed memory systems) of LAPACK

Provides parallel version of LAPACK routines, usually by adding P to the LAPACK name (eg. ZGETRF -> PZGETRF)

Uses PBLAS (Parallel BLAS), as well as LAPACK and BLAS for local operations

Communications functions necessary are gathered in library called BLACS

Using more than one language

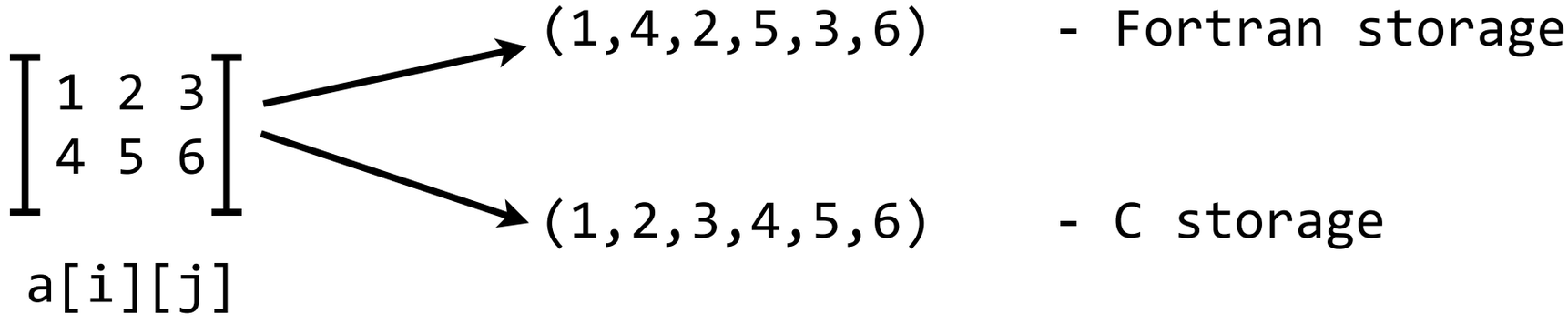
Most of ScaLAPACK is written in Fortran (C versions may also be available). What are the issues involved in calling Fortran subprograms from C?

1. Many compilers change the name of Fortran subprograms. Fortran subroutine moo(x) may be have to be called in C as moo_(x)

2. Linking issues. C does not automatically link with Fortran libraries

3. In Fortan all parameters are passed by address, while in C all parameters are passed by value. But arrays in C are pointers, so they can be passed without changes. Solution is to pass each scalar argument as a pointer. Can also use wrappers.

4. Two-dimensional arrays in Fortran are stored in column-major order, while in C they are stored in row-major order



```
! Fortran example of LAPACK usage
! compile on SHARCNET with: f90 -llapack
! get inverse of matrix
program main
implicit none
integer,parameter :: n=3
integer,parameter :: lwork=2*n ! needs to be >n
integer info,i,j
integer, dimension(n) :: ipiv
double precision, dimension (n,n) :: a
double precision, dimension (lwork) :: work
do i=1,n
do j=1,n
a(i,j)=dble(2*i+j) ! initialize matrix with something
enddo
enddo
! inverse, first step LU decomposition
call dgetrf(n,n,a,n,ipiv,info)
print *,”info dgetrf “,info
call dgetri(n,a,n,ipiv,work,lwork,info)
print *,”info dgetri “,info
do i=1,n
do j=1,n
print*,a(i,j)
enddo
enddo
end
```

```c
#include <stdio.h>
// compile with cc -llapack
// get inverse of matrix
int main(){
const int n=3;
const int lwork=2*n; // needs to be >n
int info,i,j;
int ipiv[n];
double a[n][n];
double work[lwork];

for (i=0;i<n;i++){
for (j=0;j<n;j++){
// Lapack routines are from Fortran so must transpose data
a[j][i]=(double) (2*(i+1)+(j+1));}}

// inverse, first step LU decomposition
dgetrf_(&n,&n,a,&n,ipiv,&info);
printf("info dgetrf %d \n",info);
dgetri_(&n,a,&n,ipiv,work,&lwork,&info);
printf("info dgetri %d \n",info);

for (i=0;i<n;i++){
for (j=0;j<n;j++){
// transpose at the end as well
printf("%lf \n",a[j][i]);}}
return 0;}
```

Distributing the data

To achieve parallelism, the data (i.e. the matrix) must be distributed by the user among the processes. This is done before ScaLAPACK routines are called.

After ScaLAPACK is done, the data must be collected from processes as required

ScaLAPACK relies on the concept of process grid and block-cyclic mapping

Libraries create a rectangular grid of processes, much like topology in MPI.

Each matrix and vector is decomposed into rectangular blocks, and the matrix blocks are mapped cyclically to virtual processes in each of the process dimensions.

Example: decompose 7x7 matrix into 3x2 blocks, then use virtual 2x2 grid of processes

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}	a_{06}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}
a_{40}	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}
a_{50}	a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}
a_{60}	a_{61}	a_{62}	a_{63}	a_{64}	a_{65}	a_{66}

Process 0		Process 1		
a_{00} a_{01} a_{10} a_{11} a_{20} a_{21}		a_{04} a_{05} a_{14} a_{15} a_{24} a_{25}	a_{02} a_{03} a_{12} a_{13} a_{22} a_{23}	a_{06} a_{16} a_{26}
a_{60} a_{61}		a_{64} a_{65}	a_{62} a_{63}	a_{66}
Process 2		Process 3		
a_{30} a_{31} a_{40} a_{41} a_{50} a_{51}		a_{34} a_{35} a_{44} a_{45} a_{54} a_{55}	a_{32} a_{33} a_{42} a_{43} a_{52} a_{53}	a_{36} a_{46} a_{56}

In order to completely describe a distributed matrix, libraries make use of a data structure called a descriptor

As libraries are callable from Fortran, descriptors are arrays of integers rather than structs

Each descriptor contains:

1. Global dimensions of matrix (number of rows and columns)
2. Dimension of matrix blocks
3. Process row and column of first entry of matrix (i.e. need not begin with process 0, though it usually will)
4. The handle of BLACS grid
5. Leading dimension of the local storage for the matrix

ScaLAPACK example

Refer to code file: linsolve.c

Solve linear system $\mathbf{Ax}=\mathbf{b}$

Generate matrix A, generate solution via b=Ae where e is vector containing all 1s

Solve for x, check accuracy

Parallel routines used:

psgemv - PBLAS function, performs matrix-vector multiplication

$\mathbf{y}=a\mathbf{Ax}+b\mathbf{y}$

psgesv - ScaLAPACK function, computes solution x to real system of equations

$\mathbf{Ax}=\mathbf{b}$

Will use wrappers for these function so they are easy to call in C

Conclusion

Parallelization of matrix communications is not easy, as we have seen before

ScaLAPACK will offer performance gains for large matrices, but not necessarily for small ones

Performance is dependent on the system and size of the problem

Linear scaling with the number of processes is generally not to be expected for most problems you try