Intermediate MPI

Le Yan

High Performance Computing Enablement
Center for Computation and Technology
Louisiana State University
April 2, 2008
Goals of training

• Learn derived datatypes in MPI
• Learn the basics about MPI communicators
Outline

• Basic MPI recap
• Derived data type - communicate non-contiguous data
• Communicators
• Case study: matrix transpose
Outline

• Basic MPI recap

• Derived data type - communicate non-contiguous data

• Communicators

• Case study: matrix transpose
What is MPI

• Context
  – Distributed memory parallel computers

• MPI is a standard
  – What is in the standard: the syntax and semantics of a set of core functions
  – What is NOT in the standard: how to compile and link MPI programs and how many processors to use
MPI Program Structure

! MPI header file
#include 'mpif.h'
...
! MPI initialization
call mpi_init(ierr)
...
call mpi_comm_size(comm,size,ierr)
call mpi_comm_rank(comm,rank,ierr)
...
<Main code>
...
! MPI termination
call mpi_finalize(ierr)
...
Point-to-point communication

- Fundamental message passing function
- One process sends message and another process receive it
- Blocking vs. Non-blocking
Collective Communication

- Collective communications involve all processes in a communicator
- Must be called by all involved processes
- All collective communications are blocking
Outline

• Basic MPI recap

• Derived data type - communicate non-contiguous data

• Communicators

• Case study: matrix transpose
# Basic Datatypes

<table>
<thead>
<tr>
<th>MPI datatype</th>
<th>C datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_CHAR</td>
<td>signed char</td>
</tr>
<tr>
<td>MPI_SHORT</td>
<td>signed short int</td>
</tr>
<tr>
<td>MPI_INT</td>
<td>signed int</td>
</tr>
<tr>
<td>MPI_LONG</td>
<td>signed long int</td>
</tr>
<tr>
<td>MPI_UNSIGNED_CHAR</td>
<td>unsigned char</td>
</tr>
<tr>
<td>MPI_UNSIGNED_SHORT</td>
<td>unsigned short int</td>
</tr>
<tr>
<td>MPI_UNSIGNED</td>
<td>unsigned int</td>
</tr>
<tr>
<td>MPI_UNSIGNED_LONG</td>
<td>unsigned long int</td>
</tr>
<tr>
<td>MPI_FLOAT</td>
<td>float</td>
</tr>
<tr>
<td>MPI_DOUBLE</td>
<td>double</td>
</tr>
<tr>
<td>MPI_LONG_DOUBLE</td>
<td>long double</td>
</tr>
<tr>
<td>MPI_BYTE</td>
<td></td>
</tr>
<tr>
<td>MPI_PACKED</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MPI datatype</th>
<th>Fortran datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>MPI_DOUBLE_PRECISION</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>MPI_COMPLEX</td>
<td>COMPLEX</td>
</tr>
<tr>
<td>MPI_LOGICAL</td>
<td>LOGICAL</td>
</tr>
<tr>
<td>MPI_CHARACTER</td>
<td>CHARACTER(1)</td>
</tr>
<tr>
<td>MPI_BYTE</td>
<td></td>
</tr>
<tr>
<td>MPI_PACKED</td>
<td></td>
</tr>
</tbody>
</table>
Why Derived Datatypes?

• Basic communication calls so far have involved only contiguous data with a sequence of elements of a single type.

• What if the data to be transferred is not contiguous? Not of the same type?

MPI_Bcast?
Possible solutions for non-contiguous data transfer (1)

• Make multiple communication calls

```fortran
... 
call mpi_bcast(a(1),1,mpi_integer,...) 
call mpi_bcast(a(3),1,mpi_integer,...) 
call mpi_bcast(a(5),1,mpi_integer,...) 
... 
```
Possible solutions for non-contiguous data transfer (2)

- Pack data into contiguous buffers manually

```plaintext
... 
tmp(1)=a(1)
tmp(2)=a(3)
tmp(3)=a(5)
call mpi_bcast(tmp,3,mpi_integer,...)
a(1)=tmp(1)
a(3)=tmp(2)
a(5)=tmp(3)
... 
```
Possible solutions for non-contiguous data transfer (3)

• Use derived datatypes
  – Tell the library what is desired
  – Let the library decide how the communication is done

... call mpi_type_vector(3,1,2,mpi_integer,newtype,ierr)
call mpi_type_commit(newtype,ierr)
call mpi_bcast(a(1),1,newtype,...)
...
MPI_TYPE_CONTIGUOUS

- Allows replication of a datatype into contiguous locations
- MPI_TYPE_CONTIGUOUS(count, oldtype, newtype, ierror)

```call
call mpi_type_contiguous(4,mpi_integer,newtype,ierr)
call mpi_type_commit(newtype,ierr)
```

MPI_INTEGER: 
NEWTYPE: 

MPI_TYPE_VECTOR

- Allows replication of a datatype into locations that consist of equally spaced blocks
- MPI_TYPE_VECTOR(count, blocklength, stride, oldtype, newtype, ierror)
- MPI_TYPE_HVECTOR: identical to MPI_TYPE_VECTOR, except that stride is in bytes, rather than in number of oldtype elements

CALL MPI_TYPE_VECTOR(3,2,4,MPI_INTEGER,NEWTYPE,IERR)
call MPI_TYPE_COMMIT(NEWTYPE,IERR)
Example: MPI_Type_Vector

... 
Integer a(6) 
... 
if (myrank.eq.0) then 
do i=1,6 
a(i)=i 
enddo 
else 
a=0 
endif 
call mpi_type_vector(3,1,2,mpi_integer,newtype,ierr) 
call mpi_type_commit(newtype,ierr) 
call mpi_bcast(a(1),1,newtype,0,mpi_comm_world,ierr) 
print
Example: MPI_TYPEVECTOR

```fortran
... Integer a(6) ...
if (myrank.eq.0) then
    do i=1,6
        a(i)=i
    enddo
else
    a=0
endif
call mpi_type_vector(3,1,2,mpi_integer,newtype,ierr)
call mpi_type_commit(newtype,ierr)
call mpi_bcast(a(1),1,newtype,0,mpi_comm_world,ierr)
print
```

Output (4 processes):
```
1 2 3 4 5 6
1 0 3 0 5 0
1 0 3 0 5 0
1 0 3 0 5 0
```
MPI_TYPE_STRUCT

- Most general type constructor
- Allows a new data type that represents arrays of types, each of which has different blocklength, displacement (in bytes) and type.

Struct: Count = 2, array_of_blocklengths={3, 1}, Array_of_displacements={0,12} (in bytes) array_of_types={MPI_COMPLEX,MPI_INTEGER}
Example: broadcast a submatrix

```
!Parameters
ILEN=4
JLEN=3
IMAX=8
IMIN=1

!Derived datatype
CALL MPI_TYPE_VECTOR(JLEN,ILEN,IMAX-IMIN+1,
&           MPI_INTEGER,ISUBMAT,IERR)
CALL MPI_TYPE_COMMIT(ISUBMAT,IERR)

!Broadcast
CALL MPI_BCAST(AMAT(2,2),1,ISUBMAT,0,
&           MPI_COMM_WORLD,IERR)
```
Outline

• Basic MPI recap

• Derived data type - communicate non-contiguous data

• Communicators

• Case study: matrix transpose
Communicators and Groups

• An MPI communicator is the context of communication
  – It is associated with a ordered group of processes (MPI group)
  – MPI communicators can have other attributes in addition to group

• We can generate additional communicators in addition to
  MPI_COMM_WORLD
  – Useful when communication is required to a subset of the processes
Create New Communicators (1)

- Split an existing communicator
  - **Syntax:** \texttt{MPI\_COMM\_SPLIT(Comm, Color, Key, NewComm)}
  
  - Partitions the group associated with \texttt{Comm} into disjoint subgroups, one for each value of \texttt{Color}
  
  - Each subgroup contains all processes of the same color
  
  - Within each subgroup, the processes are ranked in the order defined by the value of the argument \texttt{Key}, with ties broken according to their rank in the old group
  
  - A new communicator \texttt{NewComm} is created for each subgroup
Example: MPI_COMM_SPLIT

...  
call mpi_comm_rank(mpi_comm_world,myoldrank,ierr)

color=mod(myoldrank,2)

call mpi_comm_split(mpi_comm_world,color,myoldrank,newcomm,ierr)

call mpi_comm_rank(newcomm,mynewrank,ierr)

write(*,*) myoldrank,mynewrank

...
Example: MPI_COMM_SPLIT

```fortran
... 
call mpi_comm_rank(mpi_comm_world,myoldrank,ierr)

color=mod(myoldrank,2)

call mpi_comm_split(mpi_comm_world,color,myoldrank,newcomm,ierr)

call mpi_comm_rank(newcomm,mynewrank,ierr)

write(*,*) myoldrank,mynewrank
...
```

Output (4 processes):

```
0 0
1 0
2 1
3 1
```
Example: MPI_COMM_SPLIT

```fortran
... 
call mpi_comm_rank(mpi_comm_world,myoldrank,ierr)

color=mod(myoldrank,2)

call mpi_comm_split(mpi_comm_world,color,myoldrank,newcomm,ierr)

call mpi_comm_rank(newcomm,mynewrank,ierr)

write(*,*)) myoldrank,mynewrank

call mpi_bcast(a,1,mpi_integer,0,newcomm,ierr)
...
```

What will happen with this mpi_bcast call?
Example: MPI_COMM_SPLIT

- We created two new communicators with the mpi_comm_split call.
- But from the point of view of each process, there is only ONE!
Example: MPI_COMM_SPLIT

- We created two new communicators with the mpi_comm_split call
- But from the point of view of each process, there is only ONE!
Create new communicators (2)

• Map the old communicator to a group
  – MPI_COMM_GROUP

• Manipulate the group
  – Include, exclude, union, intersection etc.

• Create a new communicator out of the modified group
  – MPI_COMM_CREATE
Example: Create A New Communicator

```fortran
... 
call mpi_comm_rank(mpi_comm_world,myoldrank,ierr)

! Map MPI_COMM_WORLD to the group ``oldgroup''
call mpi_comm_group(mpi_comm_world,oldgroup,ierr)

! Every process is included in ``newgroup'' except process 0
rank_excl=0
call mpi_group_excl(oldgroup,1,rank_excl,newgroup,ierr)

! Create a new communicator
call mpi_comm_create(mpi_comm_world,newgroup,newcomm,ierr)

... 
```
Example: Create A New Communicator

... 

call mpi_comm_rank(mpi_comm_world,myoldrank,ierr)

! Map MPI_COMM_WORLD to the group ```oldgroup''
call mpi_comm_group(mpi_comm_world,oldgroup,ierr)

! Every process is included in ```newgroup'' except process 0
rank_excl=0
call mpi_group_excl(oldgroup,1,rank_excl,newgroup,ierr)

! Create a new communicator
call mpi_comm_create(mpi_comm_world,newgroup,newcomm,ierr)

! What will happen?
call mpi_comm_rank(newcomm,mynewrank,ierr)

...
Example: Create A New Communicator

... 

call mpi_comm_rank(mpi_comm_world,myoldrank,ierr)

! Map MPI_COMM_WORLD to the group ``oldgroup''
call mpi_comm_group(mpi_comm_world,oldgroup,ierr)

! Every process is included in ``newgroup'' except process 0
rank_excl=0
call mpi_group_excl(oldgroup,1,rank_excl,newgroup,ierr)

! Create a new communicator
call mpi_comm_create(mpi_comm_world,newgroup,newcomm,ierr)

! What will happen?
call mpi_comm_rank(newcomm,mynewrank,ierr)

...
Group management functions

- MPI_Group_union(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

- MPI_Group_intersection(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

- MPI_Group_difference(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

- MPI_Group_incl(MPI_Group group, int n, int *ranks, MPI_Group *newgroup)
  - New group with n elements of group

- MPI_Group_excl(MPI_Group group, int n, int *ranks, MPI_Group *newgroup)
  - New group with all but n elements of group
Freeing Communicators & Groups

- We can destroy communicators and groups too
  - MPI_Comm_free(MPI_Comm *comm)
  - MPI_Group_free(MPI_Group *group)
Topology

- An attribute of MPI communicators in addition to group
- Can be either Cartesian or Graph
- Useful for domain decomposition
Example: Create a 2-D Cartesian Topology

...!

! Create a 2-D cartesian topology
call mpi_cart_create(mpi_comm_world, ndim, dim_size,
    periods, reorder, newcomm, ierr)

! Returns the coordinates of the given rank
call mpi_cart_coords(newcomm, rank, ndims, coords, ierr)

! Found the neighbor processes along the given direction
call mpi_cart_shift(newcomm, direction, displacement,
    source_rank, destination_rank, ierr)

...
Outline

• Basic MPI recap

• Derived data type - communicate non-contiguous data

• Communicators

• Case study: matrix transpose
Matrix transpose

\[ A = \begin{bmatrix}
11 & 12 & 13 & 14 \\
21 & 22 & 23 & 24 \\
31 & 32 & 33 & 34 \\
41 & 42 & 43 & 44 \\
\end{bmatrix} \quad A^T = \begin{bmatrix}
11 & 21 & 31 & 41 \\
12 & 22 & 32 & 42 \\
13 & 23 & 33 & 43 \\
14 & 24 & 34 & 44 \\
\end{bmatrix} \]
Matrix transpose

\[ A = \begin{pmatrix}
11 & 12 & 13 & 14 \\
21 & 22 & 23 & 24 \\
31 & 32 & 33 & 34 \\
41 & 42 & 43 & 44
\end{pmatrix} \quad A^T = \begin{pmatrix}
11 & 21 & 31 & 41 \\
12 & 22 & 32 & 42 \\
13 & 23 & 33 & 43 \\
14 & 24 & 34 & 44
\end{pmatrix} \]
Divide and Conquer

- Divide the matrix into sub-matrices and perform transpose on each sub-matrix
- Single program multiple data (SPMD) model
The task

• The root process has an $m \times n$ matrix

• After the execution of the code each process has the transposed $n \times m$ matrix
Algorithm

• Broadcast the matrix to all processes

• Create a 2-D grid and determine the sub-matrix for each processor

• Each processor transposes its own sub-matrix

• Each processor transfer its sub-matrix to others
Algorithm

• Broadcast the matrix to all processes (mpi_bcast)

• Create a 2-D grid and determine the sub-matrix for each processor (mpi_cart_create, mpi_cart_coords)

• Each processor transposes its own sub-matrix (local)

• Each processor transfer its sub-matrix to others (mpi_bcast)
Topology

(0,0) (0,1)
(1,0) (1,1)
(2,0) (2,1)
The sample code

- The code is too long to present here and is included in the lab material

- Alternative approaches
  - Use derived datatypes and other MPI functions not covered
  - Sample code provided in the lab material
Questions?