MA471
Lecture 5

Collective MPI Communication
Today:
When all the processes want to send, receive or both

Excellent website for MPI command syntax available at:

http://www-unix.mcs.anl.gov/mpi/www/
Names

For the following exercise I need to know each student’s first name.

In return I will give you a global ID.

Please write down your global ID, making sure you know which is which.

To make things easier you should copy down the list of names and global IDs which I will read out.
If Chris is determined to tell everyone else something important – then there are a number of ways the information can be disseminated.

He could tell everyone individually. But clearly while he communicates with each person in turn – everyone else is twiddling their thumbs.

Alternatively, he can start off a chain of communications which can thought of in a tree-like sequence:
Example tree communication for Bcast
Comments

• In this case there are 8 processes, so the minimum number of communications is 7.

• Other tree constructions are possible.

• We will construct a tree for the global group

• Using this tree we will try a Bcast!!.

• Now I need a volunteer to build a Bcast tree
Global Exercise – Mimic MPI_Bcast

1) Init

2) Barrier.

3) Process 0 sends message to right “leaf” node.

4) Advance to next level of tree.

5) Processes on this level communicate to their right “leaf node”

6) If you have received message and have no “leaf” nodes put your left hand up.

7) Return to step 4

8) Finalize
Diagramatic Description of MPI_Bcast

Comments/Observations?
MPI_Bcast
Broadcasts a message from the process with rank "root" to all other processes of the group.

Synopsis
int MPI_Bcast ( void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm )

Input/output Parameters
buffer starting address of buffer (choice)
count number of entries in buffer (integer)
datatype data type of buffer (handle)
root rank of broadcast root (integer)
comm communicator (handle)

Notes on MPI_Bcast

Note:

1) all processes must make the call to MPI_Bcast
   i.e. they all need to know that it is going to happen.

2) If a process does not join in the Bcast then the rest
   Of the processes will wait..

3) Process “root” will send the same message to all
   other process
MPI_Allreduce
Imagine you each give the last exercise a grade out of 10 as to how much it sucked (10 = real bad).

Then how can you all find out what the average suckiness rating is???.

One way is to use MPI_ALLREDUCE 😊

MPI_ALLREDUCE combines values from all processes and distributes the result back to all processes.
Diagramatic Description of MPI_Allreduce

For example: $\text{Op}(\mathbf{A})$ could be $\text{Op}(\mathbf{A}) = A_0 + A_1 + A_2 + A_3 + A_4$

http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/CommonDoc/MessPass/MPIColl.html
**MPI_Allreduce**
Combines values from all processes and distribute the result back to all processes

**Synopsis**
```c
int MPI_Allreduce ( void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm );
```

**Input Parameters**
- `sendbuf` starting address of send buffer (choice)
- `count` number of elements in send buffer (integer)
- `datatype` data type of elements of send buffer (handle)
- `op` operation (handle)
- `comm` communicator (handle)

**Output Parameter**
- `recvbuf` starting address of receive buffer (choice)
Some of the different operations available:

- `MPI_MAX` — returns the maximum
- `MPI_MIN` — returns the minimum
- `MPI_SUM` — returns the sum
- `MPI_PROD` — returns the product
MPI_Allreduce example

```c
int Nprocs, rating, ratinglen, ratingsum, ierr;
double ratingave;

/* everyone has their own opinion */
rating = some number;

/* there is only one entry in the rating data */
ratinglen = 1;

/* find number of processes in world */
ierr = MPI_Comm_Size (MPI_COMM_WORLD, &Nprocs);

/* all processes send their rating and receive the sum of all ratings */
ierr = MPI_Allreduce (&rating,&ratingsum,ratinglen,
    ,MPI_INT,MPI_SUM,MPI_COMM_WORLD);

/* convert sum to average */
ratingave = ratingsum/Nprocs;
```
Now suppose that you all have something to say to each other. Further, you all have the same length message to send. We can think of the messages as a matrix of messages:
Diagramatic Description of MPI_Alltoall

http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/CommonDoc/MessPass/MPIColl.html
INPUT to MPI ALLTOALL

Proc. 0  Proc. 1  Proc. 2  Proc. 3

SAD  SLY  EAT  HAT
CAT  WAS  YOU  MAN
BAD  DOG  DID  ROT
BIG  RED  BOB  TOP
OUTPUT from MPI Alltoall

Proc. 0  |  BIG  |  RED  |  BOB  |  TOP
Proc. 1  |  BAD  |  DOG  |  DID  |  ROT
Proc. 2  |  CAT  |  WAS  |  YOU  |  MAN
Proc. 3  |  SAD  |  SLY  |  EAT  |  HAT

In essence the Alltoall has transposed the data
Global Exercise – mimic Alltoall

- **DO NOT** USE COMMUNICATION TREE
  1) Init
  2) Barrier
  3) Recall your global ID.
  4) Write down Nprocs 3 letter words
  5) Send your first word to process 0
  6) Receive proc. 0’s ID’th word
  7) Send your second word to process 1
  8) Receive proc. 1’s ID’th word
  9) .......
  10) Send your last word to process Nprocs-1
  11) Receive proc. Nprocs-1 ID’th word
  12) Barrier
  13) Finalize
Comments?, Observations?

• Now that should have been really tough 😊

• When everyone has something to say at the same time then communication becomes a real bottle neck

• This is one of the least “desirable” approaches to parallelism, it implies that all processes are tightly coupled and have to share data.

• If one of these is frequently necessary in a computation you should probably reconsider the methods you are using and their appropriateness for parallel computation.
**MPI_Alltoall**
Sends data from all processes to all processes

**Synopsis**
```
int MPI_Alltoall( void *sendbuf, int sendcount, MPI_Datatype sendtype,
                 void *recvbuf, int recvcount, MPI_Datatype recvtype,
                 MPI_Comm comm )
```

**Input Parameters**
- `sendbuf` starting address of send buffer (choice)
- `sendcount` number of elements to send to each process (integer)
- `sendtype` data type of send buffer elements (handle)
- `recvcount` number of elements received from any process (integer)
- `recvtype` data type of receive buffer elements (handle)
- `comm` communicator (handle)

**Output Parameter**
- `recvbuf` address of receive buffer (choice)

### Other Global MPI Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_ALLGATHER,</td>
<td>gather data from all processors in a group and distributes to all processors</td>
</tr>
<tr>
<td>MPI_ALLGATHERV</td>
<td>in the group</td>
</tr>
<tr>
<td>MPI_ALLREDUCE,</td>
<td>combines data from all processors in a group and distributes the result back</td>
</tr>
<tr>
<td>MPI_ALLREDUCEV</td>
<td>to all processors in the group</td>
</tr>
<tr>
<td>MPI_ALLTOALL,</td>
<td>sends data from all processors in a group to all processors in the group</td>
</tr>
<tr>
<td>MPI_ALLTOALLV</td>
<td></td>
</tr>
<tr>
<td>MPI_REDUCE</td>
<td>reduces data on all processors in a group to a single value</td>
</tr>
<tr>
<td>MPI_REDUCE_SCATTER</td>
<td></td>
</tr>
<tr>
<td>MPI_GATHER,</td>
<td>gather data from all processes in a group</td>
</tr>
<tr>
<td>MPI_GATHERV</td>
<td></td>
</tr>
<tr>
<td>MPI_SCATTER,</td>
<td>sends data from one task to all other tasks in a group</td>
</tr>
<tr>
<td>MPI_SCATTERV</td>
<td></td>
</tr>
</tbody>
</table>

So far we have covered MPI calls for housekeeping:
MPI_Init, MPI_Finalize, MPI_Comm_rank, MPI_Comm_size

Process to process message passing:
MPI_Send, MPI_Recv, MPI_Isend, MPI_Irecv

Global processes to processes message passing:
MPI_Bcast, MPI_Gather, MPI_Scatter, MPI_Alltoall

Synchronization
MPI_Barrier
Lab Activity

1) Parallel card game development time

2) Presentations if we have any volunteers.
Next Lecture

Class:

Introduction of a very simple finite difference method for solving a very simple PDE

Lab:

Mini-presentations of MPI card playing in action using Powerpoint.
Handing in of finished project