Parallel Programming Using MPI

Gregory G. Howes
Department of Physics and Astronomy
University of Iowa

Iowa High Performance Computing Summer School
University of Iowa
Iowa City, Iowa
6-8 June 2012
Thank you

Ben Rogers  Information Technology Services
Glenn Johnson  Information Technology Services
Mary Grabe  Information Technology Services
Amir Bozorgzadeh  Information Technology Services
Preston Smith  Purdue University

and

National Science Foundation
Rosen Center for Advanced Computing, Purdue University
Great Lakes Consortium for Petascale Computing
Outline

• General Comments

• Concepts

• Environment Management Routines

• Point-to-Point Communication Routines

• Collective Communication Routines

• Asynchronous Communication
General Comments

• Biggest hurdle to parallel computing is just getting started

• I will not cover all of the functionality of the MPI library
  - Focus on basic point-to-point and collective communications.

• You can do almost everything you ever need to do with just 8 commands.
  - Another 4 commands for collective communications are also useful.
  - Probably 95% of MPI users can get away with just these 12 commands, so I will focus on these commands here, and briefly mention a few others.
Concepts

• Communicators

• Point-to-point vs. collective communications

• Buffering of messages

• Issues of Synchronization and Determinism
  - Deadlocks and Race Conditions
Outline

• General Comments

• Concepts

• Environment Management Routines

• Point-to-Point Communication Routines

• Collective Communication Routines

• Asynchronous Communication
Environment Management Routines

Basic Requirements:
• Include Header File:

C  #include "mpi.h"
Fortran  include 'mpif.h'

• General Format of calls differs between C and Fortran

C  rc = MPI_Bsend(&buf,count,type,dest,tag,comm)
Fortran  CALL MPI_BSEND(buf,count,type,dest,tag,comm,ierr)

Initializing and Finalizing parallel tasks in MPI:
• Initialization

C  MPI_Init (&argc,&argv)
Fortran  MPI_INIT (ierr)

• Finalization

C  MPI_Finalize ()
Fortran  MPI_FINALIZE (ierr)
Environment Management Routines

Size and Rank:

• Determine number of MPI tasks
  
  C       MPI_Comm_size (comm,&size)
  Fortran MPI_COMM_SIZE (comm,size,ierr)

  Arguments:
  
<table>
<thead>
<tr>
<th>Intent</th>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>comm</td>
<td>handle</td>
<td>Communicator</td>
</tr>
<tr>
<td>OUT</td>
<td>size</td>
<td>integer</td>
<td>Number of MPI tasks associated with comm</td>
</tr>
</tbody>
</table>

• Determine rank of this MPI task:
  
  C       MPI_Comm_rank (comm,&rank)
  Fortran MPI_COMM_RANK (comm,rank,ierr)

  Rank is the Task ID
Serial Version:

!--------------------------------------------------
!
! HELLO WORLD
!
!--------------------------------------------------

program helloworld_serial
  implicit none

    !Write out message to screen
    write(*,'(a)')'Hello World.'

end program helloworld_serial
Example: Hello World

Parallel Version:

!--------------------------------------------------
!          HELLO WORLD
!--------------------------------------------------

program helloworld
  implicit none
  include 'mpif.h'
  integer :: nproc !Number of Processors
  integer :: iproc !Number of local processors
  integer :: ierror !Integer error flag

!Initialize MPI message passing
  call mpi_init (ierror)
  call mpi_comm_size (mpi_comm_world, nproc, ierror)
  call mpi_comm_rank (mpi_comm_world, iproc, ierror)

!Write out message to screen
  write(*,'(a,i4,a,i4)')'Hello World. I am processor ',iproc, &
                       ' of ',nproc

!Finalize MPI message passing
  call mpi_finalize (ierror)
end program helloworld
Outline

• General Comments

• Concepts

• Environment Management Routines

• Point-to-Point Communication Routines

• Collective Communication Routines

• Asynchronous Communication
Point-to-Point Communication Routines

• Send a Message

C     MPI_Send (&buf, count, datatype, dest, tag, comm)
Fortran   MPI_SEND (buf, count, datatype, dest, tag, comm, ierr)

Arguments:

<table>
<thead>
<tr>
<th>Intent</th>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>buf</td>
<td>choice</td>
<td>Address of data array to send</td>
</tr>
<tr>
<td>IN</td>
<td>count</td>
<td>integer(≥ 0)</td>
<td>Number of array elements to send</td>
</tr>
<tr>
<td>IN</td>
<td>datatype</td>
<td>handle</td>
<td>Type of data to send</td>
</tr>
<tr>
<td>IN</td>
<td>dest</td>
<td>integer</td>
<td>Rank (task ID) of destination task</td>
</tr>
<tr>
<td>IN</td>
<td>tag</td>
<td>integer</td>
<td>Message tag</td>
</tr>
<tr>
<td>IN</td>
<td>comm</td>
<td>handle</td>
<td>Communicator</td>
</tr>
</tbody>
</table>
Point-to-Point Communication Routines

- Receive a Message

```c
MPI_Recv (&buf, count, datatype, source, tag, comm, &status)
```

```fortran
MPI_RECV (buf, count, datatype, source, tag, comm, status, ierr)
```

**Arguments:**

<table>
<thead>
<tr>
<th>Intent</th>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT</td>
<td>buf</td>
<td>choice</td>
<td>Address of data array to receive</td>
</tr>
<tr>
<td>IN</td>
<td>count</td>
<td>integer(≥ 0)</td>
<td>Number of array elements to receive</td>
</tr>
<tr>
<td>IN</td>
<td>datatype</td>
<td>handle</td>
<td>Type of data to recv</td>
</tr>
<tr>
<td>IN</td>
<td>source</td>
<td>integer</td>
<td>Rank (task ID) of source task</td>
</tr>
<tr>
<td>IN</td>
<td>tag</td>
<td>integer</td>
<td>Message tag</td>
</tr>
<tr>
<td>IN</td>
<td>comm</td>
<td>handle</td>
<td>Communicator</td>
</tr>
<tr>
<td>OUT</td>
<td>status</td>
<td>status</td>
<td>Status object</td>
</tr>
</tbody>
</table>
Point-to-Point Communication Routines

• Blocking vs. Non-blocking

• Synchronous vs. Asynchronous

• Determinism

• Deadlocks, or Race Conditions
Outline

• General Comments

• Concepts

• Environment Management Routines

• Point-to-Point Communication Routines

• Collective Communication Routines

• Asynchronous Communication
Collective Communication Routines

- Broadcast:

```
(1) MPI_BCAST
```

Arguments:

<table>
<thead>
<tr>
<th>Intent</th>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INOUT</td>
<td>buffer</td>
<td>choice</td>
<td>Address of input data array, or output data array at root</td>
</tr>
<tr>
<td>IN</td>
<td>count</td>
<td>integer(≥ 0)</td>
<td>Number of array elements to receive</td>
</tr>
<tr>
<td>IN</td>
<td>datatype</td>
<td>handle</td>
<td>Type of data to recv</td>
</tr>
<tr>
<td>IN</td>
<td>root</td>
<td>integer</td>
<td>Rank (task ID) of root task</td>
</tr>
<tr>
<td>IN</td>
<td>comm</td>
<td>handle</td>
<td>Communicator</td>
</tr>
</tbody>
</table>
Collective Communication Routines

• Reduction:
  - This operation takes the data in the same variable on each processor, or array of variables, and performs an operation on all of the variables, for example computing the sum or finding the maximum value.
  - The result is either collected at the root process (MPI_Reduce) or distributed to all processes (MPI_Allreduce).

C  
```c
MPI_Reduce (&sendbuf,&recvbuf,count,datatype,op,root,comm)
```

Fortran  
```fortran
MPI_REDUCE (sendbuf,recvbuf,count,datatype,op,root,comm,ierr)
```

Arguments:

<table>
<thead>
<tr>
<th>Intent</th>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>sendbuf</td>
<td>choice</td>
<td>Address of input data array to send</td>
</tr>
<tr>
<td>OUT</td>
<td>recvbuf</td>
<td>choice</td>
<td>Address of output data array for result at root</td>
</tr>
<tr>
<td>IN</td>
<td>count</td>
<td>integer(≥ 0)</td>
<td>Number of array elements to receive</td>
</tr>
<tr>
<td>IN</td>
<td>datatype</td>
<td>handle</td>
<td>Type of data to recv</td>
</tr>
<tr>
<td>IN</td>
<td>op</td>
<td>handle</td>
<td>Operation to perform</td>
</tr>
<tr>
<td>IN</td>
<td>root</td>
<td>integer</td>
<td>Rank (task ID) of root task</td>
</tr>
<tr>
<td>IN</td>
<td>comm</td>
<td>handle</td>
<td>Communicator</td>
</tr>
</tbody>
</table>

(4)  
```c
MPI_REDUCEALL
```
Collective Communication Routines

• Example of Reduction Operation:

Processes...

Initial Data:

0 | 1 | 2 | 3
---|---|---|---
24 | 57 | 03 | 62

MPI_REDUCE with MPI_MIN, root = 0:

02 | -- | -- | --

MPI_ALLREDUCE with MPI_MIN:

02 | 02 | 02 | 02

MPI_REDUCE with MPI_SUM, root = 1:

-- | 1316 | -- | --
Collective Communication Routines

• Barrier: Wait for all tasks to synchronize
  
  | C          | MPI_Barrier (comm) |
  | Fortran    | MPI_BARRIER (comm,ierr) |

• Others: Scatter and Gather

(2) MPI_SCATTER

(5) MPI_GATHER
Outline

• General Comments

• Concepts

• Environment Management Routines

• Point-to-Point Communication Routines

• Collective Communication Routines

• Asynchronous Communication
Asynchronous Communication Routines

• **Asynchronous communication** allows an MPI task continue with local computational operations while waiting for a message to be sent or delivered.

• This can be very computationally efficient, but requires care in ensuring data is only used after a message has been received.

• Some of the MPI calls are MPI_ISEND, MPI_IPROBE, and MPI_IRECV.
Information on Message Passing Interface (MPI)

- **Designing and Building Parallel Programs**, Ian Foster
  - Somewhat dated (1995), but an excellent online textbook with detailed discussion about many aspects of HPC. This presentation borrowed heavily from this reference

- **Message Passing Interface (MPI)**, Blaise Barney
  [https://computing.llnl.gov/tutorials/mpi/](https://computing.llnl.gov/tutorials/mpi/)
  - Excellent tutorial on the use of MPI, with both Fortran and C example code