

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

Glenn Johnson

Mary Grabe

Amir Bozorgzadeh

Preston Smith

Information Technology Services

Information Technology Services

Information Technology Services

Information Technology Services

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	<code>MPI_Comm_size (comm,&size)</code>
Fortran	<code>MPI_COMM_SIZE (comm,size,ierr)</code>

Arguments:

Intent	Argument	Type	Description
IN	<code>comm</code>	handle	Communicator
OUT	<code>size</code>	integer	Number of MPI tasks associated with <code>comm</code>

- Determine rank of this MPI task:

C	<code>MPI_Comm_rank (comm,&rank)</code>
Fortran	<code>MPI_COMM_RANK (comm,rank,ierr)</code>

-Rank is the Task ID

Example: Hello World

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:

Intent	Argument	Type	Description
IN	buf	choice	Address of data array to send
IN	count	integer(≥ 0)	Number of array elements to send
IN	datatype	handle	Type of data to send
IN	dest	integer	Rank (task ID) of destination task
IN	tag	integer	Message tag
IN	comm	handle	Communicator

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:

Intent	Argument	Type	Description
OUT	buf	choice	Address of data array to receive
IN	count	integer(≥ 0)	Number of array elements to receive
IN	datatype	handle	Type of data to recv
IN	source	integer	Rank (task ID) of source task
IN	tag	integer	Message tag
IN	comm	handle	Communicator
OUT	status	status	Status object

Point-to-Point Communication Routines

- Blocking vs. Non-blocking
- Synchronous vs. Asynchronous
- Determinism
- Deadlocks, or Race Conditions

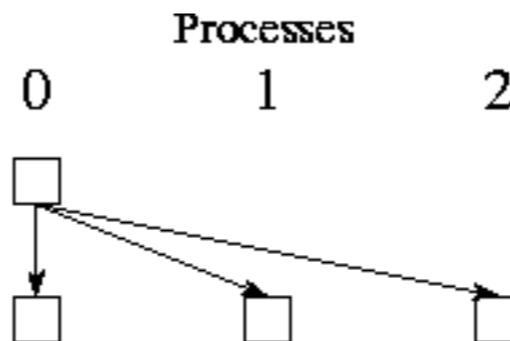
Outline

- General Comments
- Concepts
- Environment Management Routines
- Point-to-Point Communication Routines
- **Collective Communication Routines**
- Asynchronous Communication

Collective Communication Routines

- Broadcast:

(1) MPI_BCAST



C	MPI_Bcast	(<code>&buffer, count, datatype, root, comm</code>)
Fortran	MPI_BCAST	(<code>buffer, count, datatype, root, comm, ierr</code>)

Arguments:

Intent	Argument	Type	Description
INOUT	<code>buffer</code>	choice	Address of input data array, or output data array at root
IN	<code>count</code>	integer(≥ 0)	Number of array elements to receive
IN	<code>datatype</code>	handle	Type of data to recv
IN	<code>root</code>	integer	Rank (task ID) of root task
IN	<code>comm</code>	handle	Communicator

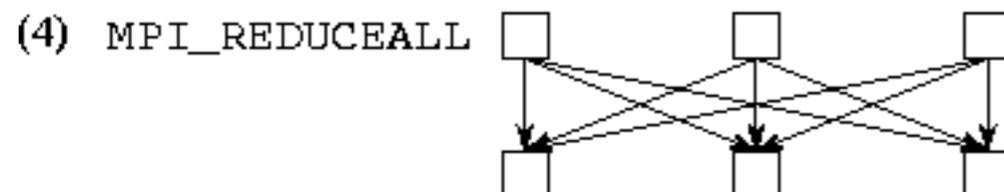
Collective Communication Routines

- 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	MPI_Reduce	(&sendbuf, &recvbuf, count, datatype, op, root, comm)
Fortran	MPI_REDUCE	(sendbuf, recvbuf, count, datatype, op, root, comm, ierr)

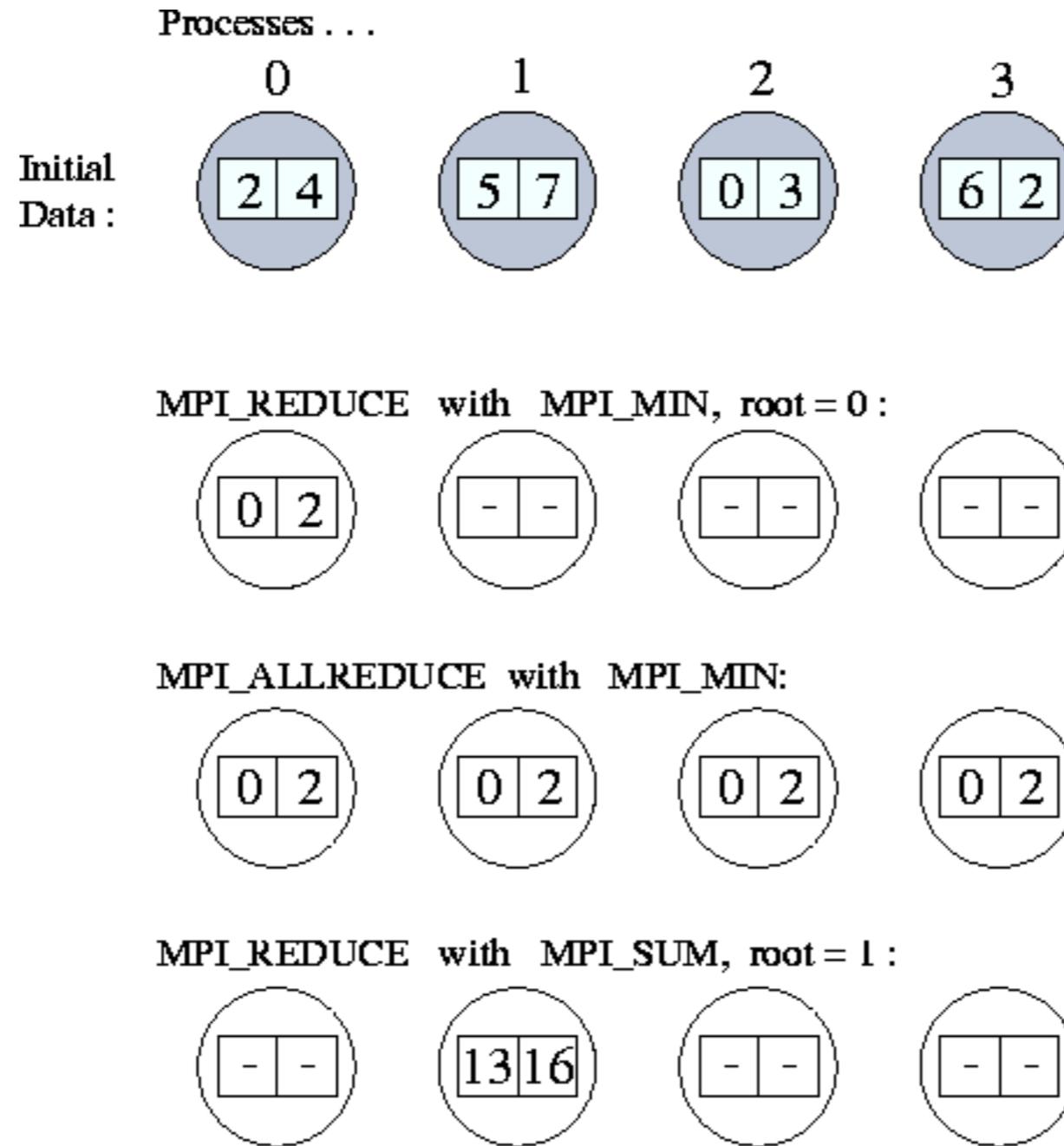
Arguments:

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



Collective Communication Routines

- Example of Reduction Operation:



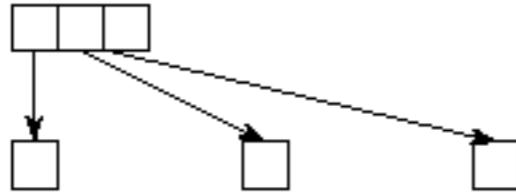
Collective Communication Routines

- Barrier: Wait for all tasks to synchronize

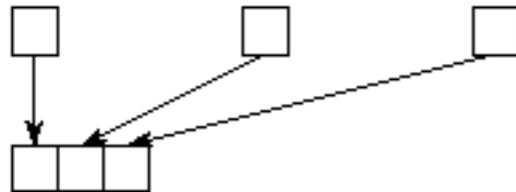
C	MPI_Barrier (comm)
Fortran	MPI_BARRIER (comm,ierr)

- 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`.

References

Information on Message Passing Interface (MPI)

- **Designing and Building Parallel Programs**, Ian Foster
<http://www.mcs.anl.gov/~itf/dbpp/>
-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/>
-Excellent tutorial on the use of MPI, with both Fortran and C example code