Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

High Performance Technical Computing

Introduction to the Message Passing Paradigm – Hands On

Irene Moulitsas

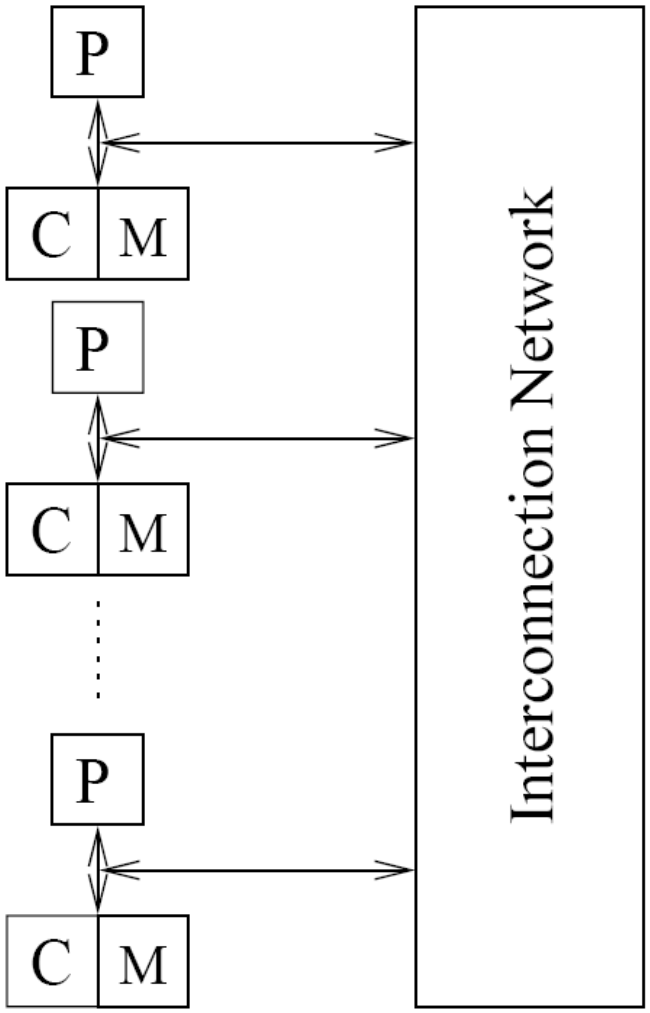
Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Outline

- Message Passing Concepts
  - Blocking vs Non-Blocking
  - Buffered vs Non-Buffered
- MPI Standard
  - Point-to-Point Communication
  - Collective Communication

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Message Passing



- Partitioned Address Space
  - How ?
  - Co-operation
- Explicit Parallel Programming
  - Intellectually demanding
  - Achieves high performance
  - Scales to large number of processors

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Programming Structure

- Asynchronous
  - Hard to reason
  - Non-deterministic behavior
- Loosely synchronous
  - Synchronize to perform interactions
  - Easier to reason
- SPMD
  - Single Program Multiple Data

Send & Receive

```
send(void *sendbuf, int nelems, int dest)
receive(void *recvbuf, int nelems, int source)
```

P0	P1
`a = 100;`	`receive(&a, 1, 0)`
`send(&a, 1, 1);`	`printf("%d\n", a);`
`a = 0;`	

Send & Receive

```
send (sendbuf, integer nelems, integer dest)
<type> sendbuf(*)
integer nelems, dest

receive (recvbuf, integer nelems, integer source)
<type> recvbuf(*)
integer nelems, source
```

P0	P1
a = 100	receive (a, 1, 0)
send (a, 1, 1)	write (*,*) 'a=',
a = 0	a

Blocking Non-Buffered Communication

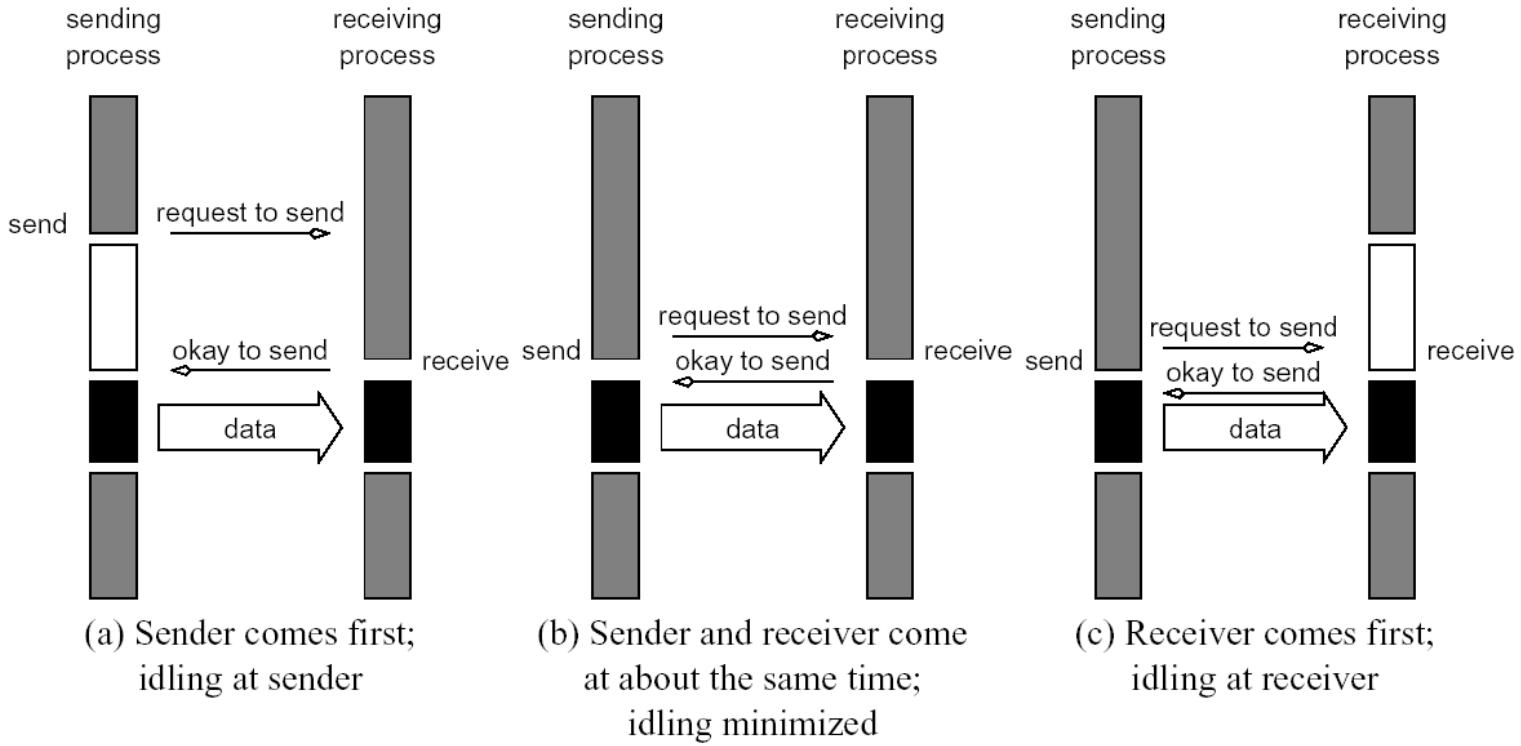


Figure 6.1 Handshake for a blocking non-buffered send/receive operation. It is easy to see that in cases where sender and receiver do not reach communication point at similar times, there can be considerable idling overheads.

Send/Receive Examples

P0

```
a = 100;
send(&a, 1, 1);
a = 0;
```

P1

```
receive(&a, 1, 0)
printf("%d\n", a);
```

P0

```
send(&a, 1, 1);
receive(&b, 1, 1);
```

P1

```
send(&a, 1, 0);
receive(&b, 1, 0);
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Send/Receive Examples

P0

```
a = 100
send (a, 1, 1)
a = 0
```

P1

```
receive (a, 1, 0)
write (*,*) 'a=', a
```

P0

```
send (a, 1, 1)
receive (b, 1, 1)
```

P1

```
send (a, 1, 0)
receive (b, 1, 0)
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Blocking Buffered Communication

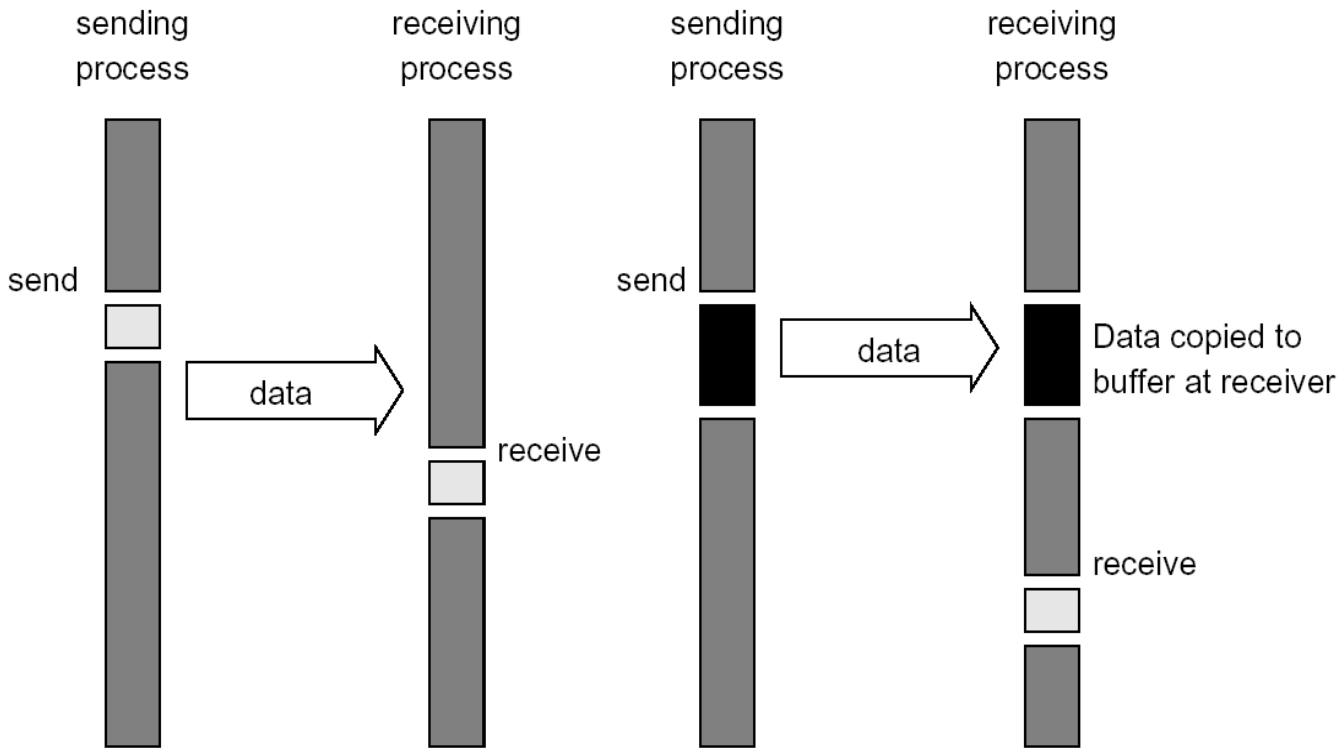


Figure 6.2 Blocking buffered transfer protocols: (a) in the presence of communication hardware with buffers at send and receive ends; and (b) in the absence of communication hardware, sender interrupts receiver and deposits data in buffer at receiver end.

Send/Receive Examples

P0

```
for (i = 0; i < 1000; i++) {
  produce_data(&a);
  send(&a, 1, 1);
}
```

P1

```
for (i = 0; i < 1000; i++) {
  receive(&a, 1, 0);
  consume_data(&a);
}
```

P0

```
receive(&a, 1, 1);
send(&b, 1, 1);
```

P1

```
receive(&a, 1, 0);
send(&b, 1, 0);
```

Send/Receive Examples

P0

```
do i=1, 1000

produce_data(a)
   send (a, 1, 1)
end do
```

P1

```
do i=1, 1000
   receive (a, 1, 0)
   consume_data(a)
end do
```

P0

```
receive (a, 1, 1)
send (b, 1, 1)
```

P1

```
receive (a, 1, 0)
send (b, 1, 0)
```

Non-Blocking Non-Buffered Communication

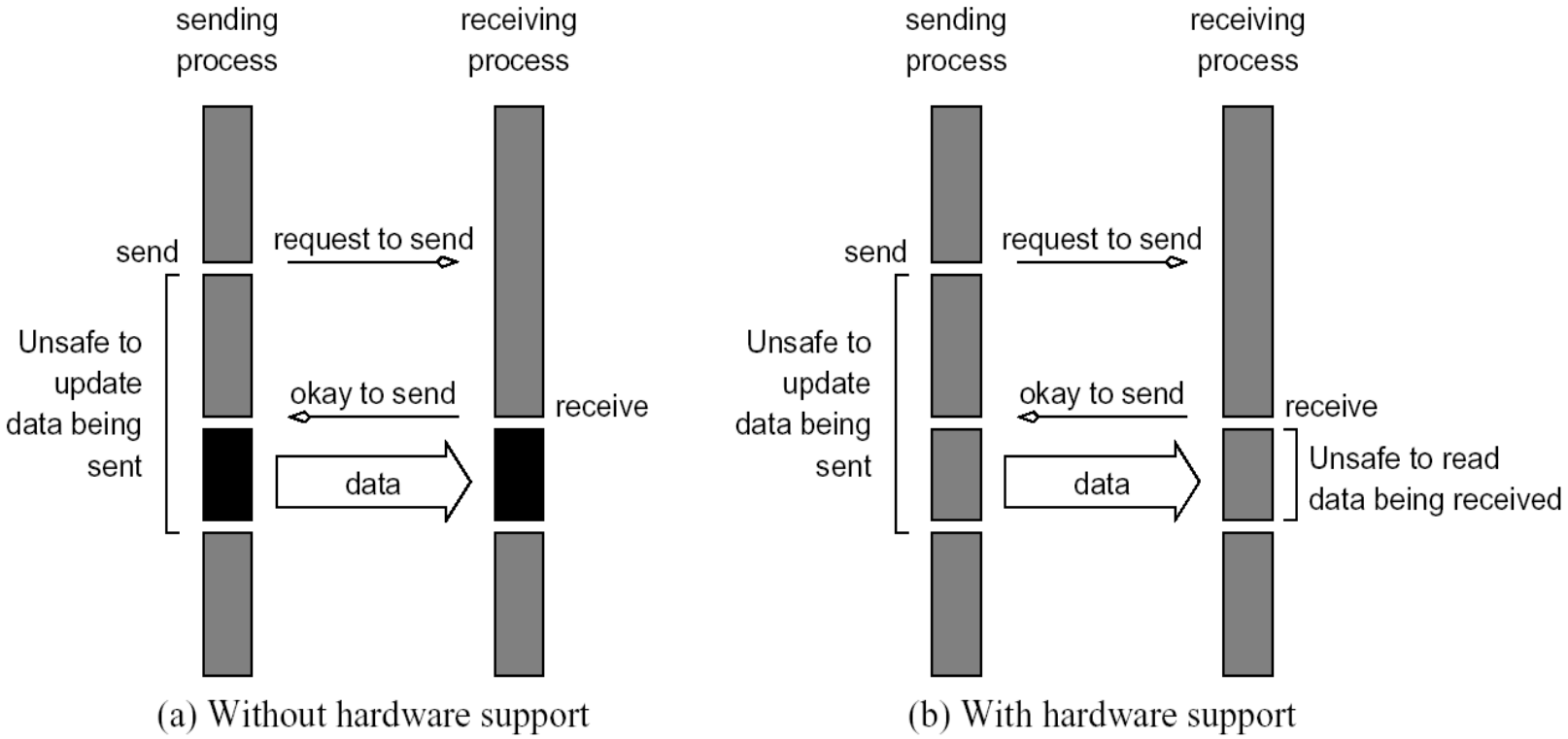


Figure 6.4 Non-blocking non-buffered send and receive operations (a) in absence of communication hardware; (b) in presence of communication hardware.

MPI Background

- MPI : Message Passing Interface
- Began in Supercomputing ’92
  - Vendors
    - IBM, Intel, Cray
  - Library writers
    - PVM
  - Application specialists
    - National Laboratories, Universities

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Why MPI ?

- One of the oldest libraries
- Wide-spread adoption. Portable.
- Minimal requirements on the underlying hardware
- Explicit parallelization

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

MPI Features

- Communicator Information
- Point to Point communication
- Collective Communication
- Topology Support
- Error Handling

Six Golden MPI Functions

- MPI is 125 functions
- MPI has 6 most used functions

Table 6.1 The minimal set of MPI routines.

`MPI_Init`	Initializes MPI.
`MPI_Finalize`	Terminates MPI.
`MPI_Comm_size`	Determines the number of processes.
`MPI_Comm_rank`	Determines the label of the calling process.
`MPI_Send`	Sends a message.
`MPI_Recv`	Receives a message.

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

MPI Functions: Initialization

```
int MPI_Init(int *argc, char ***argv)
int MPI_Finalize()
```

```
MPI_Init (ierror)
integer ierror

MPI_Finalize (ierror)
integer ierror
```

- Must be called by all processes
- MPI_SUCCESS
  - "mpi.h"
  - "mpif.h"

MPI Functions: Communicator

```
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

```
MPI_Comm_Size (comm, size, ierror)
integer comm, size, ierror

MPI_Comm_Rank (comm, rank, ierror)
integer comm, rank, ierror
```

- MPI_Comm
  - Info for the communication domain
  - MPI_COMM_WORLD

Hello World !

```
#include <mpi.h>

main(int argc, char *argv[])
{
   int npes, myrank;

   MPI_Comm_size(MPI_COMM_WORLD, &npes);

   MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
   printf("From process %d out of %d, Hello World!\n",
          myrank, npes);
   MPI_Finalize();
 }
```

Hello World !

```
Program main

include 'mpif.h'

integer npes, myrank, ierror

call MPI_Comm_Size(MPI_COMM_WORLD, npes, ierror)
call MPI_Comm_Rank(MPI_COMM_WORLD, myrank,
                   ierror)

write (*,*) 'From process', myrank, 'out of',
              npes, 'Hello World!'

call MPI_Finalize(ierror)

end
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Hello World !

```
#include <mpi.h>

main(int argc, char *argv[])
{
  int npes, myrank;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &npes);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  printf("From process %d out of %d, Hello World!\n",
         myrank, npes);
  MPI_Finalize();
}
```

Hello World !

```
Program main

include 'mpif.h'

integer npes, myrank, ierror

call MPI_Init(ierror)

call MPI_Comm_Size(MPI_COMM_WORLD, npes, ierror)
call MPI_Comm_Rank(MPI_COMM_WORLD, myrank,
                   ierror)

write (*,*) 'From process', myrank, 'out of',
              npes, 'Hello World!'

call MPI_Finalize(ierror)

end
```

Compile and Run on CRESCENT

- To compile our `hello_world.c` program

```
mpicc hello_world.c -o hello_world
```

- To run our `hello_world` program create a script and

```
qsub sample_script
```

- To check the status of a job

```
qstat -a
```

- To check the status of a job

```
qdel <job_num>
```

Compile and Run on CRESCENT

- To compile our `hello_world.f90` program

```
mpif90 hello_world.f90 -o hello_world
```

- To run our `hello_world` program create a script and

```
qsub sample_script
```

- To check the status of a job

```
qstat -a
```

- To check the status of a job

```
qdel <job_num>
```

Sample script on CRESCENT…

- To find more sample scripts go to `/apps/examples/pbs/`

```
#!/bin/bash
##
## MPI submission script for PBS on CRESCENT
## -----------------------------------------
##
## Follow the 6 steps below to configure your job.
##
## If you edit this from Windows, *before* submitting
## via "qsub" run "dos2unix" on this file - or you will
## get strange errors. You have been warned.
##
## STEP 1:
## The following line contains the job name:
##
#PBS -N mpi4cpu
##
##
```

…Sample script on CRESCENT…

```
## STEP 2:
## Select the number of cpus/cores required by modifying
## the #PBS -l select line below
##
## Normally you select cpus in chunks of 16 cpus
## The Maximum value for ncpus is 16 and mpiprocs MUST
## be the same value as ncpus.
##
## If more than 16 cpus are required then select
## multiple chunks of 16
## e.g. 16 CPUs: select=1:ncpus=16:mpiprocs=16
##      32 CPUs: select=2:ncpus=16:mpiprocs=16
##      48 CPUs: select=3:ncpus=16:mpiprocs=16
##      ..etc..
##
#PBS -l select=1:ncpus=16:mpiprocs=16
##
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

…Sample script on CRESCENT…

```
##
## STEP 3:
##
## Select the correct queue by modifying the
## #PBS -q line below
##
## half_hour     -  30 minutes
## one_hour      -   1 hour
## half_day      -  12 hours
## one_day       -  24 hours
## two_day       -  48 hours
## five_day      - 120 hours
## ten_day       - 240 hours (by special arrangement)
##
#PBS -q half_hour
##
```

…Sample script on CRESCENT…

```
## STEP 4:
##
## Replace the hpc@cranfield.ac.uk email address with
## your Cranfield email on the #PBS -M line below:
##
#PBS -m abe
#PBS -M hpc@cranfield.ac.uk
##
## DO NOT CHANGE the following lines
##--------------------------------------------
#PBS -j oe
#PBS -W sandbox=PRIVATE
#PBS -k n
ln -s $PWD $PBS_O_WORKDIR/$PBS_JOBID
## Change to working directory
cd $PBS_O_WORKDIR
## Calculate number of CPUs
export cpus=`cat $PBS_NODEFILE | wc -l`
```

…Sample script on CRESCENT…

```
## STEP 5:
##
##  Load the default application environment
##  For a specific version add the version number, e.g.
##  module load intel/2016b
##
module load intel
##
##-------------------------------------------
##
```

…Sample script on CRESCENT

```
##
## STEP 6:
##
## Run MPI code
##
## The main parameter to modify is your mpi program name
## - change YOUR_EXECUTABLE to your own filename
##

mpirun -machinefile $PBS_NODEFILE -np ${cpus}
   YOUR_EXECUTABLE

## Tidy up the log directory
## DO NOT CHANGE THE LINE BELOW
## ============================
rm $PBS_O_WORKDIR/$PBS_JOBID
```

MPI Functions: Send, Recv

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype,
        int dest, int tag, MPI_Comm comm)
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
        int source, int tag, MPI_Comm comm, MPI_Status *status)
```

- source
  - MPI_ANY_SOURCE
- Tag
  - MPI_ANY_TAG

- MPI_Status
  - MPI_SOURCE
  - MPI_TAG
  - MPI_ERROR

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

MPI Functions: Send, Recv

```
MPI_Send(buf, count, datatype, dest, tag, comm,
         ierror)

MPI_Recv(buf, count, datatype, source, tag,
                comm, status, ierror)

<type> buf(*)
integer count, datatype, dest, source, tag,
         comm, status(MPI_STATUS_SIZE), ierror
```

- Source
  - MPI_ANY_SOURCE
- Tag
  - MPI_ANY_TAG

- MPI_Status
  - MPI_SOURCE
  - MPI_TAG
  - MPI_ERROR

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

MPI Functions: C language Datatypes

Table 6.2 Correspondence between the datatypes supported by MPI and those supported by C.

MPI Datatype	C Datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

MPI Functions: FORTRAN Datatypes

MPI datatype	Fortran datatype
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

Example in C

- A processor sends 2 messages

```
1   int a[10], b[10], myrank;
2   MPI_Status status;
3   ...
4   MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
5   if (myrank == 0) {
6     MPI_Send(a, 10, MPI_INT, 1, 1, MPI_COMM_WORLD);
7     MPI_Send(b, 10, MPI_INT, 1, 2, MPI_COMM_WORLD);
8   }
9   else if (myrank == 1) {
10    MPI_Recv(b, 10, MPI_INT, 0, 2, MPI_COMM_WORLD);
11    MPI_Recv(a, 10, MPI_INT, 0, 1, MPI_COMM_WORLD);
12  }
13  ...
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Example in FORTRAN

- A processor sends 2 messages

```
integer a(10), b(10), status(MPI_STATUS_SIZE), myrank
…
call MPI_Comm_Rank(MPI_COMM_WORLD, myrank, ierror)
if (myrank .eq. 0)
   call MPI_Send(a, 10, MPI_INTEGER, 1, 1, MPI_COMM_WORLD,
                 ierror)
   call MPI_Send(b, 10, MPI_INTEGER, 1, 2, MPI_COMM_WORLD,
                 ierror)
else if (myrank .eq. 1)
call MPI_Recv(b,10, MPI_INTEGER, 0, 2, MPI_COMM_WORLD,
                 status, ierror)
call MPI_Recv(a,10, MPI_INTEGER, 0, 1, MPI_COMM_WORLD,
                 status, ierror)
endif
…
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Example in C

- Circular fashion communication

```
int a[10], b[10], npes, myrank;
MPI_Status status;
...
MPI_Comm_size(MPI_COMM_WORLD, &npes);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Send(a, 10, MPI_INT, (myrank+1)%npes, 1, MPI_COMM_WORLD);
MPI_Recv(b, 10, MPI_INT, (myrank-1+npes)%npes, 1, MPI_COMM_WORLD);
...
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Example in FORTRAN

- Circular fashion communication

```
integer a(10), b(10), npes, myrank,
status(MPI_STATUS_SIZE)
…
call MPI_Comm_Size(MPI_COMM_WORLD, npes, ierror)
call MPI_Comm_Rank(MPI_COMM_WORLD, myrank,
                   ierror)

call MPI_Send(a, 10, MPI_INTEGER,
              (myrank+1)%npes, 1,
          MPI_COMM_WORLD, ierror)
call MPI_Recv(b, 10, MPI_INTEGER,
              (myrank-1+npes)%npes, 1,
          MPI_COMM_WORLD, status, ierror)
…
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Example in C

```
int a[10], b[10], npes, myrank;
MPI_Status status;
...
MPI_Comm_size(MPI_COMM_WORLD, &npes);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank%2 == 1) {
  MPI_Send(a, 10, MPI_INT, (myrank+1)%npes, 1, MPI_COMM_WORLD);
  MPI_Recv(b, 10, MPI_INT, (myrank-1+npes)%npes, 1, MPI_COMM_WORLD);
}
else {
  MPI_Recv(b, 10, MPI_INT, (myrank-1+npes)%npes, 1, MPI_COMM_WORLD);
  MPI_Send(a, 10, MPI_INT, (myrank+1)%npes, 1, MPI_COMM_WORLD);
}
...
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Example in FORTRAN

```
integer a(10), b(10), npes, myrank,
status(MPI_STATUS_SIZE)
…
call MPI_Comm_Size(MPI_COMM_WORLD, npes, ierror)
call MPI_Comm_Rank(MPI_COMM_WORLD, myrank, ierror)
if (mod(myrank,2) .eq. 1) then
  call MPI_Send(a, 10, MPI_INTEGER, (myrank+1)%npes, 1,
                MPI_COMM_WORLD, ierror)
  call MPI_Recv(b, 10, MPI_INTEGER, (myrank-1+npes)%npes,
                1, MPI_COMM_WORLD, status, ierror)
else
  call MPI_Recv(b, 10, MPI_INTEGER, (myrank-1+npes)%npes,
                1, MPI_COMM_WORLD, status, ierror)
  call MPI_Send(a, 10, MPI_INTEGER, (myrank+1)%npes, 1,
                MPI_COMM_WORLD, ierror)
endif
…
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

MPI Functions: SendRecv

```
int MPI_Sendrecv(void *sendbuf, int sendcount,
        MPI_Datatype senddatatype, int dest, int sendtag,
        void *recvbuf, int recvcount, MPI_Datatype recvdatatype,
        int source, int recvtag, MPI_Comm comm,
        MPI_Status *status)
```

MPI Functions: SendRecv

```
MPI_SendRecv(sendbuf, sendcount, senddatatype,
             dest, sendtag, recvbuf, recvcount,
     recvdatatype, source, recvtag,
     comm, status, ierror)


<type> sendbuf(*), recvbuf(*)
integer sendcount, senddatatype, dest, sendtag,
     recvcount, recvdatatype, source, recvtag,
     comm, status(MPI_STATUS_SIZE), ierror
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Example in C

```
int a[10], b[10], npes, myrank;
MPI_Status status;
...
MPI_Comm_size(MPI_COMM_WORLD, &npes);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_SendRecv(a, 10, MPI_INT, (myrank+1)%npes, 1,
             b, 10, MPI_INT, (myrank-1+npes)%npes, 1,
             MPI_COMM_WORLD, &status);
...
```

Example in FORTRAN

```
integer a(10), b(10), npes, myrank,
status(MPI_STATUS_SIZE)
…
call MPI_Comm_Size(MPI_COMM_WORLD, npes, ierror)
call MPI_Comm_Rank(MPI_COMM_WORLD, myrank,
                   ierror)

call MPI_SendRecv(a, 10, MPI_INTEGER,
                  (myrank+1)%npes, 1, b, 10,
                   MPI_INTEGER,
                  (myrank-1+npes)%npes, 1,
            MPI_COMM_WORLD, status, ierror)
…
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

MPI Functions: ISend, IRecv

```
int MPI_Isend(void *buf, int count, MPI_Datatype datatype,
        int dest, int tag, MPI_Comm comm, MPI_Request *request)
int MPI_Irecv(void *buf, int count, MPI_Datatype datatype,
        int source, int tag, MPI_Comm comm, MPI_Request *request)
```

- Non-blocking
- MPI_Request

MPI Functions: ISend, IRecv

```
MPI_ISend(buf, count, datatype, dest, tag,
          comm, request, ierror)
MPI_IRecv(buf, count, datatype, source, tag,
          comm, request, ierror)


<type> buf(*)
integer count, datatype, dest, tag, source,
          comm, request, ierror
```

- Non-blocking
- MPI_Request

MPI Functions: Test, Wait

```
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)
int MPI_Wait(MPI_Request *request, MPI_Status *status)
```

- MPI_Test tests if operation finished
  - Flag = True ? False
- MPI_Wait blocks until operation is finished

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

MPI Functions: Test, Wait

```
MPI_Test(request, flag, status, ierror)
MPI_Wait(request, status, ierror)

logical flag
integer request, status(MPI_STATUS_SIZE), ierror
```

- MPI_Test tests if operation finished
  - Flag = True ? False
- MPI_Wait blocks until operation is finished

Example in C

- A processor sends 2 messages

```
1   int a[10], b[10], myrank;
2   MPI_Status status;
3   MPI_Request requests[2];
4   ...
5   MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
6   if (myrank == 0) {
7     MPI_Send(a, 10, MPI_INT, 1, 1, MPI_COMM_WORLD);
8     MPI_Send(b, 10, MPI_INT, 1, 2, MPI_COMM_WORLD);
9   }
10  else if (myrank == 1) {
11    MPI_Irecv(b, 10, MPI_INT, 0, 2, &requests[0], MPI_COMM_WORLD);
12    MPI_Irecv(a, 10, MPI_INT, 0, 1, &requests[1], MPI_COMM_WORLD);
13  }
14  ...
```

Example in FORTRAN

- A processor sends 2 messages

```
integer a(10), b(10), status(MPI_STATUS_SIZE), myrank,
requests(2)
…
call MPI_Comm_Rank(MPI_COMM_WORLD, myrank, ierror)
if (myrank .eq. 0)
   call MPI_Send(a, 10, MPI_INTEGER, 1, 1, MPI_COMM_WORLD,
                 ierror)
   call MPI_Send(b, 10, MPI_INTEGER, 1, 2, MPI_COMM_WORLD,
                 ierror)
else if (myrank .eq. 1)
   call MPI_IRecv(b, 10, MPI_INTEGER, 0, 2, requests(0),
                  MPI_COMM_WORLD, ierror)
   call MPI_IRecv(a, 10, MPI_INTEGER, 0, 1, requests(1),
                  MPI_COMM_WORLD, ierror)
endif
…
```

MPI Functions: Synchronize

```
int MPI_Barrier(MPI_Comm comm)
```

•Returns only after all of the processes have called this function

MPI Functions: Synchronize

```
MPI_Barrier(comm, ierror)

integer comm, ierror
```

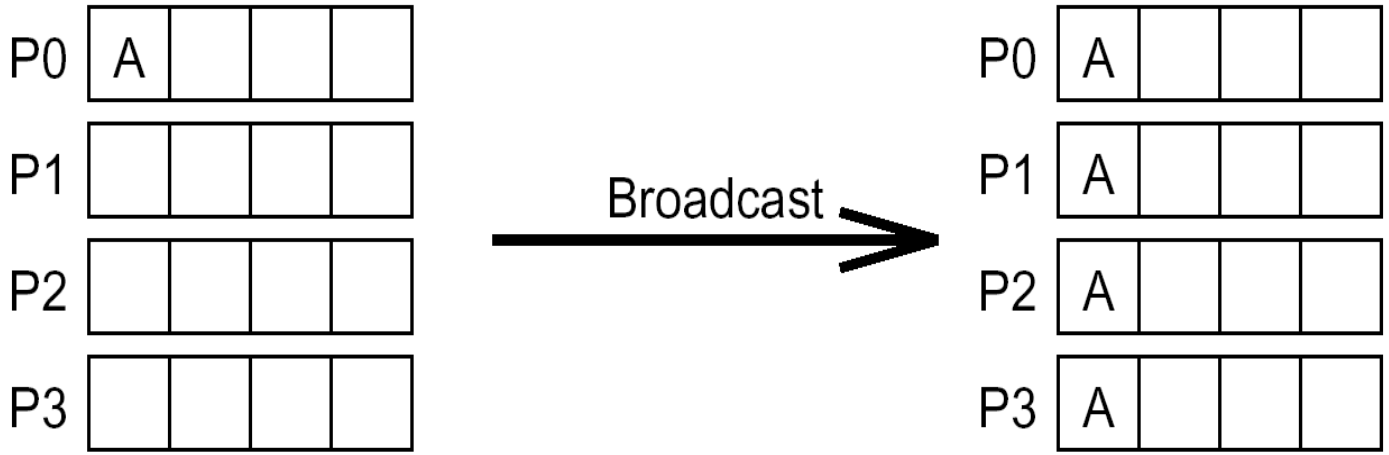
- Returns only after all of the processes have called this function

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Collective Communications

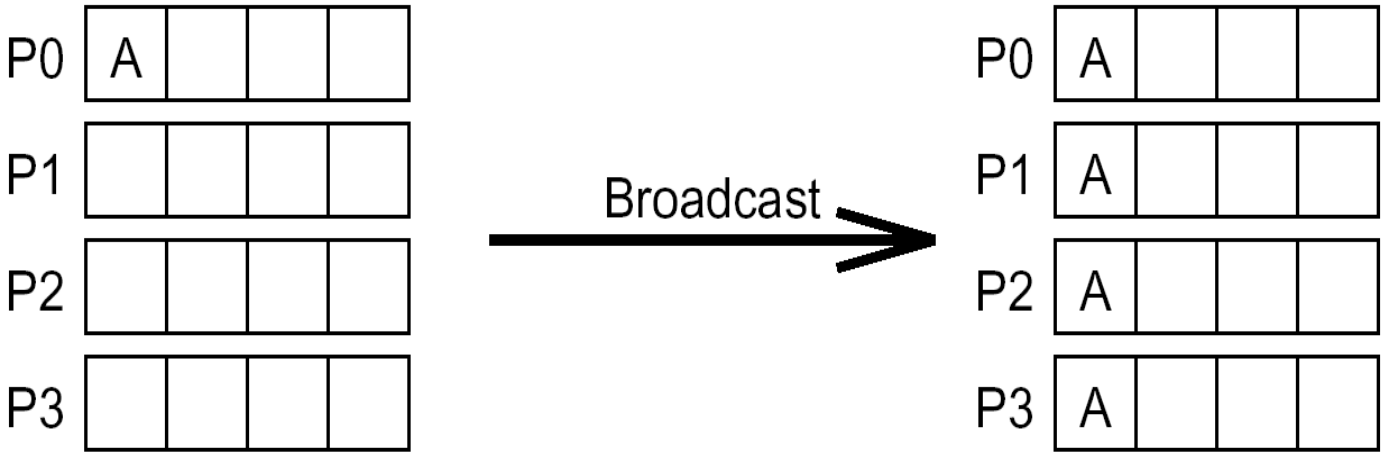
- One-to-All Broadcast
- All-to-One Reduction
- All-to-All Broadcast & Reduction
- All-Reduce & Prefix-Sum
- Scatter and Gather
- All-to-All Personalized

MPI Functions: Broadcast



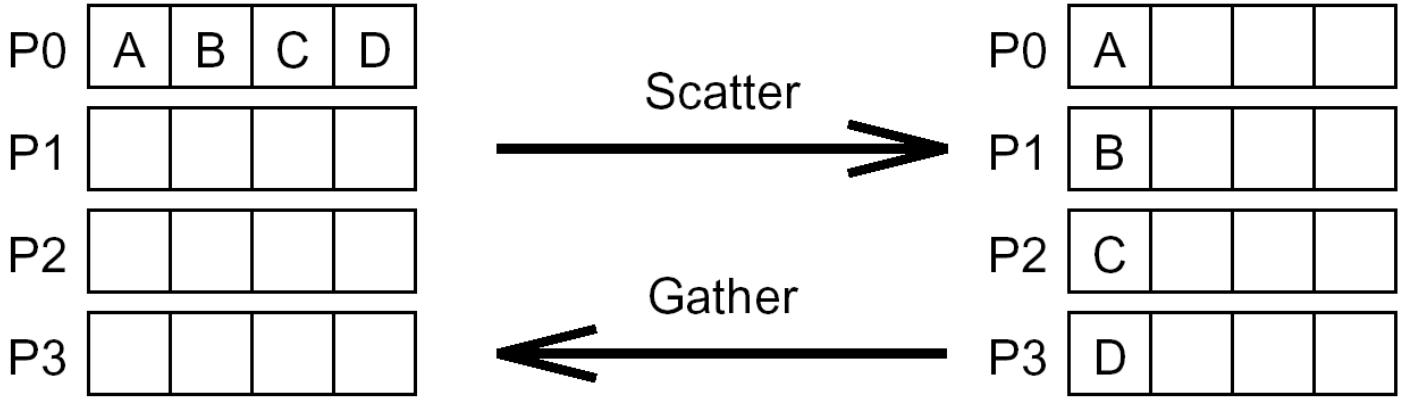

```
int MPI_Bcast(void *buf, int count, MPI_Datatype datatype,
        int source, MPI_Comm comm)
```

MPI Functions: Broadcast



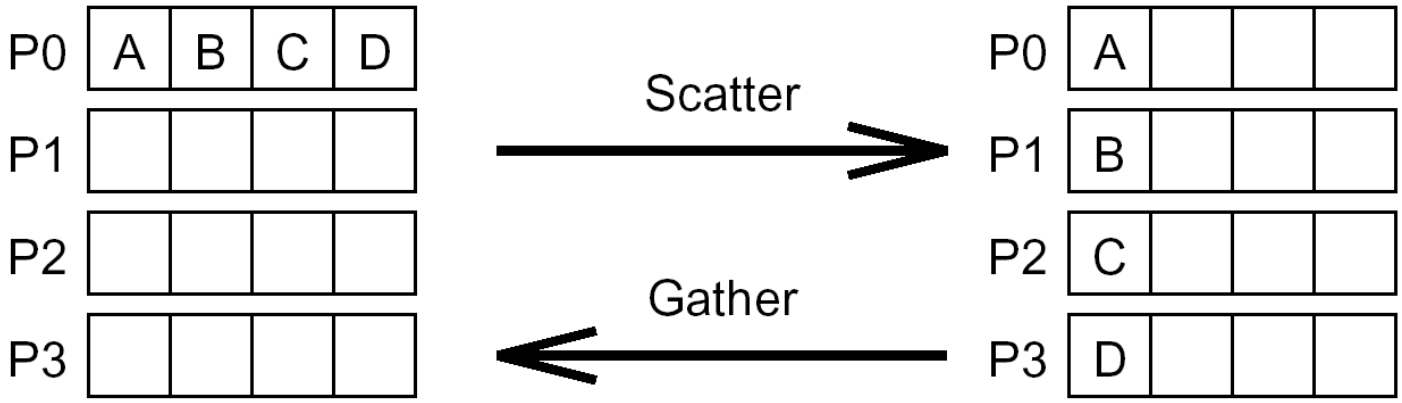

```
MPI_Bcast(buf, count, datatype, source,
     comm,  ierror)

<type> buf(*)
integer count, datatype, source, comm, ierror
```

MPI Functions: Scatter & Gather



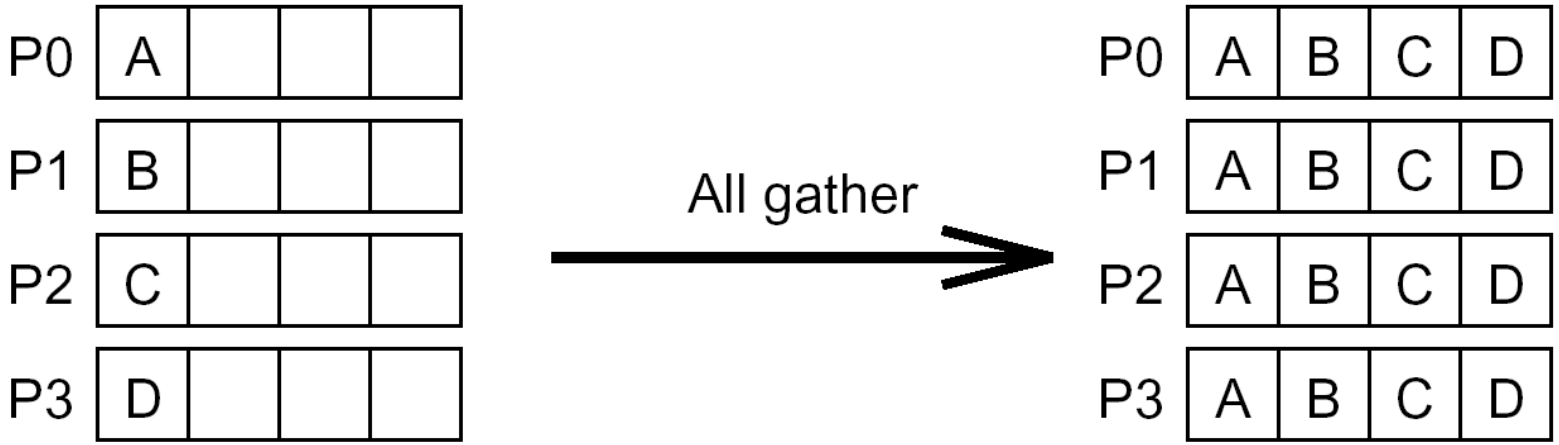

```
int MPI_Scatter(void *sendbuf, int sendcount,
        MPI_Datatype senddatatype, void *recvbuf, int recvcount,
        MPI_Datatype recvdatatype, int source, MPI_Comm comm)

 int MPI_Gather(void *sendbuf, int sendcount,
         MPI_Datatype senddatatype, void *recvbuf, int recvcount,
         MPI_Datatype recvdatatype, int target, MPI_Comm comm)
```

MPI Functions: Scatter & Gather



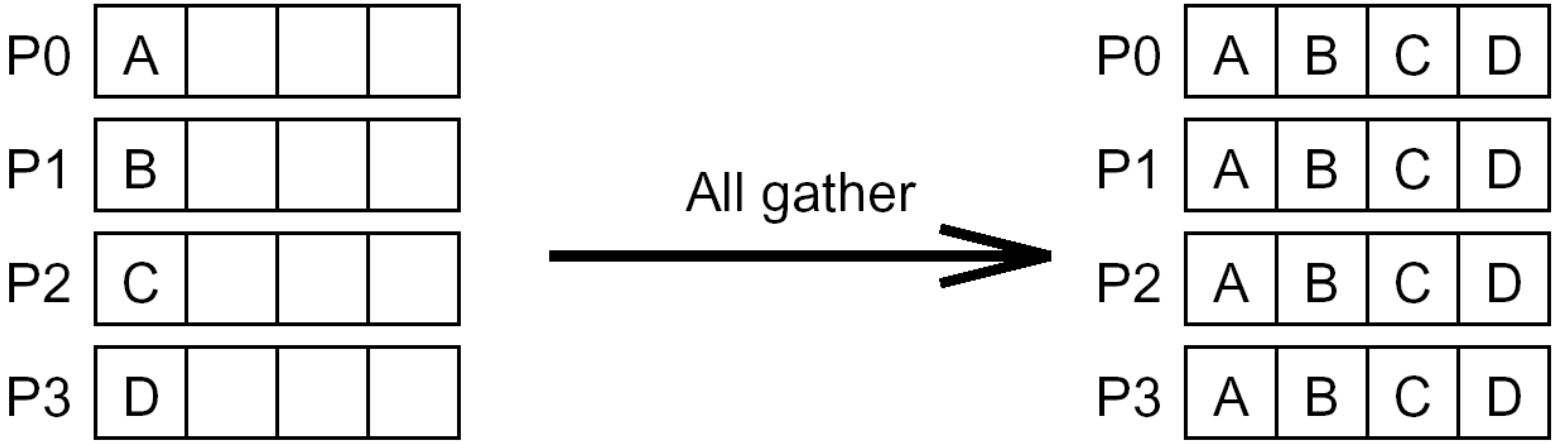

```
MPI_Scatter(sendbuf, sendcount, senddatatype, recvbuf,
      recvcount, recvdatatype, source, comm, ierror)
MPI_Gather(sendbuf, sendcount, senddatatype, recvbuf,
      recvcount, recvdatatype, target, comm, ierror)

<type> sendbuf(*), recvbuf(*)
integer sendcount, senddatatype, recvcount, recvdatatype,
      source, target, comm, ierror
```

MPI Functions: All Gather



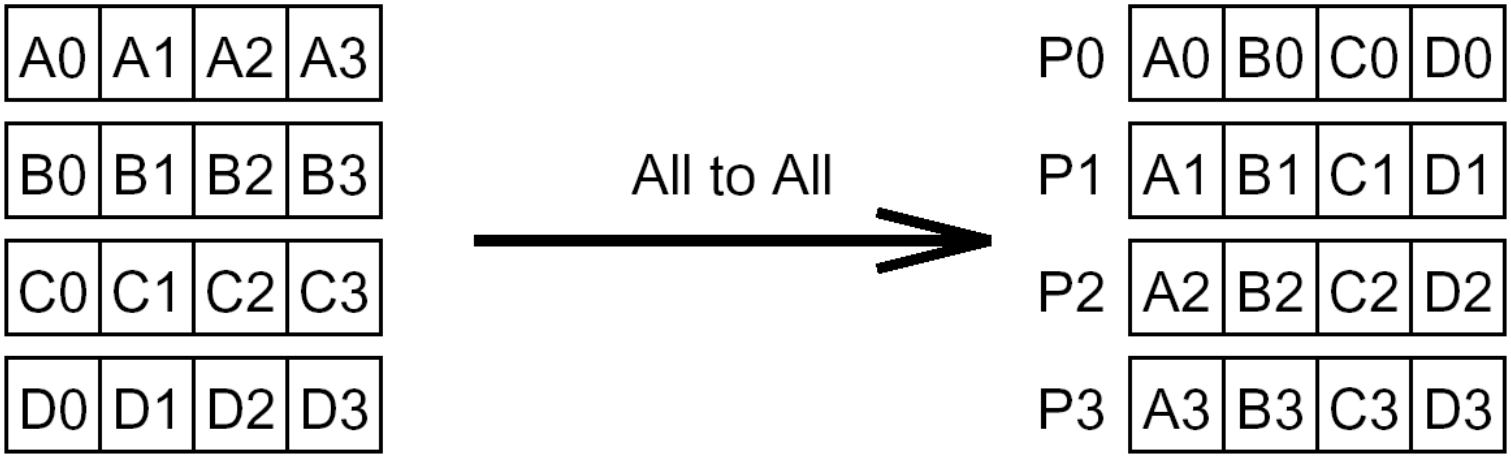

```
int MPI_Allgather(void *sendbuf, int sendcount,
        MPI_Datatype senddatatype, void *recvbuf, int recvcount,
        MPI_Datatype recvdatatype, MPI_Comm comm)
```

MPI Functions: All Gather



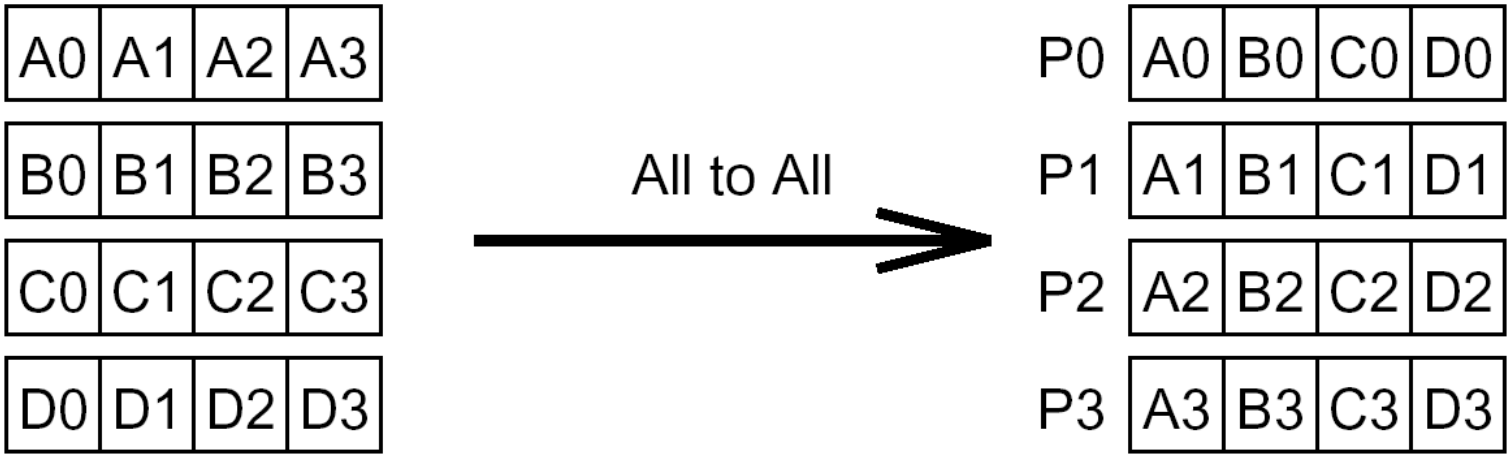

```
MPI_AllGather(sendbuf, sendcount, senddatatype,
     recvbuf, recvcount, recvdatatype, comm,
     ierror)

<type> sendbuf(*), recvbuf(*)
integer sendcount, senddatatype, recvcount,
      recvdatatype, comm, ierror
```

MPI Functions: All-to-All Personalized



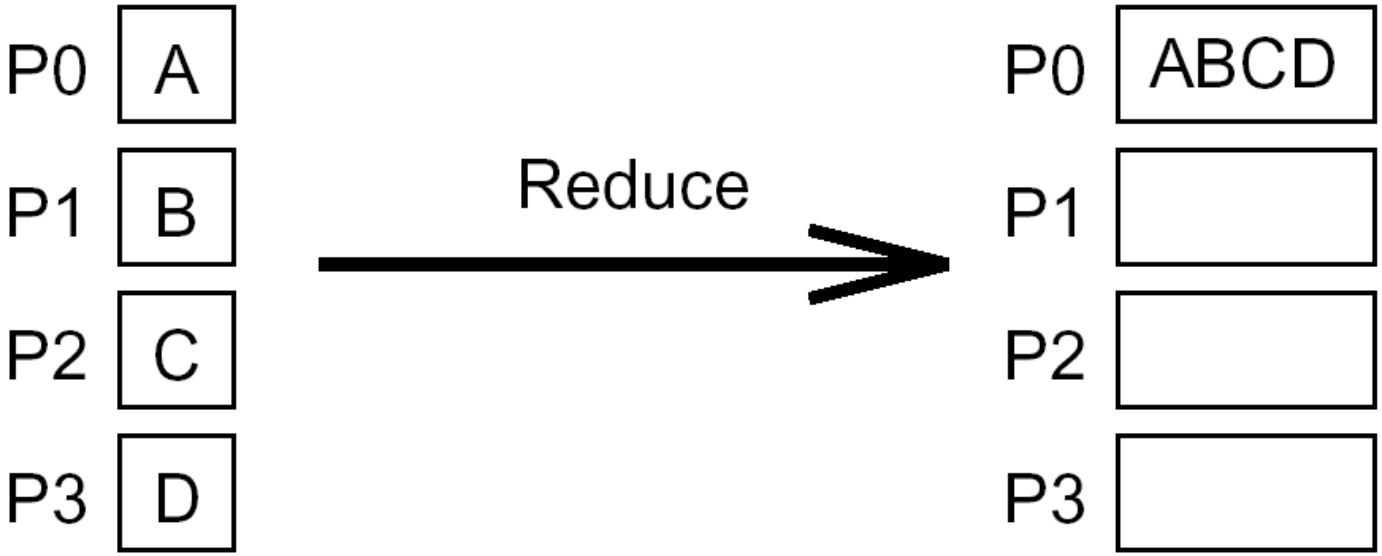

```
int MPI_Alltoall(void *sendbuf, int sendcount,
        MPI_Datatype senddatatype, void *recvbuf, int recvcount,
        MPI_Datatype recvdatatype, MPI_Comm comm)
```

MPI Functions: All-to-All Personalized

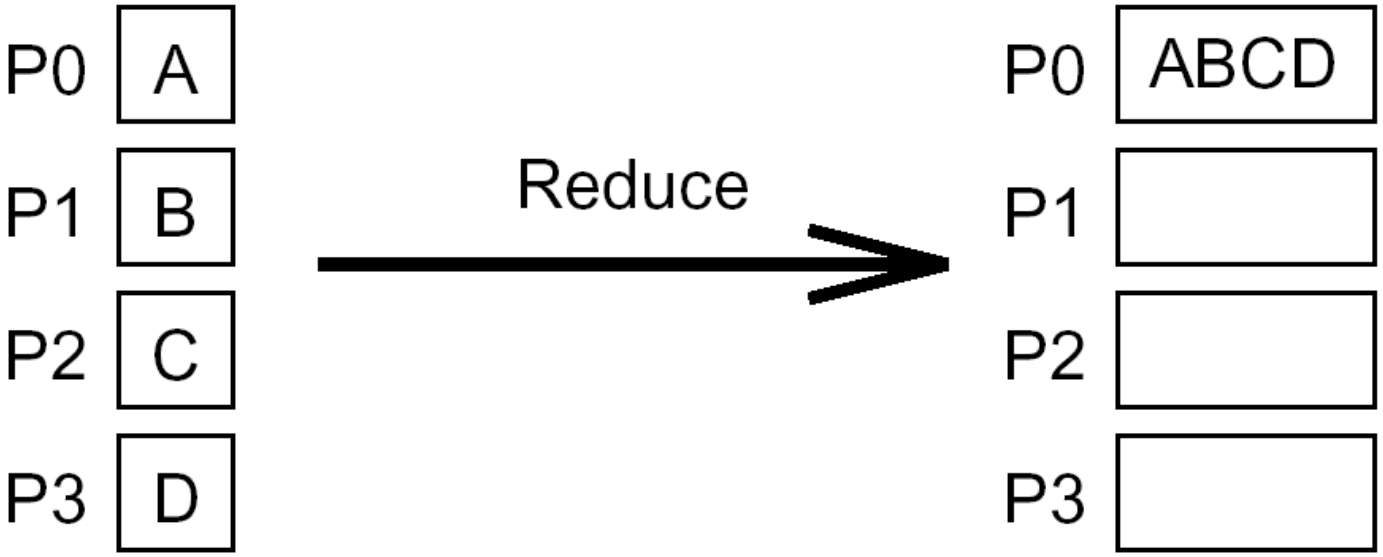



```
MPI_AllToAll(sendbuf, sendcount, senddatatype,
     recvbuf, recvcount, recvdatatype, comm,
     ierror)

<type> sendbuf(*), recvbuf(*)
integer sendcount, senddatatype, recvcount,
      recvdatatype, comm, ierror
```

MPI Functions: Reduction



MPI Functions: Reduction




```
MPI_Reduce(sendbuf, recvbuf, count, datatype, op,
target, comm, ierror)

<type> sendbuf(*), recvbuf(*)
integer count, datatype, op, target, comm, ierror
```

MPI Functions: Operations

Table 6.3 Predefined reduction operations.

Operation	Meaning	Datatypes
MPI_MAX	Maximum	C integers and floating point
MPI_MIN	Minimum	C integers and floating point
MPI_SUM	Sum	C integers and floating point
MPI_PROD	Product	C integers and floating point
MPI_LAND	Logical AND	C integers
MPI_BAND	Bit-wise AND	C integers and byte
MPI_LOR	Logical OR	C integers
MPI_BOR	Bit-wise OR	C integers and byte
MPI_LXOR	Logical XOR	C integers
MPI_BXOR	Bit-wise XOR	C integers and byte
MPI_MAXLOC	max-min value-location	Data-pairs
MPI_MINLOC	min-min value-location	Data-pairs

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

MPI Functions: All-reduce

- Same as MPI_Reduce, but all processes receive the result of MPI_Op operation.

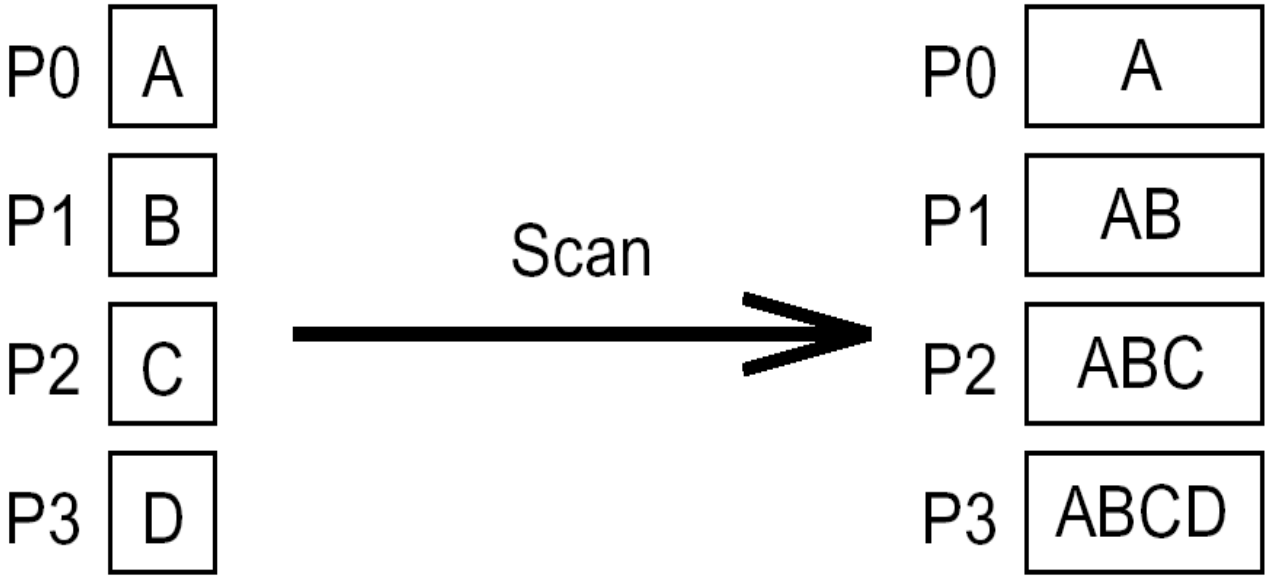
```
int MPI_Allreduce(void *sendbuf, void *recvbuf, int count,
        MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
```

MPI Functions: All-reduce

- Same as MPI_Reduce, but all processes receive the result of MPI_Op operation.

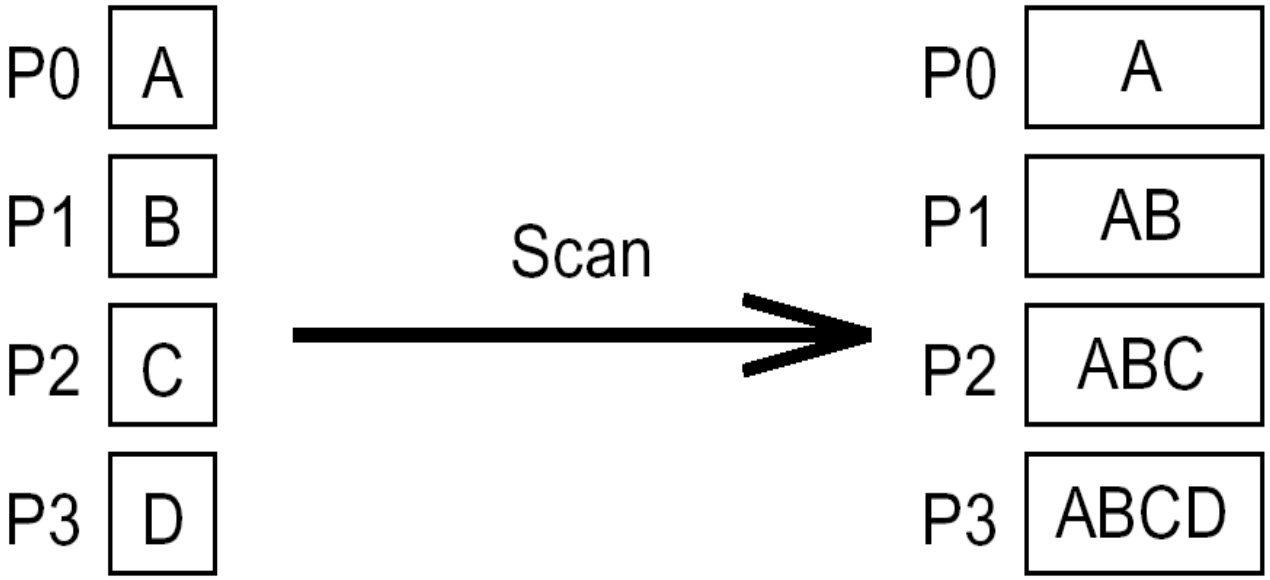
```
MPI_AllReduce(sendbuf, recvbuf, count,
     datatype, op, comm, ierror)

<type> sendbuf(*), recvbuf(*)
integer count, datatype, op, comm, ierror
```

MPI Functions: Prefix Scan




```
int MPI_Scan(void *sendbuf, void *recvbuf, int count,
        MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
```

MPI Functions: Prefix Scan




```
MPI_Scan(sendbuf, recvbuf, count, datatype, op,
      comm, ierror)

<type> sendbuf(*), recvbuf(*)
integer count, datatype, op, comm, ierror
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

MPI Names

Table 4.2 MPI names of the various operations discussed in this chapter.

Operation	MPI Name
One-to-all broadcast	MPI_Bcast
All-to-one reduction	MPI_Reduce
All-to-all broadcast	MPI_Allgather
All-to-all reduction	MPI_Reduce_scatter
All-reduce	MPI_Allreduce
Gather	MPI_Gather
Scatter	MPI_Scatter
All-to-all personalized	MPI_Alltoall

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

MPI Functions: Topology

- Cartesian Topology

```
int MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims,
        int *periods, int reorder, MPI_Comm *comm_cart)
```

```
int MPI_Cart_rank(MPI_Comm comm_cart, int *coords, int *rank)
int MPI_Cart_coord(MPI_Comm comm_cart, int rank, int maxdims,
        int *coords)
```

Performance Evaluation

- Elapsed (wall-clock) time

```
double t1, t2;
t1 = MPI_Wtime();
...
t2 = MPI_Wtime();
printf( "Elapsed time is %f\n", t2 - t1 );
```

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Impact of Memory: Example

- Consider the following code fragment:

```
for (i = 0; i < 1000; i++)
     column_sum[i] = 0.0;
    for (j = 0; j < 1000; j++)
          column_sum[i] += b[j][i];
```

- The code fragment sums columns of the matrix b into a vector column_sum.

- Consider the case where matrices are stored in a row-wise fashion.

Dr. Irene Moulitsas
i.moulitsas@cranfield.ac.uk

Impact of Memory: Example

- We can fix the above code as follows:

```
for (i = 0; i < 1000; i++)
    column_sum[i] = 0.0;
for (j = 0; j < 1000; j++)
    for (i = 0; i < 1000; i++)
        column_sum[i] += b[j][i];
```

- In this case, the matrix is traversed in a row-order and performance can be expected to be significantly better.

Impact of Memory: Example

- Use your data while it's there!

- Given three vectors: `v_a, v_b,` and `v_c`, compute the sum of each one's elements.

```
sum_a = 0;
sum_b = 0;
sum_c = 0;
for (i=0; i<1000; i++){
    sum_a += v_a[i];
    sum_b += v_b[i];
    sum_c += v_c[i];
}
```

```
sum_a = 0;
sum_b = 0;
sum_c = 0;
for (i = 0; i < 1000; i++)
    sum_a += v_a[i];
for (i = 0; i < 1000; i++)
    sum_b += v_b[i];
for (i = 0; i < 1000; i++)
    sum_c += v_c[i];
```

Matrix/Vector Multiply

gram 6.4 Row-wise Matrix-Vector Multiplication

```
RowMatrixVectorMultiply(int n, double *a, double *b, double *x,
                        MPI_Comm comm)
{
  int i, j;
  int nlocal;         /* Number of locally stored rows of A */
  double *fb;         /* Will point to a buffer that stores the entire vector b */
  int npes, myrank;
  MPI_Status status;

  /* Get information about the communicator */
  MPI_Comm_size(comm, &npes);
  MPI_Comm_rank(comm, &myrank);

  /* Allocate the memory that will store the entire vector b */
  fb = (double *)malloc(n*sizeof(double));

  nlocal = n/npes;

  /* Gather the entire vector b on each processor using MPI's ALLGATHER operation */
  MPI_Allgather(b, nlocal, MPI_DOUBLE, fb, nlocal, MPI_DOUBLE,
comm);

/* Perform the matrix-vector multiplication involving the locally stored submatrix */
for (i=0; i<nlocal; i++) {
  x[i] = 0.0;
  for (j=0; j<n; j++)
    x[i] += a[i*n+j]*fb[j];
}

free(fb);
}
```