Message Passing (cont...)

Submitting your jobs (all files in /opt/tools/mpi)

- Job Name

- n = num of nodes (Max 8)
- Max time for Job

- All transactions relative to this directory

- np → num of proc
- Name of parallel executable

```
#!/bin/sh
#PBS -N HelloWorld
# nodes is the number of computers to use. Adjust as needed. MAX is
8. Leave ppn alone. Adjust walltime to max time to allow your binary
to run.
#PBS -l nodes=4:ppn=32, mem=2GB , walltime=00:00:15
#PBS -q batch
# If you want email notifications, remove the extra # in front of
the #PBS -M and #PBS -m lines
##PBS -M <YOUR EMAIL ADDRESS HERE>
##PBS -m abe
# Set the correct PATH and env for the MPI implelementation
#/usr/lib64/{openmpi,mpich,mvapich2}/bin
MPIRUN='/usr/lib64/openmpi/bin/mpirun -mca plm_rsh_agent rsa --map-
by node --display-allocation -mca orte_base_help_aggregate 0 -mca
btl ^openib'
WORK_HOME=/home/ashish/MPI/HelloWorld
cd $WORK_HOME
$MPIRUN -np 6 --machinefile $PBS_NODEFILE ./hello_world
```

0. Simple Setup

(5mins)

- Work individually or 2/group
- Login
- Makefile and PBS scripts (git)

1. Hello World

(15mins)

1. Each node figures out it's rank/identity
2. Each node figures out size of community
3. Print "I am rank/size"

- Edit and run on 4 and 8 nodes

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv) {
    ...
}
```

Setup/Cleanup

```
MPI_Init(&argc, &argv);
MPI_Finalize();
```

Gather Information

```
MPI_Comm_rank (MPI_COMM_WORLD,
&my_rank);

MPI_Comm_size (MPI_COMM_WORLD,
&num_procs);
```

2. Comm.. in a ring

(10mins)

1. 0→1, 1→2, .. N→0
2. Send your rank
3. Print "<rank> got <data> from <sender>"

- Run on 8 nodes

Basics

MPI_Init(&argc, &argv);

MPI_Finalize();

MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);

MPI_Comm_size (MPI_COMM_WORLD, &num_procs);

Send/Recv.

```
MPI_Send (&message,  sizeof(message),
          MPI_INT,  destination,  tag,
          MPI_COMM_WORLD);

MPI_Recv (&message, sizeof(message)+1,
          MPI_INT,  source,  tag,
          MPI_COMM_WORLD, &status)

MPI_Status status
```

3. Broadcast

(10mins)

1. Node 0 will send a number (say 27)
2. Nodes !0 will receive the number
3. Nodes !0 will print rank, size and number received.

- Edit and run on 8 nodes

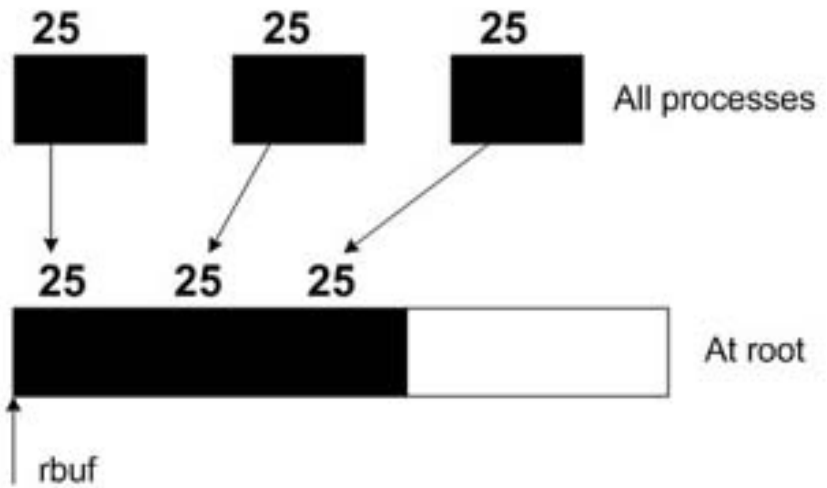
Basics

```
MPI_Init(&argc, &argv);
MPI_Finalize();
MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
MPI_Comm_size (MPI_COMM_WORLD, &num_procs);
```

Broadcast

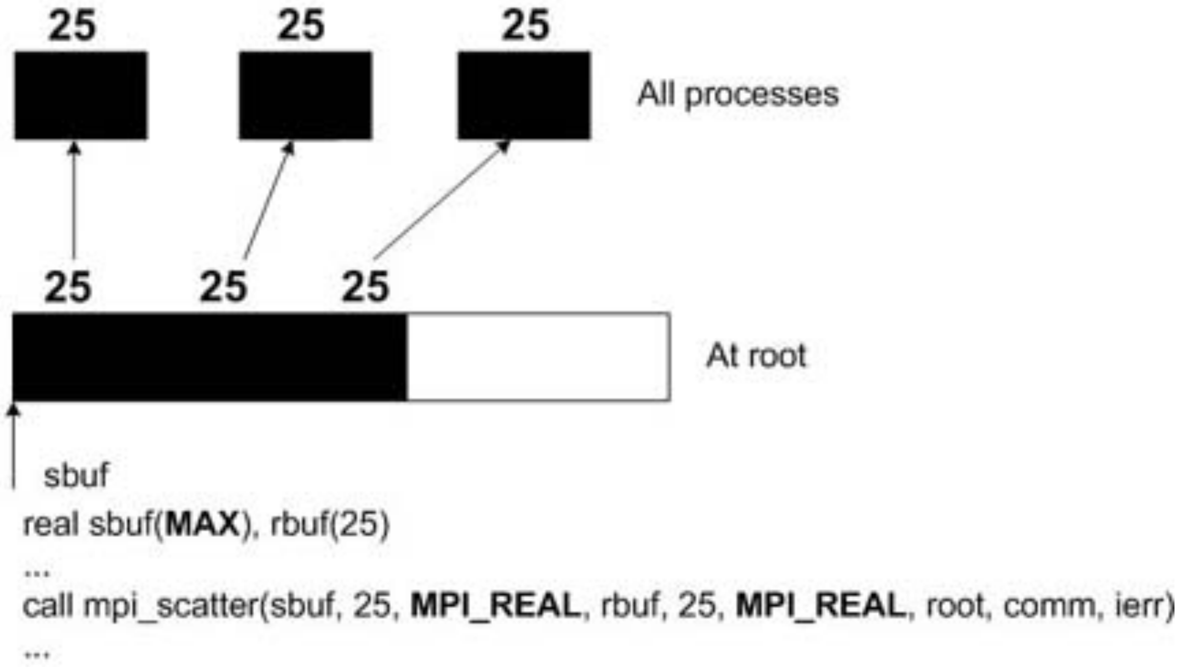
```
MPI_Bcast (&message, sizeof(message),
           MPI_INT,   sender_rank,
           MPI_COMM_WORLD);
```

MPI_GATHER



```
real a(25), rbuf(MAX)
...
call mpi_gather(a, 25, MPI_REAL, rbuf, 25, MPI_REAL, root, comm, ierr)
...
```

MPI_SCATTER



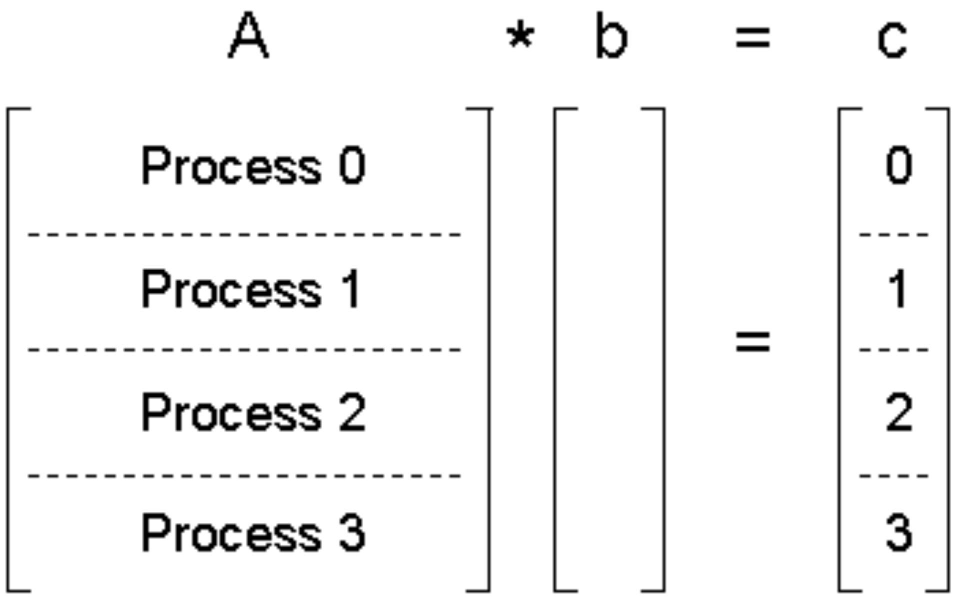
```
real sbuf(MAX), rbuf(25)
...
call mpi_scatter(sbuf, 25, MPI_REAL, rbuf, 25, MPI_REAL, root, comm, ierr)
...
```

4. Matrix Multiply

(40mins)

C[N] = A[N][N] x B[N]

What is your decomposition strategy?
What is your aggregation strategy?



A * b = c

Process 0

Process 1

Process 2

Process 3

=

0

1

2

3

- Matrix is distributed by rows (i.e. row-major order)
- Vector is needed by all
- MPI_Gather will be used to collect the product

Solutions

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv)
{
int rank, size;
MPI_Init(&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  printf(”I am %d / %d\n”, rank, size);
MPI_Finalize();
}
```

Hello World (lab1.c)

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv)
{
int rank, size;
MPI_Init(&argc, &argv);
MPI_Comm_rank (MPI_COMM_WORLD, &rank);
MPI_Comm_size (MPI_COMM_WORLD, &size);
int send_msg = 27;
int recv_msg;
int dest;
MPI_Status status;
        if(rank == size-1)
                dest = 0;
        else
                dest = rank+1;

        MPI_Send(&send_msg, sizeof(send_msg), MPI_INT, dest, 22, MPI_COMM_WORLD);
        MPI_Recv(&recv_msg, 20, MPI_INT, MPI_ANY_SOURCE, 22, MPI_COMM_WORLD, &status);

        printf("%d got %d from %d\n",rank, recv_msg, status.MPI_SOURCE);
        fflush(stdout);
MPI_Finalize();
}
```

Ring Network (lab3.c)

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv)
{
  int message;
  int  rank, size;
  MPI_Status status;
  int sender= 0;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD,  &size);
  MPI_Comm_rank(MPI_COMM_WORLD,  &rank);

  if (rank == sender)
  {
     message = 20;
  }
  MPI_Bcast(&message, sizeof(message), MPI_INT, sender, MPI_COMM_WORLD);
  if (rank != sender)
     printf( "Message from process %d : %d\n", rank, message);
  MPI_Finalize();
}
```

3. Broadcast (lab5.c)