MPI – I

Jan 11, 2019

Assignments

- Create private project named ‘CS633-2018-19-2’ on git.cse.iitk.ac.in and add pmalakar and TAs
- Submissions
  - Individual
  - Subdirectories for each assignment (AssignmentN)
  - Use git.cse.iitk.ac.in as a *real* git repo for your code
  - Compulsory files
    - Readme (steps, issues, etc.)
    - Job script
    - Source code
    - Plot

Grading of Assignments

- Bonus
  - Neat coding style
  - Fully automated execution
  - Documentation
- Graded out of 100
- A total of 4 extra days may be taken
- Credit for early submissions (+5)
- Score reduction for late submissions (-10 per day)

Resources for MPI

- Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker and Jack Dongarra, MPI - The Complete Reference, Second Edition, Volume 1, The MPI Core.
- William Gropp, Ewing Lusk, Anthony Skjellum, Using MPI : portable parallel programming with the message-passing interface, 3rd Ed., Cambridge MIT Press, 2014.
- https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

Compute nodes for today

- CSE login names starting with A – K:
  - 172.27.19.10 – 16
- CSE login names starting with L – P:
  - 172.27.19.17 – 23
- CSE login names starting with Q – Z:
  - 172.27.19.24 – 30

Running Jobs on a Cluster

- Passwordless ssh
  - ssh-keygen
- Host fingerprint
  - for node in `seq 1 30` ; do ssh 172.27.19.$node uptime ; done

Environment Setup

- vim machinefile
  - <enter IP addresses>
- for node in `seq x y` ; do ssh 172.27.19.$node uptime ; done
  - Remove the unreachable nodes from your machinefile
- which mpiexec
  - If empty string is returned, add <mpich-install>/bin to PATH environment variable
- Download code
  - http://web.cse.iitk.ac.in/users/pmalakar/cs633/2019/code/jan11.tar.gz

Recap

- Process is identified by rank
- Communicator specifies communication context

Message Envelope

Source

Destination

Communicator

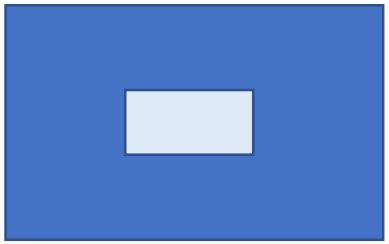
Tag (0:MPI_TAG_UB)

MPI_Datatype

- MPI_BYTE
- MPI_CHAR
- MPI_INT
- MPI_FLOAT
- MPI_DOUBLE

Point-to-point Communication

- MPI_Send



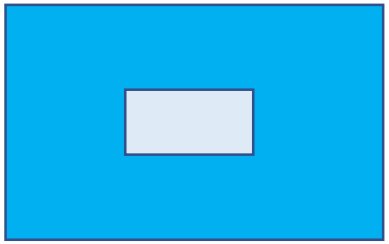
SENDER

Blocking send and receive

int MPI_Send (const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Tags should match

- MPI_Recv



RECEIVER

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

Example 1

```
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
if (myrank == 0)    /* code for process 0 */
{
   strcpy(message,"Hello, there");
   MPI_Send(message, strlen(message)+1, MPI_CHAR, 1, 99,
MPI_COMM_WORLD);
}
else if (myrank == 1)  /* code for process 1 */
{
   MPI_Recv(message, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD,
&status);
   printf("received :%s\n", message);
}
```

Example 1

- eg1 directory
  - sendrecv_simple.c
- Compile
  - make
- Execute
  - symlink your machinefile or specify path
  - which mpiexec
  - mpiexec –n 2 –f <machinefile> ./sendrecv_simple.x
- Output
  - received :Hello, there

Example 2

- eg2 directory
  - sendrecv_procname.c
- Output
  - Rank 1 of 2 received :Hello, there

MPI_Status

- Source rank
- Message tag
- Message length
  - MPI_Get_count

Example 3

- eg3 directory
  - status.c
- Output
  - Rank 1 of 4 received 20 elements

MPI_ANY_*

- MPI_ANY_SOURCE
  - Receiver may specify wildcard value for source
- MPI_ANY_TAG
  - Receiver may specify wildcard value for tag

Example 4

a send operation must specify a unique receiver

```
if (myrank == 0 || myrank == 2)
   MPI_Send(arr, 20, MPI_INT, 1, 99, MPI_COMM_WORLD);
else if (myrank == 1)
{
   int count, recvarr[3][20];
   MPI_Recv(recvarr[0], 20, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
   printf("Rank %d of %d received from rank %d\n", myrank, size,
status.MPI_SOURCE);
   MPI_Recv(recvarr[2], 20, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
   printf("Rank %d of %d received from rank %d\n", myrank, size,
status.MPI_SOURCE);
 }
```

Example 4

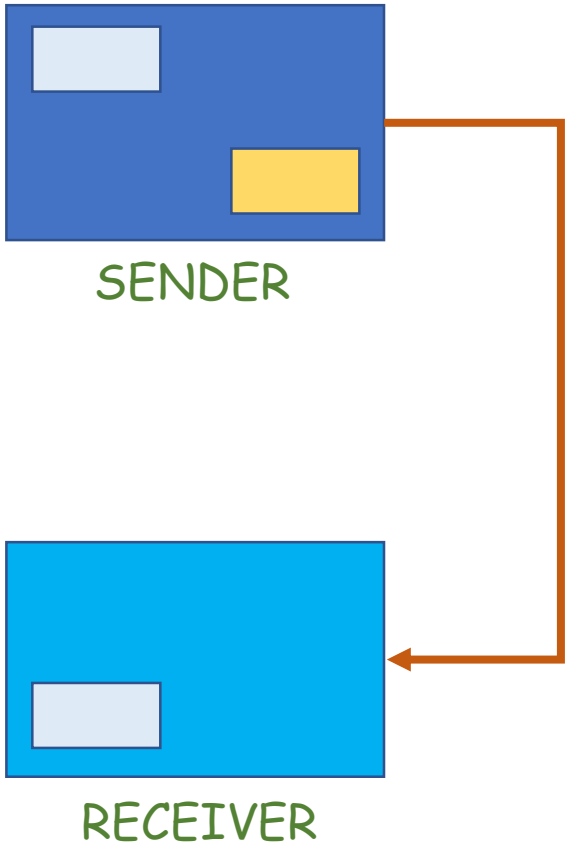
- eg4 directory
  - mpi_any.c
- Output
  - Rank 1 of 4 received from rank 0
  - Rank 1 of 4 received from rank 2
  - Rank 1 of 4 received from rank 3
- Blocked?
- No specific order

Example 5

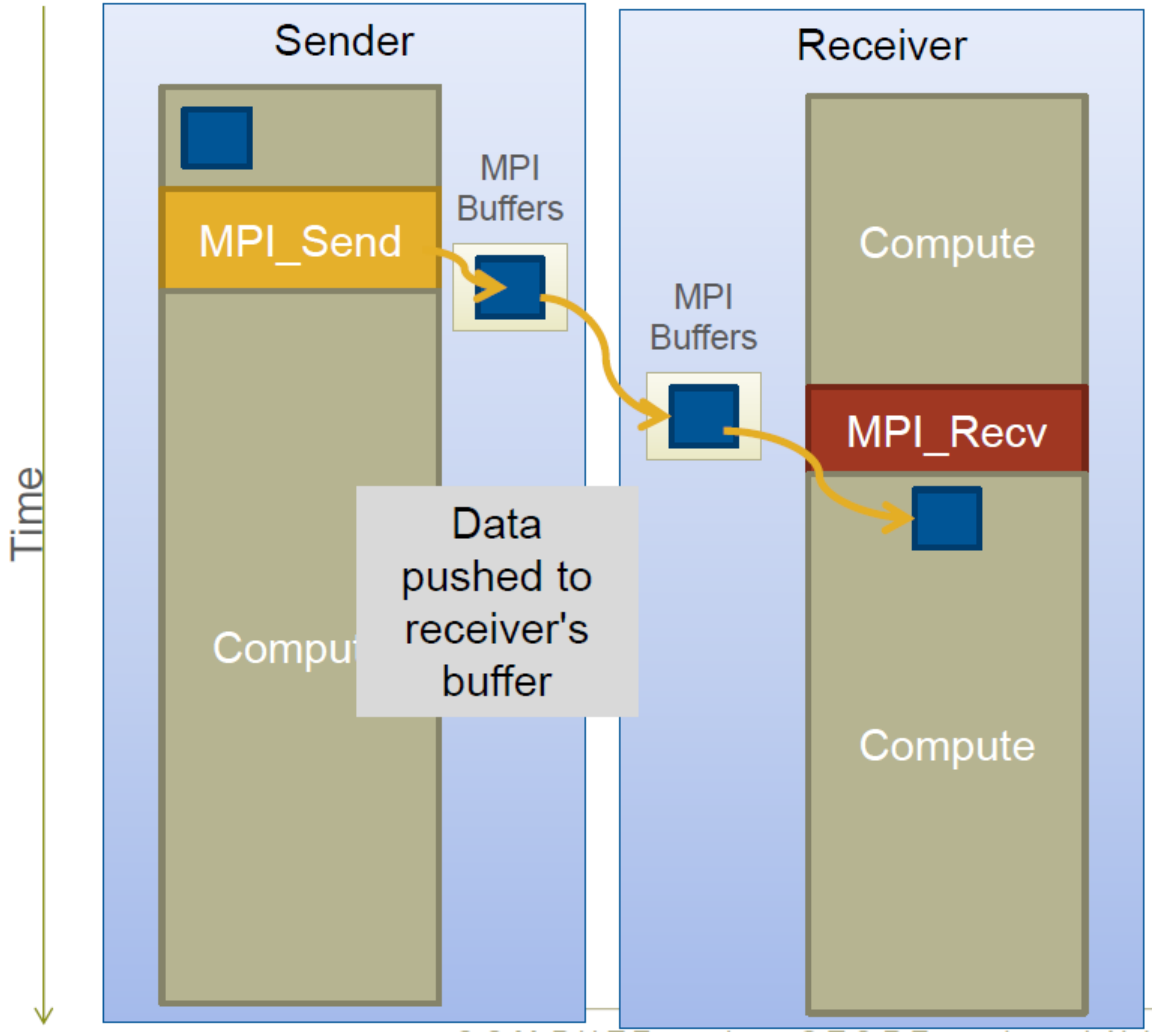
- Timing different buffer sizes
- Simple profiling
  - MPI_Wtime()
- eg5 directory
- printf ("Rank %d: time=%lf\n", myrank, MPI_Wtime () - start_time);

MPI_Send (Blocking)

- Does not return until buffer can be reused
- Message buffering
- Implementation-dependent
- Standard communication mode



Buffering



[Source: Cray presentation]

Safety

0	1	
MPI_Send	MPI_Recv	Safe
MPI_Send	MPI_Recv	
MPI_Send	MPI_Send	Unsafe
MPI_Recv	MPI_Recv	
MPI_Send	MPI_Recv	Safe
MPI_Recv	MPI_Send	
MPI_Recv	MPI_Recv	Unsafe
MPI_Send	MPI_Send	

Eager vs. Rendezvous Protocol

- Eager
  - Send completes without acknowledgement from destination
  - MPIR_CVAR_CH3_EAGER_MAX_MSG_SIZE
  - Small messages - typically 128 KB (at least in MPICH)
- Rendezvous
  - Requires an acknowledgement from a matching receive
  - Large messages

Other Send Modes

- MPI_Bsend Buffered
  - May complete before matching receive is posted
- MPI_Ssend Synchronous
  - Completes only if a matching receive is posted
- MPI_Rsend Ready
  - Started only if a matching receive is posted

Summary of Communication Modes

Mode	**Start**	**Completion**
Standard (MPI_Send)	Before or after recv	Before recv (buffer) or after (no buffer)
Buffered (MPI_Bsend)	Before or after recv	Before recv
Synchronous (MPI_Ssend)	Before or after recv	Particular point in recv
Ready (MPI_Rsend)	After recv	After recv

[From Sathish Vadhiyar's slides]

Non-blocking Point-to-Point

- MPI_Isend (buf, count, datatype, dest, tag, comm, request)
- MPI_Irecv (buf, count, datatype, source, tag, comm, request)
- MPI_Wait (request, status)

MPI_Isend
MPI_Recv

MPI_Isend
MPI_Recv

Safe

Computation Communication Overlap