MPI

Message Passing Interface

Jing Liu

TDB & LMB, Uppsala University

"Programming of Parallel Computers, Jan 2016

What to Know...

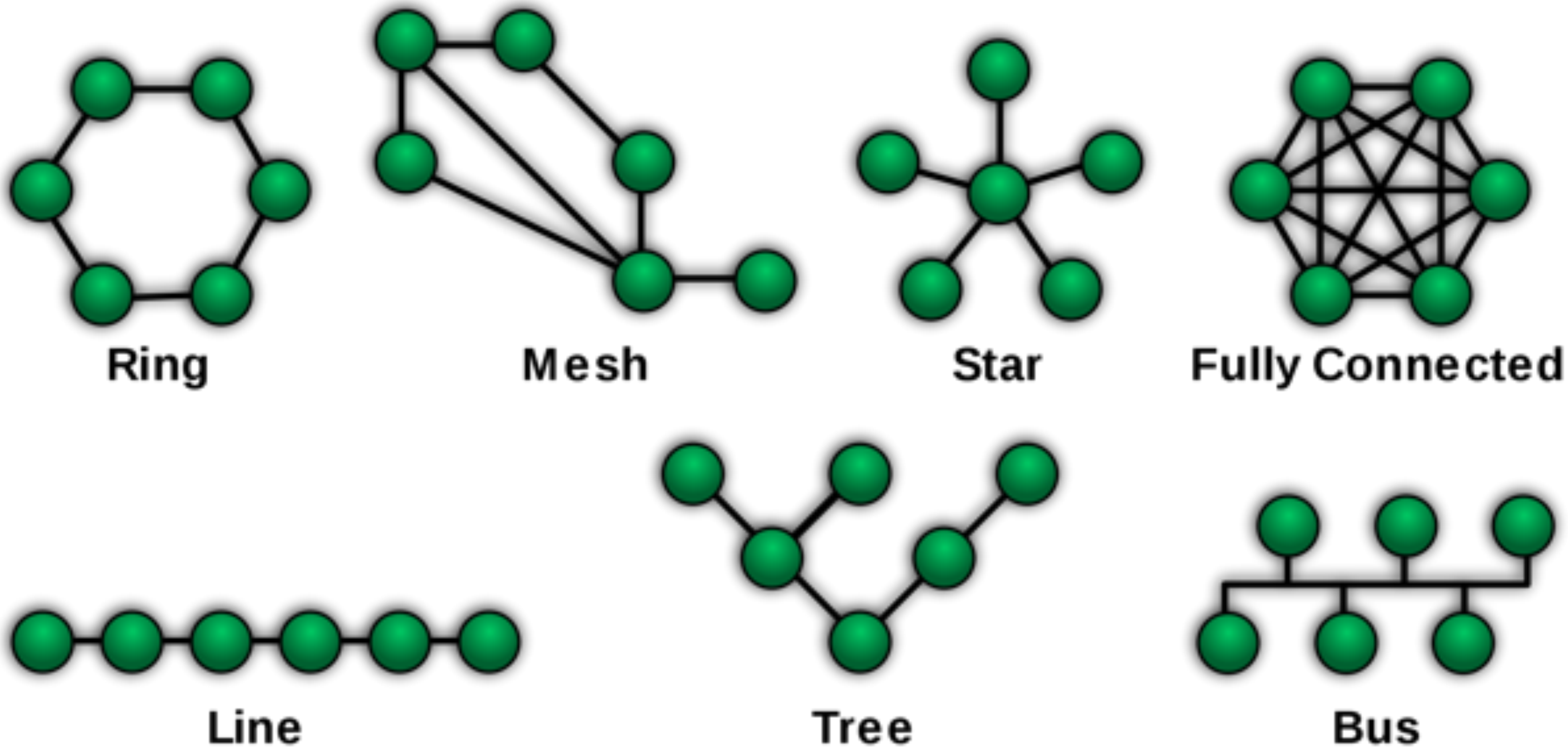
- Preliminary knowledge
  - Programming in FORTRAN/C/C++/Java
  - Basics in hardware - CPU, RAM, Network
- Outcomes:
  - MPI: ready to go
  - Note that APIs in the lecture note are for Linux/ Unix systems. They may differ in Windows systems.

Outline

- Introduction and motivation
- Code Body
- Communicators
- Send and Receive
- Other Point-to-Point Functions
- Global Functions
- Datatypes
- Topology

Dep. of Information Technology

Distributed Computing



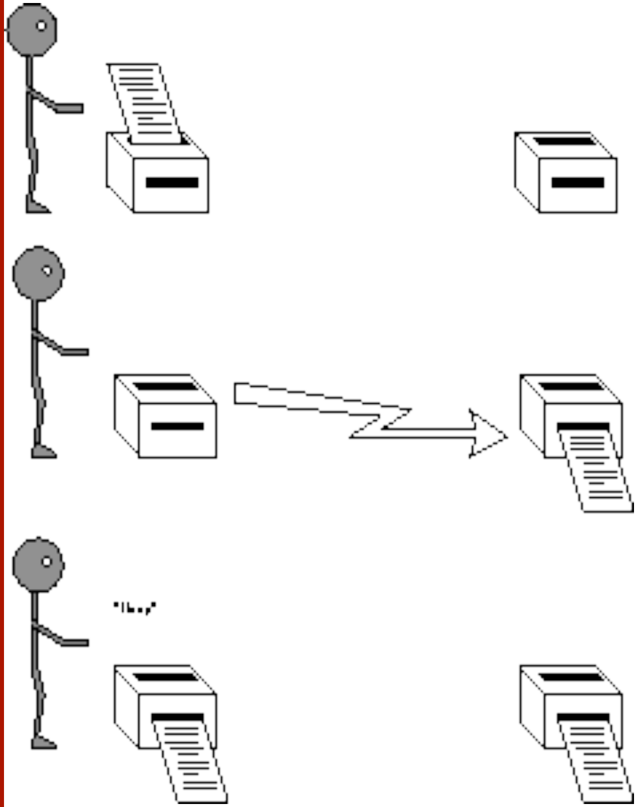
Distributed Computing -- Paradigms

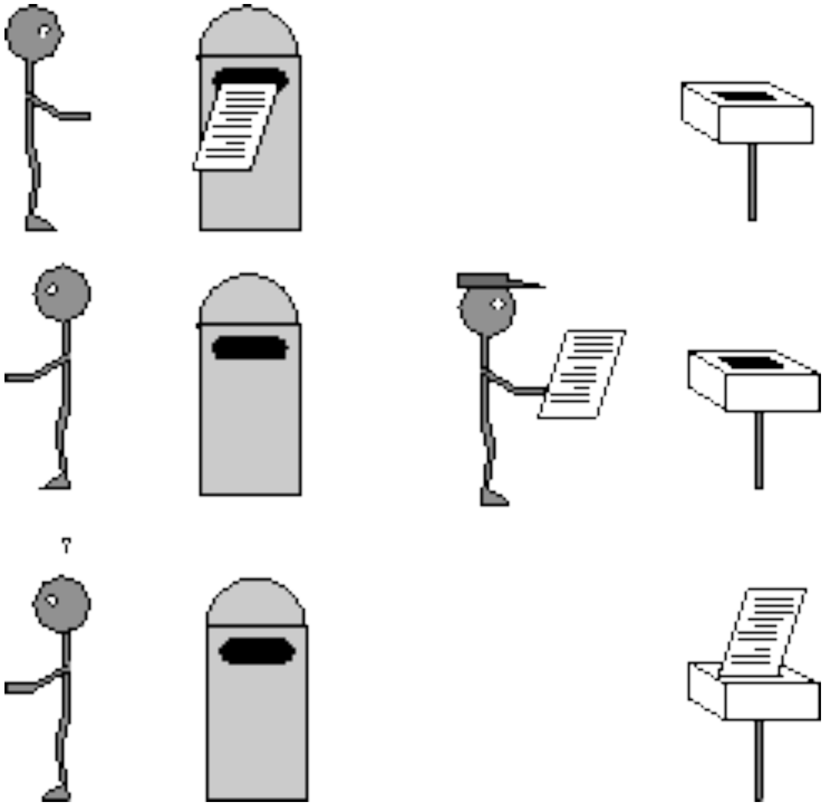
- Communication Models:
  - Message Passing
  - Shared Memory
- Computation Models:
  - Functional Parallel (task / control parallel)
  - Data Parallel

Distributed Computing

- How to communicate?
  - Sending data
  - Receiving data
  - Waiting for data
  - Waiting for synchronization

Synchronous Vs. Asynchronous





What is MPI?

- A message-passing library specifications:
  - Extended message-passing model
  - Not a language or compiler specification
  - Not a specific implementation or produce
- For parallel computers, clusters, and heterogeneous networks.
- Designed to permit the development of parallel software libraries.

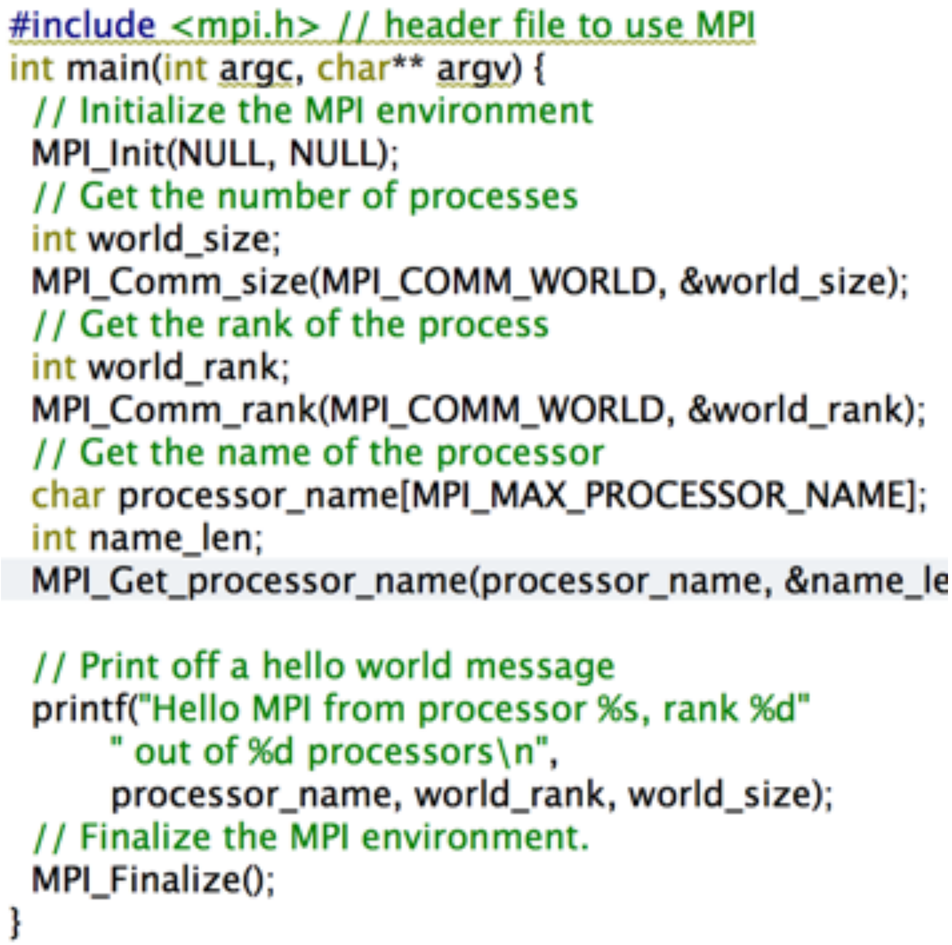
Brief History

- 1992 - draft of the project
- 1994 - first version MPI 1.0
  - Point-to-point
  - Global communication, groups
- 1997 - MPI 2.0
  - One-sided communication
  - Dynamic management
- 2008/09 - MPI 2.1 and 2.2
- 2012 - MPI-3.0
  - New one-sided communication

Outline

- Introduction and motivation
- Code Body
- Communicators
- Send and Receive
- Other Point-to-Point Functions
- Global Functions
- Datatypes
- Topology

Hello MPI



```
#include <mpi.h> // header file to use MPI
int main(int argc, char** argv) {
  // Initialize the MPI environment
  MPI_Init(NULL, NULL);
  // Get the number of processes
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
  // Get the rank of the process
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  // Get the name of the processor
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  int name_len;
  MPI_Get_processor_name(processor_name, &name_le

  // Print off a hello world message
  printf("Hello MPI from processor %s, rank %d"
         " out of %d processors\n",
         processor_name, world_rank, world_size);
  // Finalize the MPI environment.
  MPI_Finalize();
}
```

A MPI program

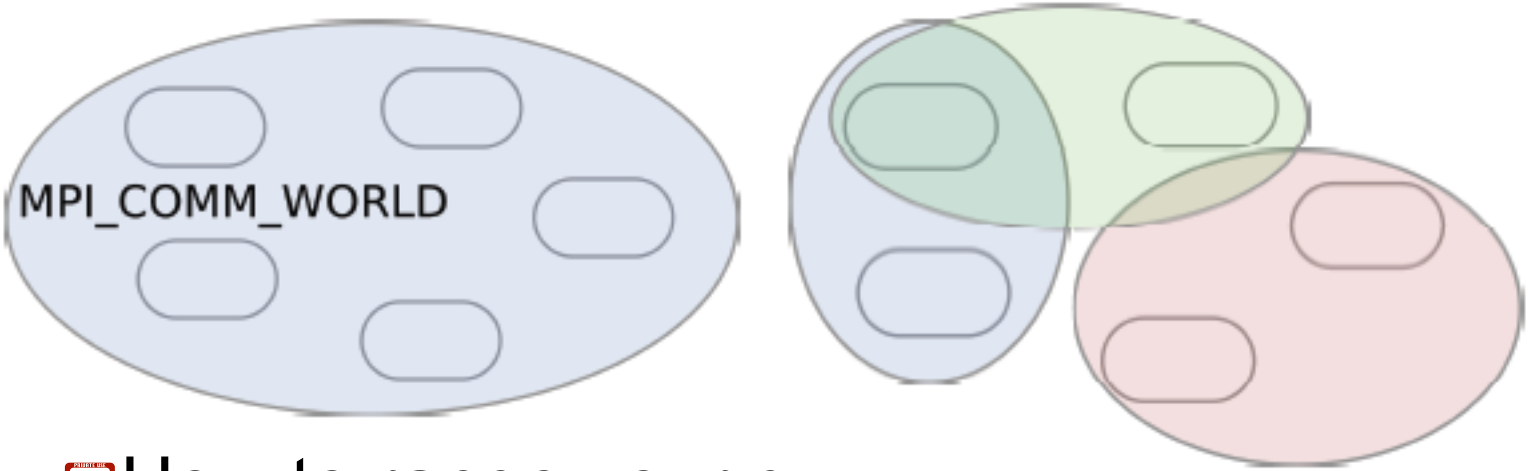
- Code body MUST have:
  - Header file: #include <mpi.h>
  - MPI Init(&agrc, &argv);
  - MPI Finalize();
- Compilation
  - Mpicc -o program program.c
  - mpiCC -o program program.cpp
- Execution
  - mpirun -np N ./program

Outline

- Introduction and motivation
- Code Body
- Communicators
- Send and Receive
- Other Point-to-Point Functions
- Global Functions
- Datatypes
- Topology

Communicator

- Groups and communicators are two important concepts

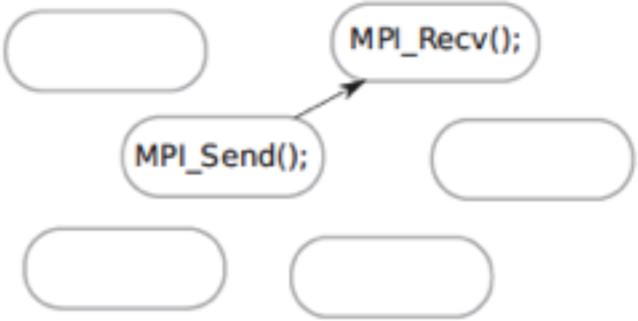


- How to range your datasets / communication patterns/ algorithm/ …

Outline

- Introduction and motivation
- Code Body
- Communicators
- Send and Receive
- Other Point-to-Point Functions
- Global Functions
- Datatypes
- Topology

Send and Receive

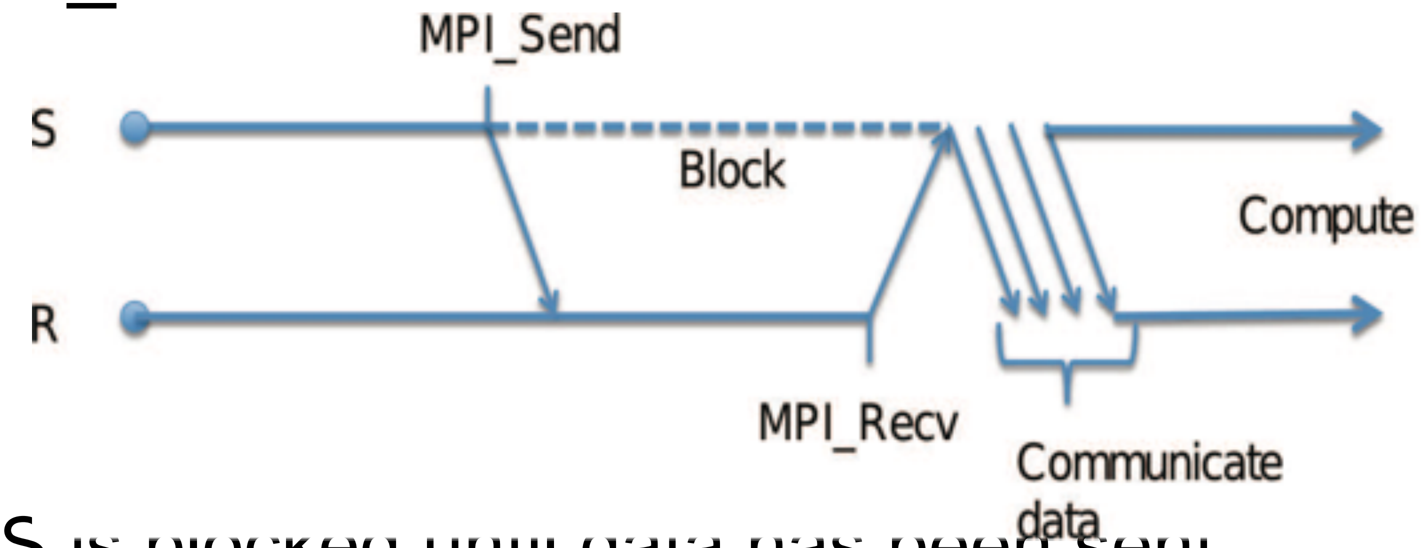


- Point to point communication
- 4 basic communication modes
  - Standard: MPI_Send / MPI_Isend
  - Synchronous: MPI_Ssend / MPI_Issend
  - Ready: MPI_Rsend / MPI_Irsend
  - Buffered: MPI_Bsend / MPI_Ibsend
- Blocking vs. Non-Blocking

Read more about communication mode in section 3.4 at MPI document or link

Send and Receive (cont.)

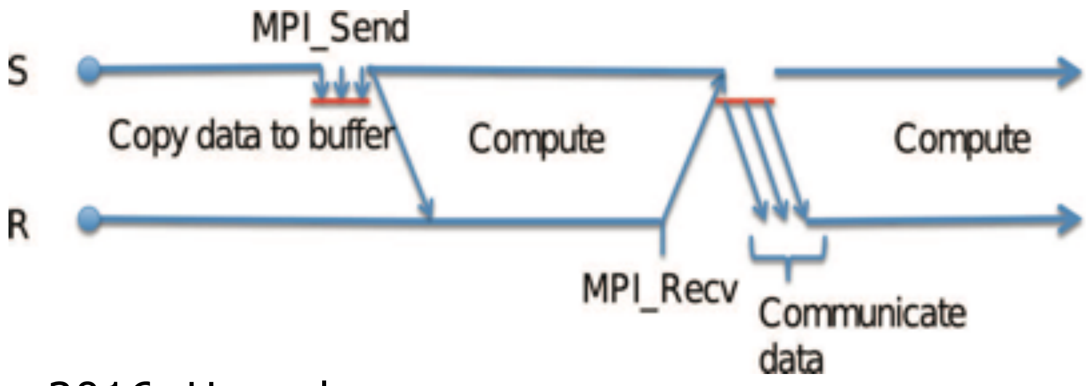
- MPI_Send – standard



  - S is blocked until data has been sent
  - MPI_Recv can be post before MPI_Send

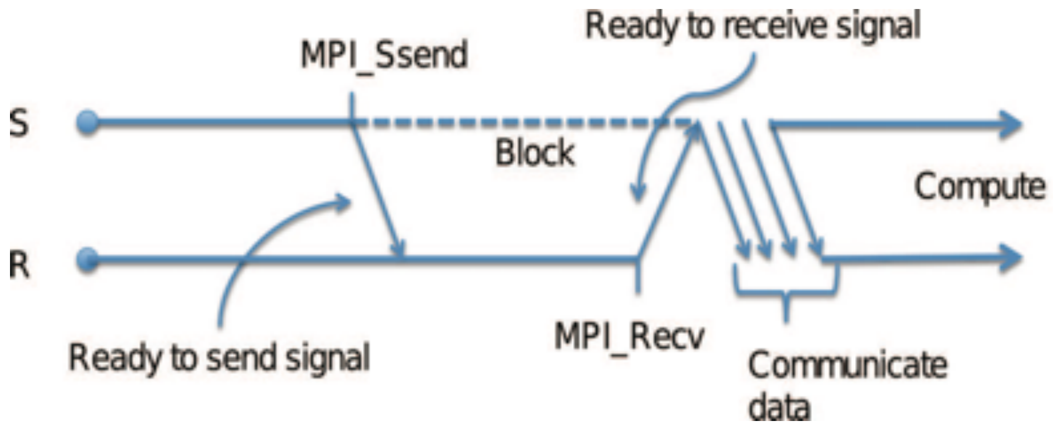
Send and Receive (cont.)

- MPI_Send with a very small dataset
  - An eager protocol may be used.
  - S assume that R can store a small message
  - R has the responsibility to buffer the message upon its arrival, especially if the receive operation has not been posted.



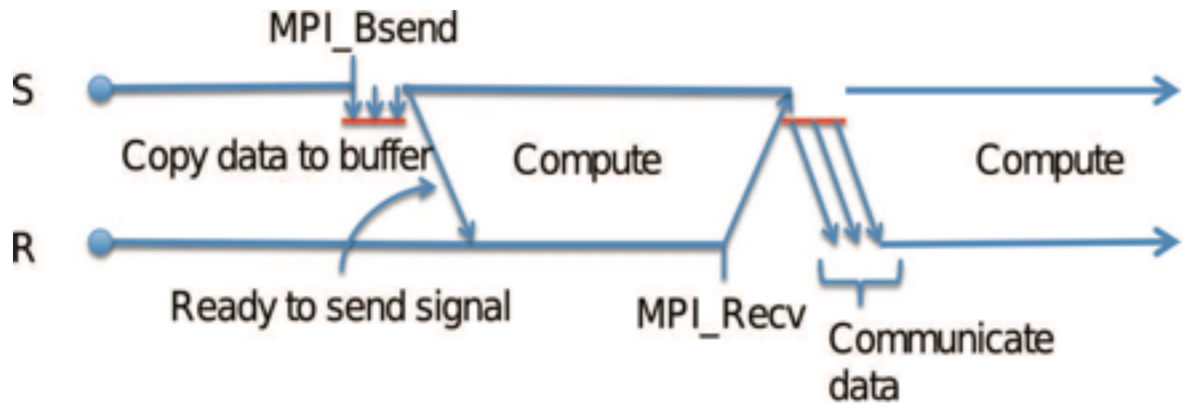
Send and Receive (cont.)

- MPI_Ssend
  - Synchronous
  - S waits until the receive has been posted on the receiving end.



Send and Receive (cont.)

- Buffered MPI_Bsend
  - S returns after coping data to a user-supplied communication buffer.
  - Safe to modify the original data.
  - S blocks also when data is transferring.



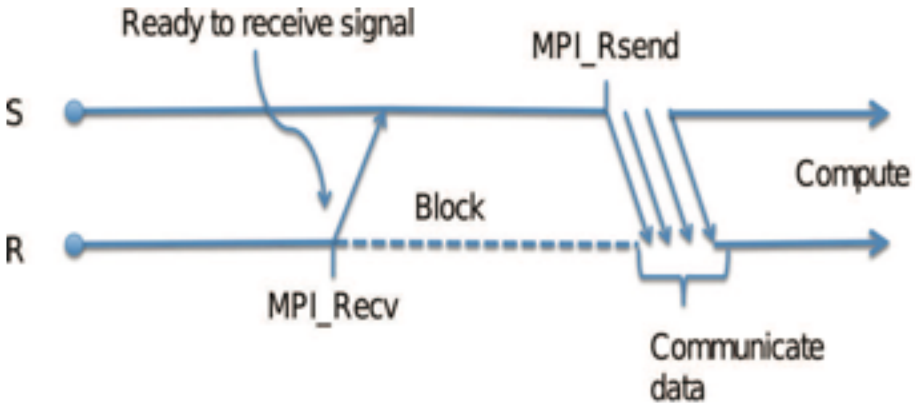
Send and Receive (cont.)

- MPI_Bsend cont.
    - Explicitly allocate the buffer first.
        - int buflen = totlen*sizeof(double) + MPI_BSEND_OVERHEAD;
        - double* buffer = malloc(buflen);
        - MPI_Buffer_attach(buffer, buflen);
        - MPI_Bsend(data,count,type,dest,tag,comm);
    - Explicitly use wait/test to ensure that the communication has completed and it is safe to modify the data.

Send and Receive (cont.)

- MPI_Rsend
  - Ready mode
  - It notifies the system that a matching receive is already posted. That information can save some overhead.
  - Replace

  MPI_Send()

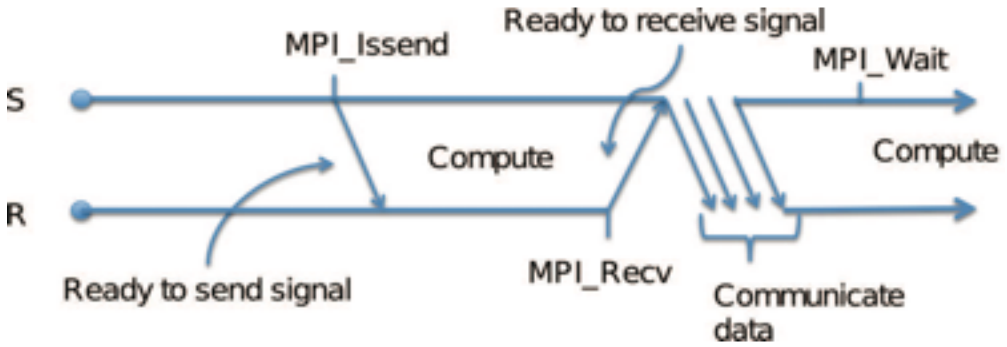


Send and Receive (cont.)—blocking message send

Standard (MPI_Send)	The sending process returns when the system can buffer the message or when the message is received and the buffer is ready for reuse.
Buffered (MPI_Bsend)	The sending process returns when the message is buffered in an application-supplied buffer.
Synchronous (MPI_Ssend)	The sending process returns only if a matching receive is posted and the receiving process has started to receive the message.
Ready (MPI_Rsend)	The message is sent as soon as possible.

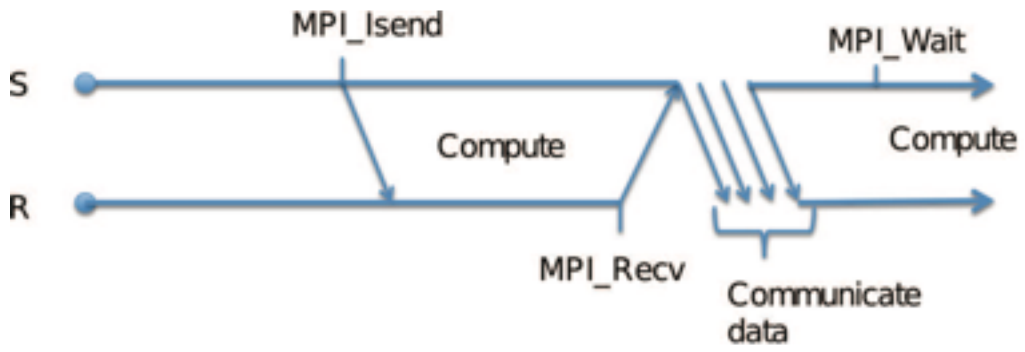
Send and Receive (cont.) – Non-blocking

- MPI_Issend
  - Initiates a synchronous mode send
  - S is not blocked
  - Explicitly use wait/test to ensure that the data buffer is safe to use.



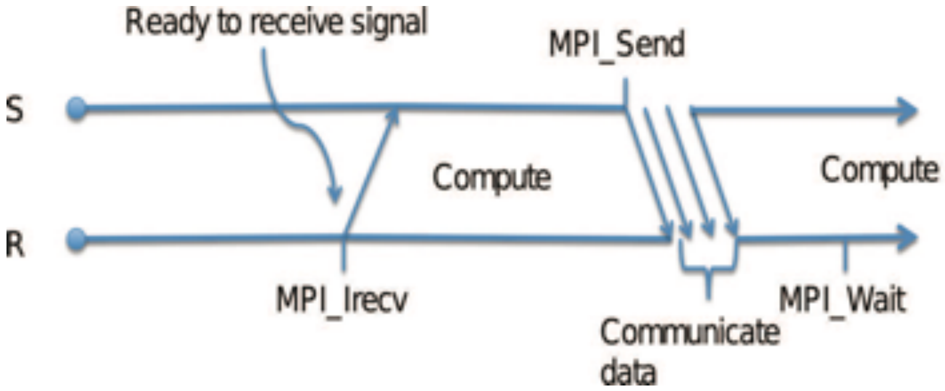
Send and Receive (cont.) – Non-blocking

- MPI_Isend
  - Similar to MPI_Issend
  - May return before data is copied out of buffer
  - Explicitly use wait/test to ensure that the data buffer is safe to use.



Send and Receive (cont.) – Non-blocking

- Non-blocking MPI_Irecv
  - May return before the message is received into the buffer.
  - Explicitly use wait/test to ensure data is safe to use
  - Non blocking



Send and Receive (cont.)

- All send calls need to be matched with a receive call, otherwise deadlock.
- Blocking calls suspend the execution until the message (data) buffer is safe to use (been sent/ received/copied).
- Non-blocking calls return immediately after initiating the communication, use test/ wait to make sure memory can be used again.

Send and Receive (cont.)

- Tips:
  - Use non-blocking operations if possible, for performance.
  - Post non-blocking operations as early as possible, so that communications overlap with computations.
  - In most cases, the standard non-blocking operations are sufficient.

Find APIs here

Outline

- Introduction and motivation
- Code Body
- Communicators
- Send and Receive
- **Other Point-to-Point Functions**
- Global Functions
- Datatypes
- Topology

Other P2P Functions

- MPI_Wait / MPI_Waitall / MPI_Waitany / MPI_Waitsome
  - Wait for (a specified / all / any/ some specified) request(s) to complete.
- MPI_Test / MPI_Testall / MPI_Testany / MPI_Testsome
  - Test the completion of (a specified / all / any/ some specified) request(s)
  - It returns immediately

Other P2P operations

- MPI_Probe, MPI_Iprobe
  - Allow checking of incoming messages without actual receipt of them
  - Can be used for allocating receive buffer dynamically.
- MPI_Cancel
  - Cancel a pending, non-blocking communication
  - User resources need to be freed

Quizzzzzzzzz~~~~~~~

1. What is the name of the global communicator? MPI_COMM_WORLD? MPI_GLOBAL_WORLD?
2. Dose MPI_Send perform better than MPI_Rsend?
3. Advantages of non-blocking MPI send/ receive calls?