

COMS3008A: Parallel Computing

Introduction to MPI I

Hairong Wang

School of Computer Science & Applied Mathematics
University of the Witwatersrand, Johannesburg

May 4, 2025

Contents

- 1 Message Passing Model and MPI
 - Message Passing Model
 - MPI
 - The Minimal Set of MPI Functions
 - Compiling and running MPI Programs
 - Installing MPI standard implementation
- 2 Data Communication
 - MPI Blocking Send and Receive
 - MPI Datatypes
 - Benchmarking the Performance
- 3 Examples
- 4 Collective Communication and Computation Operations
- 5 Running MPI Program With MPICH
- 6 Summary

Expected Learning Outcome

- Understanding message passing programming model
- Apply the basic set of MPI functions to write simple MPI programs
- Compile and run MPI program on a cluster

Outline

1 Message Passing Model and MPI

- Message Passing Model
- MPI
- The Minimal Set of MPI Functions
- Compiling and running MPI Programs
- Installing MPI standard implementation

2 Data Communication

- MPI Blocking Send and Receive
- MPI Datatypes
- Benchmarking the Performance

3 Examples

4 Collective Communication and Computation Operations

5 Running MPI Program With MPICH

6 Summary

Outline

1 Message Passing Model and MPI

- Message Passing Model
- MPI
- The Minimal Set of MPI Functions
- Compiling and running MPI Programs
- Installing MPI standard implementation

2 Data Communication

- MPI Blocking Send and Receive
- MPI Datatypes
- Benchmarking the Performance

3 Examples

4 Collective Communication and Computation Operations

5 Running MPI Program With MPICH

6 Summary

Message Passing Model

- The usual underlying hardware: a collection of processors, each with its own local memory (or address space).
- A process: a task
 - A process is (traditionally) a program counter and address space.
 - Processes may have multiple threads (program counters and associated stacks) sharing a single address space.
- Every process can communicate with every other processes.

1 Message Passing Model and MPI

- Message Passing Model
- **MPI**
- The Minimal Set of MPI Functions
- Compiling and running MPI Programs
- Installing MPI standard implementation

2 Data Communication

- MPI Blocking Send and Receive
- MPI Datatypes
- Benchmarking the Performance

3 Examples

4 Collective Communication and Computation Operations

5 Running MPI Program With MPICH

6 Summary

- MPI (Message Passing Interface) standard is the most popular message passing specification that supports parallel programming.
- MPI is a message passing library specification.
- MPI is for communication among processes, which have separate address spaces.
- Inter-process communication consists of
 - synchronization
 - movement of data from one process' address space to another's.

The Message Passing Model and MPI cont.

- MPI is not
 - a language or compiler specification
 - a specific implementation or product
- MPI is for parallel computers, clusters, and heterogeneous networks.
- MPI versions:
 - MPI-1 supports the classical message-passing programming model: basic point-to-point communication, collectives, datatypes, etc. MPI-1 was defined (1994) by a broadly based group of parallel computer vendors, computer scientists, and applications developers.
 - MPI-2 was released in 1997. MPI-2.1 (2008) and MPI-2.2 (2009) with some corrections to the standard and small features
 - MPI-3 (2012) added several new features to MPI.
 - MPI-4 (2021) added several new features to MPI.
 - MPI-5 (in proposal stage).
 - The Standard itself: at <http://www.mpi-forum.org>. All MPI official releases, in both postscript and HTML.

1 Message Passing Model and MPI

- Message Passing Model
- MPI
- **The Minimal Set of MPI Functions**
- Compiling and running MPI Programs
- Installing MPI standard implementation

2 Data Communication

- MPI Blocking Send and Receive
- MPI Datatypes
- Benchmarking the Performance

3 Examples

4 Collective Communication and Computation Operations

5 Running MPI Program With MPICH

6 Summary

The Minimal set of MPI Functions

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

Starting and Terminating MPI Programs

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

The parameters are from the parameters of the main C program.

- Initialization: must call this prior to other MPI functions
- `int MPI_Finalize()`
 - Must call at the end of MPI program
- Return codes

`MPI_SUCCESS`

`MPI_ERROR`

- `MPI_Comm`: **communicator** – communication domain
 - Group of processes that can communicate with one another
 - Supplied as an argument to all MPI message passing functions
 - Process can belong to multiple communication domain
- `MPI_COMM_WORLD`: **root communicator** – includes all the processes

Communicator Inquiry Functions

- `int MPI_Comm_size(MPI_Comm comm, int *size)`
 - Determine the number of processes
- `int MPI_Comm_rank(MPI_Comm comm, int *rank)`
 - index of the calling process
 - value: 0 — communicator size - 1

Simple MPI Program — “Hello World”

```
1  #include <mpi.h>
2  #include <stdio.h>
3  int main(int argc, char *argv[])
4  {
5      int num_procs, myrank;
6
7      MPI_Init(&argc, &argv);
8
9      MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
10     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
11
12     printf("From process %d out of %d, Hello World!\n",
13           myrank, num_procs);
14
15     MPI_Finalize();
16     return 0;
17 }
```

Simple MPI Program cont.

- Header file for MPI program — `#include <mpi.h>`
- Each active MPI process executes its own copy of the program.
- The first MPI function call made by every MPI process is the call to `MPI_Init`, which allows the system to do any setup needed to handle further calls to the MPI library.
- When MPI has been initialized, every active process becomes a member of a communicator called `MPI_COMM_WORLD`.

Simple MPI Program cont.

- A communicator defines a *communication domain* – a set of processes that are allowed to communicate with each other – a communication context.
- A default communicator: `MPI_COMM_WORLD`, which includes all the processes started by the user.
- Communicators are used as arguments to all message transfer MPI routines.
- A process can belong to many different (possibly overlapping) communication domains.

Simple MPI Program cont.

- A process calls function `MPI_Comm_rank` to determine its rank within a communicator.
- It calls `MPI_Comm_size` to determine the total number of processes in a communicator.

```
int id, p;
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &id);
```

```
MPI_Comm_size(MPI_COMM_WORLD, &p);
```

- After a process has completed all of its MPI library calls, it calls function `MPI_Finalize`, allowing the system to free up resources (such as memory) that have been allocated to MPI, and shuts down MPI environment.

1 Message Passing Model and MPI

- Message Passing Model
- MPI
- The Minimal Set of MPI Functions
- **Compiling and running MPI Programs**
- Installing MPI standard implementation

2 Data Communication

- MPI Blocking Send and Receive
- MPI Datatypes
- Benchmarking the Performance

3 Examples

4 Collective Communication and Computation Operations

5 Running MPI Program With MPICH

6 Summary

Compiling and Running MPI Programs

- MPI is a library. Applications can be written in C, C++ or Fortran and appropriate calls to MPI can be added where required
- Compilation
 - Regular applications: `gcc test.c -o test`
 - MPI applications: `mpicc hello_world.c -o test`
- Execution
 - Regular applications: `./test`
 - MPI applications (running with 16 processes):
 - `mpiexec -n 16 ./test` or
 - `mpirun -np 16 ./test`

1 Message Passing Model and MPI

- Message Passing Model
- MPI
- The Minimal Set of MPI Functions
- Compiling and running MPI Programs
- **Installing MPI standard implementation**

2 Data Communication

- MPI Blocking Send and Receive
- MPI Datatypes
- Benchmarking the Performance

3 Examples

4 Collective Communication and Computation Operations

5 Running MPI Program With MPICH

6 Summary

Installing MPI standard implementation

MPI standard implementation can be already provided as part of the OS, often as MPICH^[1] or OpenMPI^[2]. If it is not, it can usually be installed through the provided package management, e.g., `apt` in Ubuntu:

- for MPICH,

```
sudo apt-get install libmpich-dev
```

- for OpenMPI,

```
sudo apt-get install libopenmpi-dev
```

[1] **MPICH**. *High performance portable MPI*. <https://www.mpich.org/>.

[2] **Open MPI**. *Open source high performance computing*. <https://www.open-mpi.org/>.

Outline

1 Message Passing Model and MPI

- Message Passing Model
- MPI
- The Minimal Set of MPI Functions
- Compiling and running MPI Programs
- Installing MPI standard implementation

2 Data Communication

- MPI Blocking Send and Receive
- MPI Datatypes
- Benchmarking the Performance

3 Examples

4 Collective Communication and Computation Operations

5 Running MPI Program With MPICH

6 Summary

- Data communication in MPI: One process sends a copy of data to another process (or a group of processes), and the other process receives it.
- Communication requires the following information:
 - Sender: the receiver, data count, data type, a user defined tag for the message
 - Receiver: the sender, data count, data type, the tag for the message

Outline

1 Message Passing Model and MPI

- Message Passing Model
- MPI
- The Minimal Set of MPI Functions
- Compiling and running MPI Programs
- Installing MPI standard implementation

2 Data Communication

- MPI Blocking Send and Receive
- MPI Datatypes
- Benchmarking the Performance

3 Examples

4 Collective Communication and Computation Operations

5 Running MPI Program With MPICH

6 Summary

MPI Basic (Blocking) Send

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

- The message buffer is described by `buf`, `count`, `datatype`.
- The target process is specified by `dest` and `comm`.
 - `dest` is the rank of the target process in the communicator specified by `comm`.
- `tag` is a user-defined type for the message
- When this function returns, the data has been delivered to the communication system and the send buffer (described by `buf`, `count`, `datatype`) can be reused. The message may not have been received by the target process.

MPI Basic (Blocking) Receive

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

- Waits until a matching (on source, datatype, tag, comm) message is received from the communication system, and the buffer can be used.
- Source is the rank of sender in communicator `comm`, or `MPI_ANY_SOURCE`.
- `status` contains further information: The MPI type `MPI_Status` is a struct with at least three members `MPI_SOURCE`, `MPI_TAG`, and `MPI_ERROR`.
- `MPI_STATUS_IGNORE` can be used if we don't need any additional information

Simple Communication in MPI

```
1 #include <mpi.h>
2 #include <stdio.h>
3
4 int main(int argc, char ** argv){
5     int rank, data[100];
6
7     MPI_Init(&argc, &argv);
8     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
9     //Array data needs to be initialized
10    if (rank == 0)
11        MPI_Send(data, 100, MPI_INT, 1, 0, MPI_COMM_WORLD);
12    else if (rank == 1)
13        MPI_Recv(data, 100, MPI_INT, 0, 0, MPI_COMM_WORLD,
14                MPI_STATUS_IGNORE);
15
16    MPI_Finalize();
17    return 0;
18 }
```

Outline

- 1 Message Passing Model and MPI
 - Message Passing Model
 - MPI
 - The Minimal Set of MPI Functions
 - Compiling and running MPI Programs
 - Installing MPI standard implementation
- 2 **Data Communication**
 - MPI Blocking Send and Receive
 - **MPI Datatypes**
 - Benchmarking the Performance
- 3 Examples
- 4 Collective Communication and Computation Operations
- 5 Running MPI Program With MPICH
- 6 Summary

MPI Datatypes

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

Sending and Receiving Messages

- MPI allows specification of wildcard arguments for both `source` and `tag`.
- If `source` is set to `MPI_ANY_SOURCE`, then any process of the communication domain can be the source of the message.
- If `tag` is set to `MPI_ANY_TAG`, then messages with any tag are accepted.
- Only a receiver can use wildcard arguments. Sender must use a process rank and nonnegative tag.
- On the receive side, the message must be of length equal to or less than the length field specified.

Sending and Receiving Messages

- On the receiving end, the `status` variable can be used to get information about the `MPI_Recv` operation.
- The corresponding data structure contains:

```
typedef struct MPI_Status {  
    int MPI_SOURCE;  
    int MPI_TAG;  
    int MPI_ERROR;  
};
```

- The `MPI_Get_count` function returns the source, tag and number of elements of datatype received

```
int MPI_Get_count(MPI_Status *status, /* in */  
                 MPI_Datatype datatype, /* in */  
                 int *count /* out */) {
```


Outline

1 Message Passing Model and MPI

- Message Passing Model
- MPI
- The Minimal Set of MPI Functions
- Compiling and running MPI Programs
- Installing MPI standard implementation

2 Data Communication

- MPI Blocking Send and Receive
- MPI Datatypes
- **Benchmarking the Performance**

3 Examples

4 Collective Communication and Computation Operations

5 Running MPI Program With MPICH

6 Summary

Benchmarking the Performance

- Running time:
 - MPI provides a function called `MPI_Wtime` that returns a double-precision floating-point number of seconds that have elapsed since some point of time in the past.
 - Function `MPI_Wtick` returns a floating-point number that is the time in seconds between successive ticks of the clock, i.e., the precision of the result returned by `MPI_Wtime`.
- ```
double MPI_Wtime()
double MPI_Wtick()
```

# Outline

- 1 Message Passing Model and MPI
  - Message Passing Model
  - MPI
  - The Minimal Set of MPI Functions
  - Compiling and running MPI Programs
  - Installing MPI standard implementation
- 2 Data Communication
  - MPI Blocking Send and Receive
  - MPI Datatypes
  - Benchmarking the Performance
- 3 Examples
- 4 Collective Communication and Computation Operations
- 5 Running MPI Program With MPICH
- 6 Summary

# Example 1: Rotating a token around a ring

## Example 1

Consider a set of  $n$  processes arranged in a ring. Process 0 sends a token message, say "Hello!", to process 1; process 1 passes it to process 2; process 2 to process 3, and so on. Process  $n - 1$  sends back the message to process 0. Write an MPI program that performs this simple token ring.

# Example 1 cont.

```
1 MPI_Init(&argc,&argv);
2 MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
3 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

5 if(myrank == 0)
6 prev = nprocs - 1;
7 else
8 prev = myrank - 1;
9 if(myrank == (nprocs - 1))
10 next = 0;
11 else
12 next = myrank + 1;
13 if(myrank == 0)
14 strcpy(token, "Hello!");
15 MPI_Send(token, MSG_SZ, MPI_CHAR, next, tag, MPI_COMM_WORLD);
16 MPI_Recv(token, MSG_SZ, MPI_CHAR, prev, tag, MPI_COMM_WORLD,
17 MPI_STATUS_IGNORE);
18 printf("Process %d received token %s from process %d.\n", myrank, token
19 , prev);
```

# Example 1 cont.

```
1 DATE: Wed Sep 29 16:26:19 SAST 2021
2 Job is running on nodes mscluster[30,47,50,55]
3 -----
4 SLURM: sbatch is running on mscluster0.ms.wits.ac.za
5 SLURM: job ID is 141789
6 SLURM: submit directory is /home-mscluster/hwang/PC/mpi_lab1
7 SLURM: number of nodes allocated is 4
8 SLURM: number of tasks is 8
9 SLURM: job name is PC_mpi_com
10 -----
11 Process 1 received token Hello! from process 0.
12 Process 7 received token hR???- from process 6.
13 Process 0 received token h?p???- from process 7.
14 Process 2 received token hRU7?- from process 1.
15 Process 3 received token h?^???- from process 2.
16 Process 4 received token h?? from process 3.
17 Process 6 received token h?^P?- from process 5.
18 Process 5 received token h?6???- from process 4.
```

# Example 1 cont.

```
1 if(myrank == 0) {
2 strcpy(token, "Hello World!");
3 MPI_Send(token, MSG_SZ, MPI_CHAR, next, tag, MPI_COMM_WORLD);
4 MPI_Recv(token, MSG_SZ, MPI_CHAR, prev, tag, MPI_COMM_WORLD,
5 MPI_STATUS_IGNORE);
6 printf("Process %d received token %s from process %d.\n", myrank,
7 token, prev);
8 }
9 else {
10 MPI_Recv(token, MSG_SZ, MPI_CHAR, prev, tag, MPI_COMM_WORLD,
11 MPI_STATUS_IGNORE);
12 MPI_Send(token, MSG_SZ, MPI_CHAR, next, tag, MPI_COMM_WORLD);
13 printf("Process %d received token %s from process %d.\n", myrank,
14 token, prev);
15 }
```

# Example 1 cont.

```
1 DATE: Wed Sep 29 16:26:19 SAST 2021
2 Job is running on nodes mscluster[30,47,50,55]
3 -----
4 SLURM: sbatch is running on mscluster0.ms.wits.ac.za
5 SLURM: job ID is 141789
6 SLURM: submit directory is /home-mscluster/hwang/PC/mpi_lab1
7 SLURM: number of nodes allocated is 4
8 SLURM: number of tasks is 8
9 SLURM: job name is PC_mpi_com
10 -----
11 Process 1 received token Hello World! from process 0.
12 Process 2 received token Hello World! from process 1.
13 Process 3 received token Hello World! from process 2.
14 Process 4 received token Hello World! from process 3.
15 Process 5 received token Hello World! from process 4.
16 Process 6 received token Hello World! from process 5.
17 Process 7 received token Hello World! from process 6.
18 Process 0 received token Hello World! from process 7.
```



# Example 1 cont.

```
1 -----
2 DATE: Wed Apr 23 01:56:07 PM SAST 2025
3 Job is running on nodes mscluster[8-9,11-14,16-17]
4 -----
5 SLURM: sbatch is running on mscluster-login1.ms.wits.ac.za
6 SLURM: job ID is 140902
7 SLURM: submit directory is /home-mscluster/hwang/PC/mpi_lab1
8 SLURM: number of nodes allocated is 8
9 SLURM: number of tasks is 8
10 SLURM: job name is PC_mpi_com
11 -----
12 ---- running mpi_ring1 ----
13 Process 3 received token from process 2.
14 Process 4 received token from process 3.
15 Process 1 received token Hello! from process 0.
16 Process 2 received token from process 1.
17 Process 5 received token from process 4.
18 Process 0 received token from process 7.
19 Process 6 received token from process 5.
20 Process 7 received token from process 6.
```

# Example 1 cont.

```
1 DATE: Wed Apr 23 01:56:07 PM SAST 2025
2 Job is running on nodes mscluster[8-9,11-14,16-17]
3 ---- running mpi_ring4 ----
4 Process 1 received token Hello World! from process 0 (node name:
 mscluster9.ms.wits.ac.za).
5 Process 2 received token Hello World! from process 1 (node name:
 mscluster11.ms.wits.ac.za).
6 Process 5 received token Hello World! from process 4 (node name:
 mscluster14.ms.wits.ac.za).
7 Process 6 received token Hello World! from process 5 (node name:
 mscluster16.ms.wits.ac.za).
8 Process 7 received token Hello World! from process 6 (node name:
 mscluster17.ms.wits.ac.za).
9 Process 3 received token Hello World! from process 2 (node name:
 mscluster12.ms.wits.ac.za).
10 Process 4 received token Hello World! from process 3 (node name:
 mscluster13.ms.wits.ac.za).
11 Process 0 received token Hello World! from process 7 (node name:
 mscluster8.ms.wits.ac.za).
```

## Example 2: The Sieve of Eratosthens – finds primes between 2 and $n$

### Example 2

- 1 Create a list of integers  $2, 3, 4, \dots, n$ , none of which is marked.
- 2 Set  $k$  to 2, the first unmarked number on the list.
- 3 Repeat until  $k^2 > n$ 
  - (a) Mark all multiples of  $k$  between  $k^2$  and  $n$
  - (b) Find the smallest number greater than  $k$  that is unmarked. Set  $k$  to this new value.
- 4 The unmarked numbers are primes

## Example 2 cont.

|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |

Table: Integers from 2 to 60

|    |   |    |  |    |  |    |  |    |  |    |  |    |  |    |  |    |  |    |  |
|----|---|----|--|----|--|----|--|----|--|----|--|----|--|----|--|----|--|----|--|
|    | 2 | 3  |  | 5  |  | 7  |  | 9  |  | 11 |  | 13 |  | 15 |  | 17 |  | 19 |  |
| 21 |   | 23 |  | 25 |  | 27 |  | 29 |  | 31 |  | 33 |  | 35 |  | 37 |  | 39 |  |
| 41 |   | 43 |  | 45 |  | 47 |  | 49 |  | 51 |  | 53 |  | 55 |  | 57 |  | 59 |  |

Table: Sieve = 2

|    |   |    |  |    |  |    |  |    |  |    |  |    |  |    |  |    |  |    |  |
|----|---|----|--|----|--|----|--|----|--|----|--|----|--|----|--|----|--|----|--|
|    | 2 | 3  |  | 5  |  | 7  |  |    |  | 11 |  | 13 |  |    |  | 17 |  | 19 |  |
|    |   | 23 |  | 25 |  |    |  | 29 |  | 31 |  |    |  | 35 |  | 37 |  |    |  |
| 41 |   | 43 |  |    |  | 47 |  | 49 |  |    |  | 53 |  | 55 |  |    |  | 59 |  |

Table: Sieve = 3

## Example 2 cont.

|    |   |    |  |   |  |    |  |    |  |    |  |    |  |  |  |    |  |    |  |
|----|---|----|--|---|--|----|--|----|--|----|--|----|--|--|--|----|--|----|--|
|    | 2 | 3  |  | 5 |  | 7  |  |    |  | 11 |  | 13 |  |  |  | 17 |  | 19 |  |
|    |   | 23 |  |   |  |    |  | 29 |  | 31 |  |    |  |  |  | 37 |  |    |  |
| 41 |   | 43 |  |   |  | 47 |  | 49 |  |    |  | 53 |  |  |  |    |  | 59 |  |

Table: Sieve = 5

|    |   |    |  |   |  |    |  |    |  |    |  |    |  |  |  |    |  |    |  |
|----|---|----|--|---|--|----|--|----|--|----|--|----|--|--|--|----|--|----|--|
|    | 2 | 3  |  | 5 |  | 7  |  |    |  | 11 |  | 13 |  |  |  | 17 |  | 19 |  |
|    |   | 23 |  |   |  |    |  | 29 |  | 31 |  |    |  |  |  | 37 |  |    |  |
| 41 |   | 43 |  |   |  | 47 |  |    |  |    |  | 53 |  |  |  |    |  | 59 |  |

Table: Sieve = 7

# The Sieve of Eratosthens – parallelization

- The key parallel computation is step 3.(a)
- Communications among the sub problems; Two communications are needed to perform step 3.(b): a reduction and a broadcast.
- To simplify the problem, let's eliminate the reduction by assuming  $n/p > \sqrt{n}$ , then the first process is responsible for finding the next value of  $k$ .
- Decomposing the problem: data decomposition –
  - Block decomposition: the range of array elements for each process  $i$  is  $\lfloor in/p \rfloor \sim \lfloor (i+1)n/p \rfloor - 1$ .
  - Block-cyclic decomposition

# Outline

- 1 Message Passing Model and MPI
  - Message Passing Model
  - MPI
  - The Minimal Set of MPI Functions
  - Compiling and running MPI Programs
  - Installing MPI standard implementation
- 2 Data Communication
  - MPI Blocking Send and Receive
  - MPI Datatypes
  - Benchmarking the Performance
- 3 Examples
- 4 Collective Communication and Computation Operations
- 5 Running MPI Program With MPICH
- 6 Summary

# Collective Communication and Computation Operations

- MPI provides an extensive set of functions for performing common collective communication operations.
- Each of these operations is defined over a group corresponding to the communicator.
- All processors in a communicator must call these operations.



# Collective Communication Operations

- The barrier synchronization operation is performed in MPI using:

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

The one-to-all broadcast operation is:

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

- The all-to-one reduction operation is:

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

# 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-max value-location | Data-pairs                    |
| MPI_MINLOC | min-min value-location | Data-pairs                    |

# A Simple Reduction Program

```
1 #include "mpi.h"
2 #include <stdio.h>
3 #include <math.h>
4 int main(int argc, char **argv){
5 int n, numprocs, myrank, myval, sum;
6 MPI_Init(&argc, &argv);
7 MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
8 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
9 if (myrank == 0){
10 printf("Enter the number to broadcast: (0 quits.)");
11 scanf("%d", &n);
12 }
13 MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
14 //printf("Rank %d received %d\n", myrank + 1, n);
15 myval = n * 2;
16 MPI_Reduce(&myval, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
17 if (myrank==0)
18 printf("Sum = %d \n", sum);
19 MPI_Finalize();
20 return 0;
21 }
```

# Function Prototypes

```
1 int MPI_Init(int *argc, char ***argv)
2 int MPI_Finalize()
3 int MPI_Comm_size(MPI_Comm comm, int *size)
4 int MPI_Comm_rank(MPI_Comm comm, int *rank)
5 int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype
 datatype, MPI_Op op, int root, MPI_Comm comm)
6 int MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int root,
 MPI_Comm comm)
7 double MPI_Wtime()
8 double MPI_Wtick()
```

# Outline

- 1 Message Passing Model and MPI
  - Message Passing Model
  - MPI
  - The Minimal Set of MPI Functions
  - Compiling and running MPI Programs
  - Installing MPI standard implementation
- 2 Data Communication
  - MPI Blocking Send and Receive
  - MPI Datatypes
  - Benchmarking the Performance
- 3 Examples
- 4 Collective Communication and Computation Operations
- 5 Running MPI Program With MPICH
- 6 Summary

# What is MPICH

- MPICH is a high-performance and widely portable open-source implementation of MPI
- It provides all features of MPI that have been defined so far (including MPI-1, MPI-2.0, MPI-2.1, MPI-2.2, and MPI-3.0)
- <http://www.mpich.org>
- Compiling MPI program using MPICH
  - For C programs: `mpicc test.c -o test`
  - For C++ programs: `mpicxx test.cpp -o test`
  - To link to a math library: `mpicc test.c -o test -lm`

# Running MPI programs with MPICH

- Launch 16 processes on the local node:

```
mpiexec -n 16 ./test
```

- Launch 16 processes on 4 nodes (each has 4 cores)

```
mpiexec -hosts h1:4,h2:4,h3:4,h4:4 -n 16 ./test
```

Runs the first four processes on h1, the next four on h2, etc.

(h1, h2, h3, h4 are host names separated by comma.)

```
mpiexec -hosts h1,h2,h3,h4 -n 16 ./test
```

Runs the first process on h1, the second on h2, etc., and wraps around. So, h1 will have the 1st, 5th, 9th and 13th processes.

# Outline





- 1 Message Passing Model and MPI
  - Message Passing Model
  - MPI
  - The Minimal Set of MPI Functions
  - Compiling and running MPI Programs
  - Installing MPI standard implementation
- 2 Data Communication
  - MPI Blocking Send and Receive
  - MPI Datatypes
  - Benchmarking the Performance
- 3 Examples
- 4 Collective Communication and Computation Operations
- 5 Running MPI Program With MPICH
- 6 **Summary**



## We learnt

- Basic set of MPI functions to write simple MPI programs
  - Initialize and finalize MPI
  - Inquire the basic MPI execution environment
  - Basic point to point communication: send and receive
  - Basic collective communication: broadcast and reduction
- Developing MPI programs through examples
- Compiling and running MPI programs
- Useful resources: *Introduction to Parallel Computing* and *Introduction to Parallel Computing: From Algorithms to Programming on State-of-the-Art Platforms*.

# References I

-  Grama, Ananth et al. *Introduction to Parallel Computing*. Addison Wesley, 2003.
-  MPICH. *High performance portable MPI*.  
<https://www.mpich.org/>.
-  Open MPI. *Open source high performance computing*.  
<https://www.open-mpi.org/>.
-  Trobec, Roman et al. *Introduction to Parallel Computing: From Algorithms to Programming on State-of-the-Art Platforms*. Springer, 2018.