COMS3008A: Parallel Computing
Introduction to MPI I

Hairong Wang

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

April 16, 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

WITS UNIVERSITY

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

WITS UNIVERSITY

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

WITS UNIVERSITY

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.

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

WITS UNIVERSITY

MPI

- 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.

WITS UNIVERSITY

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

WITS UNIVERSITY

The Minimal set of MPI Functions

`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.

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`

Communicators

- `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

WITS UNIVERSITY

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”

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
  int num_procs, myrank;

  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

  printf("From process %d out of %d, Hello World!\n",
    myrank, num_procs);

  MPI_Finalize();
  return 0;
}
```

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`.

WITS UNIVERSITY

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.

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

WITS UNIVERSITY

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`

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

WITS UNIVERSITY

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

WITS UNIVERSITY

Data Communication

- 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

WITS UNIVERSITY

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

.

WITS UNIVERSITY

Simple Communication in MPI

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

int main(int argc, char ** argv){
    int rank, data[100];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  //Array data needs to be initialized
    if (rank == 0)
        MPI_Send(data, 100, MPI_INT, 1, 0, MPI_COMM_WORLD);
    else if (rank == 1)
        MPI_Recv(data, 100, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

    MPI_Finalize();
    return 0;
}
```

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

WITS UNIVERSITY

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.

WITS UNIVERSITY

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.

```
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &nproces);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

if (myrank == 0)
  prev = nproces - 1;
else
  prev = myrank - 1;
if (myrank == (nproces - 1))
  next = 0;
else
  next = myrank + 1;
if (myrank == 0)
  strcpy(token, "Hello!");
MPI_Send(token, MSG_SZ, MPI_CHAR, next, tag,
   MPI_COMM_WORLD);
MPI_Recv(token, MSG_SZ, MPI_CHAR, prev, tag,
   MPI_COMM_WORLD, MPI_STATUS_IGNORE);
printf("Process %d received token %s from process %d.\n",
    myrank, token, prev);
```

Example 1 cont.

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

Example 1 cont.

```
if (myrank == 0) {
strcpy(token, "Hello World!");
  MPI_Send(token, MSG_SZ, MPI_CHAR, next, tag,
      MPI_COMM_WORLD);
  MPI_Recv(token, MSG_SZ, MPI_CHAR, prev, tag,
      MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  printf("Process %d received token %s from process %d.\n
      ", myrank, token, prev);
}
else {
  MPI_Recv(token, MSG_SZ, MPI_CHAR, prev, tag,
      MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  MPI_Send(token, MSG_SZ, MPI_CHAR, next, tag,
      MPI_COMM_WORLD);
  printf("Process %d received token %s from process %d.\n
      ", myrank, token, prev);
}
```

Example 1 cont.

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

WITS UNIVERSITY

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

Example 2

1. Create a list of integers $2, 3, 4, \ldots, n$, 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

WITS UNIVERSITY

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)
  ```

WITS UNIVERSITY

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

```c
#include "mpi.h"
#include <stdio.h>
#include <math.h>
int main(int argc, char **argv){
  int n, numprocs, myrank, myval, sum;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
  if (myrank == 0){
    printf("Enter the number to be broadcasted: (0 quits.)");
    scanf("%d", &n);
  }
  MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
  //printf("Rank %d received %d\n", myrank + 1, n);
  myval = n * 2;
  MPI_Reduce(&myval, &sum, 1, MPI_INT, MPI_SUM, 0,
      MPI_COMM_WORLD);
  if (myrank==0)
    printf("Sum = %d \n", sum);
  MPI_Finalize();
  return 0;
}
```

Function Prototypes

```c
int MPI_Init(int *argc, char ***argv)
int MPI_Finalize()
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
int MPI_Reduce(void *sendbuf,void *recvbuf,int count,
  MPI_Datatype datatype,MPI_Op op,int root,MPI_Comm comm)
int MPI_Bcast(void *buf,int count,MPI_Datatype datatype,
  int root,MPI_Comm comm)
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

WITS UNIVERSITY

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`

WITS UNIVERSITY

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

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.