

MPI

Spring Training 2025

Yoshihiro Izawa

2025/05/13

Contents

1. MPI Overview	2
2. Basic Learning of MPI	9
3. References	19

1. MPI Overview

1.1 What is MPI (Message Passing Interface)?

- **A standard API** for message passing between distributed memories in parallel computing.
- MPI assumes a **distributed-memory computing system**
- MPI can run on **shared-memory computing system**
- MPI programming model (basically) uses **SIMD**

1.2 Parallel Programming Classification

- **Multi-Process**: MPI(Message Passing Interface), HPF(High Performance Fortran)
- **Multi-Thread**: OpenMP, Pthread(POSIX Thread)

1.3 MPI Features

- **Communication Model:** Uses message passing for communication between processes.
- **Distributed Memory Support:** Each process has its own memory space, no shared memory.
- **Multi-node Capacity:** Can run across multiple nodes; abstracts network communication.
- **Standardized API:** Standardized interface in C, C++, and Fortran; highly portable.
- **Multiple Implementation:** Available implementations include OpenMPI, MPICH, and Intel MPI, etc.
- **Difficult to Debug:** Debugging is challenging due to concurrency and communication complexity.

1.4 Typical example of Usage

- Simulation on a supercomputer(Physics, Meteorology, Chemistry, etc.)
- Data processing in large-scale data analysis (e.g., genomics, astronomy).
- Machine learning training on large datasets (e.g., distributed deep learning).

1.5 Comparison between implementations

	OpenMPI	MPICH	Intel MPI
Developer	Universities, Companies	Argonne National Laboratory	Intel Corporation
Distribution	Open source	Open source	Free version included
Optimization Target	General purpose	Lightweight, stable	Optimized for Intel architecture
Performance	Medium to high	Lightweight, stable, scalable	Best performance on Intel CPUs
Main Use	Academic clusters, general HPC	Research, education	Commercial HPC, Intel clusters

1.6 Key Communication Primitives

- **System function:** MPI_Init, MPI_Finalize, MPI_Comm_size, MPI_Comm_rank
- **Point-to-point communication:** MPI_Send, MPI_Recv
- **Collective communication:** MPI_Bcast, MPI_Reduce, MPI_Alltoall
- **Synchronization:** MPI_Barrier, MPI_Wait, MPI_Test
- **Derived data types:** MPI_Type_create_struct, MPI_Type_vector
- **Non-blocking communication:** MPI_Isend, MPI_Irecv
- **Remote memory access:** MPI_Put, MPI_Get
- **Process management:** MPI_Comm_spawn, MPI_Comm_free

2. Basic Learning of MPI

2.1 Minimum MPI Program

2.1.1 Hello World (C)

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

int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);

    int num_procs;
    int my_rank;

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

    printf("Num of Proc : %d\n", num_procs);
    printf("My Rank      : %d\n", my_rank);

    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

```
mpicc mpi_hello.c -o mpi_hello
mpirun -np 4 ./mpi_hello
Num of Proc : 4
My Rank      : 3
Num of Proc : 4
My Rank      : 2
Num of Proc : 4
My Rank      : 0
Num of Proc : 4
My Rank      : 1
```

2.1 Minimum MPI Program

2.1.2 Hello World (C++)

```
#include <mpi.h>
#include <iostream>
#include <cstdlib>

int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);

    int num_procs;
    int my_rank;

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

    std::cout << "Num of Proc : " << num_procs <<
std::endl;
    std::cout << "My Rank      : " << my_rank << std::endl;

    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

```
mpic++ mpi_hello.cpp -o mpi_hello
mpirun -np 4 ./mpi_hello
Num of Proc : 4
My Rank      : 3
Num of Proc : 4
My Rank      : 1
Num of Proc : 4
My Rank      : 0
Num of Proc : 4
My Rank      : 2
```

2.1 Minimum MPI Program

2.1.3 Hello World (Fortran)

```
program hello_mpi
  use mpi
  implicit none

  integer :: ierr, rank, size

  call MPI_Init(ierr)
  call MPI_Comm_rank(MPI_COMM_WORLD, rank, ierr)
  call MPI_Comm_size(MPI_COMM_WORLD, size, ierr)

  print *, "Num of Proc:", size
  print *, "My Rank:    ", rank

  call MPI_Finalize(ierr)
end program hello_mpi
```

```
mpif90 mpi_hello.f90 -o mpi_hello
mpirun -np 4 ./mpi_hello
Num of Proc: 4
My Rank:    2
Num of Proc: 4
My Rank:    0
Num of Proc: 4
My Rank:    3
Num of Proc: 4
My Rank:    1
```

2.2 Important Terms of MPI

2.2.1 Overview

- **Process:**
 - computing unit in parallel computing in MPI.
 - process num is determined by `mpirun -np`
- **Group:**
 - a set of processes that can communicate with each other.
- **Communicator:**
 - a group of processes that can communicate with each other.
- **Rank:**
 - unique identifier for each process in MPI.
 - Ranks are assigned from 0 to `num_procs - 1`.

2.2 Important Terms of MPI

2.2.2 Communicator Image

- Each process belongs to some **group**.
- A group is associated with a **communicator**.
- Each process in a communicator has a unique **rank**.

- Example:
 - Process 0, 1, 2, 3 belong to a group.
 - Communicator: MPI_COMM_WORLD
 - Group: [P0, P1, P2, P3]
 - Rank: 0, 1, 2, 3

Communicator: MPI_COMM_WORLD
Group: [P0, P1, P2, P3]
Rank: 0 1 2 3

2.2 Important Terms of MPI

2.2.3 MPI Functions for Communicator

- **MPI_COMM_WORLD:**
 - the default communicator that includes all processes.
 - all processes first belong to this communicator.
 - becomes the default communicator for most MPI functions.
- **MPI_Comm_rank:**
 - retrieves the rank of the calling process in specified communicator.
 - usually use MPI_COMM_WORLD as the communicator.
- **MPI_Comm_size:**
 - retrieves the number of processes in the specified communicator.
 - usually use MPI_COMM_WORLD as the communicator.
- **MPI_Comm_split:**
 - creates a new communicator by splitting the existing one based on a color and key.
 - in other words, create a new communicator with the same color processes.

2.2 Important Terms of MPI

- **MPI_Comm_rank** and **MPI_Comm_size**

```
int rank;  
MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
printf("I am process %d\n", rank);
```

- **MPI_Comm_size**

```
int size;  
MPI_Comm_size(MPI_COMM_WORLD, &size);  
printf("There are %d processes\n", size);
```

2.2 Important Terms of MPI

- **MPI_Comm_split**

```
// ランクの3の剰余を基に color=0, 1, 2 に分ける
int color = rank % 3;

MPI_Comm new_comm;
MPI_Comm_split(MPI_COMM_WORLD, color, rank, &new_comm);

int new_rank, new_size;
MPI_Comm_rank(new_comm, &new_rank);
MPI_Comm_size(new_comm, &new_size);

printf("World Rank %d => Group %d, New Rank %d of %d\n",
       rank, color, new_rank, new_size);

MPI_Comm_free(&new_comm); // 新しいコミュニケータの解放
```

```
$ mpicc comm_split.c -o comm_split
$ mpirun -np 8 ./comm_split
World Rank 5 => Group 2, New Rank 1 of 2
World Rank 2 => Group 2, New Rank 0 of 2
World Rank 1 => Group 1, New Rank 0 of 3
World Rank 4 => Group 1, New Rank 1 of 3
World Rank 7 => Group 1, New Rank 2 of 3
World Rank 3 => Group 0, New Rank 1 of 3
World Rank 6 => Group 0, New Rank 2 of 3
World Rank 0 => Group 0, New Rank 0 of 3
```

2.2 Important Terms of MPI

World Rank	color(= rank % 3)	new group	new rank	new_size
0	0	{0, 3, 6}	0	3
1	1	{1, 4, 7}	0	3
2	2	{2, 5}	0	2
3	0	{0, 3, 6}	1	3
4	1	{1, 4, 7}	1	3
5	2	{2, 5}	1	2
6	0	{0, 3, 6}	2	3
7	1	{1, 4, 7}	2	3

3. References

3.1 MPI Reference

- MPI「超」入門（C 言語編） - 東京大学情報基盤センター
- 並列プログラミング入門