Introduction to MPI (Message Passing Interface)

Jian-Jin LI

0

MPI – Message-Passing Interface

- Open standard interface to write parallel programs
- Moves data from the address space of one process to that of another process
- Distributed memory paradigm
- Support distributed memory parallel machine, homogenous or heterogenous cluster, …
- Advantages
  - Practical
  - Portability
  - Flexibility
  - Efficiency

MPI is a library, we have to use it with a programming language

1

MPI – Message-Passing Interface

- History
  - Version 1.0: 1994
  - Version 2.0: 1997
    - Process creation & management
    - Collective communication
    - One-sided communication, parallel I/O
  - Version 3.0: 2012
    - Non-blocking collective communication
    - One-sided communication improvement
  - Version 3.1: 2015
    - Non-blocking collective I/O
  - Version 4.0: 2021
  - Version 4.1: 2023

2

References

- W. Gropp, E. Lusk and A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, 3rd edition, The MIT Press, 2014.
- W. Gropp, E. Lusk and R. Thakur, Using Advanced MPI: Modern Features of the Message-Passing Interface, The MIT Press, 2014.
- https://www.open-mpi.org, consulted in January 2025.
- https://www.mpich.org, consulted in January 2025.
- https://www.mpi-forum.org, the standardization forum
- http://www.idris.fr/formations/mpi, consulted in January 2025.
- B. Barney, Introduction to Parallel Computing, https://computing.llnl.gov/tutorials/parallel_comp, consulted in January 2025.

3

4

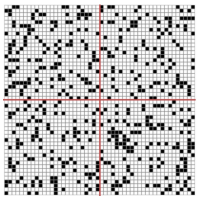
MPI - MPI Programming Model

- Group of processes to solve together a problem

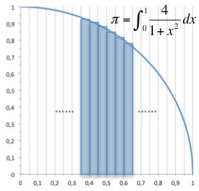
4 sub-domains

4 tasks

4 processes



Conway's Game of Life
(Source: internet)



Calculation of PI by integration

m processes

n sub-intervals on *x*

n%m==0 or n%m≠0

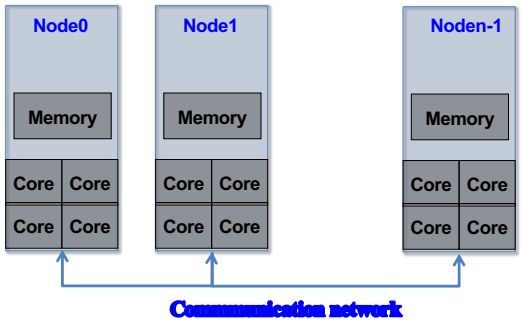
- New issues
  - Parallel tasks identification; problem decomposition
    - data parallelism
    - domain decomposition
    - functional decomposition
  - Tasks communication, synchronization, optimization
  - New bug types (deadlock, interference), starvation…

4

5

MPI - MPI Programming Model

- Distributed memory API

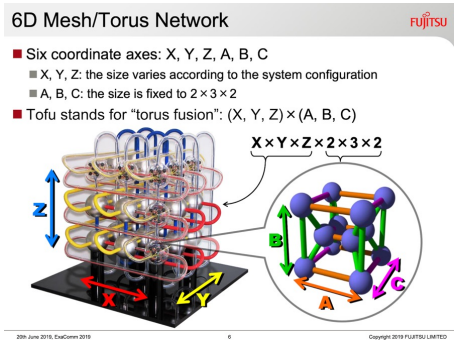


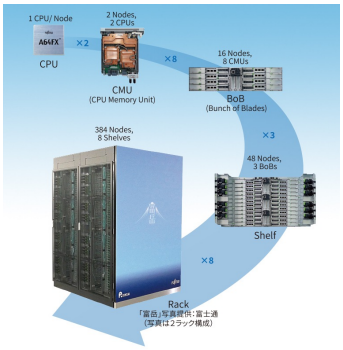
- **Distributed memory between nodes**
- **Shared memory between the cores of a node**

5

MPI Programming

- Suitable architectures – Fugaku (TOP 1, 06/2020 – 11/2021)
  - **Distributed Memory MIMD**
  - Fujitsu MPI (Based on OpenMPI), MPICH-Tofu (Based on MPICH)



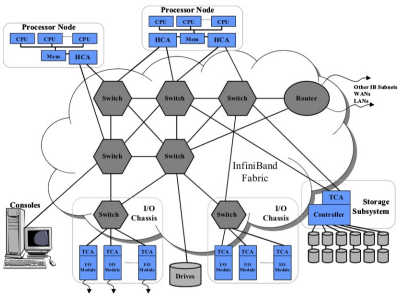


source: internet

6

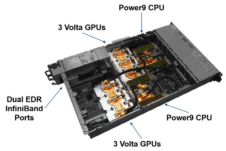
MPI Programming

- Suitable architectures – Summit (TOP 1, 06/2018 – 11/2019)
  - **Distributed Memory MIMD**



InfiniBand Fabric

256 cabinets, 18 nodes per cabinet
=> 4608 nodes



1 node:
2 IBM Power9 CPU (22 cores)
6 NVIDIA V100 GPU (640 cores)

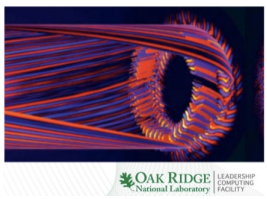
Source: ORNL – Mellanox Technologies

7

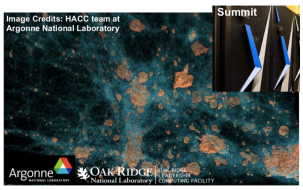
8

MPI Programming

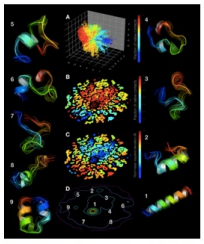
- Suitable architectures – Summit
  - Software
    - OS: RHEL 7.4; Compiler: XLC 13.1, nvcc 9.2
    - Math Lib.: ESSL, CUBLAS 9.2; MPI: Spectrum MPI
    - AI software: PowerAI DDL
  - Applications



Plasma Simulation



Cosmological Survey Simulation

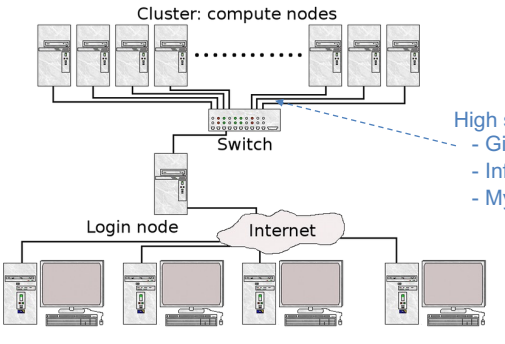


Cancer Surveillance
Source : National Cancer Institute

9

MPI Programming

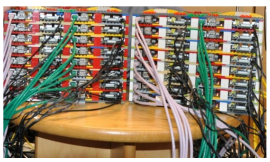
- Suitable architectures
  - **Cluster**



NOW-Cluster (Source : Google)

Beowulf project – NASA 1994

High speed interconnect technologies:
- Gigabit Ethernet
- InfiniBand
- Myrinet

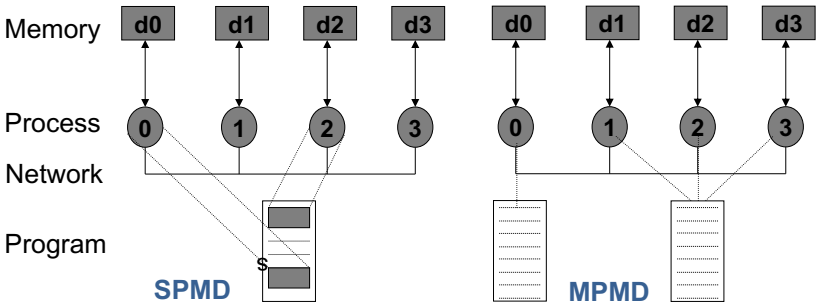


Raspberry Pi Supercomputer
University of Southampton 2012

10

MPI Programming Model

- SPMD / MPMD
  - Single / Multiple Program Multiple Data
  - Data distributed over process
  - Communication between processes via interconnexion network



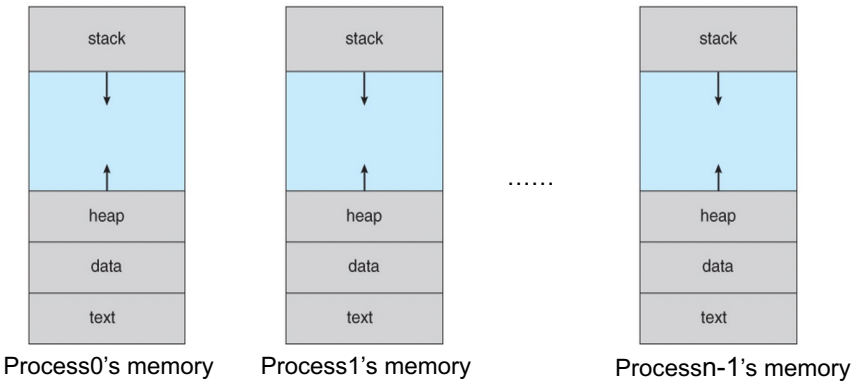
10

11

MPI Program

Multi-processing

- One program – Multiple processes

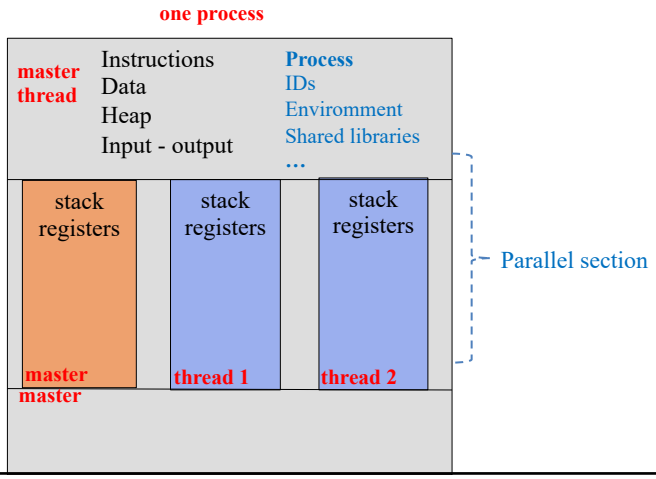


11

12

Rappel – OpenMP Program

- One program – Multiple threads **Multi-threading**

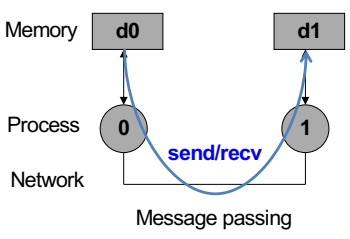


12

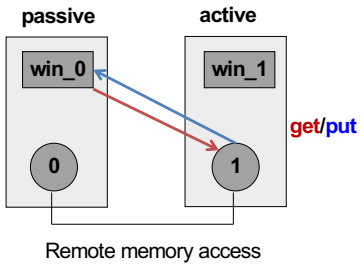
13

MPI - MPI Programming Model

- Communication model
  - Message passing (two sides operation)



- Communication model
  - Remote memory access (one side operation)



13

14

MPI – Message passing Interface

- Include:
  - Environment management routines
  - Point-to-point communication
  - Datatypes management
  - Collective communications
  - Groups of processes and communicators
  - Process topologies
  - Parallel I/O, RMA, dynamic process (MPI-2)
  - Non-blocking collective communication, RMA improvement (MPI-3)

to use with C (C++), Fortran

Version of MPI with OpenMPI: `$ ompi_info`

14

15

MPI – Environment Management Routines --- Examples

- Process enrolment into MPI environment

```
double MPI_init(int *ptArgc, char ***ptArgv);
```

- Get out of MPI environment

```
double MPI_Finalize(void);
```

- Get the processor name and its length

```
int MPI_Get_processor_name(char *name, int *nameLength);
```

- Terminates all process of communicator if exception: ex. after `malloc()`

```
int MPI_Abort(MPI_Comm comm, int errorcode);
```

15

16

16

17

MPI's world

- Group of processes and Communicator
  - MPI process are enrolled into groups
  - Group + context = Communicator
  - Default communicator: `MPI_COMM_WORLD`
  - Process identificator (`rank`): `0, 1, …, nbProcs-1`
  - `MPI_Comm_rank( MPI_COMM_WORLD, &rank );`
  - `MPI_Comm_size( MPI_COMM_WORLD, &nbProcs );`

`MPI_COMM_WORLD`

17

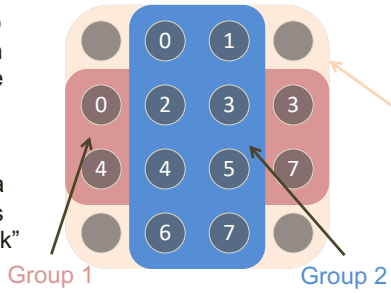
18

MPI's Communicators

Total processes number = 16

Communicators do not need to contain all processes in the system

Every process in a communicator has an ID called as "rank"



When you start an MPI program, there is one predefined communicator `MPI_COMM_WORLD`

Can make copies of this communicator (same group of processes, but different "aliases")

The same process might have different ranks in different communicators

19

MPI - Environment Management Routines

- hello_mpi.c

```
#include <stdio.h>
#include <mpi.h>        MPI's header file

int main(int argc, char **argv)
{
    int myRank, nbProcs;    ! one copy per process

    MPI_Init( &argc, &argv );        Execution environment initialization
    MPI_Comm_rank( MPI_COMM_WORLD, &myRank );
    MPI_Comm_size( MPI_COMM_WORLD, &nbProcs);

    printf( " Hello from proc. %d/%d\n ",
            myRank, nbProcs);        myRank: 0,1, … nbProcs-1

    MPI_Finalize(); End of MPI execution
    return 0;
}
```

MPI – Environment Management Routines

- hello_mpi.c with processor's information

lab work 1

```
#include <stdio.h>
#include "mpi.h"
#include <sched.h>

int main(int argc, char **argv)
{
   int  myRank=-1, nbProcs=0, nameLength=0;
   int  cpuId=-1; /* Number of core used */
   char procName[MPI_MAX_PROCESSOR_NAME]; /* name of node used */

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &myRank );
   MPI_Comm_size( MPI_COMM_WORLD, &nbProcs );

   MPI_Get_processor_name(procName, &nameLength) ;
   cpuId = sched_getcpu();
   printf( " Hello from proc. %d/%d on CPU %d of %s\n ",
           myRank, nbProcs, cpuId, procName);

   MPI_Finalize(); return 0;
}
```

20

MPI – Compiling and running MPI applications

- Connecting to `frontalhpc2020`

```
$ ssh my_account@frontalhpc2020.local.isima.fr
```

- Compiling

```
$ mpicc hello_mpi.c -o hello_mpi   # ( +compiling options )
```

- Direct running

Running without SLURM Do NOT do it!

  - 8 processes on local node (`frontalhpc2020`)

```
$ mpiexec -np 8 ./hello_mpi
```

21

22

MPI – Compiling and running MPI applications

- Running with SLURM

```
$ sbatch submit_hello_mpi.sh
$ more slurm-xxxxxx.out # result is in this file
```

```
#!/bin/bash
# submit_hello_mpi.sh execution script

#SBATCH --partition=peda     # execution partition 'peda'
#SBATCH --ntasks=8            # 8 tasks => 8 processes in parallel
#SBATCH --cpus-per-task=1     # of 1 thread
#SBATCH --ntasks-per-core=1   # 1 task per core
#SBATCH --job-name=hello_mpi

#execution
mpiexec ./hello_mpi
```

22

23

23