Day10

Table of Contents

1. Scripts

1.1. compile script

```
#!/bin/sh

#source /opt/ohpc/pub/apps/spack/share/spack/setup-env.sh
#spack load gcc/5i5y5cb
#spack load openmpi/c7kvqyq
source ~/git/spack/share/spack/setup-env.sh
spack load openmpi

inputFile=$1
outputFile="${1%.*}.out"      # extract the name of the file without extension and adding extension .out
#cmd=`mpicc $inputFile -o $outputFile`
cmd="mpicc $inputFile -o $outputFile"     # running code using MPI
echo "-----------------------------------------------------------------------"
echo "Command executed: $cmd"
echo "-----------------------------------------------------------------------"
$cmd

echo "Compilation successful. Check at $outputFile"
echo "-----------------------------------------------------------------------"
```

1.2. run script

```
#!/bin/sh

#source /opt/ohpc/pub/apps/spack/share/spack/setup-env.sh
#spack load gcc/5i5y5cbc
source ~/git/spack/share/spack/setup-env.sh
spack load openmpi

cmd="mpirun -np $2 $1"
echo "-----------------------------------------------------------------------"
echo "Command executed: $cmd"
echo "-----------------------------------------------------------------------"
echo "#######################################################################"
echo "##########                    OUTPUT                     ##########"
echo "#######################################################################"
echo
mpirun -np $2 $1
```

```
echo
echo "################################################################"
echo "##########                     DONE                     ##########"
echo "################################################################"
```

2. Introduction to MPI

MPI (Message Passing Interface) is a standardized library for message-passing in parallel programming. It allows multiple processes to communicate and coordinate tasks across distributed-memory systems. MPI is widely used for high-performance computing applications.

- **Key Highlights**:
  - Enables communication between processes running on different nodes or cores.
  - Portable across various hardware and software platforms.
  - Scalable to thousands or even millions of processes.
  - Provides fine-grained control over communication patterns.

3. Basics of MPI

- **Processes**:
  - Each instance of an MPI program is a separate process.
  - Processes do not share memory and communicate explicitly via messages.
- **Communicator**:
  - A group of processes that can communicate with each other.
  - The default communicator `MPI_COMM_WORLD` includes all processes.
- **Rank**:
  - Each process in a communicator is assigned a unique **rank** (an integer).
  - Ranks are used to identify and address processes.
- **Execution Model**:
  - All processes start execution from the same program code.
  - They can follow different execution paths based on their rank.

_

4. Processes vs Threads

Feature	Processes	Threads
Memory	Separate memory for each process.	Shared memory within the process.
Communication	Message passing (explicit).	Shared variables (implicit).
Scalability	Highly scalable.	Limited by shared memory capacity.
Example	MPI programs.	OpenMP programs.

—

5. Distributed Memory Programming Model

- In distributed memory systems, processes execute on separate nodes, each with its own private memory.
- Communication between processes occurs explicitly using message-passing.
- **Key Characteristics**:
  - No shared memory: Processes cannot directly access each other's data.
  - Explicit communication: Processes exchange data via messages.
  - Suitable for large-scale distributed systems like clusters and supercomputers.

—

6. Distributed vs Shared Memory

Feature	Shared Memory	Distributed Memory
Memory Access	All threads share a global memory.	Each process has private memory.
Communication	Implicit via shared variables.	Explicit via message passing.
Programming Models	OpenMP, Pthreads.	MPI, Sockets.
Scalability	Limited by shared memory size.	Highly scalable for large systems.

—

7. Why MPI?

1. **Scalability**:
   - Handles thousands of processes efficiently on distributed systems.
2. **Portability**:
   - Works on diverse hardware architectures and operating systems.
3. **Flexibility**:
   - Provides control over data distribution, load balancing, and communication.
4. **Efficiency**:
   - Optimized for high-performance computing on clusters and supercomputers.

8. Real-World Applications of MPI

- Climate modeling.
- Computational fluid dynamics.
- Genome sequencing.
- Financial simulations.

—

9. How MPI Works

1. **Initialization**:
   - The MPI environment is set up using `MPI_Init`.
   - All processes start executing from the same program.
2. **Communication**:
   - Processes exchange data via point-to-point or collective communication.
   - Use communicators (e.g., `MPI_COMM_WORLD`) to define the scope of communication.
3. **Synchronization**:
   - Processes can synchronize using barriers or other mechanisms.
4. **Finalization**:
   - The MPI environment is cleaned up using `MPI_Finalize`.

—

10. MPI Communications

- **Point-to-Point Communication**:
  - Direct communication between two specific processes.
  - Example Functions:
    - `MPI_Send`: Sends a message.
    - `MPI_Recv`: Receives a message.
- **Collective Communication**:
  - Involves all processes in a communicator.
  - Example Functions:
    - `MPI_Bcast`: Broadcasts a message to all processes.
    - `MPI_Reduce`: Combines data from all processes.

_

11. Downloading and Installing MPI

To get started with MPI, you need to download and install an MPI implementation. Here are general steps for downloading and installing Open MPI:

1. **Download Open MPI**: Visit the [Open MPI website](https://www.open-mpi.org) and download the latest version of Open MPI.

2. **Extract the tarball**:

   ```
   tar -xvf openmpi-x.y.z.tar.gz
   cd openmpi-x.y.z
   ```

3. **Configure, Build, and Install**:

   ```
   ./configure --prefix=/path/to/install
   make
   make install
   ```

4. **Set Environment Variables**: Add the following lines to your `.bashrc` or `.bash_profile`:

```
export PATH=/path/to/install/bin:$PATH
export LD_LIBRARY_PATH=/path/to/install/lib:$LD_LIBRARY_PATH
```

12. Loading MPI on PARAM shavak

```
source /home/apps/spack/share/spack/setup.env.sh # source spack package manager
spack find openmpi  # check if mpi is installed or not
# spack install -j40 openmpi   # if not installed then this command will install the latest version of openmpi
spack load openmpi/_your_hash   # load mpi with specific has if multiple version is installed
```

13. MPI Hello World Example

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

int main(int argc, char** argv) {
    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

    // Get the size of the communicator (number of processes)
    int world_size      ;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the current process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    // Print a message from each process
    printf("Hello from process %d of %d\n", world_rank, world_size);

    // Finalize the MPI environment
    MPI_Finalize();
    return 0;
}
```

—

14. Detailed Explanation of Hello World Code

1. **MPI_Init**:
    - Initializes the MPI environment.
    - Required before calling any other MPI functions.
    - Syntax: ```c MPI_Init(&argc, &argv); ```
2. **MPI_COMM_WORLD**:
    - Default communicator that includes all processes in the MPI program.
    - Every process is part of this communicator.
3. **MPI_Comm_size**:
    - Retrieves the total number of processes in the communicator.
    - Syntax: ```c MPI_Comm_size(MPI_COMM_WORLD, &world_size); ```
    - Example:
        - If there are 4 processes, `world_size` will be `4`.
4. **MPI_Comm_rank**:
    - Retrieves the rank of the current process in the communicator.
    - Syntax: ```c MPI_Comm_rank(MPI_COMM_WORLD, &world_rank); ```
    - Example:
        - If there are 4 processes, their ranks will be `0`, `1`, `2`, and `3`.
5. **MPI_Finalize**:
    - Cleans up the MPI environment.
    - Syntax: ```c MPI_Finalize(); ```

—

15. Hello World in C

15.1. code

```c
#include<stdio.h>
int main(){
    printf("Hello, World\n");
    return 0;
}
```

15.2. compile

```
gcc hello.c -o hello.out
```

15.3. run

```
./hello.out
```

```
Hello, World
```

16. Hello World in using MPI

16.1. code

```c
#include<stdio.h>
#include<mpi.h>
int main(){
    MPI_Init(NULL, NULL);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Hello from process %d of %d\n", rank, size);
    MPI_Finalize();
    return 0;
}
```

16.2. compile

```
#source ~/git/spack/share/spack/setup-env.sh
#spack load openmpi
#mpicc hello.c
bash compile.sh hello1.c
```

```
-----------------------------------------------------------------
Command executed: mpicc hello1.c -o hello1.out
-----------------------------------------------------------------
Compilation successful. Check at hello1.out
-----------------------------------------------------------------
```

16.3. run

```
#source ~/git/spack/share/spack/setup-env.sh
#spack load openmpi
#mpirun -np 4 ./a.out
bash run.sh ./hello1.out 4
```

```
-----------------------------------------------------------------
Command executed: mpirun -np 4 ./hello1.out
-----------------------------------------------------------------
#################################################################
##########                    OUTPUT                   ##########
#################################################################

Hello from process 0 of 4
Hello from process 1 of 4
Hello from process 3 of 4
Hello from process 2 of 4

#################################################################
##########                     DONE                    ##########
#################################################################
```

17. task1

```c
#include<stdio.h>
#include<mpi.h>
#define N 1000
int main(){
    int size, rank;
    int a[N];
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int chunksize = N / size;
    int start = rank * chunksize;
    int end = start + chunksize;
    if(rank == size - 1) end = N;
    for(int i = start; i < end; i++){
        a[i] = i + 1;
    }

    for(int i = start; i < end; i++){
        printf("%d ", a[i]);
    }
    printf("\n");
    MPI_Finalize();
}
```

```
bash compile.sh task1.c
```

```
---------------------------------------------------------------------
Command executed: mpicc task1.c -o task1.out
---------------------------------------------------------------------
Compilation successful. Check at task1.out
---------------------------------------------------------------------
```

```
bash run.sh ./task1.out 4 > output.txt
```

18. Point-to-point communication

```
#include"stdio.h"
#include"mpi.h"

int main()
{
    int myid, size;
    int myval;
    MPI_Init(NULL,NULL);

    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
```

```
    if(myid==0){
        myval = 100;
        printf("\nmyid: %d \t myval = %d", myid, myval);
        for(int i = 1; i < size; i++){
            MPI_Send(&myval, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
        printf("\nmyid: %d \t Data sent.\n", myid);
    }
    else{   // Process with ID exactly equal to 1
        if(myid == size - 1){
            printf("I left\n");
        }
        else{
               myval = 200;
               MPI_Recv(&myval, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
               printf("\nmyid: %d \t Data received.\n", myid);
               printf("\nmyid: %d \t myval = %d\n", myid, myval);
        }
    }

    MPI_Finalize();
}
```

```
bash compile.sh p2p_mpi.c
```

```
---------------------------------------------------------------------
Command executed: mpicc p2p_mpi.c -o p2p_mpi.out
---------------------------------------------------------------------
Compilation successful. Check at p2p_mpi.out
---------------------------------------------------------------------
```

```
bash run.sh ./p2p_mpi.out 4
```

```
---------------------------------------------------------------------
Command executed: mpirun -np 4 ./p2p_mpi.out
---------------------------------------------------------------------
#####################################################################
##########                   OUTPUT                    ##########
#####################################################################

I left
```

```
myid: 0          myval = 100
myid: 0          Data sent.

myid: 2          Data received.

myid: 2          myval = 100

myid: 1          Data received.

myid: 1          myval = 100

#####################################################################
##########                    DONE                     ##########
#####################################################################
```

18.1. Sending array to process 1

```c
#include"stdio.h"
#include"mpi.h"
#define N 100

int main()
{
    int myid, size;
    int myval;

    int arr[N];
    //Initialize MPI environment
    MPI_Init(NULL,NULL);

    //Get total number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    //Get my unique ID among all processes
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    // Process with ID exactly equal to 0
    if(myid==0){
        //Initialize data to be sent
        for(int i = 0; i < N; i++) arr[i] = i + 1;
        //Send data
        MPI_Send(arr, N, MPI_INT, 1, 0, MPI_COMM_WORLD);
        printf("\nmyid: %d \t Data sent.\n", myid);
    }
```

```
    else if(myid==1){   // Process with ID exactly equal to 1
        //Initialize receive array to some other data
        MPI_Recv(arr, N, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("\nmyid: %d \t Data received.\n", myid);
        //Print received data
        for(int i = 0; i < N; i++)
          printf("%d ", arr[i]);
    }

    //End MPI environment
    MPI_Finalize();
}
```

```
bash compile.sh p2p_mpi_array.c
```

```
---------------------------------------------------------------------
Command executed: mpicc p2p_mpi_array.c -o p2p_mpi_array.out
---------------------------------------------------------------------
Compilation successful. Check at p2p_mpi_array.out
---------------------------------------------------------------------
```

```
bash run.sh ./p2p_mpi_array.out 2
```

```
---------------------------------------------------------------------
Command executed: mpirun -np 2 ./p2p_mpi_array.out
---------------------------------------------------------------------
#####################################################################
##########                    OUTPUT                     ##########
#####################################################################


myid: 1          Data received.

myid: 0          Data sent.
1 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
#####################################################################
##########                     DONE                      ##########
#####################################################################
```

19. Point to point communication

This will create 1000 send calls and 1000 recv calls which is not good for your network.

```c
#include<stdio.h>
#include<mpi.h>
#define N 1000
int main(){
    int size, rank;
    int a[N];
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if(rank == 0){
        for(int i = 0; i < N; i++){
                a[i] = i + 1;
        }

        for(int i = 0; i < N; i++){
            MPI_Send(&a[i], 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        }
    }
    else{
        for(int i = 0; i < N; i++){
            MPI_Recv(&a[i], 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
        for(int i = N - 10; i < N; i++){
            printf("%d ", a[i]);
        }
    }

    MPI_Finalize();
}
```

```
bash compile.sh p2p.c
```

```
----------------------------------------------------------------
Command executed: mpicc p2p.c -o p2p.out
----------------------------------------------------------------
Compilation successful. Check at p2p.out
----------------------------------------------------------------
```

```
bash run.sh ./p2p.out 2
```

```
--------------------------------------------------------------------
Command executed: mpirun -np 2 ./p2p.out
--------------------------------------------------------------------
######################################################################
##########                    OUTPUT                    ##########
######################################################################

991 992 993 994 995 996 997 998 999 1000
######################################################################
##########                     DONE                     ##########
######################################################################
```

19.1. Better way

```c
#include<stdio.h>
#include<mpi.h>
#define N 1000
int main(){
    int size, rank;
    int a[N];
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if(rank == 0){
        for(int i = 0; i < N; i++){
                a[i] = i + 1;
        }


        for(int i = 1; i < size; i++){
            MPI_Send(a, N, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
    }
    else{
        MPI_Recv(a, N, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        for(int i = N - 10; i < N; i++){
            printf("%d ", a[i]);
        }
        printf("\n");
    }

    MPI_Finalize();
}
```

```
bash compile.sh p2p1.c
```

```
---------------------------------------------------------------------
Command executed: mpicc p2p1.c -o p2p1.out
---------------------------------------------------------------------
Compilation successful. Check at p2p1.out
---------------------------------------------------------------------
```

```
bash run.sh ./p2p1.out 10
```

```
---------------------------------------------------------------------
Command executed: mpirun -np 10 ./p2p1.out
---------------------------------------------------------------------
#####################################################################
##########                    OUTPUT                     ##########
#####################################################################

991 992 993 994 995 996 997 998 999 1000
991 992 993 994 995 996 997 998 999 1000
991 992 993 994 995 996 997 998 999 1000
991 992 993 994 995 996 997 998 999 1000
991 992 993 994 995 996 997 998 999 1000
991 992 993 994 995 996 997 998 999 1000
991 992 993 994 995 996 997 998 999 1000
991 992 993 994 995 996 997 998 999 1000
991 992 993 994 995 996 997 998 999 1000
991 992 993 994 995 996 997 998 999 1000

#####################################################################
##########                     DONE                      ##########
#####################################################################
```

Author: Abhishek Raj
Created: 2025-01-03 Fri 12:06