Day1

Table of Contents

1. Agenda

- Recap
- Communicators
- MPI Communications

- MPI Programs
- Point to point communications

2. MPI Communicators

Communicators in MPI define a group of processes that can communicate with each other. The default communicator is `MPI_COMMWORLD`, which includes all the processes. Custom communicators can be created to define subgroups of processes for specific communication patterns.

3. Types of MPI Communications

MPI offers various communication mechanisms to facilitate different types of data exchanges between processes:

3.1. Point-to-Point Communication:

- **Blocking**: The sending and receiving operations wait until the message is delivered (e.g., `MPI_Send`, `MPI_Recv`).
- **Non-Blocking**: The operations return immediately, allowing computation and communication to overlap (e.g., `MPI_Isend`, `MPI_Irecv`).

3.2. Collective Communication:

These operations involve a group of processes and include:

- **Broadcast**: Send data from one process to all other processes (`MPI_Bcast`).
- **Scatter**: Distribute distinct chunks of data from one process to all processes (`MPI_Scatter`).
- **Gather**: Collect chunks of data from all processes to one process (`MPI_Gather`).
- **All-to-All**: Every process sends and receives distinct chunks of data (`MPI_Alltoall`).

Collectives can also include operations like reductions (`MPI_Reduce`, `MPI_Allreduce`) which perform computations on data from all processes and distribute the result.

4. Sample MPI code

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

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

    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;
}
```

This example demonstrates a simple MPI program where each process prints its rank and the total number of processes.

5. Scripts

5.1. compile script

```sh
#!/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 "------------------------------------------------------------------"
```

5.2. 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 "##################################################################"
```

6. Hello World in C

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

```
Hello World
```

7. Hello World just by using MPI

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

```
Hello World
```

```
bash compile.sh hello_mpi.c
```

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

```
bash run.sh ./hello_mpi.out 6
```

```
----------------------------------------------------------------
Command executed: mpirun -np 6 ./hello_mpi.out
----------------------------------------------------------------
################################################################
##########                    OUTPUT                    ##########
################################################################

Hello World
Hello World
Hello World
Hello World
Hello World
```

```
Hello World

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

8. Hello World with MPI routines

```c
#include<stdio.h>
#include<mpi.h>         // for using mpi functions
int main(){
    int size, rank;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Hello from process %d\n", rank);
    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh hello_mpi_processes.c
```

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

```
bash run.sh ./hello_mpi_processes.out 8
```

```
----------------------------------------------------------------------
Command executed: mpirun -np 8 ./hello_mpi_processes.out
----------------------------------------------------------------------
######################################################################
##########                    OUTPUT                    ##########
######################################################################

Hello from process 5
```

```
Hello from process 4
Hello from process 6
Hello from process 7
Hello from process 2
Hello from process 1
Hello from process 3
Hello from process 0

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

9. Hello from only even rank of processes

```c
#include<stdio.h>
#include<mpi.h>         // for using mpi functions
int main(){
    int size, rank;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if(rank % 2 == 0)
        printf("Hello from process %d\n", rank);
    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh hello_mpi_even_processes.c
```

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

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

```
----------------------------------------------------------------------
```

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

Hello from process 0
Hello from process 2
Hello from process 6
Hello from process 8
Hello from process 4

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

10. MPI Initialization: MPI_{Init} vs. $MPI_{Initthread}$

MPI provides two main functions to initialize the MPI environment: `$MPI_{Init}$` and `$MPI_{Initthread}$`. The primary difference is that `$MPI_{Initthread}$` allows you to specify the desired level of thread support.

10.1. Levels of Thread Support

- `$MPI_{THREADSINGLE}$`: Only one thread will execute.
- `$MPI_{THREADFUNNELED}$`: The process may be multi-threaded, but only the main thread will make MPI calls.
- `$MPI_{THREADSERIALIZED}$`: Multiple threads may make MPI calls, but only one at a time.
- `$MPI_{THREADMULTIPLE}$`: Multiple threads may make MPI calls with no restrictions.

10.2. MPI_{Init} Example

This example uses `$MPI_{Init}$` to initialize the MPI environment.

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

int main(int argc, char** argv) {
```

```
    // Initialize the MPI environment
    MPI_Init(&argc, &argv);

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

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

    // Print off a hello world message
    printf("Hello world from processor %d out of %d processors\n", world_rank, world_size);

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

10.3. Compilation and Execution (MPI_{Init})

- Compile the program:

```
bash compile.sh mpi_init.c
```

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

- Run the program:

```
  bash run.sh ./mpi_init.out 6
```

```
  ----------------------------------------------------------------
  Command executed: mpirun -np 6 ./mpi_init.out
  ----------------------------------------------------------------
```

```
##################################################################
##########                    OUTPUT                    ##########
##################################################################

Hello world from processor 1 out of 6 processors
Hello world from processor 2 out of 6 processors
Hello world from processor 5 out of 6 processors
Hello world from processor 0 out of 6 processors
Hello world from processor 4 out of 6 processors
Hello world from processor 3 out of 6 processors

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

10.4. MPI$_{Initthread}$ Example

This example uses `MPI$_{Initthread}$` to initialize the MPI environment with thread support.

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

int main(int argc, char** argv) {
    int provided;

    // Initialize the MPI environment with thread support
    MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &provided);

    // Check the level of thread support provided
    if (provided < MPI_THREAD_MULTIPLE) {
        printf("MPI does not provide required thread support\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

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

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

    // Print off a hello world message
```

```
    printf("Hello world from processor %d out of %d processors with thread support level %d\n", world_rank, world_size, provided);

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

10.5. Compilation and Execution ($MPI_{Initthread}$)

- Compile the program:

```
bash compile.sh mpi_init_thread.c
```

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

- Run the program:

```
bash run.sh ./mpi_init_thread.out 5
```

```
  ------------------------------------------------------------------
  Command executed: mpirun -np 5 ./mpi_init_thread.out
  ------------------------------------------------------------------
  ##################################################################
  ##########                   OUTPUT                   ##########
  ##################################################################

  Hello world from processor 0 out of 5 processors with thread support level 3
  Hello world from processor 2 out of 5 processors with thread support level 3
  Hello world from processor 3 out of 5 processors with thread support level 3
  Hello world from processor 1 out of 5 processors with thread support level 3
  Hello world from processor 4 out of 5 processors with thread support level 3

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

```
##############################################################
```

10.6. Summary

- `MPI_Init` is used for standard MPI initialization without considering threading.
- `MPI_Initthread` allows the program to specify and check the level of thread support.
  - Important for applications that require multi-threading in conjunction with MPI.
  - Ensures that the required thread support is available.

11. Point-to-point communication

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

int main(int argc, char **argv)
{
    int myid, size;
    int myval;
    MPI_Status status;

    //Initialize MPI environment
    MPI_Init(&argc,&argv);

    //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
        myval = 100;
        //Print the data to be sent
        printf("\nmyid: %d \t myval = %d", myid, myval);
        //Send data
        MPI_Send(&myval, 1, 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
        myval = 200;
        MPI_Recv(&myval, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("\nmyid: %d \t Data received.", myid);
        //Print received data
        printf("\nmyid: %d \t myval = %d", myid, myval);
        printf("\n\nProgram exit!\n");
    }

    //End MPI environment
    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 2
```

```
---------------------------------------------------------------------
Command executed: mpirun -np 2 ./p2p_mpi.out
---------------------------------------------------------------------
#####################################################################
##########                    OUTPUT                     ##########
#####################################################################


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

myid: 1          Data received.
myid: 1          myval = 100

Program exit!

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

```
################################################################
```

11.1. 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
#################################################################
##########                     DONE                    ##########
#################################################################
```

Author: Abhishek Raj
Created: 2024-07-01 Mon 16:59