Day12

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. sum4.c

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

```c
#include<stdlib.h>
#include<mpi.h>
int main(){
    int size, rank;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    const int n = 1000;
    int chunksize = n / size;
    int start = rank * chunksize;
    int end = start + chunksize;
    if(rank == size - 1){
        end = n;
        chunksize += n % size;
    }
    int arr[chunksize];

    int index = 0;
    for(int i = start; i < end; i++){
        arr[index] = i + 1;
        index++;
    }

    index = 0;
    int localsum = 0;
    for(int i = start; i < end; i++){
        localsum += arr[index];
        index++;
    }

    if(rank != 0){
        MPI_Send(&localsum, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    else{
        int totalsum = 0;
        totalsum += localsum;
        for(int i = 1; i < size; i++){
            MPI_Recv(&localsum, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            totalsum += localsum;
        }
        printf("Total sum = %d\n", totalsum);
    }

    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh sum4.c
```

```
---------------------------------------------------------------------
Command executed: mpicc sum4.c -o sum4.out
---------------------------------------------------------------------
Compilation successful. Check at sum4.out
---------------------------------------------------------------------
```

```
bash run.sh ./sum4.out 7
```

```
---------------------------------------------------------------------
Command executed: mpirun -np 7 ./sum4.out
---------------------------------------------------------------------
#####################################################################
##########                    OUTPUT                     ##########
#####################################################################

Total sum = 500500

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

3. calculate sum of rank of all the process

```c
#include<stdio.h>
#include<mpi.h>
int main(){
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    //rank sum = 0 + 1 + 2 + 3 + 4
    if(rank != 0){
        MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    else{
        int rankSum = rank;
```

```
        for(int i = 1; i < size; i++){
            MPI_Recv(&rank, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            rankSum+= rank;
        }
        printf("Sum of all the ranks = %d\n", rankSum);
    }
    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh rankSum.c
```

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

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

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

Sum of all the ranks = 45

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

4. send this ranksum to all the process

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

```c
int main(){
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    //rank sum = 0 + 1 + 2 + 3 + 4
    int rankSum;
    if(rank != 0){
        MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        MPI_Recv(&rankSum, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("process %d received rank sum = %d\n", rank, rankSum);
    }
    else{
        rankSum = rank;
        for(int i = 1; i < size; i++){
            MPI_Recv(&rank, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            rankSum+= rank;
        }
        printf("Sum of all the ranks = %d\n", rankSum);
        for(int i = 1; i < size; i++){
            MPI_Send(&rankSum, 1, MPI_INT, i, 1, MPI_COMM_WORLD);
        }
    }

    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh rankSum1.c
```

```
---------------------------------------------------------------------
Command executed: mpicc rankSum1.c -o rankSum1.out
---------------------------------------------------------------------
Compilation successful. Check at rankSum1.out
---------------------------------------------------------------------
```

```
bash run.sh ./rankSum1.out 10
```

```
---------------------------------------------------------------------
Command executed: mpirun -np 10 ./rankSum1.out
---------------------------------------------------------------------
```

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

Sum of all the ranks = 45
process 1 received rank sum = 45
process 4 received rank sum = 45
process 2 received rank sum = 45
process 3 received rank sum = 45
process 7 received rank sum = 45
process 8 received rank sum = 45
process 6 received rank sum = 45
process 5 received rank sum = 45
process 9 received rank sum = 45

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

5. MPI Broadcast

5.1. MPI_Bcast Example

5.1.1. mpi_bcast_example.c

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

int main() {
    MPI_Init(NULL, NULL);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int data;
    if (rank == 0) {
        data = 100;  // Root process initializes the data
    }

    // Broadcast the data from the root process to all processes
    MPI_Bcast(&data, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```
    printf("Process %d received data %d\n", rank, data);

    MPI_Finalize();
    return 0;
}
```

5.1.2. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_bcast.c
```

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

- Run the program:

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

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

Process 0 received data 100
Process 1 received data 100
Process 3 received data 100
Process 2 received data 100
Process 4 received data 100

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

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

In this example, the integer `data` is initialized to 100 in the root process (process 0). The `MPI_Bcast` function is called to broadcast the value of `data` to all processes in the communicator. After the broadcast, each process prints the received value.

5.2. MPI_Bcast Example with array

5.2.1. mpi_bcast_example.c

```c
#include <mpi.h>
#include <stdio.h>
#define N 10

int main() {
    MPI_Init(NULL, NULL);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int arr[N];
    if (rank == 0) {
        for(int i = 0; i < N; i++) arr[i] = i + 1;
    }

    // Broadcast the data from the root process to all processes
    MPI_Bcast(arr, N, MPI_INT, 0, MPI_COMM_WORLD);

    for(int i = 0; i < N; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");

    MPI_Finalize();
    return 0;
}
```

5.2.2. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_bcast1.c
```

```
---------------------------------------------------------------------
Command executed: mpicc mpi_bcast1.c -o mpi_bcast1.out
---------------------------------------------------------------------
Compilation successful. Check at mpi_bcast1.out
---------------------------------------------------------------------
```

- Run the program:

```
bash run.sh ./mpi_bcast1.out 5
```

```
---------------------------------------------------------------------
Command executed: mpirun -np 5 ./mpi_bcast1.out
---------------------------------------------------------------------
#####################################################################
##########                    OUTPUT                     ##########
#####################################################################

1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10

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

In this example, the integer `data` is initialized to 100 in the root process (process 0). The `MPI_Bcast` function is called to broadcast the value of `data` to all processes in the communicator. After the broadcast, each process prints the received value.

5.3. send this ranksum to all the process using bcast

```c
#include<stdio.h>
#include<mpi.h>
int main(){
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    //rank sum = 0 + 1 + 2 + 3 + 4
    int rankSum;
    if(rank != 0){
        MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    else{
        rankSum = rank;
        int temp;
        for(int i = 1; i < size; i++){
            MPI_Recv(&temp, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            rankSum+= temp;
        }
    }
    MPI_Bcast(&rankSum, 1, MPI_INT, 0, MPI_COMM_WORLD);
    printf("Rank %d : Sum of all the ranks = %d\n", rank, rankSum);

    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh rankSum2.c
```

```
---------------------------------------------------------------------
Command executed: mpicc rankSum2.c -o rankSum2.out
---------------------------------------------------------------------
Compilation successful. Check at rankSum2.out
---------------------------------------------------------------------
```

```
bash run.sh ./rankSum2.out 10
```

```
---------------------------------------------------------------------
Command executed: mpirun -np 10 ./rankSum2.out
---------------------------------------------------------------------
```

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

Rank 0 : Sum of all the ranks = 45
Rank 3 : Sum of all the ranks = 45
Rank 1 : Sum of all the ranks = 45
Rank 2 : Sum of all the ranks = 45
Rank 9 : Sum of all the ranks = 45
Rank 8 : Sum of all the ranks = 45
Rank 5 : Sum of all the ranks = 45
Rank 4 : Sum of all the ranks = 45
Rank 7 : Sum of all the ranks = 45
Rank 6 : Sum of all the ranks = 45

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

6. Tag

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

int main() {
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int data1, data2, data3, data4, data5;
    if(rank == 0){
        data1 = 100;
        data2 = 200;
        data3 = 300;
        data4 = 400;
        data5 = 500;
        MPI_Request request;
        MPI_Isend(&data1, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
        MPI_Isend(&data2, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
        MPI_Isend(&data3, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
        MPI_Isend(&data4, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
        MPI_Isend(&data5, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
    }
```

```
    else if(rank == 1){
        MPI_Recv(&data1, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&data2, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&data3, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&data4, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&data5, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("process %d\n", rank);
        printf("data1 %d\n", data1);
        printf("data2 %d\n", data2);
        printf("data3 %d\n", data3);
        printf("data4 %d\n", data4);
        printf("data5 %d\n", data5);
    }

    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh tag.c
```

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

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

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

process 1
data1 100
data2 200
data3 300
data4 400
data5 500
```

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

7. Anytag, Anysource

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

int main() {
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int data1, data2;
    if(rank != 0){
        data1 = rank;
        data2 = 234;
        MPI_Request request;
        MPI_Isend(&data1, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
        MPI_Isend(&data2, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &request);
    }
    if(rank == 0){
        MPI_Recv(&data1, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("data1 %d\n", data1);
    }

    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh anySource.c
```

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

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

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

data1 7

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

8. MPI_Status

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

int main() {
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int data1, data2;
    if(rank != 0){
        data1 = rank;
        data2 = 234;
        MPI_Request request;
        MPI_Isend(&data1, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
        MPI_Isend(&data2, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &request);
    }
    if(rank == 0){
        MPI_Status status;
        MPI_Recv(&data1, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        printf("data1 %d\n", data1);
        printf("sender %d\n", status.MPI_SOURCE);
        printf("tag %d\n", status.MPI_TAG);
    }
```

```
    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh mpi_status.c
```

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

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

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

data1 7
sender 7
tag 0

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

9. task1

```
#include<stdio.h>
#include<stdlib.h>
#include<mpi.h>
int main(){
    int size, rank;
    MPI_Init(NULL, NULL);
```

```
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    const int n = 10;
    int chunksize = n / size;
    int start = rank * chunksize;
    int end = start + chunksize;
    if(rank == size - 1){
        end = n;
        chunksize += n % size;
    }
    int* arr;
    if(rank == 0){
        arr = (int*)malloc(sizeof(int) * n);
        for(int i = 0; i < n; i++){
                arr[i] = i + 1;
        }
        for(int i = 1; i < size; i++){
            MPI_Send(&arr[i * chunksize], chunksize, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
    }
    else{
        arr = (int*)malloc(sizeof(int) * chunksize);
        MPI_Recv(arr, chunksize, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    for(int i = 0; i < chunksize; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");

    free(arr);

    MPI_Finalize();
    return 0;
}
```

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

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

```
bash run.sh ./task1.out 5
```

```
--------------------------------------------------------------------
Command executed: mpirun -np 5 ./task1.out
--------------------------------------------------------------------
######################################################################
##########                    OUTPUT                    ##########
######################################################################

3 4
1 2
7 8
5 6
9 10

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

Author: Abhishek Raj
Created: 2025-01-03 Fri 13:54