Day2

Table of Contents

1. Agenda

- Point-to-point communication
- Synchronous and Asynchronous calls
- Non-blocking calls
- Sum using p2p communication

2. Scripts

2.1. 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 "------------------------------------------------------------------"
```

2.2. 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 "##################################################################"
```

3. MPI Communication: Synchronous and Asynchronous

3.1. Synchronous Communication using MPI$_{\text{Send}}$ and MPI$_{\text{Recv}}$

In synchronous communication, the send operation does not complete until the matching receive operation has been started.

3.1.1. mpi$_{\text{sync.c}}$

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

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

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (size < 2) {
        fprintf(stderr, "World size must be greater than 1 for this example\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    int number;
    if (rank == 0) {
        number = -1;
        MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        printf("Process 0 sent number %d to process 1\n", number);
    } else if (rank == 1) {
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process 1 received number %d from process 0\n", number);
    }

    MPI_Finalize();
    return 0;
```

```
}
```

3.1.2. Compilation and Execution (Synchronous)

- Compile the program:

```
bash compile.sh mpi_sync.c
```

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

- Run the program:

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

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

Process 0 sent number -1 to process 1
Process 1 received number -1 from process 0

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

3.2. Asynchronous Communication using MPI_{Isend} and MPI_{Irecv}

In asynchronous communication, the send operation can complete before the matching receive operation starts. Non-blocking operations allow computation and communication to overlap.

3.2.1. mpi_async.c

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

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

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (size < 2) {
        fprintf(stderr, "World size must be greater than 1 for this example\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    int number;
    if (rank == 0) {
        number = -1;
        MPI_Request request;
        MPI_Isend(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
        //MPI_Wait(&request, MPI_STATUS_IGNORE);
        printf("Process 0 sent number %d to process 1\n", number);
    } else if (rank == 1) {
        MPI_Request request;
        MPI_Irecv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, MPI_STATUS_IGNORE);
        printf("Process 1 received number %d from process 0\n", number);
    }

    MPI_Finalize();
    return 0;
}
```

3.2.2. Compilation and Execution (Asynchronous)

- Compile the program:

```
bash compile.sh mpi_async.c
```

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

- Run the program:

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

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

Process 0 sent number -1 to process 1
Process 1 received number -1 from process 0

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

4. MPI Array Sum Calculation Example

4.1. mpi_arraysum.c

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

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

```c
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

int n = 10000; // Size of the array
int *array = NULL;
int chunk_size = n / world_size;
int *sub_array = (int*)malloc(chunk_size * sizeof(int));

if (world_rank == 0) {
    array = (int*)malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        array[i] = i + 1; // Initialize the array with values 1 to n
    }

    // Distribute chunks of the array to other processes
    for (int i = 1; i < world_size; i++) {
        MPI_Send(array + i * chunk_size, chunk_size, MPI_INT, i, 0, MPI_COMM_WORLD);
    }

    // Copy the first chunk to sub_array
    for (int i = 0; i < chunk_size; i++) {
        sub_array[i] = array[i];
    }
} else {
    // Receive chunk of the array
    MPI_Recv(sub_array, chunk_size, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}

// Compute the local sum
int local_sum = 0;
for (int i = 0; i < chunk_size; i++) {
    local_sum += sub_array[i];
}

if (world_rank != 0) {
    // Send local sum to process 0
    MPI_Send(&local_sum, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
} else {
    // Process 0 receives the local sums and computes the final sum
    int final_sum = local_sum;
    int temp_sum;
    for (int i = 1; i < world_size; i++) {
        MPI_Recv(&temp_sum, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        final_sum += temp_sum;
```

```
        }
        printf("The total sum of array elements is %d\n", final_sum);
    }

    free(sub_array);
    if (world_rank == 0) {
        free(array);
    }

    MPI_Finalize();
    return 0;
}
```

4.2. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_array_sum.c
```

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

- Run the program:

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

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

The total sum of array elements is 50005000
```

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

5. Task1

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

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

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (size < 2) {
        fprintf(stderr, "World size must be greater than 1 for this example\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    int number;
    if (rank == 0) {
        number = 100;
        MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        printf("Process 0 sent number %d to process 1\n", number);
        MPI_Recv(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process 0 received number %d from process 1\n", number);
    } else if (rank == 1) {
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process 1 received number %d from process 0\n", number);
        number = 200;
        MPI_Send(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        printf("Process 1 sent number %d to process 0\n", number);
    } else{
        printf("I am process %d and I have nothing to do\n", rank);
    }

    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 2
```

```
------------------------------------------------------------------
Command executed: mpirun -np 2 ./task1.out
------------------------------------------------------------------
##################################################################
##########                    OUTPUT                    ##########
##################################################################

Process 0 sent number 100 to process 1
Process 1 received number 100 from process 0
Process 1 sent number 200 to process 0
Process 0 received number 200 from process 1

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

6. Task2

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

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

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```c
    if (size < 2) {
        fprintf(stderr, "World size must be greater than 1 for this example\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    int number1, number2;
    if (rank == 0) {
        number1 = 100;
        number2 = 200;
        MPI_Send(&number1, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Send(&number2, 1, MPI_INT, 1, 2, MPI_COMM_WORLD);
        printf("Process 0 sent number %d to process 1\n", number1);
        printf("Process 0 sent number %d to process 1\n", number2);
    } else if (rank == 1) {
        MPI_Recv(&number1, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&number2, 1, MPI_INT, 0, 2, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process 1 received number %d from process 0\n", number1);
        printf("Process 1 received number %d from process 0\n", number2);
    } else{
        printf("I am process %d and I have nothing to do\n", rank);
    }

    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh task2.c
```

```
------------------------------------------------------------------
Command executed: mpicc task2.c -o task2.out
------------------------------------------------------------------
Compilation successful. Check at task2.out
------------------------------------------------------------------
```

```
bash run.sh ./task2.out 2
```

```
------------------------------------------------------------------
Command executed: mpirun -np 2 ./task2.out
------------------------------------------------------------------
##################################################################
```

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

Process 0 sent number 100 to process 1
Process 0 sent number 200 to process 1
Process 1 received number 100 from process 0
Process 1 received number 200 from process 0

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

7. Task3

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

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

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (size < 2) {
        fprintf(stderr, "World size must be greater than 1 for this example\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    int number1, number2;
    if (rank == 0) {
        number1 = 100;
        number2 = 200;
        MPI_Request request;
        MPI_Isend(&number1, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
        MPI_Isend(&number2, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
        printf("Process 0 sent number %d to process 1\n", number1);
        printf("Process 0 sent number %d to process 1\n", number2);
    } else if (rank == 1) {
        MPI_Request request;
        MPI_Irecv(&number1, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, MPI_STATUS_IGNORE);
        MPI_Irecv(&number2, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
```

```c
        MPI_Wait(&request, MPI_STATUS_IGNORE);
        printf("Process 1 received number %d from process 0\n", number1);
        printf("Process 1 received number %d from process 0\n", number2);
    } else{
        printf("I am process %d and I have nothing to do\n", rank);
    }

    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh task3.c
```

```
------------------------------------------------------------------
Command executed: mpicc task3.c -o task3.out
------------------------------------------------------------------
Compilation successful. Check at task3.out
------------------------------------------------------------------
```

```
bash run.sh ./task3.out 2
```

```
------------------------------------------------------------------
Command executed: mpirun -np 2 ./task3.out
------------------------------------------------------------------
##################################################################
##########                  OUTPUT                      ##########
##################################################################

Process 0 sent number 100 to process 1
Process 0 sent number 200 to process 1
Process 1 received number 100 from process 0
Process 1 received number 200 from process 0

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

8. Task4

```c
#include <mpi.h>
#include <stdio.h>
#define N 10000

int main(int argc, char** argv) {
    int arr[N];
    for(int i = 0; i < N; i++){
        arr[i] = i + 1;
    }
    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int chunksize = N / size;
    int start = chunksize * rank;
    int end = (rank + 1) * chunksize;
    if (size < 2) {
        fprintf(stderr, "World size must be greater than 1 for this example\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }
    int localSum = 0;
    for(int i = start; i < end; i++){
        localSum+= arr[i];
    }
    if(rank != 0){
        MPI_Send(&localSum, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    if (rank == 0) {
        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 task4.c
```

```
-----------------------------------------------------------------
Command executed: mpicc task4.c -o task4.out
-----------------------------------------------------------------
Compilation successful. Check at task4.out
-----------------------------------------------------------------
```

```
bash run.sh ./task4.out 10
```

```
-----------------------------------------------------------------
Command executed: mpirun -np 10 ./task4.out
-----------------------------------------------------------------
#################################################################
##########                   OUTPUT                    ##########
#################################################################

Total sum = 50005000

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

Author: Abhishek Raj
Created: 2024-07-03 Wed 07:05