Day15

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. MPI Alltoall Example

2.1. mpi_alltoall_example.c

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.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);
    int *send_data = (int*)malloc(size * sizeof(int));
    int *recv_data = (int*)malloc(size * sizeof(int));
    // Initialize send_data such that process i sends its rank to all processes
    for (int i = 0; i < size; i++) {
        send_data[i] = rank + i * 10;
    }

    printf("Process %d send data: ", rank);
    for (int i = 0; i < size; i++) {
        printf("%d ", send_data[i]);
    }
    printf("\n");
    // Perform all-to-all communication
    MPI_Alltoall(send_data, 1, MPI_INT, recv_data, 1, MPI_INT, MPI_COMM_WORLD);
    printf("Process %d received data: ", rank);
    for (int i = 0; i < size; i++) {
        printf("%d ", recv_data[i]);
    }
    printf("\n");
    free(send_data);
```

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

2.2. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_alltoall_example.c
```

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

- Run the program:

```
bash run.sh ./mpi_alltoall_example.out 3
```

```
------------------------------------------------------------------
Command executed: mpirun -np 3 ./mpi_alltoall_example.out
------------------------------------------------------------------
######################################################################
##########                    OUTPUT                    ##########
######################################################################

Process 0 send data: 0 10 20
Process 0 received data: 0 1 2
Process 1 send data: 1 11 21
Process 1 received data: 10 11 12
Process 2 send data: 2 12 22
Process 2 received data: 20 21 22
```

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

In this example, each process sends its rank and an incremented value to all other processes. The `MPI_Alltoall` function is used to exchange these values among all processes. Each process then prints the received values.

3. Task2

```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);
    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) {
        number = 200;
        MPI_Send(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        printf("Process 1 sent number %d to process 0\n", number);
        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);
    } 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 received number 200 from process 1
Process 1 sent number 200 to process 0
Process 1 received number 100 from process 0

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

4. MPI_Sendrecv Example

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

    MPI_Finalize();
    return 0;
}
```

4.1. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_sendrecv_example.c
```

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

- Run the program:

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

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

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

Process 1 received 100 and sent number 200
Process 0 sent 100 and received number 200

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

In this example, `MPI_Sendrecv` is used to send and receive messages in a single call. Process 0 sends the number 100 to process 1 and receives a number from process 1. Process 1 receives the number from process 0, modifies it to 200, and sends it back to process 0.

5. MPI_Sendrecv Example2

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

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

5.1. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_sendrecv_example2.c
```

```
---------------------------------------------------------------------
Command executed: mpicc mpi_sendrecv_example2.c -o mpi_sendrecv_example2.out
---------------------------------------------------------------------
Compilation successful. Check at mpi_sendrecv_example2.out
---------------------------------------------------------------------
```

- Run the program:

```
bash run.sh ./mpi_sendrecv_example2.out 2
```

```
---------------------------------------------------------------------
Command executed: mpirun -np 2 ./mpi_sendrecv_example2.out
---------------------------------------------------------------------
#####################################################################
##########                    OUTPUT                     ##########
#####################################################################

Process 0 received number 200
Process 1 received number 100

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

In this example, `MPI_Sendrecv` is used to send and receive messages in a single call. Process 0 sends the number 100 to process 1 and receives a number from process 1. Process 1 receives the number from process 0, modifies it to 200, and sends it back to process 0.

6. MPI_Sendrecv_replace Example

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

    MPI_Finalize();
    return 0;
}
```

6.1. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_sendrecv_replace_example.c
```

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

- Run the program:

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

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

Process 0 sent and received number 200
Process 1 sent and received number 100

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

In this example, `MPI_Sendrecv_replace` is used to send and receive messages using the same buffer. Process 0 sends the number 100 to process 1 and receives a number from process 1 into the same buffer. Process 1 receives the number from process 0, modifies it to 200, and sends it back to process 0 using the same buffer.

7. MPI_Send and recv Example with larger data

```c
#include <mpi.h>
#include <stdlib.h>
#include <stdio.h>
#define N 100000

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    MPI_Status status;
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int *buffer;
    buffer = (int*) malloc(sizeof(int) * N);
    for(int i = 0; i < N; i++){
        buffer[i] = (rank * i) + 1;
    }
    if(rank == 0){
        MPI_Send(buffer, N, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Recv(buffer, N, MPI_INT, 1, 0, MPI_COMM_WORLD, &status);
        printf("Process %d is having : ", rank);
        for(int i = N - 10; i < N; i++){
            printf("%d ", buffer[i]);
        }
        printf("\n");
    }
    else if(rank == 1){
        MPI_Recv(buffer, N, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        MPI_Send(buffer, N, MPI_INT, 0, 0, MPI_COMM_WORLD);
        printf("Process %d is having : ", rank);
        for(int i = N - 10; i < N; i++){
            printf("%d ", buffer[i]);
        }
        printf("\n");
    }

    MPI_Finalize();
    return 0;
}
```

7.1. Compilation and Execution

- Compile the program:

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

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

- Run the program:

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

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

Process 1 is having : 1 1 1 1 1 1 1 1 1 1
Process 0 is having : 1 1 1 1 1 1 1 1 1 1

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

In this example, `MPI_Sendrecv` is used to send and receive messages in a single call. Process 0 sends the number 100 to process 1 and receives a number from process 1. Process 1 receives the number from process 0, modifies it to 200, and sends it back to process 0.

8. Swap data of two process

```c
#include <mpi.h>
#include <stdlib.h>
#include <stdio.h>
#define N 100000

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    MPI_Status status;
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int *buffer;
    buffer = (int*) malloc(sizeof(int) * N);
    for(int i = 0; i < N; i++){
        buffer[i] = (rank * i) + i;
    }
    if(rank == 0){
        MPI_Send(buffer, N, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Recv(buffer, N, MPI_INT, 1, 0, MPI_COMM_WORLD, &status);
        printf("Process %d is having : ", rank);
        for(int i = N - 10; i < N; i++){
            printf("%d ", buffer[i]);
        }
        printf("\n");
    }
    else if(rank == 1){
        int *tempBuffer = (int*) malloc(sizeof(int) * N);
        MPI_Recv(tempBuffer, N, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        MPI_Send(buffer, N, MPI_INT, 0, 0, MPI_COMM_WORLD);
        for(int i = 0; i < N; i++) buffer[i] = tempBuffer[i];
        free(tempBuffer);
        printf("Process %d is having : ", rank);
        for(int i = N - 10; i < N; i++){
            printf("%d ", buffer[i]);
        }
        printf("\n");
    }

    MPI_Finalize();
    return 0;
}
```

8.1. Compilation and Execution

- Compile the program:

```
bash compile.sh task4.c
```

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

- Run the program:

```
bash run.sh ./task4.out 2
```

```
---------------------------------------------------------------------
Command executed: mpirun -np 2 ./task4.out
---------------------------------------------------------------------
#####################################################################
##########                    OUTPUT                       ##########
#####################################################################

Process 0 is having : 199980 199982 199984 199986 199988 199990 199992 199994 199996 199998
Process 1 is having : 99990 99991 99992 99993 99994 99995 99996 99997 99998 99999

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

9. Swap data of two process with MPI_Sendrecv_replace

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

```c
#include <stdlib.h>
#include <stdio.h>
#define N 100000

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    MPI_Status status;
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int *buffer;
    buffer = (int*) malloc(sizeof(int) * N);
    for(int i = 0; i < N; i++){
        buffer[i] = (rank * i) + i;
    }
    if(rank == 0){
        MPI_Sendrecv_replace(buffer, N, MPI_INT, 1, 0, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process %d is having : ", rank);
        for(int i = N - 10; i < N; i++){
            printf("%d ", buffer[i]);
        }
        printf("\n");
    }
    else if(rank == 1){
        MPI_Sendrecv_replace(buffer, N, MPI_INT, 0, 0, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process %d is having : ", rank);
        for(int i = N - 10; i < N; i++){
            printf("%d ", buffer[i]);
        }
        printf("\n");
    }

    MPI_Finalize();
    return 0;
}
```

9.1. Compilation and Execution

- Compile the program:

```
bash compile.sh task5.c
```

```
-----------------------------------------------------------------
Command executed: mpicc task5.c -o task5.out
-----------------------------------------------------------------
Compilation successful. Check at task5.out
-----------------------------------------------------------------
```

- Run the program:

```
bash run.sh ./task5.out 2
```

```
-----------------------------------------------------------------
Command executed: mpirun -np 2 ./task5.out
-----------------------------------------------------------------
#################################################################
##########                    OUTPUT                   ##########
#################################################################

Process 0 is having : 199980 199982 199984 199986 199988 199990 199992 199994 199996 199998
Process 1 is having : 99990 99991 99992 99993 99994 99995 99996 99997 99998 99999

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

10. MPI_Bsend

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.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);
    int number;
    if (rank == 0) {
        number = 100;

        int buffer_size = MPI_BSEND_OVERHEAD + sizeof(int);
        void* buffer = malloc(buffer_size);
        MPI_Buffer_attach(buffer, buffer_size);

        MPI_Bsend(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        printf("Process 0 sent number %d to process 1\n", number);

        MPI_Buffer_detach(&buffer, &buffer_size);
        free(buffer);
    } 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;
}
```

10.1. Compilation and Execution

- Compile the program:

```
  bash compile.sh mpi_bsend_example.c
```

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

- Run the program:

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

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

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

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

In this example, `MPI_Bsend` is used to send a number from process 0 to process 1 using a buffered send.

11. MPI_Bsend Array

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.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 array_size = 1000000;
    int* array = (int*)malloc(array_size * sizeof(int));

    if (rank == 0) {
        // Initialize the array with some values
        for (int i = 0; i < array_size; i++) {
            array[i] = i + 1;
        }

        int buffer_size = MPI_BSEND_OVERHEAD + array_size * sizeof(int);
        void* buffer = malloc(buffer_size);
        MPI_Buffer_attach(buffer, buffer_size);

        MPI_Bsend(array, array_size, MPI_INT, 1, 0, MPI_COMM_WORLD);
        printf("Process 0 sent array to process 1\n");

        MPI_Buffer_detach(&buffer, &buffer_size);
        free(buffer);
    } else if (rank == 1) {
        MPI_Recv(array, array_size, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process 1 received array from process 0: ");
        for (int i = array_size - 5; i < array_size; i++) {
            printf("%d ", array[i]);
        }
        printf("\n");
    }

    free(array);

    MPI_Finalize();
    return 0;
}
```

11.1. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_bsend_array_example.c
```

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

- Run the program:

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

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

Process 0 sent array to process 1
Process 1 received array from process 0: 999996 999997 999998 999999 1000000

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

In this example, `MPI_Bsend` is used to send an array of integers from process 0 to process 1 using a buffered send. The received data is printed by process 1.

Author: Abhishek Raj
Created: 2025-01-03 Fri 17:51