Day5

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. Task1

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

```c
#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) {
        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 0 received number 200 from process 1
Process 1 received number 100 from process 0
Process 1 sent number 200 to process 0

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

3. $\text{MPI}_{\text{Sendrecv}}$ 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;
}
```

3.1. 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.

4. 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;
}
```

4.1. 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.

5. MPI_Sendrecvreplace 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 received and sent number %d\n", number);
    } 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_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 received and sent number 100

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

In this example, `MPI_Sendrecvreplace` 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.

6. Task2: Cyclic sum of all the ranks

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

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int rank, sum = 0;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int number;
    if (rank == 0) {
        sum = 0;
        MPI_Send(&sum, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD);
        MPI_Recv(&sum, 1, MPI_INT, size - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Cyclic sum: %d\n", sum);
    } else if (rank == size - 1) {
        MPI_Recv(&sum, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        sum+= rank;
        MPI_Send(&sum, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    } else{
        MPI_Recv(&sum, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        sum+= rank;
        MPI_Send(&sum, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh cyclic_sum.c
```

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

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

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

Cyclic sum: 45

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

7. MPI_{Ssend}

```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_Ssend(&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;
}
```

7.1. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_ssend_example.c
```

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

- Run the program:

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

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

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

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

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

In this example, `MPI$_{Ssend}$` is used to send a number from process 0 to process 1 synchronously.

8. 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;
}
```

8.1. 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.

9. 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 = 10;
    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 = 0; i < array_size; i++) {
            printf("%d ", array[i]);
        }
        printf("\n");
    }

    free(array);

    MPI_Finalize();
    return 0;
}
```

9.1. 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: 1 2 3 4 5 6 7 8 9 10

#################################################################
##########                     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: 2024-07-06 Sat 16:42