Day4

Table of Contents

- 1. Scripts
  - 1.1. compile script
  - 1.2. run script
- 2. MPI Allreduce Example with long datatype
  - 2.1. mpi_allreducelong.c
  - 2.2. Compilation and Execution
- 3. mpi_gather and broadcast
- 4. MPI Allgather Example
  - 4.1. allgather
  - 4.2. mpi_allgatherexample.c
  - 4.3. Compilation and Execution
- 5. MPI Alltoall Example
  - 5.1. mpi_alltoallexample.c
  - 5.2. Compilation and Execution
- 6. MPI_Status Example
  - 6.1. mpi_statusexample.c
  - 6.2. Compilation and Execution
- 7. MPI_Test with MPI_Isend and MPI_Irecv Example
  - 7.1. mpi_testexample.c
  - 7.2. Compilation and Execution

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 Allreduce Example with long datatype

2.1. mpi_allreducelong.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);

    long n = 100000000; // Size of the array
    long *array = NULL;
    long chunk_size = n / size;
    long *sub_array = (long*)malloc(chunk_size * sizeof(long));

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

    // Scatter the chunks of the array to all processes
    MPI_Scatter(array, chunk_size, MPI_LONG, sub_array, chunk_size, MPI_LONG, 0, MPI_COMM_WORLD);

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

    // Compute the total sum using allreduce
    long total_sum = 0;
    MPI_Allreduce(&local_sum, &total_sum, 1, MPI_LONG, MPI_SUM, MPI_COMM_WORLD);

    printf("Process %d: The total sum of array elements is %ld\n", rank, total_sum);

    if (rank == 0) {
        free(array);
    }
    free(sub_array);
```

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

2.2. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_allreduce_long.c
```

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

- Run the program:

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

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

Process 3: The total sum of array elements is 5000000050000000
Process 4: The total sum of array elements is 5000000050000000
Process 9: The total sum of array elements is 5000000050000000
Process 7: The total sum of array elements is 5000000050000000
Process 5: The total sum of array elements is 5000000050000000
Process 8: The total sum of array elements is 5000000050000000
Process 2: The total sum of array elements is 5000000050000000
Process 6: The total sum of array elements is 5000000050000000
Process 1: The total sum of array elements is 5000000050000000
Process 0: The total sum of array elements is 5000000050000000
```

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

In this example, the array is initialized with long integers and the `MPI_Scatter` function is used to distribute chunks of the array to all processes. Each process computes the local sum of its chunk and the `MPI_Allreduce` function is used to compute the total sum and distribute it to all processes.

3. mpi_gather and broadcast

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

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int send_data = rank; // Each process sends its rank
    int *recv_data = NULL;
    recv_data = (int*)malloc(size * sizeof(int)); // Allocate memory for receiving data
    // Gather the data from all processes to the root process
    MPI_Gather(&send_data, 1, MPI_INT, recv_data, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(recv_data, size, MPI_INT, 0, MPI_COMM_WORLD);

    for (int i = 0; i < size; i++) {
        printf("Data at %d index = %d and printed by %d\n", i, recv_data[i], rank);
    }
    printf("\n");
        free(recv_data);
    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh mpi_gather_and_bcast.c
```

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

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

```
bash run.sh ./mpi_gather_and_bcast.out 4
```

```
--------------------------------------------------------------------
Command executed: mpirun -np 4 ./mpi_gather_and_bcast.out
--------------------------------------------------------------------
######################################################################
##########                    OUTPUT                    ##########
######################################################################

Data at 0 index = 0 and printed by 0
Data at 1 index = 1 and printed by 0
Data at 2 index = 2 and printed by 0
Data at 3 index = 3 and printed by 0

Data at 0 index = 0 and printed by 2
Data at 1 index = 1 and printed by 2
Data at 2 index = 2 and printed by 2
Data at 3 index = 3 and printed by 2

Data at 0 index = 0 and printed by 1
Data at 1 index = 1 and printed by 1
Data at 2 index = 2 and printed by 1
Data at 3 index = 3 and printed by 1

Data at 0 index = 0 and printed by 3
Data at 1 index = 1 and printed by 3
Data at 2 index = 2 and printed by 3
Data at 3 index = 3 and printed by 3


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

4. MPI Allgather Example

4.1. allgather

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

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int send_data = rank; // Each process sends its rank
    int *recv_data = NULL;
    recv_data = (int*)malloc(size * sizeof(int)); // Allocate memory for receiving data
    // Gather the data from all processes to the root process
    MPI_Allgather(&send_data, 1, MPI_INT, recv_data, 1, MPI_INT, MPI_COMM_WORLD);

    for (int i = 0; i < size; i++) {
        printf("Data at %d index = %d and printed by %d\n", i, recv_data[i], rank);
    }
    printf("\n");
        free(recv_data);
    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh mpi_allgather1.c
```

```
---------------------------------------------------------------------
Command executed: mpicc mpi_allgather1.c -o mpi_allgather1.out
---------------------------------------------------------------------
Compilation successful. Check at mpi_allgather1.out
---------------------------------------------------------------------
```

```
bash run.sh ./mpi_allgather1.out 4
```

```
---------------------------------------------------------------------
Command executed: mpirun -np 4 ./mpi_allgather1.out
---------------------------------------------------------------------
```

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

Data at 0 index = 0 and printed by 2
Data at 1 index = 1 and printed by 2
Data at 2 index = 2 and printed by 2
Data at 3 index = 3 and printed by 2

Data at 0 index = 0 and printed by 3
Data at 1 index = 1 and printed by 3
Data at 2 index = 2 and printed by 3
Data at 3 index = 3 and printed by 3

Data at 0 index = 0 and printed by 0
Data at 1 index = 1 and printed by 0
Data at 2 index = 2 and printed by 0
Data at 3 index = 3 and printed by 0

Data at 0 index = 0 and printed by 1
Data at 1 index = 1 and printed by 1
Data at 2 index = 2 and printed by 1
Data at 3 index = 3 and printed by 1


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

4.2. mpi_allgatherexample.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 = rank; // Each process sends its rank
```

```
    int *recv_data = (int*)malloc(size * sizeof(int));

    // Allgather the data from all processes
    MPI_Allgather(&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(recv_data);
    MPI_Finalize();
    return 0;
}
```

4.3. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_allgather_example.c
```

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

- Run the program:

```
bash run.sh ./mpi_allgather_example.out 4
```

```
---------------------------------------------------------------------
Command executed: mpirun -np 4 ./mpi_allgather_example.out
---------------------------------------------------------------------
#####################################################################
##########                   OUTPUT                    ##########
#####################################################################
```

```
Process 1 received data: 0 1 2 3
Process 2 received data: 0 1 2 3
Process 0 received data: 0 1 2 3
Process 3 received data: 0 1 2 3

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

In this example, each process sends its rank as `send$_{data}$`. The `MPI$_{Allgather}$` function is called to gather the values from all processes and distribute them to all processes. Each process then prints the gathered values.

5. MPI Alltoall Example

5.1. mpi$_{alltoallexample.c}$

```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;
    }
    // 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;
}
```

5.2. 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 4
```

```
------------------------------------------------------------------
Command executed: mpirun -np 4 ./mpi_alltoall_example.out
------------------------------------------------------------------
######################################################################
##########                    OUTPUT                    ##########
######################################################################

Process 1 received data: 10 11 12 13
Process 0 received data: 0 1 2 3
Process 3 received data: 30 31 32 33
Process 2 received data: 20 21 22 23

######################################################################
##########                     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.

6. MPI_Status Example

6.1. mpi_statusexample.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 number;
    MPI_Status status;

    if (rank == 0) {
        number = 100;
        MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
    } else if (rank == 1) {
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        printf("Process 1 received number %d from process %d with tag %d\n",
               number, status.MPI_SOURCE, status.MPI_TAG);
    }

    MPI_Finalize();
    return 0;
}
```

6.2. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_status_example.c
```

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

- Run the program:

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

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

Process 1 received number 100 from process 0 with tag 0

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

In this example, process 0 sends an integer to process 1. Process 1 receives the integer and uses `MPI_Status` to print the source, tag, and error code of the received message.

7. MPI_Test with MPI_Isend and MPI_Irecv Example

7.1. mpi_testexample.c

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

```c
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;
    MPI_Request request;
    MPI_Status status;
    int flag = 0;

    if (rank == 0) {
        number = 100;
        MPI_Isend(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
        while (!flag) {
            MPI_Test(&request, &flag, &status);
        }
        printf("Process 0: Send complete\n");
    } else if (rank == 1) {
        MPI_Irecv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
        while (!flag) {
            MPI_Test(&request, &flag, &status);
        }
        if (flag) {
            printf("Process 1 received number %d\n", number);
        } else {
            printf("Process 1: Receive incomplete\n");
        }
    }

    MPI_Finalize();
    return 0;
}
```

7.2. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_test_example.c
```

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

- Run the program:

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

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

Process 1 received number 100
Process 0: Send complete

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

In this example, process 0 sends an integer to process 1 using `MPI$_{Isend}$`. Process 1 receives the integer using `MPI$_{Irecv}$`. Both processes use `MPI$_{Test}$` in a loop to check if the communication is complete. When the communication is complete, process 0 prints a message indicating that the send is complete, and process 1 prints the received integer.

Author: Abhishek Raj
Created: 2024-07-04 Thu 14:18