Day14

Table of Contents

1. Scripts

1.1. compile script

```sh
#!/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. Array sum

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

#define N 100
```

```c
int main() {
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int chunksize = N / size;
    int* global_arr = NULL;
    int* local_arr = (int*)malloc(chunksize * sizeof(int));

    if (rank == 0) {
        global_arr = (int*)malloc(N * sizeof(int));
        for (int i = 0; i < N; i++) {
            global_arr[i] = i + 1;
        }
    }

    MPI_Scatter(global_arr, chunksize, MPI_INT, local_arr, chunksize, MPI_INT, 0, MPI_COMM_WORLD);

    int local_sum = 0;
    for (int i = 0; i < chunksize; i++) {
        local_sum += local_arr[i];
    }

    int* global_sums = NULL;
    if (rank == 0) {
        global_sums = (int*)malloc(size * sizeof(int));
    }

    MPI_Gather(&local_sum, 1, MPI_INT, global_sums, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        int total_sum = 0;
        printf ("Array of local sums: \n");
        for (int i = 0; i < size; i++) {
            printf("%d ", global_sums[i]);
            total_sum += global_sums[i];
        }
        printf("\nTotal sum = %d\n", total_sum);
        free(global_arr);
        free(global_sums);
    }

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

```
bash compile.sh arrSum.c
```

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

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

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

Array of local sums:
55 155 255 355 455 555 655 755 855 955
Total sum = 5050

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

3. Array sum with Reduce

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

#define N 100

int main() {
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int chunksize = N / size;
    int* global_arr = NULL;
    int* local_arr = (int*)malloc(chunksize * sizeof(int));
    if (rank == 0) {
        global_arr = (int*)malloc(N * sizeof(int));
        for (int i = 0; i < N; i++) {
            global_arr[i] = i + 1;
        }
    }
    MPI_Scatter(global_arr, chunksize, MPI_INT, local_arr, chunksize, MPI_INT, 0, MPI_COMM_WORLD);
    int local_sum = 0;
    for (int i = 0; i < chunksize; i++) {
        local_sum += local_arr[i];
    }
    int final_sum;
    MPI_Reduce(&local_sum, &final_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    printf("rank = %d\ttotal sum = %d\n", rank, final_sum);
    if(rank == 0){
        free(global_arr);
    }
    free(local_arr);
    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh arrSum_reduce.c
```

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

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

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

rank = 6        total sum = 0
rank = 8        total sum = 0
rank = 9        total sum = 0
rank = 2        total sum = 0
rank = 4        total sum = 0
rank = 3        total sum = 0
rank = 5        total sum = 0
rank = 7        total sum = 0
rank = 1        total sum = 0
rank = 0        total sum = 5050

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

4. Array sum with and broadcasting the total sum to all the process

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

int main() {
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int chunksize = N / size;
    int* global_arr = NULL;
    int* local_arr = (int*)malloc(chunksize * sizeof(int));
    if (rank == 0) {
        global_arr = (int*)malloc(N * sizeof(int));
        for (int i = 0; i < N; i++) {
            global_arr[i] = i + 1;
        }
    }
    MPI_Scatter(global_arr, chunksize, MPI_INT, local_arr, chunksize, MPI_INT, 0, MPI_COMM_WORLD);
    int local_sum = 0;
    for (int i = 0; i < chunksize; i++) {
        local_sum += local_arr[i];
    }
    int final_sum;
    MPI_Reduce(&local_sum, &final_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    MPI_Bcast(&final_sum, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```c
    printf("rank = %d\ttotal sum = %d\n", rank, final_sum);
    if(rank == 0){
        free(global_arr);
    }
    free(local_arr);
    MPI_Finalize();
    return 0;
}
```

```bash
bash compile.sh arrSum_reduce1.c
```

```
------------------------------------------------------------------
Command executed: mpicc arrSum_reduce1.c -o arrSum_reduce1.out
------------------------------------------------------------------
Compilation successful. Check at arrSum_reduce1.out
------------------------------------------------------------------
```

```bash
bash run.sh ./arrSum_reduce1.out 10
```

```
------------------------------------------------------------------
Command executed: mpirun -np 10 ./arrSum_reduce1.out
------------------------------------------------------------------
##################################################################
##########                    OUTPUT                    ##########
##################################################################

rank = 1        total sum = 50005000
rank = 0        total sum = 50005000
rank = 5        total sum = 50005000
rank = 2        total sum = 50005000
rank = 4        total sum = 50005000
rank = 3        total sum = 50005000
rank = 8        total sum = 50005000
rank = 6        total sum = 50005000
rank = 7        total sum = 50005000
rank = 9        total sum = 50005000

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

5. Array sum with all reduce

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

int main() {
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int chunksize = N / size;
    int* global_arr = NULL;
    int* local_arr = (int*)malloc(chunksize * sizeof(int));
    if (rank == 0) {
        global_arr = (int*)malloc(N * sizeof(int));
        for (int i = 0; i < N; i++) {
            global_arr[i] = i + 1;
        }
    }
    MPI_Scatter(global_arr, chunksize, MPI_INT, local_arr, chunksize, MPI_INT, 0, MPI_COMM_WORLD);
    int local_sum = 0;
    for (int i = 0; i < chunksize; i++) {
        local_sum += local_arr[i];
    }
    int final_sum;
    MPI_Allreduce(&local_sum, &final_sum, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);

    printf("rank = %d\ttotal sum = %d\n", rank, final_sum);
    if(rank == 0){
        free(global_arr);
    }
    free(local_arr);
    MPI_Finalize();
    return 0;
}
```

```
bash compile.sh arrSum_allreduce.c
```

```
-----------------------------------------------------------------
Command executed: mpicc arrSum_allreduce.c -o arrSum_allreduce.out
```

```
------------------------------------------------------------------
Compilation successful. Check at arrSum_allreduce.out
------------------------------------------------------------------
```

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

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

rank = 1        total sum = 50005000
rank = 3        total sum = 50005000
rank = 7        total sum = 50005000
rank = 9        total sum = 50005000
rank = 5        total sum = 50005000
rank = 2        total sum = 50005000
rank = 6        total sum = 50005000
rank = 0        total sum = 50005000
rank = 4        total sum = 50005000
rank = 8        total sum = 50005000

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

6. allgather

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

#define N 100

int main() {
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int chunksize = N / size;
    int* global_arr;
    int* local_arr = (int*)malloc(chunksize * sizeof(int));

    if (rank == 0) {
        global_arr = (int*)malloc(N * sizeof(int));
        for (int i = 0; i < N; i++) {
            global_arr[i] = i + 1;
        }
    }

    MPI_Scatter(global_arr, chunksize, MPI_INT, local_arr, chunksize, MPI_INT, 0, MPI_COMM_WORLD);

    int local_sum = 0;
    for (int i = 0; i < chunksize; i++) {
        local_sum += local_arr[i];
    }

    int* global_sums = NULL;
    global_sums = (int*)malloc(size * sizeof(int));

    MPI_Allgather(&local_sum, 1, MPI_INT, global_sums, 1, MPI_INT, MPI_COMM_WORLD);

    for (int i = 0; i < size; i++) {
        printf("%d ", global_sums[i]);
    }
    printf("\n");
    if(rank == 0) free(global_arr);
    free(global_sums);

    free(local_arr);
    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 10
```

```
------------------------------------------------------------------
Command executed: mpirun -np 10 ./task3.out
------------------------------------------------------------------
##################################################################
##########                    OUTPUT                    ##########
##################################################################

55 155 255 355 455 555 655 755 855 955
55 155 255 355 455 555 655 755 855 955
55 155 255 355 455 555 655 755 855 955
55 155 255 355 455 555 655 755 855 955
55 155 255 355 455 555 655 755 855 955
55 155 255 355 455 555 655 755 855 955
55 155 255 355 455 555 655 755 855 955
55 155 255 355 455 555 655 755 855 955
55 155 255 355 455 555 655 755 855 955
55 155 255 355 455 555 655 755 855 955

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

7. MPI Array Sum Calculation with Timing

7.1. Introduction to MPI_Wtime

`MPI_Wtime` is a function in MPI that returns the elapsed wall-clock time in seconds since an arbitrary point in the past. It is used to measure the performance and execution time of parallel programs.

7.2. Syntax

```
double MPI_Wtime(void);
```

7.3. mpi_array_sum_timed.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 = 10000000000; // Size of the array
    long *array = NULL;
    int chunk_size = n / size;
    long *sub_array = (long*)malloc(chunk_size * sizeof(long));

    double start_time, end_time;

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

        // Start timing the computation
        start_time = MPI_Wtime();

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

    // Compute the local sum
    long local_sum = 0;
    for (int i = 0; i < chunk_size; i++) {
        local_sum += sub_array[i];
    }
    if (rank != 0) {
        // Send local sum to process 0
        MPI_Send(&local_sum, 1, MPI_LONG, 0, 0, MPI_COMM_WORLD);
    } else {
        // Process 0 receives the local sums and computes the final sum
        long final_sum = local_sum;
        long temp_sum;
        for (int i = 1; i < size; i++) {
            MPI_Recv(&temp_sum, 1, MPI_LONG, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            final_sum += temp_sum;
        }

        // Stop timing the computation
        end_time = MPI_Wtime();
        printf("The total sum of array elements is %ld\n", final_sum);
        printf("Time taken: %f seconds\n", end_time - start_time);

        free(array);
    }

    free(sub_array);

    MPI_Finalize();
    return 0;
}
```

7.4. Compilation and Execution

- Compile the program:

```
bash compile.sh mpi_array_sum_timed.c
```

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

- Run the program:

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

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


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

7.5. Explanation of Timing

- `MPI_Wtime()`: Returns the current time in seconds. It is called before and after the computation to measure the elapsed time.
- `start_time = MPI_Wtime();`: Captures the start time before distributing the array.
- `end_time = MPI_Wtime();`: Captures the end time after collecting the local sums and computing the final sum.
- `printf(“Time taken: %f seconds\n”, end_time - start_time);`: Prints the total time taken for the computation.

This updated program measures the time taken to distribute the array, compute local sums, gather the results, and compute the final sum. The timing information helps in evaluating the performance of the parallel program.

8. Array sum with timing

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


int main() {
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    const long long N = 1000000000;
    double startTime, endTime;
    long long chunksize = N / size;
    long long* global_arr = NULL;
    long long* local_arr = (long long*)malloc(chunksize * sizeof(long long));

    if (rank == 0) {
        global_arr = (long long*)malloc(N * sizeof(long long));
        for (int i = 0; i < N; i++) {
            global_arr[i] = i + 1;
        }
    }

    if(rank == 0) startTime = MPI_Wtime();
    MPI_Scatter(global_arr, chunksize, MPI_LONG_LONG_INT, local_arr, chunksize, MPI_LONG_LONG_INT, 0, MPI_COMM_WORLD);

    long long local_sum = 0;
    for (int i = 0; i < chunksize; i++) {
        local_sum += local_arr[i];
    }

    long long* global_sums = NULL;
    if (rank == 0) {
        global_sums = (long long*)malloc(size * sizeof(long long));
    }

    MPI_Gather(&local_sum, 1, MPI_LONG_LONG_INT, global_sums, 1, MPI_LONG_LONG_INT, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        long long total_sum = 0;

        printf ("Array of local sums: \n");
        for (int i = 0; i < size; i++) {
            //printf("%d ", global_sums[i]);
            total_sum += global_sums[i];
        }
        endTime = MPI_Wtime();
        printf("rank %d : timing %lf\n", rank, endTime - startTime);
        printf("\nTotal sum = %lld\n", total_sum);
        free(global_arr);
        free(global_sums);
    }

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

```
bash compile.sh arrSum1.c
```

```
------------------------------------------------------------------
Command executed: mpicc arrSum1.c -o arrSum1.out
------------------------------------------------------------------
Compilation successful. Check at arrSum1.out
------------------------------------------------------------------
```

```
bash run.sh ./arrSum1.out 10
```

```
------------------------------------------------------------------
Command executed: mpirun -np 10 ./arrSum1.out
------------------------------------------------------------------
##################################################################
##########                    OUTPUT                    ##########
##################################################################

Array of local sums:
rank 0 : timing 4.847307

Total sum = 500000000500000000

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

Author: Abhishek Raj
Created: 2025-01-03 Fri 13:59