

Computer Architecture

Lab 4

Lab 4: Parallel Programming

학번 : 2016707079 이름 : 하상천

1. Full code of MPI parallel implementation with SERIAL implementation.

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

#define ARRAYSIZE 100000000 // Array size
#define MASTER 0 // Task id of MASTER

// Initialize data array
double data1[ARRAYSIZE];
double data2[ARRAYSIZE];

// Function prototype
double compute_sum(int offset, int chunksize, int taskid);

int main(int argc, char *argv[]) {

// Initialize MPI environment
MPI_Init(&argc, &argv);

// Get the number of tasks
int numtasks;
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

// Get the task id
int taskid;
```

```

MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
printf ("MPI task %d has started...\n", taskid);

// Status for MPI_Recv
MPI_Status status;

// Compute equal data chunk size for each task
int chunksize = ARRAYSIZE / numtasks;
int leftover = ARRAYSIZE % numtasks;

// Initialize message tags
int tag1 = 1;
int tag2 = 2;
int tag3 = 3;

// Define variable for local task
int offset;

// MASTER task only
if (taskid == MASTER) {

// Initialize the data by MASTER only
for (int i = 1; i <= ARRAYSIZE i++) {
data1[i] = i * 1.0;
data2[i] = i * 1.0;
}
double max_rand = 32767; // Equal to ((2^15)-1)
for (int i = 1; i <= ARRAYSIZE i++) {
data1[i] = (double)random() / max_rand;
data2[i] = (double)random() / max_rand;
}

// Get start time to measure serial process elapsed time
clock_t t;
t = clock();

// Execute serial addtion
double serial_sum = 0.0;
double temp = 0.0;
for (int i = 0; i < ARRAYSIZE i++) {

```

```

temp = data1[i] * data2[i];
serial_sum = serial_sum + temp;
}

printf("\n*** Serial sum = %e\n\n", serial_sum);

// Get final time and compute total elapsed time of serial process
t = clock() - t;
double elapsed_ser = ((double)t)/CLOCKS_PER_SEC;

printf(">> Serial time = %f seconds\n\n", (float)elapsed_ser);
}

// Get start time to measure parallel process elapsed time
double w1 = MPI_Wtime();

// MASTER task only
if (taskid == MASTER) {

// Send each portion of the array to each task while -
// - the MASTER keeps the 1st part plus leftover
offset = chunksize + leftover;
for (int dest_id = 1; dest_id < numtasks; dest_id++) {
// Send offset value to NON-MASTER
MPI_Send(&offset, 1, MPI_INT, dest_id, tag1, MPI_COMM_WORLD);
// Send data value to NON-MASTER
MPI_Send(&data1[offset], chunksize, MPI_DOUBLE, dest_id, tag2,
MPI_COMM_WORLD);
MPI_Send(&data2[offset], chunksize, MPI_DOUBLE, dest_id, tag2,
MPI_COMM_WORLD);
offset = offset + chunksize;
}

// Do summation for MASTER
offset = 0;
double sum = compute_sum(offset, chunksize + leftover, taskid);

// Wait to receive results from all NON-MASTER

```

```

for (int source_id = 1; source_id < numtasks; source_id++) {
// Receive offset value from NON-MASTER
MPI_Recv(&offset, 1, MPI_INT, source_id, tag1, MPI_COMM_WORLD, &status);
// Receive the data of sum value from NON-MASTER
MPI_Recv(&data1[offset], 1, MPI_DOUBLE, source_id, tag3, MPI_COMM_WORLD,
&status);
MPI_Recv(&data2[offset], 1, MPI_DOUBLE, source_id, tag3, MPI_COMM_WORLD,
&status);

}

// Compute the final sum
offset = 0;
double final_sum = data1[offset];
offset = chunksize + leftover;
for (int i = 1; i < numtasks; i++) {
final_sum = final_sum + data1[offset];
offset = offset + chunksize;
}

printf("\n*** Parallel sum = %e\n\n", final_sum);
}

// NON-MASTER tasks
if (taskid > MASTER) {

// Recieve portion of the array from the MASTER
int source_id = MASTER
// Recieve offset value from MASTER
MPI_Recv(&offset, 1, MPI_INT, source_id, tag1, MPI_COMM_WORLD, &status);
// Receive data value from MASTER
MPI_Recv(&data1[offset], chunksize, MPI_DOUBLE, source_id, tag2,
MPI_COMM_WORLD, &status);
MPI_Recv(&data2[offset], chunksize, MPI_DOUBLE, source_id, tag2,
MPI_COMM_WORLD, &status);

// Do summation for NON-MASTER
double sum = compute_sum(offset, chunksize, taskid);

// Send result back to the MASTER

```

```

int dest_id = MASTER
// Send offset value to MASTER
MPI_Send(&offset, 1, MPI_INT, dest_id, tag1, MPI_COMM_WORLD);
// Send the data of sum value to MASTER
MPI_Send(&data1[offset], 1, MPI_DOUBLE, dest_id, tag3, MPI_COMM_WORLD);
MPI_Send(&data2[offset], 1, MPI_DOUBLE, dest_id, tag3, MPI_COMM_WORLD);

}

// Get final time and compute total elapsed time of parallel process
double w2 = MPI_Wtime();
    double local_time = w2 - w1;
    double elapsed;
    MPI_Reduce(&local_time, &elapsed, 1, MPI_DOUBLE, MPI_MAX, 0,
MPI_COMM_WORLD);

// MASTER task only
if (taskid == MASTER) printf(">> Parallel time = %f seconds\n", (float)elapsed);

// Finalize MPI environment
MPI_Finalize();

return 0;
}

double compute_sum(int offset, int chunksize, int taskid) {

double sum = 0.0;
double temp = 0.0;
for (int i = offset; i < offset + chunksize; i++) {
temp = data1[i] * data2[i];
sum = sum + temp;
}
// Store the sum in data
data1[offset] = sum; //data1 배열의offset 자리에저장해준다.


printf("Task %d sum = %e\n", taskid, sum);
return sum;
}


```

2. Execution output and speed-up for each number (1, 2, 4, and 8) of processes and threads.

<p># processes and threads : 1</p>	<pre>[8] !mpiexec --allow-run-as-root -n 1 ./mpi_add ❏ MPI task 0 has started... *** Serial sum = 1.073835e+17 >> Serial time = 0.357925 seconds Task 0 sum = 1.073835e+17 *** Parallel sum = 1.073835e+17 >> Parallel time = 0.361853 seconds</pre>
------------------------------------	--

<p># processes and threads : 2</p>	<pre>[9] !mpiexec --allow-run-as-root -n 2 ./mpi_add ❏ MPI task 1 has started... ❏ MPI task 0 has started... *** Serial sum = 1.073835e+17 >> Serial time = 0.425877 seconds Task 0 sum = 5.369712e+16 Task 1 sum = 5.368637e+16 *** Parallel sum = 1.073835e+17 >> Parallel time = 7.235996 seconds</pre>
------------------------------------	--

<p># processes and threads : 4</p>	<pre>[11] !mpiexec --allow-run-as-root -n 4 ./mpi_add</pre> <div>  <pre> MPI task 1 has started... MPI task 3 has started... MPI task 0 has started... MPI task 2 has started... *** Serial sum = 1.073835e+17 >> Serial time = 0.428929 seconds Task 1 sum = 2.684907e+16 Task 2 sum = 2.684431e+16 Task 3 sum = 2.684206e+16 Task 0 sum = 2.684805e+16 *** Parallel sum = 1.073835e+17 >> Parallel time = 13.941960 seconds </pre> </div>
------------------------------------	---

<p># processes and threads : 8</p>	<pre>!mpiexec --allow-run-as-root -n 8 ./mpi_add</pre> <div>  <pre> MPI task 5 has started... MPI task 6 has started... MPI task 0 has started... MPI task 1 has started... MPI task 2 has started... MPI task 3 has started... MPI task 4 has started... MPI task 7 has started... *** Serial sum = 1.073835e+17 >> Serial time = 0.434232 seconds Task 1 sum = 1.342544e+16 Task 2 sum = 1.342845e+16 Task 3 sum = 1.342062e+16 Task 4 sum = 1.342857e+16 Task 5 sum = 1.341574e+16 Task 6 sum = 1.341919e+16 Task 7 sum = 1.342287e+16 Task 0 sum = 1.342261e+16 *** Parallel sum = 1.073835e+17 >> Parallel time = 27.526600 seconds </pre> </div>
------------------------------------	--

3. Discussions regarding your implementation and the execution results.

이번 lab에서는 배열의 값을 0에서 65,535 사이의 값으로 대입하고, 두 배열의 값을 곱하고 그것을 모두 더한 결과 값을 얻었다. 우선 마스터 task가 나머지 task들에게 두 개의 배열을 send하고, 각각의 task들은 두 개의 배열을 receive하여 배열들의 값을 곱하고 그것을 모두 더한다. 더한 값을 첫 번째 배열의 offset 자리에 저장해주고, 마스터 task에 send 한다. 마스터 task는 각각의 task가 보낸 배열을 receive하여 모두 더해주고 그 값을 출력한다. processes 와 threads 수를 1, 2, 4, 8로 하여 프로그램을 실행시켜보면 Serial time은 거의 다 비슷하지만, Parallel time은 processes와 threads가 많을수록 더 오래 걸리는 것을 확인 할 수 있었다. 평범하게 생각해봤을 때, 일을 나눠서 하니깐 당연히 시간이 적게 걸릴 것이라고 예상했지만 오히려 실행시간이 더 길어졌다. 그 이유는 각각 프로세스의 send-receive 오버헤드가 실제 연산과정보다 더 많은 시간을 차지하기 때문이라고 생각한다. 따라서 이런 간단한 연산과정은 Parallel로 계산하는 것보다 Serial로 계산하는 것이 더 효과적임을 알 수 있었다.