Project Report for Parallel Programming

Chang Liu
020033910005
only-changer@sjtu.edu.cn

1 Project 1: Have a fun with parallel programming with MPI

1.1 MPI_ALLGATHER

In this sub project, we need to use MPI_SEND and MPI_RECV to implement the MPI_ALLGATHER function, and compare the performance of our implementation and the original MPI implementation.

Here, the idea of our implementation is very straightforward, that we distribute the data of each process to each process in order to make the multi processes cross-communication work. In detail, we let each process calls N-1 MPI_Send calls to send the local message buffer to other processes, and then call N-1 MPI_Recv calls to accept messages from the remaining processes correspondingly.

Then, I will show the way to run our code in Figure 1 and the results of our code in Figure 2:

```
echo "Project 1_1 (MPI_ALLGATHER) : "
mpiexec -n 1 ./p1_1 0 1000000
mpiexec -n 2 ./p1_1 0 1000000
mpiexec -n 4 ./p1_1 0 1000000
mpiexec -n 8 ./p1_1 0 1000000
mpiexec -n 1 ./p1_1 1 1000000
mpiexec -n 2 ./p1_1 1 1000000
mpiexec -n 4 ./p1_1 1 1000000
mpiexec -n 8 ./p1_1 1 1000000
```

Figure 1. How to run code for project 1.1

```
(base) changer@DESKTOP-238GEJ6:/mnt
Project 1_1 (MPI_ALLGATHER) :
Method 0 | Time (1)   0.003229
Method 0 | Time (2)   0.003530
Method 0 | Time (4)   0.004269
Method 0 | Time (8)   0.014305
Method 1 | Time (1)   0.005655
Method 1 | Time (2)   0.005630
Method 1 | Time (4)   0.019544
Method 1 | Time (8)   0.067096
```

Figure 2. The results of code for project 1.1

There are two arguments in our code: the first one is type, where “0" denoting the original MPI_ALLGATHER and “1" denoting our self-developed MPI_ALLGATHER; The second one is the size of the array for testing performance. Here we show the results with “1, 2, 4, 8". We can see the performance boost is not clear in both types, it may because I use my PC and its multi-process performance is not good.

1.2 Gemm

In this sub project, we need to parallelize the program in the tile unit with the following three operations : 1. Initialize two random 1024 *1024 matrices, and implement the matrix multiplication function. 2. Using a 4 * 4 kernel to do the pooling operation for the matrix. 3. Using a 4 * 4 kernel to do the convolution operation for the matrix.

Here, I use the idea in the slides, that divide the matrix in to tiles, and one tile for each processor. Then, we can calculate the data inside each tile in parallel.

Then, I will show the way to run our code in Figure 3 and the results of our code in Figure 4:

```
echo "Project 1_2 (Gemm) : "
mpiexec -n 1 ./p1_2 0
mpiexec -n 4 ./p1_2 0
mpiexec -n 1 ./p1_2 1
mpiexec -n 4 ./p1_2 1
mpiexec -n 1 ./p1_2 2
mpiexec -n 4 ./p1_2 2
```

Figure 3. How to run code for project 1.2

```
(base) changer@DESKTOP-238GEJ6:/mnt/c/Users/chang
Project 1_2 (Gemm) :
Result of Matrix Multiply is 1073741824
Method 0 | Time (1)   0.853664
Result of Matrix Multiply is 1073741824
Method 0 | Time (4)   0.854091
Result of Conv is 2097152
Method 1 | Time (1)   0.009024
Result of Conv is 2097152
Method 1 | Time (4)   0.009258
Result of Pooling is 1048576
Method 2 | Time (1)   0.009020
Result of Pooling is 1048576
Method 2 | Time (4)   0.010064
```

Figure 4. The results of code for project 1.2

There are one arguments in our code: that is the type, where “0" denoting the matrix multiply, “1" denoting the convolution operation, and “2" denoting the pooling operation. Here we show the results with “1, 4". We can see the performance boost is not clear in both types, it may because I use my PC and its multi-process performance is not good.

1.3 Word Count

In this sub project, we need to implement the word count algorithm respectively for the small data and big data situations and print the results.

My idea of this project is: first, I scan all files and maintain a dictionary that contains all words that have appeared in the files. For the small file situation, each process will be assigned with some files, and each process can acquire the whole dictionary. It only need to count the appearance of every word in the dictionary and MPI_REDUCE the count. For the big file situation, each process will only acquire a part of the dictionary, and scan the whole file, count the appearance of every word in the part of dictionary and MPI_REDUCE the count.

Then, I will show the way to run our code in Figure 5 and the results of our code in Figure 6:

```
echo "Project 1_3 (Wordcount) : "
mpiexec -n 1 ./p1_3 0
mpiexec -n 2 ./p1_3 0
mpiexec -n 4 ./p1_3 0
mpiexec -n 8 ./p1_3 0
mpiexec -n 1 ./p1_3 1
mpiexec -n 2 ./p1_3 1
mpiexec -n 4 ./p1_3 1
mpiexec -n 8 ./p1_3 1
```

Figure 5. How to run code for project 1.3

```
(base) changer@DESKTOP-238GEJ6:/mnt/c/Users/changer/Desktop/work/courses/PP/project/project1$ ./run.sh
Project 1_3 (Wordcount) :
the : 176438
of : 140409
and : 98035
ref : 81143
in : 71340
Method 0 | Time (1)   4.358656
the : 176438
of : 140409
and : 98035
ref : 81143
in : 71340
Method 0 | Time (2)   2.034608
the : 176438
of : 140409
and : 98035
ref : 81143
in : 71340
Method 0 | Time (4)   1.176283
the : 176438
of : 140409
and : 98035
ref : 81143
in : 71340
Method 1 | Time (1)   2.528409
the : 176438
of : 140409
and : 98035
ref : 81143
in : 71340
Method 1 | Time (2)   2.304609
the : 176438
of : 140409
and : 98035
ref : 81143
in : 71340
Method 1 | Time (4)   2.154736
```

Figure 6. The results of code for project 1.3

There are one arguments in our code: that is the type, where "0" denoting the small file situation, "1" denoting the big file situation. Here we show the results with "1, 2, 4, 8". We can see the performance boost is very clear in both situations.

2 Project 2: Have a fun with parallel programming with OPENMP

2.1 Monte Carlo Algorithm

In this sub project, we need to use OpenMP to implement the Monte Carlo algorithm.

The idea of this problem is quite straight forward: we divide the simulation into several processors. Every processor finish its random simulation and then calculate the pi. Note that the function rand() can not be done in parallel, so I choose erand48() in my implementation.

Then, I will show the way to run our code in Figure 7 and the results of our code in Figure 8:

```
echo "Project 2_1 (Monte Carlo) : "
./p2_1 1 10000000
./p2_1 2 10000000
./p2_1 4 10000000
./p2_1 8 10000000
```

Figure 7. How to run code for project 2.1

```
(base) changer@DESKTOP-238GEJ6:/mnt/c/Use
Project 2_1 (Monte Carlo) :
Result pi :   3.141898
Time (1)   0.156868
Result pi :   3.141312
Time (2)   0.076512
Result pi :   3.142064
Time (4)   0.040691
Result pi :   3.141389
Time (8)   0.029940
```

Figure 8. The results of code for project 2.1

There are two arguments in our code: the first one is the number of processors; The second one is the size of simulation. We show the results with the processors number "1, 2, 4, 8". We can see the performance boost is very clear.

2.2 Quick Sort

In this sub project, we need to use OpenMP to implement a quick sorting algorithm with large data volume which contains 1000000 number.

The idea of this problem is a little different. Inspired by the slides, we can run the divide and conquer step in quick sort in parallel. Quick sort will divide the array to two parts by the pivot, then, we can use two processor the run each part in parallel.

Then, I will show the way to run our code in Figure 9 and the results of our code in Figure 10:

```
echo "Project 2_2 (Quick Sort) : "
./p2_2 1 1000000
./p2_2 2 1000000
```

Figure 9. How to run code for project 2.2

There are two arguments in our code: the first one is the number of processors; The second one is the size of array for quick sort. We show the results with the processors number "1, 2". We can see the performance boost is very clear.

```
(base) changer@DESKTOP-238GEJ6:/mnt/c/
Project 2_2 (Quick Sort) :
Time (1)   0.160597
Time (2)   0.082275
```

Figure 10. The results of code for project 2.2

2.3 PageRank

In this sub project, we need to initialize a graph has 1, 024, 000 nodes and the edge count of different nodes ranges from 1 to 10. Implement the PageRank algorithm and run for 100 iterations.

We implement the PageRank algorithm following the equation on the slides. The iterations of PageRank can not be paralleled, but the update steps inside one iteration can. In every iteration, we need to update all nodes in the graph, of which can be done individually, in other words, in parallel.

Then, I will show the way to run our code in Figure 11 and the results of our code in Figure 12:

```
echo "Project 2_3 (PageRank) : "
./p2_3 1 1000
./p2_3 2 1000
./p2_3 4 1000
./p2_3 8 1000
```

Figure 11. How to run code for project 2.3

```
(base) changer@DESKTOP-238GEJ6:/mnt/c/Users/chang
Project 2_3 (PageRank) :
Result gap : 0.000000
Time (1)   1.313420
Result gap : 0.000000
Time (2)   1.124502
Result gap : 0.000000
Time (4)   0.800723
Result gap : 0.000000
Time (8)   0.674763
```

Figure 12. The results of code for project 2.3

There are two arguments in our code: the first one is the number of processors; The second one is the number of iterations. We show the results with the processors number "1, 2, 4, 8". We can see the performance boost is very clear.

3 Big Data Analysis in Hadoop System

In this project, we need to first implement weather data program using MapReduce, and then use the implemented weather data program to get the statistics of the highest and lowest temperature for all the files in the data directory, respectively.

I set up the Hadoop system and combine it with the editor Intellij. I make a suitable development environment with JAVA and Hadoop system, as shown in Figure 13.

We follow the idea in the official example code Word Count. That is, we define the input and output of our program in the main class ' MaxTemperature", and set the format of

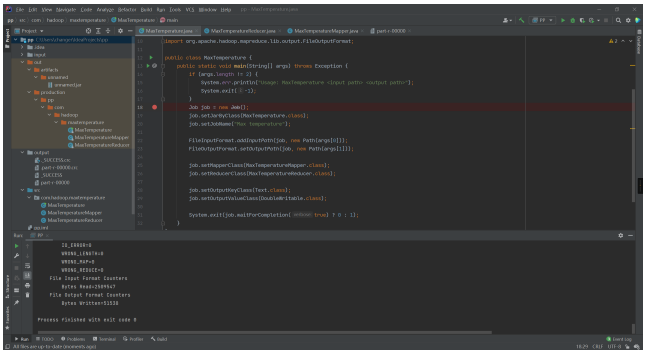

Figure 13. The develop environment and running console for project 3

them respectively. Then, we read all files in the input, and use the class "MaxTemperatureMapper" the process the string to get the temperature in every line. Then, we define the class "MaxTemperatureReducer" to define the rule for reduce, such as calculate the max number of all returned temperature by the class "MaxTemperatureMapper". Please note that we preprocess the data in R format to csv format, just for the convenience for JAVA to process it.

Then, I will show the results of our code in Figure 14. It can also be find under the "output" folder.

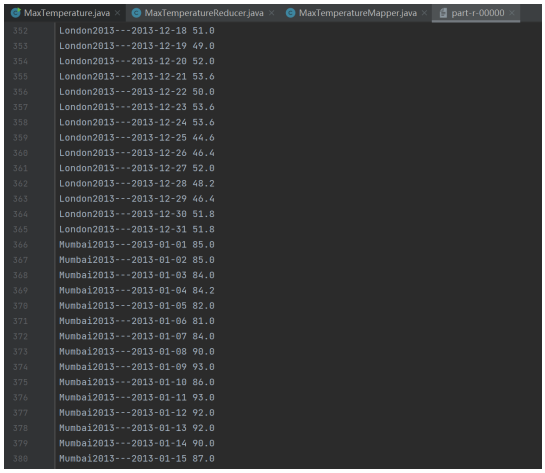

Figure 14. The results for project 3

The results of our code is a file that contains the max temperature of every day in every file. For every line, the part before "- - -" is the file name, and continues by the day and the max temperature of that day. Instead of submitting the "run.sh", I directly submit the ".idea" folder generated by Intellij, you can open my code folder in Intellij, change the location of Hadoop folder into yours, and can run my code normally.

That's all, thank you!