Parallel Programming Final Project

Zixuan Chen 120033910005
m13953842591@sjtu.edu.cn

ACM Reference Format:
Zixuan Chen 120033910005. 2018. Parallel Programming Final Project. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/1122445.1122456

1 PROJECT1

1.1 MPI_Allgather

Use MPI_Send and MPI_Recv to implement the MPI_ALLGATHER function. For each process, it first send it data to all of the rest process using MPI_Send. For simplicity, each process send to itself by **memcpy**. Then each process recive data from other processes by MPI_Recv. We test the algorithm by generating send_buffer and recv_buffer of length 10000, and number of process is set to 4.

```
==================================================
Using my_MPI_Allgather:
Total elapsed time:   0.000029
Using MPI_Allgather:
Total elapsed time:   0.000010
==================================================
```

1.2 Gemm

The Matrix is required to be parallizaed in the tile unit. Which is cannon algorithm. Cannon's algorithm is a distributed algorithm for matrix multiplication for two-dimensional meshes. It is especially suitable for computers laid out in an N × N mesh. Consider two n × n matrices A and B partitioned into p blocks.$A(i, j)$ and $B(i, j)$ $(0i, jp)$ of size$(n/\sqrt{p})(n/\sqrt{p})$each. The initial step of the algorithm regards the alignment of the matrixes. Align the blocks of A and B in such a way that each process can independently start multiplying its local submatrices. This is done by shifting all submatrices $A(i, j)$ to the left (with wraparound) by i steps and all submatrices $B(i, j)$ up (with wraparound) by j steps.Perform local block multiplication.Each block of A moves one step left and each block of B moves one step up (again with wraparound).Perform next block multiplication, add to partial result, repeat until all blocks have been multiplied.

In our experiment, we randomly generated 1024x1024 matrix and test the time elapsed during multiplication.

```
==================================================
Large matrix multiplication
random generate 1024x1024 matrix...
Multiplication size: (1024, 1024) x (1024, 1024).
Total elapsed time: 17.719502
==================================================
Convolution operation Using a 4 * 4 kernel to do the
pooling operation for the matrix.
Matrix size: (10214, 1024), Kernel size (4, 4).
Total elapsed time: 0.121011
==================================================
Pooling operation Using a 4 * 4 kernel to do the
convolution operation for the matrix
Matrix size: (1024, 1024), kernel size (4, 4).
Total elapsed time: 0.107230
==================================================
```

1.3 Wordcount

In this problem we use map<string, int> as a dict to map a word to its frequency. We define the word as a sequence which is made up of only english letters(lowercase or uppercase). The common function of this two algorithms is performing wordcounting, given the file reader and the block size to read. For this function, we extract all the suitable word and add the lowercase of the word to the dict. The big file problem deals with a single file of size 820kb, while the small file problem deals with 100 file of size ranging from 0kb to 400 kb.

1.3.1 big file. For big file, we divide the whole data block into **p** pieces(p is the total number of process), and assign each process to one of its block. After getting the word frequency dictionary of all processes, we need to merge all the dictionaries into a final dictionary. Each process first send the size of the dictionary, then for each word, it first send the length of the word, then send the word and the frequency. The reason to send the size of dictionary and the length of the word is MPI_Recv need the block size before it recive the actual word. Finally, we sort the final dictionary to get the few most frequent words.

The output is shown as follows.

```
wordcount big file
the most frequent words top 20
<Word>      <Freq>
==================================================
the         8233
of          3895
and         3159
to          2752
that        2456
a           2312
in          2298
he          2005
was         1857
had         1694
she         1638
```


Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

```
her          1619
with         1467
his          1327
it            876
on            868
him           836
not           825
for           770
they          723
=====================================================
Total elapsed time:   2.327309
=====================================================
```

1.3.2 small file. The idea of the small file wordcount is same as the big file wordcount, except that it distribute the small files to all the process. For each process, it perform wordcount of each file and merge all frequencies locally. Then it merge all the dictionaries into a final dictionary.

The result is shown as follows

```
wordcount small files
the most frequent words top 20
<Word>      <Freq>
=====================================================
the           485
is            218
to            208
of            156
a             150
for           134
as            122
file          117
in            116
and           114
you           110
jpeg           99
if             97
are            93
this           82
image          79
or             71
that           71
use            70
be             68
=====================================================
Total elapsed time:   0.117455
=====================================================
```

2 PROJECT2

2.1 Monte Carlo

The idea of Monte Carlo is randomly throw a niddle into a $1x1$ square area several times(denoted as n). and count times it fall into the circle area(r=1) inside the square area(denoted as m). and compute π by

$$\pi_{estimate} = \frac{m}{n}$$

We just parallelize all the individual trials. The result of our experiment is shown as follows

```
=====================================================
Exercise 1: Monte Carlo
=====================================================
total trial time=10000000
pi = 3.14145
=====================================================
```

2.2 Quick Sort

The idea of quick sort is parallel the quick sort in recursive form

In our experiment we generate a big array of size 100000, and we assign random value each of it element. after quicksort we check its correctness and extimate the time elapsed.

```
=====================================================
allocting a array of length 1000000...
Total elapsed time:   0.046374
checking quick sort result..
=====================================================
```

2.3 Page Rank

For Pagerank algorithm we need to implement two classes, Graph and SubGraph. Graph class contain all the nodes of a whole graph. SubGraph class contain partial nodes of the whole graph. Beside, it contain the map from global index of father graph to local index in the subgraph. The general idea of pagerank is to first divide it into several subgraphs and then parallelize each iteration. The main body of the code is shown as follows.

```
for (int t = 0; t < iterations; t++){
    cout << "iter :  " << t << " running" << endl;
    // #pragma omp parallel
    {
        #pragma omp for
        for (int tid = 0; tid < threads; tid++){
            // do pagerank on the local sub-Graph
            sgs[tid].updatePageRankLocal();
        }
        syncGraph(sgs, threads);
    }

    graphMerge(g, threads, sgs);
}
```

2.3.1 graphPartition. To make sure each sub graph have as most inside link as possible. We first count degree of each node, choose few nodes with most degree as root, and perform DFS to get the rest of subgraph.

2.3.2 updatePageRankLocal. Since for each subgraph, there exist edge linked to other subgraph. we only perform pagerank locally, which means we don't accept pagerank from outside the subgraph.

2.3.3 syncGraph. At the end of each iteration all subgraph must be synchronized to get the real pagerank result.

2.3.4 graphMerge. At the end of program, we need to merge all subgraphs into the orignal graph.

The number of node is set as 1024000, each node have degree ranging from 1 to 10. The threads is set as 100. we first check the correctness of our algorithm by set the initial pagerank of each

node as 1. If it is correct the pagerank of each node would stay unchanged(Assume that the edges in the graph are bi-directional).

the result is shown as follows

```
=====================================================
Initiallzing graph of node=1024000...
generate edge count of different nodes ranges from 1 to 10.
executing page rank
iter :  0 running
iter :  1 running
iter :  2 running
iter :  3 running
iter :  4 running
Total elapsed time:  77.663224s
=====================================================
```

The elapsed time is too long for few iterations..

3 PROJECT3

3.1 Build Hadoop distributed system

First, you need to prepare four virtual machines named master, slave1, slave2, and slave3. The network of four virtual machines is configured as bridge mode, and static IP is allocated.

```
192.168.3.77  master
192.168.3.74 slave1
192.168.3.76 slave2
192.168.3.78 slave3
```

The four machines then exchange the SSH public key with each other, allowing them to ping each other.Then install Java8-OpenJKD and Hadoop and add them to the environment variables.Finally, configure the Hadoop configuration files, including core-site.xml, hdfs-site.xml, yarn-site.xml, mapred-site.xml. Enter command "jps" test the system

```
hadoop@master:~/hadoop-3.3.1/etc/hadoop$ jps
4288 Jps
3009 NameNode
3923 NodeManager
3736 ResourceManager
3422 SecondaryNameNode
3198 DataNode
```

The configuration was successful.

3.2 Mapper

The Mapper file combine the input file name with every piece of temperature data in that file

```
<filename, temperature>
```

3.3 Reducer

The Reducer reduce to the max temperature among data with same filename.