Parallel Dijkstra's algorithm using OpenMP and CUDA

0756518 王致翔,
Institute of Computer Science and Engineering , NCTU

0756089 陳昱翔,
Institute of Computer Science and Engineering , NCTU

0756085 曹宏彰,
Institute of Computer Science and Engineering , NCTU

ABSTRACT Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph. In small cases, we can easily use it to calculate the shortest paths in few microseconds . However, if there are tens of thousands of nodes, it would take us seconds to find the answer. And this may become the bottleneck when we applied Dijkstra's algorithm in other project. So, we tried to use OpenMP and CUDA in order to parallelize the parts of selection node and update table in this algorithm. Overall, we have 360%~380% speedup using OpenMP (with 4 thread), and ? speedup using CUDA.

1 INTRODUCTION

In recent years, with new technological advances, the accurate and timely logistics and distribution system plays a more and more significant role in this rapidly developing society. Therefore, finding the shortest paths becomes a non-negligible problem. Although the Dijkstra's algorithm could help us to quickly find the shortest path, it still need seconds to calculate if there are billions of edges. It would become a major obstacle when serving the logistics service or other applications which needs to find the shortest paths. Hence we tried to applied what we learn in the parallel programming course to solve this problem.

2 IMPLEMENTATION

Dijkstra's algorithm could be divided into two part, selection and update. The selection part is to choose the node which is closest to the current node as the new current node. When we get the new current node, we use the update part to update the nearby nodes's distance table and previous node table. And then loop it until finished the whole table.

Since the Dijkstra's algorithm took most of time to select node and update table, we decided to parallel these two part.

2.1 OpenMP

1. Selection Serial version of selection is using a for loop to update the variable of closest distance and closest node, and output this node in the end. However, when using the parallel for loop of openmp, each thread would maintain its own variable. So, in the paralleled version, we need to compare with all threads' output and then find the closest one as the final result.
2. *Update Since there is no dependency in the for loop of update part, we could just use openmp to paralleled update these two table.*

2.2 CUDA

1. Memory preprocess:
Allocation memory and coping graph to cuda memory for Selection and Update.
2. Selection:
All node were cut into 300 block, selecting minimum distance in parallel, then the global minimum was choose in these local minimum.
3. Update:
With 1024 thread in each block, each thread update corresponding value in distance, visited list.

3 EXPERIMENT

3.1 environment

CPU: i7-8700K
GPU: RTX-2070
RAM: 16GB DDR4-2400
DISK: 256GB SATA3 SSD
OS: ubuntu 16.04

3.2 test data

We wrote another program to generate the graph file, it could automatically generate a certain nodes and edges with random value. We generated two kinds of cases, fully connected and non-fully connected, and the numbers of nodes are from 10000 to 60000.

4 RESULT

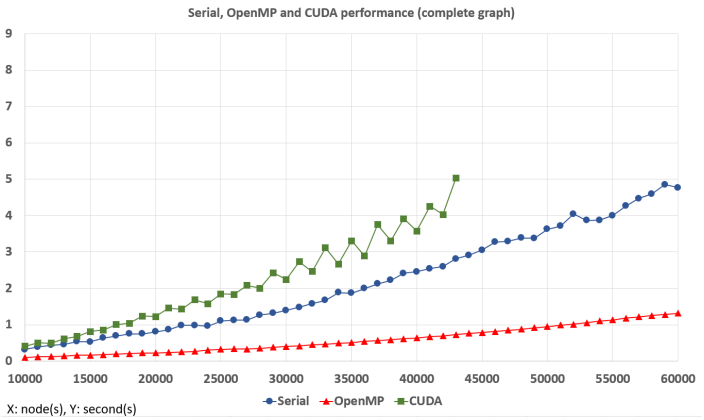

Fig.1 Serial, OpenMP and CUDA performance (complete graph)

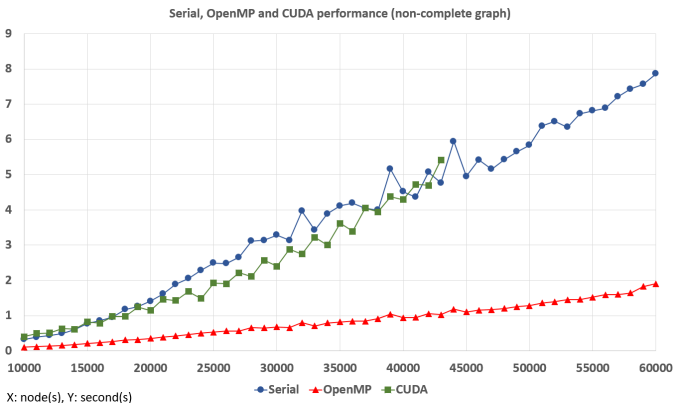

Fig.2 Serial, OpenMP and CUDA performance (non-complete graph)

With 4 thread OpenMP, we could have 360%~380% speedup. With CUDA, Since the edge table is too large, the overhead when doing CUDA memory preprocessing would be high. Another overhead is calling cuda kernel. If there are n nodes, it means overhead would be 3*n times of overhead. Since the time cost of each iteration is not much higher than the overhead, the effect of overhead would slow down the process.

5 CONCLUSIONS

To Dijkstra's algorithm, the OpenMP is a better choice than CUDA to parallel the calculation. We can find that when the data which is needed to copy to coda memory is large, it may not be suitable for coda. Another point is that the execution time of each iteration is needed much higher than cuda overhead.

6 CODE

https://github.com/chwang1996/Dijkstra_parallel

7 REFERENCE

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

https://github.com/AlexDWong/dijkstra-CUDA