Forbidden Triad Hypothesis on Social Network

Yu Liang, Jiabo Liang

Abstract

This paper is to show a method to verify the Forbidden Triad Hypothesis in Social Network. The dataset is from YouTube social network. (<https://snap.stanford.edu/data/com-Youtube.html>) We also try to find a best way that smallest time of cost.

Background

Method

1. input data

The data display as the following figure 1 in the .txt file. Each row represents one edge in the social networks. The number is the node’s ID. We can identify the node by this ID.

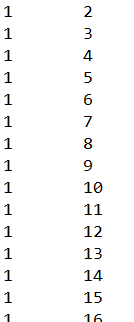


figure 1.

2. compute

To verify the Forbidden Triad, we need to know if Node 1 is tied with Node 2 and Node 3, is there also a tie between Node 2 and Node 3? So, our mean idea is to compute the number of this closed triangle and unclosed triangle in figure 2. Then we can get the Fraction of closed triangles. We will know if a tie between Node 2 and Node 3.

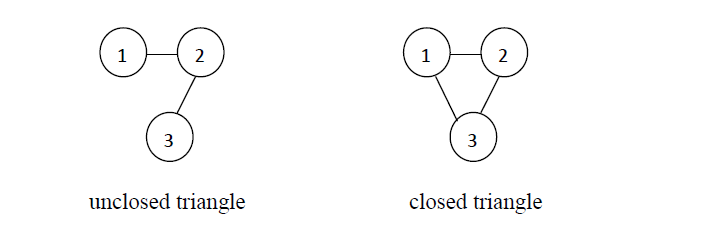


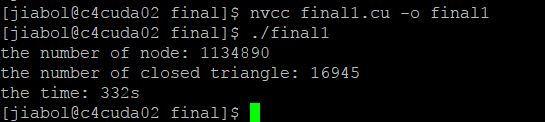
Figure 2.

First, we read each edge from the file and store them in the linkedlist to get the number of node that is linked by main node. Then we store it in one-dimension array. And use another array as the index array to store every main nodes’ starting position.

We compute both in CUDA and multithreading in CPU these two methods in order to get the smallest cost of time.

Result and Analysis

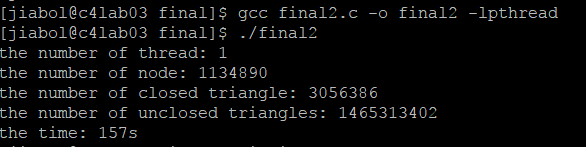
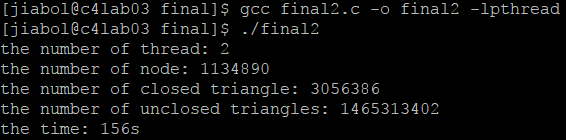
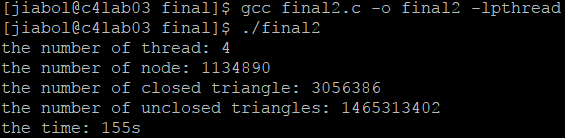
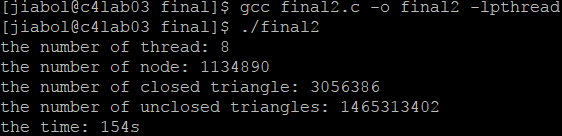
First, we compute in CUDA. But after 20 minutes the program is still running. We check code many times and make sure the code is right. Then we try to only compute one node instead of all node (1134890). The program is immediately finished. Then I compute five nodes, and find it still need over 300 seconds time (Figure 3).

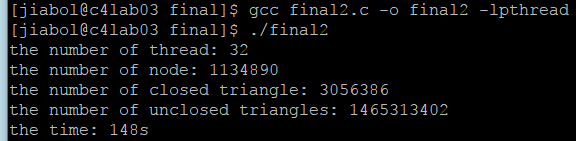
Figure 3.

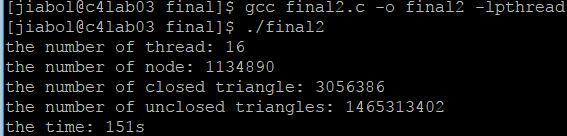
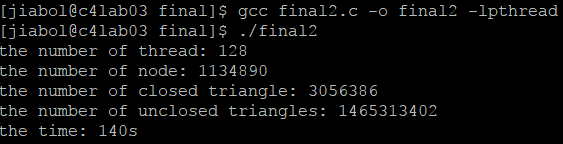
So, we are thinking the CUDA’s property. The CUDA is efficient in the large repeated computations. But in this algorithm, we don’t have much repeated computation. In this program, it need search the array again and again. The array’s size is over one million. So, we guess it’s one reason that influences the GPU’s performance.

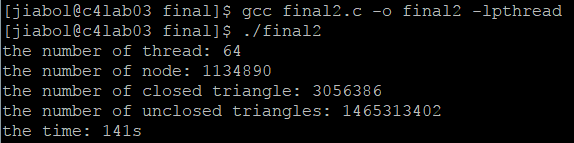
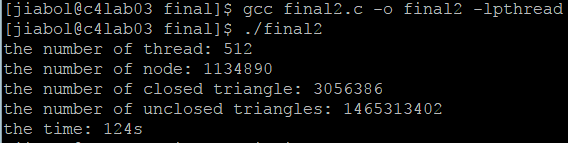
Then, we rewrite the code to run on CPU in multithreading. We find it’s really fast. It only spends around 2 to 3minutes. The consequence is in the following table.

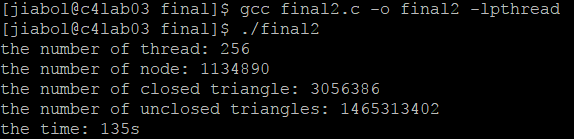
|  |  |
| --- | --- |
| thread number | cost time |
| 1 | 157s |
| 2 | 156s |
| 4 | 155s |
| 8 | 154s |
| 16 | 151s |
| 32 | 148s |
| 64 | 141s |
| 128 | 140s |
| 256 | 135s |
| 512 | 124s |
| 1024 | 115s |

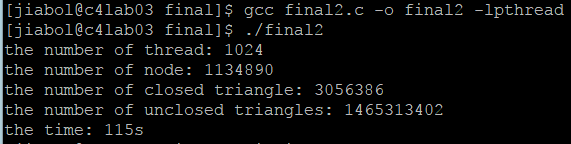
 the number of thread:1 the number of thread:2

 the number of thread:4 the number of thread:8

 the number of thread:16 the number of thread:32

 the number of thread:64 the number of thread:128

 the number of thread:256 the number of thread:512



the number of thread:1024

From the table, we can find the cost time decreases with the increase of the number of thread. Multithreading is really increase the performance. Finally, we get the number of closed triangle: 3056386. The number of unclosed triangle is 1465313402. Then the fraction of closed triangles is . So, we can find the possibility that there is a tie between Node 2 and Node 3 is only 0.2 percent.

What we learn

We find that not all dataset or algorithm is applicable to compute on GPU. Sometime, the CPU is more efficient. So many large platforms use the mixed CPU-GPU architecture to get the highest performance.