Kth Clique detection

Jayant Anand

Bachelors of Technology in Electronics and Communication Enginnering
Indian Institute of Technology, Dhanbad
Dhanbad, India

Jayantanand2001@gmail.com

***Abstract* – An algorithm for counting the K-th clique in an undirected graph with the help of parallel processing has been discussed here.**

I. Introduction

A clique is a subset of vertices of an undirected graph G such that every two distinct vertices in the clique are adjacent; that is, its induced subgraph is complete. Cliques are one of the basic concepts of graph theory and are used in many other mathematical problems and constructions on graphs.
The task of finding whether there is a clique of a given size in a graph (the clique problem) is NP-complete, but despite this hardness result, many algorithms for finding cliques have been studied.
Finding the number of cliques can be solved easily by parallel processing.

II. APPROACH

A. Concept

The kth clique can be found from a k-1th clique by adding one such node to it that is connected to all the nodes in that k-1th clique. In unidirectional graph any pair of node connected via an edge is a 2-clique.
Thus kth clique can be found from 2-cliques.
For each k-1th clique all the nodes will be iterated testing that whether or not its connected to all the elements in the k-1 th clique, thus finding the kth clique.
This will be done in parallel for each k-1th clique.

B. Finding unique cliques

While finding the K-th clique it would be required to constantly monitor that the found cliques are unique. For verifying the uniqueness, it may require a time complexity of O(k*n^2) where n is the number of cliques.
But instead of doing so the duplicity can be avoided by:

1. Consider all the k-1th cliques that are found are unique.
2. To construct a kth clique from a particular k-1th clique only those nodes with a value greater than the values of all nodes in that k-1th clique will be considered for testing.

C. Improvement

Finding the k-th clique from k-1th clique will have a complexity of O(m). This complexity can be reduced by minimizing number of nodes tested.

To do so let's consider a k-1-th clique
a1 a2 a3 .. ak-1
We need to find the node "c" such that most of the elements in the clique "a" are connected to c,
It can be done by considering only those nodes "c" such that there is another k-1th clique "b"
b1 b2 b3 .. bk-1 such that
a2=b1 , a3=b2 …. ak = bk-1 and a1 is connected to node "c".
Proof of validity: If "c" node has to be a part of the kth node a1 a2 … ak c then the sub graph a2 a3 .. ak c is connected which makes is a k-1th clique.

This will reduce the complexity to O((M)/N) where M is the number of k-1th clique and N is the number of nodes.

III. EXPERIMENTAL RESULTS

While testing on youtube graph data (38.7mb)

K	Result	Execution Time
3	3056386	13608 ms
4	4986965	17516ms
5	7211947	18543ms
6	8443803	18635ms
7	7959704	18924ms

While testing on As-skitter graph data (149.1mb)

K	Result	Execution Time
3	28769868	39717ms
4	148834439	54042ms
5	1183885507	98051ms

IV. CONCLUSION

The algorithm attains a time complexity of O(k*M/N)
Where k is clique size, M is the number of k-1th cliques and N is the number of nodes.

The space complexity was found to be O(k*M) where k is the clique size and M is the number of k-1th cliques.
Since M can be as large as NC(k-1).
where nCm means number of ways of selecting m objects out of n objects.
Time complexity: O(k*NC(k-1)/N)
Space complexity: O(k*NC(k-1))

V. REFERENCES

[1] J. G. Augustson and J. Minker, "An analysis of some graph theoretical cluster techniques,"*J. ACM* **17**(4):571–588 (1970).

[2] A. R. Bednarek and O. E. Taulbee, "On maximal chains,"*Rev. Roum. Math. Pures et Appl.* **XI**(1):23–25 (1966).

Appendix:

1. Link of the source code:
   https://github.com/jayant-ism/hipc/blob/master/new_model.cu
2. Required GCC :
   gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-28)
   Required NVCC:
   Cuda compilation tools, release 11.4, V11.4.48
   Build cuda_11.4.r11.4/compiler.30033411_0
   C++11
3. Instructions:
   For running on clusters install slrum and nvidia hpc toolkit
   For running on jupyter notebook follow the link.
   https://www.geeksforgeeks.org/how-to-run-cuda-c-c-on-jupyter-notebook-in-google-colaboratory/
4. For compiling use :
   nvcc new_model.cu –std=c++11
5. For running use
   Find the partition with gpu access using “sinfo”
   Replace the “gpu” with the partition name in the below command to run the compiled file
   srun –partition=“gpu” ./a.out