A High-Performance Efficient Implementation of K-Cliques Using GPUs

Jayesh Patidar
jaysptdr20@iitk.ac.in

Parth Sharma
prthshrma20@iitk.ac.in

***Abstract*–Counting k-cliques in a graph has applications in several domains and is an important problem in graph analysis. Finding all k-cliques in a sequential manner is expensive as it requires traversing the graph from each node. We present an efficient way of counting k-cliques on GPU which parallely traverse the graph from each node.**

Our evaluation shows that our GPU implementation performs better than the sequential CPU implementation. We have experimented with our implementation on k = 3, 4, 5 parameters.

1. Introduction

A k-clique (or a k-vertex clique) in a graph is a complete sub graph with exactly k vertices and k x (k-1) edges, such that every vertex in the clique is connected to every other vertex. A k-clique has great importance in many applications such as community detection, graph partition and compression, spam and anomaly detection, social network analysis, and the discovery of patterns in biological networks. Since,the real world graphs are very large in size and developing analysis algorithms for such graphs are challenging. Even designing sequential algorithms in an efficient way is not enough.

A common approach to counting k-cliques is to traverse the graph from each vertex and count the number of k-cliques found. To avoid the redundant cliques, while traversing from each node, we have considered only successive nodes from that node. We have evaluated our parallel implementation on GPU by comparing it with the sequential implementation on CPU of the same algorithm and we have got decent speedup also.

2. Approach

We have used a recursive state-of-the-art algorithm [1]. For the GPU implementation, we have come up with the iterative version of the algorithm. In our GPU implementation, each thread starts traversing the graph from a particular node in parallel. For example, thread with id 0 starts traversing from node 0, thread with id 1 starts from node 1 and so on. These threads work in parallel and update the counter with proper synchronization.

3. Experimental results

We experimented on three different values of parameter k i.e. 3,4, and 5 .We have used a dataset of ca-GrQc given in the datasets[2] provided for our experiment. To compare our GPU implementation, we gathered data by executing our sequential iterative version on CPU. We got the following results on different values of k.

k-clique	CPU time	GPU time	Speed-up
3	10982.1	220.54	49.79
4	26637.1	1477.28	18.03
5	118080	13051.6	9.04

Table 1: Timing Stats(*in miliseconds*)

k-clique	No. of cliques
3	48260
4	329297
5	2215500

Table 2: No. of k-Cliques

3. Conclusion

We developed a parallel algorithm for listing all k-cliques in very large real-world graphs. Our parallel GPU implementation shows speedup in comparison to its sequential CPU implementation.

4. References

[1] https://www.geeksforgeeks.org/find-all-cliques-of-size-k-in-an-undirected-graph/

[2] https://graphchallenge.s3.amazonaws.com/snap/ca-GrQc/ca-GrQc_adj.tsv

[3] https://arxiv.org/pdf/2104.13209.pdf

[4] http://www.dcs.gla.ac.uk/ pat/jchoco/clique/enumeration/papers/bronl

[5] https://forums.developer.nvidia.com/t/implementing-stack-in-cuda/12401/6

[6] https://docs.nvidia.com/cuda/thrust/index.html

Appendix

* **Git Repo** : `https://github.com/prthshrma/HiPC_Y21_IITK.git`

* **A-100 GPU specifications :** All experiments were performed on the A100 GPU provided by the HiPC team.
    nvcc version : Cuda compilation tools, release 11.0, V11.0.221
    gcc version : gcc (Ubuntu 9.3.0-17ubuntu1 20.04) 9.3.0

* CPU configurations (on which sequential code was executed) :
    Architecture : x86_64
    CPU(s) : 8
    Model Name : Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz

* **Compilation command** : nvcc -arch=sm_80 kClique.cu -o kClique

* **Execution command :** ./kClique

Source code takes 2 inputs as given in the problem statement, first path to the data file and second, parameter k.

Note: Provided source code works on maximum 10^4 *nodes, so increase in number of nodes may result in error or wrong result.*