Finding Maximum Clique in Hybrid System

Rintu Panja*, Ponnezhil Dass M P*
*Department of Computational and Data Sciences
Indian Institute of Science, Bangalore, India
{rintu,ponnezhil}@grads.cds.iisc.ac.in

***Abstract*—Finding Maximum Clique in a graph is an NP-Complete problem. So, when the size of the input graph increases the time to solve the problem increases exponentially. We present a new hybrid approach (using multi-core CPU and GPU) to find a upper and lower bound of the size of the clique. Our algorithm finds these bounds within few seconds for real world large graphs. These bounds helps us to prune out more than 90% of the vertices of these real world graphs. In final step we find out the exact clique recursively searching over all possible cases using open-mp threads and successively pruning out vertices that already has been searched. The pruning at last step help us to further reduce the search space.**

I. Introduction

In a given graph, a clique is a sub-graph in which all the vertices are connected to each other. Given a graph $G = (V, E)$ with $|V|$ vertices and $|E|$ edges we are interested in finding maximum of all such sub-graphs. The naive algorithm to find maximum clique checks all possibilities by including all vertices one by one if it can be a part of the maximum clique or not. So, if the graph has n vertices it checks for all 2^n such possibilities. This puts the problem into the interesting category of NP-hard problem.

The Maximum Clique problem has huge application in Social Networking, Biological Networks, Retweet Networks, Wikipedia Links, Web Links and Collaborative Networks. In Social Network we are often interested in largest group of people who are connected to each other. In Networking we are interested in largest part of the network where each part is connected to each other or in in Retweet network largest group of people who tweet to each other.

We have implanted a new hybrid approach to find a heuristic clique of any given graph. Low cost high performance GPU have already caught the attention of many researches. Its thousands of cores gives massive parallelism than CPU. Now many commodity clusters contains a high end GPU device. If we can utilize this GPU device along with CPU then it will help us to solve the problem faster. This motivated us to solve the Maximum Clique problem using both CPU and GPU system.

The primary problem to properly utilize both the system simultaneously is to reduce the communication cost between CPU and GPU. The best way to resolve the problem is to partition the graph in such a way that two devices can work independently and combine the results at the end.

Let $G = (V, E)$ be a graph with V vertices and E edges. A sub-graph $H = (W, E|W)$ Induced by set W is a core of order k if $\forall v \in W : deg_H(v) \geq k$ and H is the maximum sub-graph with that property [1]. If the graph contains a clique of size k then all the vertices of that clique will have degree at least $k-1$. Hence, maximum core value gives the upper bound of the clique size of any graph. Prior works using sequential algorithm to determine core number.

The maximum core number gives a upper bound of the size of the clique. If the maximum core number is denoted as $K(G)$ and size of the maximum clique of the graph is $MC(G)$ then $MC(G) \leq K(G) + 1$. As there is a high probability of containing the vertex which has higher core number taking vertices in decreasing order of their core number helps to find heuristic clique faster. If the size of the heuristic clique is same as $K(G)+1$ then we have found the maximum clique and the algorithm ends. For some real world graphs heuristic clique size reaches $K(G) + 1$. If we denote heuristic clique size by $HC(G)$, then $HC(G) \leq MC(G) \leq K(G) + 1$ [2]. So, after finding this i.e $HC(G)$, we can prune the vertices with core number less than $HC(G)$. We have found in real world graphs it is pruning 90% of the vertices on an average and for some large graphs with more than 500M edges, it is pruning more than 99% of the vertices.

After pruning the graph becomes smaller. Then finally perform exhaustive search technique to find the exact clique searches in the newly reduced graph. Hence, for larger graphs with millions of vertices and more than 500 millions of edges it takes few minutes to find the maximum clique.

Our Contribution:

A. We are partitioning the graph according to the size that GPU can accommodate using PaToH[1] partitioning tool and renaming the vertices in each local partition to reduce the required storage space.

B. We have implemented a new hybrid core finding algorithm that will work on two different partitions simultaneously one in GPU and another in multi-core CPU and at the same time sort the vertices in decreasing order of their core number.

C. We have implemented a new iterative algorithm to find heuristic clique using core ordering of the vertices.

D. While extensively searching for all the vertices using multiple threads we are pruning out vertices that cannot be a part of maximum clique.

II. Related Work

Schmidt et al. [3] has implemented parallel maximum clique in distributed system. But instead of partitioning the graph

[1]bmi.osu.edu/umit/software.html

they are copying the entire graph to every nodes. So, for large graphs a huge memory will be wasted. Cruz et al. [4] has implemented maximum clique in GPU using matrix vector multiplication but they are checking for all possible cases by making vector element 1 or 0. So, if the graph has n vertices they are checking for all 2^n possibilities which bound the algorithm to work for small graphs only. Pattabiraman et al. [5] has used branch and bound and after finding heuristics they are pruning large number of vertices. But they are using serial method. Rossi et al. [2] has implemented on top of Pattabiraman et al. work and using k core computation to find a stronger bound with openmp threads. But they are finding k core sequentially.

III. Methodology

A. Graph Partitioning

We are implementing the algorithm to work on CPU and GPU simultaneously. To reduce communication overhead at every step we are making variable partitioning of the graph according to memory GPU can accommodate for any graph size. We rename the vertices individually in each partition. The potential clique may contain the ghost vertices so we are copying edges between ghost vertices to every partition. So there is no intermediate communication. To partition the graph we use a partitioning tool called PaToH(Partitioning Tools for Hypergraph).

B. Finding K-core Number

A vertex induced sub-graph is said to be a core of order k if all the vertices in the sub-graph has a degree at least k. The core number of a vertex v is the largest k such that v is in core of order k. If the graph contains a clique of size k then all the vertices of that clique will have degree at least $k-1$. Hence, maximum core value gives the upper bound on the maximum clique size of the graph. There exists an $O(E)$ sequential algorithm [1] to determine the core number of all the vertices of the graph. A modified parallel algorithm [6] for finding the core number of the vertices is given in algorithm 1. A modified version of this algorithm is used in GPU. We are using the level synchronous approach used in [7] for BFS to find core number in GPU.

C. Finding Heuristics

We order the vertices of the graph according to their core number (decreasing) at the time of computation of core number. As the potential clique has a higher probability to contain vertices with larger core numbers. We then search iteratively using open-mp threads for a potential clique. If it is equal to maximum $(corenumber + 1)$ then we have already found a maximum clique. If not then prune all the vertices that have core number less than the size of the heuristic clique. This will reduce the size of the graph significantly. The algorithm for the heuristic clique is given in algorithm 2.

Algorithm 1 Parallel Find Core Number

```
procedure FIND CORE NUMBER ($G = (V, E)$)
    for all $v \in V$ do
        $core[v] \leftarrow degree[v]$
    end for
    $level \leftarrow 1$
    $remaining \leftarrow |V|$
    while $remaining > 0$ do
        $current\_set \leftarrow$ FIND CURR SET$(level)$
        while $|current\_set| > 0$ do
            $remaining \leftarrow remaining - |current\_set|$
            $next\_set \leftarrow$ FIND NEXT SET$(current\_set)$
            $current\_set \leftarrow next\_set$
        end while
        $level \leftarrow level + 1$
    end while
end procedure
procedure FIND CURR SET(level)
    $current\_set = \emptyset$
    for $i = 1$ to $|V|$ do                              ▷ In Parallel
        if $core[i] == level$ then
            Add $i$ to $current\_set$
        end if
    end for
    return $current\_set$
end procedure
procedure FIND NEXT SET($current\_set, level$)
    $next\_set = \emptyset$
    for $i = 1$ to $|current\_set|$ do                   ▷ In Parallel
        for all $u \in$ adjacent to $current\_set[i]$ do
            if $core[u] > core[v]$ then
                $atomic\_decrement(core[u], 1)$
            end if
            if $core[u] <= level$ then
                $core[u] = level$
                Add $u$ to $next\_set$ once
            end if
        end for
    end for
    return $next\_set$
end procedure
```

D. Finding exact Clique

The finding of the maximum clique procedure is done only on the multi-core CPU. We begin with the pruned graph and naive way is search exhaustively through all combinations. But for large graph this takes longer time even in parallel systems. So we use intelligent stopping criteria (or) bounds. Parallelization strategy we have used is exploratory parallelization. We did further optimization of pruning the graph even further. The algorithm for the exact clique is given in algorithm 3. The input for the procedure Clique Finder is the graph remaining after pruning.

Algorithm 2 Finding heuristic Clique

```
 1: procedure FIND HEURISTIC CLIQUE($G = (V, E), core$)
 2:     $ub \leftarrow max(core) + 1$
 3:     $max\_Cl\_size \leftarrow 0$
 4:     $clique \leftarrow \emptyset$
 5:     for $i = 1$ to $|V|$ do                      ▷ In Parallel
 6:         $v \leftarrow V[i]$
 7:         $candidate \leftarrow \emptyset$
 8:         if $core[v] > max\_Cl\_size$ then
 9:             for all $u \in$ adjacent to $v$ do
10:                 if $core[u] > max\_Cl\_size$ then
11:                     Add $u$ to $candidate$
12:                 end if
13:             end for
14:             if $|candidate| > max\_Cl\_size$ then
15:                 Sort $candidate$ by $core$
16:                 Add $v$ to $clique$
17:                 loop
18:                     $m \leftarrow$ first $vertex$ from $candidate$
19:                     Add $m$ to $clique$
20:                     $upd \leftarrow candidate \cap adjacent(m)$
21:                     if $|upd| == 0$ then
22:                         if $|clique| > max\_Cl\_size$ then
23:                             Update $clique$
24:                             Update $max\_Cl\_size$
25:                         end if
26:                         break;
27:                         if $max\_Cl\_size == ub$ then
28:                             Stop Procedure
29:                             Communicate to other cores
30:                         end if
31:                     else
32:                         $candidate \leftarrow upd$
33:                     end if
34:                 end loop
35:             end if
36:         end if
37:     end for
38: end procedure
```

Algorithm 3 Find Exact Clique

```
 1: procedure CLIQUE FINDER($G' = (V, E)$)
 2:     $max\_Cl\_size \leftarrow 0$
 3:     $clique \leftarrow \emptyset$
 4:     for $i = 0$ to $|V|$ do                      ▷ In Parallel
 5:         $N \leftarrow$ neighbours of $V[i]$
 6:         if $|N| + 1 > max\_Cl\_size$ then
 7:             EXPAND($N, \emptyset, clique$)
 8:         end if
 9:         prune $V[i]$
10:     end for
11:     return $clique$
12: end procedure
13: procedure EXPAND($N, prev\_clique, clique$)
14:     while $|N|! = 0$ do
15:         if $|N| + |prev\_clique| < |clique|$ then
16:             return
17:         end if
18:         $v \leftarrow first$ vertex from $N$
19:         Add $v$ to prev_clique
20:         $N = N \cap neighbour(v)$
21:         if $|N| == 0$ then
22:             if $|prev\_clique| > |clique|$ then
23:                 $clique \leftarrow prev\_clique$
24:             end if
25:         else
26:             EXPAND($N, prev\_clique, clique$)
27:         end if
28:         Remove $v$ from $N$
29:         Remove $v$ from $prev\_clique$
30:     end while
31: end procedure
```

IV. Experiments and Results

A. Experimental Setup

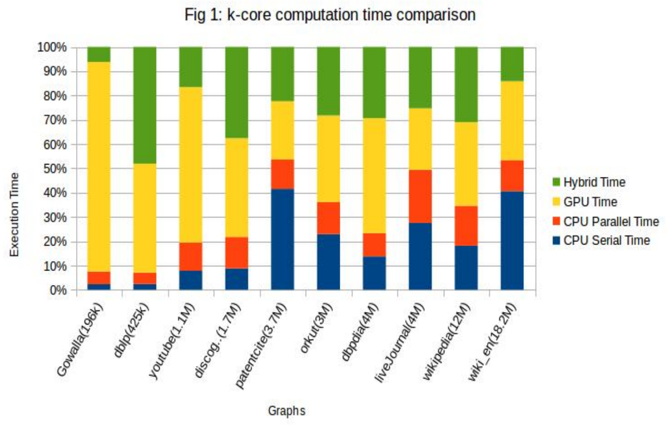
All our experiments were performed on a GPU server consisting of a dual octo-core Intel Xeon E5-2670 2.6 GHz server with CentOS 6.4, 128 GB RAM, and 1 TB GB hard disk. The CPU is connected to two NVIDIA Kepler K20 cards. We denote the existing parallel CPU code of Rossi et al. as pmc. We have compared results of every step with our standalone CPU code. We are performing the last step of finding exact clique of both the partitions separately in CPU. At every step of GPU we are comparing with pmc code also. We are creating variable partitioning using Patoh tool according to the data size GPU can accommodate. We are also comparing this hybrid version with all the three algorithm mentioned above.

Graph	Vertices	Edges
Gowalla	196k	2M
dblp	425k	2M
youtube	1.1M	6M
discogsaffiliation	1.7M	10.6M
patentcite	3.7M	33M
orkut	3M	234M
dbpdia	4M	25M
liveJournal	4M	69M
wikipedialink	12M	576M
wikieng	18.2M	181M

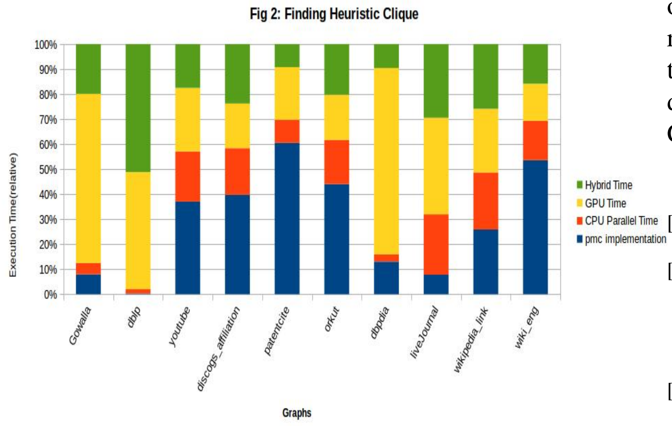
TABLE I
TABLE 1 : GRAPHS FOR EXPERIMENTS

B. Results

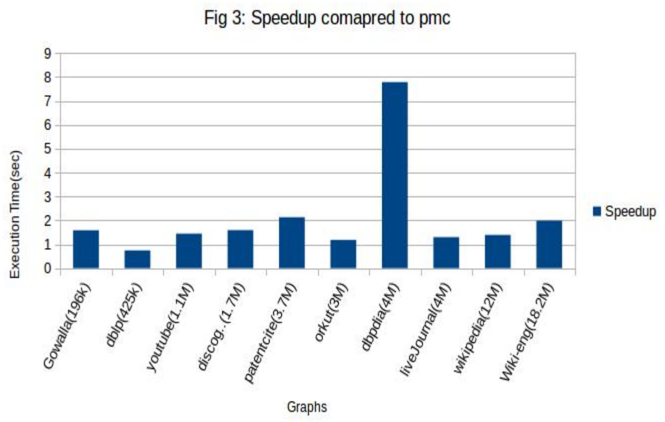
The graphs used in our experiments are shown in Table 1. We have made the graphs symmetric so if there is an edge between (u,v) we are making an edge (v,u) also. We have taken the graphs from the Stanford Network Analysis Platform (SNAP) and Koblenz Network Collection. The graphs belong to different kind of applications.



First we are finding the core number of all the vertices and we have compared the result of this step with existing pmc work(Fig 1). From that graph we can see that our CPU parallel algorithm is always performing better than existing algorithm. But our GPU algorithm is sometimes performing bad because to find the core number we are increasing the core number one by one starting from 1 (refer Algorithm 1). So, when the core number is high, at every step we need to synchronize with CPU to increase the core number to next level. So, we are getting a huge synchronization overhead that makes the performance to degrade.



We are finding the heuristic clique using Algorithm 2. Previous work was using recursion to search heuristic clique. But recursion always have the stack overhead and also using some local vectors to store. We have optimized and introduced a new iterative algorithm which reduces these overheads and our CPU always performing faster than their pmc algorithm(Fig 2) and sometimes we are able to get good bounds also compared to their version. In hybrid as we are partitioning the graph in two partitions and finding heuristic for two different partition simultaneously it is even performing better than CPU standalone version.



After finding the lower bound and pruning the vertices we are performing the last step in CPU for finding exact clique. In the final experiment we have taken the whole time taken by CPU and compared the execution time taken by pmc algorithm. We are getting descent speedup compared to pmc(Fig 3).

V. Conclusions

We were able to find maximum Clique on a graph of size (12 million vertices, 576 million edges) within couple of minutes. We have proposed a new heuristic finding algorithm on hybrid system that performs better in terms of time and in some cases a better bound. We have done optimizations on the existing work and we have obtained speedup. The algorithm uses integer vectors and one can do better by using efficient data structures such as bit representation and perform optimizations on that. Since the bit representation of graphs reduces the size, larger graphs can be accommodated in GPU to find the maximum clique. Also this could be extended to distributed systems with each node consisting of a multi-core CPU and a high-end GPU device.

References

[1] V. Batagelj and M. Zaversnik, "An o (m) algorithm for cores decomposition of networks," *arXiv preprint cs/0310049*, 2003.

[2] R. A. Rossi, D. F. Gleich, A. H. Gebremedhin, and M. M. A. Patwary, "Fast maximum clique algorithms for large graphs," in *Proceedings of the companion publication of the 23rd international conference on World wide web companion*. International World Wide Web Conferences Steering Committee, 2014, pp. 365–366.

[3] M. C. Schmidt, N. F. Samatova, K. Thomas, and B.-H. Park, "A scalable, parallel algorithm for maximal clique enumeration," *Journal of Parallel and Distributed Computing*, vol. 69, no. 4, pp. 417–428, 2009.

[4] R. Cruz, N. Lopez, and C. Trefftz, "Parallelizing a heuristic for the maximum clique problem on gpus and clusters of workstations," in *IEEE International Conference on Electro-Information Technology, EIT 2013*, 2013.

[5] B. Pattabiraman, M. M. A. Patwary, A. H. Gebremedhin, W.-k. Liao, and A. Choudhary, "Fast algorithms for the maximum clique problem on massive sparse graphs," in *Algorithms and Models for the Web Graph*. Springer, 2013, pp. 156–169.

[6] N. S. Dasari, R. Desh, and M. Zubair, "Park: An efficient algorithm for k-core decomposition on multicore processors," in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 9–16.

[7] S. Hong, T. Oguntebi, and K. Olukotun, "Efficient parallel graph exploration on multi-core cpu and gpu," in *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*. IEEE, 2011, pp. 78–88.