A contribution to acceleration of graphlet counting

Aleksandar Milinković
Belgrade University, School of Electrical Engineering
Belgrade, Serbia
amilinko@gmail.com

Stevan Milinković
Union University, School of Computing
Belgrade, Serbia
smilinkovic@raf.edu.rs

Ljubomir Lazić
Metropolitan University, Faculty of Information Technology
Belgrade, Serbia
ljubomir.lazic@metropolitan.ac.rs

Abstract – **Graphlets are small non-isomorphic connected subgraphs used for different kinds of analyses in social networks, bioinformatics and other areas described by large networks, where their number can provide a characterization of the network properties. Much of existing methods for counting the graphlets are based on direct enumeration. However, in case of large networks, this type of counting becomes computationally very demanding. Fortunately, for very sparse networks the computational cost is much less prohibitive than in dense networks, leading to the more efficient graphlet counting algorithms. This work is our first attempt to accelerate one of such algorithms by parallelization using graphic hardware.**

Key words – graphlet; orbit; network; parallel algorithm;

I. INTRODUCTION

The network generally describes a set of objects (nodes), as well as the relationship between these objects. By studying the technological, sociological or biological networks we can get information about the interactions between objects, their individual characteristics and their links with the characteristics of the network as a whole.

Graphlet counting is a topologically rigorous way to characterize the structure around a network node. The number of appearances of graphlets in the network provides a description of the network's structural properties. This method has been used to solve several network analysis problems such as comparing networks [1], network alignment [2], network clustering [3], etc. Unfortunately, graphlet counting can be very expensive, especially when network is very large.

We need some graphlet definitions here, so we follow [4]. Assume, $G(V, E)$ is a graph, then V is the set of vertices and E is the set of edges. Each edge $e \in E$ can be represented by a pair of vertices (v_i, v_j) where, $v_i, v_j \in V$. A graph is called simple, if it does not contain a self loop, and at most one edge exists between two of its vertices. We consider simple, connected, and undirected graphs. A graph $G' = (V', E')$ is a subgraph of G if $V' \subseteq V$ and $E' \subseteq E$. A graph $G' = (V', E')$ is a vertex-induced subgraph of G if $V' \subseteq V$ and $E' \subseteq E$ and $\{e = (v_a, v_b) : v_a, v_b \in V', e \in E, e \notin E'\} = \emptyset$. A vertex-induced subgraph is a subset of the vertices of a graph G together with any edges whose both endpoints are in this subset. We will refer to vertex-induced subgraph as "induced subgraph". Two graphs G and G' are isomorphic, denoted by $G \cong G'$, if there exists a structure-preserving (both adjacency and non-adjacency preserving) bijection $f : V \to V'$; such a function f is called an isomorphism from G to G'.

An embedding of a graph G' in another graph G is a subgraph S of G, such that S and G' are isomorphic; when the subgraph S is a vertex-induced subgraph of G, the embedding is called an induced embedding. Therefore, graphlets can be defined as small, non-isomorphic, connected, induced subgraphs of a large network. We are interested in all possible graphlets having k vertices, $k \in \{3, 4, 5\}$. We refer a graphlet with k vertices, as *k-node graphlet*. Note that, 1-node graphlet is simply a vertex, and 2-node graphlet is an edge. A k-node graphlet is called a tree graphlet if it is a tree, i.e., it has $k - 1$ edges. A graphlet that is not a tree is called a cyclic graphlet.

An isomorphism from a graph $G(V, E)$ to itself is called an automorphism. For a finer description, the nodes of every graphlet are partitioned into a set of automorphism groups called orbits [1]. Two nodes belong to the same orbit if they map to each other in some isomorphic projection of the graphlet onto itself.

The remainder of this paper is organized as follows. The next section provides background, and compares and contrasts some graphlet counting algorithms. Section three demonstrates the need for parallel counting, whereas the fourth section describes one successful sequential algorithm. The fifth section describes or main contribution, and the sixth section shows some of our results. Finally, section seven provides conclusions and recommendations for future work.

II. RELATED WORK

In terms of complexity, counting all 5-node graphlets in a graph G has a time complexity of $O(|V|^5)$, and therefore becomes a computational bottleneck. Common approaches to speed it up include sampling [5]–[7], exploiting pattern symmetries [8] or using reconfigurable hardware accelerators based on FPGA chips [9].

There are several software implementations for graphlet counting that are mainly used in bioinformatics. FANMOD

Results are part of the research that is supported by Ministry of Education and Science of the Republic of Serbia, Grants No. III-45003 and TR-35026.

[10] is a network motif detection tool based on sampling random subgraphs and comparing their counts with those from random network models. Note that motif is partial subgraphs, whereas graphlet must be induced subgraph. Besides implementing a novel sampling algorithm [5] it also provides a full enumeration procedure for graphlets on 2–8 nodes. GraphletCounter, described in [11], works as a Cytoscape [12] plug-in and merges graphlet analysis with visual inspection of the network.

GraphCrunch [13] is a tool for large network analysis. It counts graphlets of up to five nodes using an enumeration procedure with correction for overcounting some of the graphlets. A well-organized enumeration method imposes constraints that eliminate the need for isomorphism testing except for distinguishing between a few different graphlets. This is further accelerated by comparing the number of edges and individual node degrees. GraphCrunch has been extended with a new method for topological network alignment and with comparison of the networks with some additional mathematical models [14]. The graphlet counting procedure in this new version of GraphCrunch remained essentially the same.

Rapid Graphlet Enumerator (RAGE) [15] takes a different approach to counting four-node graphlets. Instead of counting the induced subgraphs directly, it reconstructs them from counts of non-induced subgraphs. For computing the latter, it uses specifically crafted methods for each of the 6 possible subgraphs. Unlike FANMOD and GraphCrunch, RAGE works only for up to four-node graphlets.

An alternative to exact graphlet counting is to adopt algorithms for approximate counting that offer significant speedup with a small counting error. This direction has become popular in some of the recent researches for counting triangles [16], [17]. Since the cost of graphlet counting is much higher than the cost of triangle counting, an approximate counting algorithm for the former will be more useful from a practical standpoint. In their research [4], the authors propose a method called Graft to perform the task of approximate counting of graphlets that have up to five vertices.

III. PARALLEL COUNTING

Graph problems have inherent parallelism, where a task is performed independently on all vertices of the graph. The shortest path algorithm is an example, where for a given graph the shortest path problem for each vertex can be solved in parallel as an independent task. As graph problems grow in size, efficient parallel graph processing becomes important as computational and memory requirements increase. Unfortunately, traditional solutions that are used to parallelize mainstream parallel applications do not necessarily work well for large-scale graph problems. There are a number of properties that make them poorly matched to computational methods which are successfully applied in mainstream parallel applications [9]:

- Graph computations are dictated by the vertex and edge structure of the graph, and the execution paths are difficult to analyze and predict using static analysis of the source code. Parallelism based on partitioning computations is a challenging task due to lack of knowledge about the structure of the computations.
- The data in graph problems are unstructured and highly irregular. This irregular structure of the graph data makes it difficult to partition the graph data to take advantage of small and fast on-chip memories, e.g. such as shared memory and registers in CUDA GPUs.
- Data-driven computations coupled with irregular data structures results in low memory access locality.
- Many graph algorithms tend to explore the structure of the graph while performing a relatively small amount of computations (e.g. counting the neighborhood nodes).
- All of this results in a higher ratio of data access to computation compared to mainstream scientific and engineering applications, and combined with poor locality leads to execution times almost dominated by memory latency.

Algorithm 1 shows a general template for the graph algorithm targeted by our work. It consists of a loop that iterates through all the vertices in the graph. Suppose that each iteration can be performed as a separate kernel. The outer-loop represents the coarse-grained parallelism required for our case, while further fine-grained parallelism may be available within the graph kernel itself .

Algorithm 1. The counting algorithm for *k*-node graphlets, $k \in 3, 4, 5$.

```
for all nodes a of G {
    for all adjacent nodes b of a {
        for all adjacent nodes c of b {
        // finding 3-node graphlets
            for all adjacent nodes d of c {
            // finding 4-node graphlets
                for all adjacent nodes e of d {
                // finding 5-node graphlets
                }
            }
        }
    }
}
```

Here we require 72 counters per vertex to enumerate all the 3-, 4-, 5-node graphlets for a given graph [18]. These counters must be stored in off-chip memory for large graphs. A counter may be incremented by two or more threads simultaneously requiring a synchronization mechanism. In the case of a system with a large number of parallel threads, synchronization due to high-contention situations can become a performance bottleneck because of the additional delays introduced by contention. This issue is often addressed by a combination of a parallel implementation of the outer loop and a serial implementation of all the inner loops. The serial implementation takes the form of a graph kernel, while the parallel implementation comes from the replication of this kernel. In other words, several kernels operate in parallel, and each kernel processes a graph node at a time; i.e. an iteration of the outer-loop. Implementing the inner loops sequentially on dedicated parallel processor would most likely result in slower execution times compared to software implementations..

Fortunately, there exists an algorithm called Orca (Orbit Counting Algorithm) [18], which reduces the time complexity by an order of magnitude by computing the orbit counts using the relations between them and directly enumerating only smaller graphlets. This algorithm is sequential in nature, however it can be parallelized, as suggested by its authors.

IV. A BRIEF DESCRIPTION OF ALGORITHM

Orca proposes a combinatorial method for counting graphlets and orbit signatures of network nodes. The algorithm builds a system of equations that connect counts of orbits from graphlets with up to 5 nodes, which allows to compute all orbit counts by enumerating just a single one. We use this algorithm as a starting point for our work.

The core of the algorithm is based on deriving a linear relationship between the orbit counts of various graphlets where the right hand side of the equation can be determined from the graph with an efficient algorithm.

Specifically, algorithm allows for computing, for each node in the network the node's graphlet degree vector (i.e. its authomorphism orbit counts) for all up to *k*-node graphlets/orbits simply by computing the same statistics for all up to (*k*–1)-node graphlets and for a single *k*-node graphlet/orbit. Then, the statistics for the remaining *k*-node graphlets/orbits can be computed through a system of equations of dependencies between different orbits/graphlets. This offers a significant empirical speed up compared to the existing methods on real-world networks.

However, although the computation time is reduced by an order of magnitude as compared to the existing, pure enumeration-based algorithms, it does not reduce the theoretical computational complexity of graphlet counting procedure. The complexity of the procedure is still bounded by the identification of 4-node (or 5-node) cliques [19]. Nevertheless, it gives impressive results even in case of not-so-sparse graphs (e.g. up to 40% density on 4-node graphlets).

Right-hand sides of equations contain terms that are computed from the graph G. Let $c(u, v) = |N(u) \cap N(v)|$ denote the number of common neighbors of nodes u and v. Let $p(u, v)$ denote the number of paths on three nodes that start at node u, continue with v can compute $p(u, v)$ as $p(u, v) = deg(v) - 1 - c(u, v)$. If some node x participates in a k-node graphlet G_i, it also participates in some $(k-1)$-node graphlet G_j. This can be seen by removing one of the graphlet's nodes that are the farthest away from x. The subgraph induced by the remaining nodes is connected (any disconnected node would have to be farther from x than the removed node), so it is isomorphic to some $(k-1)$-node graphlet G_j. We will use this observation in reverse: every four-node graphlet can be constructed by adding a node to some three-node graphlet(s). To find the relations between counts of orbits in four-node graphlets for a certain node x, we enumerate all three-node graphlets touching the node and count their possible extensions with the fourth node.

There are only two three-node graphlets and relatively few possible extensions. Investigating all possibilities in a similar manner yields 10 equations with 11 variables that correspond to counts of 11 orbits in four-node graphlets [18].

Right-hand sides depend on the graph G and need to be computed for each point x. To accelerate their computation, we precompute values of $c(u, v)$ and $p(u, v)$. In all equations, except for the last one, $c(u, v)$ is computed on pairs of nodes (u, v) that are connected; in $p(u, v)$, they are connected by the definition of p. It therefore suffices to precompute $c(u, v)$ and $p(u, v)$ only for all pairs of connected nodes u and v, which requires $O(e)$ space. The last equation, in which the new node closes a cycle, is treated separately. Nodes x and z are not adjacent but we can pre-compute the number of paths of length 2 that start at node x and end at node y. This requires $O(n)$ space for each point; since we compute orbits for one point at a time, this memory can be recycled.

$$o_{12} + 3o_{14} = \sum_{y,z|y<z,G[\{x,y,z\}]\cong G_2} c(y,z) - 1$$

$$2o_{13} + 6o_{14} = \sum_{y,z|y<z,G[\{x,y,z\}]\cong G_2} (c(x,y)-1) + (c(x,z)-1)$$

$$o_{10} + 2o_{13} = \sum_{y,z|y<z,G[\{x,y,z\}]\cong G_2} p(y,z) + p(z,y)$$

$$2o_{11} + 2o_{13} = \sum_{y,z|y<z,G[\{x,y,z\}]\cong G_2} p(y,x) + p(z,x)$$

$$6o_{7} + 2o_{11} = \sum_{y,z|y<z,y,z\in N(x),G[\{x,y,z\}]\cong G_1} (p(y,x)-1) + (p(z,x)-1)$$

$$o_{5} + 2o_{8} = \sum_{y,z|y<z,y,z\in N(x),G[\{x,y,z\}]\cong G_1} p(x,y) + p(x,z)$$

$$2o_{6} + 2o_{9} = \sum_{y,z|x,z\in N(y),G[\{x,y,z\}]\cong G_1} p(x,y) - 1$$

$$2o_{9} + 2o_{12} = \sum_{y,z|x,z\in N(y),G[\{x,y,z\}]\cong G_1} c(y,z)$$

$$o_{4} + 2o_{8} = \sum_{y,z|x,z\in N(y),G[\{x,y,z\}]\cong G_1} p(y,z)$$

$$2o_{8} + 2o_{12} = \sum_{y,z|x,z\in N(y),G[\{x,y,z\}]\cong G_1} c(x,z) - 1$$

In these equations $\cong$ denotes graph isomorphism (e.g., G[{x, y, z}], a subgraph on nodes x, y, z, is isomorphic to G1, a path with three nodes).

V. METHODS

From original source code [22], we can see that the algorithm is implemented in three stages:

stage 1 - precomputing common nodes;

stage 2 - counting full graphlets;

stage 3 - building systems of equations relating orbits for every node. After right-hand sides are determined, all equations are solved.

Our measurements show that in first two stages, the algorithm spends only 24% of total execution time, and third stage consumes the rest of execution time. Also in the first two stages, there are precedence relations among computations. In

contrast, in the third stage results are mutually independent. Therefore, we put more emphasis on the third stage.

Our code is implemented on an x86 Linux machine (Ubuntu 3.2.0-60.91) with the graphic card nVIDIA GTX 690 (compute capability 3.0). The GTX 690 features two Kepler-based GeForce GPUs (same architecture as Tesla K-series GPU accelerators), 4GB of GDDR5 RAM and 3072 CUDA cores. As a part of the installed CUDA Toolkit, we used Nsight Eclipse Edition 5.5.0 to edit, build and debug our application.

A. Graph representation on CUDA

A graph $G(V, E)$ is commonly represented as an adjacency matrix. For sparse graphs such a representation wastes a lot of space. Adjacency list is a more compact representation for graphs. Because of variable size of edge lists per vertex, we take an advantage of CUDA which allows arrays of arbitrary sizes to be created and hence can represent graph using adjacency lists. We represent graphs in compact adjacency list form, with adjacency lists packed into a single large array. Each vertex points to the starting position of its own adjacency list in this large array of edges. Vertices of graph $G(V, E)$ are represented as an array V_a. Another array E_a of adjacency lists stores the edges with edges of vertex i+1 immediately following the edges of vertex i for all i in V. Each entry in the vertex array V_a corresponds to the starting index of its adjacency list in the edge array E_a. Each entry of the edge array E_a refers to a vertex in vertex array V_a.

B. CUDA kernels

We have implemented three kernels, one for each stage, which are executed sequentially, one after another. Each kernel is launched with 1024 blocks, and 1024 threads per block. Each thread is assigned to one graph vertex. Since in this work we implement only 4-node counting, it runs as follows.

Initialization

- Reading graph from input file
- Checking for duplicate undirected edges, self loops and other irregularities
- Determining degree of nodes

Stage 1

- Device memory allocation and initialization
- Copy vertices and edges pairs, and adjacency list
- Launch kernel for precomputing common vertices
- Precompute triangles that span over edges

Stage 2

- Launch kernel for counting full graphlets
- Each device thread performs its own binary search in order to find adjacent nodes

Stage 3

- Building and solving systems of equations
- Per thread binary search is also executed here

For execution time measurements on GPU we used recordings of event occurrences, as suggested in [20].

If we want to engage both GTX690 GPUs (devices 0 and 1) we need to do some kind of graph partitioning, which is not a trivial task. Then we have to start two separate threads in host, and since nowadays hosts are likely to be multicore, we should schedule these threads to different cores. Of course, we have to allocate portable pinned host memory here in order to take advantage of this 2GPU/2CPU threads arrangement.

VI. Results and Discussion

We have tested our code with three sets of input data, available with the original code [21]. They were generated random graphs using Barabási-Albert, Erdős-Rényi and geometric algorithms [22]. All sets had in common the number of vertices (1000), however number of edges varied from 4000 to 200000. Hence, for comparison we used number of edges and execution time (in seconds) for counting four-node graphlets in our networks (Fig.1). We can easily see better results for more connected graphs.

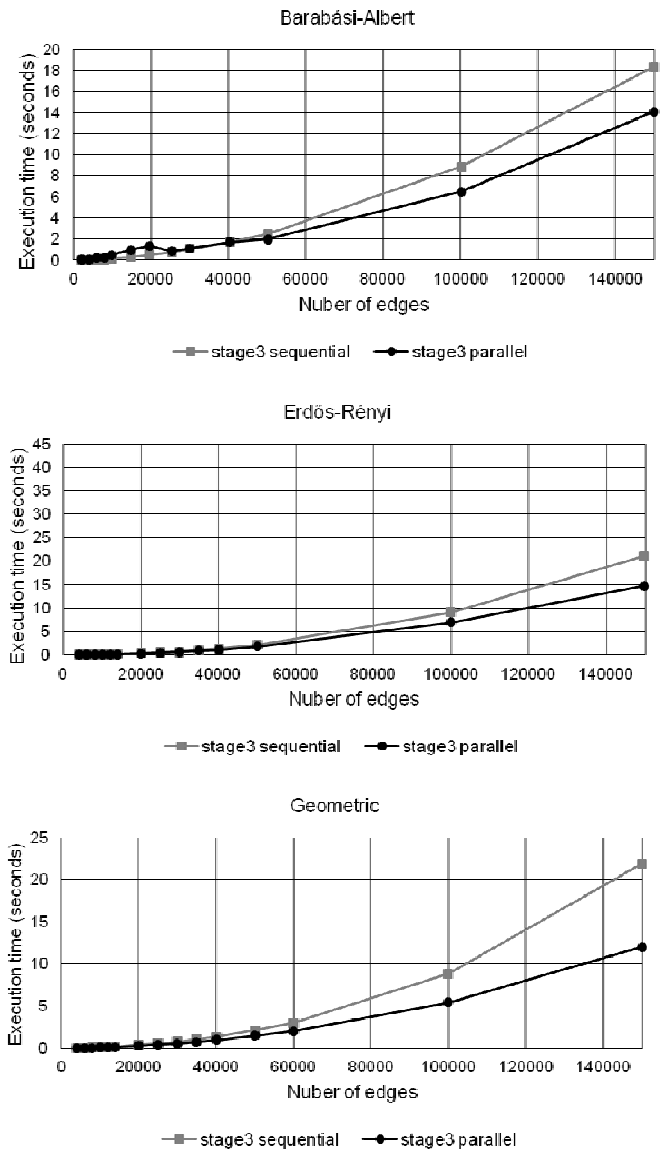

Figure 1. The speedup of algorithm execution vs. number of graph edges for three data sets: scale-free, Erdős-Rényi and geometric.

Compared to original Orca sequential code, the time complexity of our solution is further reduced as can be seen from Table 1.

TABLE 1. TIME COMPLEXITY FOR 4-NODE GRAPHLET COUNTING

Solution	Time complexity
Genuine Algorithm 1	$O(n^4)$
Orca	$O(n^3)$
Our solution	$O(n^2)$

where $n = |V|$ denotes the number of vertices.

Of course, our result holds provided we have enough cores in GPU in order to schedule one thread per core (as is the case in our experiments). Otherwise, CUDA runtime will schedule more threads per core, introducing context switching. Since we have non-blocking threads, this would cause overhead that would make this part of execution slower than sequential. In such a case, we need to activate another device, as we suggested in Section V, or even more, deploy additional GPUs.

VII. CONCLUSION

Many real-world problems, such as various types of social networks, computer networks and biological interactions, have been represented as large graphs or networks involving millions of vertices. As graph problems grow in size, efficient parallel graph processing becomes important as computational and memory requirements increase. Our work is based on modification of Orca, a new algorithm for counting graphlet orbits. The original (sequential) algorithm counts orbits in large protein-protein interactions networks 50-100 times faster than other state-of-the-art algorithms. Our proposal improves the third stage of this algorithm almost two times for higher number of edges, by means of simple parallelization using commercially available graphic card. In future we plan to extend our program in order to parallelize counting of 5-node graphlet, since it differs from 4-node algorithm and is more complex. We also plan to refine our procedure and to try to circumvent dependencies in first two stages, which will allow us to cover the whole algorithm in one kernel with little use of synchronization mechanisms.

REFERENCES

[1] N. Pržulj, "Biological network comparison using graphlet degree distribution," *Bioinformatics*, vol. 23, no. 2, pp. e177–e183, Jan. 2007.

[2] O. Kuchaiev, T. Milenkovic, V. Memisevic, W. Hayes, and N. Przulj, "Topological network alignment uncovers biological function and phylogeny," *Journal of The Royal Society Interface*, vol. 7, no. 50, pp. 1341–1354 , Oct. 2010.

[3] T. Milenković, V. Memišević, A.K. Ganesan, and N. Pržulj, "Systems-level cancer gene identification from protein interaction network topology applied to melanogenesis-related functional genomics data," *J. of The Royal Society Interface*, vol. 7, no. 44, pp. 423–437, 2010.

[4] M. Rahman, M. Bhuiyan, and M. Al Hasan, "GRAFT: An Approximate Graphlet Counting Algorithm for Large Graph Analysis", in Proceedings of the 21st ACM international conference on Information and knowledge management (CIKM'12), Maui, HI, USA, 2012, pp. 1467–1471.

[5] N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon, "Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs," *Bioinformatics*, vol. 20, no. 11, pp.1746–1758, July 2004.

[6] S. Wernicke, "Efficient detection of network motifs," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 3, no. 4, pp. 347–359, Oct. – Dec. 2006.

[7] N. Pržulj, D.G. Corneil, and I. Jurisica, "Efficient estimation of graphlet frequency distributions in protein-protein interaction networks," *Bioinformatics*, vol. 22, no. 8, pp. 974–980, Apr. 2006.

[8] A. Stoica and C. Prieur, "Structure of Neighborhoods in a Large Social Network," in International Conference on Computational Science and Engineering (CSE'09), Vancouver, BC, Canada, 2009, pp. 26–33.

[9] B. Betkaoui, D.B. Thomas, W. Luk, and N. Przulj, "A framework for FPGA acceleration of large graph problems: Graphlet counting case study," in 2011 International Conference on Field-Programmable Technology (FPT), New Delhi, India, 2011, pp. 1–8.

[10] S. Wernicke F. Rasche, "FANMOD: a tool for fast network motif detection," *Bioinformatics*, vol. 22, no. 9, pp. 1152–1153, May 2006.

[11] C. Whelan and K. Sönmez, "Computing graphlet signatures of network nodes and motifs in Cytoscape with GraphletCounter," *Bioinformatics*, vol. 28, no. 2, pp. 290–291, Jan. 2012.

[12] Available: http://www.cytoscape.org/

[13] T. Milenković, J. Lai, and N. Pržulj, "GraphCrunch: a tool for large network analyses," *BMC Bioinformatics*, vol. 9, no. 70, pp. 1 – 11, Jan. 2008.

[14] O. Kuchaiev, A. Stevanović, W. Hayes, and N. Pržulj, "GraphCrunch 2: Software tool for network modeling, alignment and clustering," *BMC bioinformatics*, vol. 12, no. 24, pp. 1 – 13, Jan. 2011.

[15] D. Marcus and Y. Shavitt, "RAGE - A rapid graphlet enumerator for large networks," *Computer Networks*, vol. 56, no. 2, pp. 810–819, Feb. 2012.

[16] C.E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "DOULION: Counting triangles in massive graphs with a coin," in Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'09), Paris, France, 2009, pp. 837–846.

[17] C.E. Tsourakakis, "Counting triangles in real-world networks using projections," *Knowledge and Information Systems*, vol. 26, no. 3, pp. 501–520, Mar. 2011.

[18] T. Hočevar and J. Demšar, "A combinatorial approach to graphlet counting," *Bioinformatics*, vol. 30, no. 4, pp. 559–565, Feb. 2014.

[19] R.D. Luce and A.D. Perry, "A method of matrix analysis of group structure," *Psychometrika*, vol. 14, no. 2, pp. 95–116, June 1949.

[20] M. Harris, "How to Implement Performance Metrics in CUDA C/C++," *nVIDIA Developer Zone,* Nov. 2012.

[21] Availiable:http://devblogs.nvidia.com/parallelforall/how-implement-performance-metrics-cuda-cc/

[22] Available: http://www.biolab.si/supp/orca/#download

[23] M. Newman. *Networks: An Introduction*. Oxford University Press, 2010.