Brief Announcement: Improved Massively Parallel Triangle Counting in $O(1)$ Rounds

Quanquan C. Liu
Yale University
New Haven, USA
quanquan.liu@yale.edu

C. Seshadhri
University of California, Santa Cruz
Santa Cruz, USA
sesh@ucsc.edu

ABSTRACT

In this short note, we give a novel algorithm for $O(1)$ round triangle counting in bounded arboricity graphs. Counting triangles in $O(1)$ rounds (exactly) is listed as one of the interesting remaining open problems in the recent survey of Im et al. [17]. The previous paper of Biswas et al. [8], which achieved the best bounds under this setting, used $O(\log \log n)$ rounds in sublinear space per machine and $O(m\alpha)$ total space where α is the arboricity of the graph and n and m are the number of vertices and edges in the graph, respectively. Our new algorithm is very simple, achieves the optimal $O(1)$ rounds without increasing the space per machine and the total space, and has the potential of being easily implementable in practice.

CCS CONCEPTS

• **Theory of computation** → **MapReduce algorithms**; **Distributed algorithms**; **Graph algorithms analysis**.

KEYWORDS

triangle counting, massively parallel computation, graph algorithms

ACM Reference Format:
Quanquan C. Liu and C. Seshadhri. 2024. Brief Announcement: Improved Massively Parallel Triangle Counting in $O(1)$ Rounds. In *ACM Symposium on Principles of Distributed Computing (PODC '24), June 17–21, 2024, Nantes, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3662158.3662819

1 INTRODUCTION

In this short extended abstract, we study the triangle counting problem in the Massively Parallel Computation (MPC) model. Given a simple, undirected input graph, $G = (V, E)$, the total triangle count is the number of cycles of length three in the graph. This problem has a wide variety of applications including community detection, spam detection, link recommendation, and social network analysis. Triangle counting and enumeration play key roles in database joins and are among the most important problems in database theory. Because of these applications, we often need to solve the problem on massive graphs and datasets. For a sample of works on triangle counting on large graphs, both in theory and practice, see [1–5, 7, 8, 12–14, 19, 21–25, 27, 28] and references therein.

PODC '24, June 17–21, 2024, Nantes, France

ACM ISBN 979-8-4007-0668-4/24/06
https://doi.org/10.1145/3662158.3662819

The Massively Parallel Computation (MPC) model [6, 16, 20] is one of the main models for modeling large distributed systems capable of processing graphs with billions or even trillions of edges. These systems include MapReduce [10], Hadoop [29], Spark [30] and Dryad [18]. In the MPC model, we are given M machines each with S memory. The initial input is partitioned arbitrarily across the machines. Computation is performed using a number of synchronous rounds. In each round, first, local computation is performed in each machine using the data that is stored in the machine. Then, data is sent between machines synchronously. No machine can receive or send more than S data. The complexity measures we care about in this setting are the number of rounds of communication, R, the space per machine, S, and the total memory used by the computation, which is equal to $M \cdot S$. This model has gained much interest within the distributed and parallel communities with an abundance of research works published within just the past few years. For a survey of such works, see [17] and references therein.

In this paper, we give a novel algorithm for constant-round triangle counting in bounded arboricity graphs. Bounded arboricity graphs are graphs whose edges can be decomposed into a small number of forests. Specifically, a graph has arboricity α if the set of edges in the graph can be decomposed into at most α forests. Various papers have demonstrated that a large number of real-world graphs have very small arboricity [11, 26]. The best previous result of Biswas et al. [8] for bounded arboricity graphs achieved $O(\log \log n)$ rounds in sublinear space per machine and $O(m\alpha)$ total space where α is the arboricity of the graph. Despite the importance of this problem for a wide variety of applications, triangle counting has remained elusive in the MPC model with either $O(1)$ round algorithms suffering from lower bounds on the count to guarantee good estimates or the best algorithms needing $\omega(1)$ rounds for an exact count. In this short note, we give a simple algorithm for the bounded arboricity setting where our algorithm returns an exact count of the number of triangles in $O(1)$ rounds, $O(n^\delta)$ space per machine for any constant $\delta > 0$, and using near-linear total space when α is small. An added bonus is that our algorithm is very simple, making it practically implementable for real-world networks, where α tends to be small.

2 PRELIMINARIES

We formally define the MPC model and arboricity in this section.

Definition 2.1 (Arboricity)**.** *The* ***arboricity****, α, of a graph $G = (V, E)$ is the minimum number of forests needed to decompose the edges of G.*

Definition 2.2 (Massively Parallel Computation (MPC) Model)**.** *In the massively parallel computation (MPC) model, there are M machines which communicate with each other in synchronous rounds. The input graph, $G = (V, E)$, is initially partitioned across the machines arbitrarily. Each machine has S space and performs the following (in order) during each round:*

(1) Each machine performs (unbounded) local computation using data stored within the machine. (Most reasonable algorithms will not use a large amount of time.)
(2) At the end of the round, machines exchange messages synchronously to inform the computation for the next round. The total size of messages sent or received by a machine is upper bounded by S.

We seek to minimize S, the number of rounds of communication, and the total space $S \cdot M$. There are three domains for the size of S: (i) Sublinear*: $S = n^{\delta}$ for some constant $\delta \in (0, 1)$; (ii)* Near-linear*: $S = \Theta(n \operatorname{poly}(\log n))$; (iii)* Superlinear*: $S = n^{1+\delta}$ for constant $\delta \in (0, 1)$.*

Throughout, we denote the degree of a vertex $v \in V$ by $\deg(v)$.

3 $O(1)$ ROUND EXACT TRIANGLE COUNTING

In this section, we give a very simple algorithm for counting the exact number of triangles in an input graph $G = (V, E)$ in $O(n^{\delta})$ space per machine for any constant $\delta > 0$. The main idea behind our algorithm consists of enumerating the wedges adjacent to the lower degree endpoint of every edge. We first show that the total memory necessary to perform such an enumeration is bounded by $O(m\alpha)$ where α is the arboricity of the input graph $G = (V, E)$. Chiba-Nishizeki [9] showed the following lemma that bounds the sum of the minimum of the degrees of the endpoints of every edge. For completeness, we include the proof of Lemma 3.1 in Appendix A.

Lemma 3.1 (Chiba-Nishizeki Sum of Minimum Degree Endpoints [9])**.** *Given an input graph $G = (V, E)$ with arboricity α, it holds that $\sum_{(u,v)\in E} \min(\deg(u), \deg(v)) \leq 2m\alpha$.*

Given Lemma 3.1, we give our MPC algorithm in Algorithm 1. We use a number of MPC primitives which are listed below and have been shown [8, 15] to take $O(1/\delta)$ rounds (where $\delta > 0$) in $O(n^{\delta})$ space per machine, and $O(|N|)$ total space (N is the input and $|N|$ is its size):

(1) MPC-Sort(N): sorts a set N of elements,
(2) MPC-Count(N): counts the number of elements in the multiset N,
(3) MPC-Duplicate(N): duplicates the elements in N,
(4) MPC-Filter(N, M): returns the set of elements that are in *both* N and M,
(5) MPC-CountDuplicates(N, i): given a list of tuples N, counts the number of copies of each distinct element in the multiset consisting of the element in the i-th index of each tuple; returns a sorted list of tuples $(element, count)$ where *element* is a distinct element and *count* is its count, and
(6) MPC-TagCount(N, D): given a sorted list N of constant-sized tuples and list of tuples $(element, count)$ in D (where every element in any tuple in N is counted in D); tag each element in each tuple in N with its count in D.

Algorithm 1 $O(1)$-Round Exact Triangle Counting

Input: Graph $G = (V, E)$, constant $\delta > 0$.
Output: An exact count of the number of triangles in G in $O(1)$ rounds, $O(n^{\delta})$ space per machine, and $O(m\alpha)$ total space.
1: $E' \leftarrow$ MPC-Duplicate(E).
2: $SortedE' \leftarrow$ MPC-Sort(E') by both endpoints of edges in E'.
3: $D \leftarrow$ MPC-CountDuplicates($SortedE', 0$) returning degree vector of each vertex.
4: $TaggedSortedE' \leftarrow$ MPC-TagCount($SortedE', D$).
5: **for** edge $e = (u, v) \in E$ **do**
6: Let $w \leftarrow \arg\min(\deg(u), \deg(v))$.
7: Split $SortedE'[w]$ into size n^{δ} partitions: $P_{e,1}, \ldots, P_{e,\lceil \deg(w)/n^{\delta} \rceil}$.
8: Send e and each partition $P_{e,i}$ to a separate machine.
9: **for** each machine M containing an edge e and partition $P_{e,i}$ of edges **do**
10: Form wedges using e and each edge in $P_{e,i}$.
11: **for** each wedge (a, b, c) **do**
12: Construct query $((a, c), e, M)$.
13: Let Q be the set of all queries.
14: $T \leftarrow$ MPC-Filter(Q, E).
15: Return $(T, |T|/3)$.

First, we assume that each vertex is assigned a unique index in $[n]$. Our algorithm first finds the adjacent edges to each vertex by duplicating each edge in Line 1 and then sorting the edges by both endpoints (Line 2). In other words, for each edge $e = (u, v)$ and its duplicate $e' = (u, v)$, we sort e by u and e' by v. Then, we count the number of edges adjacent to each vertex in Line 3 which allows us to obtain the endpoint with smaller degree for each edge. Let D be a vector containing the degree of each vertex. We then use our degrees stored in D to tag each edge in $SortedE'$ with its degree (Line 4). Then, for each edge (Line 5), we find the endpoint with smaller degree in Line 6 using $TaggedSortedE'$. For the endpoint w with smaller degree, we split the adjacency list for w into chunks of size n^{δ} (Line 7). Let these partitions be $P_{e,1}, \ldots, P_{e,\lceil \deg(w)/n^{\delta} \rceil}$. Then, we send each e and a partition $P_{e,i}$ to a separate machine (Line 8). Partitioning can be done in $O(1/\delta)$ MPC rounds by first counting the number of times c_u each vertex u is the lower degree endpoint, then duplicating each $\{u, v\}$ edge $c_u + c_v$ times and tagging the duplicate with a unique $i \in [c_u]$ or $j \in [c_v]$; finally, we sort all (duplicated) edges by their smaller degree endpoint and tag, and partition the sorted list.

For each machine which contains an edge e and corresponding partition $P_{e,i}$ (Line 9), we form paths of vertex length 3 (2 edges each), otherwise known as ***wedges***, between e and each edge in $P_{e,i}$ (Line 10). For each wedge (a, b, c) where b is the middle vertex (Line 11), the wedge is a triangle if edge (a, c) exists in the graph. Thus, we form a query for edge (a, c) (Line 12). We tag the query with the edge e and the machine M to distinguish between different queries for the same edge. We define Q to be the set of all queries (Line 13). We then determine the set of queries which are existing edges by using the MPC filter primitive (Line 14). We use the filter to filter all queries $((a, c), e, M)$ where (a, c) is an edge. The filter primitive can be implemented by sorting all queries together with

the actual edges. Then, if the count of all sorted elements that contain (a, c) is greater than the number of queries for (a, c), the edge (a, c) exists. The filter then returns all queries for which edge (a, c) exists. We let this set of returned queries be T (Line 14). From (a, c) and e, one can enumerate the triangle. Finally, we return all filtered queries and the triangle count which is equal to the number of filtered queries divided by 3.

We now prove the number of rounds, space per machine, and total space used by Algorithm 1.

Theorem 3.2. *Algorithm 1 outputs the set of triangles and triangle count using $O(1/\delta)$ MPC rounds, $O(n^\delta)$ space per machine for any constant $\delta > 0$, and $O(m\alpha)$ total space where α is the arboricity of the graph.*

Proof. We first prove that our algorithm enumerates the set of triangles in the input graph. For each edge e, we enumerate all wedges formed using e and the adjacency list of the smaller degree endpoint. For each triangle, (a, b, c), three wedges are enumerated, one initiated by each of the edges in the triangle. Then, for each wedge, we check whether the edge that completes the wedge into a triangle exists in the graph. If it exists, then the wedge is returned as a query that is a triangle. The triangle can be obtained from (a, c) and e, both contained in the query. Since each edge of (a, b, c) initiates the formation of exactly one wedge, the set of enumerated queries is exactly three times the number of distinct triangles.

We now show that our algorithm takes $O(1/\delta)$ MPC rounds, $O(n^\delta)$ space per machine for any constant $\delta > 0$, and $O(m\alpha)$ total space where α is the arboricity of the graph. All of our primitives satisfy these measures. We call a constant number of primitives; hence the number of rounds necessary to run all of the primitives is $O(1/\delta)$. Then, sending e and the associated partitions each into a separate machine takes one round. Since the adjacency list is partitioned into n^δ sized chunks, the total space necessary per machine is $O(n^\delta)$. Finally, $O(n^\delta)$ queries are formed per machine. And by Lemma 3.1, at most $O(m\alpha)$ total queries are formed. Thus, the total space usage is $O(m\alpha)$, the number of rounds is $O(1/\delta)$, and space per machine is $O(n^\delta)$ for any constant $\delta > 0$. □

3.1 Lower Bound on Number of Rounds

In this section, we show that our number of rounds is optimal. There exists a simple lower bound of $\Omega(1/\delta)$ rounds for triangle counting when the initial graph is partitioned across multiple machines. Suppose we have multiple disjoint subgraphs partitioned across multiple machines. We show that in this worst case setting, computing the number of triangles in the input graph requires $\Omega(1/\delta)$ rounds. This means that our bounds are tight up to constant factors.

Lemma 3.3. *Counting the number of triangles in an input graph with m edges and $n < m$ vertices requires $\Omega(1/\delta)$ rounds when the space per machine is n^δ and total space is $m\alpha$.*

Proof. Suppose that each of m/n^δ machines contains the edges of a disjoint subgraph with number of triangles unknown to the other machines. Thus, in order to obtain the triangle count, we must aggregate the counts of the triangles on each individual machine to one machine. Since each machine has space n^δ, each machine can receive only up to n^δ counts from other machines. The problem of aggregating the counts onto one machine then reduces to the problem of constructing constructing a tree with m/n^δ leaves and where every internal node has degree at most n^δ. The height of the tree is then the minimum number of rounds required by any triangle counting algorithm. Such a tree must have height $\log_{n^\delta}(m/n^\delta) = \log_{n^\delta}(m) - 1 = \Omega(1/\delta)$. □

4 OPEN QUESTIONS

The remaining open question is to show that counting the number of triangles, even when we allow for a $(1+\varepsilon)$-approximation, can be done in $O(1)$ rounds in near-linear or sublinear space per machine, without any assumptions on the minimum number of triangles that are present in the graph and without sparsity assumptions for the graph. For very dense graphs, one can potentially use matrix multiplication techniques. Thus, the interesting setting is when the graph has arboricity $\alpha = \omega(\text{poly}(\log n))$ but has $o(n^2)$ edges. Another interesting question is to extend our results to other types of subgraphs beyond triangles.

A PROOF OF LEMMA 3.1

Proof. Consider the arboricity decomposition of graph $G = (V, E)$. Let $F_1, \ldots, F_\alpha$ be the α forests of the decomposition. Suppose we pick an arbitrary root for each tree in each forest F_i. Then, we orient the edges from the root to the children (toward the leaves). We assign each edge to the node that it is oriented towards. Then, each node in forest F_i has at most one edge assigned to it. We denote the vertex that edge e is directed towards by $to(e)$. Then, we can show that the sum of the minimum degrees is as follows:

$$\sum_{(u,v)\in E} \min(\deg(u), \deg(v)) \leq \sum_{1\leq i\leq \alpha} \sum_{e\in F_i} \deg(to(e)) \quad (1)$$

$$\leq \sum_{1\leq i\leq \alpha} \sum_{v\in V} \deg(v) \quad (2)$$

$$\leq 2m\alpha. \quad (3)$$

Eq. (1) follows because $\deg(to(e)) \geq \min(\deg(u), \deg(v))$ where $e = \{u, v\}$. Eq. (2) follows because each vertex has at most one edge associated with it; since each vertex has at most one edge associated with it, it holds that $\deg(to(e))$ of vertex $to(e)$ is counted at most once per forest F_i. □

REFERENCES

[1] Mohammad Al Hasan and Vachik S Dave. 2018. Triangle counting in large networks: a review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 2 (2018), e1226.

[2] Maryam Aliakbarpour, Amartya Shankha Biswas, Themis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. 2018. Sublinear-time algorithms for counting star subgraphs via edge sampling. *Algorithmica* 80, 2 (2018), 668–697.

[3] Shaikh Arifuzzaman, Maleq Khan, and Madhav Marathe. 2013. Patric: a parallel algorithm for counting triangles in massive networks. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 529–538.

[4] Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. 2019. A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling. In *ITCS (LIPIcs, Vol. 124)*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 6:1–6:20.

[5] Ziv Bar-Yossef, Ravi Kumar, and D Sivakumar. 2002. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 623–632.

[6] Paul Beame, Paraschos Koutris, and Dan Suciu. 2013. Communication Steps for Parallel Query Processing. In *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*. 273–284.

[7] Suman K Bera and Amit Chakrabarti. 2017. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *34th Symposium on Theoretical Aspects of Computer Science*.

[8] Amartya Shankha Biswas, Talya Eden, Quanquan C. Liu, Ronitt Rubinfeld, and Slobodan Mitrovic. 2022. Massively Parallel Algorithms for Small Subgraph Counting. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference) (LIPIcs, Vol. 245)*, Amit Chakrabarti and Chaitanya Swamy (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 39:1–39:28. https://doi.org/10.4230/LIPICS.APPROX/RANDOM.2022.39

[9] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and Subgraph Listing Algorithms. *SIAM J. Comput.* 14, 1 (feb 1985), 210–223. https://doi.org/10.1137/0214017

[10] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[11] Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. 2017. Julienne: A Framework for Parallel Graph Algorithms Using Work-efficient Bucketing. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 293–304.

[12] Laxman Dhulipala, Quanquan C. Liu, Julian Shun, and Shangdi Yu. 2021. Parallel Batch-Dynamic k-Clique Counting. In *SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*. 129–143.

[13] Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. 2017. Approximately counting triangles in sublinear time. *SIAM J. Comput.* 46, 5 (2017), 1603–1646.

[14] Talya Eden, Dana Ron, and C. Seshadhri. 2020. Faster sublinear approximation of the number of k-cliques in low-arboricity graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*. 1467–1478. https://doi.org/10.1137/1.9781611975994.89

[15] Michael T. Goodrich and Paweł Pszona. 2011. External-Memory Network Analysis Algorithms for Naturally Sparse Graphs. In *European Symposium on Algorithms*. 664–676.

[16] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. 2011. Sorting, Searching, and Simulation in the Mapreduce Framework. In *Proceedings of the 22Nd International Conference on Algorithms and Computation* (Yokohama, Japan) *(ISAAC'11)*. Springer-Verlag, Berlin, Heidelberg, 374–383. https://doi.org/10.1007/978-3-642-25591-5_39

[17] Sungjin Im, Ravi Kumar, Silvio Lattanzi, Benjamin Moseley, Sergei Vassilvitskii, et al. 2023. Massively Parallel Computation: Algorithms and Applications. *Foundations and Trends® in Optimization* 5, 4 (2023), 340–417.

[18] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS operating systems review*, Vol. 41. ACM, 59–72.

[19] Daniel M Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. 2012. Counting arbitrary subgraphs in data streams. In *International Colloquium on Automata, Languages, and Programming*. Springer, 598–609.

[20] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A model of computation for MapReduce. In *SODA*. 938–948.

[21] Tamara G Kolda, Ali Pinar, Todd Plantenga, C Seshadhri, and Christine Task. 2014. Counting triangles in massive graphs with MapReduce. *SIAM Journal on Scientific Computing* 36, 5 (2014), S48–S77.

[22] Longbin Lai, Lu Qin, Xuemin Lin, and Lijun Chang. 2015. Scalable subgraph enumeration in mapreduce. *Proceedings of the VLDB Endowment* 8, 10 (2015), 974–985.

[23] Andrew McGregor, Sofya Vorotnikova, and Hoa T Vu. 2016. Better algorithms for counting triangles in data streams. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 401–411.

[24] Rasmus Pagh and Charalampos E Tsourakakis. 2012. Colorful triangle counting and a mapreduce implementation. *Inform. Process. Lett.* 112, 7 (2012), 277–281.

[25] Ha-Myung Park, Francesco Silvestri, U Kang, and Rasmus Pagh. 2014. Mapreduce triangle enumeration with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. 1739–1748.

[26] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. 2018. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowledge and Information Systems* 54, 3 (2018), 677–710.

[27] Julian Shun and Kanat Tangwongsan. 2015. Multicore triangle computations without tuning. In *2015 IEEE 31st International Conference on Data Engineering*. 149–160. https://doi.org/10.1109/ICDE.2015.7113280

[28] Siddharth Suri and Sergei Vassilvitskii. 2011. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th international conference on World wide web*. 607–614.

[29] Tom White. 2012. *Hadoop: The definitive guide.* " O'Reilly Media, Inc.".

[30] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.