TC-$Stream$: Large-Scale Graph Triangle Counting on a Single Machine Using GPUs

Jianqiang Huang, *Member, IEEE*, Haojie Wang, *Member, IEEE*, Xiang Fei, *Member, IEEE*, Xiaoying Wang, *Member, IEEE*, and Wenguang Chen

Abstract—In this paper, we build a TC-$Stream$, a high-performance graph processing system specific for a triangle counting algorithm on graph data with up to tens of billions of edges, which significantly exceeds the device memory capacity of Graphics Processing Units (GPUs). The triangle counting problem is a broad research topic in data mining and social network analysis in the graph processing field. As the scale of the graph data grows, a portion of the graph data must be loaded iteratively. In the existing literature, graphs with billions of edges need to be done distributively, which is cost-intensive. Also, many disk-based triangle counting systems are proposed for CPU architectures, but their tackling performances are inefficient. To solve the above problem, we propose TC-$Stream$, and it focuses on three issues: 1) For power-law graphs, because the amount of tasks of each vertex or edge is inconsistent, it is bound to cause different demands of computing and memory resources for different task types. We propose a parallel vertex approach and the reordering of vertices for graph data that can be placed in the GPU device memory to ensure the maximum workload balancing; 2) A binary-search-based set intersection method is designed to achieve the maximum parallelism in GPU; 3) For the graph data that exceeds the GPU device memory capacity, we develop a novel vertical partition algorithm to guarantee the independent computing on each partition so that the three computation processes, i.e., the computation on GPU, the data transmission between main memory of CPU and SSD, and the communication between the CPU and the GPU can be perfectly overlapped. Moreover, the TC-$Stream$ optimizes edge-iterator models and benefits from multi-thread parallelism. Extensive experiments conducted on large-scale datasets showed that the TC-$stream$ running on a single Tesla V100 GPU performs $2.4-6\times$ and $1.8-4.4\times$ faster than the state-of-the-art single-machine in-memory triangle counting system and GPU-based triangle counting system, respectively, and achieves $2.4\times$ faster than the state-of-the-art out-of-core distributed system PDTL running on an 8-node cluster when processing the graph data with 42.5 billion edges, which demonstrates the high performance and cost-effectiveness of the TC-$Stream$.

Index Terms—Triangle counting, vertical partition, out-of-core, GPU, parallel processing

1 Introduction

Graph data plays a critical role in many fields in the big data area, and the sparse patterns can be used to reveal hidden patterns in complex data. For example, linked data in various fields such as social networks, bioinformatics, web pages, road networks, and brain neural networks can generally be stored in the form of graphs. With the growth of graph data scales, the road network graphs often contain hundreds of millions of edges, the social network graphs contain hundreds of billions of edges, while the web page graphs contain trillions of edges. With such a large graph size, it is difficult to count the number of triangles quickly. Triangle Counting (*TC*) and Clustering Coefficient are often used together in graph neural networks [1], [2], [3], [4]. Graph pattern mining [5], [6], triangle counting algorithm has been widely used in fraud detection, community discovery [7], and in clusters in social and data mining [8]. At the same time, the rapid development of the general-purpose GPU (GPGPU) has brought about a revolution in many fields of computing [9], [10], [11]. From Table 1, we can see that massive cores and large memory bandwidth make the GPU become a graph data mining algorithm with high time complexity, such as accelerating triangle counting, which brings new hope.

Existing works have achieved great success in accelerating graph processing on GPU, such as Breadth-First Search (BFS) [12], [13], PageRank [14], Graph Label Propagation [15], etc. The GPU has a higher acceleration ratio than the CPU in the triangle counting algorithm for graph data when a data size fits into a GPU device's memory capacity. However, this does not come for free. For graph data whose size exceeding the GPU device memory capacity, costly data transfer between the CPUs and GPUs is inevitable [16], [17], [18], [19], [20], [21], and the speed of which is limited by the maximum bandwidth of PCIe (16GB/s). The device memory capacity is fixed in PCB and cannot be expanded

- *Jianqiang Huang, Haojie Wang, and Xiang Fei are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: {hjq16, feix16}@mails.tsinghua.edu.cn, wanghaojie@tsinghua.edu.cn.*
- *Xiaoying Wang is with the Department of Computer Technology and Appications, Qinghai University, Qinghai 810016, China. E-mail: wangxiaofu163@163.com.*
- *Wenguang Chen is with the Department of Information Technology Center, Qinghai University, Qinghai 810016, China. E-mail: cwg@tsinghua.edu.cn.*

Manuscript received 31 Aug. 2021; revised 29 Oct. 2021; accepted 30 Nov. 2021. Date of publication 14 Dec. 2021; date of current version 23 May 2022.
This work was supported by the National Natural Science Foundation of China under Grants 62062059, 62162053, 61762074, and in part by National Natural Science Foundation of Qinghai Province under Grant 2019-ZJ-7034, "Qinghai Province High-end Innovative Thousand Talents Program-Excellent Talents," The Open Project of State Key Laboratory of Plateau Ecology and Agriculture, Qinghai University under Grant 2020-ZZ-03.
(Corresponding author: Wenguang Chen.)
Recommended for acceptance by A. J. Peña, M. Si and J. Zhai.
Digital Object Identifier no. 10.1109/TPDS.2021.3135329

1045-9219 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Charles University. Downloaded on June 16,2022 at 08:37:13 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Hardware Features of the GPUs Used for Performance Measures

	P100 PCIE 16GB	Titan X Pascal	Tesla V100 16GB	Tesla K40	Tesla 1080Ti
Architecture	Pascal	Pascal	Volta	Kepler	Pascal
Cores/SMs	3,584 / 56	3,584 / 28	5120/80	2,880 / 15	3,584 / 28
Memory	16GB	12GB	16GB	12GB	11GB
Bandwidth	732GB/s	480GB/s	900GB/s	388GB/s	388GB/s
Max clock	1.33GHz	1.53GHz	1.45GHz	1.11GHz	1.1GHz

like RAM architecture. The most advanced GPU device memory size is 32GB, corresponding to the storage of 8 billion edges. Some works research out-of-core strategy [22], [23], [24] that divides data into blocks and overlaps the data transfer with GPU computation to hide the communication overhead, the multi-GPU strategy [25], [26], [27] as well as the distributed GPU cluster strategy [28], [29]. How to partition to overlap the calculation and data transfer is a challenge.

Since triangle counting (*TC*) algorithm has been widely used in fraud detection, community discovery [7], clusters in social and data mining [30], many researchers have studied the *in-memory* systems [31], [32], [33], [34], [35], [36], [37], *out-of-core* systems [38], [39], [40], [41], [42] and the approximate counting systems [43], [44], [45], [46], [47] that support triangle counting algorithm. However, these systems are all optimized for the CPU architecture. In recent years, the GPU heterogeneous computing platforms take advantage in terms of computational performance, memory bandwidth, and energy efficiency compared to the general-purpose CPU computing platforms, which are gradually being widely used in many general-purpose computing fields, and also provide opportunities in the processing of large-scale graph data with efficiency. The existing single-node GPU-based triangle counting systems [48], [49], [50] cannot process large scale graph data due to the limited GPU device memory capacity. In contrast, the distributed ones [51], [52], [53], [54] need to use a large number of servers, which is cost-intensive. MapReduce-based graph processing [23] first propose an out-of-core GPU data management technique that can process large-scale graph data due to the limited GPU device memory capacity, GraphReduce [24] adopts a combination of edge- and vertex-centric implementations, achieved state-of-the-art out-of-core GPU implementation.

On the CPU architecture, the state-of-the-art approach [33] is CPU-aware merge-based method [37], [55], [56], [57], [58]. The intersection algorithm uses the CPU Single Instruction Multiple Data(SIMD) instructions to accelerate the set intersections in graph algorithms. Unfortunately, due to the different architecture of the GPU and the CPU, the same design brings non-trivial technical challenges to the GPUs. The fundamental strategy for dealing with graphs on the GPUs is to distribute neighbor vertex of the adjacency list evenly into a balanced workload spread across multiple threads. That is different from the triangle count on CPU because the GPUs kernel cannot actively call a thread from the CPUs memory to the GPUs memory copy via the GPU threads. In contrast, all the GPU memory copying operations must be done before computation can be performed on the GPU. There are three main challenges in designing a system that supports high performance based on *out-of-core* storage (such as SSD) and GPU: (1) during vertex parallelism, and the traditional parallel approach will evenly distribute all vertices to each thread. Due to the *power-law* distribution of the graph, the workload for each block is not balanced. There comes the first challenge on how to design a dynamic scheduling algorithm for the load balance of each thread; (2) since the memory accesses of merge-based set intersection operations are mainly random accesses on the CPU, the second challenge is how to design a new GPU-aware set intersection approach; (3) for graph data with data size exceeding the GPU device memory capacity, how to develop a graph partitioning approach to overlap the GPU computing and the data transfer between CPU and GPU to achieve efficient parallelism becomes the third challenge.

To address these challenges, we develop the *TC-Stream*. Our main contributions are listed as follows:

- For *power-law* graphs skewed out-degree distribution. We propose a parallel vertex approach and the reordering of vertices for graph data that can be placed in the GPU device memory, which allows each block to acquire the new vertices counting by recording the newest vertex and the offset to ensure the load balance of each thread and improve the parallel efficiency.
- We devise a novel GPU-aware binary search tree and lookup tree optimizations to support an efficient memory access pattern that is more suitable for the GPU architecture so that the in-warp parallelism on the GPU can be maximized.
- We design a Vertical Partition approach, which makes the computing on each block independent. Then we perform a compressed sparse row (CSR) for each partition so that the computation on GPU, the data transmission between main memory of CPU and SSD, and the communication between the CPU and the GPU can perfectly overlap.

Our system *TC-Stream* is evaluated on real-world graphs, using a Tesla V100 GPU. The result shows that our system is $2.4-6\times$ and $1.8-4.4\times$ faster when compared with the current state-of-the-art single-machine in-memory CPU-based system [33] and single-GPU system [24], [48], respectively. Furthermore, *TC-Stream* performs $2.4\times$ faster than the out-of-core CPU system PDTL running on eight nodes when processing the graph with 42.5 billion edges, which provides low cost and high efficiency.

The rest of the paper is organized as follows: Section 2 gives the Background and the Related Work. Section 3 describes the motivation of *TC-Stream*. Section 4 presents the main challenges and optimization techniques for *TC-Stream*.

Authorized licensed use limited to: Charles University. Downloaded on June 16,2022 at 08:37:13 UTC from IEEE Xplore. Restrictions apply.

Section 5 evaluates the performance of our policy, and Section 6 concludes the paper.

2 Background and Related Work

In this section, we survey some relevant studies. First, we review the literature on GPU-based graph processing. Second, we review the existing methods for triangle counting.

2.1 GPU-Based Graph Processing

With the development of GPGPU technology, the GPU has naturally become an option in graph computing acceleration [59], [60], [61]. As a standard general processor, the CPU not only sets a certain number of computing units but also focuses on the design of complex cache systems, branch prediction systems, and various control logic, which will bring extra overhead. By contrast, the GPU uses most of the transistors on the chip as pure computing units. Such an architectural design determines that the GPU has a strong computing capacity, especially for the computing tasks suitable for the SIMD parallel model [62], [63], [64], [65], [66], [67], [68]. Since the graph has many vertices, many edges, and complex data dependence, and most graph vertex processing tasks have the same execution function, then the graph data processing tasks are very suitable for the GPU acceleration.

Medusa[69] system has designed a set of API interfaces, and users can easily use the GPU to accelerate chart processing. Medusa defines the basic data structure, adopts the CAA storage model, integrates thread access to access address, and reduces I/O overhead. The system still adopts the synchronous computing model, based on the NVIDIA-CUDA architecture, which simplifies the GPU programming process. However, Medusa did not design a data partition strategy and directly transferred all the data into the device memory for processing. In this way, the size of data processing is limited by the size of the GPU device memory. No attempt was made to solve the problem of low task concurrency during the GPU accelerated processing. CuSha[70] system is an open-source CUDA-based GPU graph processing system, which designs two new data structures to overcome the low efficiency of parallel access caused by traditional CSR structure. The two data structures are G-shards and Concatenated Windows (CW) format, respectively.

Besides, some work in the GPU graph computing system solves the problem of computing load imbalance by setting the concept of virtual Warp, which allows the nodes that compute the negative load to be executed by virtual Warp, which can be composed of 32 or even 64 threads to simultaneously perform the same computing task, to disperse the computing tasks with the negative load. In terms of implementing the hybrid system, based on the BFS algorithm, Hong's work has designed a kind of CGC execution model. The task in the early stage of the algorithm is performed by the CPU, which involves few vertices. If the responsibility is handed to the GPU, it cannot fully exert its concurrency capacity. In the middle of the algorithm, tasks involving more vertices and higher concurrent demands are transmitted to the GPU for execution. In the finishing part of the algorithm, the few functions in the final section are performed back in the CPU. However, the GPU graph computing system is mainly based on the BSP synchronous processing model, and communication overhead severely impacts GPU performance. It fails to solve the problem of processing graph data exceeding the GPU memory size efficiently. This work proposes a Vertical partition approach to make the GPU computing and I/O overlap to achieve efficient parallelism. The Vertical Partition is not limited to triangle counting since its computation pipeline generalizes to a wide range of parallel graph algorithms on the GPUs, such as Clique, Motif Counting (MC), and Frequent Sub-graph Mining (FSM).

2.2 Existing Triangle Counting Models

The standard method to solve the triangle counting problem is to intersect two adjacency-list of two adjacent vertices, and the algorithm has four views: vertex-centric (Algorithm 1), edge-centric (Algorithm 2), merge-based, and hash-map-based (Algorithm 3).

Algorithm 1. Vertex-centric: Each Thread Works on one Vertex

1: $G = (E, V)$;
2: **for** $u \in V$**inparallel do**
3: **for** $v \in N(u)$ **do**
4: $count+ = Intersection(u, v)$;
5: **end for**
6: **end for**

Algorithm 2. Edge-Centric: Each Thread Works one Edge

1: $G = (E, V)$;
2: **for** $(u, v) \in E$**inparallel do**
3: $count+ = Intersection(u, v)$;
4: **end for**

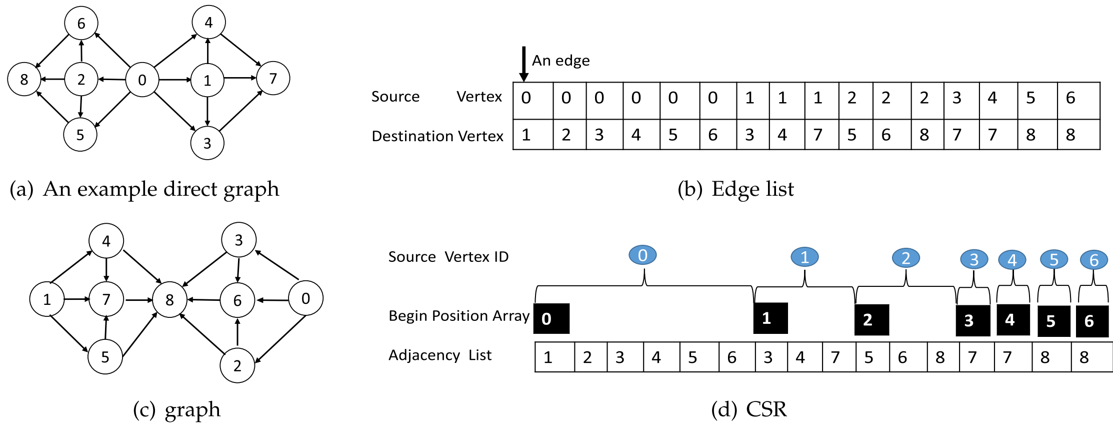
We adopt the following measures to insure that each triangle will be counted only once: first, calculate each vertex degree, denoted by $deg(v)$; second, sort $deg(v)$ and get an increasing sequence, $order(v)$, to ensure that $deg(v) <= deg(u)$ if $order(v) < order(u)$; third, delete the repeating and self-looping edges using $order(v)$ to make a new adjacency list, which guarantees that each triangle is counted only once. Let $N_+(v)$=$u|u \in N(v)$ and $order(v) < order(u)$.

$$T(u, v) = intersect(N_+(u) \cap N_+(v)). \quad (1)$$

We can then conclude that for each $(u, v) \in E$, the number of element in $T(u, v)$ is the number of triangles that covered $edge(u, v)$. Obviously, the number of all triangles in $G(V, E)$ is $\sum_{v \in V} \sum_{u \in N_+ v} T(u, v)$ (1), and the time complexity of this algorithm is $O(dE)$, where $|E|$ is the total number of edges, while d is the average time complexity to compute $N_+(u) \cap N_+(v)$, which can be equal or less than the average number of the two sets.

Merge-based approach: it's an efficient method in the CPU that solves the intersection using the merge-based process. Two points are needed when we compute the intersection of two adjacency vertices from $edge(0, 1)$. They point to the first element of the sets, respectively, at the beginning. When the part is the same in two groups, we find a new triangle, adding 1 for two points. Otherwise, only the point

Authorized licensed use limited to: Charles University. Downloaded on June 16,2022 at 08:37:13 UTC from IEEE Xplore. Restrictions apply.



(a) An example direct graph (b) Edge list (c) graph (d) CSR

Fig. 1. Direct graph representation in external storage.

with a minor element needs to add 1. In a word, the time complexity of the merge-based algorithm is $\sum_{v \in V} d^2(v)$

Hash-map-based approach: Algorithm 3 describes the pseudo-code of the hash-map-based algorithm. Suppose all the vertices are sorted in increasing order by their IDs, then we use the hash-map to record the adjacency of the vertex v_i, and the bitwise AND operator to find quickly. The lower M bits of vertex ID is the hash value. This will produce a hash-map-base with a size of 2^M.

Algorithm 3. Hash-map-based Triangle Counting Approach

```
1: procedure TC-MAPBASED(G(V, E^U))
2:   tc ← 0          ▷ Initialize triangle count in G
3:   for i ∈ {1, ..., msz} do ▷ msz is the size of Map
4:     Map[i] ← 0
5:   end for
6:   for all v_i ∈ V do
7:     for all v_j ∈ N_+(v_i) do
8:       Map.hash(v_j)
9:     end for
10:    for all v_j ∈ N_+(v_i) do
11:      for all v_k ∈ N_+(v_j) do
12:        if Map.exists(v_k) then
13:          tc ← tc + 1
14:        end if
15:      end for
16:    end for
17:    for all v_j ∈ N_+(v_i) do
18:      Map.delete(v_j)
19:    end for
20:  end for
21:  return tc       ▷ Total triangle count in G
```

Summary. The existing graph processing system is optimized for the triangle counting algorithm, and it achieves a satisfactory performance. The Vertical partition proposed in this paper is orthogonal to the two-dimensional division. Since we focused on guaranteeing the independent computing on each partition effectively, computing on the GPU, I/O from the SSD, and communication between the CPU and the GPU can be perfectly overlapped. In other words, the existing two-dimensional partition method can be used as the preprocessing step of the vertical partition.

3 Motivation of TC-Stream

In this section, we first introduce the Optimization of the edge-based Triangle Counting algorithms. Next, we present an overview of our *TC-Stream* system.

3.1 Graph Format on GPUs

A graph can be defined as $G = (V, E)$, where V is the set of all vertices and E is the set of all edges, while $|V|$ and $|E|$ are the number of vertices and edges, respectively. Fig. 1a is an example of a directed graph, Fig. 1b is the corresponding edge list, and Fig. 1d is the *CSR* format. As shown in Fig. 1a, vertex v_0, v_1, and v_3 constitute a triangle as any two of them have an edge to connect. The triangle counting problem is to find all the triangles in a given graph.

- The adjacency matrix needs $O(V^2)$ space to store a graph. Because the Scale-free property is too sparse in the adjacency matrix, i.e., wasting much storage space. Another way is to use a conventional adjacency list, which allocates a memory block for each vertex. However, it results in many allocations and releases operations, so the available memory is fragmented, reducing the operating efficiency.
- The edge list is also a common way to represent a graph. It stores a pair of source and destination vertices of an edge one by one, which occupies $O(2|E|)$ space. Since this method has no point operation, many existing graph frameworks adopted it, even though it uses a larger space.
- The *CSR* is a format that compresses the adjacency matrix by row. In an adjacency matrix, row K represents all the out-vertex from vertex K, containing many zeros. So we have a chance to compress them. *CSR* contains two arrays, idx and nbr. The nbr stores all the out-vertex from all the vertices one by one, and idx records the place of the first out-vertex of each vertex in nbr. In another words, $idx[i]$ and $idx[i+1]$ is the first and the last place in nbr that stores the all out-vertex of vertex i. The storage space of *CSR* format is $|V| + |E|$. Note that $|E|$ is one or two orders of magnitudes larger than $|V|$ in general.

Based on the locality and similarity of a graph, the neighbors of a vertex are mainly close to it topologically. Their

Authorized licensed use limited to: Charles University. Downloaded on June 16,2022 at 08:37:13 UTC from IEEE Xplore. Restrictions apply.

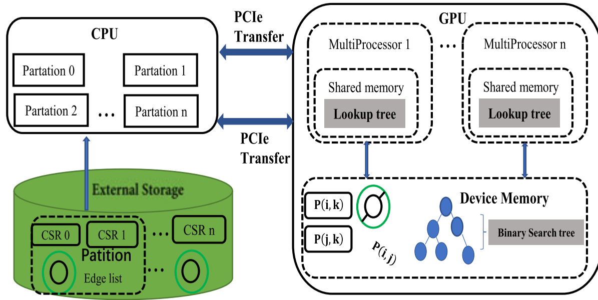


Fig. 2. $TC\text{-}Stream$ overview.

IDs are also close, so there is a high probability that all of them will be distributed into one vertex set. We find that a diagonal block, whose block row ID equals the block column ID, contains many edges, several orders of magnitudes more significant than the non-diagonal block.

3.2 Triangle Counting Algorithms

In Fig. 1a, we use $N_+(u)$ to represent the out-vertex of u. Thus , $N_+(0) = \{1,2,3,4,5,6\}$, $N_+(1) = \{3,4,7\}$, $N_+(2) = \{5,6,8\}$, $N_+(3) = \{7\}$, $N_+(4) = \{7\}$, $N_+(5) = \{8\}$, $N_+(6) = \{8\}$. The total degree of the whole graph is 16, and the total number of intersection is 48, as shown Algorithm 4 processing:

Algorithm 4. Edge-Iterator: Each Thread Works one Edge

Input: input parameters V, E , deg(vertex)
Output: output adjacency list N_+
1: $G = (E, V)$;
2: **for** $(u, v) \in E$ **do**
3: **if** $deg(u) > deg(v)$ or $(deg(u) == deg(v)$ and $u > v)$ **then**
4: $swap(u, v)$
5: $N_+(u) = N_+(u) \cup (v)$
6: **end if**
7: **end for**

$N_+(0) = \{2,3,6\}$, $N_+(1) = \{4,5,7\}$, $N_+(2) = \{6,8\}$, $N(3) = \{6,8\}$, $N(4) = \{7,8\}$, $N(5) = \{7,8\}$, $N(6) = \{8\}$, $N(7) = \{8\}$, total degree is 16, and the total number of intersection is 26.

3.3 $TC\text{-}Stream$ Overview

Fig. 2 describes the execution process between heterogeneous devices, with two components: a two-dimensional partitioning algorithm for external memory and scheduling based on CPU and GPU heterogeneity intersection. We evaluated the state-of-the-art GPU-based triangle counting project. First of all, the merge-based intersection bandwidth has a lot of device memory bandwidth. Therefore, we design a GPU-friendly intersection scheme to solve this problem. Because triangle counting is a computational-intensive workload, we develop a graph partitioning strategy and hide SSD-CPU-GPU data transfer overheads by overlapping computations on GPUs and SSD-CPU-GPU data transfers. To this end, we design the Vertical Partition approach of the model, which uses the data structure of the edge list and the CSR to store the original graph data in the SSD, respectively, and adopt the edge-based calculation model. Each partition is composed of a sub-edge list and the corresponding partial CSR. The triangle counting algorithm between divisions is independent. We found that the GPU of each MP of the Shared memory of 48KB can perform efficiently concurrent execution than operation. A set intersection method based on a binary search tree is designed on GPU: an adjacency list is used as the search table. An adjacency list acts as a binary search tree, and the binary search tree is then traversed to see if every element in the lookup table exists in the tree. At the same time, two sub-CSR blocks and one sub-edge block are loaded into GPU's video memory, and each sub-task can saturate GPU's computing capacity.

4 $TC\text{-}Stream$: Overlapped and Parallel Triangulation

The GPU-based external memory data processing is mainly divided into four stages: Data preprocessing in external memory (the original graph was transformed into a graph model represented by CSR and Edgelist, and the edge set was divided into blocks containing an equal number of edges (the number of incoming edges + outgoing edges), data loading in CPU (each edge set was divided into blocks and CSR data blocks were loaded into the main memory space), data transmission in CPU-GPU, and calculation in GPU. In the process of edge set block partitioning, the block size is determined according to the device access memory that each block can be stored entirely in the GPU. In the process of constructing the graph data system, in order to support the diversified graph data types, the uniform division of large-scale graph data, and the parallel processing of graph data, the uniform division of diversified large-scale graph data can effectively reduce the Straggler overhead problem in the parallel processing of graph data. Second, the GPU's external memory data transmission cost in parallel processing usually takes much running time. Therefore, in parallel data transmission and calculation, adopting overlapping round scheduling and asynchronous data transmission can significantly reduce the external memory data transmission and synchronization cost. In this paper, we propose a $TC\text{-}Stream$, which mainly solves three problems: A $TC\text{-}Stream$ uses a parallel vertex approach for graph data that can fit in the GPU device memory, dynamically allocating the vertices into different GPU blocks for parallel processing. Design a GPU-aware intersection scheme to maximize the GPU's computing and memory access capacity.

Partition the graph data that exceeds the GPU device memory, since the GPU device memory can be loaded with only one partition or most so that the partitions are independent in triangle counting algorithm, the $SSD- > CPU- > GPU$ overhead is relatively large. We design a method that makes it overlap to improve system performance.

4.1 Graph Vertical Partition

As for a 1-D partitioning[71], regardless of using an edge-centric(Algorithm 2) or a vertex-centric (Algorithm 1) scheme, the partitions will inevitably have the problem of data interdependence, which results in an incomplete overlap of the I/O and the CPU calculations. A significant problem we face is that the lousy partition will cause workload imbalance on different devices. On the other hand, we also must ensure that all the three vertexes of the triangle we count should be in the same partition. In this section, we will

Authorized licensed use limited to: Charles University. Downloaded on June 16,2022 at 08:37:13 UTC from IEEE Xplore. Restrictions apply.

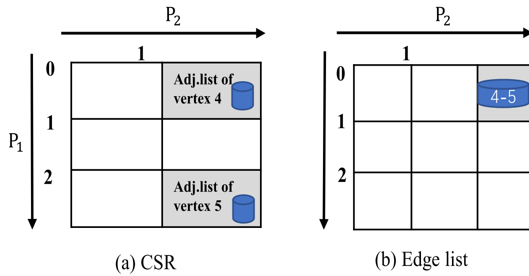


Fig. 3. Vertical partition management.

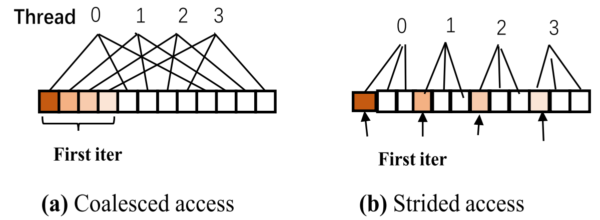


Fig. 4. Memory access patterns.

use the vertical partition. The vertical partitioning algorithm is used to divide the triangle counting task into multiple sub-tasks. Each sub-task can be regarded as a local triangle counting job on the sub-graph, and each partition consists of two sub-CSR blocks and one sub-edge block. Specifically, each partition needs to load two CSR chunks and one edge list into the GPU memory and then read the next one from a more significant external memory such as the disk, the solid-state drive (SSD), or the CPU memory. This is different from the CPU triangle counting because the GPU triangle counting kernel cannot actively call the threads of copying data from the CPU memory to the GPU memory through the GPU threads. Instead, all the GPU memory copy operations must be performed before the calculations are done on the GPU. Therefore, our algorithm scans the edge list to determine which CSR partition to load. Also, all the sub-tasks are independent of each other so that they can be calculated independently. However, the downside of this storage method is increasing the amount of the storage from CSR ($|V+E|$) to CSR and edge list ($|V+E|+2|E|$). Although each sub-task after vertical partitioning can be stored in the memory, the amount in the storage also brings a significant overhead to the transmission through the $SSD->CPU->GPU$ chain. The two problems are addressed as follows: First, we notice that the workload is related to the number of edges for the load balance problem, whether the algorithm is vertex-based or edge-based. However, taking the edge-based as an example, we can split the edge list into equal parts, but we cannot be sure that all of them will consume the same time because the neighbor list of each edge does not always have the same size. Second, the vertical partition can address this problem efficiently for the independent sub-task problem, as shown in Fig. 3. If there is an intersection, it must appear in the same column. Therefore, if the partition is vertical, the partition will contain all the intersections.

Fig. 3a depicts the vertical partition management process. First, do a 1-D vertical division to get P_2 vertical partitions. The block sizes of each vertical partition are then approximately equal by using different range boundaries for the vertices of each vertical partition. Then horizontally divide each vertical partition so that each horizontal partition will also have a diverse vertex range. For example, the solid line divides the first horizontal partition of each vertical partition. Each Vertical Partition has a size no larger than M. Fig. 3b shows that the size of the final partition result is $P_1 \times P_2$. The set intersection task is divided into many small partitions. Assume that the number of partitions is $P_1 \times P_1 \times P_2$, much larger than the GPU's memory. Therefore, designing the pipelines of $I/O, CPU->GPU$ transport, and computation can help improve overall system performance.

As we did before, rank the graph first. However, one thing is different, and we need to collect the in-degree of vertex because we need to do a vertical partition. After we get the in-begin position array, we can use it to find the cut points. Unlike the traditional 2-D graph partition strategy (partition by source and destination vertices), *TC-Stream* is suitable for multi-layer memory storage of GPU nodes and can maximize each memory level's storage utilization rate and execution efficiency.

4.2 GPU-Friendly Intersection Design

The advantages of binary-search-based intersection on GPU: (1)Finer-grained parallelization. No partition is needed for the imbalance issue because of the tree structure. When different threads search in the binary tree, the max depth of the tree is subject to the log-relation to the length of the longer CSR. (2) Using on-chip shared memory to optimize the performance. We cache top levels of the tree in shared memory as they are the most frequently accessed part.

The disadvantages of binary-search-based intersection on GPU: For the merge based algorithm, the time complexity is $O(d(u)+d(v))$. For the binary-search-based algorithm, $O(d(u) \times logd(v))$. When the $d(u) << d(v)$, the binary search based algorithm can perform better. However, when $d(u)$ and $d(v)$ are equally large, the case can become worse.

The inadaptability of the merge-based intersection of the CPU's and the GPU's architecture means that the GPU architecture is very different from CPU's architecture. The most advanced CPUs have dozens of cores, and each of them has its L1 cache running independently. Nevertheless, many cores form a warp group in the GPU and execute together in SIMD mode, sharing a small last-level cache. Warp threads are not as independent as the CPU cores, but they work together. Therefore, the traditional merge-based triangle counting algorithm is not suitable for GPUs.

As shown in Fig. 4, a small adjacency list fits in the Shared memory; the long adjacency list acts as a binary sort tree. For sets A and B, $|A|=m$, $|B|=n$ and $m<n$. Let i be the middle of the smaller set, i.e., $i=m/2$. The algorithm first searches for the element $A[i]$ in B, and if found, $A[i]$ will be the element in the intersection, and returns the position j in B (if not, return j to satisfy $B[j]<A[i] \leq B[j+1]$, where $B[0]=-\infty, B[n]=\infty$). Regardless of whether the search is successful or not, A will be divided into two parts, $A_l=A_1,\ldots,A_i-1$ and $A_r=A_i+1,\ldots,A_m$. Similarly, B is also divided into two parts, B_l and B_r; the above algorithm is then recursively applied to the left and right sets of subsets until any sub-set is empty, and the algorithm obtains and returns the intersection result.

For the binary search Algorithm 5, it goes to search for the central element of the target set each time so that for a

Authorized licensed use limited to: Charles University. Downloaded on June 16,2022 at 08:37:13 UTC from IEEE Xplore. Restrictions apply.

set of length n, $1 + logn$ locations will be probed in one lookup regardless of whether the search is successful, which is referred to as a Dynamic Probes method by the authors, for example, the number of comparisons $logn \ll n$ for a binary search. Therefore, when a set intersects, the number of similarities is reduced when a small set is used to search in an extensive collection. Another advantage of this method is that the smaller index created can be fully cached.

Algorithm 5. Backward Unbounded Search Algorithm

binary_rearch$(tag, list, len)$
1: $pos \leftarrow 2$
2: $len \leftarrow len - 1$
3: **while** $pos\ <= len$ **do**
4: **if** $tag = list[pos]$ **then**
5: **return** $len - pos$
6: **else if** $tag\ <\ list[pos]$ **then**
7: $pos \leftarrow pos \times 2 + 2$
8: **else**
9: **return** $binary_search(tag, list, len - pos + 1, len - pos \div 2)$
10: **end if**
11: **end while**
12: **return** $binary_search(tag, list, 0, len - pos \div 2)$

4.3 Overlapped Processing of the TC-Stream

As we mentioned in the last subsection, edge-based algorithms face the problem of overused memory. The GPU, which has limited memory, throw out of memory error when the graph is too large. Therefore, we come up with an edge buffer. The idea is that we can use a small buffer in the GPU to store the edge we are calculating. After each calculation has been done, load the next batch of edges into the buffer. In addition to that, we also plan to use double buffers to save the copy time of the buffer. The idea is because we notice that the copy time from the host memory to the device memory can spend a while. Therefore, we use two buffers to do this work. When a buffer is running on the GPU, the other buffer is copying the data. Since each partition brought by the vertical partitioning method is an independent, almost complete overlap of I/O, $CPU- > GPU$ transmission, then the calculation can be achieved. Each sub-task uses three suffix identifiers: i, j, and k, representing one partition in the vertical partition. Partition (i,j) represents a set of edges, called the base-partition, which determines which two vertex neighbors should be intersected. Then, two adjacency lists are selected from the partitions (i, k) and (j, k) respectively of the corresponding vertices intersect. Therefore, each sub-task can be scheduled independently on the GPU. However, using the partition (i, j) in CSR format as the base-partition does not provide fine-grained parallelism on the GPU. It can only provide vertex-centric coarse-grained parallelism, where each warp must handle all edges of a source vertex, resulting in an unbalanced workload within the GPU's warp. To this end, we design an edge-centric work division, using a vertical partitioned edge list as the base group. As shown in Fig. 3b, the edge list of the vertical partition contains the graph data in the edge list format, so it is easy to divide the work in the GPU. Each time to process an edge is to compute the intersection of the neighbor lists of the two vertices of the edge. The edge list of the vertical partition may not correspond to the CSR partition. This project uses the edge list partition as the base edge, so it is aligned with the horizontal partition of the corresponding CSR, a partition with $P_1 \times P_1$ size. This simplifies the representation of sub-tasks because the two sub-tasks never overlap. In addition, the I/O overhead is entirely flat in the stream buffer.

- Pre-process. When ranking by degree, we can collect the edge list in the meantime. ChunkNum means the number of buffer chunks.
- Load the Edge. For a single buffer, we can load the Edge in the following manner.
- Performance Establish. In the datasets with high IOs, regardless of whether the input data size is suitable for the GPU memory capacity, the use of double buffers provides the additional benefit of hiding the CPU-GPU data transmission from the intersection of the GPU.

5 Experiments

A *TC-Stream* is 3000 lines of code implemented using c++ and CUDA. We used the CUDA9.0 toolkit, NVCC, NVPROF, GCC 5.5.0, and OpenMP 3.1 to compile the source code.

5.1 Graph Data Benchmark

In our experiment, we use five real-world graphs and three synthetic graphs. The real-world graph is UK-2007, twitter-2010, Livejournal, Yahoo, clueweb12 [72], which contains the web pages in the UK field and its hyperlink information. We first fix the number of nodes for synthetic graphs and then randomly add the edges that do not form loops to generate the graph. We generated three synthetic graphs: Kronecker26 (scale 26, degree 16), Kronecker27 (scale 27, degree 16), and Rmat-27(scale 27, degree 16). Kronecker26, Kronecker27 are graphics that the GPU's device Memory can put down; Yahoo, UK-2007, and clueweb12 have data scales far beyond the GPU's device memory. The representation of graph data is in the form of a side table. It can be seen that when using binary files to save the graph data, the space use for saving is smaller than that using text files, which is about 40%. The average degree of vertices in the twitter-2010 graph and the UK-2007 graph are about 57.7 and 41.2, respectively, the average degree of vertices in the Yahoo graph is about 17.9, the average degree of vertices in the clueweb12 graph is about 46.9, and the average degree of vertices in the composite graph is 16. The maximum out-degree of the vertex of the social network graph is larger than the maximum in degree.

We evaluated the *TC-Stream* and preprocessing time of each state-of-the-art system using the applications described in the section and analyzed its performance on a selection of large graphs; Table 2 shows the preprocessing overhead of the *TC-Stream* Han, OPT, PDTL, Bisson, and Graphreduce on the SSD, which is capped at 500MB/s. Fig. 5 shows the preprocessing times for the TC-Stream in multicore speed up $18\times$ over a single core.).

Authorized licensed use limited to: Charles University. Downloaded on June 16,2022 at 08:37:13 UTC from IEEE Xplore. Restrictions apply.

TABLE 2
Real-World and Synthetic Graphs Data That are Used in Each State-of-the-art System's Experiments and Preprocessing Time(in Seconds)

Dataset	Vertex Num	Edge Num	Avg Deg	Triangle	Size(binary)	TC-Stream	Han	OPT	PDTL	Bisson	Graphreduce
LiveJournal	4.8M	68M	17.8	285M	0.52 GB	0.89	2.4	97.3	3.5	2.7	1.2
Twitter-2010	41.6M	1.5B	57.7	34B	11GB	17.9	24.3	57.7	401.1	33.7	32
UK-2007	134M	5.5B	41.2	286B	28GB	176	262	1876	191	*	343
Rmat-27	128M	2B	16	114B	32GB	128	317	2016	84	*	312
Kronecker-26	67M	1B	16	49B	8GB	18	26.8	62.4	352	47	36
Kronecker-27	134M	2B	16	106B	16GB	47	61.3	154	977	*	219
Yahoo	1.4B	6.6B	17.9	85.7B	49.4 GB	194	1012	2187	217	*	7189
clueweb12	978M	42.5B	46.9	1995B	318 GB	2871	*	9106	3026	*	11253

"" indicates that device memory is limited and failed to run successfully.*

5.2 Environment

The experiments used a server with an e5-2670@v3 processor, two sockets running at 2.3GHz, 32MB L3 cache, 12 cores per socket, and a disabled CPU hyper-threading feature. The server has 128 GByte of memory and 2TByte of the disk (SSD), and the operating system was 64-bit Ubuntu 16.04 LTS. We compared the performance of different types of GPUs(Volta V100, which has device memory of 16GB. Tesla P100, which has device memory of 16GB. GTX 1080Ti, which has a device memory of 11GB).

We compare the following five versions:

- Han: is the state-of-the-art intersection system based CPU [33].
- OPT: is the state-of-the-art triangle counting out-of-core system based CPU [40].
- PDTL: is the state-of-the-art triangle counting distributed out-of-core system based CPU [73].
- Bisson: is the state-of-the-art triangle counting in-memory system based GPU [48].
- GraphReduce: is the state-of-the-art out-of-core system based GPU [24].

5.3 Performance

5.3.1 In Device Memory GPU

Currently, the device memory of the high-end GPUs is 16GB. For placed in the GPU device memory (LiveJournal, twitter-2010, Kronecker-26(Kron-26), Kronecker-27(Kron-27)),

We compared the state-of-the-art in-memory GPU system [48] and the state-of-the-art CPU system [33], [40], respectively. From Fig. 6 and Table 3, it can be seen that Tesla V100 GPU is used to accelerate triangle counting, the *TC-stream* is $2.4\times$ and $3.5\times$ faster than Han [33] and OPT [40] on the twitter-2010 graph, respectively. The set intersection method of the binary sort tree is designed to maximize parallelism in the GPU. Using On-chip Shared Memory to optimize the performance, we cache top levels of the tree in the shared memory as they are the most frequently accessed part, the *TC-stream* is $5.6\times$, $6\times$, and $5.8\times$ faster than the state-of-the-art in-memory GPU system Bisson [48] on twitter-2010, Kronecker-26, Kronecker-27 graph, respectively.

Fig. 7 shows the triangle counting performance of the Tesla GPU and the GPU Pascal, the mainstream GPU in the market. V100 is $1.5\times$, $1.9\times$, $2.2\times$ and $3\times$ faster than P100, TiTanX, 1080Ti and K40 on the Twitter graph, because V100 GPU Tesla has a strong advantage in both the access bandwidth and the quantity of the SM, it can be seen from Table 1.

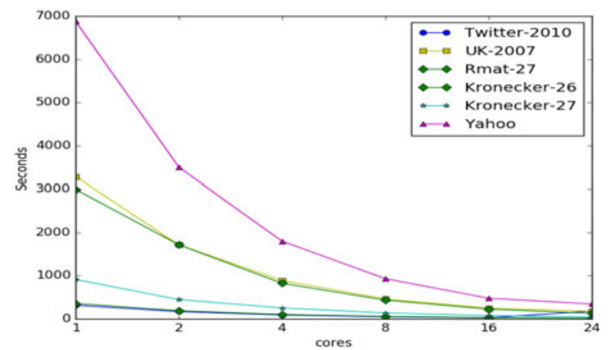


Fig. 6. Pre-processing times (in seconds) for *TC-Stream* in multicore.

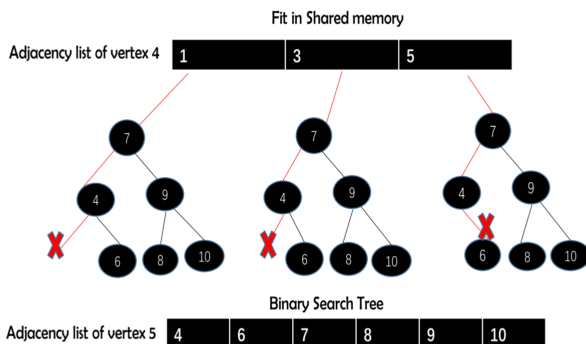


Fig. 5. Binary search-based intersection: Short adjacency lists fit in Shared memory, the long adjacency list acts as a binary sort tree.

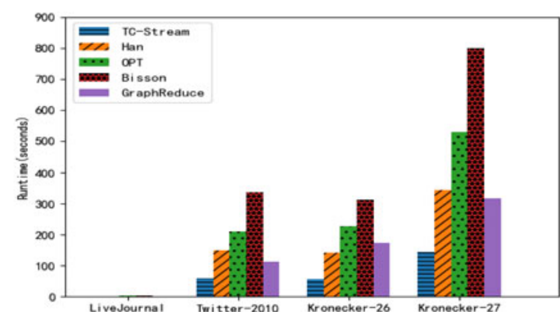


Fig. 7. Execution times for TC-Stream, Han, OPT, Bisson and Graphreduce.

Authorized licensed use limited to: Charles University. Downloaded on June 16,2022 at 08:37:13 UTC from IEEE Xplore. Restrictions apply.

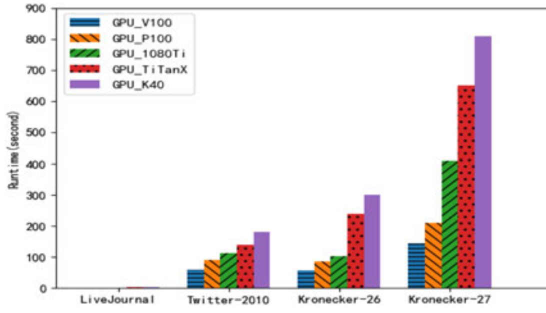


Fig. 8. Execution time of the run different GPUs.

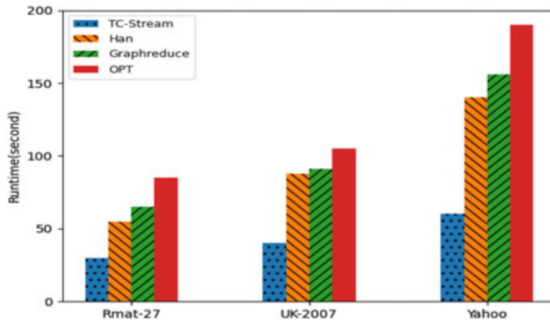


Fig. 9. Execution times for TC-stream, han, graphreduce, and OPT.

5.3.2 Exceeds the Device Memory Capacity GPU

Graph data is partitioned to fit GPU memory and moved from the host to the GPU, consisting of edges with destination or source vertices in some well-defined graph partition (see Section 4.1 for details). They reside in memory buffers that experience different access patterns. In terms of data movement, buffers can be divided into static and streaming buffers. Static buffers are copied to GPU memory only once, usually during initialization. An example is the set of vertices for a graph that fits into GPU memory. On the other hand, stream buffers are moved in and out of GPU memory as processing proceeds. At any point in time, a particular instance of a stream buffer resides in GPU memory.

Bisson [48] is an in-memory GPU-based system, which cannot run successfully on three data sets (Yahoo, UK-2007, Rmat27). For the graph data that exceeds the device memory capacity GPU, from Fig. 8, it can be seen that

TABLE 3
Runtime (s) of TC-Stream and Han [33], and GraphReduce [24], and OPT [40], and Bisson [48]

System	LiveJournal	Twitter-2010	Kron-26	Kron-27
TC-Stream	0.49	60	56	143
Han	1.3	148	141	343
OPT	2.9	210	227	528
Bisson	2.1	336	331	798
GraphReduce	1.6	112	173	316

A Tesla V100 GPU is used to test TC-Stream, Bisson [48], and GraphReduce [24], and a CPU with 24 threads for Han [33], and OPT [40].

Tesla V100 GPU is used to accelerate triangle counting, the *TC-stream* is 2.3×, 2.2×, and 1.9× faster than Han [33] on the Yahoo, UK-2007, and Rmat27 graph, respectively, the *TC-stream* is 2.8×, 2.6×, and 2.1× faster than OPT [40] on the Yahoo, UK-2007, and Rmat27 graph, respectively. Because we design a novel vertical partition algorithm that can ensure that the computing is in-dependent on each partition. each sub-task can load into the device memory, computing on the *GPU*, *I/O* from the *SSD* and the communication between the *CPU* and *GPU* can be perfectly overlapped. The *TC-stream* is 2.6×, 2.3×, and 2.2× faster than the state-of-the-art out-of-core GPU system GraphReduce [24] on Yahoo, UK-2007, Rmat-27 graph, respectively.

5.3.3 Comparison to Distributed System Implementation

We compare OPT(the state-of-the-art triangle counting out-of-core system based CPU [40]), Graphreduce(the state-of-the-art out-of-core system based GPU [24]), and the state-of-the-art distributed external storage graph processing the PDTL [73] on clueweb12 graph, respectively. From Table 4, it can be seen that Tesla V100 GPU is used to accelerate triangle counting. The *TC-stream* is 2.4×, 4.4× and 3.6× faster than the PDTL(8 machines), OPT, and Graphreduce on clueweb12 graph, respectively. Although it needs to move data between the GPU and CPU via PCIe, OPT and PDTL benefit from local (host) memory access. Tc-stream achieves significant acceleration, e.g., 2.4× on OPT, 4.4× on PDTL, 3.6× on Graphreduce, and ClueWeb12 for Triangle Counting processing. These performance improvements were due to its (i)the asynchronous model and leveraging CUDA Streams;(ii)Avoid unnecessary kernel startup and GPU kernel idling; (iii)And Computing on

TABLE 4
Comparison With Out-of-Memory Framework in Clueweb12

Framework	Hardware setting	machines	SSD- > CPU(Memory) time(s)	CPU- > GPU time(s)	Computation time(s)	Elapsed total time(s)
TC-Stream	1 GPUs, V100 16GB Device Memroy	1	1113	1706	1918	2075
PDTL	2 CPUs, 12 cores/CPU 128GB RAM	8	439	X	4743	4937
OPT	2 CPUs, 12 cores/CPU 128GB RAM	1	1119	X	8368	9406
Graphreduce	1 GPUs, V100 16GB Device Memroy	1	1126	1884	5324	7302

"X" indicates No transmission time. 'Out-of-memory' means that the input graphs neither fit into the limited GPU memory, nor fit into the limited CPU memory.

Authorized licensed use limited to: Charles University. Downloaded on June 16,2022 at 08:37:13 UTC from IEEE Xplore. Restrictions apply.

GPU, I/O from SSD, and the communication between the CPU and the GPU can be perfectly overlapped.

6 Conclusion

In this paper, we proposed the *TC-Stream*, a graph system that supports an accurate triangle counting algorithm on the graph data with up to tens of billions of edges, which significantly exceeds the device memory capacity of a single GPU card. Compared with the *state-of-the-art* CPU system, a single-card GPU system, and a distributed system, the *TC-Stream* has a good performance on both small-scale graph data that can be stored in the device memory capacity ($2.4 - 6\times$) and a large-scale graph data that exceeds the device memory capacity ($1.8 - 4.4\times$).

References

[1] S. Pandey, L. Li, A. Hoisie, X. S. Li, and H. Liu, "C-SAW: A framework for graph sampling and random walk on GPUS," in *Proc. SC20: Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2020, pp. 1–15.

[2] K. Yang, J. Zhu, and X. Guo, "POI neural-rec model via graph embedding representation," *Tsinghua Sci. Technol.*, vol. 26, no. 2, pp. 208–218, 2021.

[3] N. Yuvaraj, K. Srihari, S. Chandragandhi, R. A. Raja, G. Dhiman, and A. Kaur, "Analysis of protein-ligand interactions of SARS-CoV-2 against selective drug using deep neural networks," *Big Data Mining Analytics*, vol. 4, no. 2, pp. 76–83, 2021.

[4] K. K. Singh and A. Singh, "Diagnosis of COVID-19 from chest X-ray images using wavelets-based depthwise convolution network," *Big Data Mining Analytics*, vol. 4, no. 2, pp. 84–93, 2021.

[5] X. Chen, T. Huang, S. Xu, T. Bourgeat, C. Chung, and A. Arvind, "FlexMiner: A pattern-aware accelerator for graph pattern mining," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit.*, 2021, pp. 581–594.

[6] T. Shi, M. Zhai, Y. Xu, and J. Zhai, "GraphPi: High performance graph pattern matching through effective redundancy elimination," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2020, pp. 1–14.

[7] A. Prat-Pérez, D. Dominguez-Sal, J. M. Brunat, and J.-L. Larriba-Pey, "Shaping communities out of triangles," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manag.*, 2012, pp. 1677–1681.

[8] F. Zhang, J. Zhai, X. Shen, O. Mutlu, and X. Du, "POCLib: A high-performance framework for enabling near orthogonal processing on compression," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 2, pp. 459–475, Feb. 2021.

[9] NVIDIA, "Gpu-accelerated libraries for computing," 2021. [Online]. Available: https://developer.nvidia.com/gpu-accelerated-libraries/

[10] F. Zhang *et al.*, "TADOC: Text analytics directly on compression," *Very Large Data Bases Conf. J.*, vol. 30, no. 2, pp. 163–188, Mar. 2021. [Online]. Available: https://doi.org/10.1007/s00778–020-00636-3

[11] J. Huang, W. Xue, H. Bian, W. Yan, X. Wang, and W. Chen, "Helmholtz solving and performance optimization in global/regional assimilation and prediction system," *Tsinghua Sci. Technol.*, vol. 26, no. 3, pp. 335–346, 2021.

[12] P. Kumar and H. H. Huang, "G-store: High-performance graph store for trillion-edge processing," in *Proc. IEEE Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2016, pp. 830–841.

[13] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the GPU," in *Proc. 21st ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2016, pp. 1–12.

[14] H. Wu, D. Li, and M. Becchi, "Compiler-assisted workload consolidation for efficient dynamic parallelism on GPU," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 534–543.

[15] C. L. Staudt and H. Meyerhenke, "Engineering parallel algorithms for community detection in massive networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 1, pp. 171–184, Jan. 2015.

[16] C. Zhang, F. Zhang, X. Guo, B. He, X. Zhang, and X. Du, "iMLBench: A machine learning benchmark suite for CPU-GPU integrated architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1740–1752, Jul. 2021.

[17] S. Zhang, Y. Yang, L. Shen, and Z. Wang, "Efficient data communication between CPU and GPU through transparent partial-page migration," in *Proc. IEEE 20th Int. Conf. High Perform. Comput. Commun. IEEE 16th Int. Conf. Smart City IEEE 4th Int. Conf. Data Sci. Syst.*, 2018, pp. 618–625.

[18] F. Zhang, J. Zhai, B. He, S. Zhang, and W. Chen, "Understanding co-running behaviors on integrated CPU/GPU architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 905–918, Mar. 2017.

[19] J. Vesely, A. Basu, M. Oskin, G. H. Loh, and A. Bhattacharjee, "Observations and opportunities in architecting shared virtual memory for heterogeneous systems," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2016, pp. 161–171.

[20] Z. Yang, A. Zhang, and Z. Mo, "PsmArena: Partitioned shared memory for NUMA-awareness in multithreaded scientific applications," *Tsinghua Sci. Technol.*, vol. 26, no. 3, pp. 287–295, 2021.

[21] J. Mabrouki, M. Azrour, D. Dhiba, Y. Farhaoui, and S. E. Hajjaji, "IoT-based data logger for weather monitoring using arduino-based wireless sensor networks with remote graphical application and alerts," *Big Data Mining Analytics*, vol. 4, no. 1, pp. 25–32, 2021.

[22] M.-S. Kim, K. An, H. Park, H. Seo, and J. Kim, "GTS: A fast and scalable graph processing method based on streaming topology to GPUs," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 447–461.

[23] K. Shirahata, H. Sato, and S. Matsuoka, "Out-of-core GPU memory management for mapreduce-based large-scale graph processing," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2014, pp. 221–229.

[24] D. Sengupta, S. L. Song, K. Agarwal, and K. Schwan, "GraphReduce: Processing large-scale graphs on accelerator-based systems," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2015, pp. 1–12. [Online]. Available: https://doi.org/10.1145/2807591.2807655

[25] T. Ben-Nun, M. Sutton, S. Pai, and K. Pingali, "Groute: An asynchronous multi-GPU programming model for irregular computations," *ACM SIGPLAN Notices*, vol. 52, no. 8, pp. 235–248, 2017.

[26] Y. Pan, Y. Wang, Y. Wu, C. Yang, and J. D. Owens, "Multi-GPU graph analytics," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2017, pp. 479–490.

[27] C. Hong, A. Sukumaran-Rajam, J. Kim, and P. Sadayappan, "MultiGraph: Efficient graph processing on GPUs," in *Proc. IEEE 26th Int. Conf. Parallel Archit. Compilation Techn.*, 2017, pp. 27–40.

[28] Y. Zhang, X. Liao, H. Jin, B. He, H. Liu, and L. Gu, "Digraph: An efficient path-based iterative directed graph processing system on multiple GPUs," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2019, pp. 601–614.

[29] L. Ma *et al.*, "NeuGraph: Parallel deep neural network computation on large graphs," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 443–458.

[30] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis, "Efficient semi-streaming algorithms for local triangle counting in massive graphs," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2008, pp. 16–24.

[31] N. Sundaram *et al.*, "GraphMat: High performance graph analytics made productive," in *Proc. Very Large Data Bases Conf. Endow.*, 2015, pp. 1214–1225.

[32] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed graph-parallel computation on natural graphs," in *Proc. Presented Part 10th USENIX Symp. Oper. Syst. Des. Implementation*, 2012, pp. 17–30.

[33] S. Han, L. Zou, and J. X. Yu, "Speeding up set intersections in graph algorithms using SIMD instructions," in *Proc. Int. Conf. Manage. Data*, 2018, pp. 1587–1602. [Online]. Available: https://doi.org/10.1145/3183713.3196924

[34] N. Ao *et al.*, "Efficient parallel lists intersection and index compression algorithms using graphics processing units," *Proc. Very Large Data Bases Conf. Endow.*, vol. 4, no. 8, pp. 470–481, 2011.

[35] C. H. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga, "Arabesque: A system for distributed graph mining," in *Proc. 25th Symp. Oper. Syst. Princ.*, 2015, pp. 425–440.

[36] K. Wang, Z. Zuo, J. Thorpe, T. Q. Nguyen, and G. H. Xu, "RStream: Marrying relational algebra with streaming for efficient graph mining on a single machine," in *Proc. 13th USENIX Symp. Oper. Syst. Des. Implementation*, 2018, pp. 763–782.

[37] J. Shun and K. Tangwongsan, "Multicore triangle computations without tuning," in *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 149–160.

[38] X. Zhu, W. Han, and W. Chen, "GridGraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning," in *Proc. USENIX Annu. Tech. Conf.*, 2015, pp. 375–386.

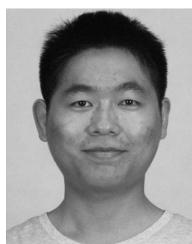
[39] X. Hu, Y. Tao, and C.-W. Chung, "Massive graph triangulation," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 325–336.

Authorized licensed use limited to: Charles University. Downloaded on June 16,2022 at 08:37:13 UTC from IEEE Xplore. Restrictions apply.

[40] J. Kim, W.-S. Han, S. Lee, K. Park, and H. Yu, "OPT: A new framework for overlapped and parallel triangulation in large-scale graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 637–648.
[41] A. Kyrola, G. Blelloch, and C. Guestrin, "GraphChi: Large-scale graph computation on just a PC," in *Proc. 10th USENIX Sym. Oper. Syst. Design Implementation*, 2012, pp. 31–46.
[42] Y. Cui, D. Xiao, and D. Loguinov, "On efficient external-memory triangle listing," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 8, pp. 1555–1568, Aug. 2018.
[43] A. P. Iyer, Z. Liu, X. Jin, S. Venkataraman, V. Braverman, and I. Stoica, "ASAP: Fast, approximate graph pattern mining at scale," in *Proc. 13th USENIX Symp. Oper. Syst. Des. Implementation*, 2018, pp. 745–761.
[44] P. Wang, P. Jia, Y. Qi, Y. Sun, J. Tao, and X. Guan, "REPT: A streaming algorithm of approximating global and local triangle counts in parallel," in *Proc. IEEE 35th Int. Conf. Data Eng.*, 2019, pp. 758–769.
[45] X. Gou and L. Zou, "Sliding window-based approximate triangle counting over streaming graphs with duplicate edges," in *Proc. Int. Conf. Manage. Data*, 2021, pp. 645–657. [Online]. Available: https://doi.org/10.1145/3448016.3452800
[46] J. Wang, Y. Wang, W. Jiang, Y. Li, and K.-L. Tan, *Efficient Sampling Algorithms for Approximate Temporal Motif Counting*. New York, NY, USA: Assoc. Comput. Machinery, 2020, pp. 1505–1514. [Online]. Available: https://doi.org/10.1145/3340531.3411862
[47] P. Wang, X. Wang, J. Tao, P. Zhang, and X. Guan, "Continuously distinct sampling over centralized and distributed high speed data streams," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 2, pp. 300–314, Feb. 2019.
[48] M. Bisson and M. Fatica, "High performance exact triangle counting on GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3501–3510, Dec. 2017.
[49] Y. Hu, H. Liu, and H. H. Huang, "High-performance triangle counting on GPUs," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, 2018, pp. 1–5.
[50] M. Bisson and M. Fatica, "Update on static graph challenge on GPU," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, 2018, pp. 1–8.
[51] Y. Hu, H. Liu, and H. H. Huang, "TriCore: Parallel triangle counting on GPUs," in *Proc. IEEE SC18: Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2018, pp. 171–182.
[52] S. Pandey *et al.*, "Trust: Triangle counting reloaded on GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2646–2660, Nov. 2021.
[53] S. Pandey, X. S. Li, A. Buluc, J. Xu, and H. Liu, "H-INDEX: Hash-indexing for parallel triangle counting on GPUs," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, 2019, pp. 1–7.
[54] S. Diab, M. G. Olabi, and I. El Hajj , "KTRUSSEXPLORER: Exploring the design space of K-truss decomposition optimizations on GPUs," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, 2020, pp. 1–8.
[55] Y. Hu, P. Kumar, G. Swope, and H. H. Huang, "TriX: Triangle counting at extreme scale," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, 2017, pp. 1–7.
[56] M. P. Blanco, S. McMillan, and T. M. Low, "Towards an objective metric for the performance of exact triangle count," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, 2020, pp. 1–7.
[57] J. Zhang, Y. Lu, D. G. Spampinato, and F. Franchetti, "FESIA: A fast and SIMD-efficient set intersection approach on modern CPUs," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2020, pp. 1465–1476.
[58] M. Ashfaq, R. Huang, and M. Omari, "Fscs-SIMD: An efficient implementation of fixed-size-candidate-set adaptive random testing using SIMD instructions," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng.*, 2020, pp. 277–288.
[59] H. Kim *et al.*, "DUALSIM: Parallel subgraph enumeration in a massive graph on a single machine," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 1231–1245.
[60] L. Hu, L. Zou, and Y. Liu, "Accelerating triangle counting on GPU," in *Proc. Int. Conf. Manage. Data*, 2021, pp. 736–748. [Online]. Available: https://doi.org/10.1145/3448016.3452815
[61] C. Gui, L. Zheng, P. Yao, X. Liao, and H. Jin, "Fast triangle counting on GPU," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, 2019, pp. 1–7.
[62] H. Inoue, M. Ohara, and K. Taura, "Faster set intersection with SIMD instructions by reducing branch mispredictions," *Proc. Very Large Data Bases Conf. Endow.*, vol. 8, no. 3, pp. 293–304, Nov. 2014. [Online]. Available: https://doi.org/10.14778/2735508.2735518
[63] W. Zheng, Y. Yang, and C. Piao, "Accelerating set intersections over graphs by reducing-merging," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, 2021, pp. 2349–2359. [Online]. Available: https://doi.org/10.1145/3447548.3467219
[64] Y. Che, Z. Lai, S. Sun, Q. Luo, and Y. Wang, "Accelerating all-edge common neighbor counting on three processors," in *Proc. 48th Int. Conf. Parallel Process.*, 2019. [Online]. Available: https://doi.org/10.1145/3337821.3337917
[65] R. Zheng and S. Pai, "Efficient execution of graph algorithms on CPU with SIMD extensions," in *Proc. IEEE/ACM Int. Symp. Code Gener. Optim.*, 2021, pp. 262–276.
[66] J. Wu *et al.*, "Effective exploitation of SIMD resources in cross-ISA virtualization," in *Proc. 17th ACM SIGPLAN/SIGOPS Int. Conf. Virt. Execution Environ.*, 2021, pp. 84–97. [Online]. Available: https://doi.org/10.1145/3453933.3454016
[67] J. Li, B. Ji, Y. Yang, P. Wei, and J. Wu, "Parallel optimization of the crystal-KMC on tianhe-2," *Tsinghua Sci. Technol.*, vol. 26, no. 3, pp. 309–321, 2021.
[68] H. Bian, J. Huang, L. Liu, D. Huang, and X. Wang, "ALBUS: A method for efficiently processing spmv using SIMD and load balancing," *Future Gener. Comput. Syst.*, vol. 116, pp. 371–392, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X2033020X
[69] J. Zhong and B. He, "Medusa: Simplified graph processing on GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1543–1552, Jun. 2013.
[70] F. Khorasani, K. Vora, R. Gupta, and L. N. Bhuyan, "CuSha: Vertex-centric graph processing on GPUs," in *Proc. 23rd Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2014, pp. 239–252.
[71] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," in *Algorithm Engineering*, Berlin, Germany: Springer, 2016, pp. 117–158.
[72] X. Zhu, W. Chen, W. Zheng, and X. Ma, "Gemini: A computation-centric distributed graph processing system," in *Proc. 12th USENIX Symp. Oper. Syst. Des. Implementation.*, 2016, pp. 301–316.
[73] I. Giechaskiel, G. Panagopoulos, and E. Yoneki, "PDTL: Parallel and distributed triangle listing for massive graphs," in *Proc. IEEE 44th Int. Conf. Parallel Process.*, 2015, pp. 370–379.

Jianqiang Huang (Member, IEEE) received the bachelor's degree from Xi'an Military Academy, Xi'an, China, in 2009, and currently working toward the PhD degree at the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He is currently an associate professor with the Department of Computer Technology and Applications, Qinghai University of China. His major research interests include Graph computing, heterogeneous computing (CPUs/GPUs), and parallel and distributed systems.

Haojie Wang (Member, IEEE) received the BS and PhD degrees from Tsinghua University, Beijing, China, in 2015 and 2021, respectively, and currently a postdoctoral with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests include high-performance computing, Performance analysis, and parallel and distributed systems. His research received Best Student Paper Award at ICS'21.



Xiang Fei (Member, IEEE) received the bachelor's degree from Zhejiang University, Hangzhou, China, in 2016, and currently working toward the PhD degree at the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His major research interests include GPU accelerated high-performance computing, graph analytics, and parallel and distributed systems.

Authorized licensed use limited to: Charles University. Downloaded on June 16,2022 at 08:37:13 UTC from IEEE Xplore. Restrictions apply.

Xiaoying Wang (Member, IEEE) received the BS and PhD degrees in computer science from Tsinghua University, Beijing, China, in 2003 and 2008, respectively. Her research interests include high-performance computing, green computing, and parallel and distributed systems.

Wenguang Chen received the BS and PhD degrees in computer science from Tsinghua University, Beijing, China, in 1995 and 2000, respectively. His research focuses on performance and modeling of extreme-scale systems, graph computing, and parallel and distributed systems. Among other awards for research and teaching excellence, he received the ACM Gordon Bell Award Finalists, in 2018 for his leading-edge work in parallel computing. He is chair of ACM China Council, chief editor of ACM Communications Chinese Edition. He has been a member of the program committee of many important academic conferences, including OSDI, SOSP, PLDI, PPoPP, SC, ASPLOS, CGO, IPDPS, CCGrid, ICPP, etc.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: Charles University. Downloaded on June 16,2022 at 08:37:13 UTC from IEEE Xplore. Restrictions apply.