Accelerating Core Decomposition in Large Temporal Networks Using GPUs

Heng Zhang[1,2], Haibo Hou[3], Libo Zhang[1(✉)], Hongjun Zhang[1], and Yanjun Wu[1]

[1] Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
zhangheng@nfs.iscas.ac.cn, zsmj@hotmail.com
[2] University of Chinese Academy of Sciences, Beijing 100040, China
[3] China Academy of Information and Communications Technology, Beijing 100191, China

Abstract. In recent times, many real-world networks are naturally modeled as temporal networks, such as neural connection in biological networks over time, the interaction between friends at different time in social networks, etc. To visualize and analysis these temporal networks, core decomposition is an efficient strategy to distinguish the relative "importance" of nodes. Existing works mostly focus on core decomposition in non-temporal networks and pursue efficient CPU-based approaches. However, applying these works in temporal networks makes core decomposition an already computationally expensive task. In this paper, we propose two novel acceleration methods of core decomposition in the large temporal networks using *the high parallelism of GPU*. From the evaluation results, the proposed acceleration methods achieve maximum 4.1 billions TEPS (traversed edges per second), which corresponds to up to 26.6× speedup compared to a single threaded CPU execution.

Keywords: Temporal network · Core decomposition · GPU

1 Introduction

Recently, complex networks are widely used to model relationships in many fields, including protein networks in bio-informatics, Internet connection networks and real-world social friendship networks. Among these network structures, many real world networks are actually temporal networks, in which nodes communicate with others at specific time instances [5,8]. For example, in biological networks, the neural connections can be modeled as temporal networks. In Fig. 1(a) and (b), node N_1 and N_2 represent two individual sensors for Electroencephalography (EEG) respectively, and temporal links between N_1 and N_2 represent the time dynamics of simultaneous brain area activations. When the signal of extracranial magnetic fields is correlated at time points of 4, 12 and 16 (e.g., Hour 4, Hour 12, Hour 16), the three edges between N_1 and N_2 are assigned at the time points of 4, 12 and 16. Moreover, N_1 follows N_2 at a point in time in social networks, N_1 spread informations to N_2 at different times in

D. Liu et al. (Eds.): ICONIP 2017, Part I, LNCS 10634, pp. 893–903, 2017.
https://doi.org/10.1007/978-3-319-70087-8_91

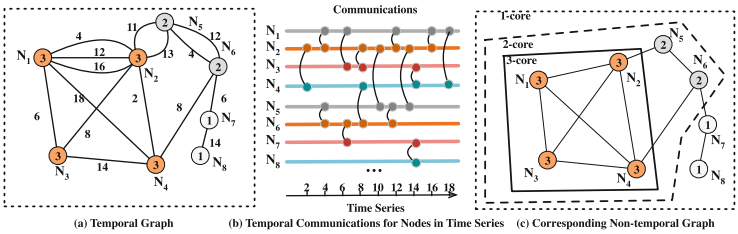


Fig. 1. A sample temporal network $\widehat{G}$ and its non-temporal network G.

information dissemination networks, to name but a few. In general, these functional connections between two nodes in temporal networks can be abstracted as the temporal connectivity with time series association. Also, after discarding the time information, the temporal networks can be transformed into non-temporal networks (or static networks) by condensing the multiple edges between two nodes into a single edge. We illustrate an example for the representation of a sample temporal network and its non-temporal network in Fig. 1.

Meanwhile, there has been a proliferation of metrics and strategies to distinguish the relative "importance" of nodes within large network structures, such as eigenvector [3], betweenness [1] and centrality indexes [7], etc. Among these metrics, *core decomposition* has been studied as a well-established method to identify a special group of cohesive subgraphs of a network, namely *k-cores*, or *k-shells* [2,10,11]. For non-temporal network, the k-core for all possible integer k values is obtained by a maximal induced subgraph such that all nodes in k-core have a degree of at least k [2]. Most strategies [2,4,9] are designed for non-temporal network core composition. Batagelj and Zaversnik [2] first propose a linear time algorithm (namely BZ algorithm), which recursively delete nodes of degree less than k in the obtained subgraph. Dasari et al. [4] design *ParK* to scale sequential BZ algorithm to multi-core machine. With respect to temporal network, Wu et al. [13] first define *k-core* of temporal networks as formulation ((k,h)-core), where k controls the connectivity of nodes and h controls the intensity of temporal activity between two nodes. Here, the (k,h)-core) of a temporal network is the largest subgraph such that every node in this network has at least k neighbors, where each neighbor must be connected with at least h temporal edges. This means there are at least h signals communicated with each other for nodes in (k,h)-core).

The quality of core decomposition in large temporal networks depends on many factors such as the amount of input graph data, the time complexity of algorithm and parallel scalability. As the size and complexity of networks increase, faster processing of core decomposition implies larger amount of graph data in a given runtime. With the popularity of SIMD (Single Instruction Multiple Data) architecture, Graphics Processing Unit (GPU) provides the processing of large network structure not only massive parallelism (approximately 10 Ks threads) but also efficient memory I/O (up to 100 GB/s memory bandwidth), which makes it an excellent hardware platform for large network structure analytic [12].

In this paper, we propose *two novel acceleration algorithms* of *core decomposition* in *the large temporal networks* using *the high parallelism of GPU*. While GPU offers massive parallelism, achieving high-performance core decomposition in large temporal networks on GPU entails efficient scheduling of massive GPU threads and effective utilization of GPU memory hierarchy. We first present an algorithm following the definition of k-core in temporal networks. This algorithm, namely *TRCore*, is designed to transform temporal network into non-temporal and then traverse network using GPU-based bottom-up approach with recursively distinguishing nodes by their degree and temporal edges. Since GPU-based graph traversal phase would lead to amount of contention overhead from concurrent threads, the second algorithm, namely *ESCore*, is proposed as a non-trivial algorithm based on the *locality property* [6] of k-core structure in temporal networks. In *ESCore*, each node estimates and updates its core number based on their neighbors' core values until convergence. By introducing the SIMD (Single Instruction Multiple Data) architectural GPU accelerator, we implement these two algorithms using CUDA C/C++ and achieve optimal performance speedup. The results represent the two algorithms achieve 1.1–4.1 billions TEPS (traversed edges per second) and a maximum speedup of 26.7× on the real-world temporal networks compared to a single thread CPU execution.

The rest of the paper is organized as follows. In Sect. 2 we describe the formula of core decomposition in large temporal networks. In Sect. 3 we present our novel GPU-based parallel methodologies for GPU-based core decomposition and the implementation details of our algorithms. Then, in Sect. 4 we evaluate and analyze our methods in various real-world network datasets. And we conclude our work and mention some future work in Sect. 5.

2 Core Decomposition

2.1 Notations

Let $\widehat{G} = (\widehat{V}, \widehat{E})$ be an undirected temporal network, where $\widehat{V}$ and $\widehat{E}$ is the set of nodes and edges in $\widehat{G}$. Each edge $\hat{e} \in \widehat{E}$ is expressed as a triple (n_s, n_e, t), where $n_s, n_e \in \widehat{V}$ and t is the time point that $\hat{e}$ is active (e.g., a signal is communicated from n_s to n_e during t). In temporal network, one node can communicate with others at multiple times. The multiple temporal edges between two nodes n_s, n_e are denoted by $\Pi(n_s, n_e)$, and $\Pi(n_s, n_e) = \{(n_s, n_e, t) | n_s \in \widehat{V}, n_e \in \widehat{V}, (n_s, n_e, t) \in \widehat{E}\}$. The number of temporal edges between two nodes is denoted as $\pi(n_s, n_e) = |\Pi(n_s, n_e)|$.

After we remove the temporal information and condense the edges in each Π, we obtain a *non-temporal* network of $\widehat{G}$, denoted by G, and $G = (V, E)$ where $V = \widehat{V}$ and $E = \{(n_s, n_e) | (n_s, n_e, t) \in \widehat{E}\}$. We define the number of vertices in $\widehat{G}$ and G as $n = |V| = |\widehat{V}|$, the number of edges in $\widehat{G}$ as $\hat{m} = |\widehat{E}|$ and in G as $m = |E|$. Furthermore, we define the neighbor set of node n_s as $\Gamma(u, G) = \Gamma(u, \widehat{G}) = \{n_e | (n_s, n_e, t) \in \widehat{E}\} = \{n_e | (n_s, n_e) \in E\}$. The degree of one node n_s is defined as $deg(u_s, \widehat{G}) = \Sigma_{u_e \in \Gamma(u_s, G)} \pi(u_s, u_e)$ in *temporal network* $\widehat{G}$ and $deg(u_s, G) = |\Gamma(u_s, G)|$ in *non-temporal* G.

We note a definition to the non-temporal subgraph at h time point for $\widehat{G}$. Specifically, given a specific time i, $G_i = (V_i, E_i)$ is a subgraph of the non-temporal graph G, where exists active edges after h time point, i.e. $V_i = V$ and $E_i = \{(n_s, n_e)|(n_s, n_e, t) \in \widehat{E}, \pi(u_s, u_e) \geq h\}$.

2.2 Definition of Core Decomposition in Networks

The core decomposition of non-temporal network is to obtain every non-empty k-core of the non-temporal G for $k \geq 1$. Though maintaining the degree values of nodes in G, k-core decomposition for *non-temporal* network is defined as follows.

Definition 1 (Non-Temporal k-Core). Given a *non-temporal* network G and integer k, the k-core of G, denoted as G_k, is a maximal induced subgraph such that all nodes in G_k have a degree of at least k, i.e., $\forall n \in V, deg(n, G_k) \geq k$. Meanwhile, the largest value of k for node n is denoted core number, i.e., $core(n, G) = max\{k|n \in V_k\}$.

After adding the temporal information in *temporal networks* $\widehat{G}$, the k-core decomposition needs to consider the constrains of temporal edges. In this paper, we follow the (k, h)-core definition in [13], and clearly define the core decomposition for temporal networks on the top of the number of neighbors. In each (k, h)-core $\widehat{G}_{(k,h)}$ in $\widehat{G}$, we enforce two limiting conditions, the k controls the connectivity of nodes and h controls the intensity of temporal activity between two nodes.

Definition 2 (Temporal (k, h)-Core). Given a *temporal* network $\widehat{G}$ and two integers k, h, the (k, h)-core of $\widehat{G}$, denoted as $\widehat{G}_{(k,h)}$, is a maximal induced subgraph such that all nodes in $\widehat{G}_{(k,h)}$ have at least k neighbor nodes, and the two nodes need to be connected with at least h temporal edges, i.e., $\forall n_s \in V, |\{n_s|n_s \in \Gamma(n_e, \widehat{G}_{(k,h)}), \pi(n_s, n_e)\}| \geq k$.

Figure 1(a) illustrates an example based on the above (k, h)-core definition. Based on the above definition, N_7, N_8 are in the $(1, 1)$-core; N_5, N_6 are in the $(2, 2)$-core; N_3 and N_4 are in the $(3, 1)$-core; and N_1 is in the $(3, 1)$-core and $(3, 3)$-core; and N_2 is in $(2, 2)$-core, $(3, 1)$-core, as well as in the $(3, 3)$-core. Thus, the $(1, 1)$-core, $(3, 1)$-core and $(3, 3)$-core in $\widehat{G}$ clearly distinguish the central temporal relationship between N_1 and N_2, and central connected community which consists of N_1, N_2, N_3 and N_4.

Since the neighbors of node u_s are the same vertex set in G and $\widehat{G}$, we can also obtain the n_e by traversing the *non-temporal* G to simplify the process. Further, we define the *temporal core number* for nodes in $\widehat{G}$.

Definition 3 (Temporal Core Number). Given a *temporal* network $\widehat{G}$ and a node $n \in \widehat{V}$, the temporal core number of $\widehat{G}$, denoted as $core(n, \widehat{G})$, is the maximal values (k_{max}, h_{max}) of k and h such that n is in the $\widehat{G}_{(k_{max}, h_{max})}$, i.e., $core(n, \widehat{G}) = max\{(k, h)|n \in \widehat{V}_{(k,h)}\}$ and $\forall k' \geq k, \forall h' \geq h, n \notin \widehat{V}_{(k',h')}$.

In this paper, we focus on the computation $core(n, \widehat{G})$ for each node n in $\widehat{G}$.

3 Proposed GPU-Based Parallel Methods for Temporal Core Decomposition

The methodologies we propose here to accelerate core decomposition are based on GPU. While GPU offers high parallelism, the irregular network structural data would lead to high contention overhead when multiple threads update one node concurrently. Thus achieving high-performance core composition on GPUs entails efficient scheduling of massive GPU threads and effective utilization of memory hierarchy. Here we present two efficient parallel algorithms for core decomposition using a GPU. The first method *TRCore* is designed as a traverse algorithm along the definition of temporal (k, h)-core, shown in Fig. 2(a). The second method *ESCore* is proposed as an estimate algorithm based on the locality property of temporal (k, h)-core in $\widehat{G}$, shown in Fig. 2(d). Compared to *TRCore*, *ESCore* method can benefit from the much less contention overhead (multiple threads simultaneously write a same memory address, see Fig. 2(b) and (c)) and represent a higher performance. We illustrate the pseudo code of the main phase in Algorithm 1. The procedure first iteratively construct the time point subgraph in each time point i in $[1, \pi_{max}]$ and execute the above two method to get the connectivity k for each node, and then merge the core number value and time point pairs (i.e., $<k, i>$) of nodes.

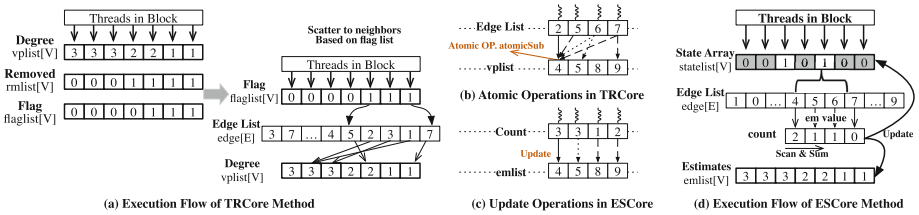


Fig. 2. Execution flow for *TRCore* and *ESCore* methods.

3.1 Method 1: Traverse Method Based on Neighborhood

Based on the Definitions 2 and 3 for (k, h)-core, we obtain the number constrain condition of neighbors for each node in the following theorem.

Theorem 1. Given temporal network $\widehat{G}$, its non-temporal network G. In G, a node n is in i-core of G if and only if n has at least i neighbors in i-core. Meanwhile, in temporal network $\widehat{G}$, let $(k, h) \in core(n, \widehat{G})$, then $core(n, G) = k$ for n in G_h at h time point.

Based on the Theorem 1, the core number (k) of the node v in non-temporal network can be easily obtained by starting from 1-core, recursively remove all nodes with degree less than or equal to the current core number, along with their

Algorithm 1. Main Phase

Input: An input undirected temporal graph $\widehat{G} = (\widehat{V}, \widehat{E})$
Output: The core number pair (k, h) of each node $n \in \widehat{V}$

Allocate vertex property and state buffers in host memory and device memory;
Initialize (k, h) pair value set for $|V|$ nodes $\leftarrow \varnothing$;
foreach $i \in [1, \pi_{max}]$ **do**
  if *i = 1* **then**
    Get non-temporal G_i: Obtain edge set E_i via condensing edges in $\widehat{E}$;
  else
    Get non-temporal G_i: Filter edge set E_i from G_{i-1};
  end
  Call K_{TR} to set core numbers of nodes in G_i **for** $k \in [1, k_{max}]$;
  or Call K_{ES} to update core number of nodes **until** *statelist* all inactive;
  foreach *node* $v \in G_i$ **do**
    Let ϕ =core(v, G_i) be the core number value of v in non-temporal G_i and add (ϕ, i) to core number pair value set of v;
  end
end

Algorithm 2. Method 1: TRCore Kernel Function

Input: An input undirected Graph G_i, vertices' degree *vplist*, remove state *rmlist*, flag *flaglist*, core level integer *kcore*
Output: The core number pair (k, h) of each node $n \in \widehat{V}$

Function K_{TR}*(*G_i*, kcore, vplist[], rmlist[], flaglist[])* **begin**
  vid ← blockIdx.x * blockDim.x + threadIdx.x;
  if *flaglist[vid] = true* **then**
    foreach $u_i \in \Gamma(vid, G_i)$ *of node vid* **do**
      if *rmlist*$[u_i]$ *= false* **then**
        atomicSub(&(vplist[u_i]), 1);
      end
    end
    flaglist[vid] ← true;
  end
  if *rmlist[vid] = false* **then**
    if *vplist[vid] < kcore* **then**
      rmlsit[vid] ← true;
      flaglist[vid] ← true;
      atomicAdd(rmcnt, 1);
    end
  end
end

edges, from the network. Then, with the varying time point h in $[1, \pi_{max}]$, core number (k, h) of each node in temporal network can be iteratively merged after processing non-temporal generated subgraphs in each time point $1 \leq h \leq \pi_{max}$.

The implementation of Method 1 is explained in Algorithm 2. The graph data for temporal network is stored in global memory in GPU while the flag and removed state data are stored in shared memory. We allocate three size n of arrays ($vplist$, $rmlist$ and $flaglist$). The $vplist[v]$ records the current degree of v, the $rmlist[v]$ indicates whether v has been fall into other core and $flaglist[v]$ labels the processed flag of v. We launch the K_{TR} GPU kernel function to do vertex-centric traverse based core decomposition, where each GPU thread processes one node in one time. Given a vertex v and a specific $kcore$ value, when $flaglist$ of v is set to true, the v incident edges are traversed to be deleted and the degrees of neighbors also decrements by one (Line 3–10). Thus when the $rmlist$ of v is false and $vplist[v] < kcore$, we ensure v belongs to this k-core and mark $flaglist[v]$ to $true$. The procedure for each G_i require k_{max} iterations.

3.2 Method 2: Estimate Method Based on Locality

We introduce a *locality property* of non-temporal network in Method 2 to enhance the performance [6]. In non-temporal network, the core number of an arbitrary node n would be $[0, deg(n)]$. And if the node v is in the k-core subgraph, it has at least k neighbors in this subgraph, where the degrees of these neighbors are in $[0, deg(n)]$. Moreover, after we sort this neighbor nodes $u_i (1 \leq i \leq n \leq deg(v))$ by degree, i.e., $core(u_{i+1}) \geq core(u_i)$, the node v has at least $deg(v) - (i-1)$ neighbors whose core number is larger or equal than $core(u_i)$. Thus, we obtain the *locality property* of *temporal network* as following.

Theorem 2. Given a temporal network $\widehat{G} = \{\widehat{V}, \widehat{E}\}$ and non-temporal $G = \{V, E\} (V = \widehat{V})$, $core(n, \widehat{G}) = (k, h)$ values $\forall n \in V, \pi(n, \Gamma(n, \widehat{G})) \geq h$ if and only if

- there exists a subset $V_k \subseteq \Gamma(n, G)$ such that $|V_k| = k$ and $\forall n \in V_k$: $core(n, G) \geq k$.
- there not exists a subset $V_{k+1} \subseteq \Gamma(n, G)$ such that $|V_{k+1}| = k + 1$ and $\forall n \in V_{k+1} : core(n, G) \geq k + 1$.

Moreover, let $u_1, u_2, ..., u_n$ be the neighbors of node $v \in V_k$ sorted by core number and G_τ be a generated non-temporal subgraph at τ time point ($\tau \in [1, \pi_{max}]$), the core number of v can be calculated using the following equation:

$$core(v, G_\tau) = max\ k\ s.t.\ |\{\forall u_i \in nbr(v) | core(u_i, G_\tau) \geq k\}| \geq k \tag{1}$$

Though merging the $<k, \tau>$ pair values of v in each G_τ and removing duplicate values, the core number values of v in temporal network $\widehat{G}$ are obtained.

The implementation of Method 2 is explained in Algorithm 3. Given a vertex v and its current core number $core_{cur}$, the $eslit$ record the current core number of all nodes in global memory. We use the $count[1 : core_{max}]$ array to denote the number of neighbors of v with their core number equals the index of $count$ (Line 5–8). Then we calculate sum, the number of neighbors of v with their core

Algorithm 3. Method 2: ESCore Method

Input: An input indirected graph G_i, vertices' current core number *eslist*, vertex state *statelist*
Output: The core number k of each node $n \in \widehat{V}$ *eslist*

```
1  Function K_ES(G_i, eslist[], statelist[]) begin
2      vid ← blockIdx.x * blockDim.x + threadIdx.x;
3      core_cur ← eslist[vid];
4      count[1: core_cur] ← 0;
5      foreach u_i ∈ Γ(vid, G_i) of node vid do
6          j ← min(core_cur, eslist[vid]);
7          count[j] ← count[j] + 1;
8      end
9      sum ← 0;
10     for k=core_cur; k>2; k-- do
11         sum ← sum + count[k];
12         if sum ≥ k then
13             break;
14         end
15     end
16     if k < core_cur then
17         eslist[vid] = k;
18         statelist[vid] = true;
19     end
20 end
```

number is larger than k (Line 9–15), i.e., $sum = |\{\forall u_i \in nbr(v)|core(u_i, G_\tau) \geq k\}|$. Once $sum \geq k$, we obtain the maximum k for conclusive updated new core number of v (Line 16–19). The procedure is convergent after state of nodes are all inactive.

4 Experimental Result

Table 1 shows the real-world temporal graphs with a broad range of sized and features from different origins, which are from Stanford Large Network Dataset Collection (http://snap.stanford.edu/data/). *Mathoverflow*, *Superuser* and *Stackoverflow* are the answer and comment graphs, where user u answered or commented user v's question at time t. *CollegeMsg* is comprised of private messages sent on an online social network. *Wiki-talk* represents Wikipedia users editing each other's Talk page. We perform the experiments on a server with NVIDIA GeForce GTX980 each having 16 Maxwell Streaming Multiprocessors (128 Cores/MP) and 4 GB GDDR5 RAM. The host side of the node is consist of two 10-core Intel Xeon E5-2650 v3, and 64 GB DDR4 main memory, running with Ubuntu 16.04 (kernel v4.4.0-38) with CUDA 7.5.

We first show the effects of our two core decomposing methods on GPU-based platform. The two comparison of core decomposition algorithms are

Table 1. Real-world and synthetic graph datasets used in this paper. 'Temp.' represents edges in temporal graph, 'NonT.' represents non-temporal graph. And the preprocessing time consist of edge list load, transformation and indexes building time.

Dataset	Nodes $\lvert\widehat{V}\rvert$	Temp. edges $\lvert\widehat{E}\rvert$	NonT. edges $\lvert E\rvert$	Avg. deg deg_{avg}	MAX π π_{max}	Preprocess $Time_{pre}$
CollegeMsg	1,899	59,835	20,296	31.5	98	0.8 s
Mathoverflow	24,818	506,550	239,978	20.4	1944	1.2 s
Superuser	194,085	1,443,339	924,886	7.4	3626	6.8 s
Wiki-talk	1,140,149	7,833,140	3,309,592	6.9	31,450	15.9 s
Stackoverflow	2,601,977	63,497,050	36,233,450	24.4	29,919	109.2 s

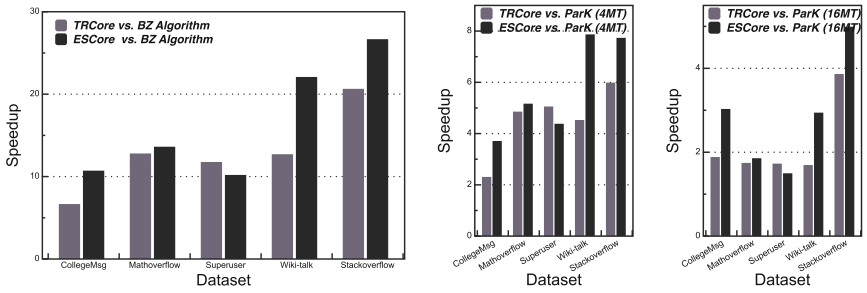
Table 2. Elapsed time (in seconds) comparison between BZ algorithm, ParK algorithm and TRCore, ESCore method. The 'MT' denote to the number of multiple threads.

Dataset	BZ	ParK (2MT)	ParK (4MT)	ParK (16MT)	TRCore	ESCore
CollegeMsg	185.43	61.85	64.20	52.44	27.81	**17.30**
Mathoverflow	321.75	194.62	122.05	43.86	25.13	**23.62**
Superuser	1352.20	956.19	581.00	198.56	**114.89**	132.57
Wiki-talk	4101.99	2626.18	1461.70	546.74	322.68	**185.73**
Stackoverflow	12970.50	7299.54	3762.14	2428.33	628.27	**486.36**

coming from the state-of-art, namely single-thread BZ algorithm [2] and multi-core CPU based ParK algorithm [4]. These two non-temporal core decomposition algorithms are integrated into detecting core number of non-temporal network given a specific time point. BZ algorithm is configured to one single thread and ParK algorithm to 2, 4, 16 threads in evaluation. Table 2 shows us the elapsed time for our two methods (TRCore, ESCore) with GPU can achieve an optimal performance compared to other execution (with overstriking marks).

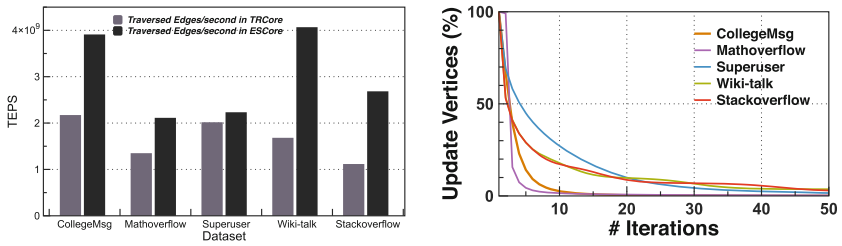
Moreover, we illustrate the speedup ratio for TRCore and ESCore. Compared to BZ algorithm, TRCore achieves 6.7–20.6 time of speedup and ESCore achieves 10.7–26.6 time of speedup. With the size of temporal graph data increasing, TRCore and ESCore with GPU represent better performance enhancement. Meanwhile, with respect to ParK algorithm, TRCore and ESCore also achieve 3–7.8 time of speedup over 4-thread ParK and 1.8–5.0 time of speedup over 16-thread ParK. The reason that GPU-based TRCore and ESCore represent excellent performance accelerating was that the optimized scalable algorithms benefit from massive parallelism and high memory bandwidth. From the result, we also conclude that ESCore achieve a better performance than TRCore, and ESCore benefits from an efficient scheduling mechanism of massive GPU threads (Fig. 3).

Figure 4(a) illustrates TEPS (traversed edges per second) metric comparison for TRCore and ESCore, and shows the ESCore achieves maximum 4.1 billion TEPS in wiki-talk graph and almost enhances 2.4× TEPS than TRCore.



(a) Speedup for BZ (b) ParK (4MT) (c) ParK (16MT)

Fig. 3. Speedup for TRCore and ESCore comparison with BZ, ParK (4 Threads) and ParK (16 Threads).



(a) TEPS in TRCore and ESCore (b) Vertex Update Ratio in ESCore

Fig. 4. Traversed edges per second in TRCore and ESCore, and vertex update ratio in ESCore for $\widehat{G}$ at $i = 1$ time point.

Figure 4(b) summaries the updated vertex ratio in ESCore method for $\widehat{G}$ core decomposition in $i = 1$ time point. From the results, the ratio of updated vertices decreases sharply in the first 5 iterations, and then flattens, represents less than 5% after 50 iterations.

5 Conclusion and Future Work

In this paper, we proposed an efficient parallelization of core decomposition in large temporal network using GPUs. By carefully considering the massive parallelism provided by GPU, two method are designed based on two core decomposition theorems. The first *TRCore* method is designed to traverse network using GPU-based bottom-up approach with recursively distinguishing nodes. In the second *ESCore* method, each node estimates and updates its core number according to current core values of their neighbors until convergence. By introducing GPU accelerator, two algorithms achieve a maximum speedup of 26.7× on the real-world temporal networks. We work now on a method to maintain the core decomposition extending temporal networks to dynamic networks.

References

1. Attal, J.-P., Malek, M., Zolghadri, M.: Overlapping community detection using core label propagation and belonging function. In: Hirose, A., Ozawa, S., Doya, K., Ikeda, K., Lee, M., Liu, D. (eds.) ICONIP 2016. LNCS, vol. 9949, pp. 165–174. Springer, Cham (2016). doi:10.1007/978-3-319-46675-0_19
2. Batagelj, V., Zaveršnik, M.: Fast algorithms for determining (generalized) core groups in social networks. Adv. Data Anal. Classif. **5**(2), 129–145 (2011)
3. Bonacich, P.: Some unique properties of eigenvector centrality. Soc. Netw. **29**(4), 555–564 (2007)
4. Dasari, N.S., Desh, R., Zubair, M.: Park: an efficient algorithm for k-core decomposition on multicore processors. In: 2014 IEEE International Conference on Big Data (Big Data), pp. 9–16. IEEE (2014)
5. Holme, P., Saramäki, J.: Temporal networks. Phys. Rep. **519**(3), 97–125 (2012)
6. Montresor, A., De Pellegrini, F., Miorandi, D.: Distributed k-core decomposition. IEEE Trans. Parallel Distrib. Syst. **24**(2), 288–300 (2013)
7. Newman, M.E.: The structure and function of complex networks. SIAM Rev. **45**(2), 167–256 (2003)
8. Nicosia, V., Tang, J., Mascolo, C., Musolesi, M., Russo, G., Latora, V.: Graph metrics for temporal networks. In: Holme, P., Saramäki, J. (eds.) Temporal networks. Understanding Complex Systems, pp. 15–40. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36461-7_2
9. OBrien, M.P., Sullivan, B.D.: Locally estimating core numbers. In: 2014 IEEE International Conference on Data Mining (ICDM), pp. 460–469. IEEE (2014)
10. Seidman, S.B.: Network structure and minimum degree. Soc. Netw. **5**(3), 269–287 (1983)
11. Shin, K., Eliassi-Rad, T., Faloutsos, C.: CoreScope: graph mining using k-core analysis-patterns, anomalies and algorithms. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 469–478. IEEE (2016)
12. Wang, Y., Davidson, A., Pan, Y., Wu, Y., Riffel, A., Owens, J.D.: Gunrock: A high-performance graph processing library on the GPU. In: Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, p. 11. ACM (2016)
13. Wu, H., Cheng, J., Lu, Y., Ke, Y., Huang, Y., Yan, D., Wu, H.: Core decomposition in large temporal graphs. In: 2015 IEEE International Conference on Big Data (Big Data), pp. 649–658. IEEE (2015)