# Parallel Strongly Connected Components Detection with Multi-partition on GPUs*

Junteng Hou[1,2(✉)], Shupeng Wang[1], Guangjun Wu[1], Ge Fu[3(✉)], Siyu Jia[1(✉)], Yong Wang[1], Binbin Li[1], and Lei Zhang[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{houjunteng,wangshupeng,wuguangjun}@iie.ac.cn
[3] National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing, China
fg@cert.org.cn
{jiasiyu,wangyong,libinbin,zhanglei1}@iie.ac.cn

**Abstract.** The graph computing is often used to analyze complex relationships in the interconnected world, and the strongly connected components (SCC) detection in digraphs is a basic problem in graph computing. As graph size increases, many parallel algorithms based on GPUs have been proposed to detect SCC. The state-of-the-art parallel algorithms of SCC detection can accelerate on various graphs, but there is still space for improvement in: (1) Multiple traversals are time-consuming when processing real-world graphs; (2) Pivot selection is less accurate or time-consuming. We proposed an SCC detection method with multi-partition that optimizes the algorithm process and achieves high performance. Unlike existing parallel algorithms, we select a pivot and traverse it forward, and then select a vice pivot and traverse the pivot and the vice pivot backwards simultaneously. After updating the state of each vertex, we can get multiple partitions to parallelly detect SCC. At different phases of our approach, we use a vertex with the largest degree product or a random vertex as the pivot to balance selection accuracy and efficiency. We also implement weakly connected component (WCC) detection and 2-SCC to optimize our algorithm. And the vertices marked by the WCC partition are selected as the pivot to reduce unnecessary operations. We conducted experiments on the NVIDIA K80 with real-world and synthetic graphs. The results show that the proposed algorithm achieves an average detection acceleration of 8.8 × and 21 × when compared with well-known algorithms, such as Tarjan's algorithm and Barnat's algorithm.

**Keywords:** Strongly connected components detection · GPU · Multi-partition scheme · Real-world graphs.

* This work was supported by the National Natural Science Foundation of China (No. 61601458) and the National Key Research and Development Program of China (2016YFB0801305).

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2

## 1 Introduction

In the interconnected world, many practical applications need to explore large-scale data sets represented by graphs. The strongly connected components (SCC) detection is a fundamental graph computing algorithm that is widely used in many applications, such as network analysis based on web archives, scientific computing [1] and model checking [2].

In a digraph, SCC is the largest subgraph in which any two vertices are mutually reachable. In the early, many efficient algorithms on SCC detection have been proposed including Tarjan's [3], Dijkstra's [4] and Kosaraju's [5]. However, these algorithms are based on depth-first search (DFS) traversal that is difficult to accelerate by parallelization [6]. Barnat et al. [7] introduced three distributed algorithms for detecting SCC: forward and backward (FB) algorithm, Coloring algorithm, and forward and backward with OWCTY elimination (OBF) algorithm. Shrinvas et al. [8] analyzed these algorithms by implementing them on GPU, and summarized that the FB algorithm is better than the other two algorithms. Fleischer et al. first proposed FB algorithm [9], which can obtain one SCC and three subgraphs in each traversal, then continue to iteratively detect SCC on each subgraph. McLendon et al. [10] designed the FB-Trim algorithm that improves the efficiency of SCC detection by trimming the 1-SCCs before forward and backward BFS.

Parallel SCC detection on real-world graphs is difficult because the vertices in real-world graphs obey the power-low distribution [13]. Hong et al. [11] proposed a two-phase algorithm to detect SCCs of different sizes, and they also detect weakly connected component (WCC) and the SCC consisting of two vertices (2-SCC) to optimize the algorithm. Li et al. [12] further propose data-level parallel scheme in the phase of large-scale SCC detection and the task-level parallel scheme in the phase of small-scale SCC detection. Shrinivas et al. [13] proposed an algorithm that selects the vertex with the largest product of in-degree and out-degree as the pivot to ensure the selected pivot is on the largest SCC. Li et al. [14] reduce the traversals by dividing the original graph into multiple partitions and parallel detect on them, but it requires more processing of the vertices on the partition boundary. Aldegheri et al. [15] concluded that no algorithm can beat all other algorithms on various types of graphs. They combine multiple algorithms and use the trained model to adjust their order for different graphs.

We propose a scheme that reduces the traversals by selecting vice pivot to increase the partitions generated on each iteration, and we also optimize some other operations. The contributions are as follows:

(1) We add a vice pivot selection between the pivot's forward and backward traversal. Therefore, the backward traversals of the vice pivot and the pivot are performed simultaneously. In parallel algorithms [9–15], detection on each partition generates one SCC and three partitions. Our approach can generate one SCC and five partitions, which can effectively reduce the number of traversals and significantly accelerate the algorithm.

(2) We balance the accuracy and runtime of the pivot selection. In the phase of large SCC detection, we select the vertex with the maximum product of in-

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2

degree and out-degree as the pivot to ensure that the selected pivot is on the largest SCC. In following phase, we use parallel random method to quickly select pivots and vice pivots.

(3) We combine WCC detection and 2-SCC detection with our algorithm to speed up the detection of trivial SCCs. After WCC detection, the vertices marked as WCC labels are directly used as the pivots to reduce a pivot selection without damaging the selection effect.

The rest of this paper is organized as follows: we illustrate the background in Section 2. Our proposed design and its operations on different algorithms are introduced in Section 3. We present the experimental results in Section 4. The conclusion is given in Section 5.

## 2 Background

In this section, we briefly introduce the background of SCC detection on directed graphs.

### 2.1 Synthetic Graphs and Real-world Graphs

The real-world graphs and synthetic graphs are different in structures, which may cause algorithms that perform well on synthetic graphs inappropriate for real-world graphs.

The most outstanding characteristic of real-world graphs is the power law distribution. Most real-world graphs contain a large SCC and lots of small SCCs. The number of these SCCs generally obey the power law distribution [19]. The small-world property is another feature of real-world graphs. It indicates that the diameter of real-world graph is very small, where the diameter of a graph is the length of the shortest path between the two most distanced vertices.

Synthetic graphs are directly generated by code to simulate real-world graphs. Common synthetic graphs methods include Random, R-MAT [17] and SSCA#2 [18]. The Random scheme directly creates vertices and generates directed edges between any two vertices. R-MAT requires four parameters (a, b, c, d) summed to 1 to respectively represent the probability of selecting four quadrants in the next iteration. Graphs created by SSCA#2 simulate the real-world graphs according to the clique characteristics. It uses an adjacency list to represent vertices and adjacency arrays with auxiliary arrays to store the created edges. It inserts edges between vertices according the clique parameters set by users.

### 2.2 Sequential Algorithms of SCC Detection

Most of classic sequential algorithms are based on depth-first search(DFS) traversal for DFS only needs to search each edge once. For example, Tarjan's algorithm is inspired by DFS, and it traverses vertices in a way of DFS except for two added mark arrays *Low* recording the minimum connected vertex and $DFN$ recording the accessed order of current vertex. The main difference between Tarjan's and DFS is: (1) If the adjacent vertex of the current vertex has not been traversed or has been stored in the stack, then the *Low* value of the adjacent vertex is updated; (2) In the backtracking process, if the *Low* value and $DFN$ value of the current vertex are equal, it means that all the vertices whose *Low* value is equal to the current vertices form an SCC, and these vertices are sequentially stored in the stack. Then we can obtain the SCC by popping the stack.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2

### 2.3 Parallel Algorithms of SCC Detection

Parallel SCC detection is mostly based on parallel breadth-first search (BFS) traversal algorithms, and GPU can significantly accelerate parallel traversal [20]. Barnat et al. [8] classified the parallel SCC detection algorithms as follows: FB algorithm [21, 9], Coloring algorithm, and Recursive OBF algorithm [21], and concluded that the FB algorithm is better than the other two algorithms. The theoretical basis is the following Theorem 1. In Theorem 1, $FW_G(u)$ represents the forward readability closure of vertex u, which is the set of vertices reachable from $u$. Correspondingly, $BW_G(u)$ represents the backward reachable closure of vertex $u$, which is the set of vertices reachable to $u$.

**Theorem 1.** *Let $G = (V, E)$ be a directed graph with a vertex $u \in V$. Then $FW_G(u) \cap BW_G(u)$ is an SCC containing $u$. Moreover, every other SCC in $G$ is contained in either $FW_G(u) \setminus BW_G(u)$, $BW_G(u) \setminus FW_G(u)$, or $V \setminus (FW_G(u) \cup BW_G(u))$.*

The main process of the algorithms [11–15] based on FB algorithm is selecting a vertex as the pivot and performing forward BFS and backward BFS from this vertex. According to subgraphs formed by traversal, it can obtain one SCC and three partitions, and the next traversal can be performed in each partition separately until all the SCCs are detected. William et al. [21] proposed the FB-Trim algorithm, which adds a Trim scheme to detect the 1-SCCs separately.

## 3 Our Approach

In this paper, we propose an effective parallel algorithm of SCC detection, and we introduce it in detail in this section.

### 3.1 Graph Representation

Aligned memory access and merged memory access are the main characteristics of GPU memory access. And compressed spare row (CSR) is a suitable graph representation for parallel calculation on GPU, which is also a common graph representation for SCC detection on GPU [11, 12]. As shown in Fig.1, the array $C$ saves the ID of vertices linked from the current vertex in order, and the *i-th* element of the array $R$ holds the starting position of all the vertices linked from the *i-th* vertex in array $C$. Therefore, all the linked vertices of the *i-th* vertex are saved from the $R[i]$-*th* element to the previous of the $R[i+1]$-*th* element in the array $C$. In our algorithm, we define an integer-type array $M$ to mark the state of each vertex when detecting SCCs.

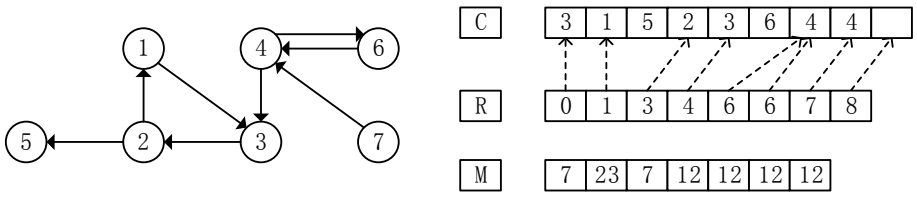
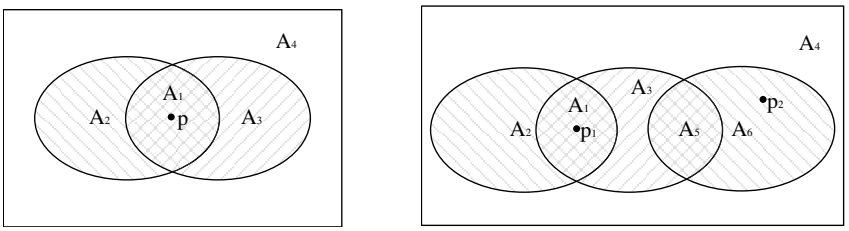



**Fig. 1.** CSR represent of an example

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2

### 3.2 SCC Detection with Multi-partition

Many algorithms [11–14] propose improvements based on the FB-Trim algorithm [10] in different aspects, which can accelerate the FB-Trim algorithm, but they do not consider how to improve the detection of medium-sized SCCs, that is, the SCCs expect the largest SCC and 1-SCCs. According to Theorem 1 in Section




(a) FB-Trim (b) our algorithm

**Fig. 2.** Partitions in each traversal

2.3, the generated partitions of parallel SCC detection algorithm is shown in Fig.2(a), where $p$ is the selected pivot, vertices traversed by the pivot forward BFS constitute the set $\{A_1, A_2\}$, vertices traversed by the pivot backward BFS constitute the set $\{A_1, A_3\}$. Then $A_1$ is an SCC, and $A_2$, $A_3$, $A_4$ are three partitions that can respectively perform SCC detection on them. Therefore, it can detect $3^{n-1}$ SCCs at most on the *n-th* traversal of SCC detection. Assuming that the number of these medium-sized SCCs is $x$, then the minimal traversal number of the FB-Trim algorithm is $O(\log_3 x)$ under ideal conductions.

We can get the following conclusion from Theorem 1: (1) After forward or backward BFS traversal by an arbitrary vertex, a graph will be divided into two partitions, then each SCC of this graph belongs to one of the two partitions. There is no SCC cross the partitions, that is, if a part of the SCC belongs to one of these partitions, the rest of the SCC belongs to this partition, too. (2) For the partitions formed by the forward or backward traversals of any two vertices, each partition independently contains some SCCs, and there is no SCC cross the partitions, too. The proof is as follows:

*Proof.* (1) Let $G = (V, E)$ be a directed graph with an arbitrarily selected vertex $v_0 \epsilon V$. Vertices that are forward traversed by $v_0$ constitute the partition $V_1$, and the remaining vertices constitute the partition $V_2$, so $V_2 = V/V_1$.

Assuming that there is an SCC $S$ crossing the two partitions, and $v_1$,$v_2$ are two vertices of $S$, $v_1 \epsilon V_1$, $v_2 \epsilon V_2$. In partition $V_1$, $v_0$ can traverse forward to $v_1$. In $S$, $v_1$ can traverse forward to $v_2$. Because forward traversal is transferable, $v_0$ can traverse forward to $v_2$. Then we can get $v_2 \epsilon V_1$. It is a contradictory to the assumption. Therefore, there is no SCC that can make a crossover in the two partitions formed by traversing.

(2) Let $G = (V, E)$ be a directed graph, and forward or backward traversals of any two vertices $v_1$, $v_2$ can obtain four partitions: vertices that can be traversed by $v_1$ and $v_2$ constitute partition $A_1$, vertices that can be traversed by $v_1$ but can't be traversed by $v_2$ constitute the partition $A_2$, the vertices that can be traversed by $v_2$ but can't traversed by $v_1$ form partition $A_3$, and the vertices that can't be traversed by $v_1$ and $v_2$ constitute partition $A_4$.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2

Suppose that $S$ is an arbitrary SCC in graph $G$. According to the conclusion obtained in (1), from the perspective of partitions formed by $v_1$, $S\epsilon\{V_1, V_2\}$ or $S\epsilon\{V_3, V_4\}$, so we only need to prove that $S$ does not have the crossover in $V_1$ and $V_2$, or $V_3$ and $V_4$. From the perspective of partitions formed by $v_2$, $S\epsilon\{V_1, V_3\}$ or $S\epsilon\{V_2, V_4\}$. So $S$ does not have the crossover in $V_1$ and $V_2$, and $S$ does not have the crossover in $V_3$ and $V_4$, either. Therefore, $S$ does not cross between any two of $V_1$, $V_2$, $V_3$, and $V_4$.

According to the above conclusions, after forward BFS of the pivot, we select another pivot called the vice pivot outside the formed partition. Then the pivot and the vice pivot perform backward BFS simultaneously. The selection of vice pivot has following advantages: (i) The pivot and the vice pivot perform backward BFS traverse with the same method, which can increase the parallelism of BFS to take advantage of the GPUs. (ii) The vice pivot is selected outside of the partition formed by forward traversal of the pivot. So the vice pivot is not on the newly formed SCC. (iii) The partition formed by forward traversal of the pivot does not intersect with the partition formed by backward traversal of the vice pivot, the reason is similar to the conclusion (1). So each partition can intersect with at most one other partition.

Partitions formed by our method are shown in Fig.2(b), where $P_1$ is the pivot, and forward BFS traversal of the pivot generates partition $\{A_1, A_2\}$, and then a vice pivot $P_2$ is selected outside this partition. $P_1$ and $P_2$ is simultaneously traversed by backward BFS, where the backward BFS traversal of pivot $P_1$ generates partitions $\{A_1, A_3, A_5\}$, and the backward BFS traversal of vice pivot $P_2$ generates partitions $\{A_5, A_6\}$. Among the six partitions, $A_1$ is an SCC, and the rest partitions can perform SCC detection respectively in the next traversal. By this way, the minimum traversal of the FB-Trim algorithm based on Theorem 1 can be reduced from $\mathrm{O}(\log_3 x)$ to $\mathrm{O}(\log_5 x)$, which effectively reduces the number of traversals of the algorithm. Our approach is presented in Algorithm 1.

**Algorithm 1** FB-Trim-MP SCC Detection Algorithm

```
procedure FB-Trim-MP(G(V, E), SCC)
  /* Phase 1 */
  Trim (G, SCC, M, P)
  Pivot-choose1 (G, SCC, M, P)
  do in parallel
    Forward-traverse (G, SCC, M, P)
    Backward-traverse (G, SCC, M, P)
  end do
  Update-state (G, SCC, M, P)
  repeat
    Trim (G, SCC, M, P)
  until no 1-SCC generated
  Trim2 (G, SCC, M, P)
  repeat
    Trim (G, SCC, M, P)
  until no 1-SCC generated
```

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2

WCC-detect $(G, SCC, M, P)$
/* Phase 2 */
**repeat in parallel**
Forward-traverse $(G, SCC, M, P)$
vicePivot-choose $(G, SCC, M, P)$
Backward-traverse $(G, SCC, M, P)$
Trim $(G, SCC, M, P)$
Update-state $(G, SCC, M, P)$
Pivot-choose2 $(G, SCC, M, P)$
**until** no SCC generated
**end procedure**

### 3.3 Selection of Pivot and Vice Pivot

Shrinivas et al. [13] took the vertex with the largest product of in-degree and out-degree as the pivot. They first apply an array *pivots* to store the pivot of the *i-th* partition in $pivots[i]$, then calculates the product of the in-degree and out-degree of the pivot and the current vertex in parallel. If the product of the current vertex is larger than that of the pivot, they will save current vertex's ID in $pivots[i]$. However, all the threads in the GPU calculate products and store pivots in parallel, so in the *i-th* partition, all the vertices whose products of in-degree and out-degree are larger than the product of the pivot will store their ID in $pivots[i]$ almost at the same time. Therefore, these storage operations are disordered, and they can only guarantee that the product of the vertex stored in $pivots[i]$ is larger than that of the vertex originally stored in $pivots[i]$, but it is not the largest. we propose a method by iteratively executing the kernel function until $pivots[i]$ is no longer changed to guarantee the obtained pivot has the largest product. The algorithm is shown in Algorithm 2.

This scheme can significantly improve the accuracy of locating the maximum SCC, but it is time consuming. So after selecting the first pivot by *Pivot-choose*1, we take another pivot selection method that randomly selects a vertex in each partition as the pivots in the *phase* 2, and it is also used to select the vice pivots. The random selection method is to store all the vertices of the same partition to the same memory location in parallel, and we take a method to prevent the partitions traversed by the pivot and vice pivot from approaching each other. When selecting the pivot, there is an atomic operation to prevent the other vertices from being stored after the first vertex is stored. When selecting the vice pivot, there is no atomic operation to block the storage of any vertices. Therefore, the pivot is the first vertex stored in the fixed position, and the vice pivot is the last stored vertex so that the pivot and the vice pivot will not be close to each other.

**Algorithm 2** choose pivot Algorithm

**procedure** PIVOT-CHOOSE1$((G, SCC, M, P))$
**repeat**
Pivotchoose-Kenel $(G, SCC, M, P, pivots)$
**until** M is not change

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2

```
        update-pivot (M, pivots)
    end procedure

    procedure PIVOTCHOOSE-KENEL((G, SCC, M, P, pivots))
        v ← threaId
        if M(v) is not marked as trimmed then
            u ← pivots(P(v))
            uDegree ← inDegree(u) * outDegree(u)
            vDegree ← inDegree(v) * outDegree(v)
            if vDegree > uDegree then
                pivots(P(v)) ← v
            end if
        end if
    end procedure
```

### 3.4 Improvement Details

In Algorithm 1, we added 2-SCC detection and WCC detection between *phase* 1 and *phase* 2. 2-SCC is a small SCC composed of two vertices, which is also abundant in real-world graphs, and most of them are easy to detect. For each undetected vertex, detect the vertex that it directly links to and directly links from simultaneously. If the two vertices have no in-degree or out-degree, they can form an independent 2-SCC. WCC detection can divide a graph into several disconnected partitions. Firstly, each vertex is initialized to form a WCC by itself, so the WCC ID of a vertex is set to its own ID. Then it is checked whether there is a vertex in the adjacent of current vertex whose WCC ID is smaller than the WCC ID of the current vertex. If it exists, the WCC ID of the current vertex is set to the smaller value, and the above process is iterated until all WCC ID do not change. In *phase* 2, we take the WCC ID as the pivot of this partition directly, which can save the time for selecting the pivots.

### 3.5 Expend algorithms Devshatwar-MP and Li-MP

The algorithm proposed by Devshatwar et al. [13] is based on the FB-Trim algorithm, which can be improved by our algorithm. Compared with the FB-Trim algorithm, Devshatwar's algorithm mainly has following improvements: (1) There are two modes to traverse vertices: vertex-centric and virtual warp-centric; (2) It increase 2-SCC detection and WCC detection; (3) When selecting the pivot of each partitions, it uses the vertex with the maximum product of in-degree and out-degree as the pivot of each partition. Besides applying the multi-partition scheme to the above processes, our Devshatwar-MP algorithm improve the Devshatwar's algorithm at the following points: (1) We still adopt vertex-centric and virtual warp-centric modes; (2) After the WCC detection, the ID of WCC region are directly used as the pivot of current region; (3) We use the pivot selection scheme proposed by Devshatwar et al. to select the pivots and vice pivots.

Li et al. [12] also propose improvements on the FB-Trim algorithm, and we can implement our method on the Li's algorithm. Compared with the FB-Trim

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2

algorithm, Li's algorithm mainly has the following improvements: (1) There are two traversal modes named data-driven and topology-driven, which is similar to virtual warp-centric and vertex-centric in Devshatwar's algorithm. (2) 2-SCC detection and WCC detection are added; (3) It increases operation of loading balance, and adapts Topo-lb in *phase* 1 and Topo in *phase* 2. Besides applying the multi-partition scheme to the above processes, our Li-MP algorithm improve Li's algorithm at the following points: (1) The algorithm framework using Topo-lb mode in *phase* 1 and Topo mode in *phase* 2 is still adopted; (2) After the WCC detection, the ID of each region marked at the WCC detection is directly used as the pivot of the current region; (3) The selection of the pivots and the vice pivots is optimized by the method mentioned in Section 3.3.

## 4 Experimental Evaluation

Graphs used in our experiment include synthetic graphs and real-world graphs. The synthetic graphs are the following three types of graphs generated by *GeorgiaTech.graphgenerator* (GTgraph) [16]: Random, R-MAT [17] and SSCA#2 [18], as shown in Table 1. Real-world graphs come from two commonly used benchmarks [11–13]: SNAP database [22] and Koblenz Network Collection database [23], as shown in Table 2.

**Table 1.** The detailed parameters of generated graphs

| Type | name | Vertices | Edges | Parameters |
|---|---|---|---|---|
| R-MAT | GT-rmata | 10,000,000 | 100,000,000 | a=0.25,b=0.25, c=0.25,d=0.25 |
| R-MAT | GT-rmatb | 10,000,000 | 100,000,000 | a=0.45,b=0.15, c=0.15,d=0.25 |
| Random | GT-randa | 10,000,000 | 100,000,000 | p=0.8 |
| Random | GT-randb | 10,000,000 | 100,000,000 | p=0.6 |
| SSCA#2 | GT-sscaa | 10,000,000 | 80,771,507 | maxCliqueSize=10, maxParalEdges=2 |
| SSCA#2 | GT-sscab | 10,000,000 | 95,068,514 | maxCliqueSize=12, maxParalEdges=2 |

**Table 2.** The details of real-world graphs

| name | Vertices | Edges | average degree | maximum degree |
|---|---|---|---|---|
| Amazon0302 | 400,727 | 3,200,440 | 7.99 | 2,757 |
| Amazon0312 | 262,111 | 1,234,877 | 4.71 | 425 |
| Amazon0505 | 410,236 | 3,356,824 | 8.18 | 2,770 |
| Slashdot0811 | 77,360 | 905,468 | 11.70 | 5,048 |
| NotreDame | 325,727 | 1,497,134 | 4.60 | 10,721 |
| Google | 875,713 | 5,105,039 | 5.83 | 6,353 |
| pokec-relationships | 1,632,803 | 30,622,564 | 18.75 | 20,518 |
| LiveJournal | 4,874,571 | 68,993,773 | 14.15 | 22,889 |

### 4.1 Experiment Setup

We compare five implementations including: (1) Tarjan: Tarjan's sequential SCC detection algorithm is a classic and representative sequential algorithm [3, 8]; (2) Barnat: Barnat's SCC detection method is a classical parallel algorithm, which is

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2

compared in many improved algorithms [8, 11–13]; (3) FB-Trim-MP: our parallel SCC detection algorithm with multi-partition; (4) Devshatwar-MP: Devshatwar's algorithm [13] improved by our multi-partition algorithm. (5) Li-MP: Li's algorithm [12] improved by our multi-partition algorithm. We use gcc and nvcc with the -O3 optimization option for compilation along with -arch=sm_37 when compiling for the GPU. We execute all the benchmarks 10 times and collect the average execution time to avoid system noise.

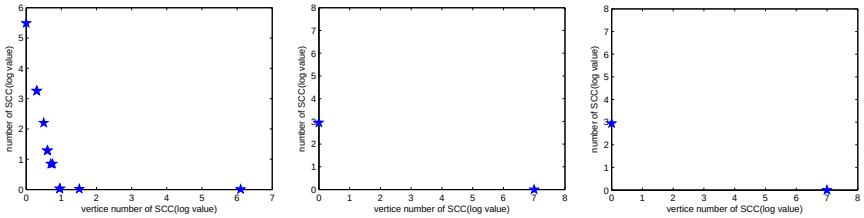
### 4.2 Effectiveness of Multi-partition scheme

In order to verify the effectiveness of the multi-partition scheme, we improve the Barnat's algorithm by our multi-partition scheme, and the other operations are not changed. The comparison of execution time and the number of traversals between Barnat algorithm and the improved algorithm (Barnat-MP) are summarized in the following Tables 3. It can be seen that the Barnat-MP doesn't accelerate at R-MAT and Random graphs. Because these two kinds of graphs only contain one large SCC and lots of 1-SCCs, and they don't need many traversals to process the medium-sized SCCs. However, in real-world graphs and SCCA#2 graphs, there are lots of medium-sized SCCs. It requires multiple traversals to detect these SCCs. For example, the Barnat's algorithm requires 12225 traversals to detect all SCCs in real-world graph *LiveJournal*. Therefore, in real-world graphs, the number of partitions generated by each traversal has a great influence on the number of traversals in SCC detection. As shown in Table 3, the Barnat-MP algorithm can reduce the number of traversals to 50% to 64% of the Barnat algorithm and the speed of Barnat-MP algorithm can also be increased to 1.3 × to 13 × compared with the Barnat algorithm. Therefore, the multi-partition scheme can significantly accelerate the parallel algorithm of SCC detection.

**Table 3.** comparison of execution time (left) and number of iterations (right)

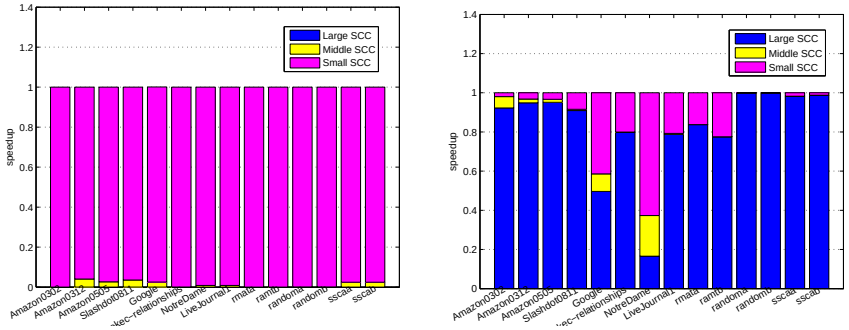
| | Comparison of traversal number | | | Comparison of running time | | |
|---|---|---|---|---|---|---|
| name | Barnat | Barnat-MP | iteration proportion | Barnat(s) | Barnat-MP(s) | time proportion |
| Amazon0302 | 426 | 242 | 56.81% | 0.365 | 0.254 | 143.70% |
| Amazon0312 | 1,250 | 668 | 53.44% | 0.829 | 0.588 | 140.99% |
| Amazon0505 | 326 | 170 | 52.15% | 0.277 | 0.199 | 139.20% |
| Slashdot0811 | 240 | 121 | 50.42% | 0.0976 | 0.059 | 165.42% |
| NotreDame | 1,180 | 611 | 51.78% | 0.814 | 0.641 | 126.99% |
| Google | 5,371 | 3,394 | 63.19% | 6.486 | 4.645 | 139.63% |
| pokec-relationships | 1,080 | 684 | 63.33% | 1.317 | 1.017 | 129.50% |
| LiveJournal | 12,225 | 7,539 | 61.67% | 33.286 | 25.236 | 131.90% |
| rmata | 1 | 1 | 100.00% | 0.263 | 0.293 | 89.76% |
| rmatb | 1 | 1 | 100.00% | 0.277 | 0.312 | 88.78% |
| randoma | 1 | 1 | 100.00% | 0.289 | 0.299 | 96.66% |
| randomb | 1 | 1 | 100.00% | 0.289 | 0.299 | 96.66% |
| sscaa | 2,236 | 1,397 | 62.48% | 9.733 | 0.727 | 1,338.79% |
| sscab | 1,520 | 973 | 64.01% | 6.928 | 0.701 | 988.30% |

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2




(a) pokec-relationships (b) rmata (c) randoma

**Fig. 3.** The number distribution of SCCs of different graphs




(a) proportion of SCCs (b) proportion of vertices in SCCs

**Fig. 4.** The proportion of SCCs and their vertices

### 4.3 Analysis of SCC distribution

Before the performance analysis, we compare and analyze the SCC distribution of the graphs used in our experiment. Fig.3 shows the distribution of SCCs in synthetic graphs and real-world graphs. In order to clearly show the number of medium-sized SCCs, we use the natural logarithm of the number of SCCs as the ordinate value. Fig.4 compares the number of the largest SCCs, small SCCs including 1-SCCs and 2-SCCs, and remaining medium-sized SCCs, and it also compares the number of vertices contained in these SCCs. It is clear in Fig.3 that synthetic graphs of R-MAT and Random are different from real-world graphs in SCC distribution. In R-MAT and Random synthetic graphs, there are only two types of SCCs: one large SCC and lots of small 1-SCCs. It also verifies the reason why SCC detection on the R-MAT and Random graphs is decelerated in Section 4.2. There are many medium-sized SCCs in real-world graphs. In Fig.4, we divide SCCs into three categories. It can be found from the R-MAT and Random graphs in Fig.4 that medium-sized SCCs don't account for any proportion on no matter the number of SCCs or the number of vertices. In real-world graphs and SCCA#2 graphs, the number of medium-sized SCCs and the number of their vertices account for a little proportion on SCCs number and vertices number. For all SCCs, the number of vertices contained in the largest SCC accounts for a large proportion. So it is necessary to be specifically detected. *Phase* 1 in our algorithm mainly deals with the largest SCC. Each small SCC only contains one or two vertices, but the number of small SCCs is large. So $Trim$ and $Trim2$ in our algorithm are used to deal with these vertices. The

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2

medium-sized SCCs are not large in vertices number and SCCs number, but they need to be traversed many times, which is the main part accelerated by our multi-partition method.

### 4.4 Performance Analysis

We implement the five algorithms mentioned in Section 4.1 on six synthetic graphs and eight real-world graphs. In order to clearly display the experiment results, we normalize the execution time of all algorithms by that of the Tarjan algorithm and display the speedup in Fig.5. The result shows that our algorithm achieves an average acceleration of 8.8 × and 21 × compared to Tarjan's algorithm and Barnat's algorithm. Li-MP, Shrinovas-MP and our algorithm are significantly faster than Tarjan's algorithm in most graphs, which can reach 10 × or even 20 × speedup. Since there is no medium-sized SCCs, the speed of them is slightly lower than Barnat's algorithm, but when detecting SSCA#2 graphs and real-world graphs, Barnat's algorithm is poor. In these graphs, the speed of Barnat's algorithm is even lower than that of Tarjan's algorithm, while Li-MP, Shrinovas-MP and our algorithm can still maintain a certain acceleration in most cases.

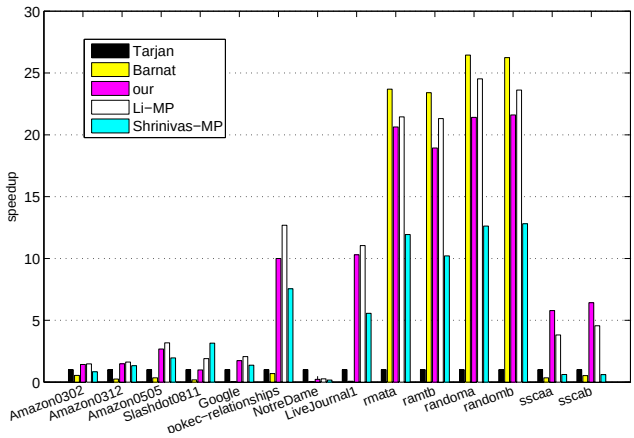



**Fig. 5.** Acceleration of various algorithms compared with Tarjan's algorithm

Compared with Li-MP and Shrinovas-MP, our algorithm is not inferior. Li-Mp is consistent with our algorithm except for loading balance. Fig.5 shows that the Li-MP is faster than our algorithm on most graphs, and the average acceleration ratio is 10%. The largest acceleration is on graph *Slashdot*, where the speed of Li-MP is about 1.9 × of our algorithm. However, on SSCA#2 graphs, the speed of Li-MP is only 0.66 × - 0.71 × of our algorithm. Therefore, the loading balance can accelerate most graphs but not all. Compared with our algorithm, Shrinivas-Mp adds virtual warp-centric mode in *phase* 1, and uses its own pivot selection scheme. The operation of virtual warp-centric in Shrinivas-MP is similar to the data-driven mode in Li-MP. As shown in Fig.5, Shrinivas-MP is not as fast as Li-Mp and our algorithm on most graphs. This is mainly because that the pivot selection scheme of Shrinivas-MP takes some time in *phase* 2, so it is necessary to optimize the pivot selection.

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2

It can be concluded in Fig.5 that parallel SCC detection algorithms show better acceleration on large graphs. The Tarjan's algorithm only processes one edge at a time, so its runtime is positively correlated with the scale of graphs. On real-world graphs *Amazon*0312, *Amazon*0302, *Amazon*0505, and *Google*, whose vertices are not more than 1M, Li-MP, Shrinovas-MP and our algorithm only accelerate several times, and the detection speed on *NotreDame* with only 0.3M vertices is even lower than that of Tarjan algorithm. This is mainly because the scale of graphs is not large, and plenty of threads on GPU are idle. The real-world graphs *LiveJournal* and *pokec-relationships* are commonly used in the experiments of many parallel SCC detection [11–13], and these algorithms can achieve about 10 × acceleration on these two graphs. The size of the two graphs is significantly larger than the previous graphs. Therefore,only when the graphs is large enough, can the parallel algorithms make full use of all threads of the GPUs.

## 5 Conclusion

Graph computing is important to abstract and solve problems in the interconnected world. Parallel SCC detection is an important part of graph computing accelerating algorithms. In this paper, we propose a parallel algorithm of SCC detection with multi-partition to reduce the number of traversals by increasing the number of partitions generated in each traversal. We select a vice pivot after the forward traversal of the pivot and make it traverse backward together with the pivot, which can increase the number of partitions in each traversal. And we also improved the selection method of pivots and vice pivots. We combine 2-SCC detection as well as WCC detection and make the vertices marked at WCC detection as the pivots to accelerate the algorithm. Experimental results demonstrate that our algorithm outperforms existing SCC detection algorithms on synthetic graphs and real-world graphs. The proposed algorithm achieves an average acceleration of 8.8 × and 21 × over Tarjan's algorithm and Barnat's algorithm. In the future, we will further discuss the structure and proportional of the largest SCC with other SCCs and combine them with various fine-grained acceleration schemes, and adopt different schemes according to the internal features of various graphs to ensure the efficiency and the stability of the algorithm.

## References

1. Xie, A., Beerel, P.A.: Implicit enumeration of strongly connected components and an application to formal verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **19**(10), 1225–1230(2000)
2. Simona, O.:On distributed verification and verified distribution, Ph.D. dissertation, Center for Mathematics and Computer Science (CWI), 2004
3. Tarjan, R.: Depth-first search and linear graph algorithms. Siam J of Comput **1**(4), 146–160 (1972)
4. Dijkstra, E.W.: A Discipline of Programming. 1st edn. Prentice Hall, ENGLEWOOD CLIFFS (1976)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.:Introduction to Algorithms. 3rd edn. The MIT Press, Cambridge (2009)

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2

6. Reif, J.H.: Depth-first search is inherently sequential. Information Processing Letters **20**(5),229-234 (1985)
7. Barnat, J., Chaloupka, J., Jaco, V.D.P.: Distributed Algorithms for SCC Decomposition. Journal of Logic and Computation **21**(1),23-44 (2011)
8. Barnat, J., Bauch, P., Brim, L., Ceska, M.: Computing Strongly Connected Components in Parallel on CUDA. In: Sussman, A., Mueller, F., Beaumont, O., Kandemir, M.T., Nikolopoulos, D.(eds.) IPDPS 2011. pp. 544–555. IEEE(2011). http://dx.doi.org/10.1109/ipdps.2011.59
9. Fleischer, L.K., Hendrickson, B., Pinar, A.: On Identifying Strongly Connected Components in Parallel. In: Weems, C., Mesirov, J., Schopf, J., Gottlieb, A.(eds.)IPDPS Workshops 2000. LNCS, vol.1800, pp. 505–511. Springer, Heidelberg(2000). https://doi.org/10.1007/3-540-45591-4_68
10. McLendon III, W., Hendrickson, B., Plimpton, S.J., Rauchwerger, L.: Finding strongly connected components in parallel in particle transport sweeps.In: SPAA 2001. pp. 328-329. ACM, Crete(2011). http://dx.doi.org/10.1145/378580.378751
11. Hong, S., Rodia, N.C., Olukotun, K.: On fast parallel detection of strongly connected components (SCC) in small-world graphs. In: SC 2013, pp. 1–11. ACM,Denver(2013). http://dx.doi.org/10.1145/2503210.2503246
12. Li, P., Chen, X, Shen, J., Fang, J., Tang, T., Yang, C.: High Performance Detection of Strongly Connected Components in Sparse Graphs on GPUs. In: PMAM@PPoPP 2017, pp. 48–57. ACM,Texas (2017).https://doi.org/10.1145%2F3026937.3026941
13. Devshatwar, S., Amilkanthwar, M., Nasre, R.: GPU centric extensions for parallel strongly connected components computation. In: GPGPU@PPoPP 2016, pp. 2–11. ACM, Barcelona(2016). https://doi.org/10.1145%2F2884045.2884048
14. Li, G.H., Zhu, Z., Zhang, Cong., Yang, F.M.: Efficient decomposition of strongly connected components on GPUs. Journal of Systems Architecture **60**(1), 1–10(2014).
15. Aldegheri, S., Barnat, J., Bombieri, N., Busato, F., Ceska, M.: Parametric Multi-step Scheme for GPU-Accelerated Graph Decomposition into Strongly Connected Components. In: Euro-Par Workshops 2016: pp. 519–531. Springer,Grenoble(2016). http://dx.doi.org/10.1007/978-3-319-58943-5_42
16. Madduri, K., Bader, D.A.: GTgraph: A suite of synthetic graph generators. https://github.com/dhruvbird/GTgraph. Last accessed 15 Sep 2012
17. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-MAT: A recursive model for graph mining. In: SDM 2004, pp. 442–446. Society for Industrial and Applied Mathematics, Orlando(2004). http://dx.doi.org/10.1137/1.9781611972740.43
18. Bader, D.A., Madduri, K.: Design and implementation of the HPCS graph analysis benchmark on symmetric multiprocessors. In: HiPC 2005,pp. 465-476. Springer, Berlin(2005) http://dx.doi.org/10.1007/11602569_48
19. Kumar, R., Novak, J., Tomkins, A.: Structure and evolution of online social networks. In: KDD 2006, pp. 611-617. ACM, New York(2006) http://dx.doi.org/10.1145/1150402.1150476
20. Defour, D., Marin, M.: Regularity versus Load-Balancing on GPU for treefix computations. Procedia Computer Science **60**, 309-318 (2013)
21. McLendon III, W., Hendrickson, B., Plimpton, S.J., Rauchwerger, L.: Finding Strongly Connected Components in Distributed Graphs. Journal of Parallel and Distributed Computing(JPDC)**65**(8), 901–910(2005).
22. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data. Last accessed Jun 2014
23. Koblenz network collection. http://konect.uni-koblenz.de/. Last accessed 25 Apr 2018

ICCS Camera Ready Version 2019
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-22747-0_2