On Fast Parallel Detection of Strongly Connected Components (SCC) in Small-World Graphs

Sungpack Hong[2], **Nicole C. Rodia[1]**, and Kunle Olukotun[1]

[1]Pervasive Parallelism Laboratory, Stanford University

[2]Oracle Labs

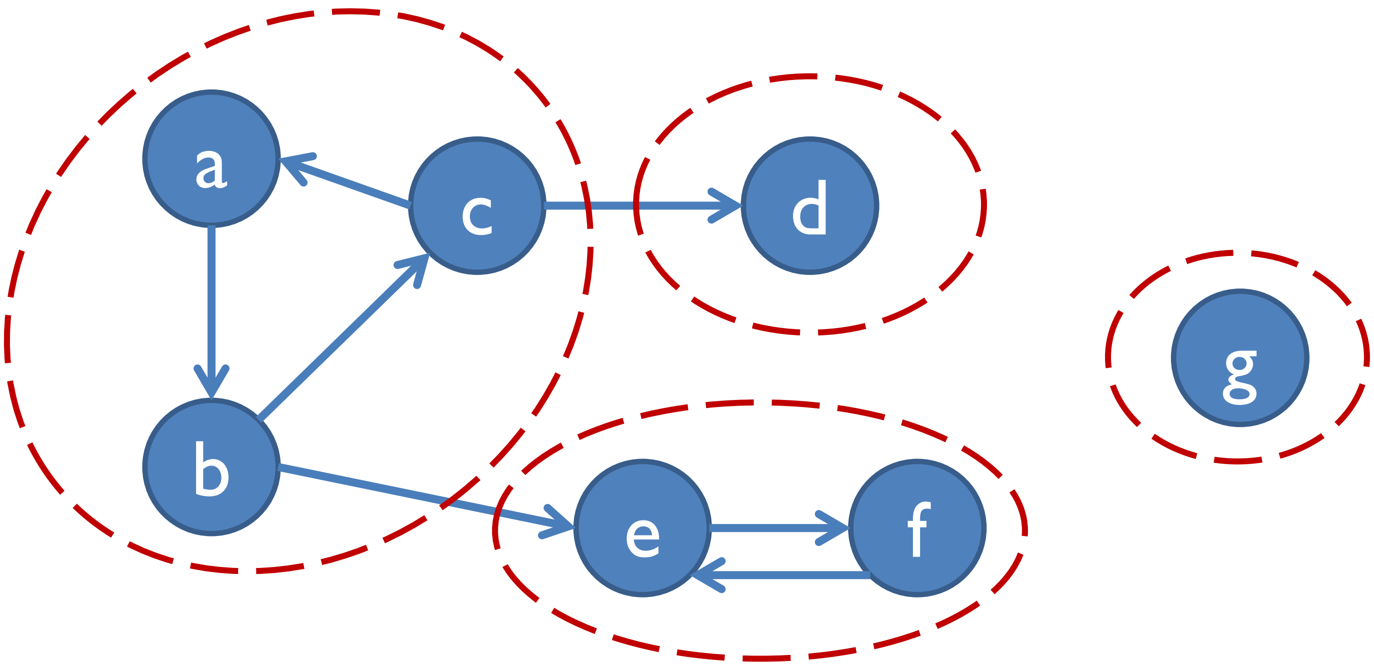
SC '13 – November 21, 2013

Outline

- SCC Background and Motivation
- Shortcomings of Existing Algorithm and Our Solutions
- Experimental Results

Strongly Connected Components (SCC)

- In a directed graph, an SCC is a maximally connected subgraph with a path in **both** directions between any two nodes



Strongly Connected Components (SCC) Applications

- Analyze and extract information from graphs
  - Characterize graph structure
  - Identify core of graph

Large Graphs

- Society, Internet, Biology, Communication, Economy, etc.

Source: Facebook Engineering and Facebook 2012 Annual Report

SCC on Large Graphs

- Datasets contain millions to billions of nodes (n) and billions of edges (m)
- Fastest sequential algorithms to compute SCC require $O(n + m)$ work

→ SCC on large graphs will take a long time!

Parallel SCC Detection

Q: How to make a faster SCC detection algorithm to compute on large graphs?

A: PARALLELIZE!

Existing Algorithms

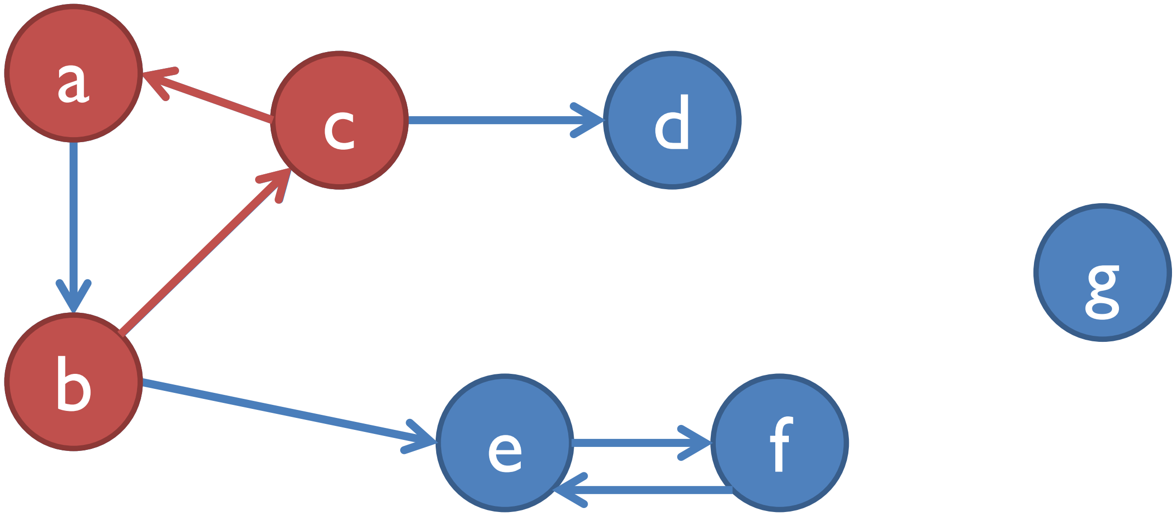
- Optimal **sequential** algorithm
  - Tarjan's Algorithm [Tarjan, SIAM 1972]
  - Cannot be parallelized effectively due to depth-first search (DFS)
- Forward-Backward-Trim **parallel** algorithm
  - Recursive application of reachability

    [Fleischer et al., IPDPS 2000]
  - Trim of trivial SCCs

    [McLendon et al., Parallel & Dist. Computing 2005]

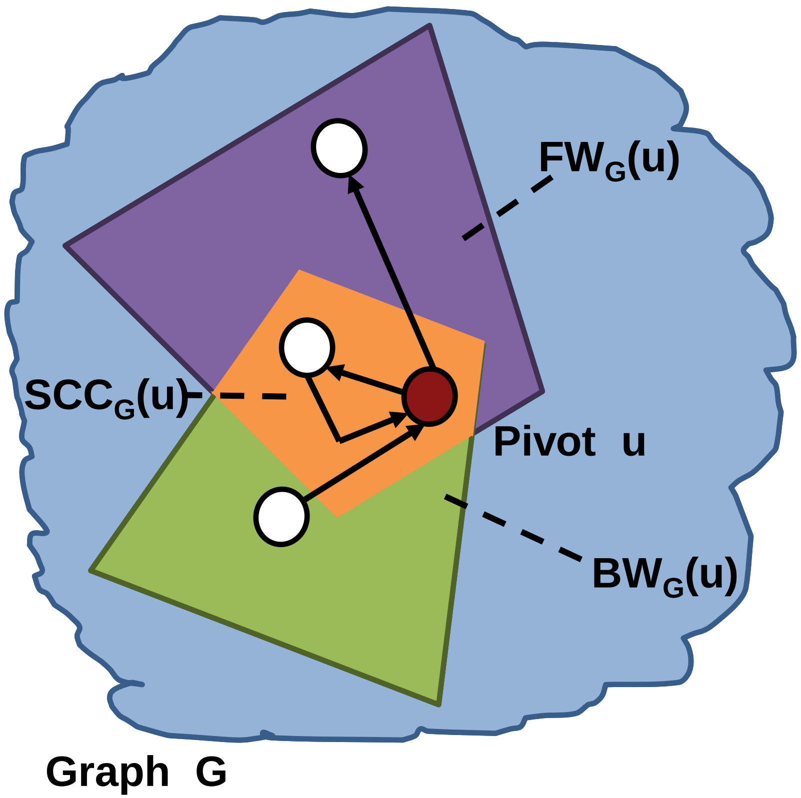
FW-BW-Trim Algorithm: Reachability

- Node a is **reachable** from node b if there is a path from b to a



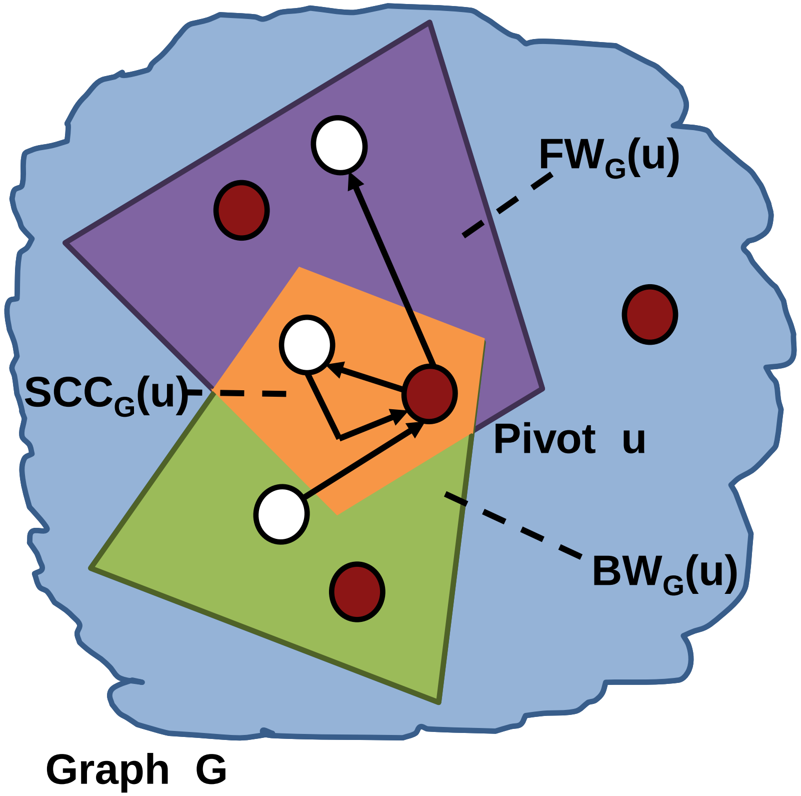
FW-BW-Trim Algorithm: Reachability

- Four partitions
  - $FW_G(i) \cap \mathrm{BW}_G(i)$ [SCC]
  - $FW_G(i) \setminus BW_G(i)$
  - $BW_G(i) \setminus FW_G(i)$
  - $V \setminus (FW_G(i) \cup BW_G(i))$
- Additional SCCs must be completely contained within one of the three additional partitions



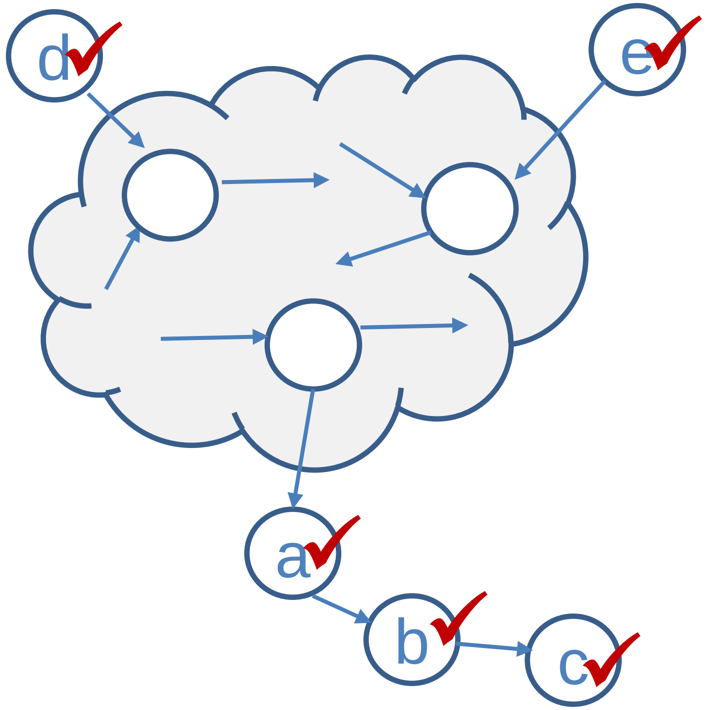
FW-BW-Trim Algorithm: Reachable Set Recursion

- Recursively apply the algorithm to each of the three partitions created besides the pivot's SCC
- Utilizes **task parallelism**



FW-BW-Trim Algorithm: Trimming

- Can identify trivial SCCs (size 1) by looking only at the number of neighbors
  - If the node has in-degree=0 or out-degree=0, it is a size 1 SCC
- Repeat iteratively
- Implement in parallel on disconnected nodes



FW-BW-Trim Algorithm

Apply iterative Trim step

Choose ANY node in the graph

Calculate forward & backward sets

New SCC is intersection of FW & BW sets

Algorithm 1: FW-BW-Trim(G, SCC)

In-Out: G: a graph (a subgraph of the original input graph)
In-Out: SCC: a collection of node sets; each set corresponds to an SCC of the original graph

Trim(G, SCC)
if $|Nodes(G)| = 0$ **then return**;
$u \leftarrow$ pick any node in G /* pivot */
$FW \leftarrow$ Forward-Reach(G, u)
$BW \leftarrow$ Backward-Reach(G, u)
$S \leftarrow FW \cap BW$
$SCC \leftarrow SCC \cup \{S\}$
begin in parallel
  FW-BW-Trim($FW \setminus S$, SCC)
  FW-BW-Trim($BW \setminus S$, SCC)
  FW-BW-Trim($G \setminus (FW \cup BW)$, SCC)

Recursively apply algorithm to each partition

Outline

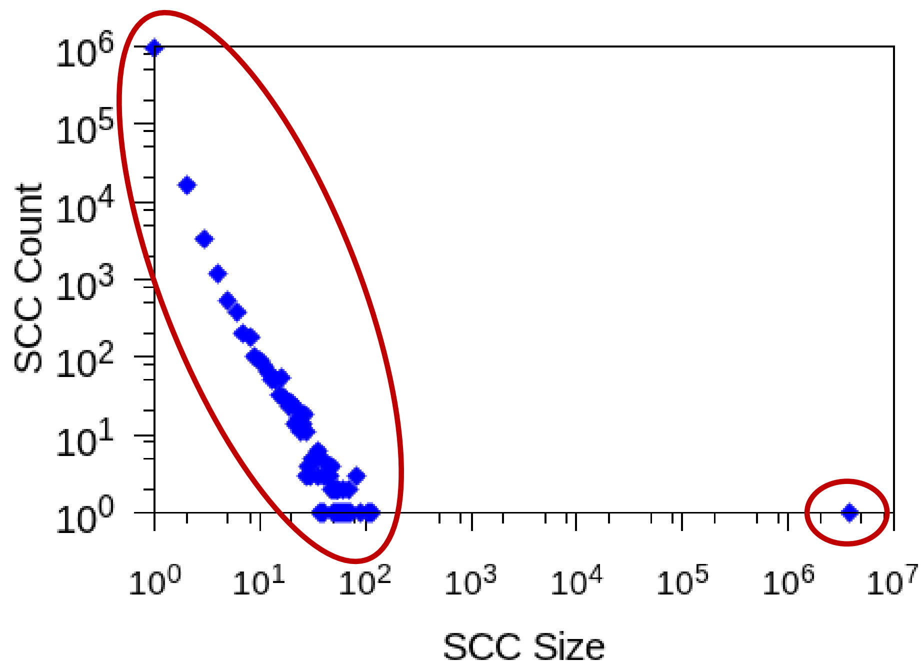
- SCC Background and Motivation
- Shortcomings of Existing Algorithm and Our Solutions
- Experimental Results

Real-World Graphs and the Small-World Property

- Social networks, web graphs, citation networks
- Relevant properties
  - Small-world property (small diameter)
  - Giant SCC size $O(N)$
  - Skewed SCC size distribution
    - Small SCCs are more frequent than large SCCs

Example Small-World Graph: LiveJournal

- N = 4,848,571; M = 68,993,773
- Estimated diameter = 18
- Largest SCC size = 3,828,682 (79% of all nodes)



Shortcomings of the FW-BW-Trim Algorithm

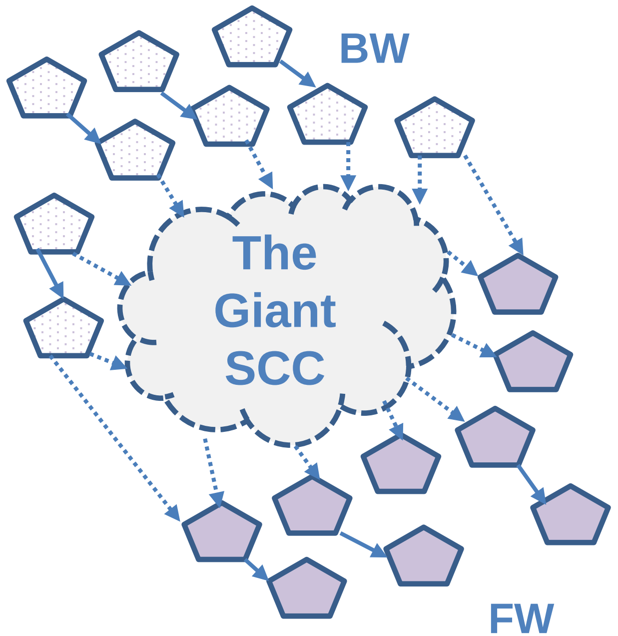
- High probability that we initially pick a pivot node in the giant SCC
- Giant SCC is likely identified at the beginning by a single thread
- Other threads idle because no other tasks yet

→ Workload imbalance

→ Insufficient parallelism

Our Algorithm Extensions
Method 1: Two-Phase Parallelization

- Adds **data parallelism**
  - All threads work on the same partition of the graph to find reachable sets
- Implement with parallel breadth-first search (BFS)



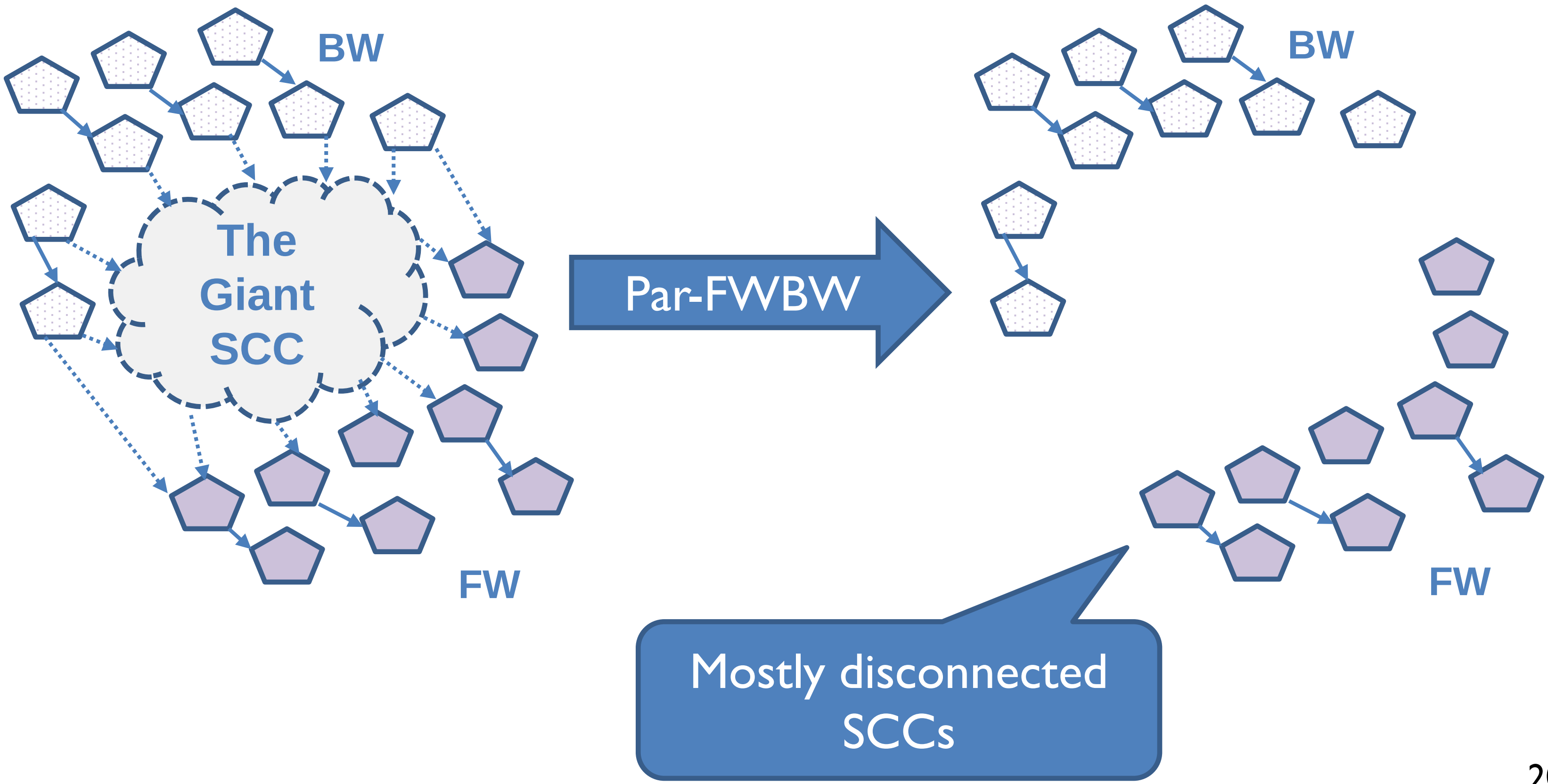
Method 1: Two-Phase Parallelization

```
FW-BW-Trim(G):
// Data parallel
Trim(G)

// Task parallel
Recur-FWBW(G)
```

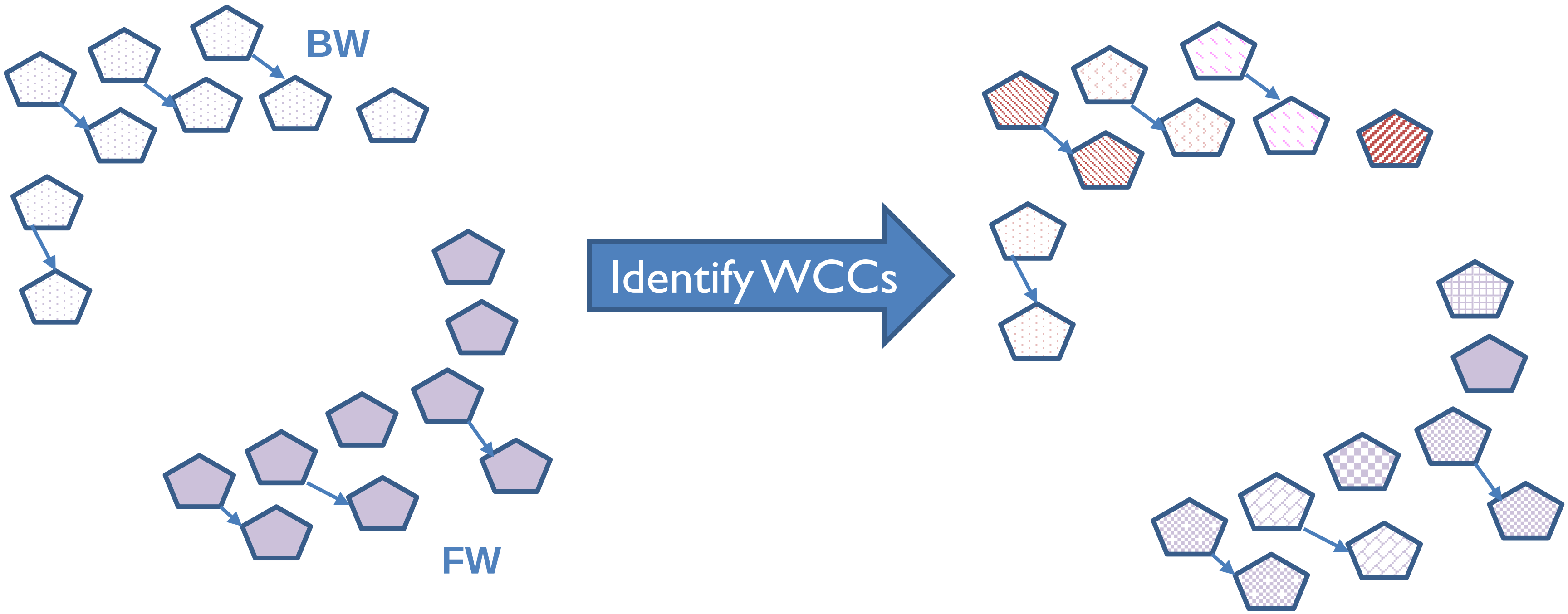
```
Method1(G):
// Data parallel
Trim(G)
Par-FWBW(G)
Trim(G)
// Task parallel
Recur-FWBW(G)
```

Shortcomings of Method 1

- Insufficient tasks in the task parallel recursive FW-BW step



Method 2: Weakly Connected Components (WCC)

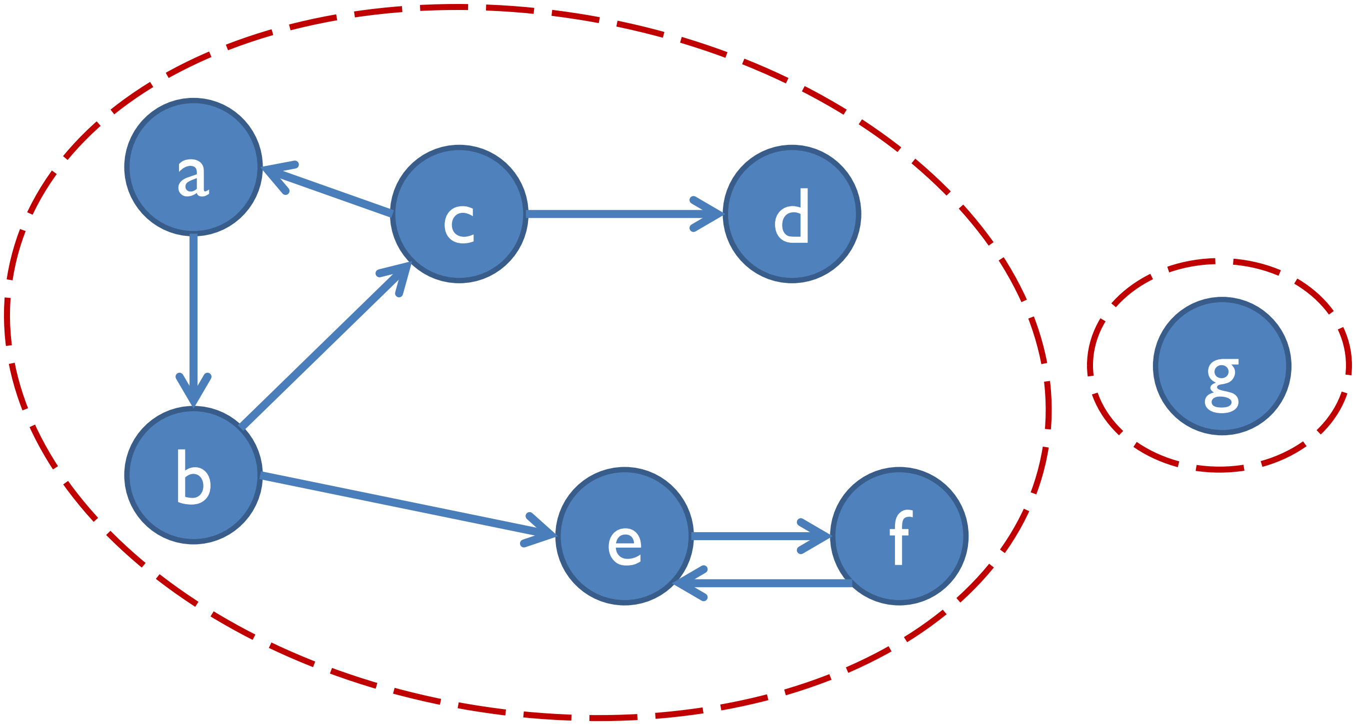


- Now each WCC is a separate parallel task

→ Significantly increases parallelism in recursive FWBW step

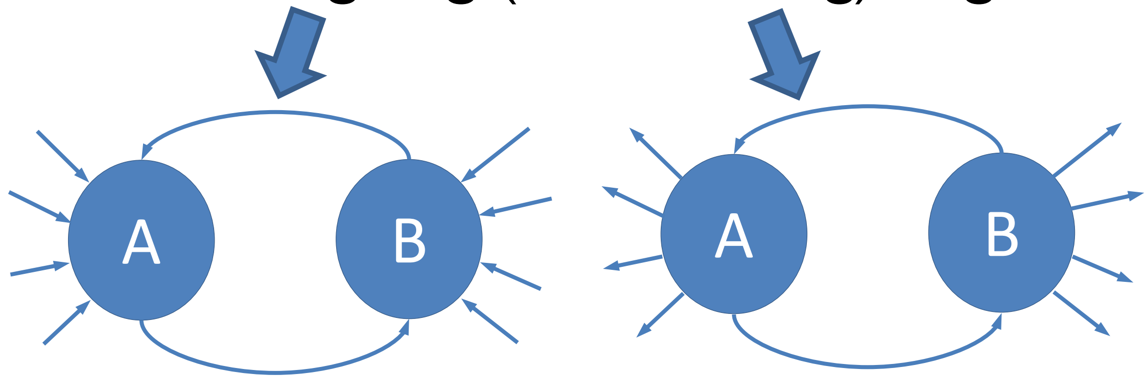
Method 2: Weakly Connected Components (WCC)

- In a directed graph, a WCC is a maximally connected subgraph with a path in **one** direction between any two nodes



Method 2: Trim2

- Parallel detection of a subset of size 2 SCCs
  - Tight loop between nodes A and B
  - No other outgoing (or incoming) edges from A and B



- Apply only once rather than iteratively
  - Higher computational cost than Trim
- Reduces execution time of WCC step by up to 50%

Method 2: WCC + Trim2

```
Method1(G):
// Data parallel
Trim(G)
Par-FWBW(G)
Trim(G)

// Task parallel
Recur-FWBW(G)
```

```
Method2(G):
// Data parallel
Trim(G)
Par-FWBW(G)
Trim'(G)
Par-WCC(G)
// Task parallel
Recur-FWBW(G)
```

```
Trim(G)
Trim2(G)
Trim(G)
```

Outline

- SCC Background and Motivation
- Shortcomings of Existing Algorithm and Our Solutions
- Experimental Results

Experimental Datasets

- Online social networks
  - Flickr
  - Friendster*
  - Twitter
  - Orkut*
- Web link networks
  - LiveJournal
  - Baidu
  - Wikipedia
- Citation
  - US Patents
- Non small-world
  - CA-road*

*the original graph is undirected; we randomly assign a direction for each edge with 50% probability for each direction

Experimental Setup

- Commodity server
  - 2 Intel Xeon E5-2660 (2.20GHz) CPUs
  - Total of 16 cores and 32 hardware threads
  - Total of 20 MB of last-level cache and 256 GB of main memory
- OpenMP threading library

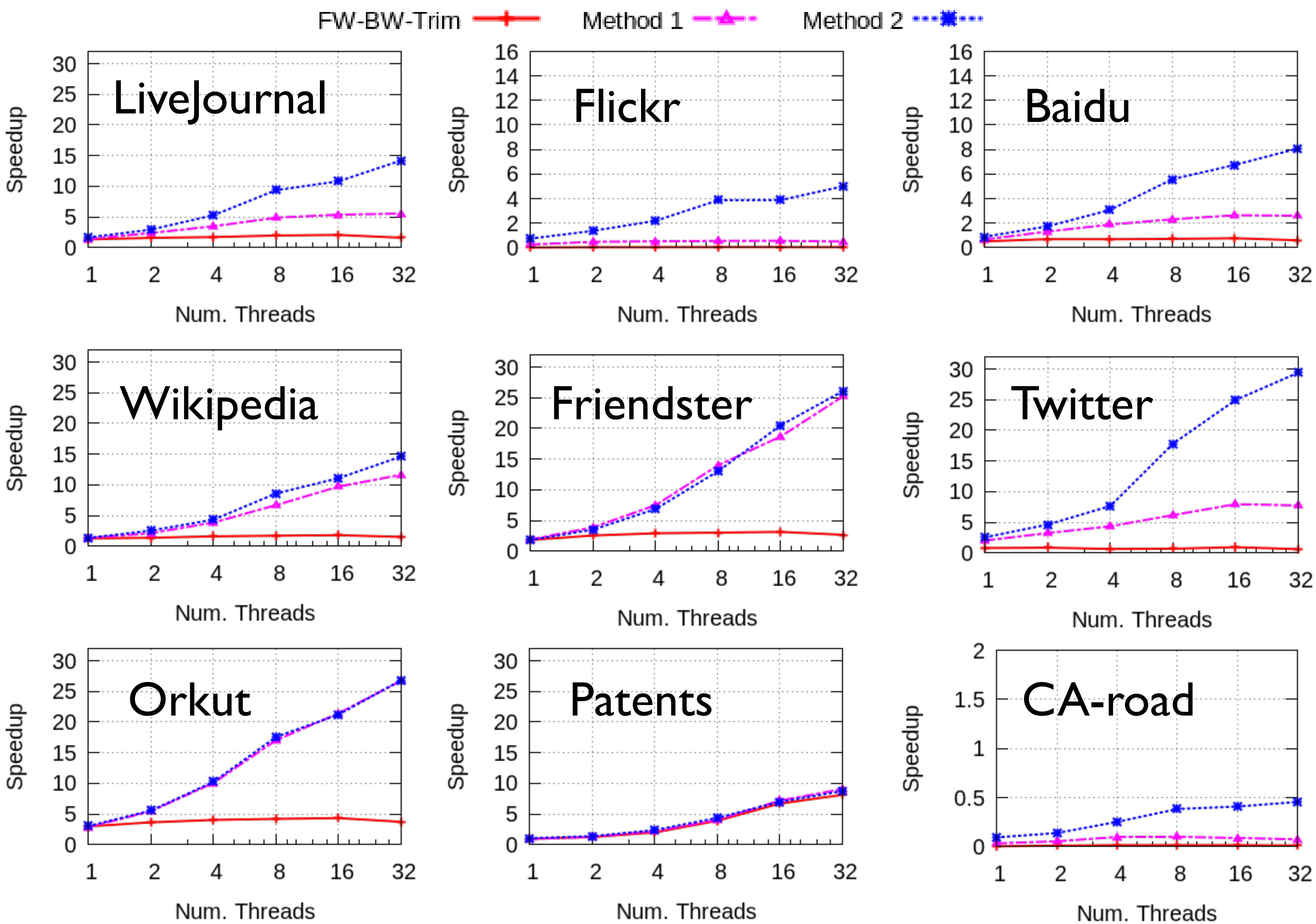
Algorithm Recap

```
FW-BW-Trim(G):
// Data parallel
Trim(G)

// Task parallel
Recur-FWBW(G)
```

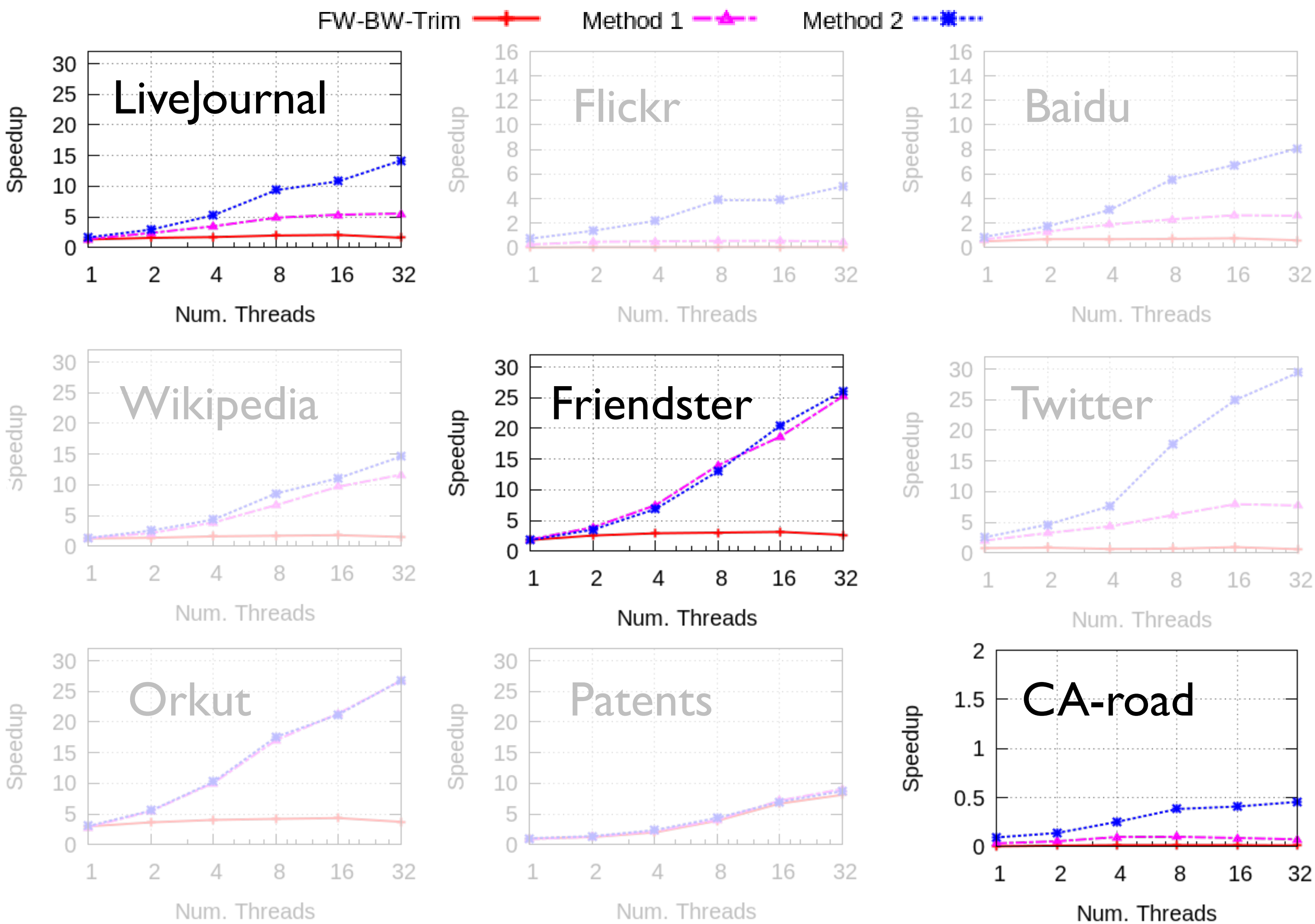
```
Method1(G):
// Data parallel
Trim(G)
Par-FWBW(G)
Trim(G)

// Task parallel
Recur-FWBW(G)
```

```
Method2(G):
// Data parallel
Trim(G)
Par-FWBW(G)
Trim’(G)
Par-WCC(G)
// Task parallel
Recur-FWBW(G)
```

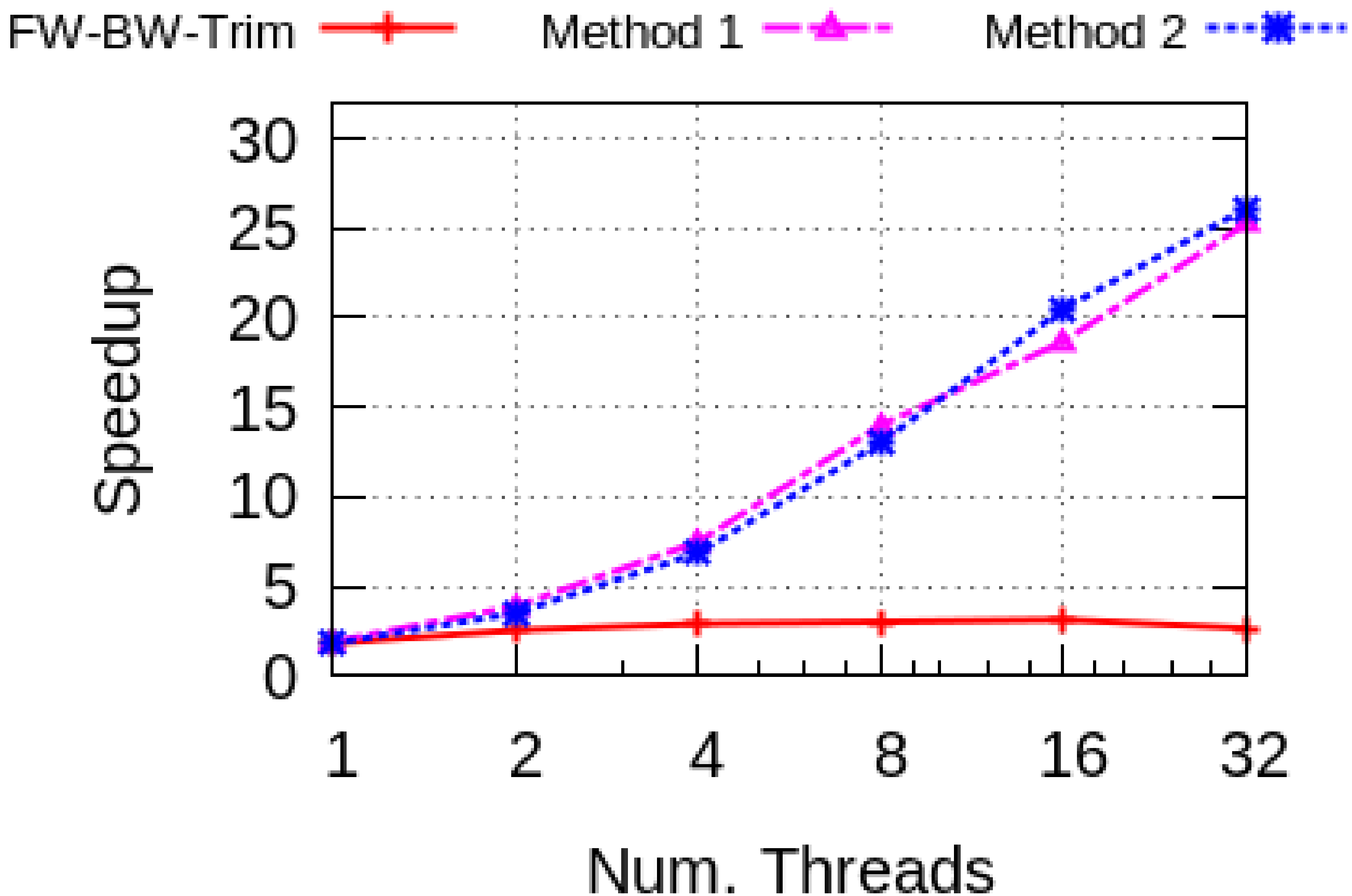
Parallel Speedup Results vs. Tarjan's Alg.



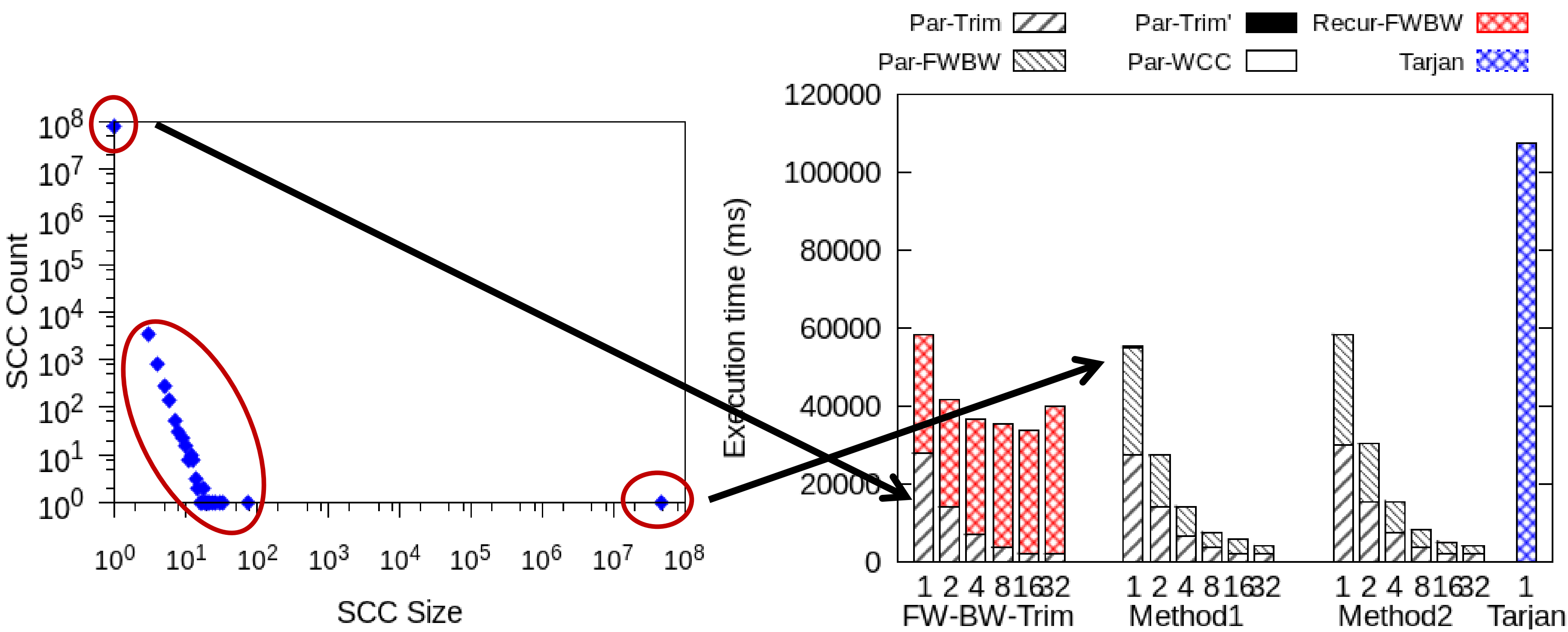
Parallel Speedup Results



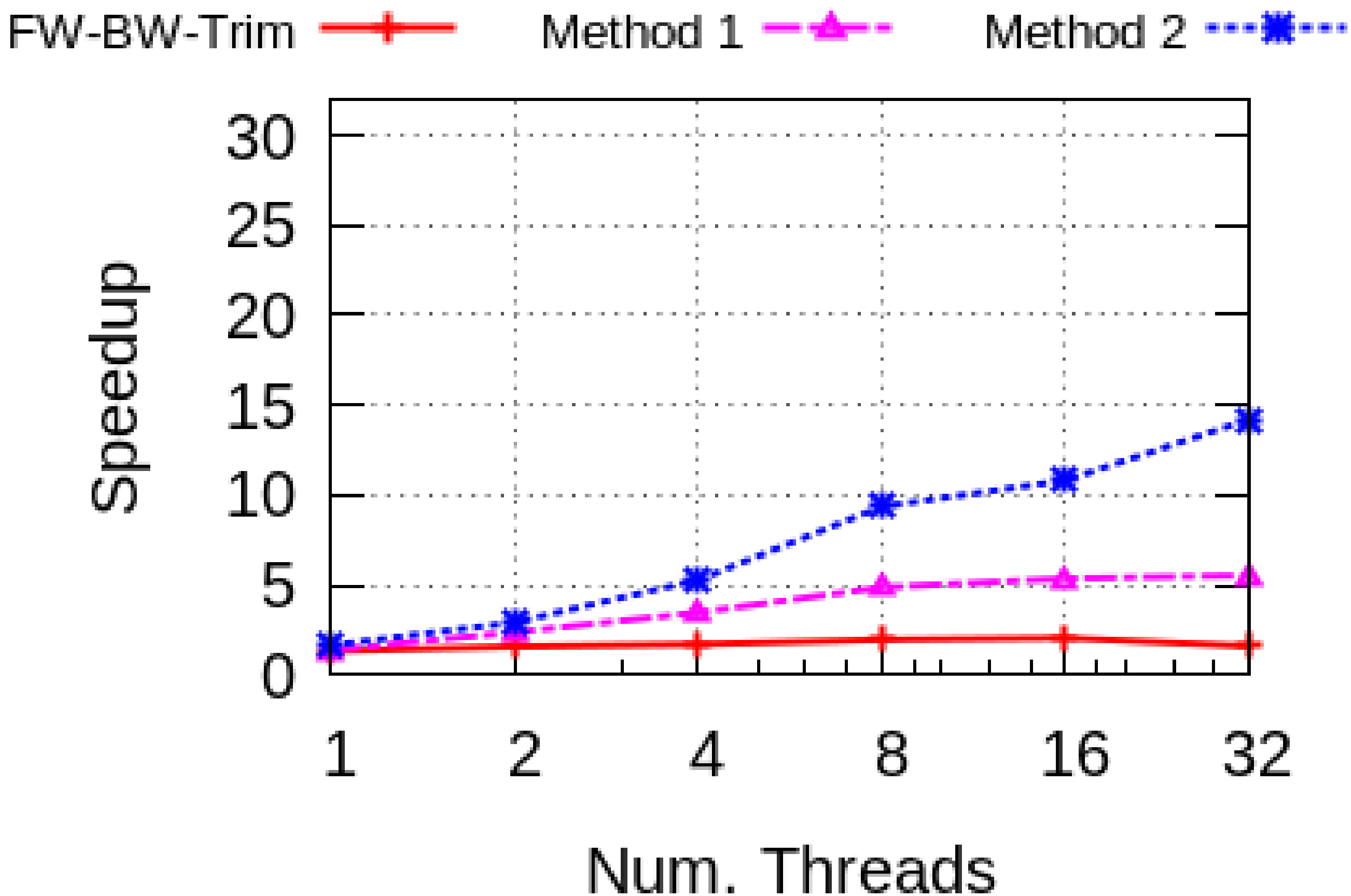
Method 2 = Method 1 Results: Friendster



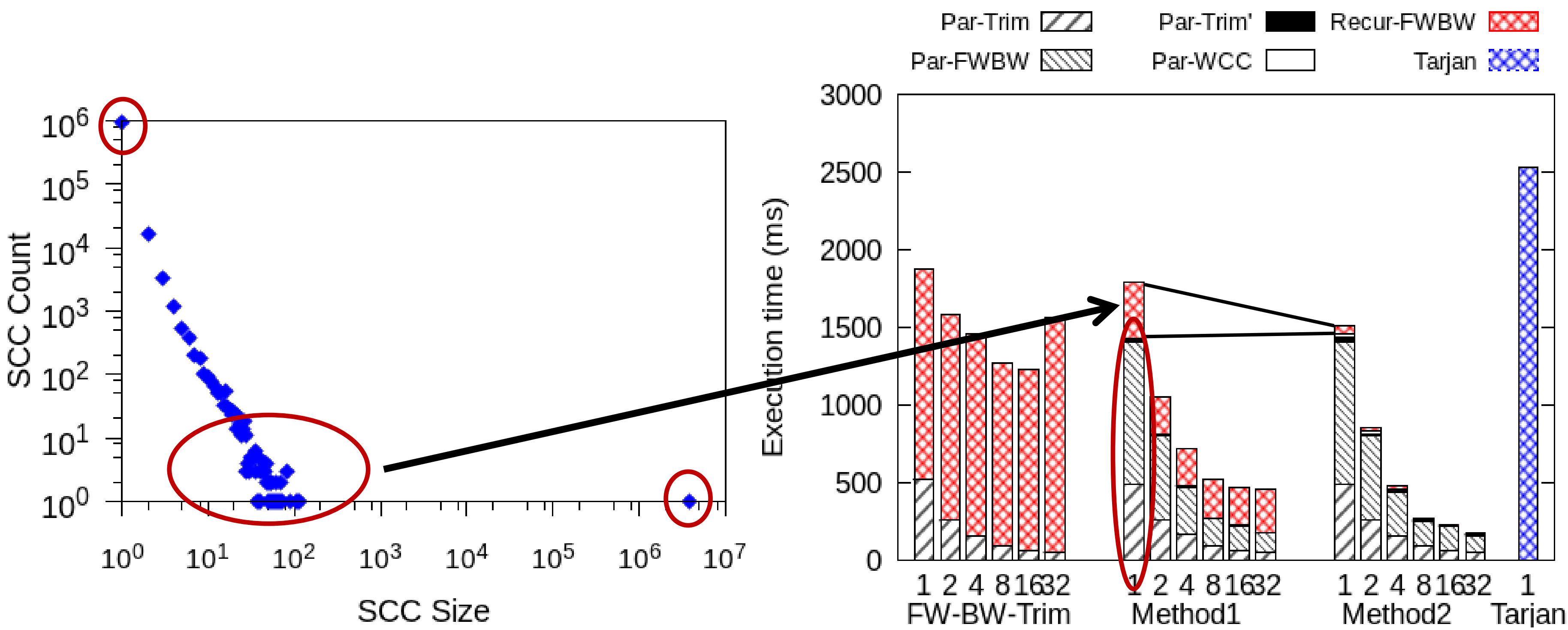
Method 2 = Method 1
Results: Friendster



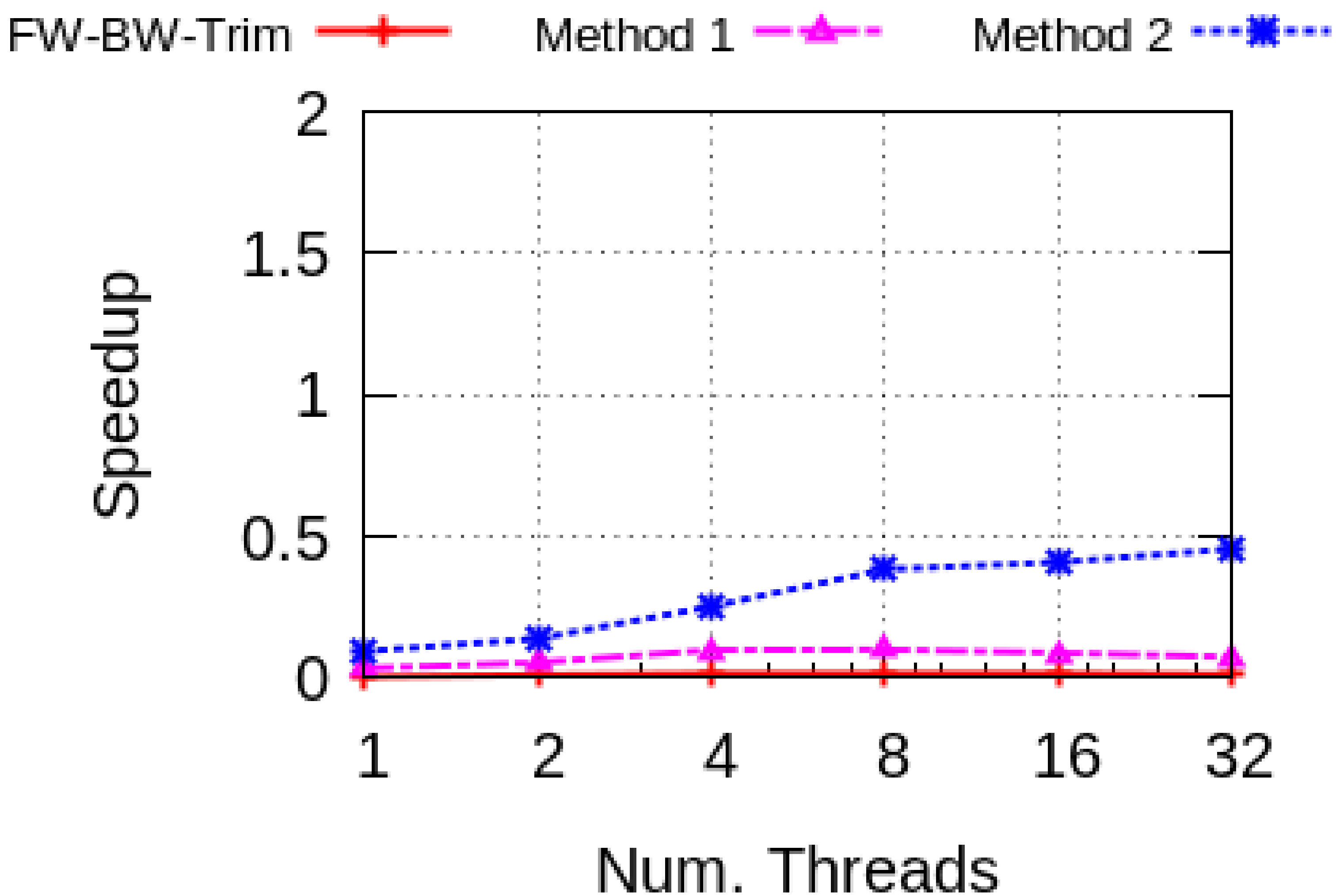
Method 2 > Method 1
Results: LiveJournal



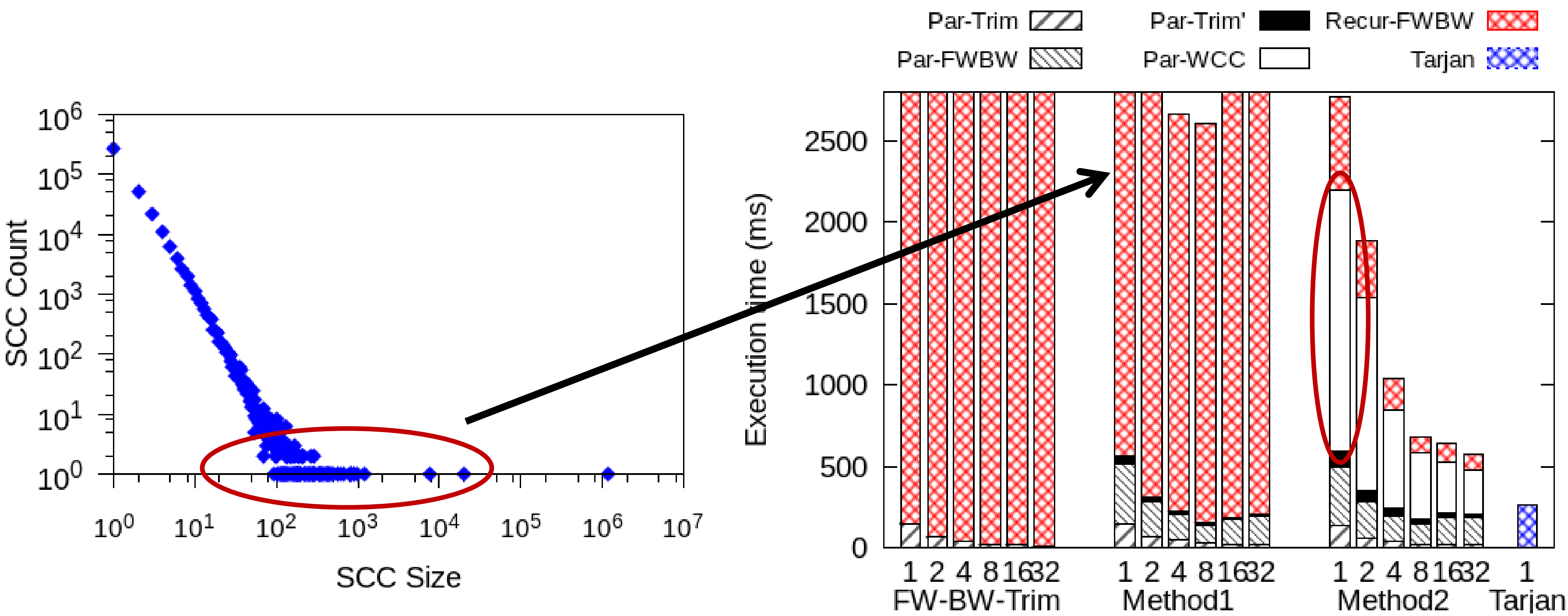
Method 2 > Method 1
Results: LiveJournal



Tarjan > Methods 1&2 Results: CA-road



Tarjan > Methods 1&2 Results: CA-road



Conclusions

- We extend the FW-BW-Trim parallel SCC detection algorithm by taking advantage of small-world graph properties
- Result: Significant parallel speedup on small-world graphs
  - Speedup from 5x to 29.4x
  - Mean speedup 14x

Questions?

Thank you

Questions: nrodia@stanford.edu

Code available from: www.stanford.edu/~nrodia