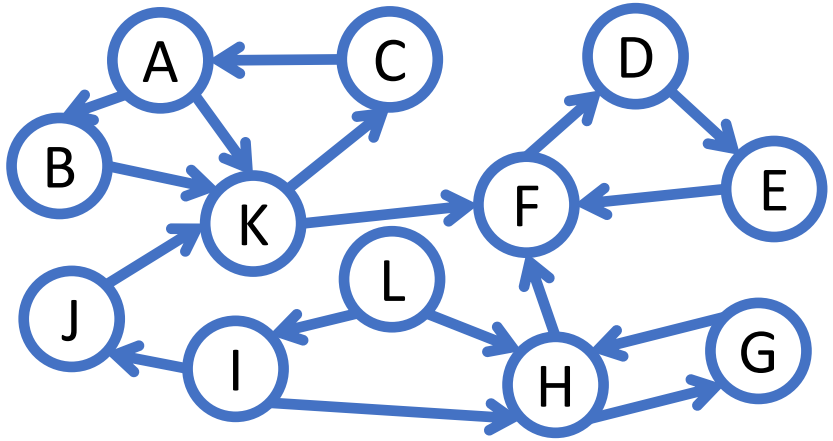
Parallel Strong Connectivity Based on Faster Reachability

Letong Wang, Xiaojun Dong, Yan Gu, Yihan Sun
University of California, Riverside

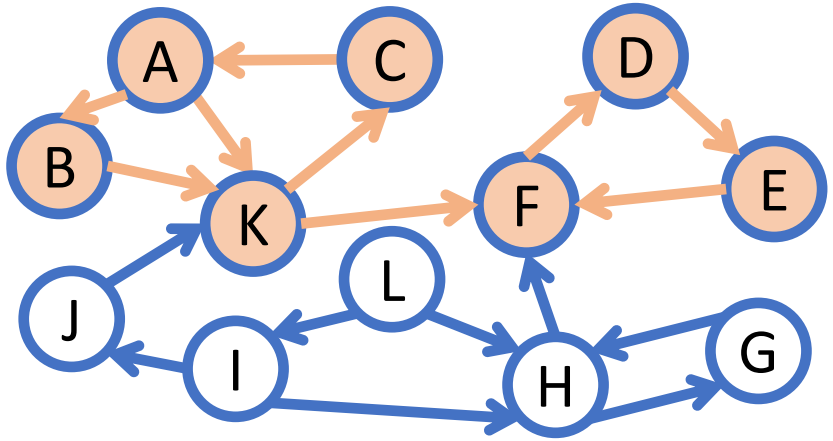
Definition: Strongly Connected Component (SCC)

- Given a directed graph $G = (V, E)$, we say
  - v is **reachable** to u it there is a path from v to u
    - **Reachability search** from v is to find all **vertices v is reachable**



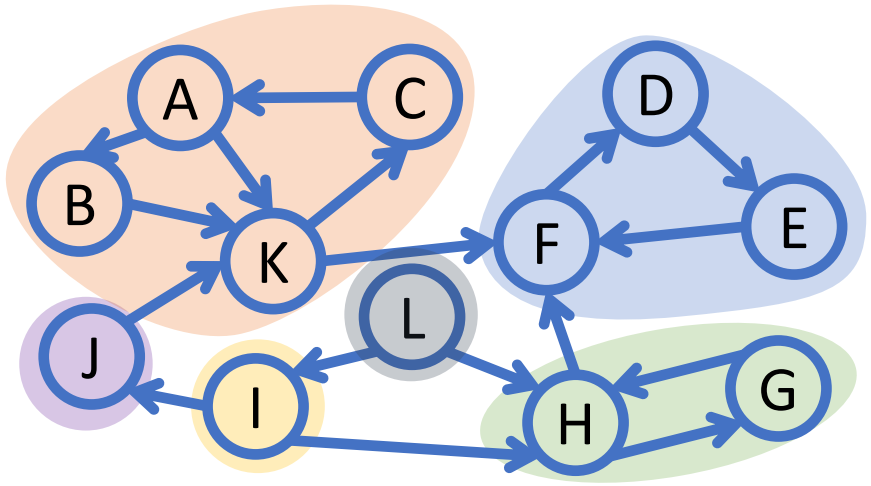
Definition: Strongly Connected Component (SCC)

- Given a directed graph $G = (V, E)$, we say
  - v is **reachable** to u it there is a path from v to u
    - **Reachability search** from v is to find all **vertices v is reachable**



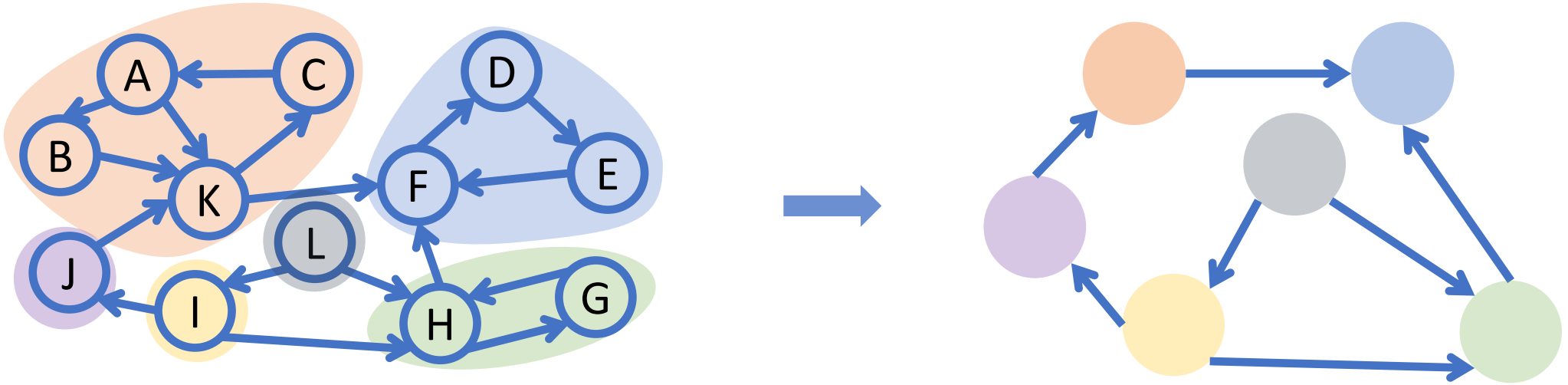
Definition: Strongly Connected Component (SCC)

- Given a directed graph $G = (V, E)$, we say
  - v is **reachable** to u it there is a path from v to u
    - **Reachability search** from v is to find all **vertices v is reachable**
  - A **Strongly Connected Component (SCC)** is a maximal subgraph that all pairs of vertices are reachable to each other
  - The SCC problem is to find all SCCs in the graph

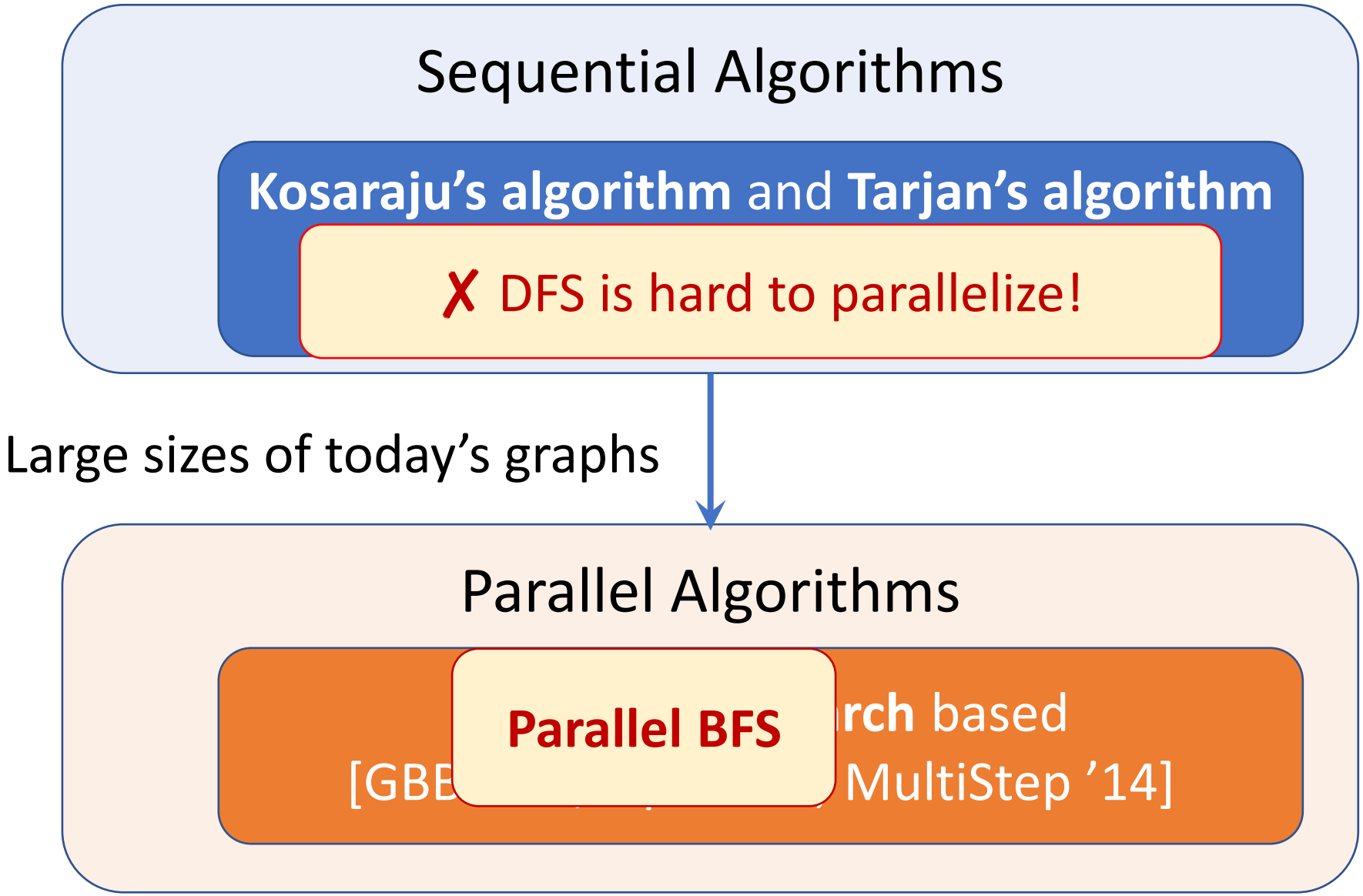


Applications

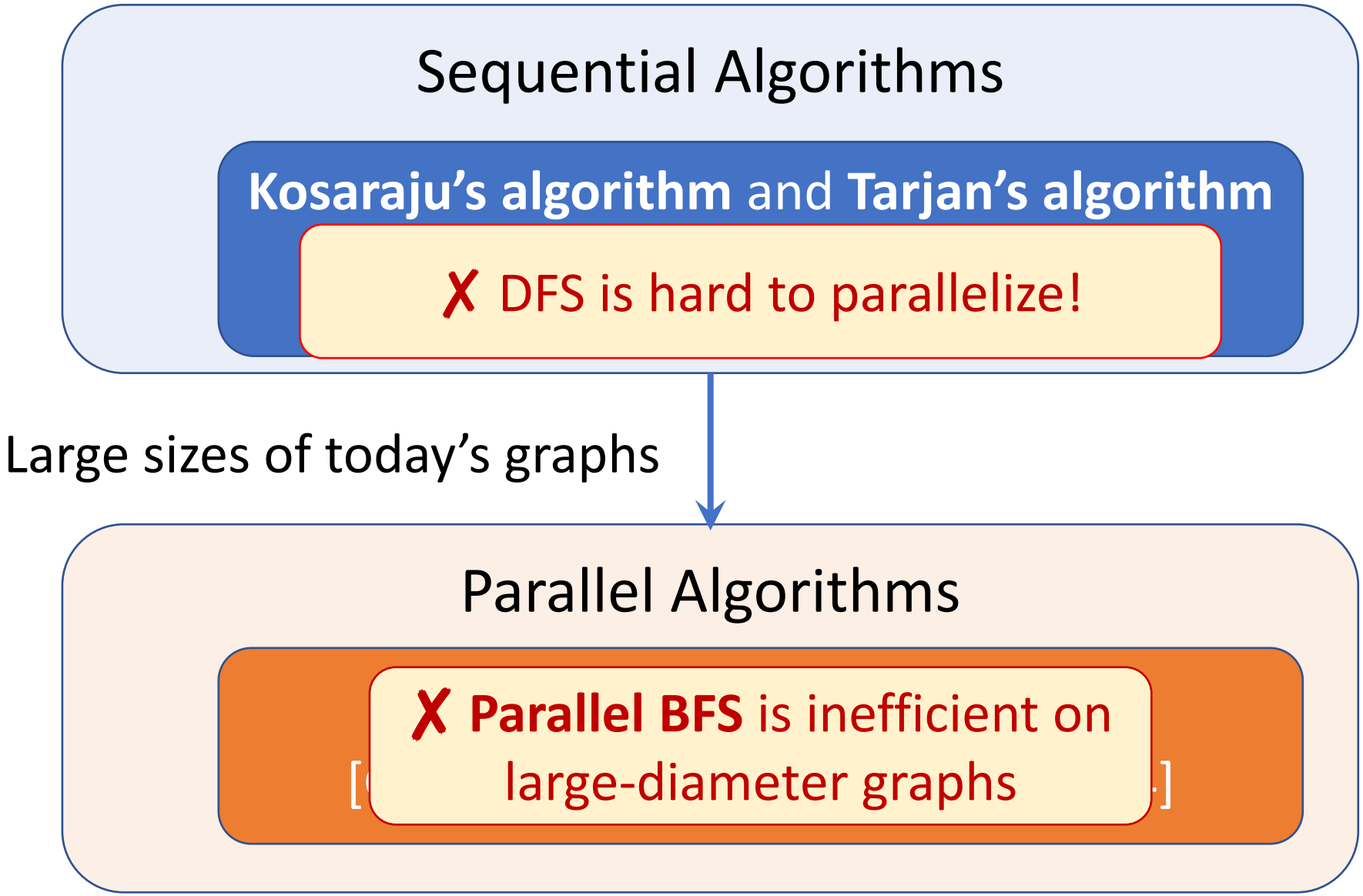
- Social network analysis
- Computational science
- Control theory
- Process a directed graph to a directed acyclic graph (DAG).



Related Work

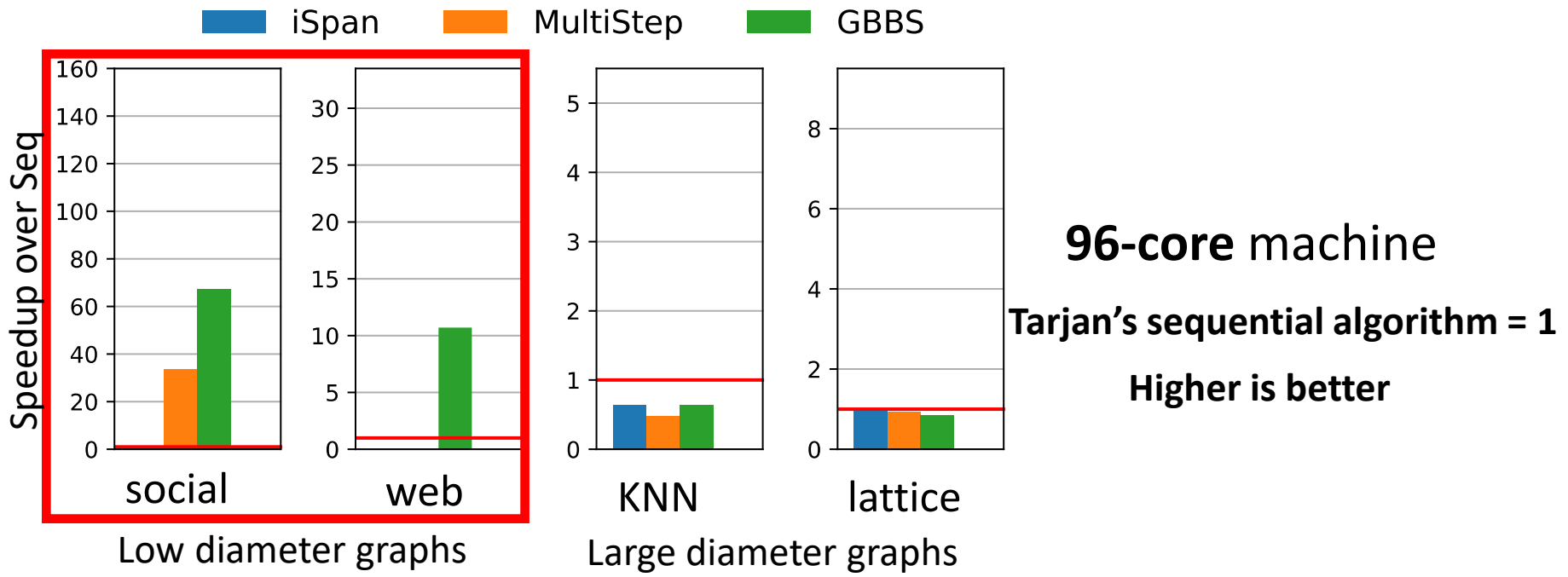


Related Work



Designing efficient parallel SCC algorithms is challenging

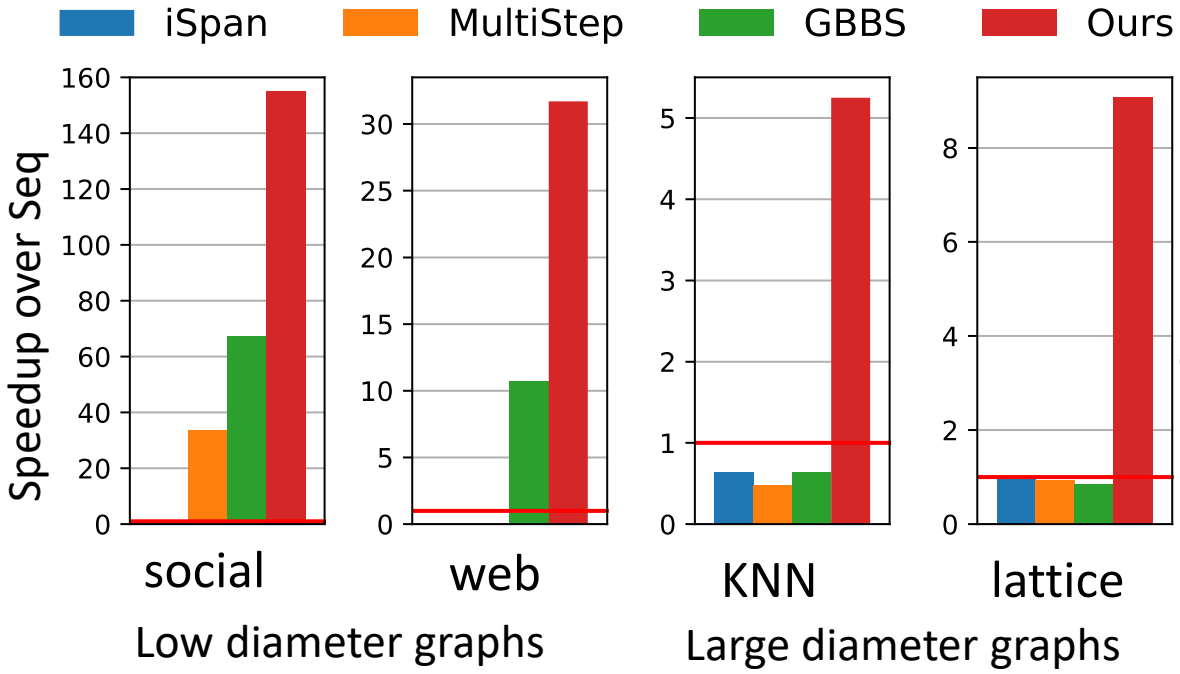
- Work well on **low-diameter graphs** (e.g. < 1000).
- On **large-diameter graphs** (e.g. > 1000) can be ***slower*** than Tarjan's sequential algorithm.



Designing efficient parallel SCC algorithms is challenging

- Work well on **low-diameter graphs** (e.g. < 1000).
- On **large-diameter graphs** (e.g. > 1000) can be ***slower*** than Tarjan's sequential algorithm.

Our algorithm is efficient on all kinds of graphs



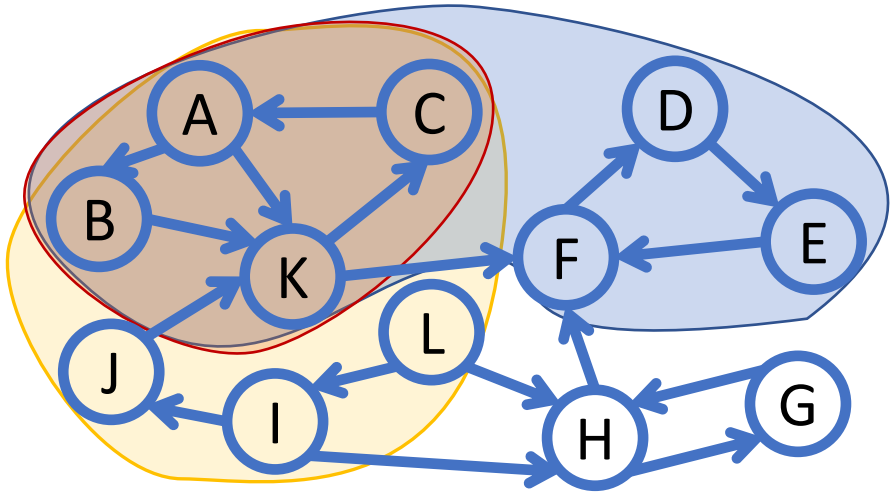
96-core machine

Tarjan's sequential algorithm = 1

Higher is better

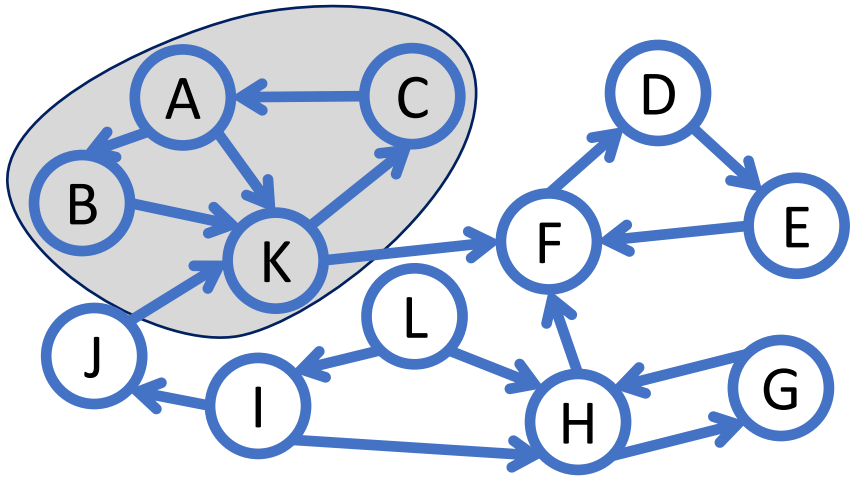
Parallel SCC Algorithms are based on Reachability Searches

- Most existing parallel algorithms are based on two facts:
  - Identifying SCCs
    - If a vertex u is both **forward and backward reachable** to a vertex v, then v is in the same SCC as u.
  - Removing cross edges
    - If two endpoints of an edge have different reachability information, then they must not be in the same SCC.



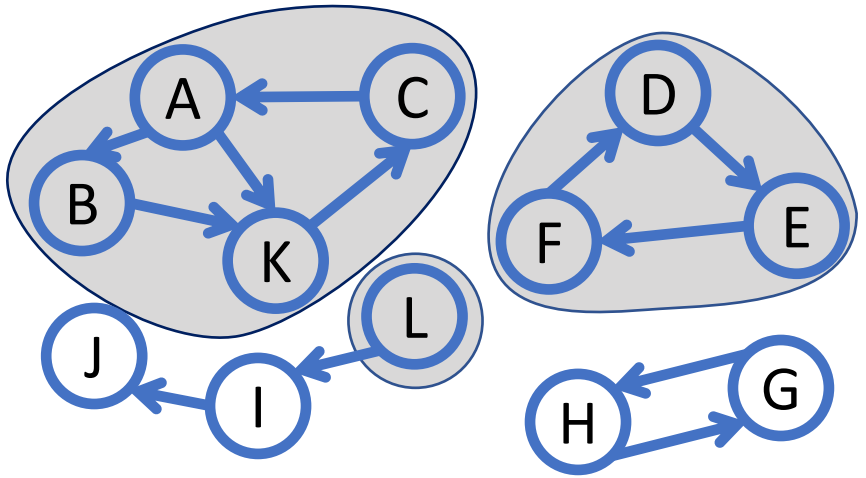
Theoretical Guaranteed Parallel SCC Algorithm: BGSS Algorithm [BGSS'20]

- It divides V into $\log n$ batches with sizes 1, 2, 4, 8,… in a **prefix-doubling** manner
- It does **forward-backward reachability** queries on each batch:
  - Use reachability info to identify SCCs



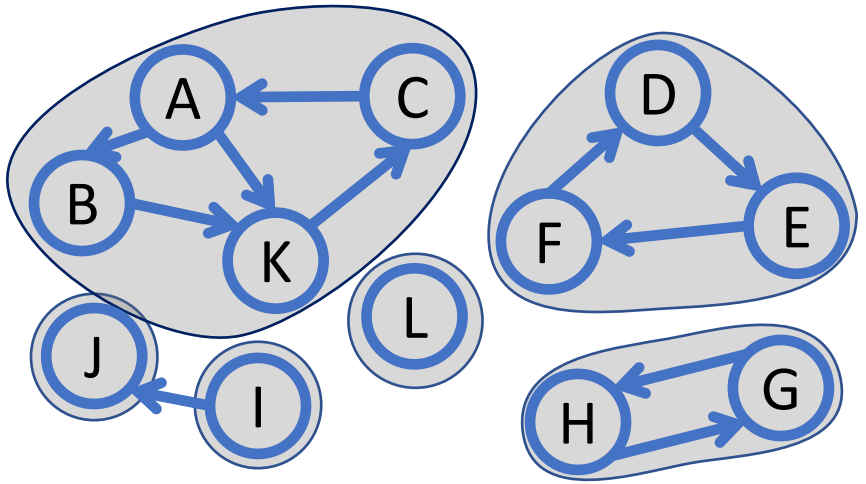
Theoretical Guaranteed Parallel SCC Algorithm: BGSS Algorithm [BGSS'20]

- It divides V into $\log n$ batches with sizes 1, 2, 4, 8,… in a **prefix-doubling** manner
- It does **forward-backward reachability** queries on each batch:
  - Use reachability info to identify SCCs



Theoretical Guaranteed Parallel SCC Algorithm: BGSS Algorithm [BGSS'20]

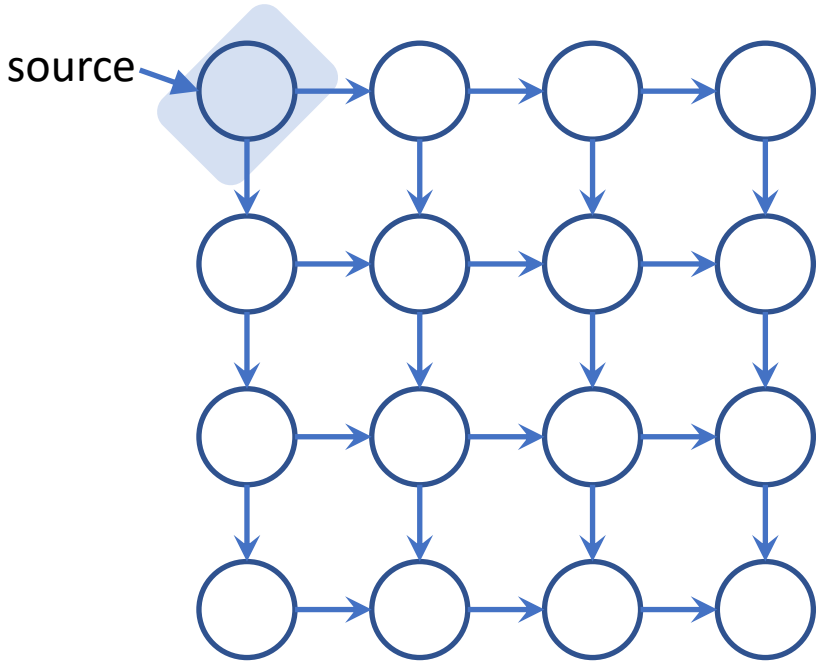
- It divides V into $\log n$ batches with sizes 1, 2, 4, 8,… in a **prefix-doubling** manner
- It does **forward-backward reachability** queries on each batch:
  - Use reachability info to identify SCCs



Existing work uses parallel BFS for reachability searches

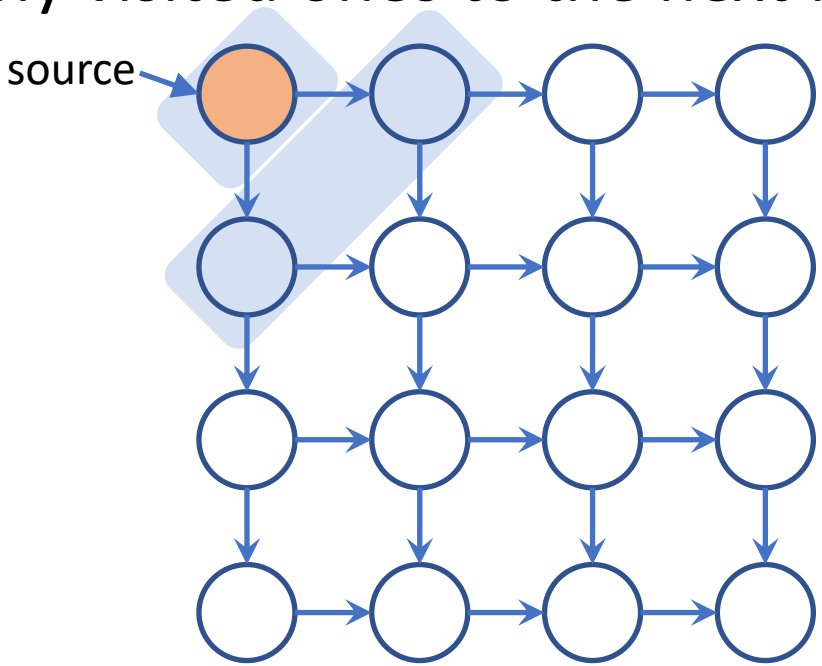
Parallel BFS is a standard approach to Implement Reachability Queries

- Given a set of sources S, it maintains a **frontier** of vertices: newly visited but not processed.



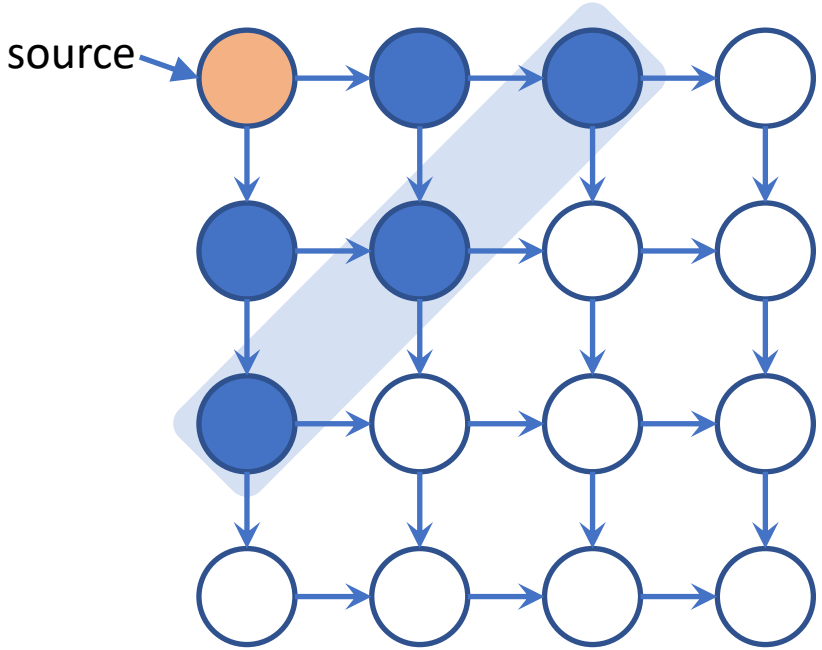
Parallel BFS is a standard approach to Implement Reachability Queries

- Given a set of sources S, it maintains a **frontier** of vertices: newly visited but not processed.
- In a round, it visits all the **neighbors** of the current frontier, and puts newly visited ones to the next frontier.



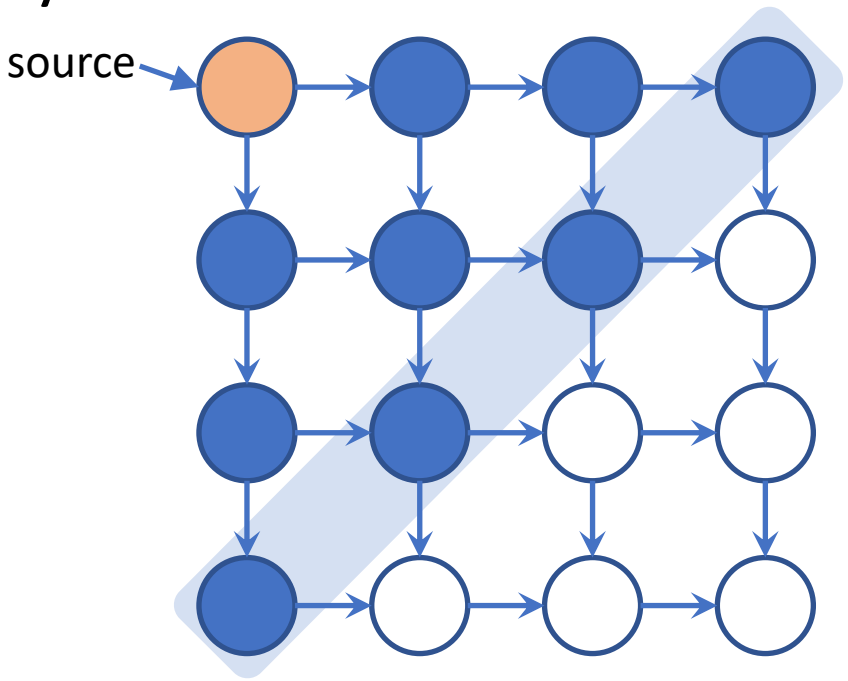
Parallel BFS is a standard approach to Implement Reachability Queries

- Given a set of sources S, it maintains a **frontier** of vertices: newly visited but not processed.
- In a round, it visits all the **neighbors** of the current frontier, and puts newly visited ones to the next frontier.



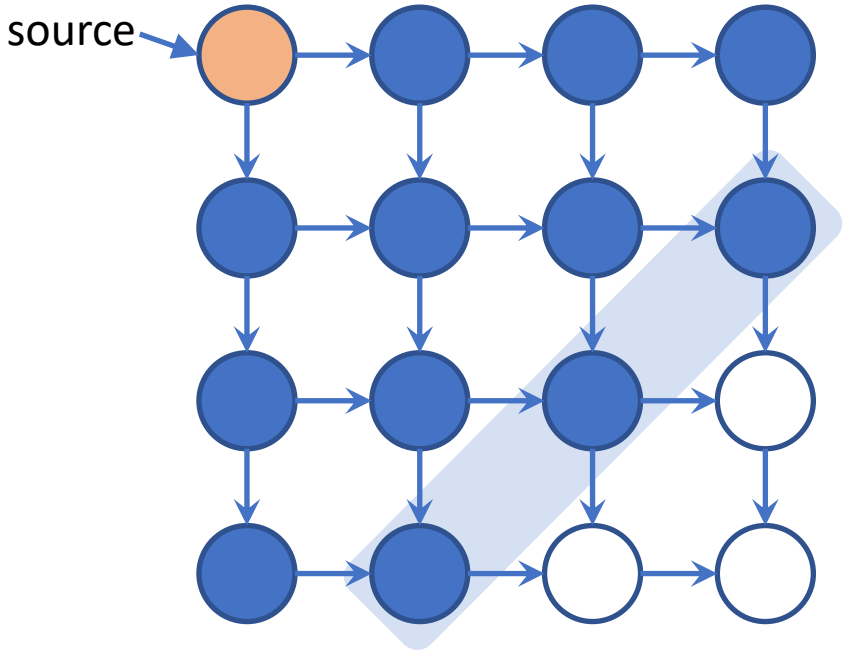
Parallel BFS is a standard approach to Implement Reachability Queries

- Given a set of sources S, it maintains a **frontier** of vertices: newly visited but not processed.
- In a round, it visits all the **neighbors** of the current frontier, and puts newly visited ones to the next frontier.



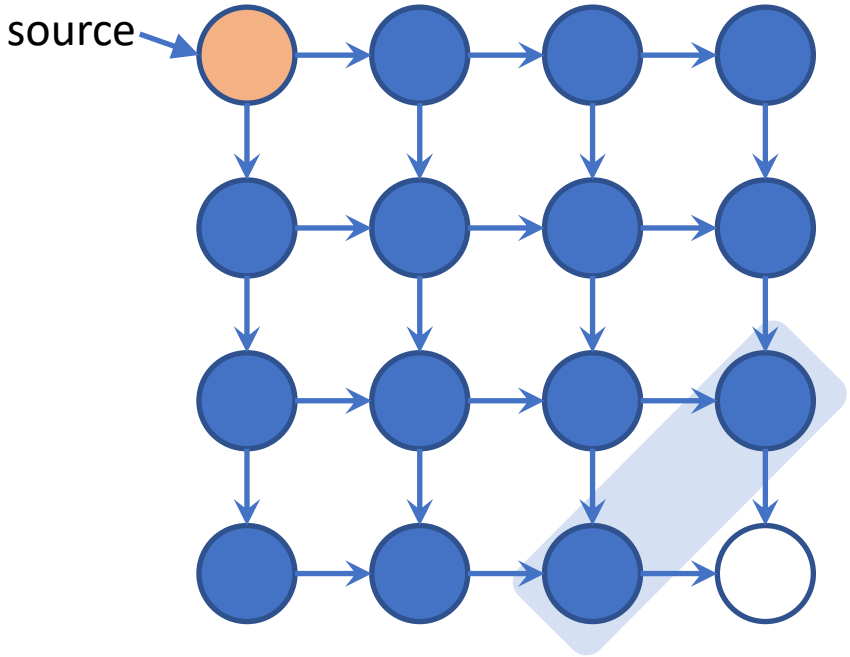
Parallel BFS is a standard approach to Implement Reachability Queries

- Given a set of sources S, it maintains a **frontier** of vertices: newly visited but not processed.
- In a round, it visits all the **neighbors** of the current frontier, and puts newly visited ones to the next frontier.



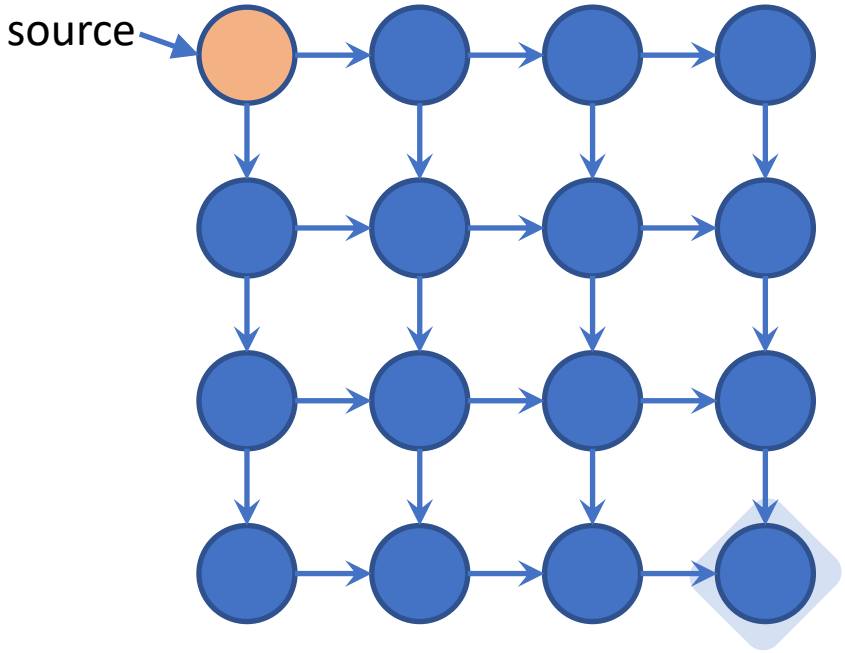
Parallel BFS is a standard approach to Implement Reachability Queries

- Given a set of sources S, it maintains a **frontier** of vertices: newly visited but not processed.
- In a round, it visits all the **neighbors** of the current frontier, and puts newly visited ones to the next frontier.



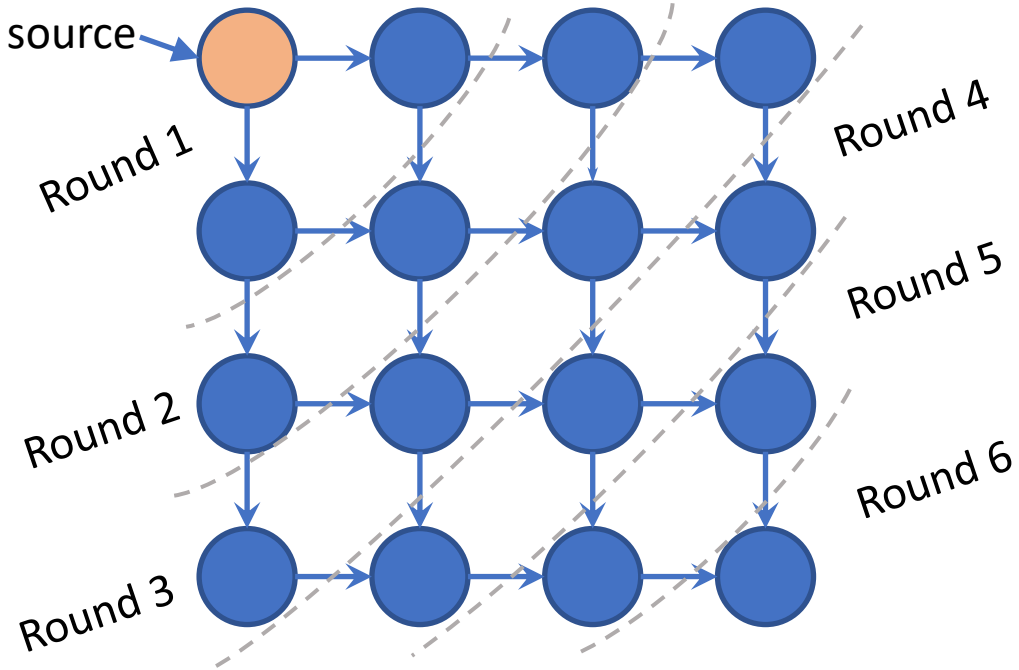
Parallel BFS is a standard approach to Implement Reachability Queries

- Given a set of sources S, it maintains a **frontier** of vertices: newly visited but not processed.
- In a round, it visits all the **neighbors** of the current frontier, and puts newly visited ones to the next frontier.



Parallel BFS is a standard approach to Implement Reachability Queries

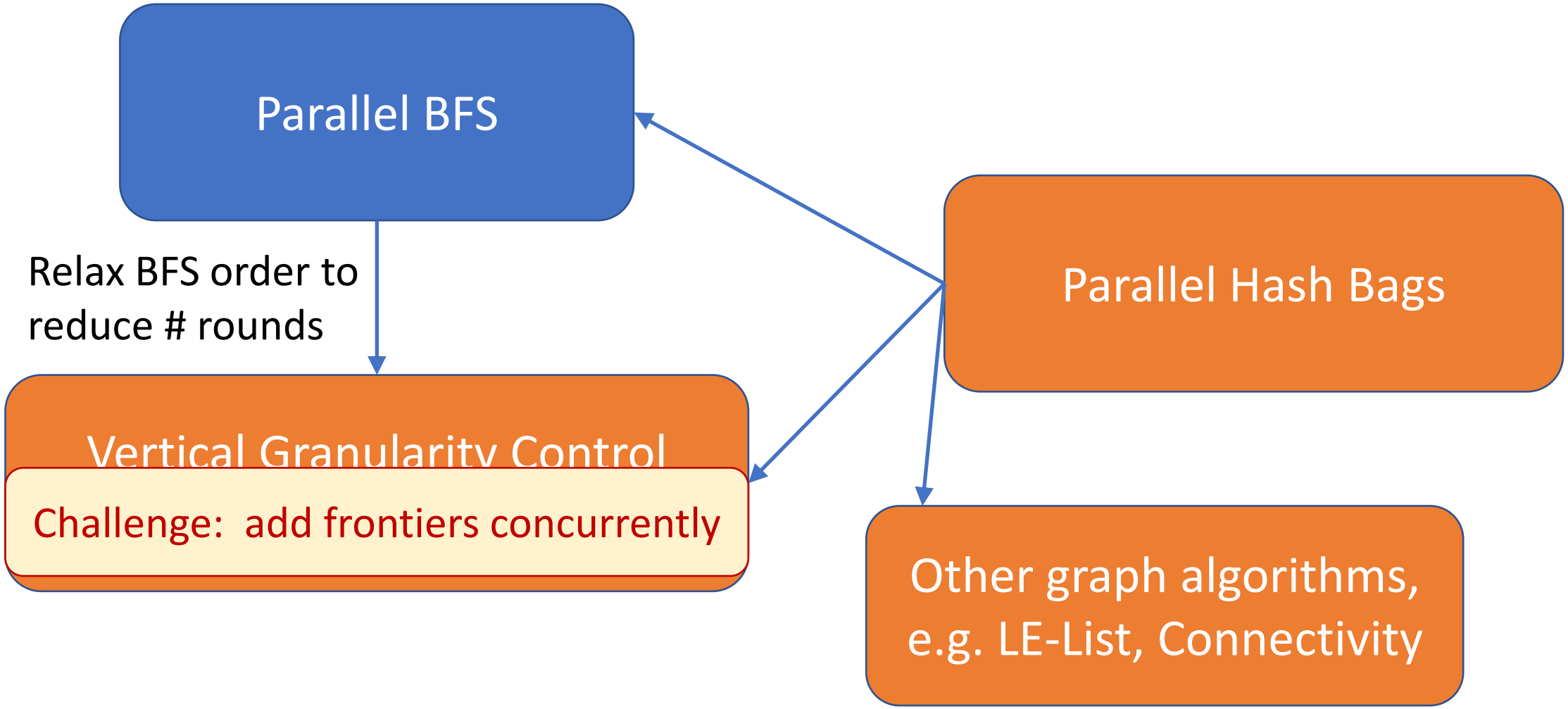
- Given a set of sources S, it maintains a **frontier** of vertices: newly visited but not processed.
- In a round, it visits all the **neighbors** of the current frontier, and puts newly visited ones to the next frontier.



- The # rounds is $O(D)$, D is the graph diameter
- Returns BFS order as a by-product
- Don't need BFS order in reachability query

Can we **relax BFS order** to design a more efficient reachability search algorithm?

Overview of Our Techniques



Overview of Our Techniques

Vertical Granularity Control (VGC)

Parallel Hash Bags

Technique 1: Vertical Granularity Control (VGC)

Sparse graphs with a **large diameter**
- Frontier size
- Average degree

} small

many rounds

Many global synchronizations

Small task size

Threads arrangement cost > actual computation

Granularity Control:
- Group computation to avoid the overhead caused by generating unnecessary parallel tasks.

Horizontal Granularity Control

Group computation in the same level.

✗ graphs are more irregular and may cause load balance problem

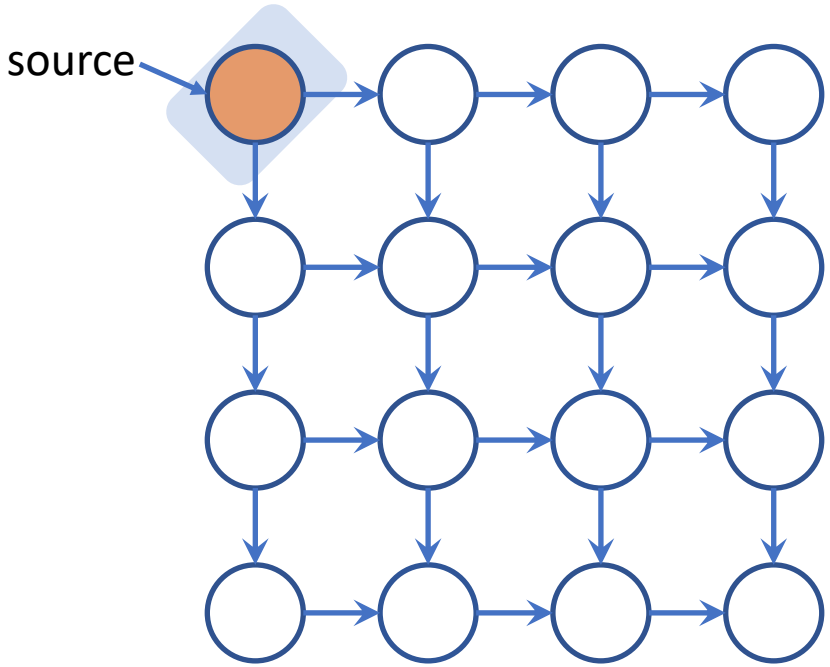
Vertical Granularity Control

Group computation across different levels.

✓ reduce number of rounds

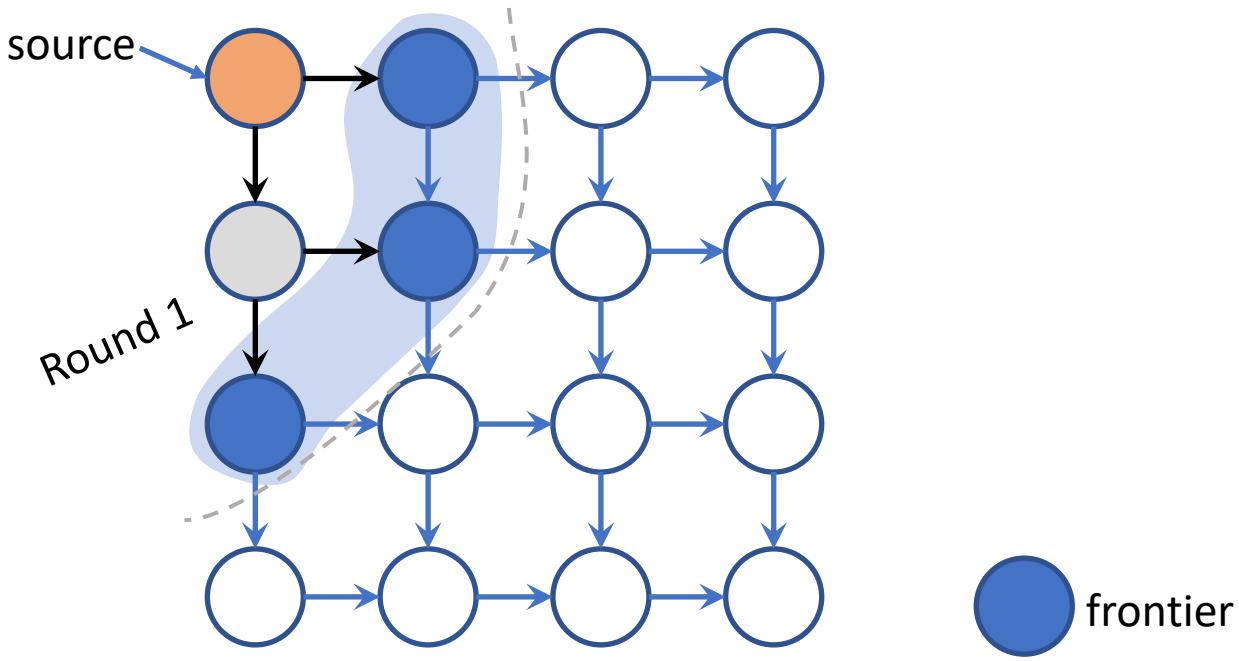
Technique 1: Vertical Granularity Control (VGC)

- Let each frontier do a local search to explore enough edges.
- Carefully control the size of each parallel tasks to be similar.



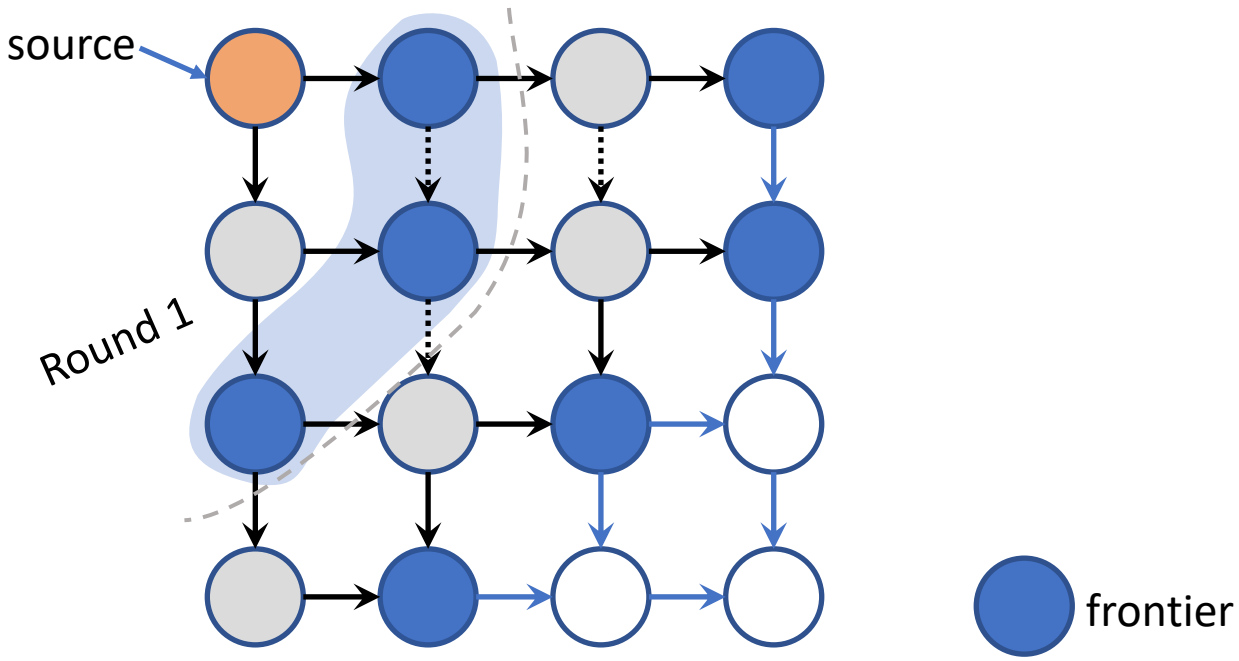
Technique 1: Vertical Granularity Control (VGC)

- Let each frontier do a local search to explore enough edges.
- Carefully control the size of each parallel tasks to be similar.



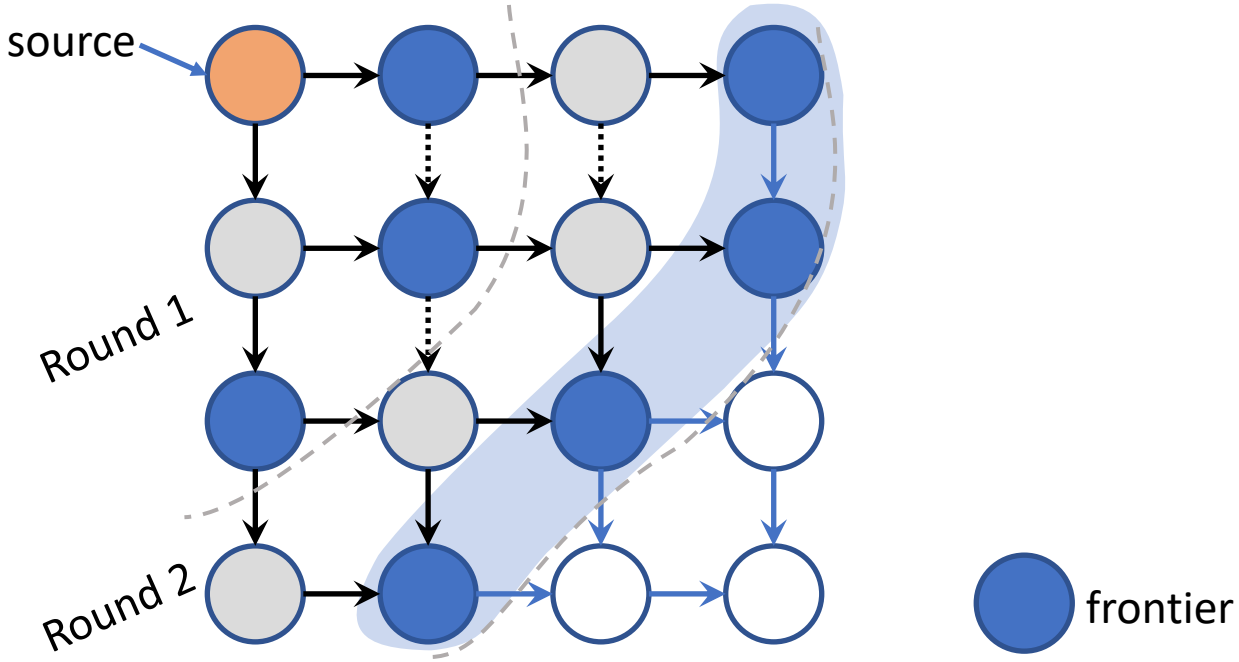
Technique 1: Vertical Granularity Control (VGC)

- Let each frontier do a local search to explore enough edges.
- Carefully control the size of each parallel tasks to be similar.



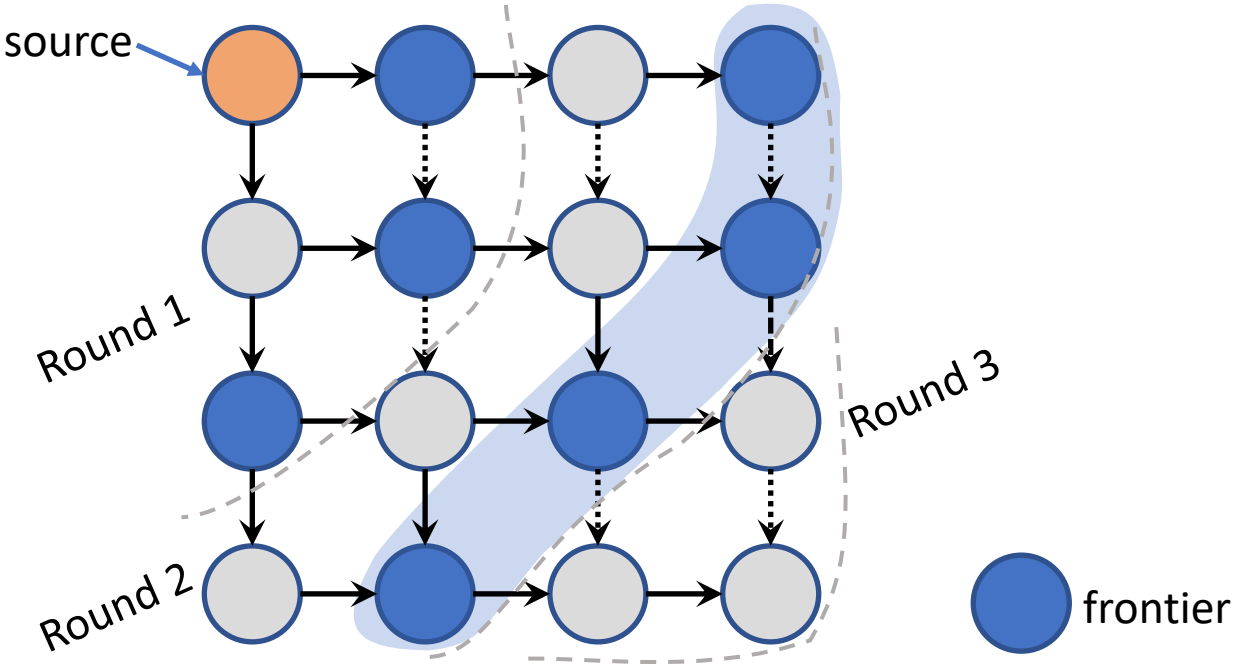
Technique 1: Vertical Granularity Control (VGC)

- Let each frontier do a local search to explore enough edges.
- Carefully control the size of each parallel tasks to be similar.



Technique 1: Vertical Granularity Control (VGC)

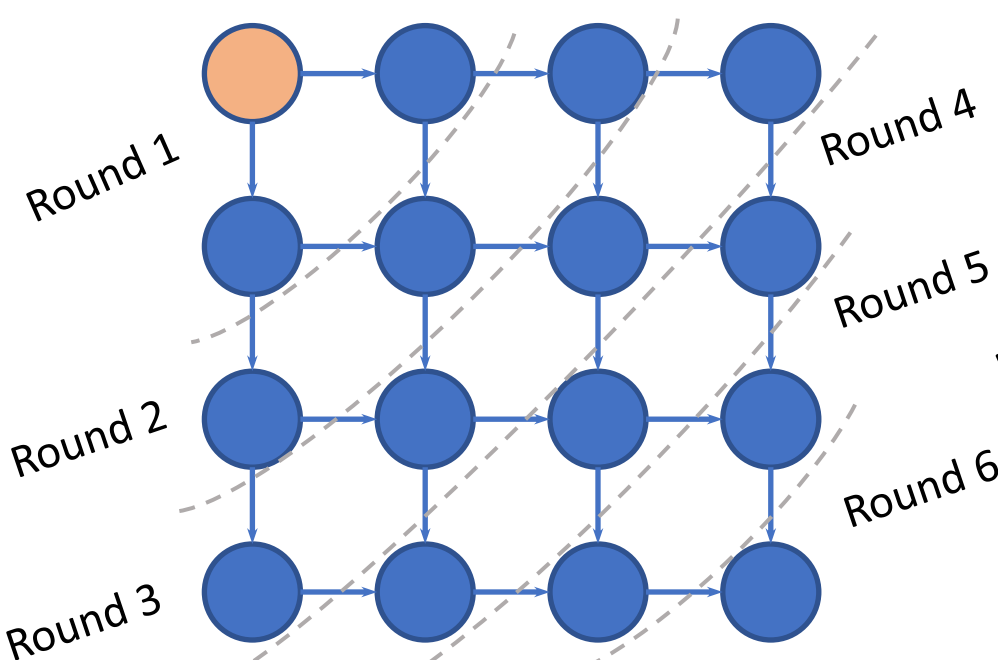
- Let each frontier do a local search to explore enough edges.
- Carefully control the size of each parallel tasks to be similar.

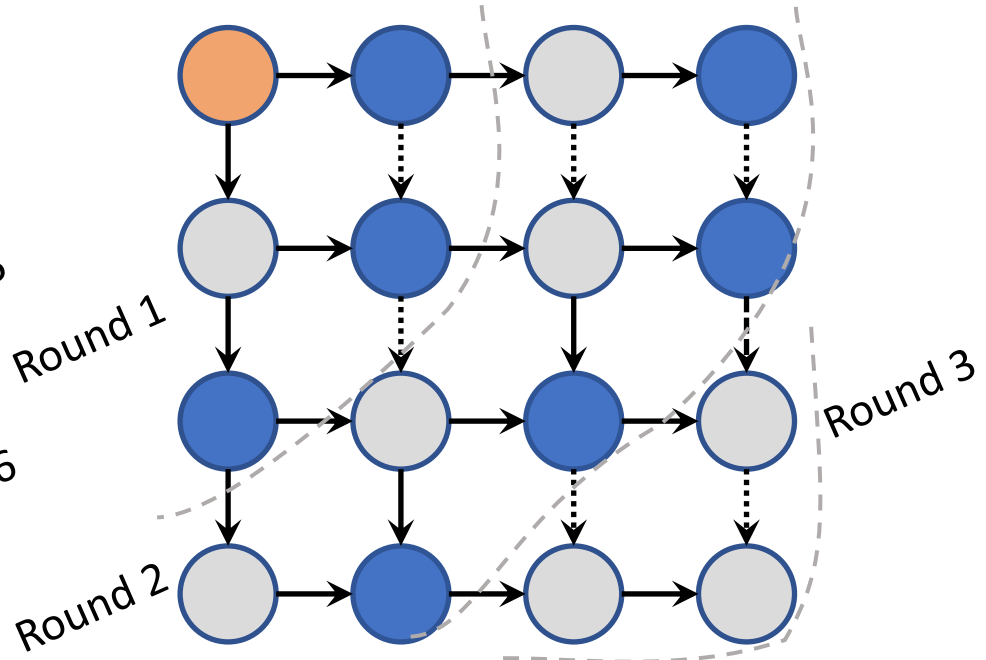


Technique 1: Vertical Granularity Control (VGC)

- Let each frontier do a local search to explore enough edges.
- Carefully control the size of each parallel tasks to be similar.

Fewer rounds





Challenge of Maintain Frontiers

- Parallel BFS uses **synchronized** “edge-revisit” scheme to add frontiers.
- VGC generates frontiers on the fly and need to **concurrently** add them.

difficult to be both correct and efficient

Overview of Our Techniques

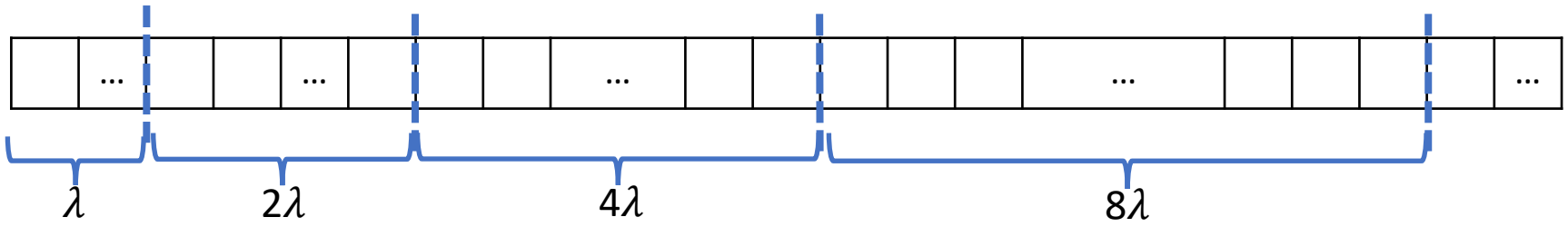
Vertical Granularity Control (VGC)

Parallel Hash Bags

- **Properties required:**
  - **Concurrent insertion** with $O(1)$ cost in expectation.
  - **Packing** frontiers together with $O(s)$ cost (s is the frontier size).

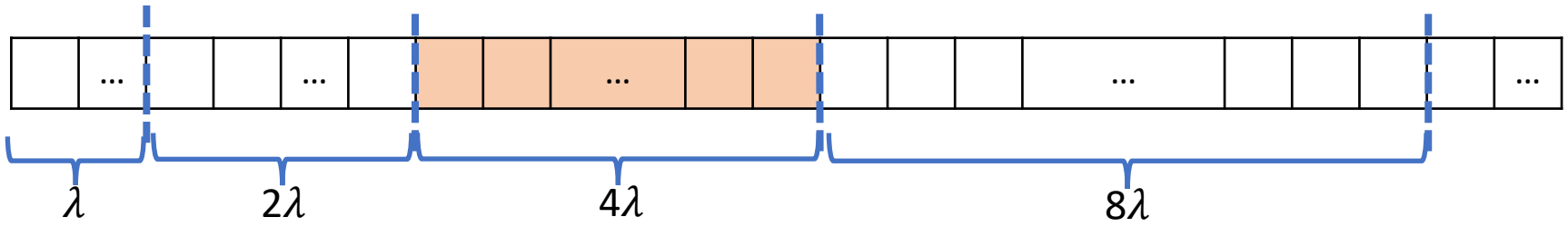
Techniques 2: Parallel Hash Bag

- **Hierarchical chunks**
  - Dividing a pre-allocated array into **chunks** with exponentially-increasing sizes.



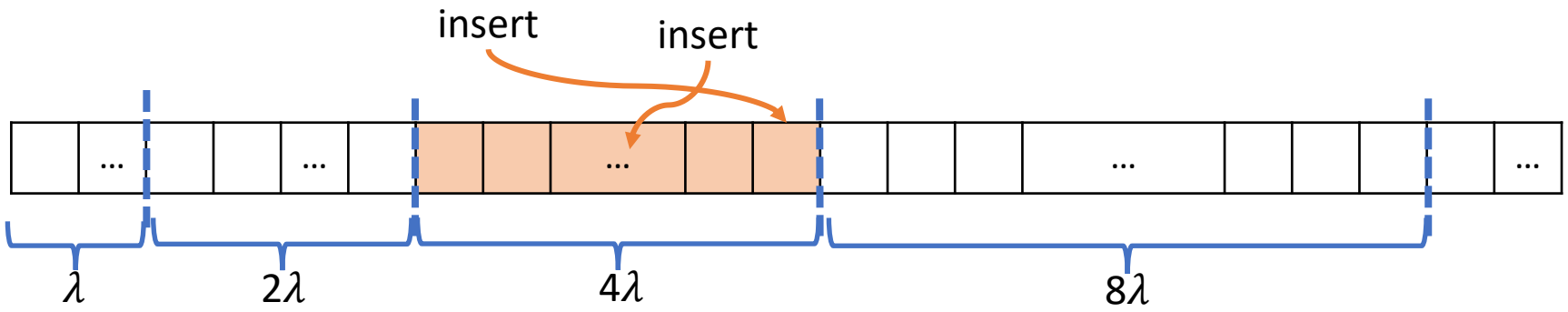
Techniques 2: Parallel Hash Bag

- **Hierarchical chunks**
  - Dividing a pre-allocated array into **chunks** with exponentially-increasing sizes.



Techniques 2: Parallel Hash Bag

- **Hierarchical chunks**
  - Dividing a pre-allocated array into **chunks** with exponentially-increasing sizes.
  - Insertion: hash a random position in the active chunk.
  - Resizing: move to use the next chunk.



Techniques 2: Parallel Hash Bag

- **Hierarchical chunks**
  - Dividing a pre-allocated array into **chunks** with exponentially-increasing sizes.
  - Insertion: hash a random position in the active chunk.
  - Resizing: move to use the next chunk.

How to know when to resize?

insert insert

Sampling scheme:

- Estimate the number of insertions in a chunk by sampling.

✓ cost efficient ✓ has theoretical guarantee

Our techniques can be applied to other algorithms

Vertical Granularity Control (VGC)

- Directly apply to reachability-based algorithms
- Need additional design if the order matters (e.g., SSSP [DGSZ'21])

Parallel Hash Bags

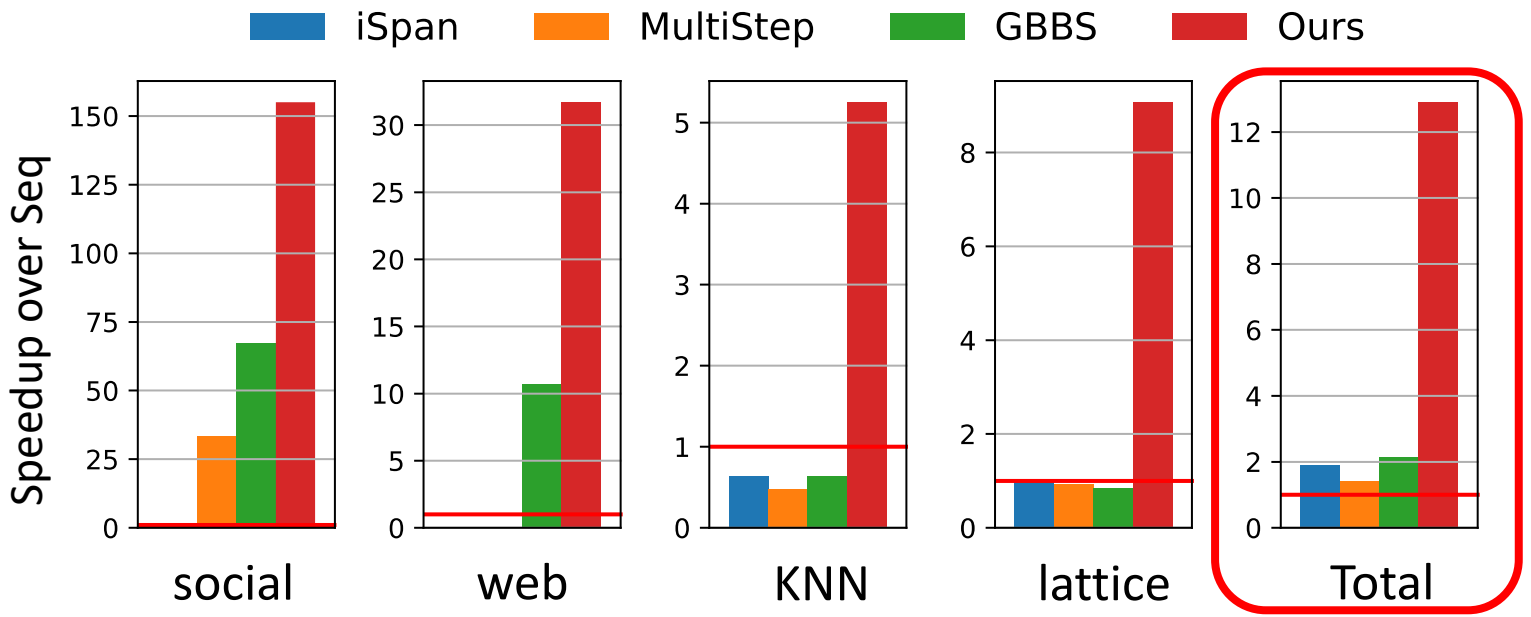
- Directly applied to algorithms that need to maintain nondeterministic elements (frontiers)
- We also applied it to Connectivity and LE-List algorithms in our paper.

Experiment Results

- Testing graphs (18 in total)
  - Social networks (2), Web graphs (4), K-NN graphs(8), Lattice graphs(4)
- Machine:
  - 96 core (192 hyperthreads)
  - 1.5TB memory
- Baselines:
  - GBBS [DSTBS'20], iSpan [JLH'18], Multi-step [SRM'18]
- Average=geometric mean

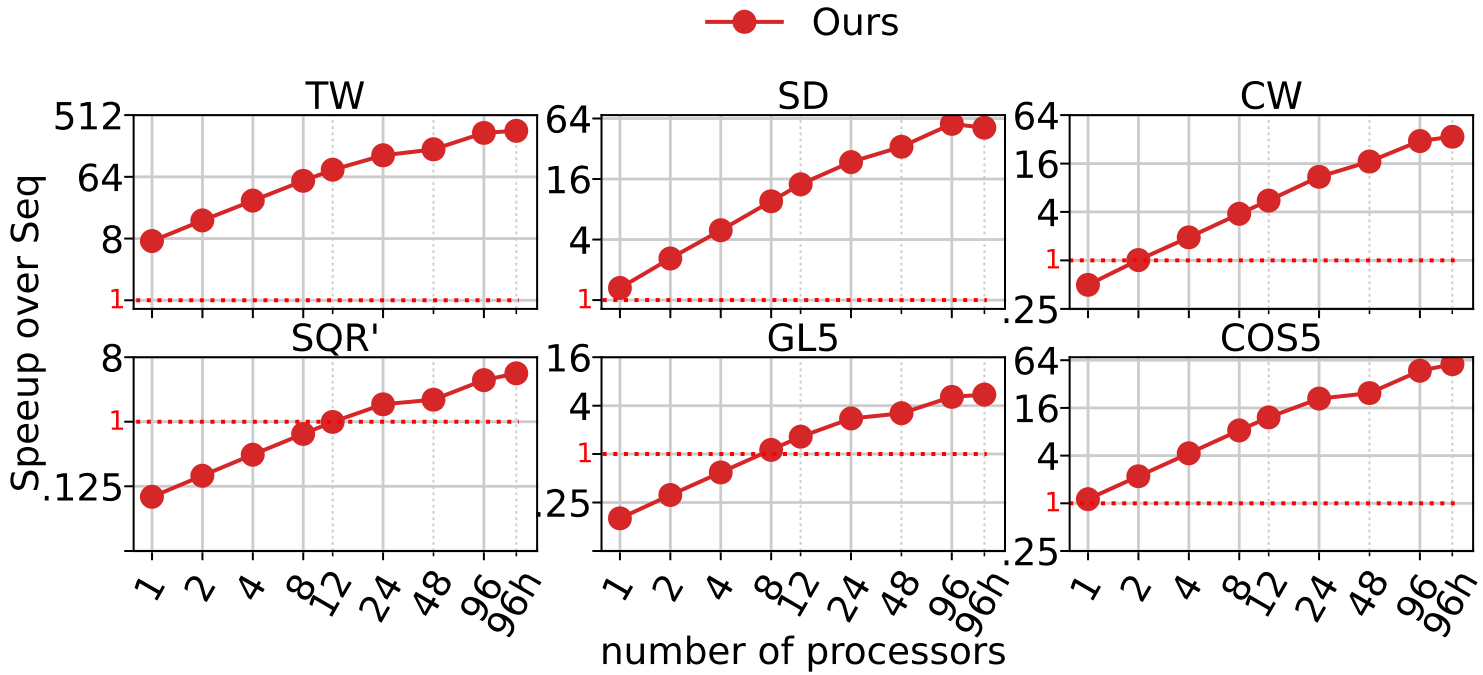
Overall Performance

- Our algorithm is **faster** than existing algorithms on all kinds of graphs.
- On average, our algorithm is **6x** faster than the best of other implementations.



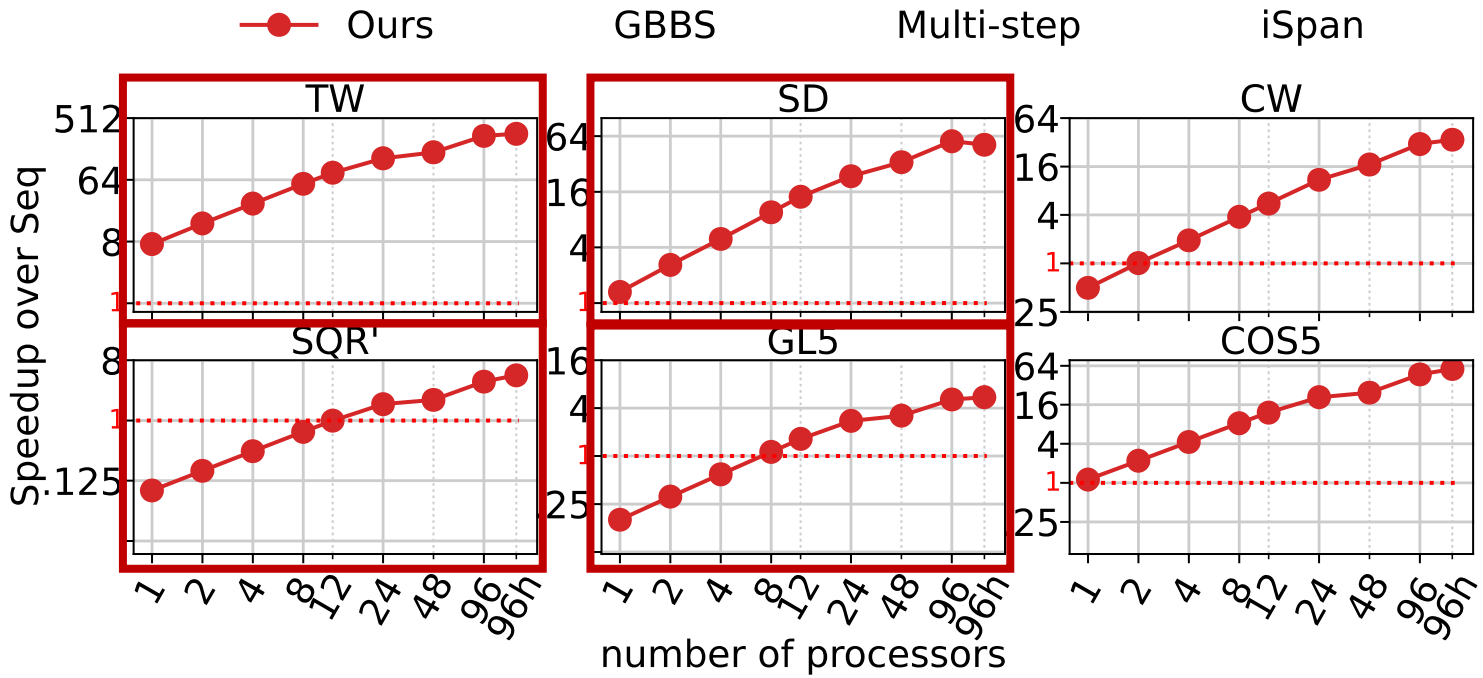
Scalability

- Our algorithm achieves an almost linear speedup on assorted graphs.



Scalability

- Our algorithm achieves an almost linear speedup on assorted graphs.
- When the number of threads is more than 24, the speedups of other existing parallel algorithms stop increasing or even decrease on some graphs (TW, SD, GL5, SQR’).



The effect of VGC and hash bags

- Evaluate hash bags and VGC
  - **Hash bags** bring **2x** speedup on average
  - **VGC** brings:
    - Up to **1.3x** speedup on low-diameter graphs
    - **1.7x**-**15x** speedup on large-diameter graphs
- See figures in the paper

Summary

- We observe that existing parallel SCC algorithms lack parallelism on large-diameter graphs
  - can be even slower than sequential algorithm on a 96-core machine
- We proposed two techniques to improve the BGSS SCC algorithm
  - Vertical Granularity Control and Parallel Hash Bags
    - Implemented a faster reachability search algorithm
  - The two techniques can be applied to other algorithms, Connectivity, LE-List, …
- Experiment Results:
  - Faster than the best of existing parallel implementation on all kinds of graphs
  - Our algorithm has an almost linear speedup on all the graphs.