Parallel Strong Connectivity Based on Faster Reachability

Letong Wang Xiaojun Dong Yan Gu Yihan Sun

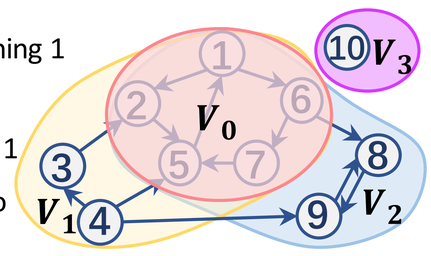
University of California, Riverside

Motivation

Given a directed graph $G = (V, E)$, we say

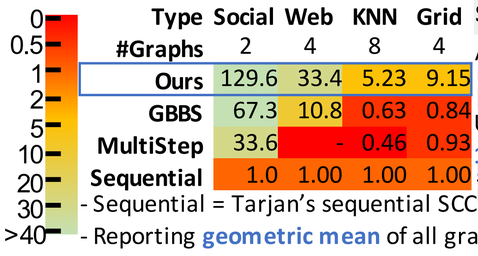
- v is **reachable** to u if there is a path from v to u.
- Two vertices v and u are **strongly connected** if v is reachable to u and u is reachable to v.

One SCC containing 1
Reachable to 1
Reachable from 1
Not reachable to or from 1



- A Strongly Connected Component (SCC) is a maximal subgraph that all pairs of vertices are strongly connected
- SCC problem is to find all SCCs in the graph
- Sequentially, Kosaraju's algorithm [1] and Tarjan's algorithm [6] have $O(n + m)$ cost.
- However, existing implementations do not parallelize well for large-diameter graphs.

We observed that existing parallel SCC algorithms on a 96-core machine can even be **slower** than Tarjan's sequential algorithm on many k-NN and lattice graphs.



Type	Social	Web	KNN	Grid
#Graphs	2	4	8	4
Ours	129.6	33.4	5.23	9.15
GBBS	67.3	10.8	0.63	0.84
MultiStep	33.6	-	0.46	0.93
Sequential	1.0	1.00	1.00	1.00

- Sequential = Tarjan's sequential SCC algorithm.
- Reporting **geometric mean** of all graphs of the same type.

Social, Web, KNN: All **real-world graphs**.
Graphs size: Up to **3.6B vertices** and **129B edges** per graph.
#cores=96

- We proposed **Vertical Granularity Control (VGC)** technique to improve the parallelism of reachability searches on large-diameter graphs.
- We design **Parallel Hash Bags** to efficiently maintain the frontiers in VGC.

Preliminary: BGSS Algorithm and Parallel BFS

Our algorithm is based on the BGSS algorithm [2] but improves the performance by avoiding $O(D)$ rounds of synchronizations in the breadth-first search (BFS), where D is the diameter of the graph, and thus reducing the scheduling overhead.

- BGSS SCC Algorithm
  - It divides V into $\log n$ batches with sizes $1, 2, 4, 8, \ldots$ in a **prefix doubling manner**.
  - It does **forward-backward reachability** queries on each batch, then finds SCCs and remove cross edges.

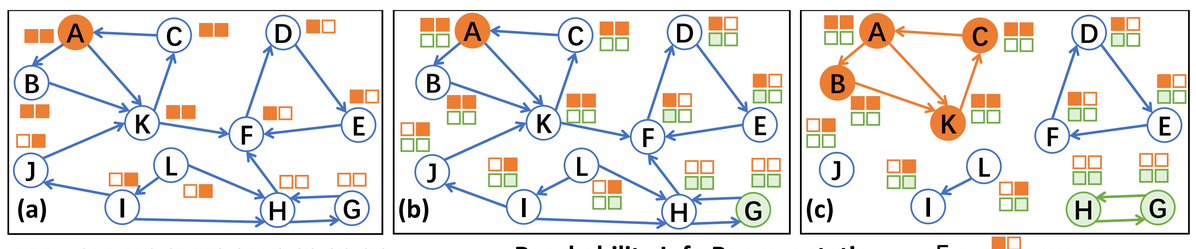
Algorithm 1: The BGSS algorithm for parallel SCC [2]

Input: A directed graph $G = (V, E)$
Output: The component label $L[\cdot]$ of each vertex.

$L \leftarrow \{-1, \ldots, -1\}$
Partition V into $\log n$ batches $P_{1..\log n}$, where $|P_i| = 2^{i-1}$
$V' \leftarrow V$
for $i \leftarrow 1, \ldots, \log n$ **do**
  $\mathcal{F} \leftarrow \{v \in P_i \cap V'\}$ ▷ Initial frontier
  `// MultiReach skips an edge` (u, v) `if` $L(u) \neq L(v)$
  $L_{out} \leftarrow \text{MultiReach}(G, L, \mathcal{F})$ ▷ Forward reachable pairs
  $L_{in} \leftarrow \text{MultiReach}(G^T, L, \mathcal{F})$ ▷ Backward reachable pairs
  Mark vertices in the same SCC with the same $L(\cdot)$, and remove *cross edges**.
return L

*cross edge**: Its two endpoints have different reachability info.

- If a vertex u is **forward and backward reachable** from a vertex v, then u and v are in the same SCC.
- If two endpoints of an edge have different reachability info, they are not in the same SCC.



SCCs: {ABCK} {DEF} {GH} {I} {J} {L}
Fig(a): Single-reachability search on A
Fig(b): Multi-reachability search on A and G
Fig(c): Result after running step (b)

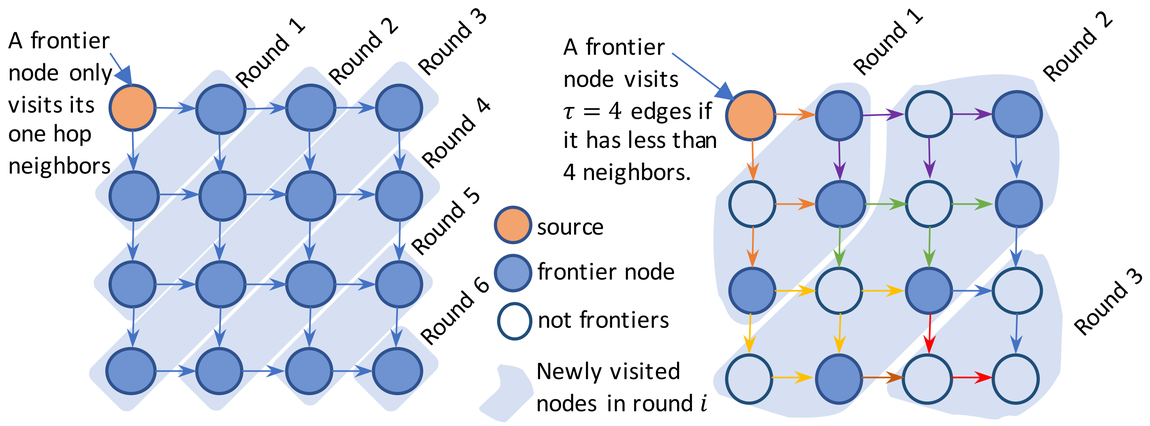
Reachability Info Representation:

From A	To A	Hollow squares mean **not reachable**
From G	To G	Solid squares mean **reachable**

E.g., ▢▢ means: reachable from A, not reachable to A; not reachable from G, reachable to G

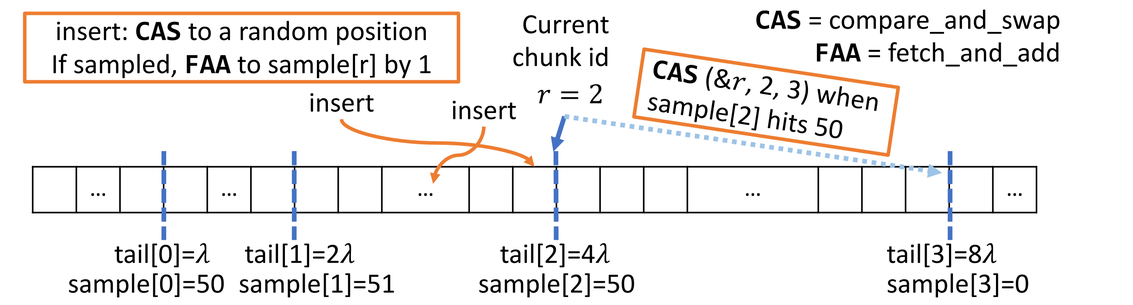
- Parallel BFS is a standard approach to implement reachability queries
  - Given a set of sources S, it maintains a **frontier** of vertices to process in each round.
  - In a round, it visits all the **neighbors** of the current frontier, and adds newly visited ones to the next frontier. We use `compare_and_swap` to guarantee that a vertex is only added once.

Vertical Granularity Control (VGC)



- Goal: Let each thread do enough work to hide the overhead of parallelism.
- Technique: Let each frontier node explore at least τ edges (τ is a tunable parameter).
  - If a vertex is dense (has more than τ neighbors), explore it the same as normal BFS.
  - If a vertex is sparse (has less than τ neighbors), do a local BFS to visit τ edges.
- Results: It can reduce the number of rounds to synchronize.

Parallel Hash Bag



- Motivation: We need a novel data structure to maintain the frontiers when using VGC, because the frontiers generated by VGC are not deterministic.
- Properties of a desired data structure:
  - $O(1)$ expected cost for concurrent insertion and dynamic resizing.
  - $O(s)$ cost to pack all elements in it, where s is the number of elements.
- A high-level approach:
  - Preallocate a $O(n)$ size of array, where n is the upper bound of elements size.
  - Divide the array into **chunks** with prefix-doubling sizes. Only one is actively in use.
- **Insertion:** Hash to a random position in the active chunk to store the element .
- **Resizing:** Mark the next chunk as the active chunk.
- **Listing all:** Pack all non-empty slots in the chunks that have been activated.

Experiment Results

We tested on 18 graphs including social networks, web graphs, k-NN graphs and lattice graphs. We run the experiments on a 96 cores (192 hyperthreads) machine with 1.5T memory. We compared our algorithm to GBBS[3], iSpan [4], and Multi-step [5].

Overall Performance

- Our algorithm is the fastest on **16** over **18** graphs. On average, our algorithm is **6**× faster than the best of other implementations.
- Our algorithm has dominant advantages on large-diameter graphs.

	Social			KNN								
	LJ	TW	MEAN	HH5	CH5	GL2	GL5	GL10	GL15	GL20	COS5	MEAN
Ours	75.8	317	155	2.16	0.77	5.67	5.58	6.24	5.42	5.60	58.6	5.26
GBBS	24.5	185	67.3	0.11	0.05	1.13	0.46	0.76	0.83	0.92	15.8	0.63
iSpan	~~58.5~~	c	-	0.57	0.20	t	t	0.26	0.39	0.49	t	0.36
MultiStep	20.7	54.4	33.6	0.20	0.02	0.41	0.25	1.30	1.07	1.08	3.29	0.46
Seq	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

	Web					Lattice					Overall
	SD	CW	HL14	HL12	MEAN	SQR	REC	SQR'	REC'	MEAN	MEAN
Ours	52.7	33.4	30.1	19.1	31.7	26.8	13.5	5.00	3.75	9.08	12.9
GBBS	19.7	14.6	9.22	5.05	10.8	1.39	0.41	1.45	0.60	0.84	2.13
iSpan	~~21.7~~	n	n	n	-	~~3.47~~	~~1.32~~	0.26	~~0.70~~	0.96	1.18
MultiStep	55.8	n	n	n	-	1.23	0.30	2.16	0.92	0.93	1.35
Seq	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

0 0.25 0.5 1 2 4 8 16 >32

Figure 1. The heatmap of relative speedup for parallel SCC algorithms over the sequential algorithm using 96 cores (192 hyperthreads).

Our algorithm is **8.3**× and **9.4**× faster than the best existing parallel algorithms on k-NN and lattice graphs separately.

Breakdown of SCC to show the effect of VGC and hash bags

- **Plain** is our implementation that only applies the parallel hash bags. **VGC** is our implementation that applies VGC and parallel hash bags.
- **First SCC** and **Multi-search** steps mainly query reachability info, which is the main focus of our optimizations.
- Comparing GBBS and Plain, hash bags bring **2**× speedup on average.
- Comparing Plain and VGC, VGC has a small effect on low-diameter graphs (**1.13**× speedup on average), but a large effect on large-diameter graphs (**3.32**× speedup).

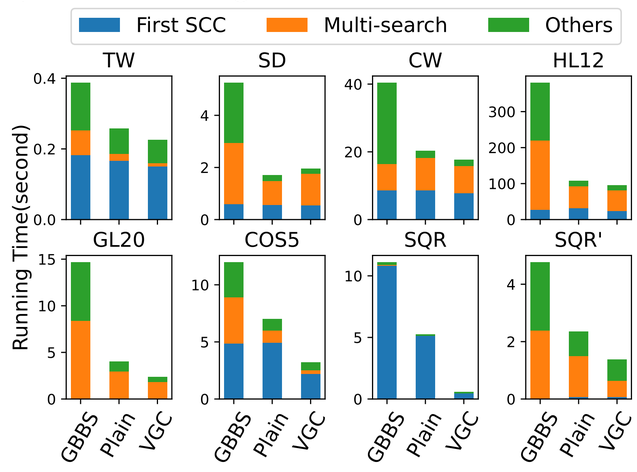


Figure 2. Comparing the breakdown with GBBS.

Scalability

- Ours is the only one that achieves **almost linear speedup** on assorted graphs.
- The speedups of other existing parallel algorithms **stop increasing** or even **decrease** on some graphs (TW, SD, GL5, SQR') when the number of threads is more than 24.

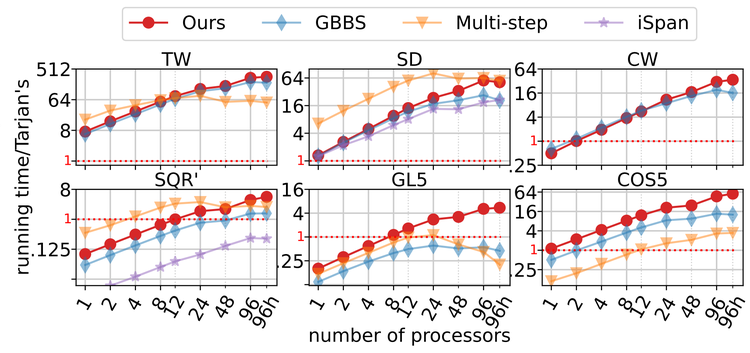


Figure 3. Comparing speedup over Tarjan's sequential algorithm on different algorithms on different number of processors.

References

[1] V Aho Alfred, E Hopcroft John, D Ullman Jeffrey, V Aho Alfred, H Bracht Glenn, D Hopkin Kenneth, C Stanley Julian, Brachu Jean-Pierre, Brown A Samler, Brown A Peter, et al. *Data structures and algorithms*. USA: Addison-Wesley, 1983.

[2] Guy E. Blelloch, Yan Gu, Julian Shun, and Yihan Sun. Parallelism in randomized incremental algorithms. *J. ACM*, 2020.

[3] Laxman Dhulipala, Jessica Shi, Tom Tseng, Guy E Blelloch, and Julian Shun. The graph based benchmark suite (GBBS). In *International Workshop on Graph Data Management Experiences & Systems (GRADES)*, pages 1–8, 2020.

[4] Yuede Ji, Hang Liu, and H Howie Huang. ispan: Parallel identification of strongly connected components with spanning trees. In *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, pages 731–742. IEEE, 2018.

[5] George M Slota, Sivasankaran Rajamanickam, and Kamesh Madduri. Bfs and coloring-based parallel algorithms for strongly connected components and related problems. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 550–559. IEEE, 2014.

[6] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM J. on Computing*, 1(2):146–160, 1972.