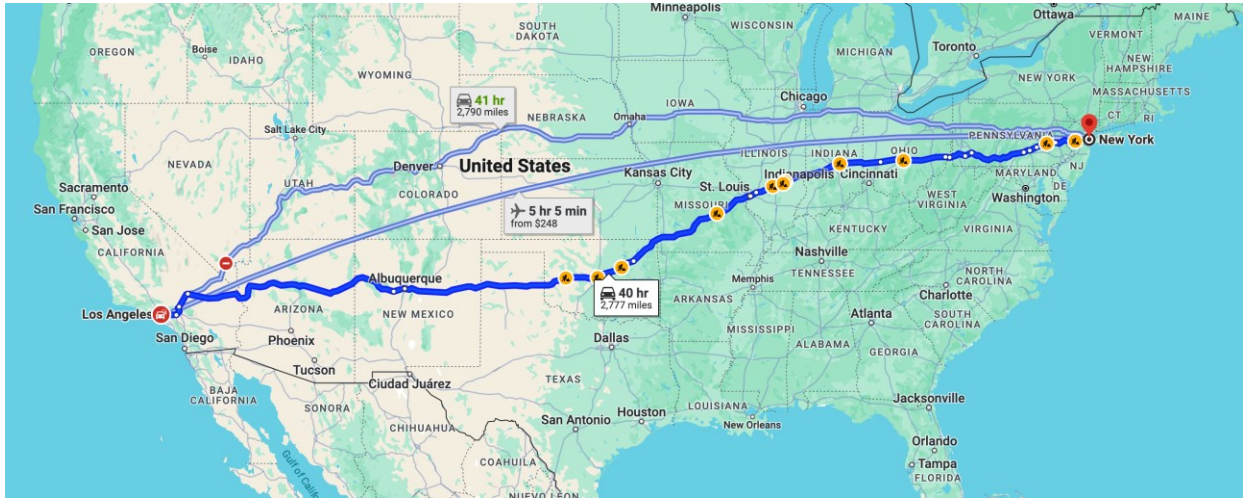
PASGAL: Parallel And Scalable Graph Algorithm Library

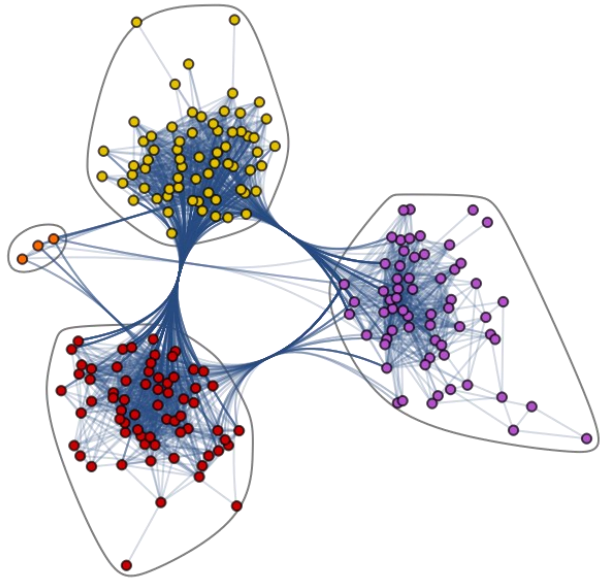
Xiaojun Dong, Yan Gu, Yihan Sun, Letong Wang
University of California, Riverside

Full version: https://arxiv.org/abs/2404.17101
Code: https://github.com/ucrparlay/PASGAL

Graphs are fundamental models for real-world problems

- Route planning
- Social network analytics
- Machine learning





https://gatton.uky.edu/about-us/stay-connected/news/2020/links-center-social-network-analysis-workshop-success
https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial

Algorithms in our library

- Breadth-first search (**BFS**) [This paper]
- Single-source shortest paths (**SSSP**) [Dong et al., SPAA ‘21]
- Strongly connected components (**SCC**) [Wang et al., SIGMOD ‘23]
- Biconnected components (**BCC**) [Dong et al., PPoPP ‘23]
- More to appear (k-core and related peeling algorithms, point-to-point shortest paths, etc.)

Many graph processing libraries have been designed

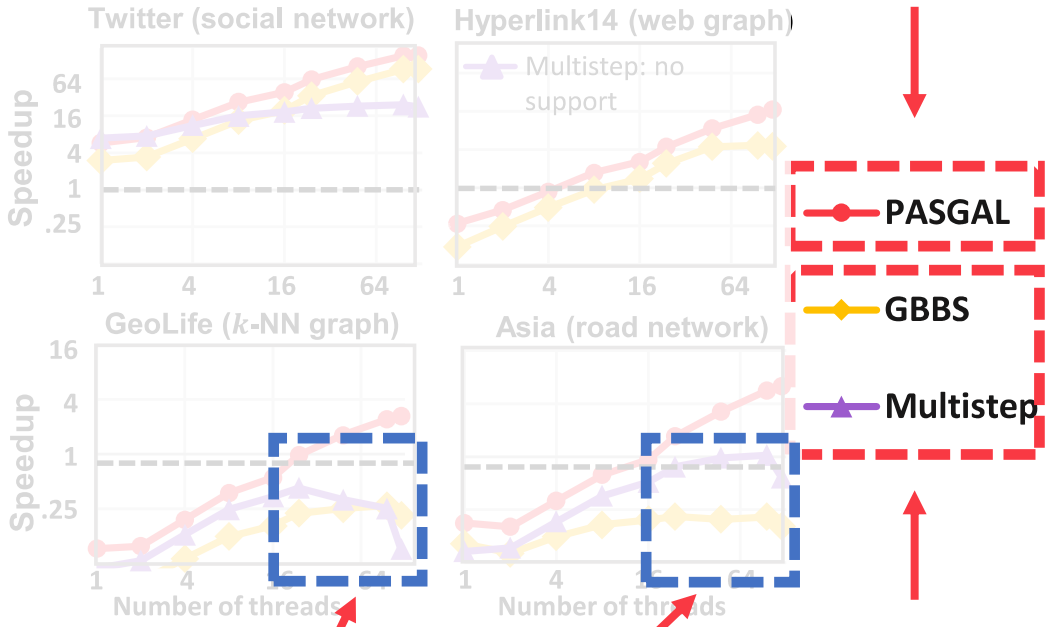
- Multi-core CPU graph processing libraries
  - GAP Benchmark Suite (GAPBS)
  - Graph Based Benchmark Suite (GBBS)
  - Galois
  - Etc.
- And many more for GPU and distributed systems
  - Gunrock, Medusa, GraphX, Lux, etc.
- Why do we still need another graph processing library?

Existing graph libraries have scalable issues on large-diameter graphs

- Speedups relative to the sequential Tarjan's algorithm on the strongly connected components (SCC) problem

- On low-diameter graphs, all libraries have good scalability

- On large-dimeter graphs, both GBBS and multistep don't scale well with increasing number of threads



Good scalability! Our library!

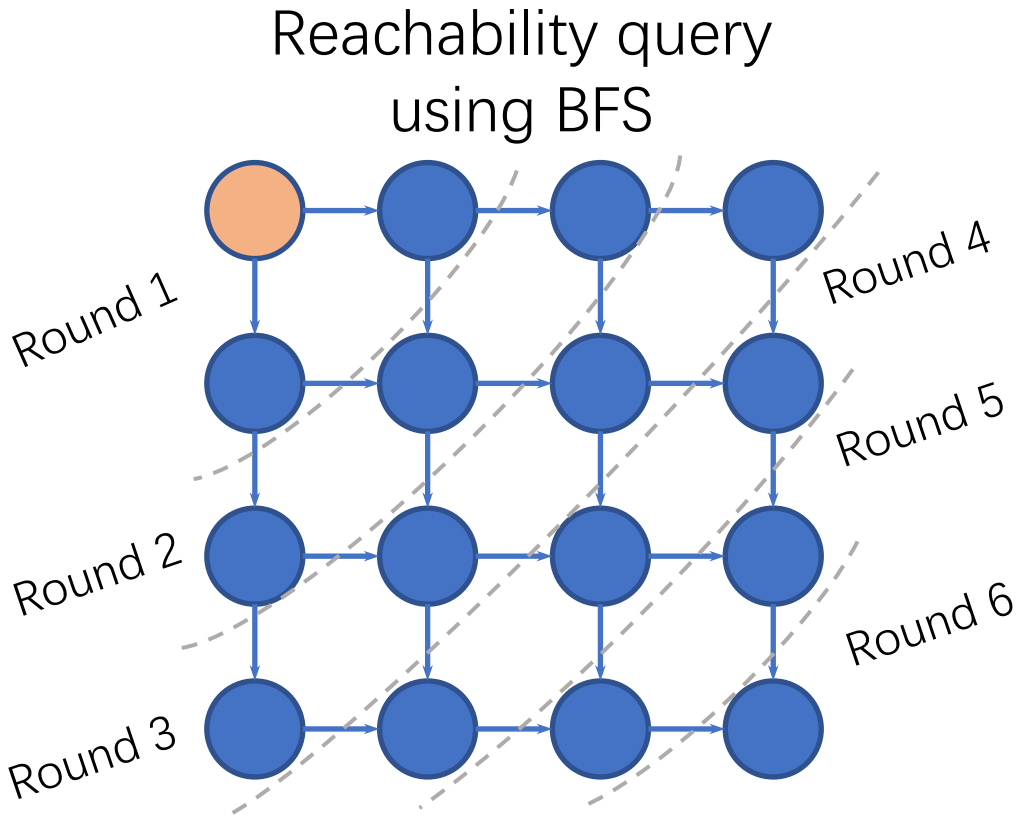
Competitors

Worse performance with more threads

Machine used:
- 96 cores
- 1.5TB memory

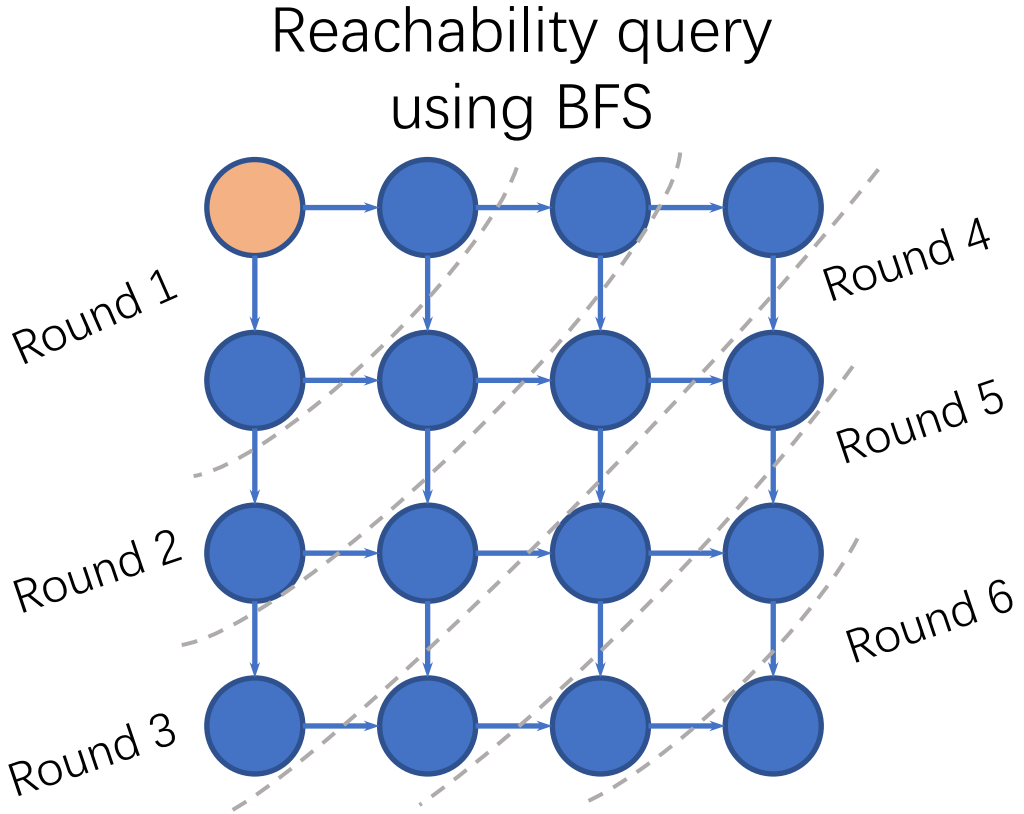
“Parallel overhead”

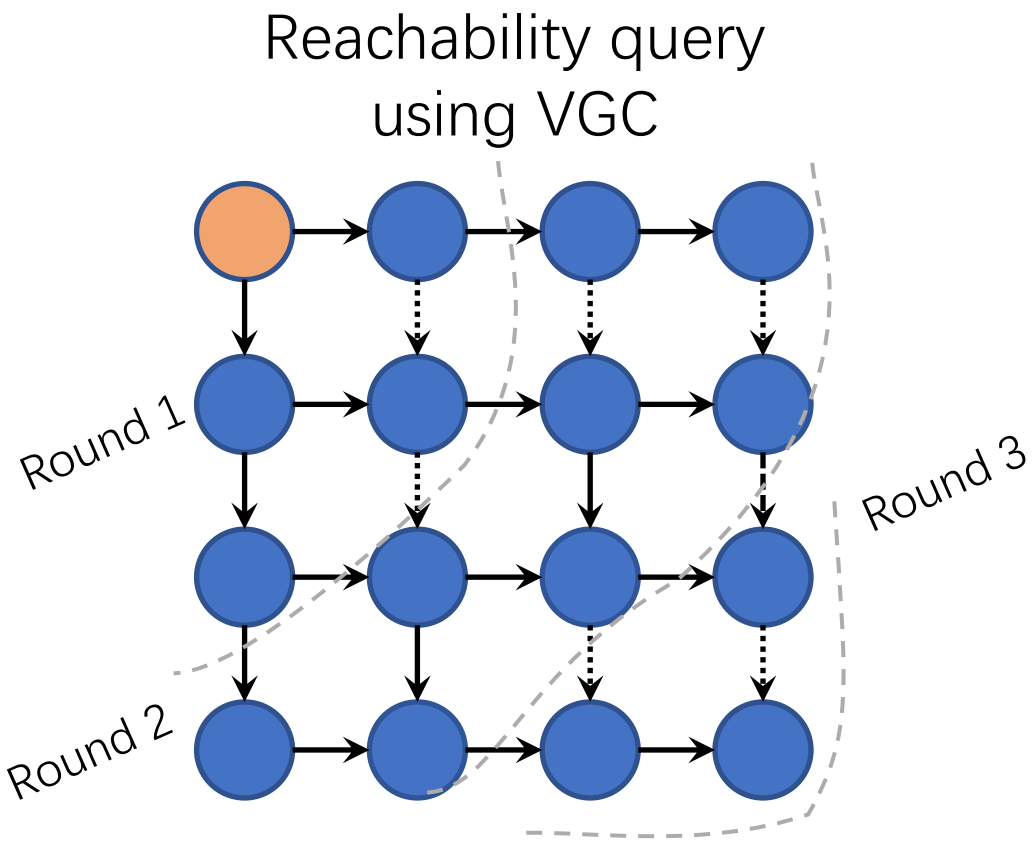
- **Parallelism comes at a cost: synchronizing threads can be expensive**
- **Typical example: compute reachability using breadth-first search (BFS)**
  - Requires $O(D)$ rounds to finish where D is the graph diameter



Vertical Granularity Control (VGC)

- **Our goal: reduce #synchronization rounds**
- **Technique: vertical granularity control (Wang et al., SIGMOD '23)**
  - Each vertex can visit multi-hop neighbors by maintaining a local queue



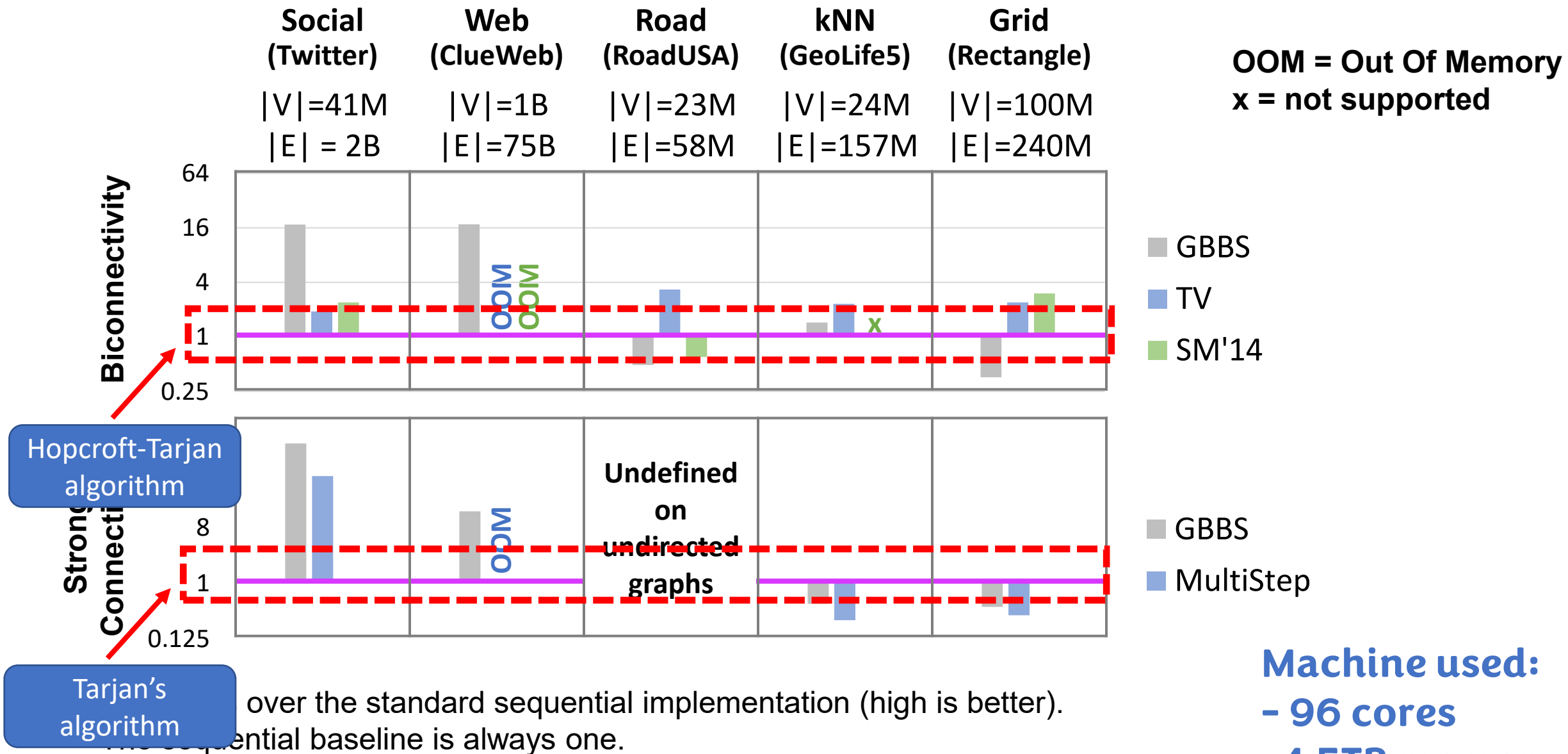


Algorithm redesign

- **Our biconnectivity algorithm, FAST-BCC, is provably fast and space-efficient [Dong et al., PPoPP '23]**
  - Optimal work: $O(n+m)$
  - Optimal span: $O((\log n)^3)$
  - Space-efficient: $O(n)$ auxiliary space

- **Polylog span v.s. $O(D)$ span in existing implementations**

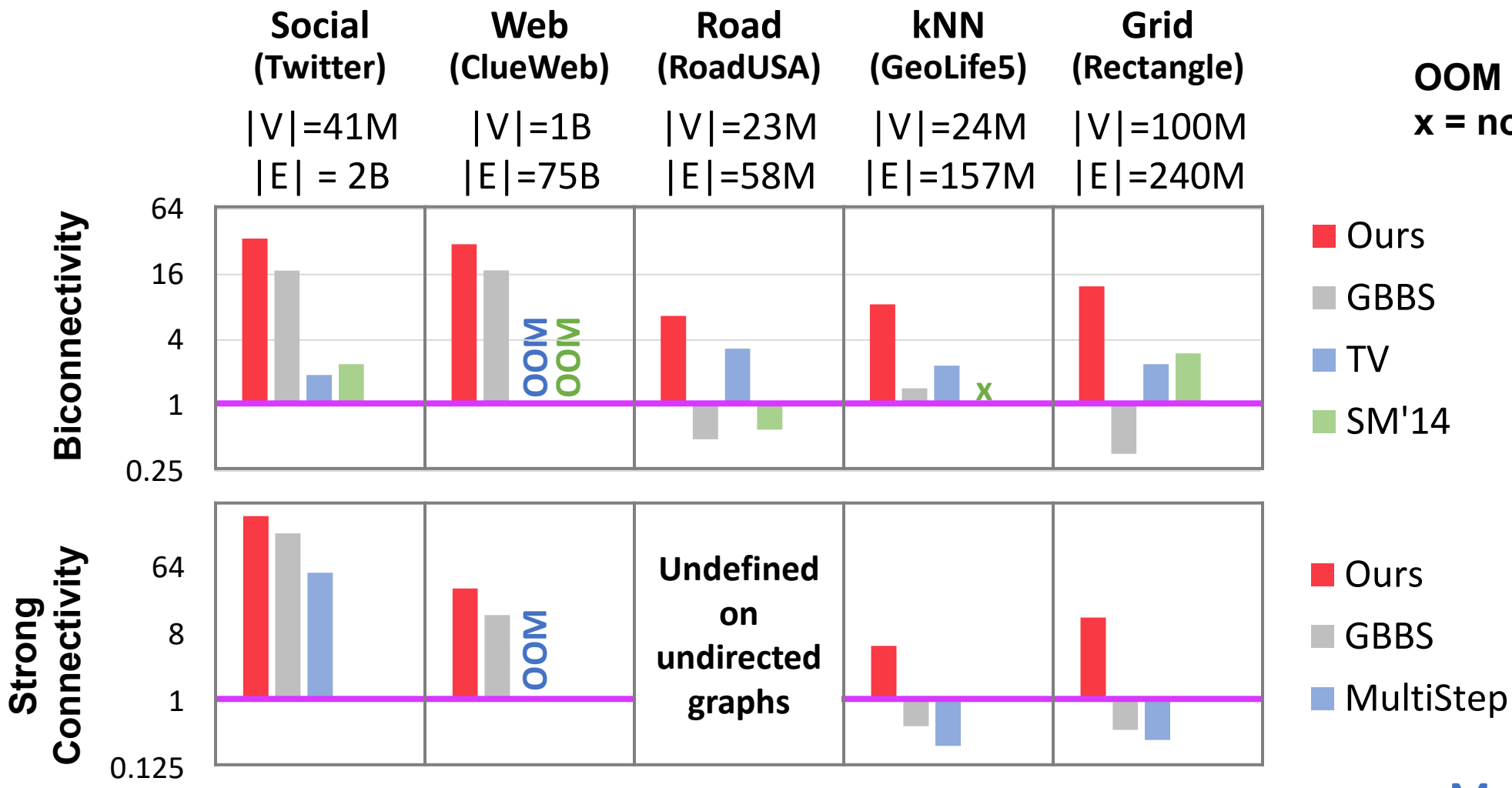
Experimental results



over the standard sequential implementation (high is better).
ential baseline is always one.
Bars below one means slower than sequential.

Machine used:
- 96 cores
- 1.5TB memory

Experimental results



OOM = Out Of Memory
x = not supported

Speedups over the standard sequential implementation (high is better).
The sequential baseline is always one.
Bars below one means slower than sequential.

Machine used:
- 96 cores
- 1.5TB memory

Summary

- **New graph processing library**
  - Efficient and scalable to various graph types, many processors, and large graph sizes
- **Core techniques**
  - Vertical granularity control (VGC) and parallel hash bags
  - Algorithm redesigns
- **Algorithms implemented**
  - Breadth-first search (BFS) [This paper]
  - Single-source shortest paths (SSSP) [Dong et al., SPAA '21]
  - Strongly connected components (SCC) [Wang et al., SIGMOD '23]
  - Biconnected components (BCC) [Dong et al., PPoPP '23]
  - More to appear (k-core and related peeling algorithms, point-to-point shortest paths, etc.)
- **Full version: https://arxiv.org/abs/2404.17101**
- **Code: https://github.com/ucrparlay/PASGAL**
- **Contact: Xiaojun Dong (xdong038@ucr.edu)**