Aquila: Adaptive Parallel Computation of Graph Connectivity Queries

Yuede Ji

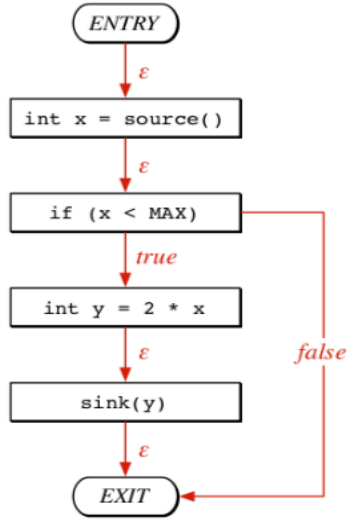
H. Howie Huang

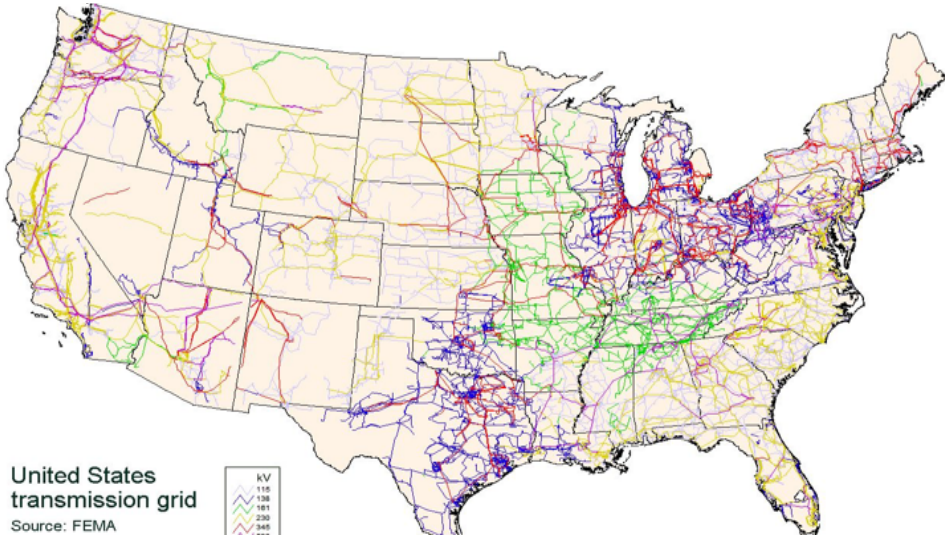
Graph Computing Lab
George Washington University

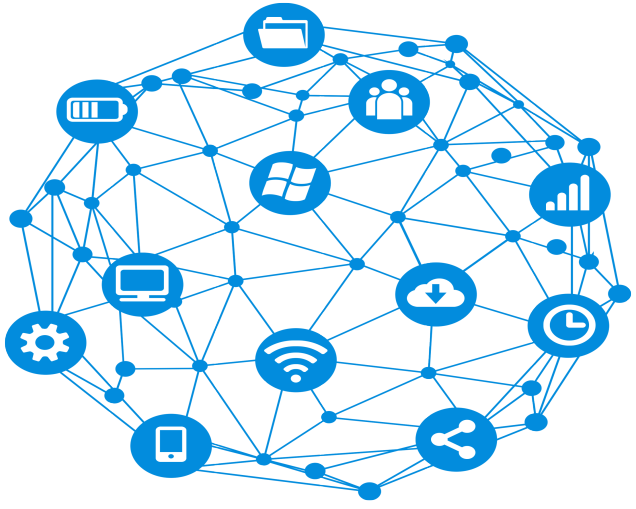
Part 1: Introduction

Graph Is All Around

```
void foo()
{
    int x = source();
    if (x < MAX)
    {
        int y = 2 * x;
        sink(y);
    }
}
```

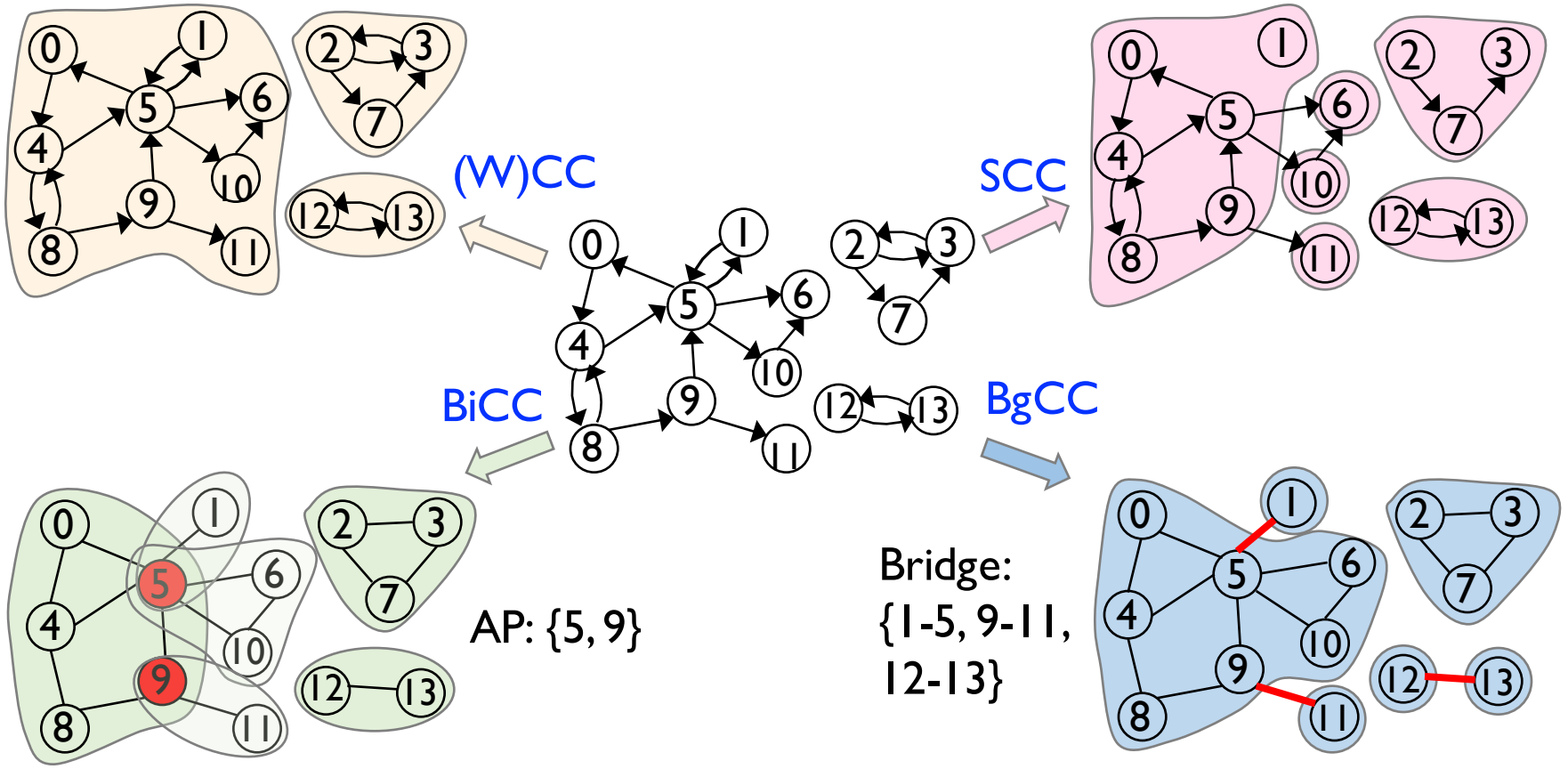






Graph Connectivity Algorithms

- Directed
  - Weakly connected component (WCC)
  - Strongly connected component (SCC)
- Undirected
  - Connected component (CC)
  - Biconnected component (BiCC), articulation point (AP)
  - Bridgeless connected component (BgCC), Bridge
- XCC to represent all



Applications

- Graph analytics
  - Directed acyclic graph (DAG) → SCC
  - Betweenness centrality (BC) → BiCC [PPoPP'16]
- Pattern recognition
  - Connected-component labeling in computer vision → CC
  - SBV-Cut node clustering → CC, AP, bridge [DKE'12]
- Cybersecurity
  - Spamming botnet detection → CC [NSDI'09]
  - Suspicious DNS query → SCC [ICNP'10]

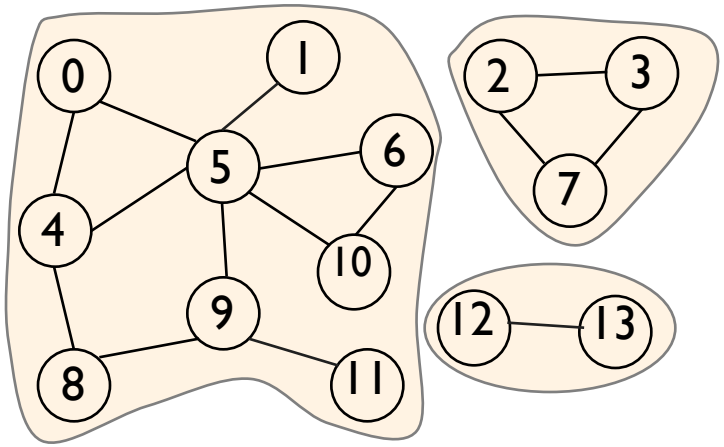
[PPoPP'16] Lei Wang, Fan Yang, Liangji Zhuang, Huimin Cui, Fang Lv, and Xiaobing Feng. Articulation points guided redundancy elimination for betweenness centrality. Proceedings of PPoPP, 2016
[DKE'12] Kim, Mijung, and K. Selçuk Candan. "SBV-Cut: Vertex-cut based graph partitioning using structural balance vertices." Data & Knowledge Engineering (2012).
[NSDI'09] YaoZhao, YinglianXie, FangYu, QifaKe, YuanYu, YanChen, andEliotGillum. 2009. BotGraph: Large Scale Spamming Botnet Detection. Proceedings of NSDI, (2009)
[ICNP'10] Nan Jiang et al. 2010. Identifying suspicious activities through dns failure graph analysis. Proceedings of ICNP (2010).

Motivation

1. Existing parallel graph computation frameworks only support a limited number of connectivity algorithms.

2. Different connected components share similar heterogeneous properties, similar computation techniques, while existing systems did not take advantage of them.

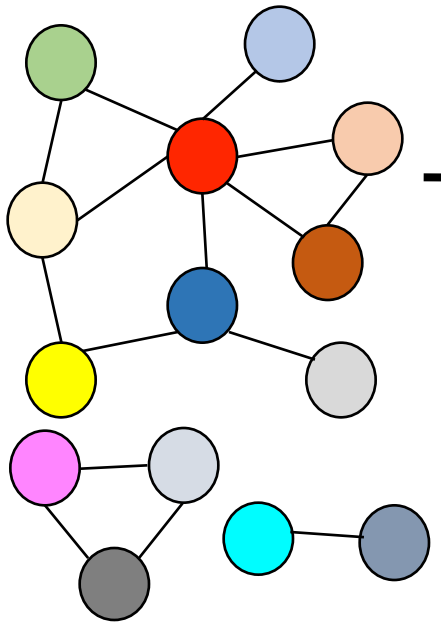
3. Existing methods only provide complete computation, while many queries can actually be answered with partial computation.



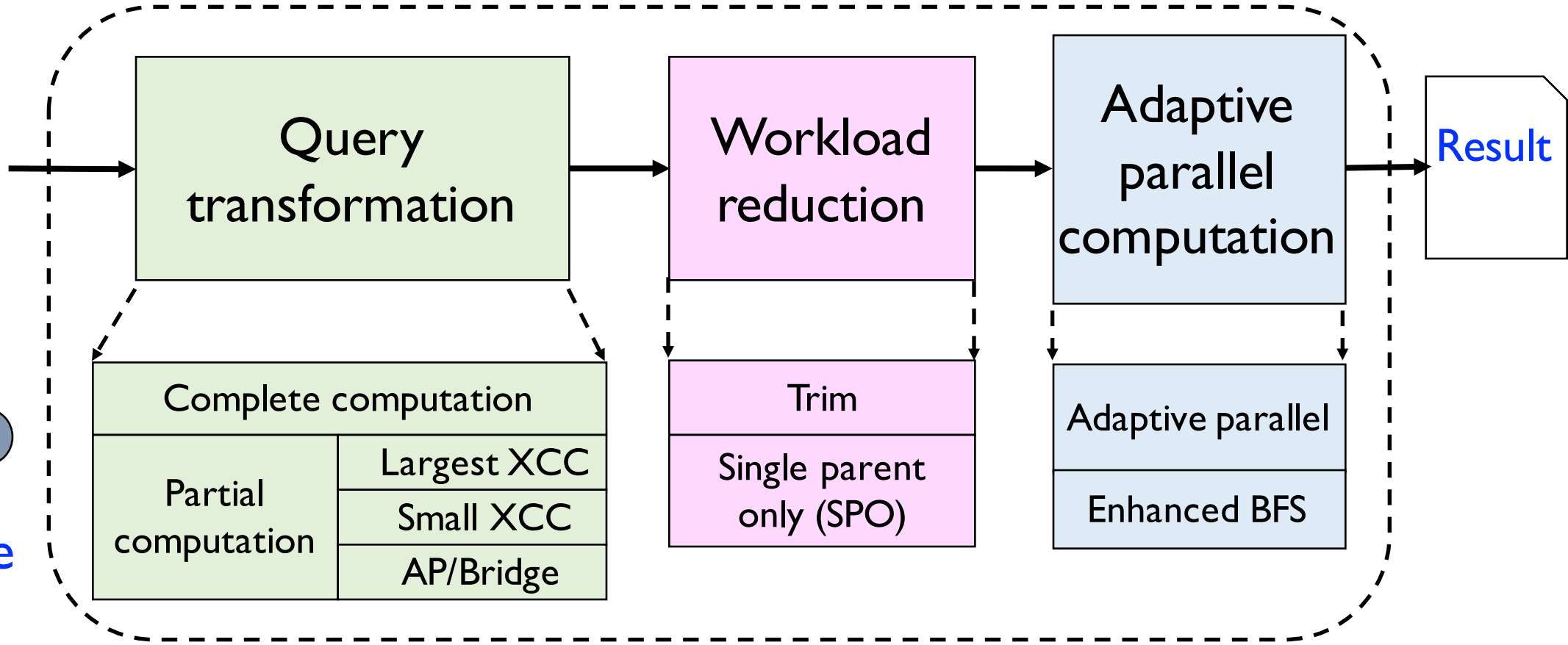
Is this graph connected?

Part 2: Aquila Framework

Aquila Overview

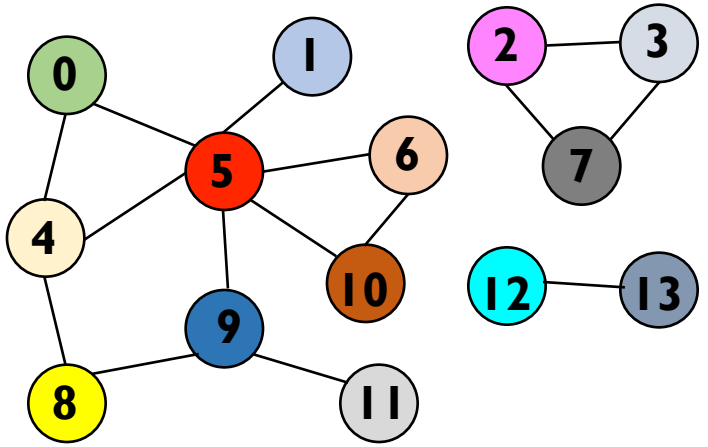


Query: Is the graph connected?

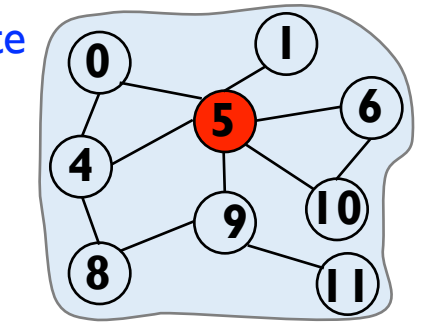


Technique #1: Query Transformation

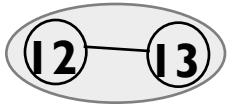
- Complete computation
- Partial computation
  - Largest XCC
  - Small XCC
  - AP and Bridge



Query 1: Compute largest CC



Query 2: Compute small CC



Query 3: Compute AP/Bridge

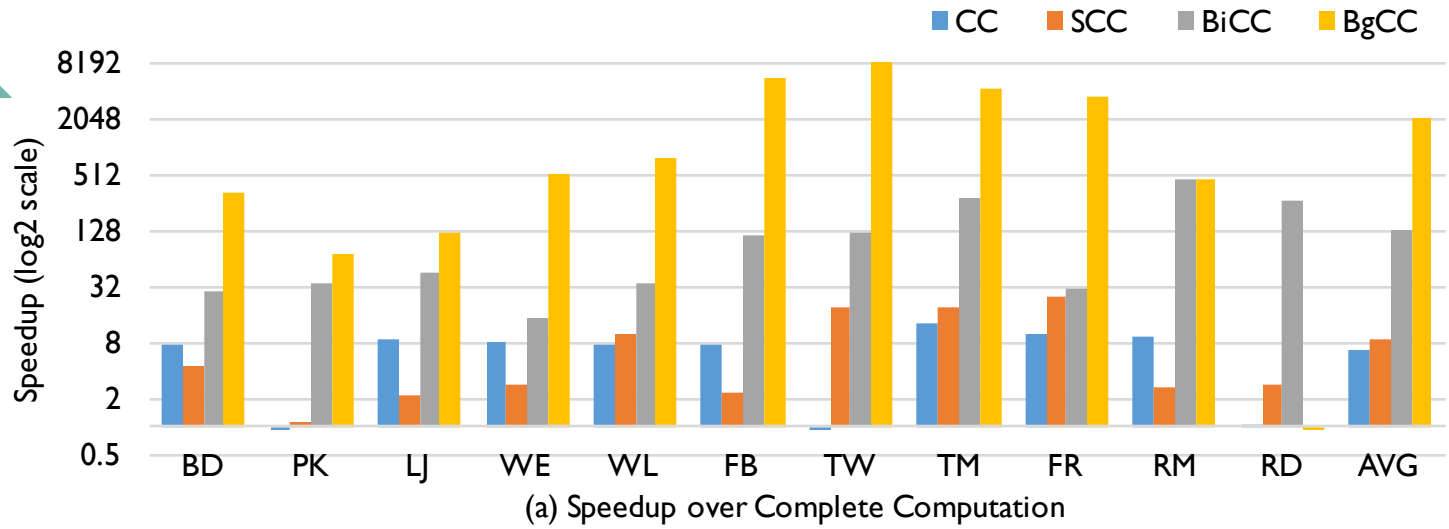
AP: {5, 9}

Bridge: {1-5, 9-11, 12-13}

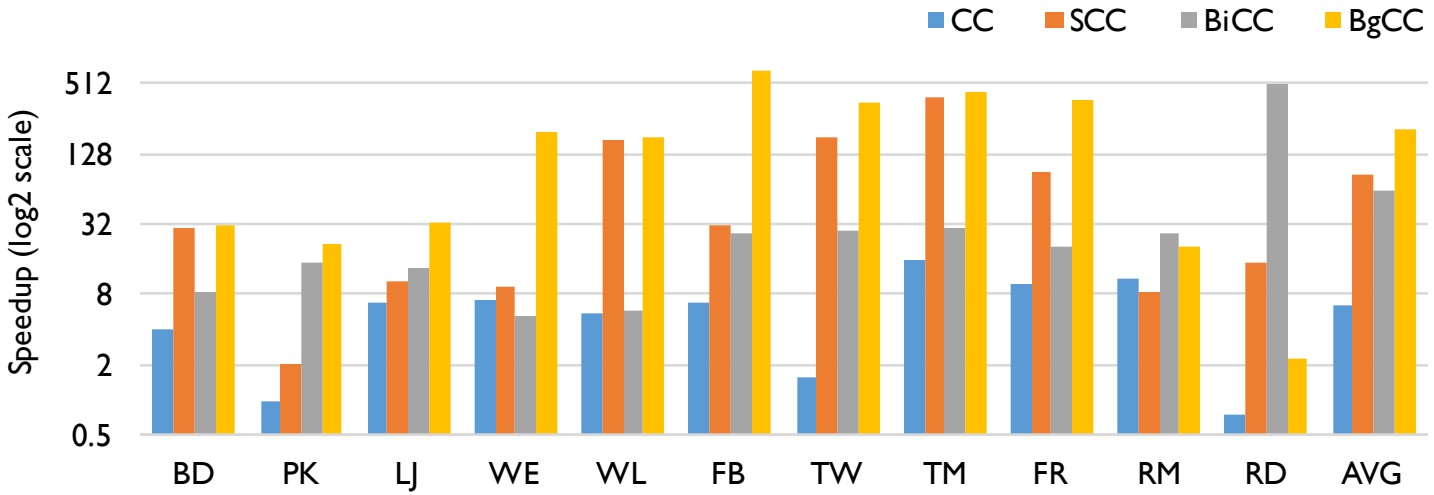
Benefit of Small XCC Query

- Over complete computation
  - 7x, 8x, 132x, 2,137x for (W)CC, SCC, BiCC, and BgCC, respectively
- Over selecting an arbitrary pivot
  - 6x, 85x, 62x, 209x for (W)CC, SCC, BiCC, and BgCC, respectively

Higher is better



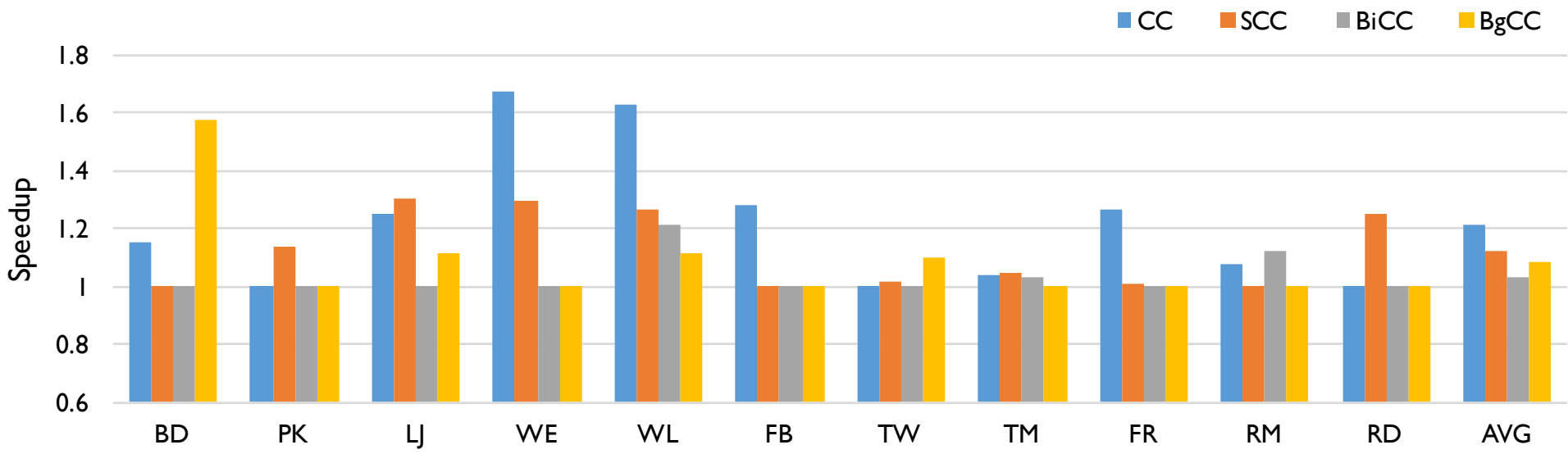
(a) Speedup over Complete Computation



(b) Speedup over selecting an arbitrary pivot

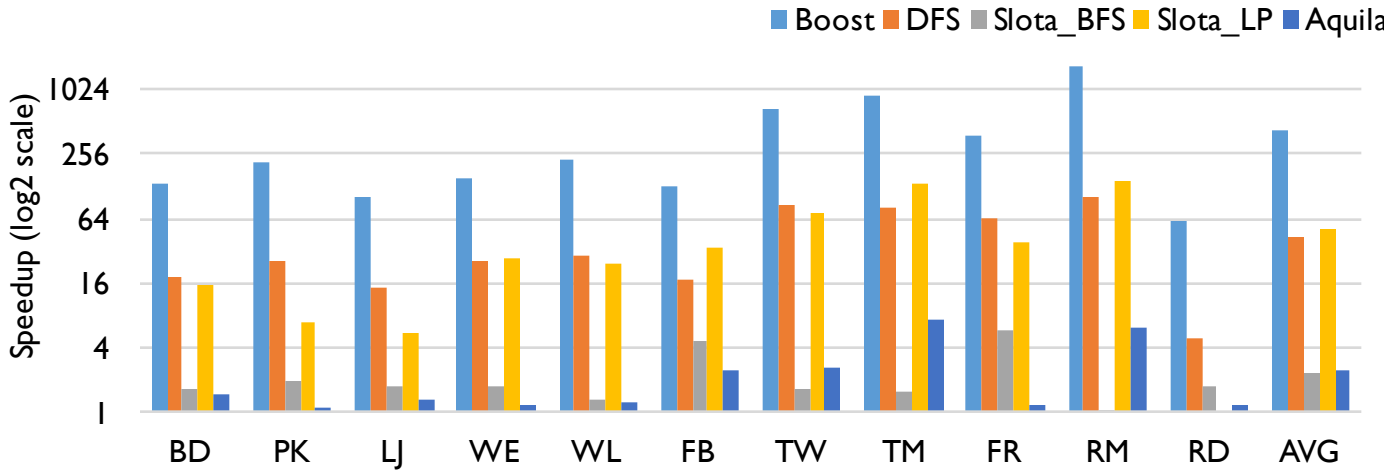
Benefit of Largest XCC Query

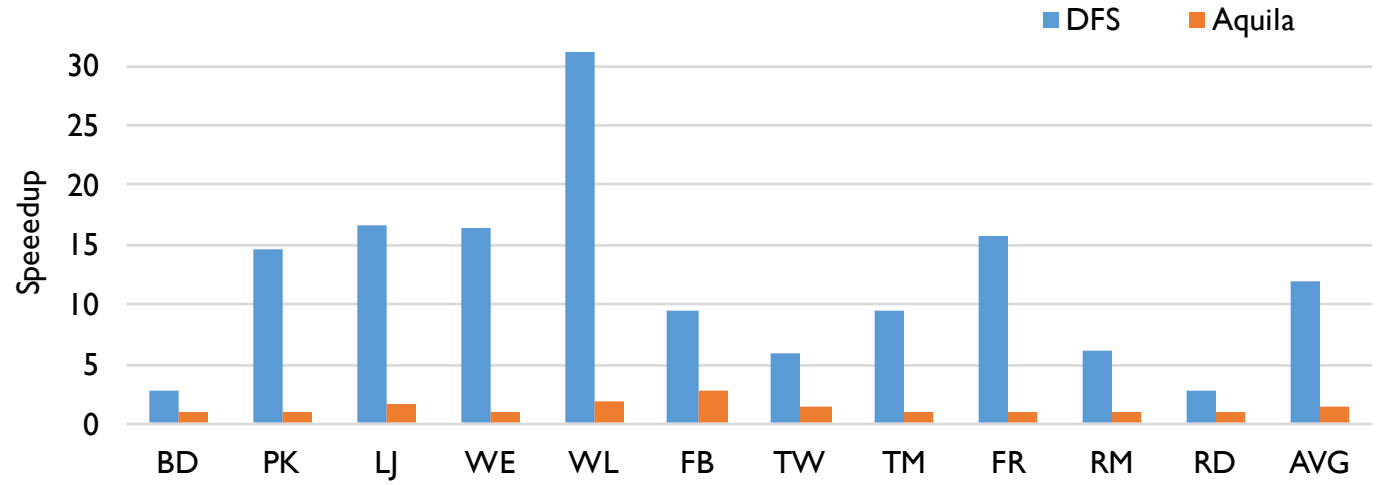
- 1.2x, 1.1x, 1.03x, and 1.1x for (W)CC, SCC, BiCC, and BgCC, respectively



Benefit of AP and Bridge Query

- Articulation point (AP)
  - 423x, 43x, 50x, 2.3x, 2.5x speedup over Boost, DFS, Slota_BFS, Slota_LP, Aquila's complete computation, respectively
- Bridge
  - 12x and 1.3x speedup over DFS and Aquila's complete computation





Technique #2: Workload Reduction

- State-of-the-art BiCC computation algorithm [HiPC'14]
  - Lemma 1: On the constructed BFS tree, for any non-root vertex *p*, after removing it, if any of its children cannot reach a vertex at the same level of *p*, then *p* is an AP. Similarly for the bridge.
- Observation
  - It needs to run up to |V| BFSes, but only less than 1% of them will find a BiCC.

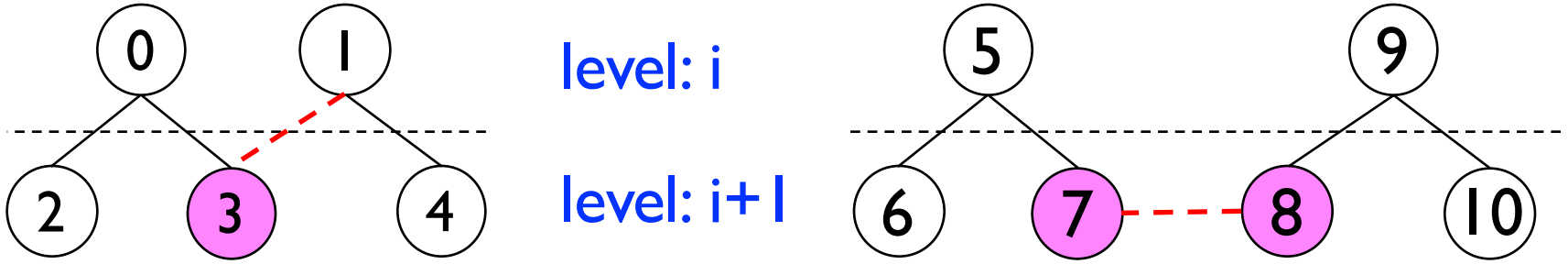
Algorithm 1: bfsBiCC(G, *level, *parent)

```
1 pivot = selectPivot(G);
2 bfs(G, level, parent);
3 foreach v ∈ reverseBfsOrder(G) do
4     p = parent[v];
5     l = bfsConstrained(G, v, p, level);
6     if l < level[p] then
7         Mark the visited edges as one BiCC;
```

[HiPC'14] George M Slota and Kamesh Madduri. 2014. Simple parallel biconnectivity algorithms for multicore platforms. Proceedings of HiPC

Technique #2: Workload Reduction

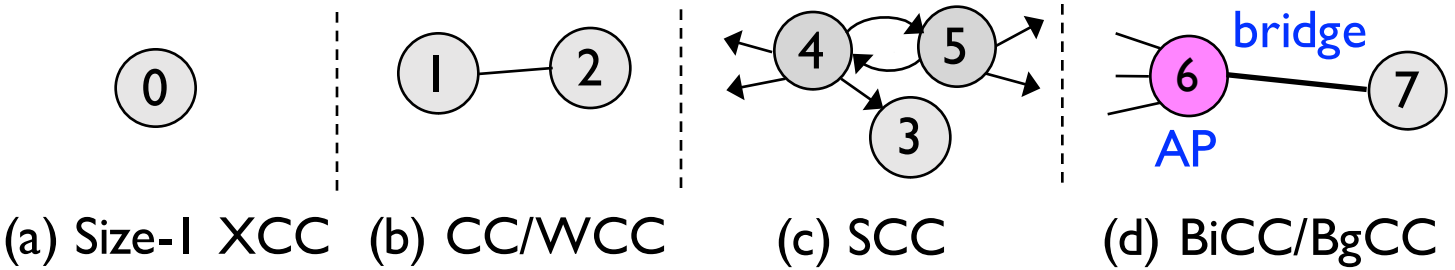
- Single parent only (SPO) technique
  - Lemma 2: For any non-root vertex *p*, after removing it, if one of its children *v* can reach a vertex at the same level of *p*, then *p* is not an AP from the view of *v*. Not checking vertex *v* will not affect the correctness. Similarly for the bridge.
  - Two types of second parent
    - Direct & Sibling induced



(a) Direct second parent (b) Sibling induced second parent

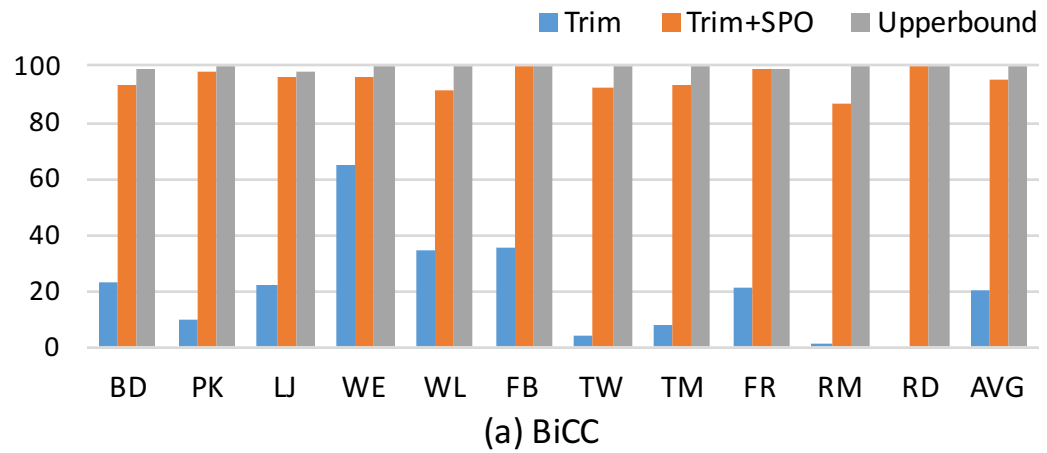
Technique #2: Workload Reduction

- Trim
  - Quickly remove trivial XCCs as the real-world graphs have a large number of them
  - Leverage the existing size-1 and size-2 trim for CC and SCC
  - Design new trim patterns for BiCC, BgCC, and related algorithms

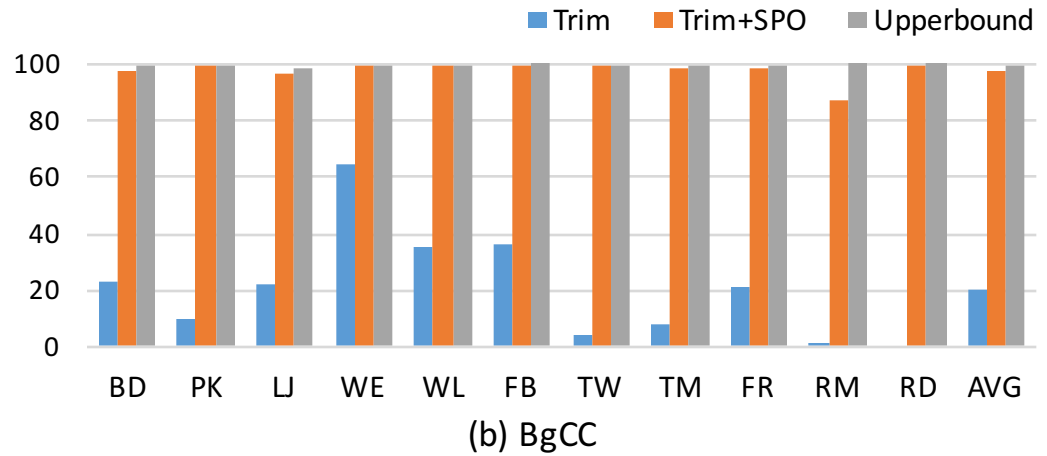


Benefits of Workload Reduction

- Trim
  - 21% for both BiCC and BgCC
- SPO
  - 74% for BiCC, 77% for BgCC
- Together
  - 95% for BiCC (upperbound 99.6%)
  - 98% for BgCC (upperbound 99.7%)



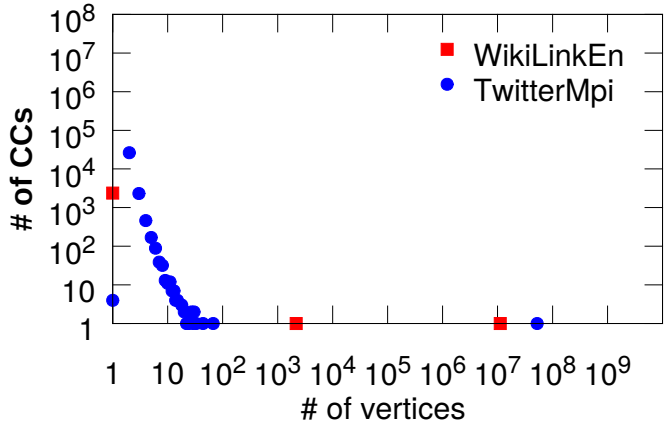
(a) BiCC

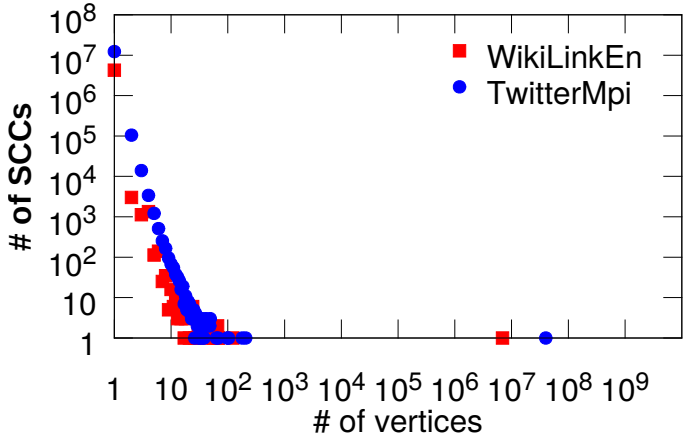


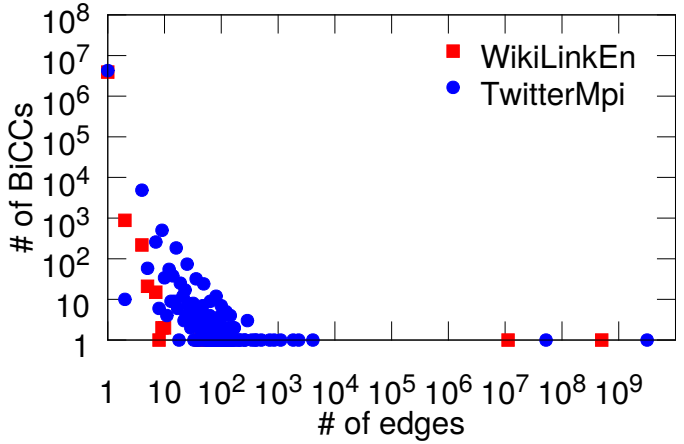
(b) BgCC

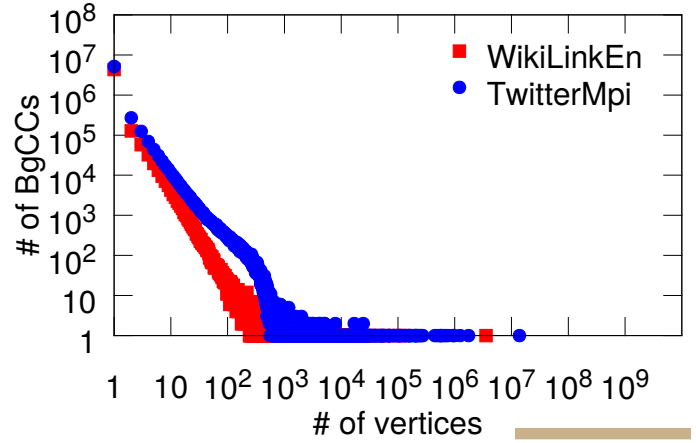
Technique #3: Adaptive Parallel Computation

- Observation: Irregular task property
  - A few large XCCs take majority of the graph and their sizes are close to the order of graph size
  - The rest is a large number of trivial XCCs whose count is close to the order of graph size



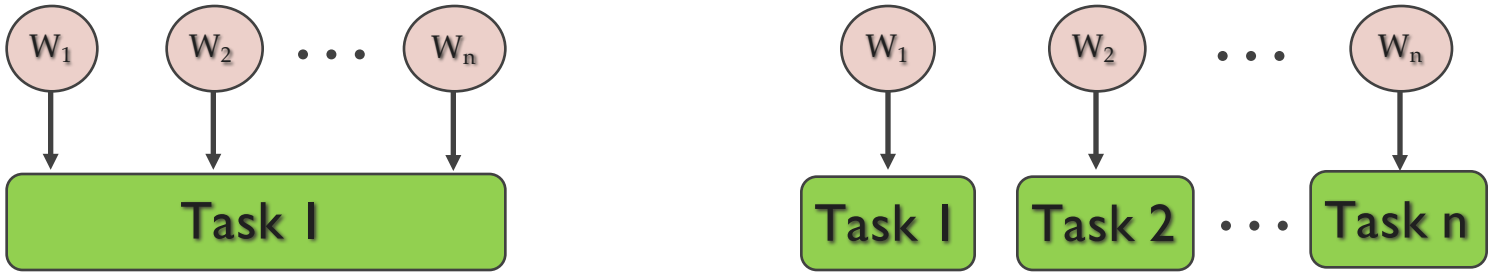






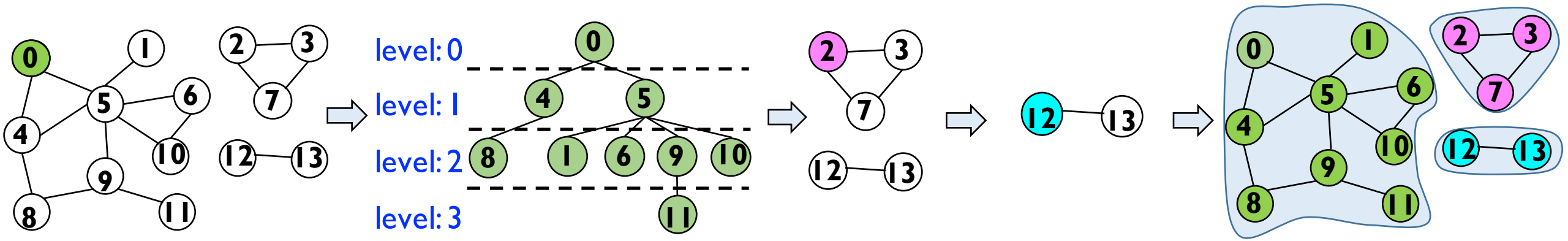
Technique #3: Adaptive Parallel Computation

- Data parallel vs. task parallel
  - Data parallel → all the "workers" work together for one task
    - Fit for the large tasks in XCC
  - Task parallel → each "worker" works for a different task
    - Fit for the rest tasks in XCC



BFS for CC Computation

- Pros: Efficient for finding one specific CC
- Cons: Low efficient for many CCs



(a) An example graph

(b) BFS tree, the identified CC

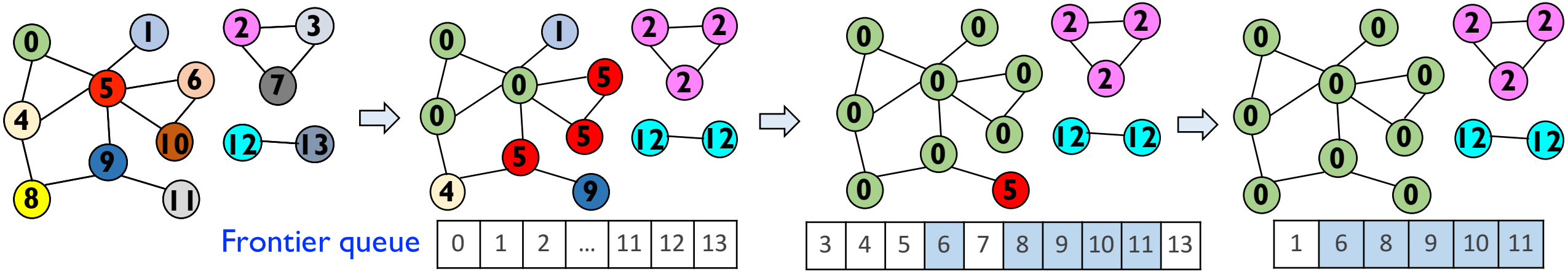
(c) Start the 2nd BFS on the remaining graph

(d) Repeat with the 3rd BFS

(e) End of CC computation

Label Propagation for CC Computation

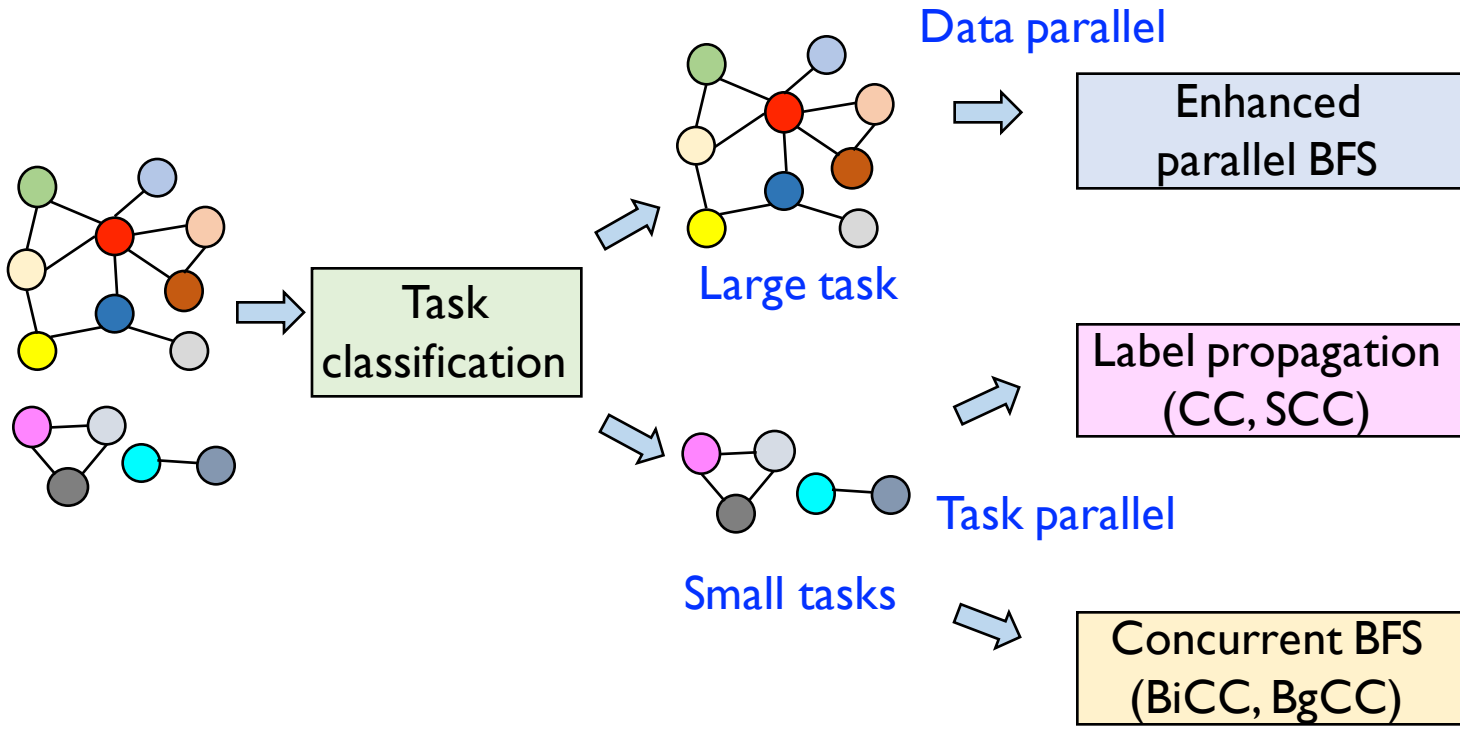
- Pros: can find all the CCs by running once (although several iterations)
- Cons: inflated workload



(f) Label initialization (g) 1st iteration of label propagation (h) 2nd iteration (i) 3rd iteration

Technique #3: Adaptive Parallel Computation

- Large task:
  - Enhanced parallel BFS
    - Multi-pivot
    - Adaptive synchronization [SC'18]
- Small tasks
  - Label propagation → CC, SCC
  - Concurrent BFS → BiCC, BgCC



[SC'18] Yuede Ji, Hang Liu, and H Howie Huang. 2018. iSpan: Parallel Identification of Strongly Connected Components with Spanning Trees. Proceedings of SC (2018).

Part 3: Experiment

Graph Benchmarks

Graph	Abbrev.	Description	# Nodes	# Directed Edges	# Undirected Edges	# CCs	Largest CC Percentage
Baidu	BD	Baidu Baike hyperlink network	2.1M	17.8M	34M	15,561	98.4%
Pokec	PK	Pokec online social network	1.6M	30.6M	44.6M	1	100%
Livejournal	LJ	Livejournal online social network	4.8M	68.5M	85.7M	1,876	99.9%
WikiEn	WE	Wikipedia hyperlink network	18.3M	172.2M	253.8M	1,366	99.9%
WikiLinkEn	WL	Wikipedia hyperlink network	11.2M	340.3M	516.9M	3,061	99.9%
Facebook	FB	Facebook connection	96.1M	679.7M	1.2B	5	99.9%
TwitterWww	TW	Twitter user following network	41.7M	1.5B	2.4B	1	100%
TwitterMpi	TM	Twitter user following network	52.6M	2B	3.2B	29,533	99.9%
Friendster	FR	Friendster friendship network	68.3M	2.6B	3.6B	323,276	98.7%
RMAT	RM	R-MAT generator	4M	256M	506.2M	1.9M	52.1%
Random	RD	Gtgraph generator	4M	256M	512M	1	100%

GW

Experiment Setting

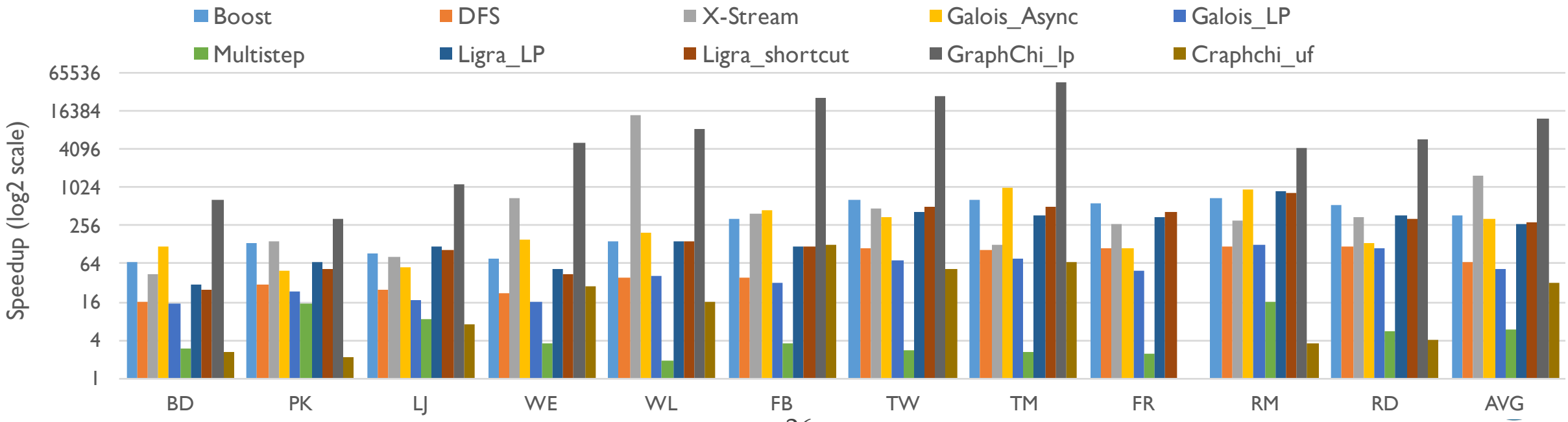
- Hardware
  - A server with two Intel Xeon Gold 6126 CPUs
  - Each CPU has 12 cores
  - Hyper-threading enabled
- Software
  - GCC 4.8.5
  - OpenMP 3.1
  - Optimization level: -O3

Compared Works

- State-of-the-art method for each XCC
  - Multistep for CC [IPDPS'14], iSpan for SCC [SC'18], Slota_BFS for BiCC [HiPC'14]
- Popular graph computation systems
  - Galois [SOSP'13], X-Stream [SOSP'13], GraphChi [OSDI'12], Ligra [PPoPP'13]
- Other works
  - Hong's FW-BW method for SCC [SC'13], Slota_LP for BiCC [HiPC'14]
  - Serial implementations: DFS-based, boost graph library

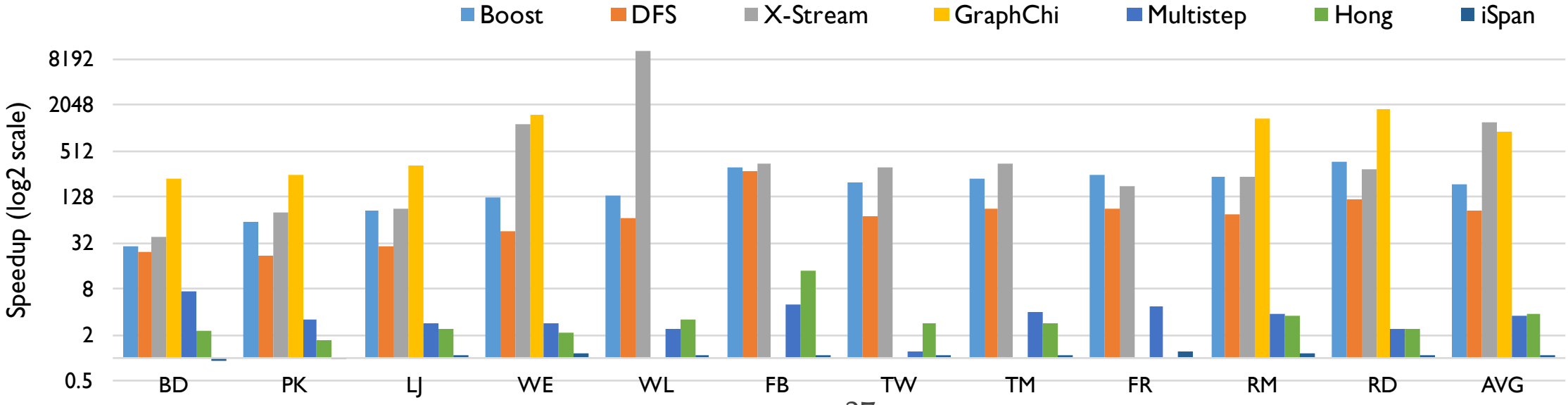
Complete Computation Performance

- (Weakly) Connected Component, (W)CC
  - State-of-the-art: Multistep (5.9x)
  - Four graph systems: Galois (53x), X-Stream (1,548), Ligra (264x), GraphChi (31x)
  - Serial implementations: DFS (67x), Boost (359x)



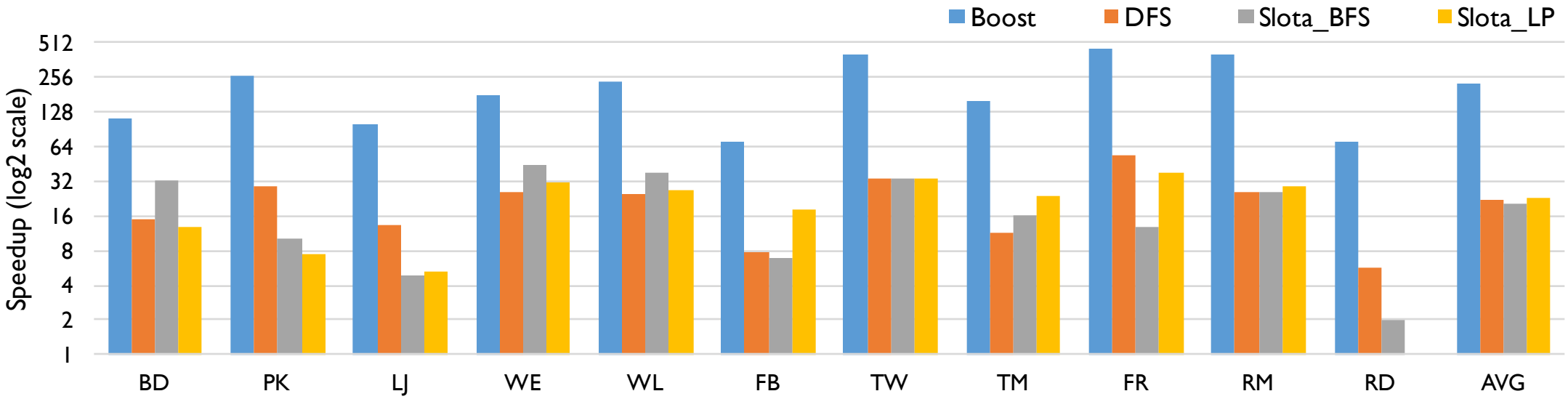
Complete Computation Performance

- Strongly Connected Component (SCC)
  - State-of-the-art: iSpan (1.1x)
  - Two parallel methods: Multistep (3.6x), Hong (3.7x)
  - Two graph systems: X-Stream (1,191x), GraphChi (918x)
  - Serial implementations: Boost (183x), DFS (83x)



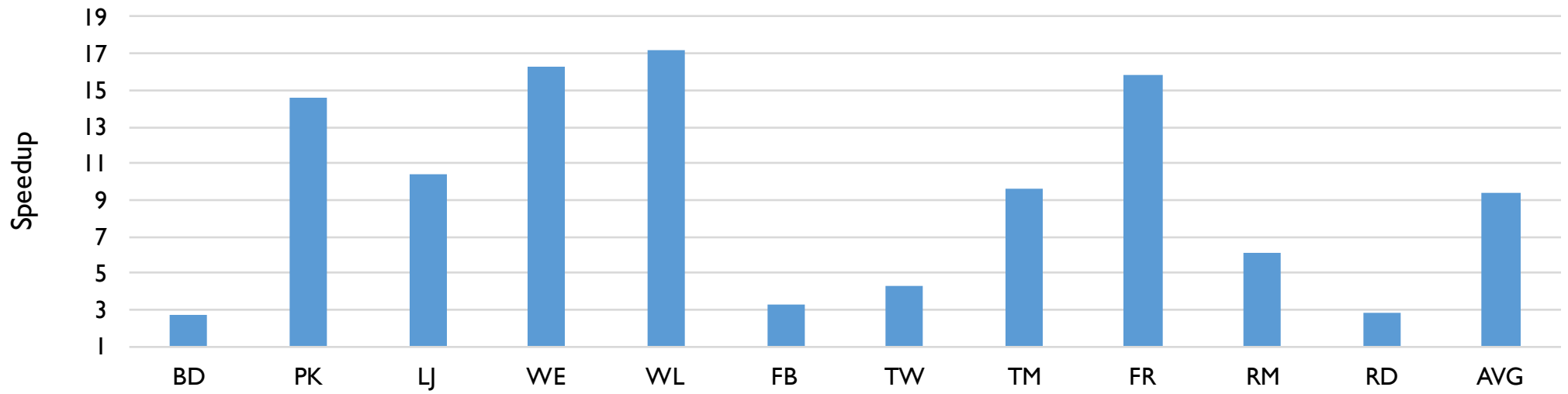
Complete Computation Performance

- BiConnected Component (BiCC)
  - State-of-the-art: Slota_BFS (21x)
  - Slota's label propagation method, Slota_LP (23x)
  - Serial implementations: Boost (223x), DFS (22x)

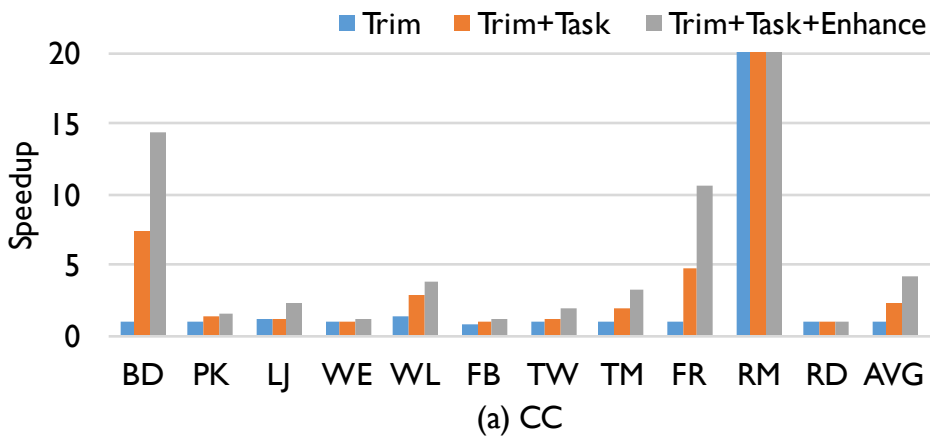


Complete Computation Performance

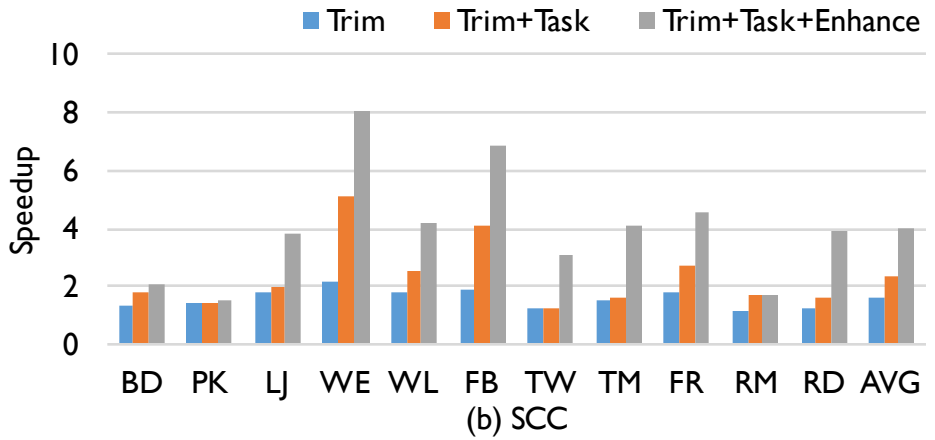
- Bridgeless Connected Component (BgCC)
  - Serial implementations: DFS (23x)



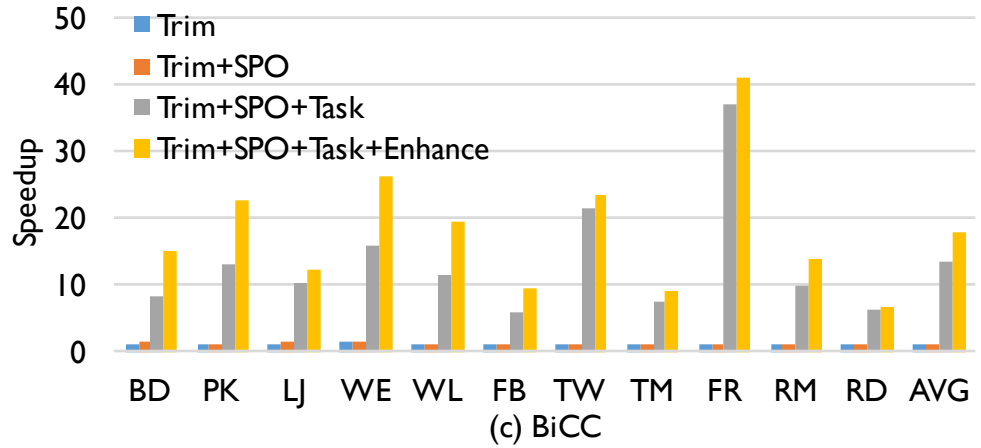
Technique Benefit



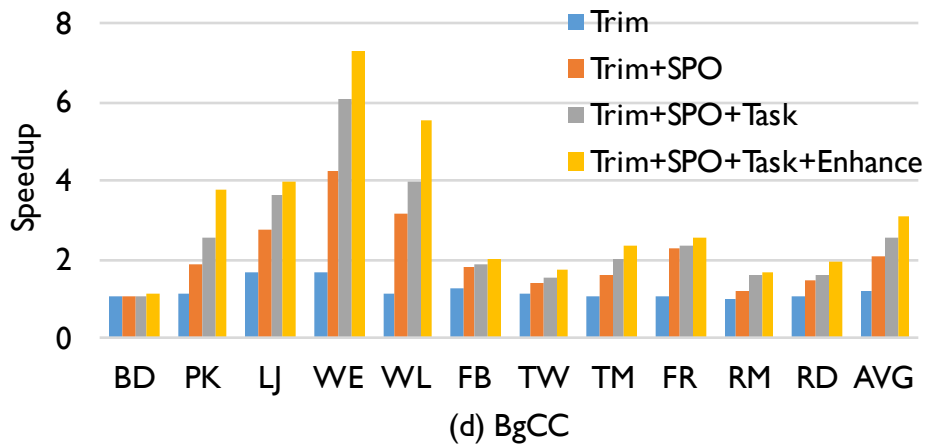
(a) CC



(b) SCC



(c) BiCC



(d) BgCC

Part 4: Conclusion

Conclusion

- New framework specialized for graph connectivity algorithms
- New computation strategies towards partial computation queries
- Adaptive parallel computation strategies with new techniques, e.g., single parent only
- Evaluation: outperform previous works for (W)CC, SCC, BiCC, and BgCC

Thank You & Questions

Source code will be available at https://github.com/iHeartGraph/Aquila

Contact us: yuedeji@gwu.edu, howie@gwu.edu