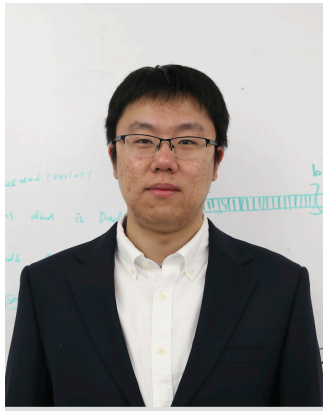
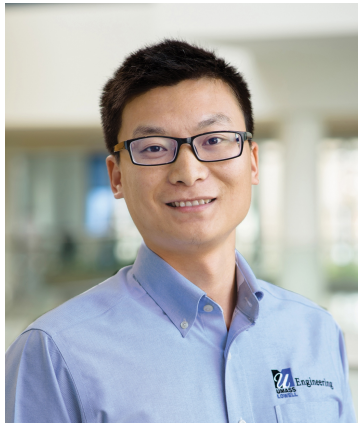
iSpan: Parallel Identification of Strongly Connected Components with Spanning Trees

Yuede Ji Hang Liu H. Howie Huang

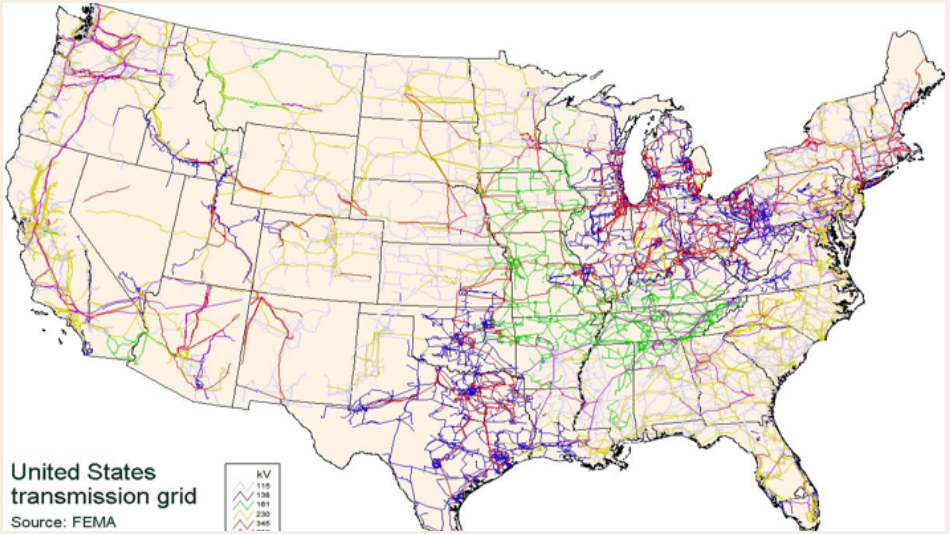




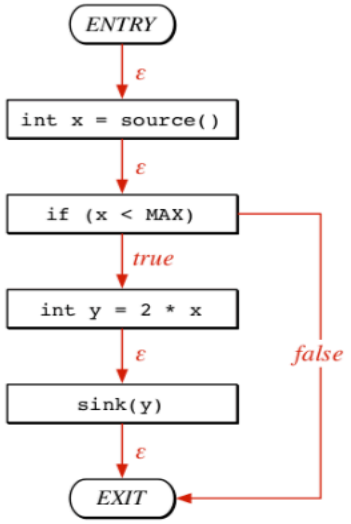
The George Washington University
University of Massachusetts Lowell

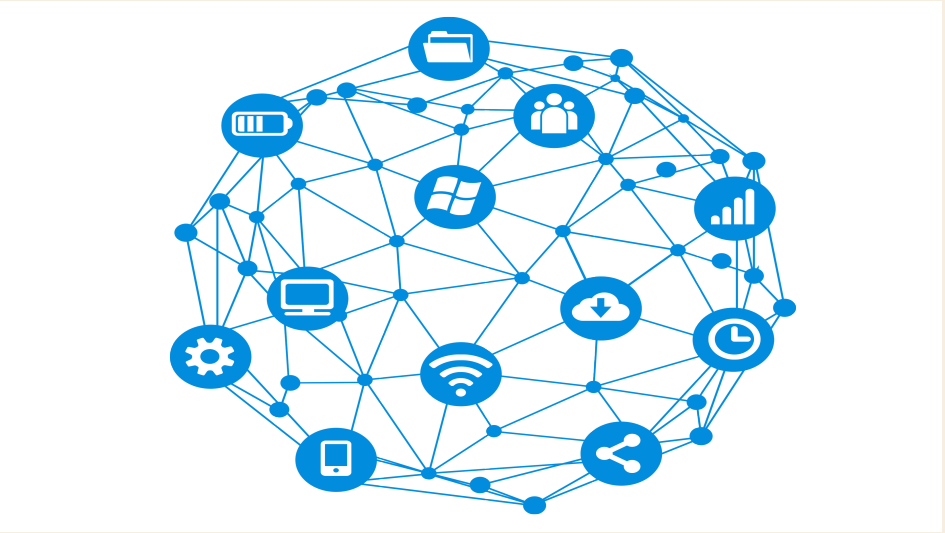
SC18
Dallas, TX | hpc inspires.

Graph is Everywhere



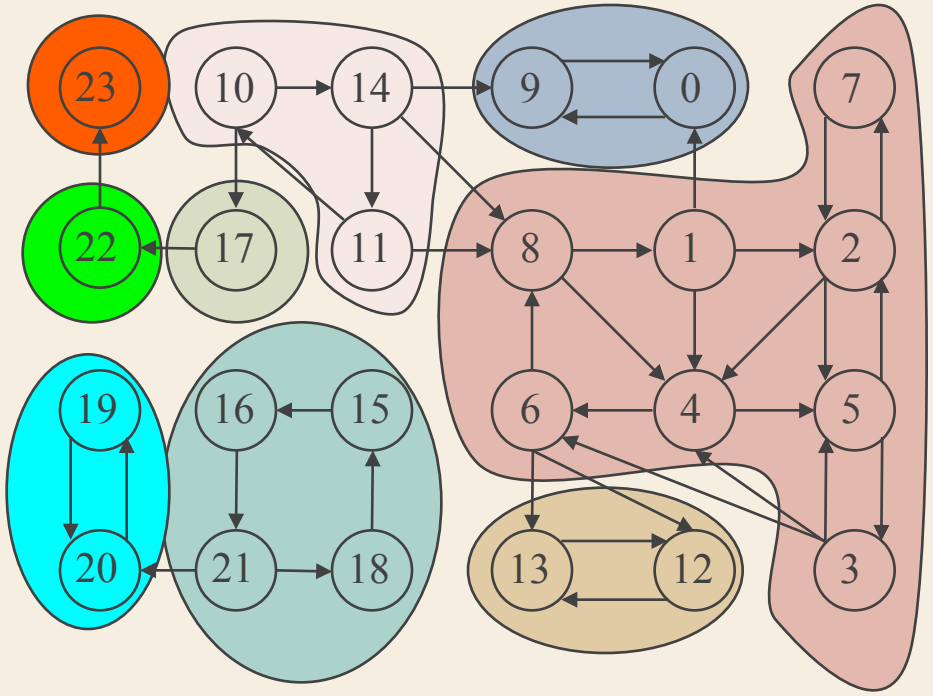

```
void foo()
{
    int x = source();
    if (x < MAX)
    {
        int y = 2 * x;
        sink(y);
    }
}
```





Strongly Connected Component (SCC)

- In a directed graph, an SCC is a maximal subset of the vertices that every vertex has a directed path to all the others
- SCC detection will find all the SCCs in the directed graph



SCC Applications

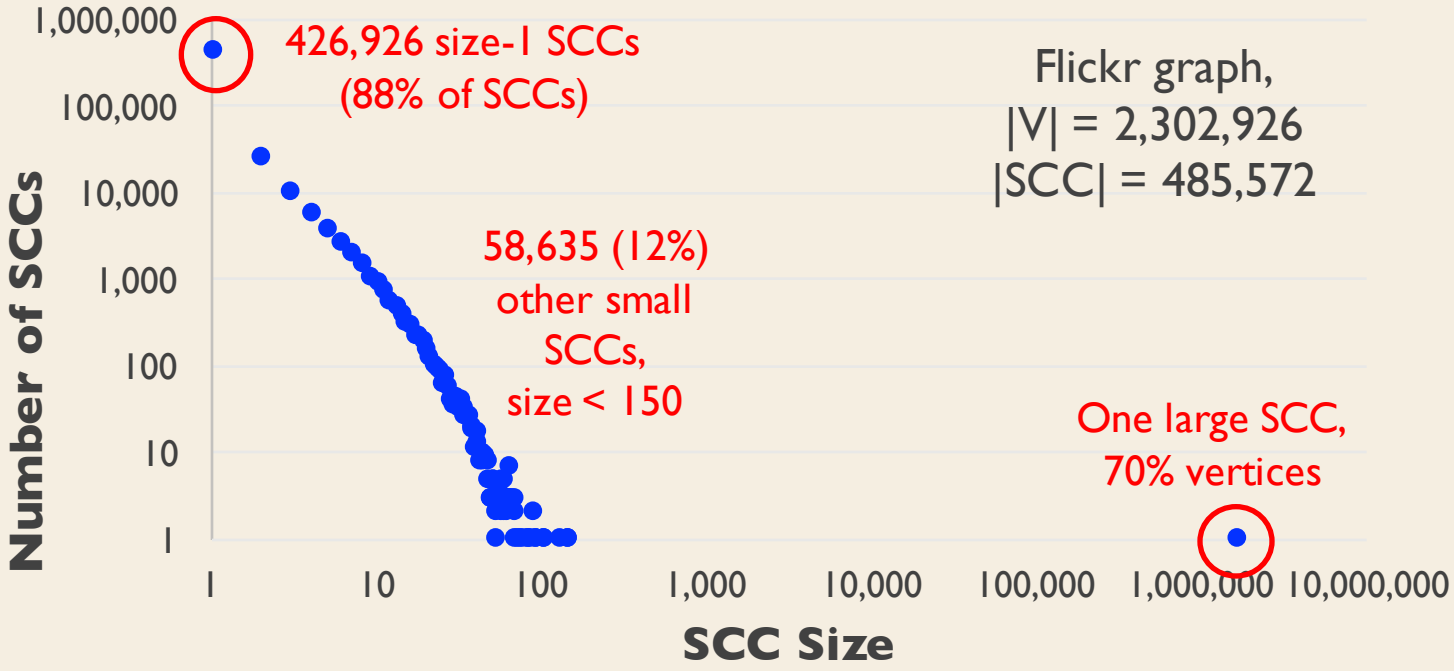
- Topological sort
  - Rank the vertices in a partial order
  - Vertices in one cycle are ranked equally
  - SCC is used to find all the cycles
- Reachability query
  - Whether a vertex can reach another one through a directed path
  - Convert a general directed graph into a directed acyclic graph (DAG) by contracting an SCC as a vertex

Outline

- Background and Observations
- iSpan
- Experiment
- Conclusion

Background: SCC Property

- Power-law property
  - A single large SCC takes the majority of the vertices
  - The rest are a large number of small SCCs, especially size-1 SCC



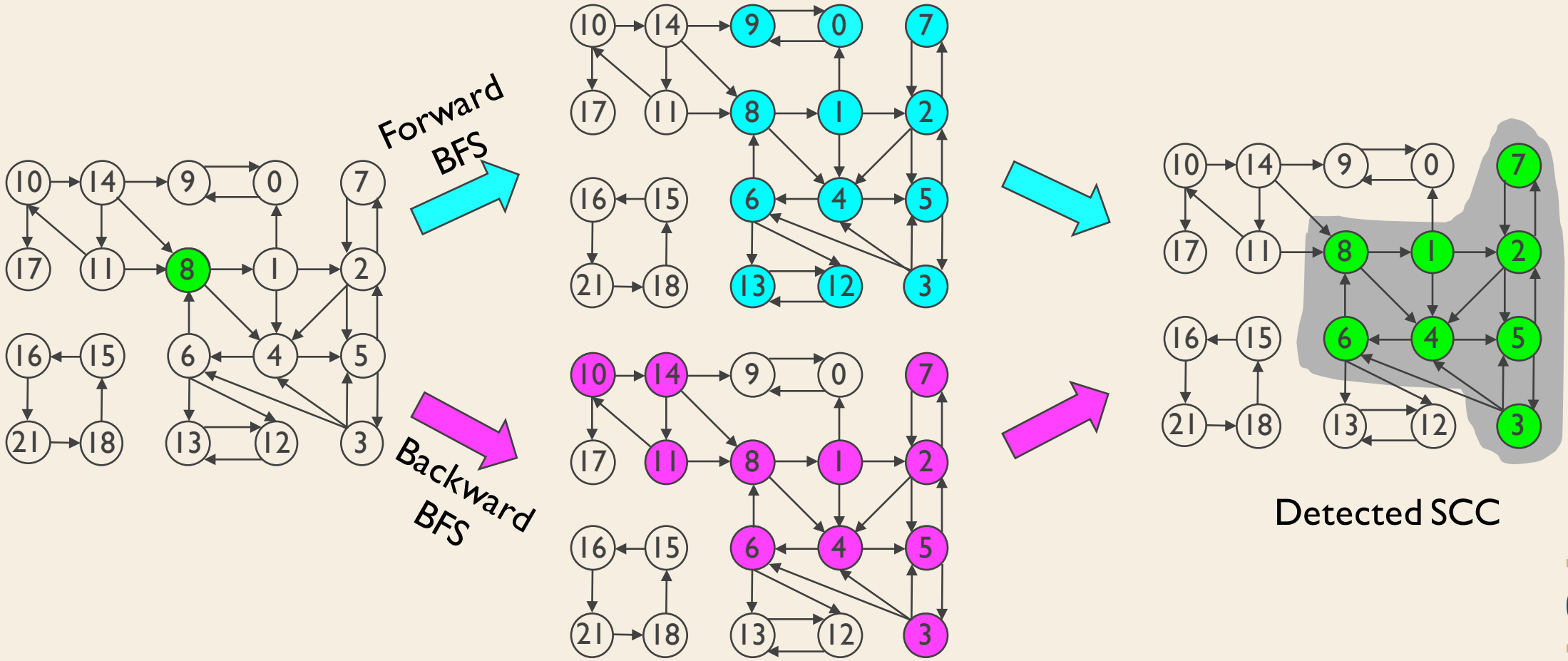
Prior Work

- Large SCC
  - Forward-Backward (FW-BW) algorithm
- Small SCC
  - Applies Trim-1/2 to quickly remove the size-1/2 SCCs
  - Applies FW-BW algorithm again for the rest

BFS FW-BW [SC'13]; Multistep [IPDPS'14]

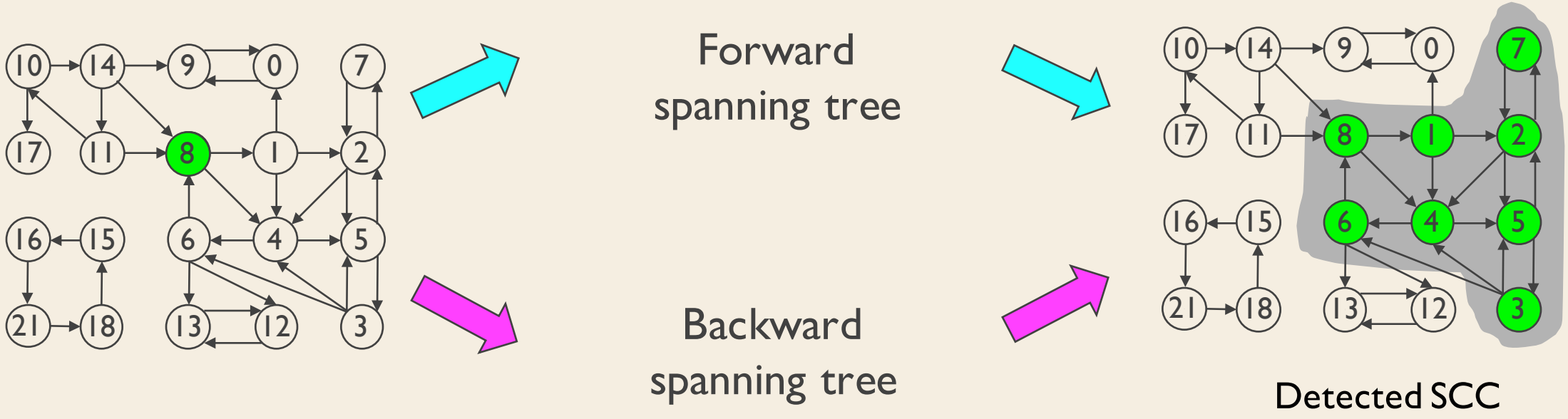
FW-BW Algorithm

Forward-Backward (FW-BW) algorithm takes ~80% of total runtime



Observation

- The FW-BW can be done by any spanning tree construction algorithm.
  - BFS produces the correct levels, which is not only unnecessary for SCC, but also requires synchronization and thus limits the performance



iSpan Techniques

- Relaxed synchronization (Rsync)
- Fast spanning tree construction method
- Scale to multiple machines

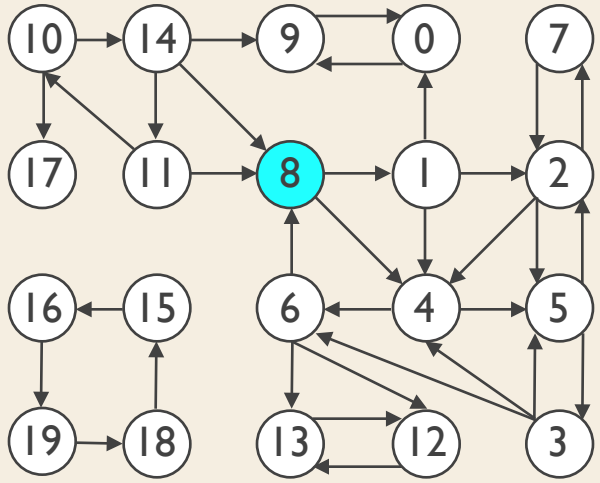
Relaxed Synchronization (Rsync)

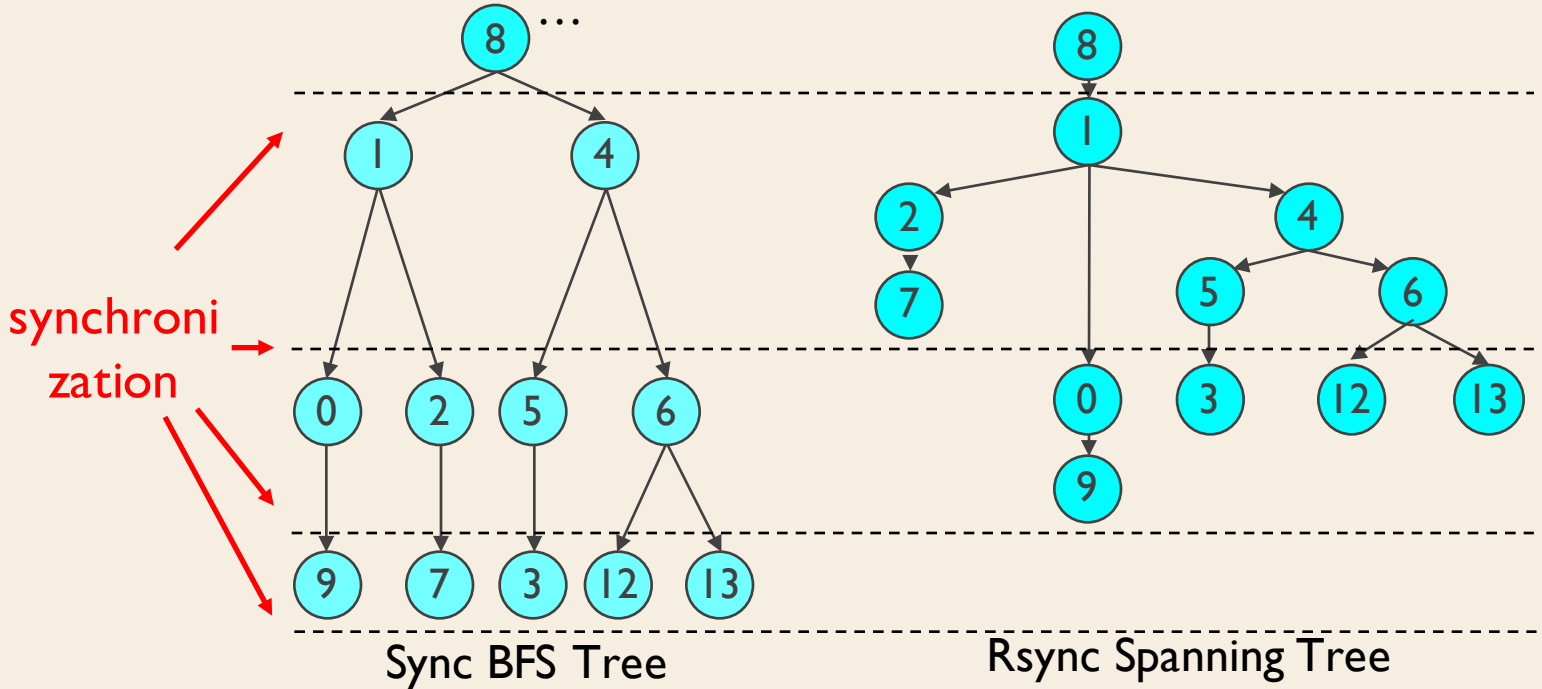
Current Sync BFS (bottom-up)

```
1 foreach unvisited vertex v in parallel do
2     foreach vertex w in InNeighbor(v) do
3         if visit[w] == level then
4             visit[v] = level + 1;
5             break;
6 barrier();
…
```

Rsync

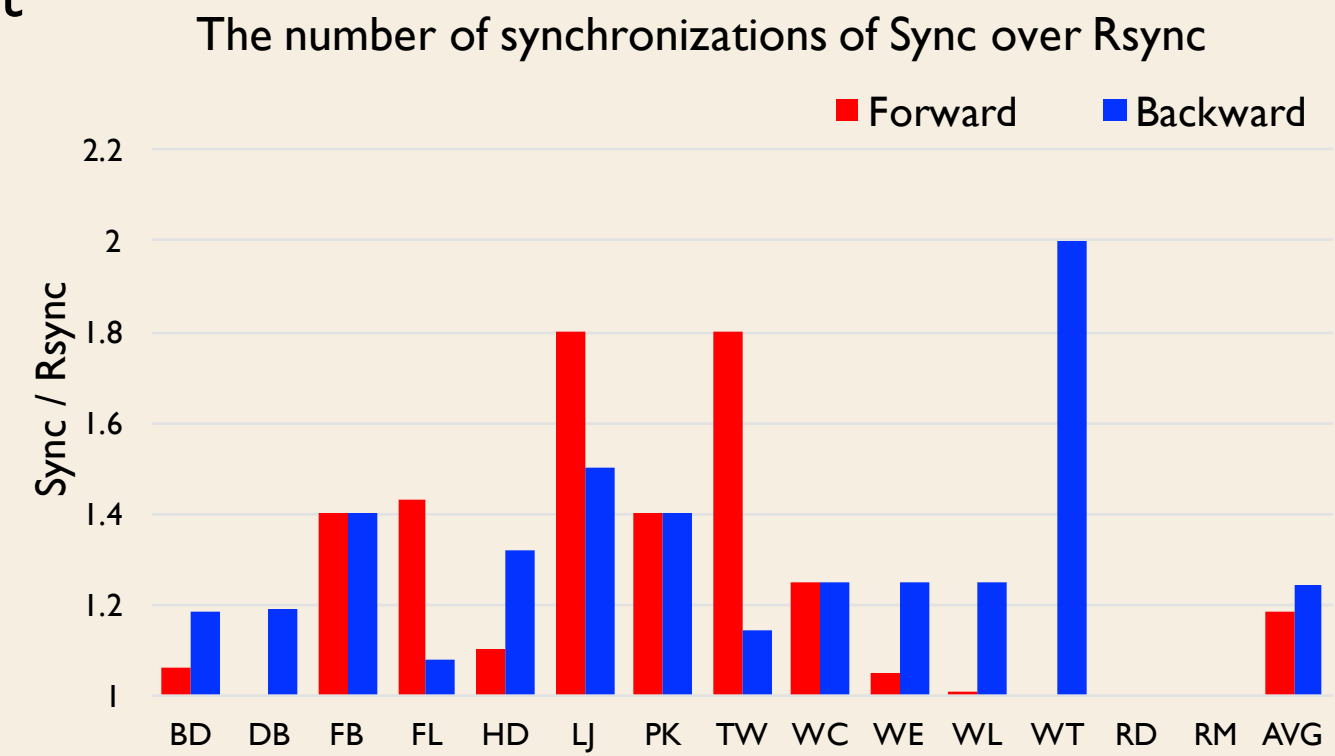
```
1 foreach unvisited vertex v in parallel do
2     foreach vertex w in InNeighbor(v) do
3         if visit[w] is visited then
4             visit[v] = visited;
5             break;
6 barrier();
…
```





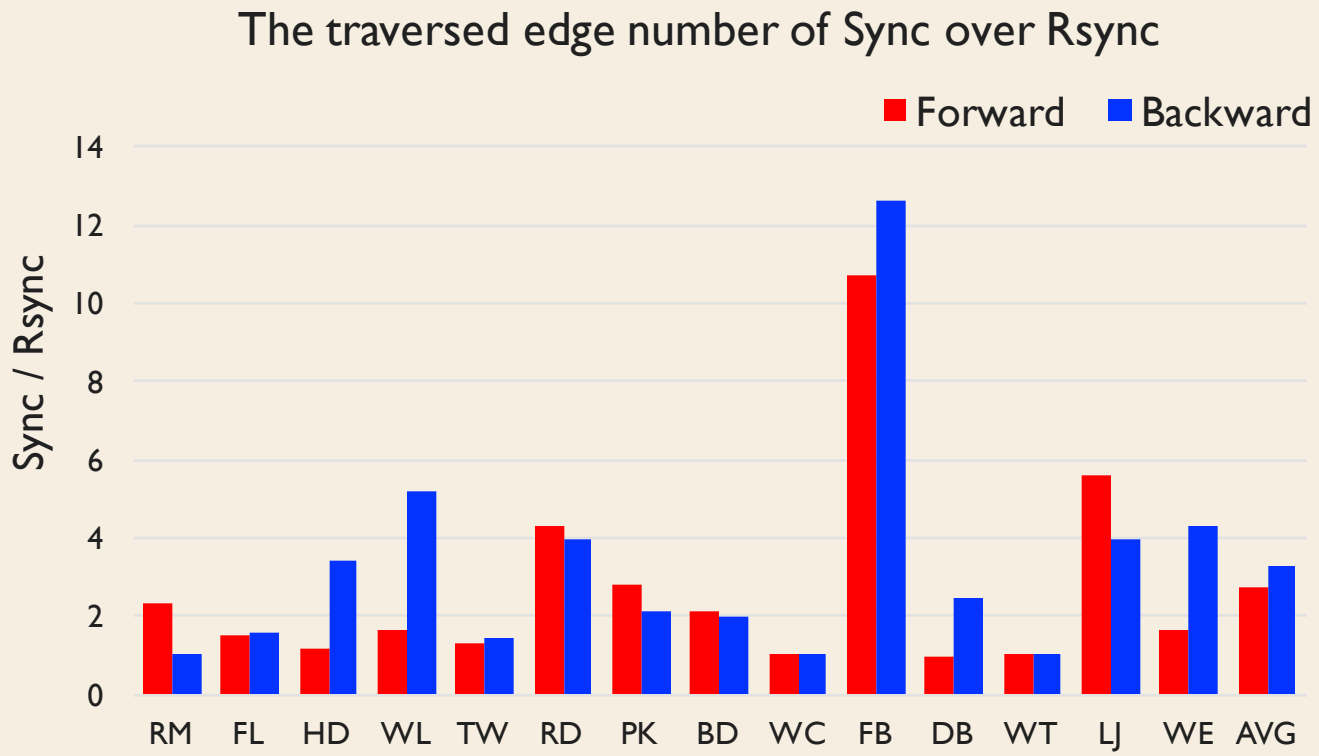
Rsync Benefits

- The vertex that should be visited at later level can be visited earlier
- Benefit #1: Reduce the number of synchronizations required
  - 18% for forward traversal
  - 25% for backward traversal



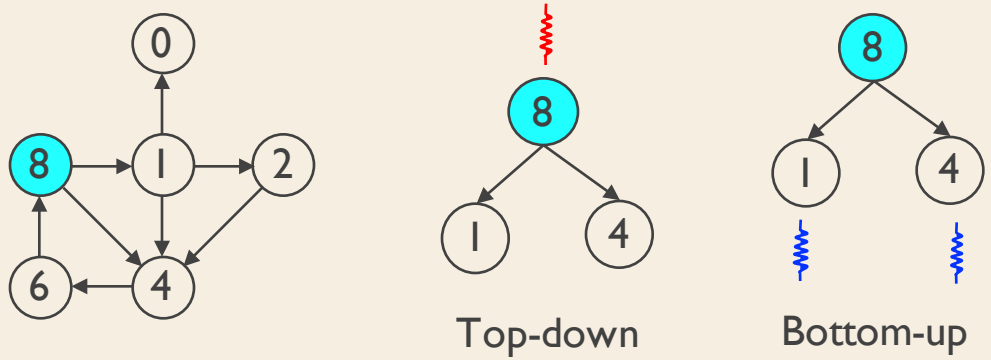
Rsync Benefits

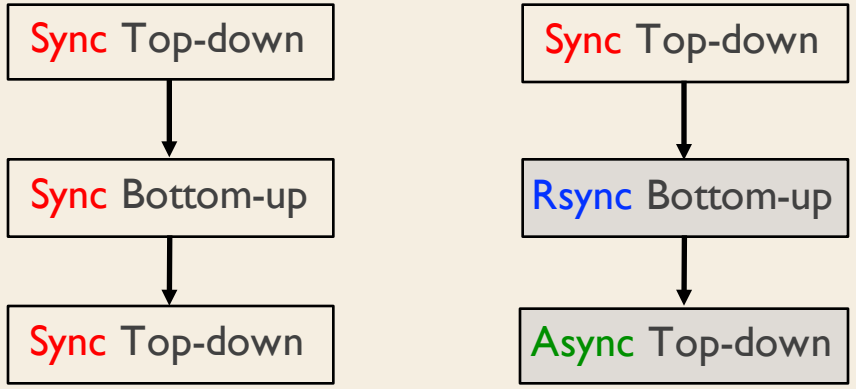
- A vertex can be early terminated by newly visited vertices
- Benefit #2: Reduce the number of traversed edges
  - 2.7x for forward traversal
  - 3.3x for backward traversal



Fast Spanning Tree Construction

- Direction-optimizing BFS [SC'12]
  - Top-down - visited vertex find unvisited
  - Bottom-up - unvisited vertex find visited
- Switch based on the workload
  - Starts from top-down
  - Switches to bottom-up in the middle (majority of the time)
  - Switches back to top-down



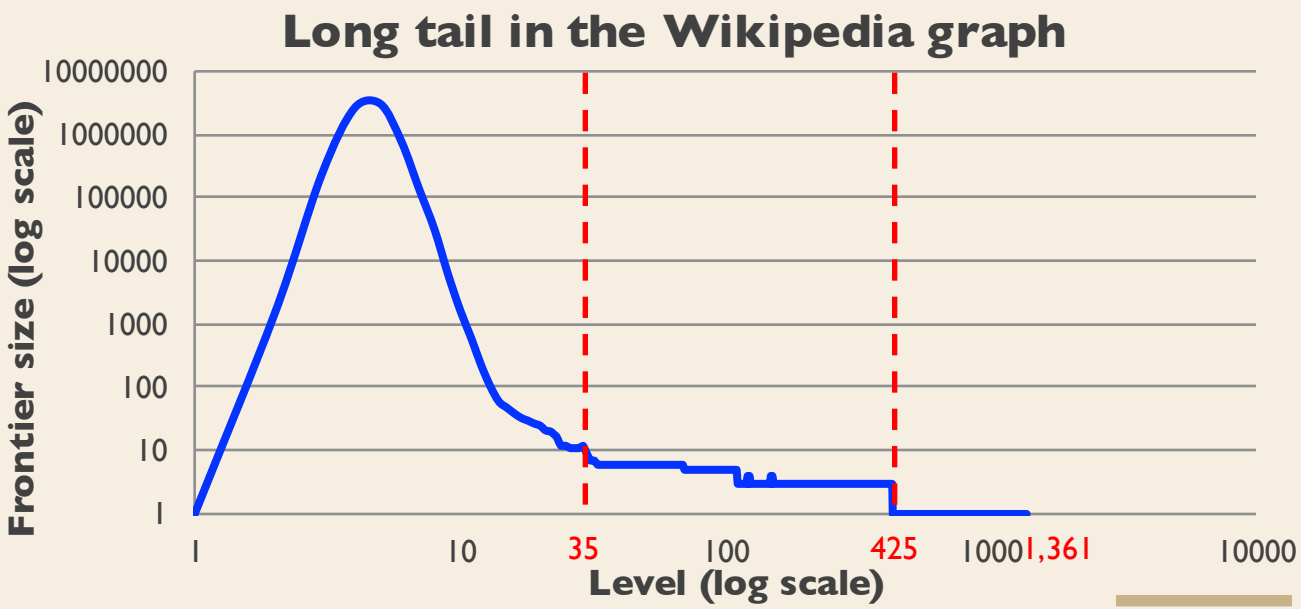


(a) Direction-optimizing BFS

(b) iSpan spanning tree construction

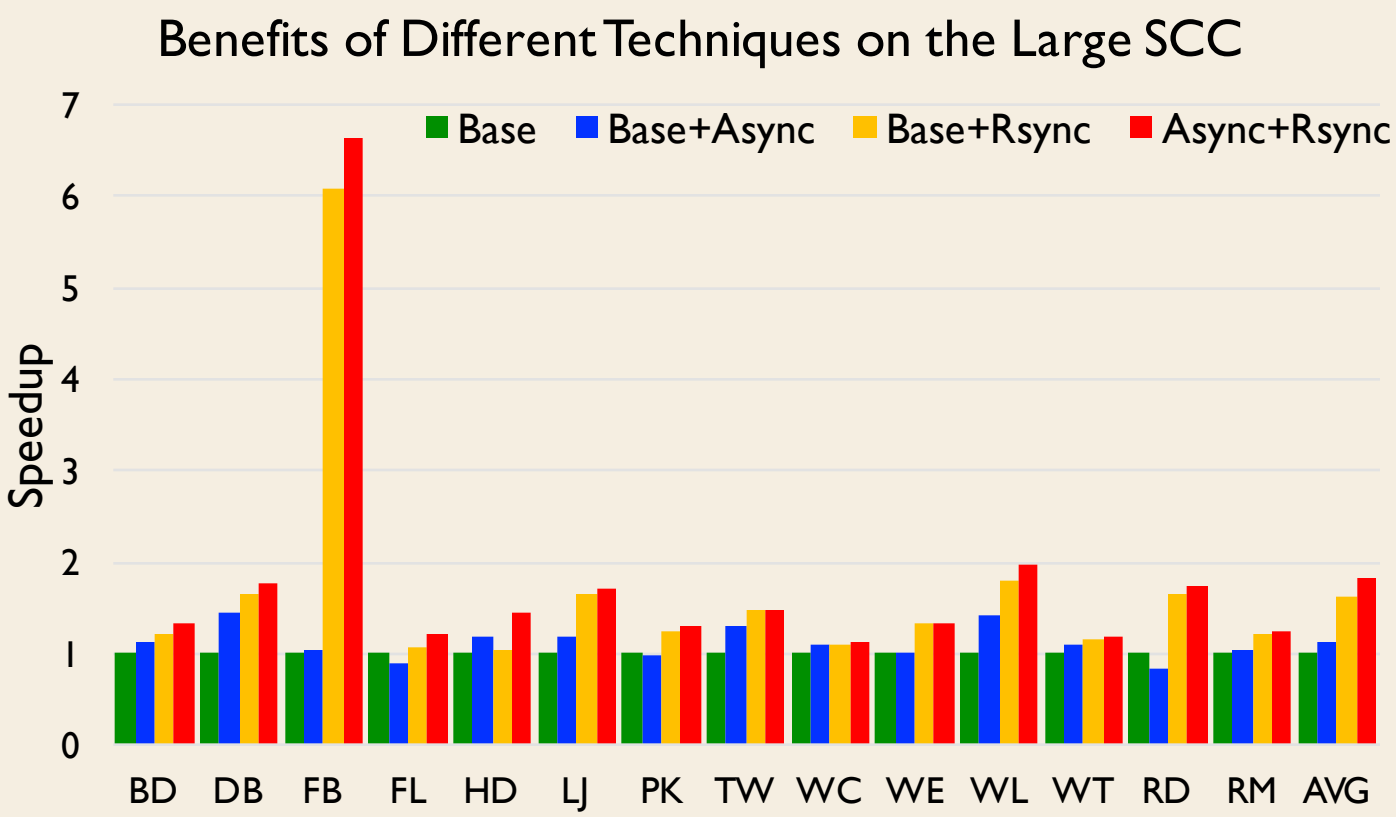
Async Top-down

- Asynchronous traversal (**Async**)
  - Each thread takes equal partition at the beginning
  - Each thread works on its own partition till the end
    - does not incur any synchronization
- Long tail frontier distribution
  - Frontier is the number of visited vertices in one level
- Comparison (switch at 35-th level)
  - Sync top-down requires 1,326 levels (thus synchronizations) vs Async top-down 1 level



Benefits of Fast Spanning Tree Construction

- Large SCC (FW-BW)
  - Speedup over our optimized direction-optimizing BFS
  - 1.1x with Async top-down,
  - 1.6x with Rsync bottom-up
  - 1.8x with Async + Rsync



Distribute iSpan to Multiple Machines

- Challenge:
  - High communication cost
- Vertex-centric 1-D partition
- **Large SCC**
  - Bitwise status compression
    - Visited, unvisited, newly visited
    - 2 bits for one status, thus |V|/8 bytes
  - Frontier queue (FQ)
    - Only the newly visited vertices
  - Hybrid queue - min(|V|/8, |FQ|)
- **Small SCCs**
  - Compact the graph for the remaining vertices and edges
    - Average 2.1% vertices and 0.5% edges left
  - Replicate this graph on every machine
    - No communication is required

GW

Outline

- Background and Observations
- iSpan
- Experiment
- Conclusion

GW

Experiments

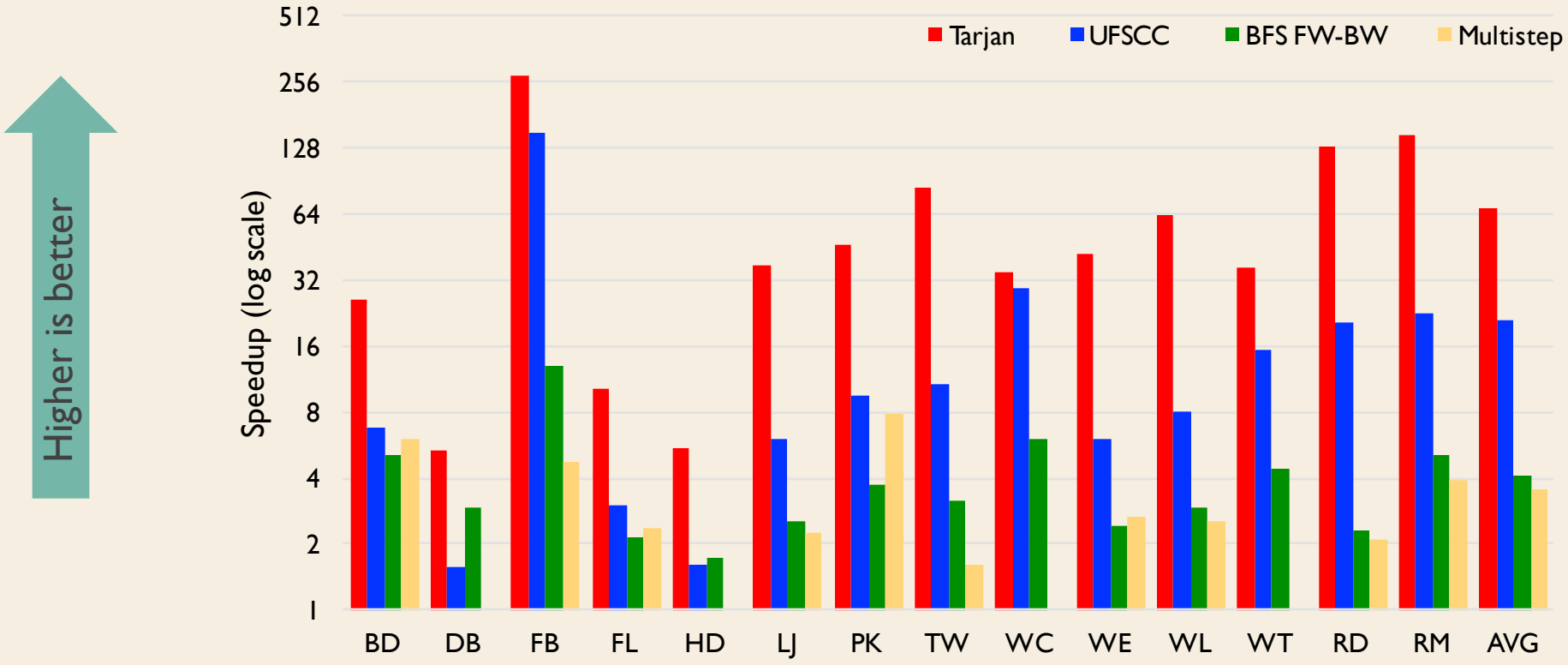
- Hardware
  - A single server with two Intel Xeon E5-2683 (2.00 GHz) CPUs with 56 hardware threads
  - 32 nodes on university clusters
- Software
  - GCC 4.8.5
  - OpenMP 3.1
  - OpenMPI 1.8
- Report average of 10 runs
- Use the same configuration for comparison

Graph Benchmarks

- 17 graphs
  - 14 real-world graphs
    - Konect, University of Koblenz-Landau
    - SNAP, Stanford
  - 3 synthetic graphs
    - R-MAT
    - Kron

Graph (Abbr.)	|Vertex|	|Edge|	# of SCC	Large SCC Size
Baidu (BD)	2,141,301	17,794,839	1,503,004	609,905
Dbpedia (DB)	3,966,925	13,820,853	3,636,316	178,593
Flickr (FL)	2,302,926	33,140,017	485,572	1,605,184
Hudong (HD)	2,452,716	18,854,882	2,189,120	185,668
Livejournal (LJ)	4,847,572	68,475,391	971,233	3,828,682
Pokec (PK)	1,632,804	30,622,564	325,893	1,304,537
Wiki_comm (WC)	2,394,386	5,021,410	2,281,880	111,881
Wikipedia_en (WE)	18,268,993	172,183,984	14,459,547	3,796,073
Wikipedia_link (WL)	11,196,008	340,309,824	4,266,559	6,916,926
Wiki_talk_en (WT)	2,987,536	24,981,161	2,736,716	249,610
Twitter_www (TW)	41,652,231	1,468,365,182	8,044,729	33,479,734
Facebook (FB)	96,079,682	679,728,426	93,892,292	2,186,877
Twitter_mpi (TM)	52,579,683	1,963,263,821	12,407,622	40,012,384
Friendster (FR)	68,349,466	2,586,147,869	18,697,703	48,928,140
Random (RD)	4,000,001	256,000,000	2	4,000,000
R-MAT (RM)	3,999,984	256,000,000	2,105,950	1,894,035
Kron_30 (KR)	1,073,741,824	17,179,869,184	708,654,254	364,818,982

Speedup of iSpan over Related Work

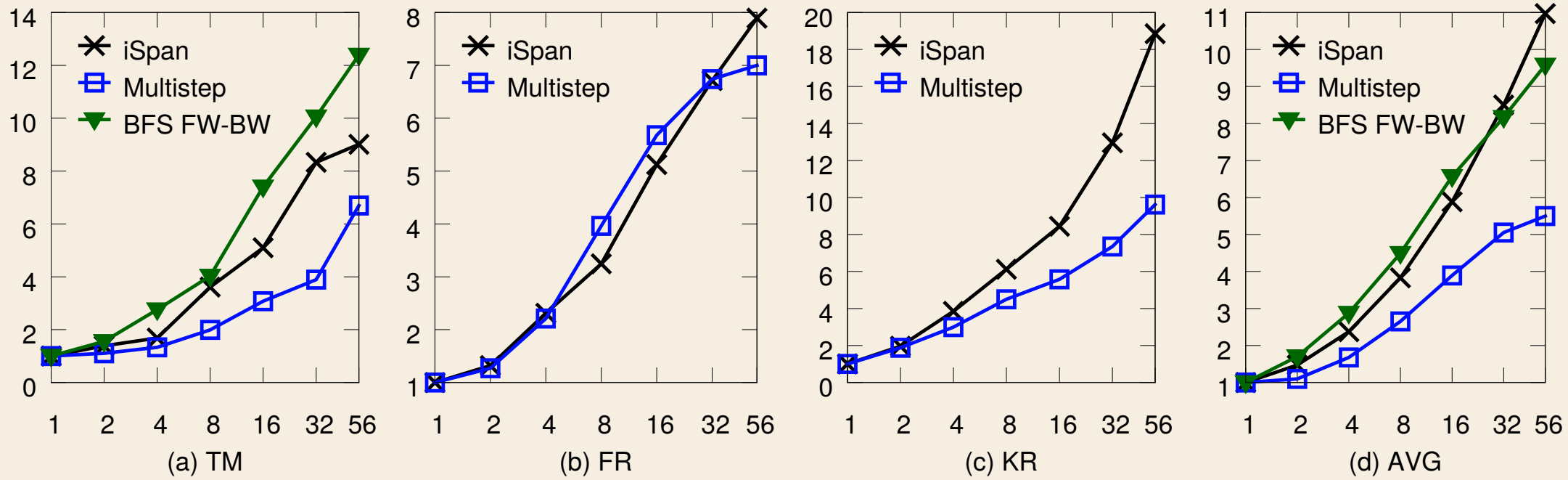


- DFS style work: 67x over Tarjan [SICOMP'72]; 21x over UFSCC [PPoPP'16];
- BFS style work: 4.1x over BFS FW-BW [SC'13]; 3.6x over Multistep [IPDPS'14]

GW

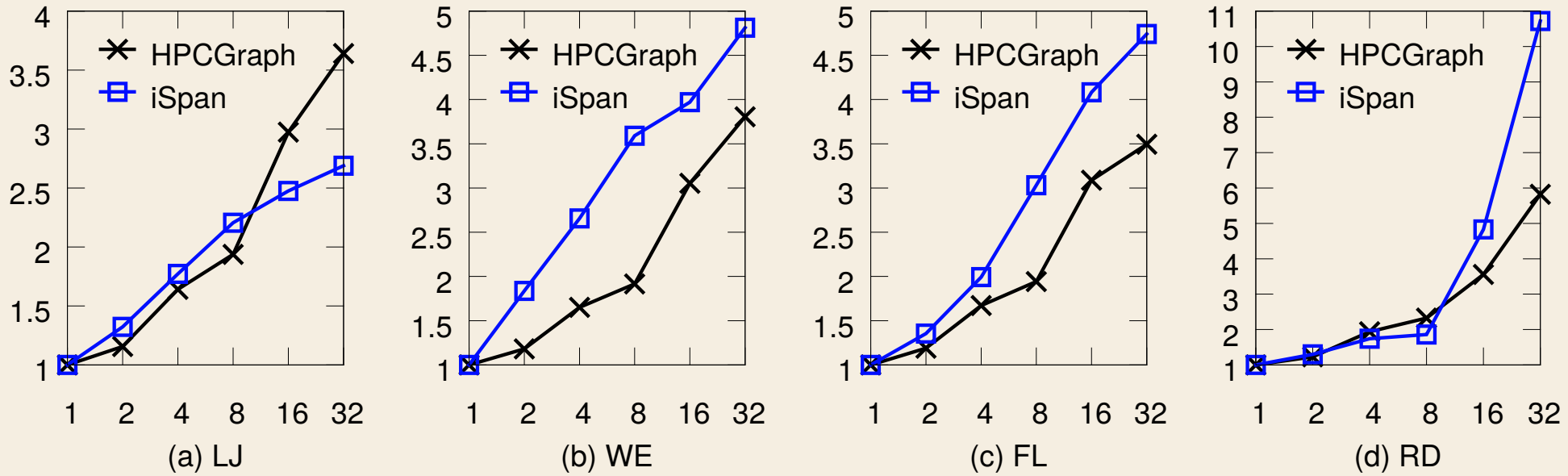
Scale to Multiple Threads

- iSpan achieves average 11x speedup over baseline, Multistep 9.6x, BFS FW-BW 5.5x
- For billion vertex graph, iSpan gets 19x, Multistep 10x



Scale to Multiple Machines

- Use four representative graphs, LJ, FL (social graph), WE (web graph), RD (synthetic graph)
- Compare with HPCGraph [IPDPS'16]
  - Better scalability for WE, FL, and RD
  - Worse scalability for LJ - faster single node runtime, 8x



Conclusion

- New insight: spanning tree is sufficient for SCC
- New relaxed synchronization approach
- New spanning tree construction method
- Results:
  - Outperform current state-of-the-art DFS and BFS- based methods by average 18x and 4x
  - Scale to billion vertex graph on tens of machines

Thank You

Check out our graph code repository at github.com/iHeartGraph/

GW