THE MORE THE MERRIER: EFFICIENT MULTI-SOURCE GRAPH TRAVERSAL

Manuel Then*, Moritz Kaufmann*, Fernando Chirigati†, Tuan-Anh Hoang-Vu† ,
Kien Pham†, Huy T. Vo†, Alfons Kemper*, Thomas Neumann*
*Technische Universität München, †New York University
Published at VLDB 2015

Presented by Victor A. Ying
6.886 – February 26, 2018

Slides are from https://db.in.tum.de/~then/data/2015-MS-BFS-Presentation.pdf?lang=en

Background

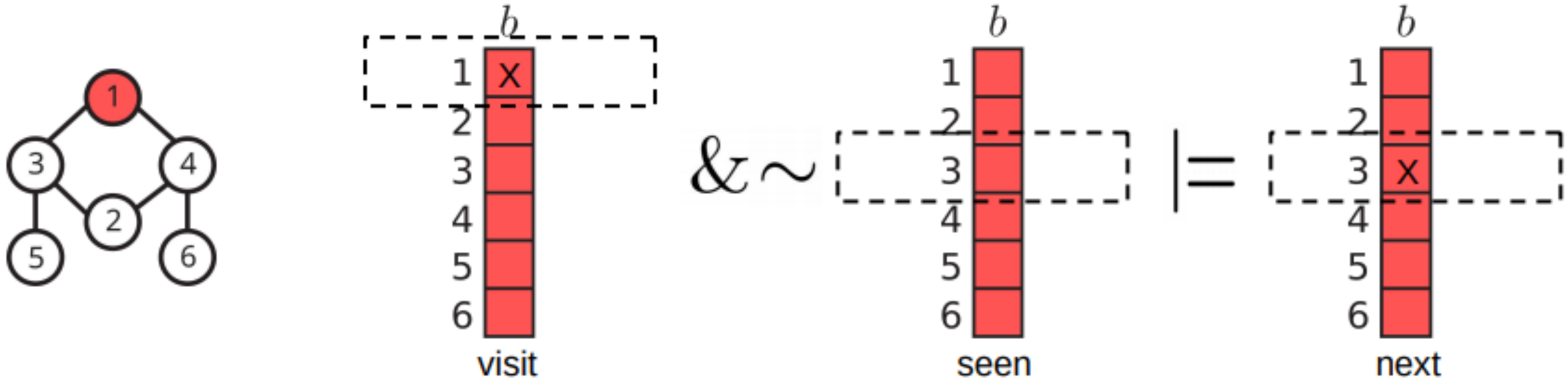
- Algorithms require many BFSs on one graph
  - *E.g., compute centrality metrics across graph*
- Prior work: parallel BFS
  - *Barrier synchronization between levels*
- Graph traversals have poor cache behavior
  - *Read a single random bit when traversing each edge*
- Small-world phenomenon: graphs have low diameter

Multi-Source BFS

- Concurrently **run many independent BFS traversals** on the same graph
  - 100s of BFSs on a single CPU core

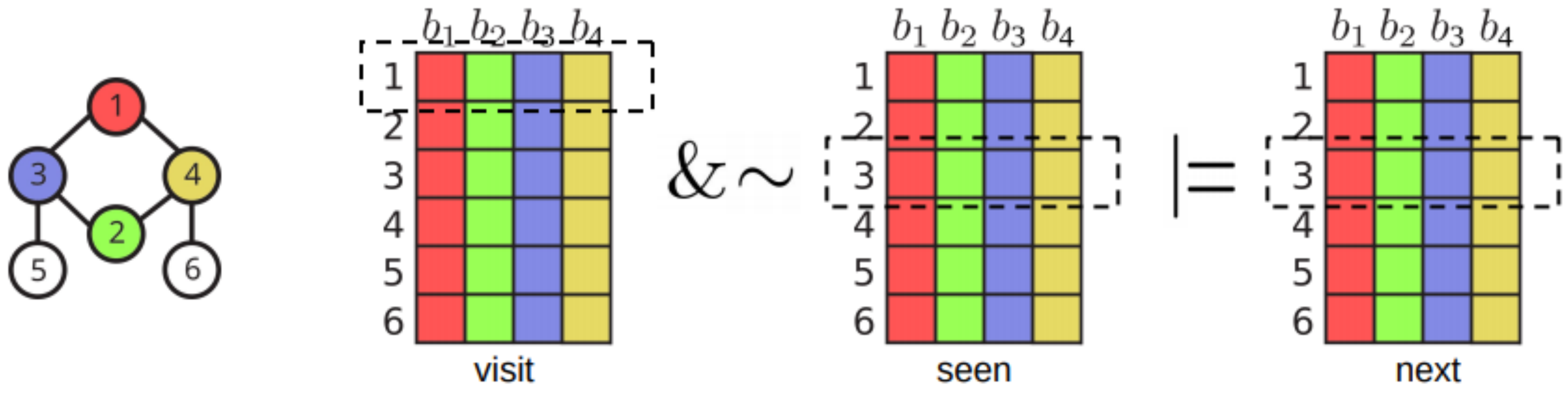
Multi-Source BFS

- Concurrently **run many independent BFS traversals** on the same graph
  - 100s of BFSs on a single CPU core



Multi-Source BFS

- Concurrently **run many independent BFS traversals** on the same graph
  - 100s of BFSs on a single CPU core
- Store concurrent **BFSs state as 3 bitsets** per vertex



- Represent **BFS traversal as SIMD bit operations** on these bitsets

Multi-Source BFS (MS-BFS)

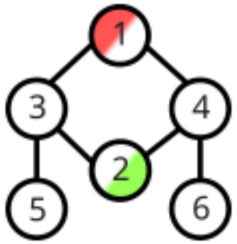
One round:

$$
\begin{aligned}
&\textbf{for } i = 1, \ldots, N \\
&\quad \textbf{if } visit[v_i] = \mathbb{B}_\varnothing\text{: } \textbf{skip} \\
&\quad \textbf{for each } n \in neighbors[v_i] \\
&\quad\quad \mathbb{D} \leftarrow visit[v_i]\ \&\ {\sim}seen[n] \\
&\quad\quad \textbf{if } \mathbb{D} \neq \mathbb{B}_\varnothing \\
&\quad\quad\quad visitNext[n] \leftarrow visitNext[n] \mid \mathbb{D} \\
&\quad\quad\quad seen[n] \leftarrow seen[n] \mid \mathbb{D}
\end{aligned}
$$

- Given frontiers, compute next frontiers
- By traversing each edge (at most) once in each direction.

Multi-Source BFS - Example

Initial



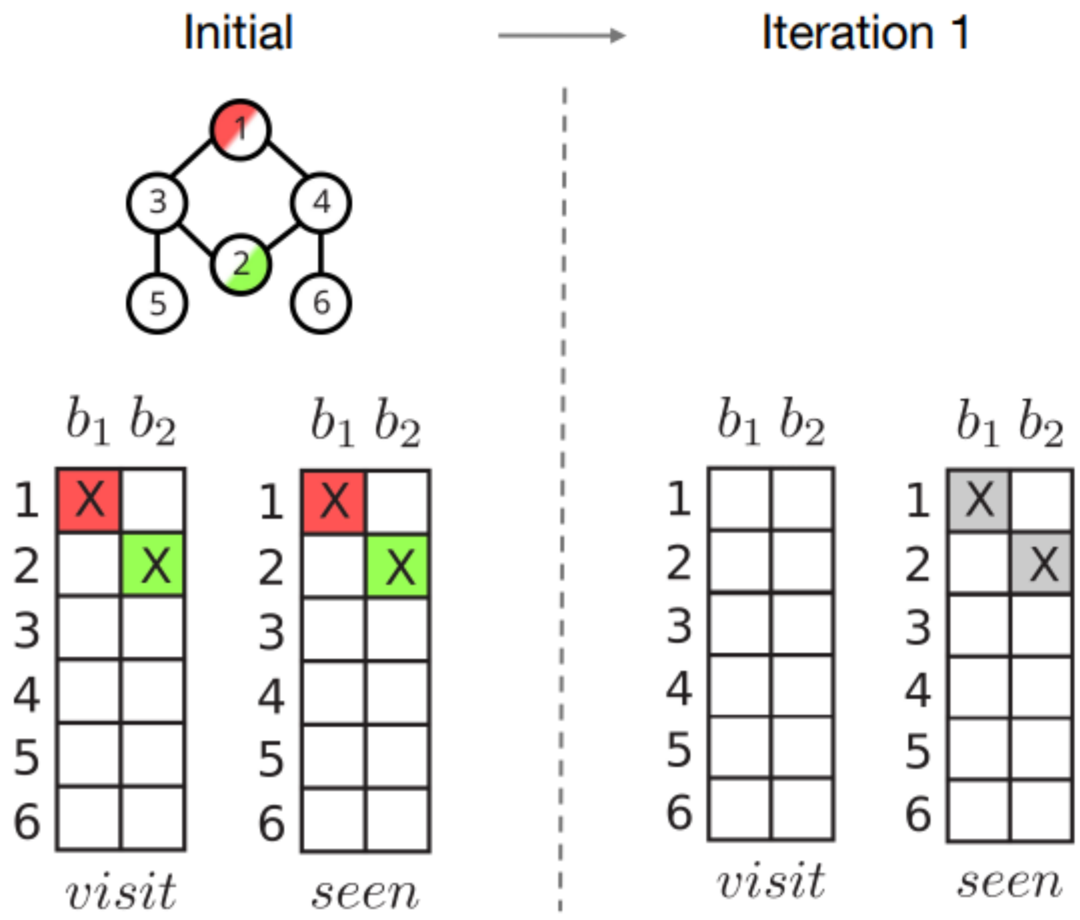
	b_1	b_2
1	X	
2		X
3		
4		
5		
6		

visit

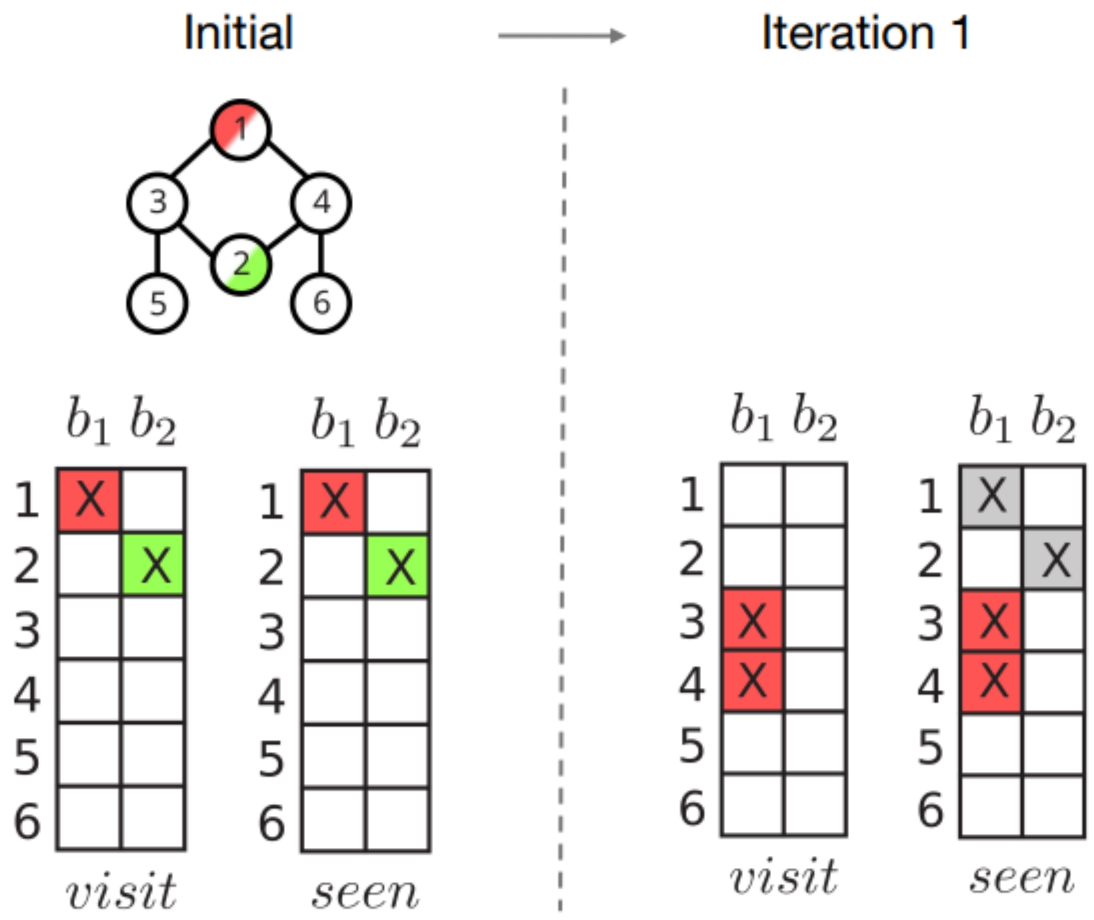
	b_1	b_2
1	X	
2		X
3		
4		
5		
6		

seen

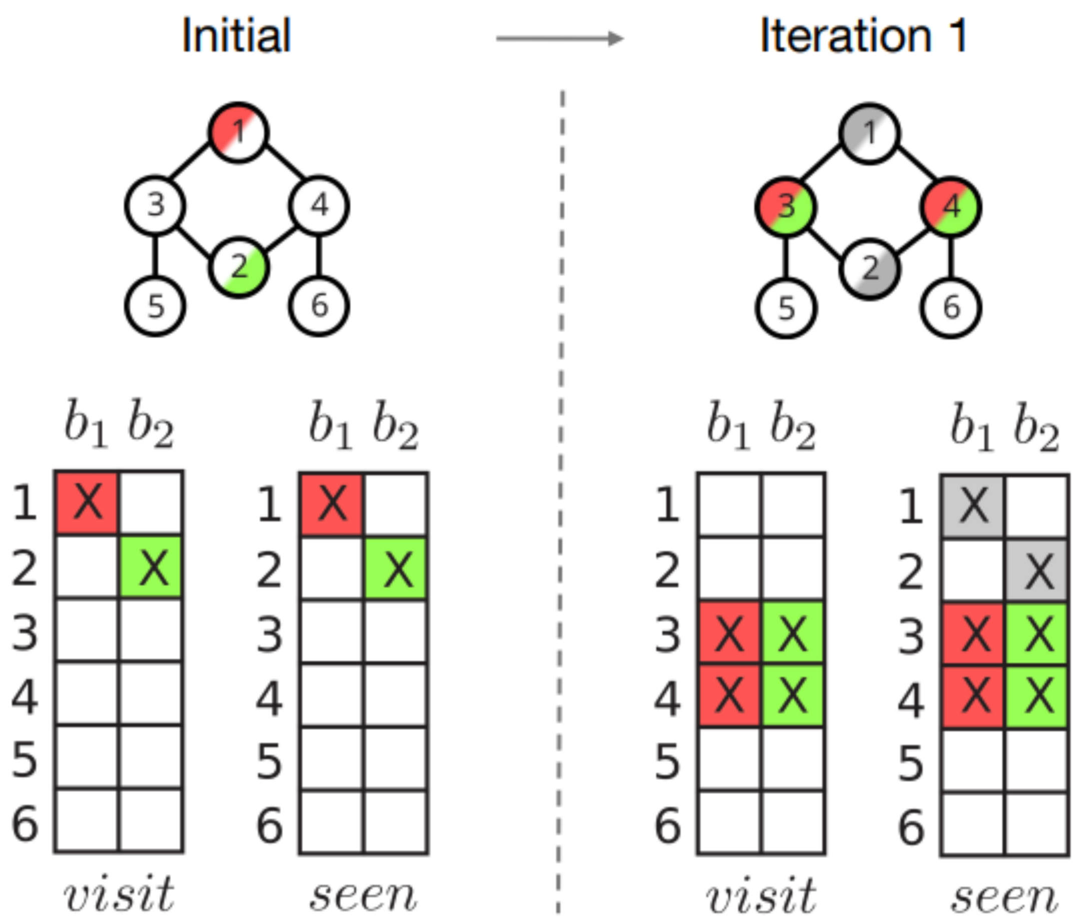
Multi-Source BFS - Example



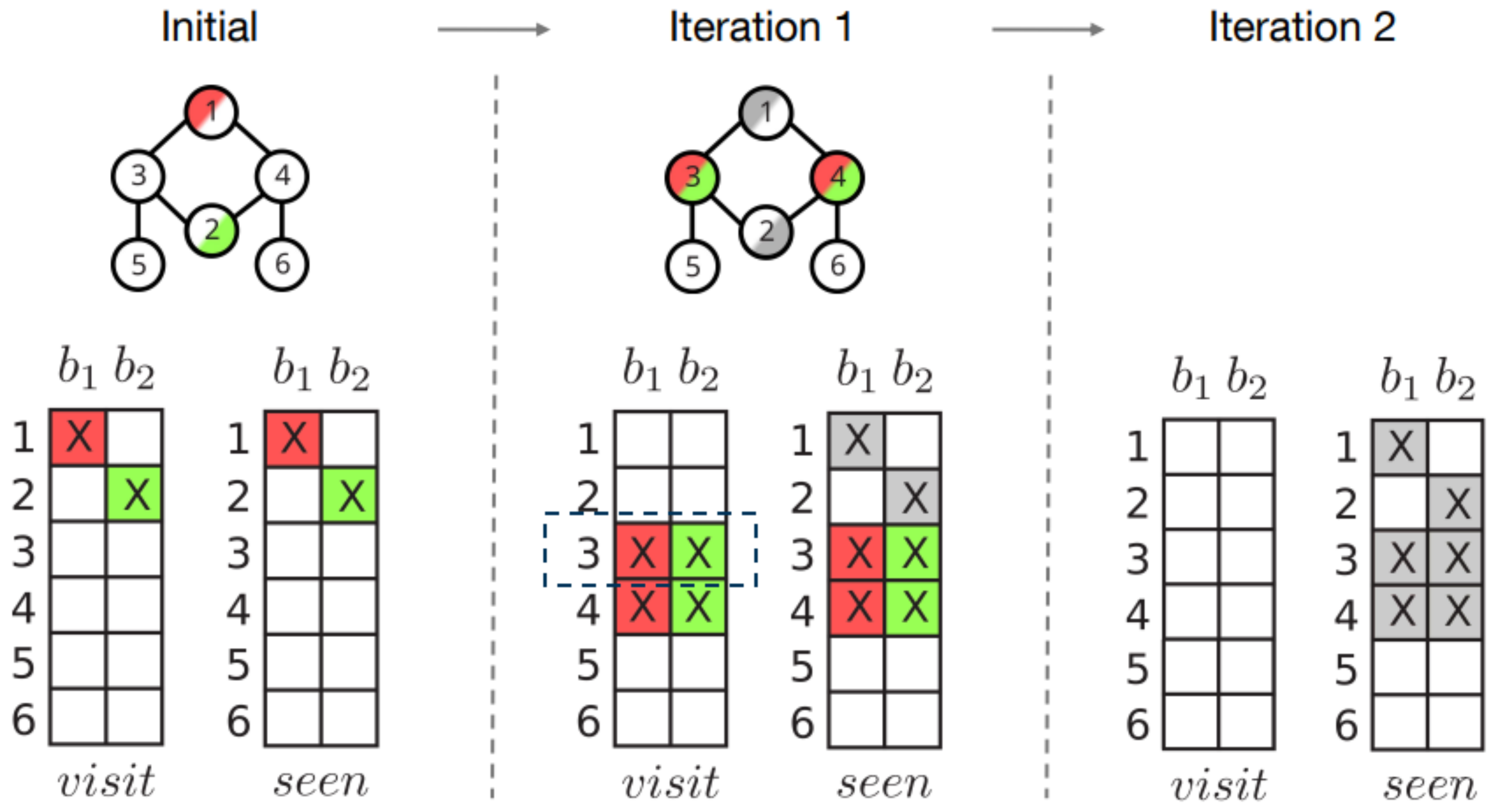
Multi-Source BFS - Example



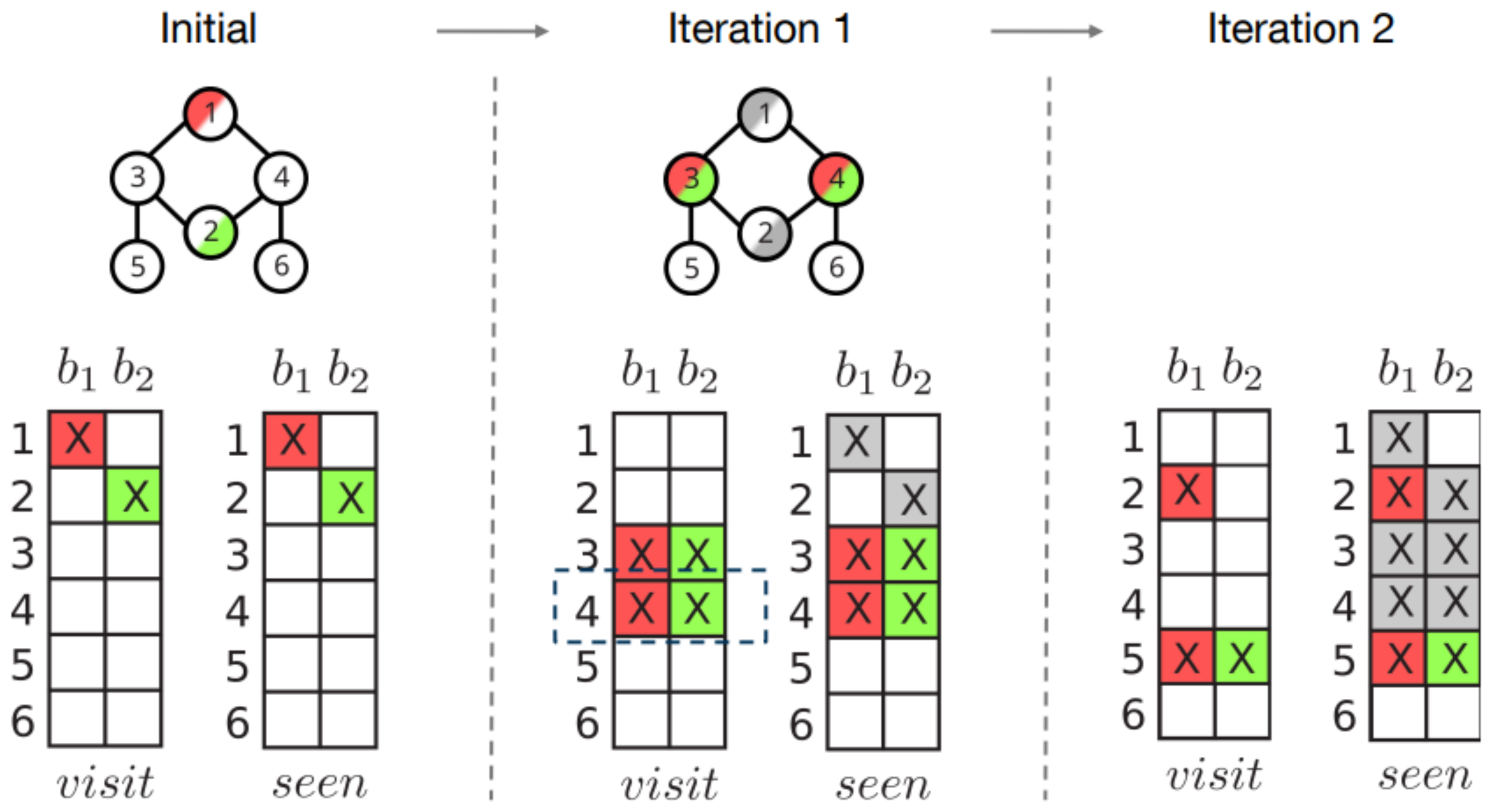
Multi-Source BFS - Example



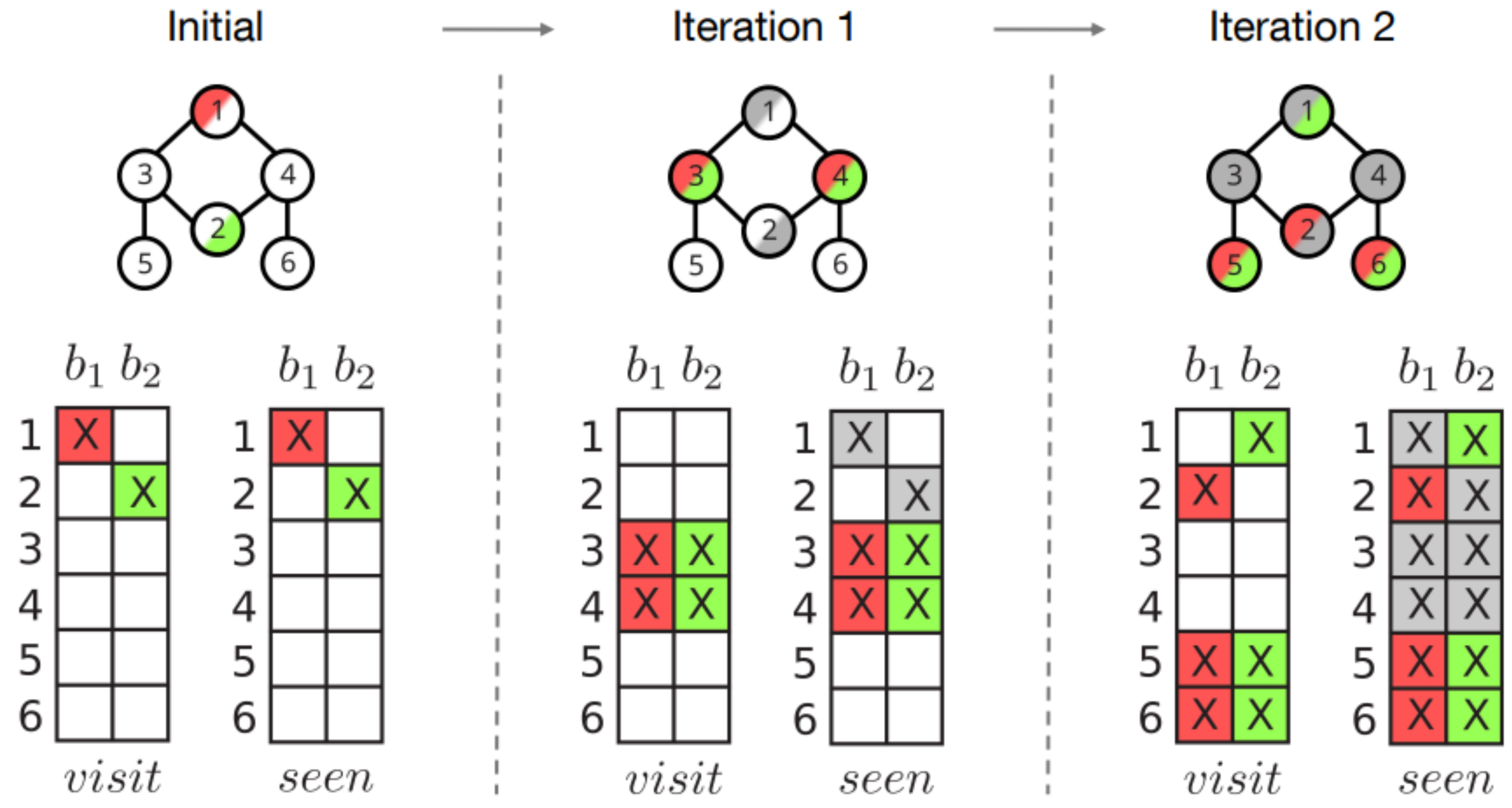
Multi-Source BFS - Example



Multi-Source BFS - Example



Multi-Source BFS - Example



MS-BFS work analysis

One round:

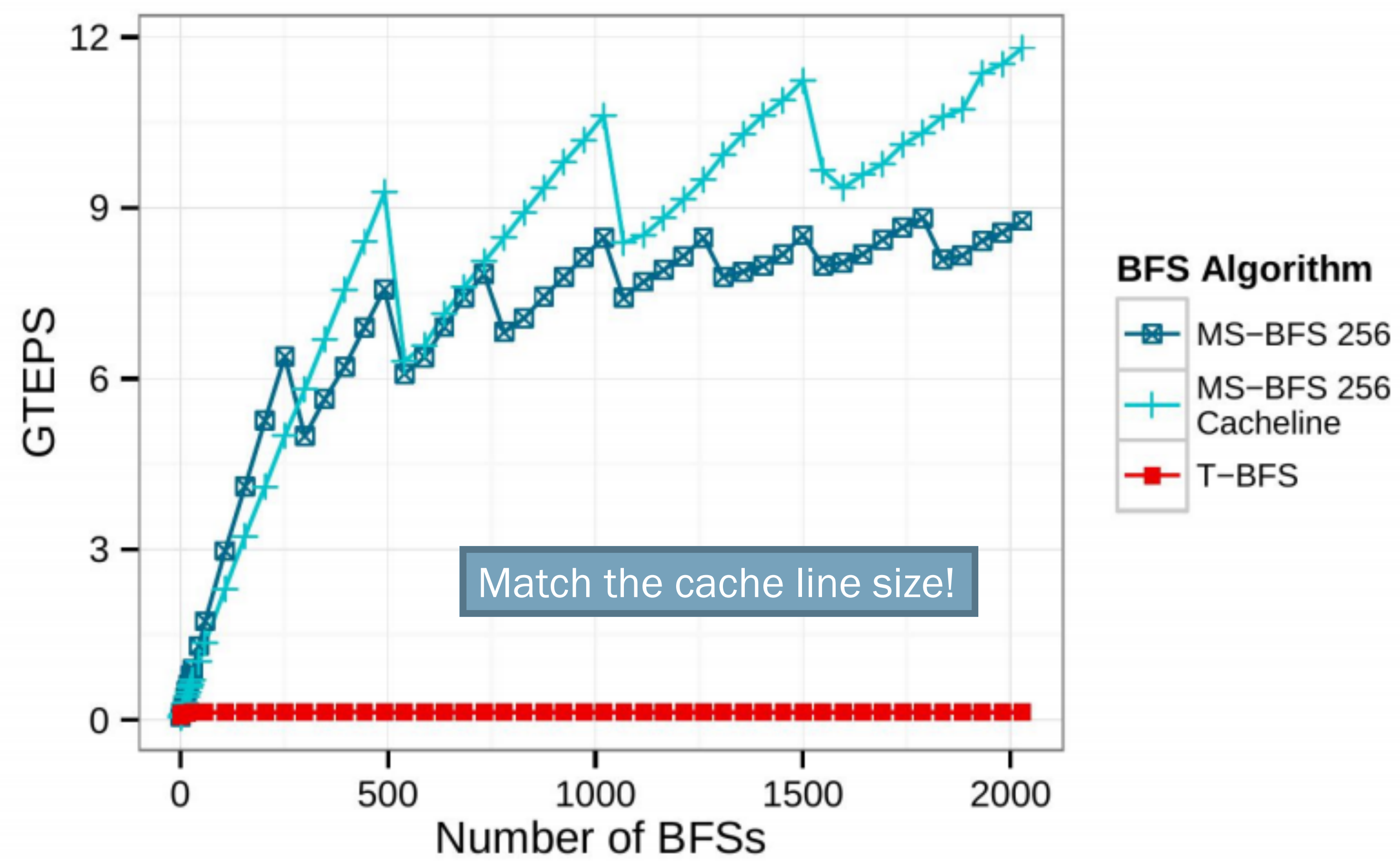
$$
\begin{array}{l}
\textbf{for } i = 1, \ldots, N \\
\quad \textbf{if } visit[v_i] = \mathbb{B}_{\varnothing}\textbf{: skip} \\
\quad \textbf{for each } n \in neighbors[v_i] \\
\quad\quad \mathbb{D} \leftarrow visit[v_i]\ \&\ {\sim}seen[n] \\
\quad\quad \textbf{if } \mathbb{D} \neq \mathbb{B}_{\varnothing} \\
\quad\quad\quad visitNext[n] \leftarrow visitNext[n] \mid \mathbb{D} \\
\quad\quad\quad seen[n] \leftarrow seen[n] \mid \mathbb{D}
\end{array}
$$

- O(m) work per round
- O(diameter) rounds needed.
- O(m × diameter) total work for ω traversals.
- Textbook BFS takes O(m) for one traversal

How wide to make the bitvectors?

- Bitvector width = number of concurrent BFSs (per thread)
- Maximize SIMD parallelism by matching the width of largest registers?
- Wider, by using multiple registers?

Evaluation - The More the Merrier



MS-BFS maximizes use of each cache miss

One round:

$$
\begin{aligned}
&\textbf{for } i = 1, \ldots, N \\
&\quad \textbf{if } visit[v_i] = \mathbb{B}_{\varnothing}\text{: } \textbf{skip} \\
&\quad \textbf{for each } n \in neighbors[v_i] \\
&\qquad \mathbb{D} \leftarrow visit[v_i]\ \&\ {\sim}seen[n] \\
&\qquad \textbf{if } \mathbb{D} \neq \mathbb{B}_{\varnothing} \\
&\qquad\quad visitNext[n] \leftarrow visitNext[n] \mid \mathbb{D} \\
&\qquad\quad seen[n] \leftarrow seen[n] \mid \mathbb{D}
\end{aligned}
$$

- Each cache line in seen[] accessed once per adjacent edge
- Many concurrent BFSs amortize cost of cache line movement.
- Working set size = 3 bits per node per concurrent BFS.

Backup 2

Table 2: Memory consumption of MS-BFS for N vertices, ω-sized bit fields, and P parallel runs.

N	ω	P	Concurrent BFSs	Memory
1,000,000	64	1	64	22.8 MB
1,000,000	64	16	1,024	366.2 MB
1,000,000	64	64	4,096	1.4 GB
1,000,000	512	1	512	183.1 MB
1,000,000	512	16	8,192	2.9 GB
1,000,000	512	64	32,768	11.4 GB
50,000,000	64	64	4,096	71.5 GB
50,000,000	512	64	32,768	572.2 GB

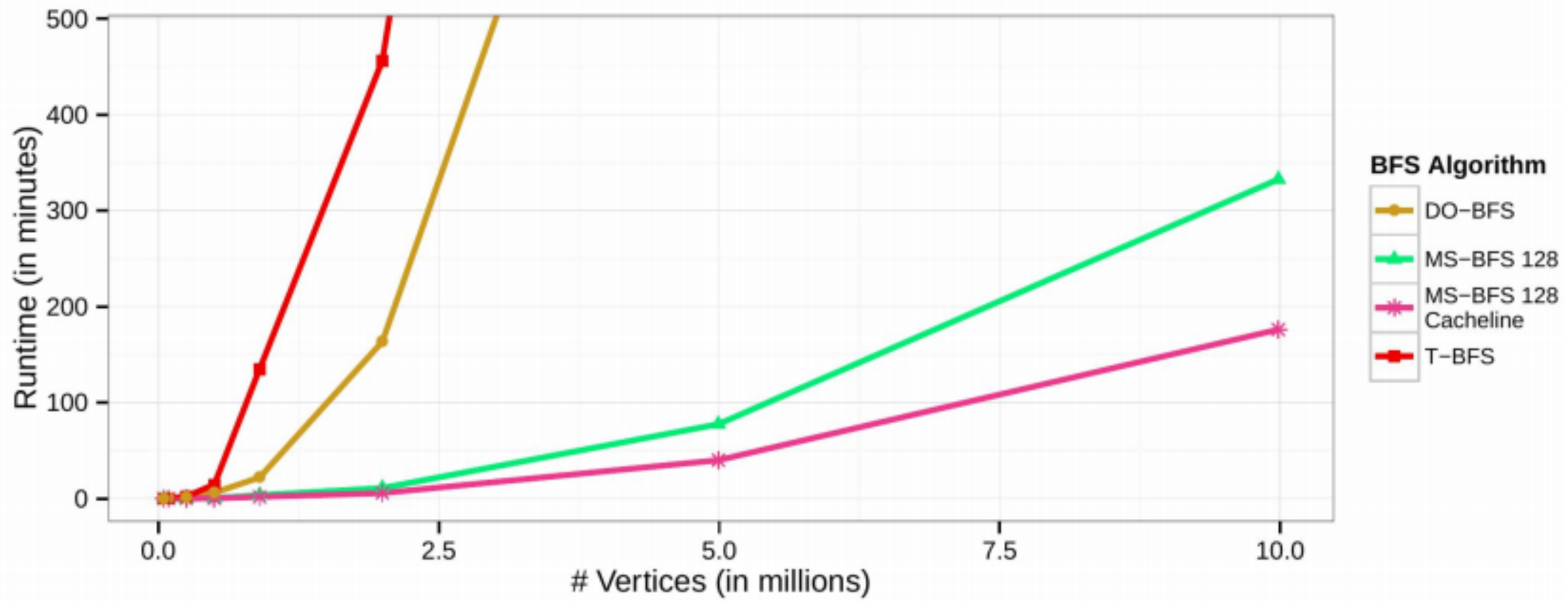
Multi-Source BFS - Further Improvements

- Aggregated neighbor processing
  - reduce number of random writes
- Batching heuristics for maximum sharing
- Direction-optimizing
- Prefetching

... see paper

Evaluation

- MS-BFS-based closeness centrality. 4x Intel Xeon E7-4870v2, 1TB



		Speedup over	
Graph	MS-BFS	T-BFS	DO-BFS
LDBC 1M	0:02h	73.8x	12.1x
LDBC 10M	2:56h	88.5x	28.7x
Wikipedia	0:26h	75.4x	29.5x
Twitter (1M)	2:52h	54.6x	12.7x

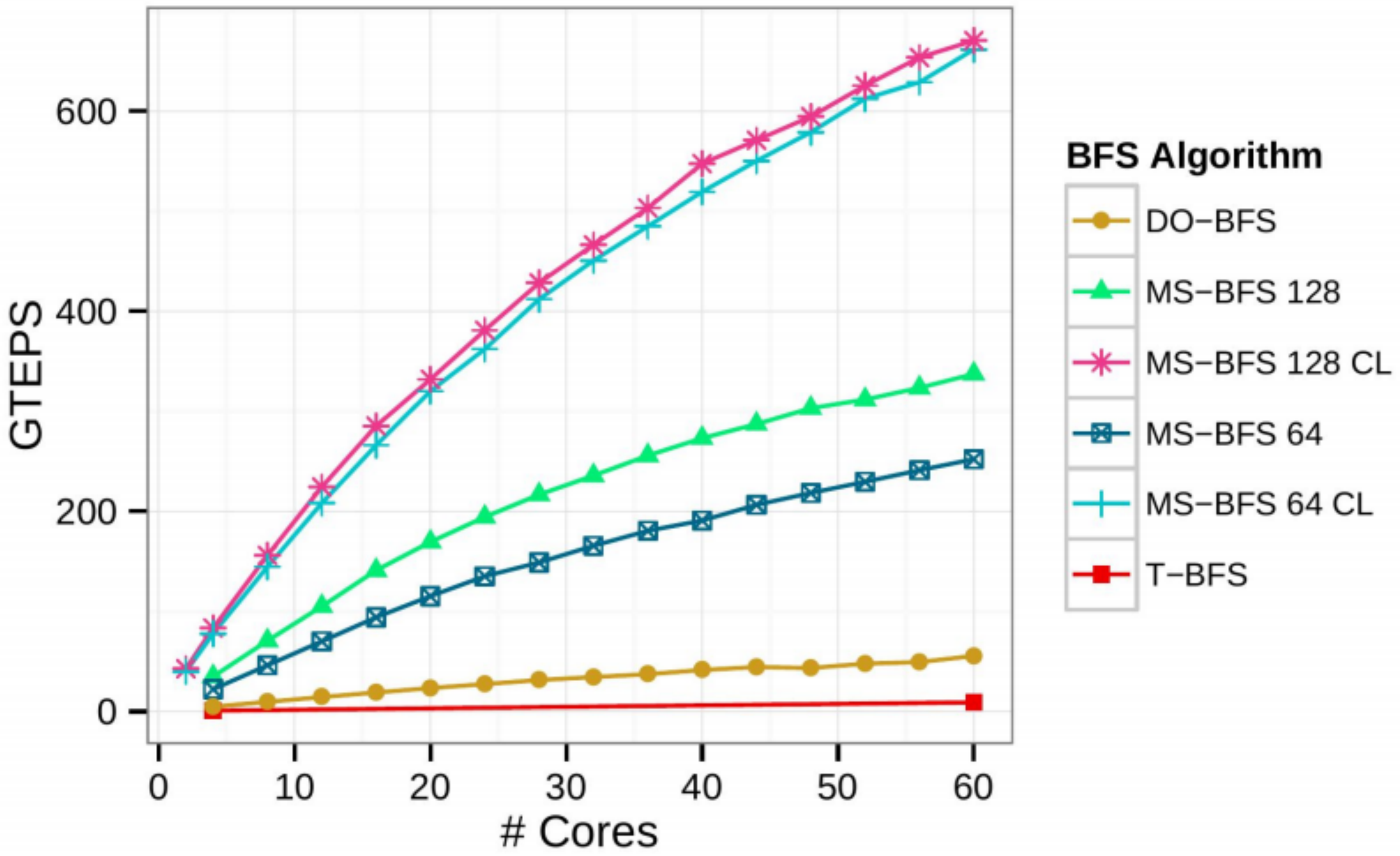
Conclusions

- Making parallel traversals aware of each other improves efficiency.
- Changing random accesses to predictable array scans improves efficiency.
- MS-BFS runs multiple BFSs
  - *On the same graph*
  - *Concurrently one core*
  - *Amortizes cache line movement cost*
- >10x speedup over parallel direction-optimizing BFS

Future work

- Combining parallelism across traversals with parallelism within traversals.
- Alternative architectures:
  - *GPUs?*
  - *Clusters?*
- Applications beyond closeness centrality.
- Other graphs. What if few long chains?
- Other types of traversals. Bellman-Ford?
- Integrating into a graph analytics framework.

Backup 1



GTEPS
600
400
200
0
0
10
20
30
40
50
60
Cores
BFS Algorithm
DO−BFS
MS−BFS 128
MS−BFS 128 CL
MS−BFS 64
MS−BFS 64 CL
T−BFS