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 Ying
6.886 – February 23, 2021

Background

- Some applications do many BFS traversals (from different starting nodes) on one graph
  - *E.g., compute centrality metrics across graph*
- Prior work: parallel BFS, direction-optimizing BFS
  - *Speed up a single BFS traversal*
- Graph traversals have poor cache behavior
  -

```
while visit ≠ ∅
    for each v ∈ visit do
        for each n ∈ neighbors_v do
            if n ∉ seen then
                seen ← seen ∪ {n}
                visitNext ← visitNext ∪ {n}
                do BFS computation on n
    visit ← visitNext
    visitNext ← ∅
```

Finding neighbors in graph is non-trivial expense

Read a single random bit in *seen* bitset when traversing each edge

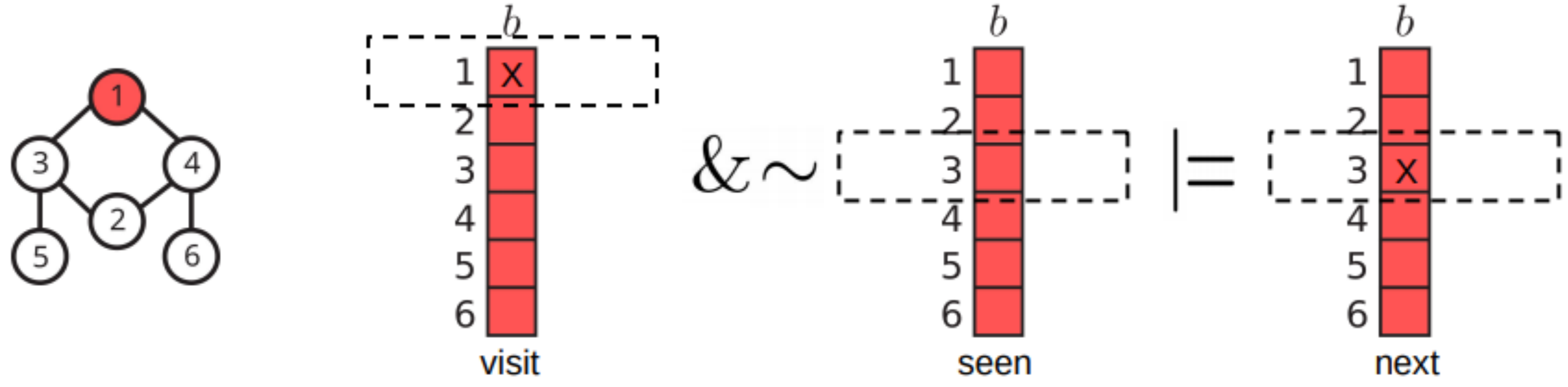
- Small-world phenomenon: **some** graphs have low diameter

Multi-Source BFS

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

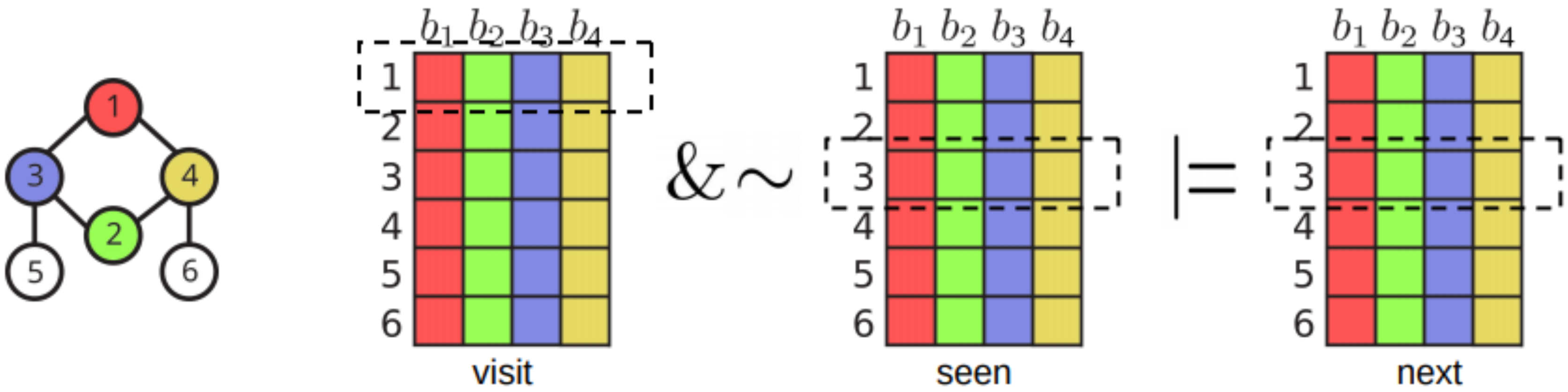
Multi-Source BFS

- 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)

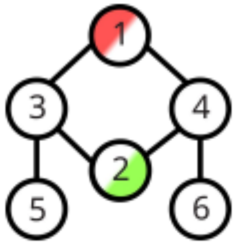
One round:

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

- Given frontiers, compute next frontiers
- By traversing **every** edge in the graph once.
- Cost of finding neighbors is amortized over several traversals

Multi-Source BFS - Example

Initial



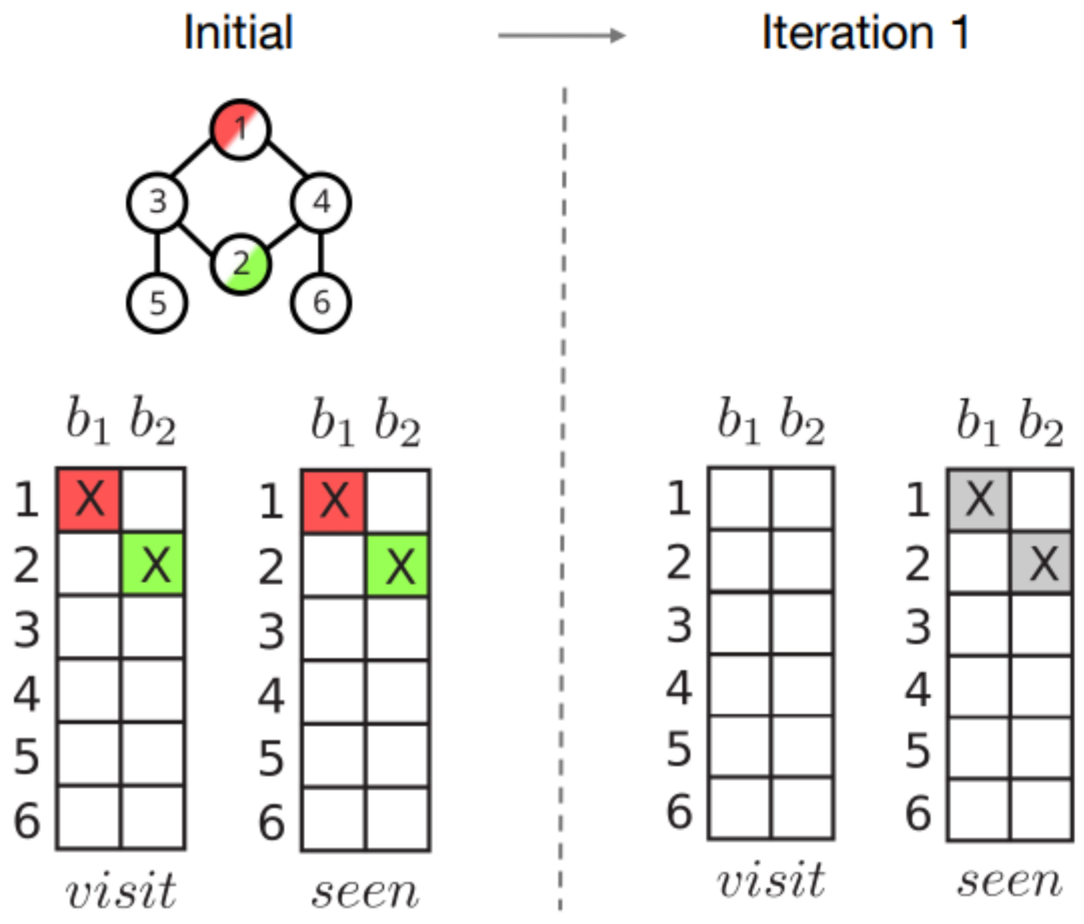
	b_1	b_2
1	X	
2		X
3		
4		
5		
6		

$visit$

	b_1	b_2
1	X	
2		X
3		
4		
5		
6		

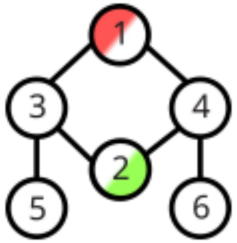
$seen$

Multi-Source BFS - Example



Multi-Source BFS - Example

Initial → **Iteration 1**



Initial

visit

	b_1	b_2
1	X	
2		X
3		
4		
5		
6		

seen

	b_1	b_2
1	X	
2		X
3		
4		
5		
6		

Iteration 1

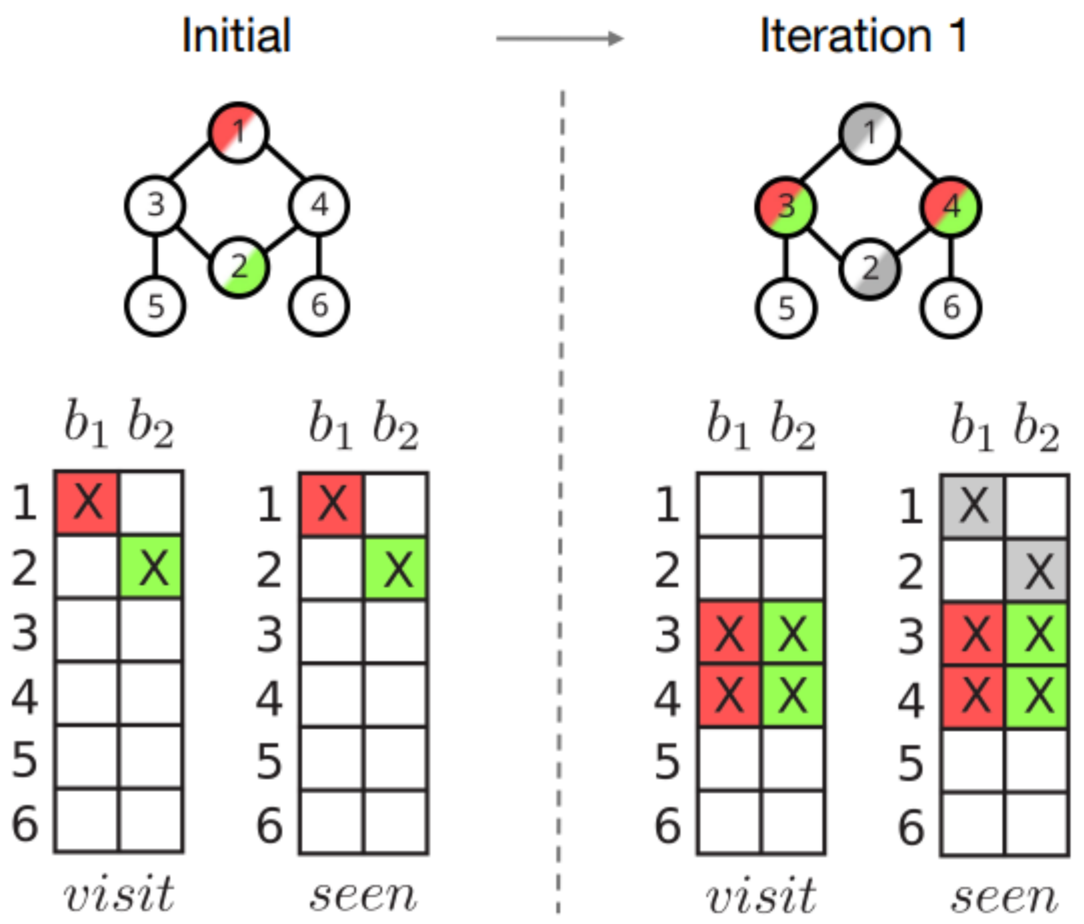
visit

	b_1	b_2
1		
2		
3	X	
4	X	
5		
6		

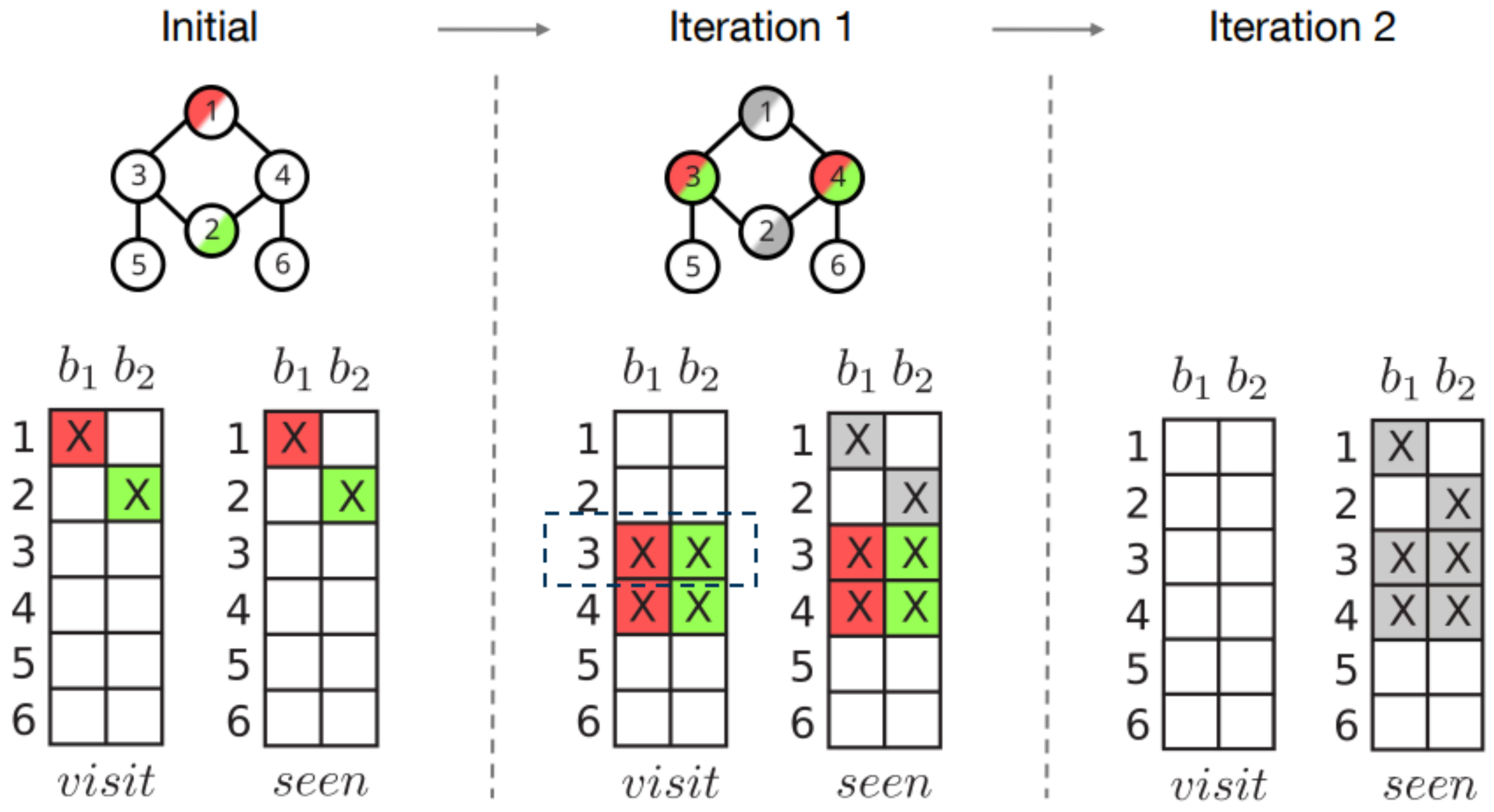
seen

	b_1	b_2
1	X	
2		X
3	X	
4	X	
5		
6		

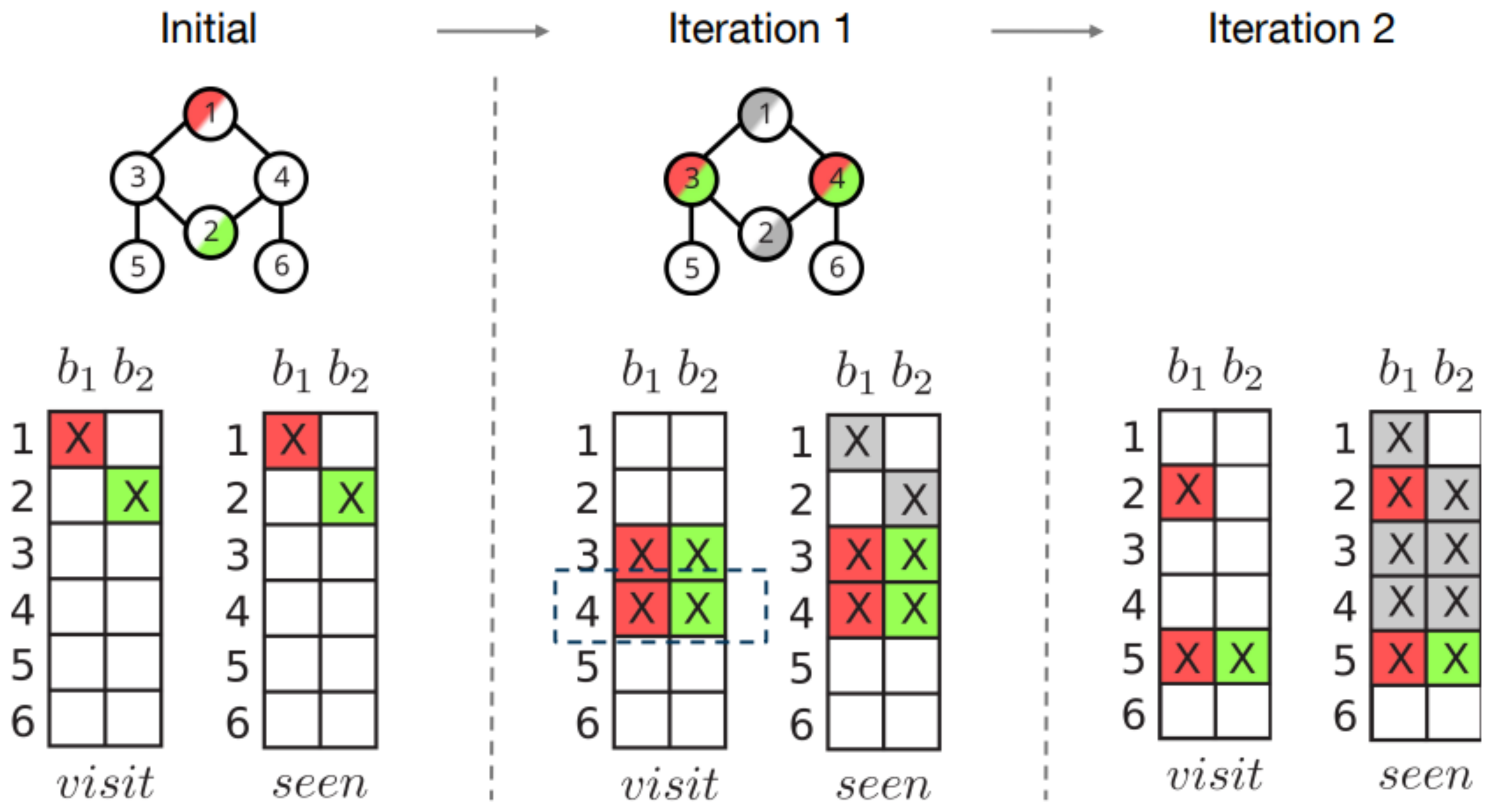
Multi-Source BFS - Example



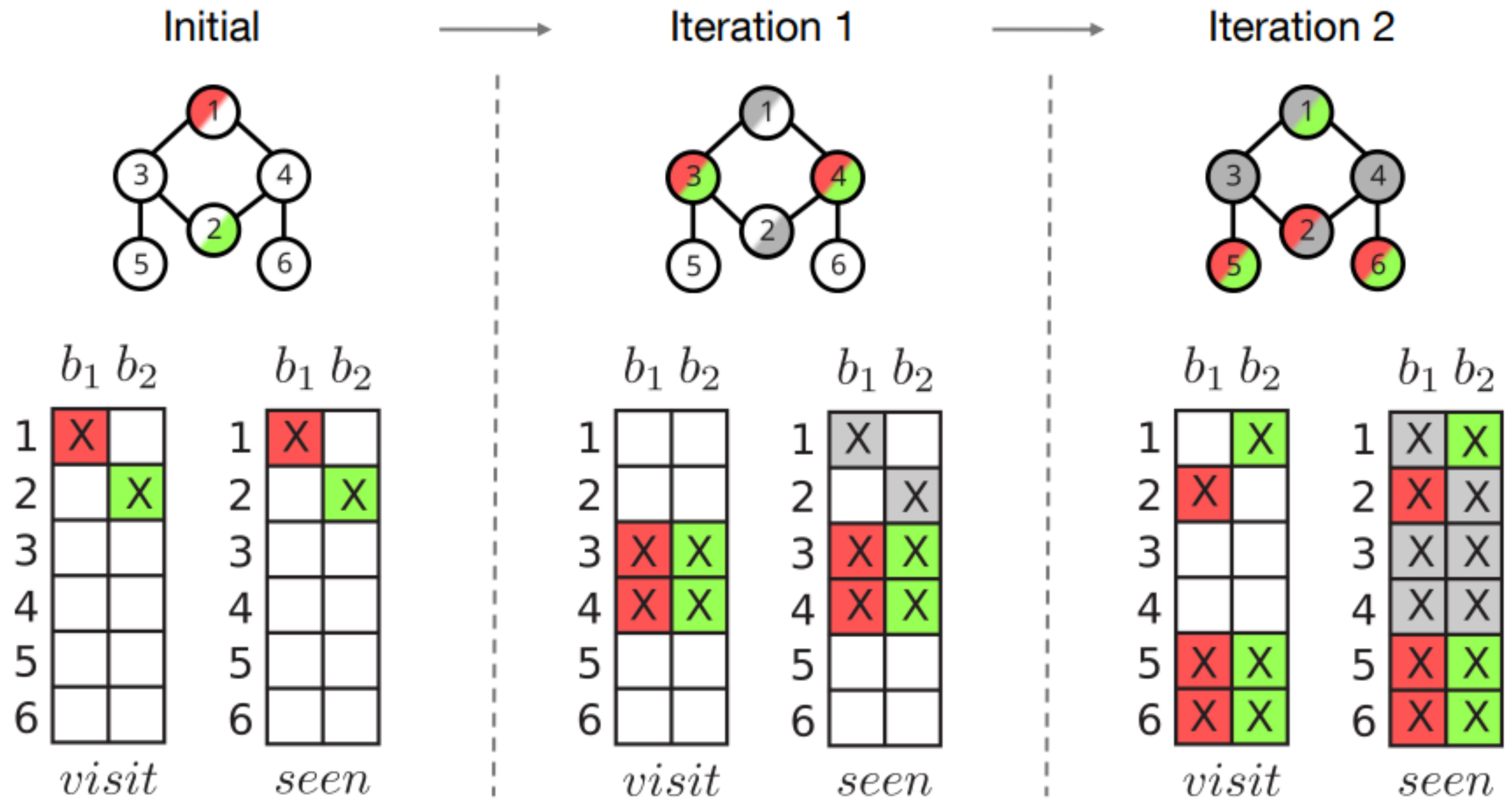
Multi-Source BFS - Example



Multi-Source BFS - Example



Multi-Source BFS - Example



MS-BFS work analysis

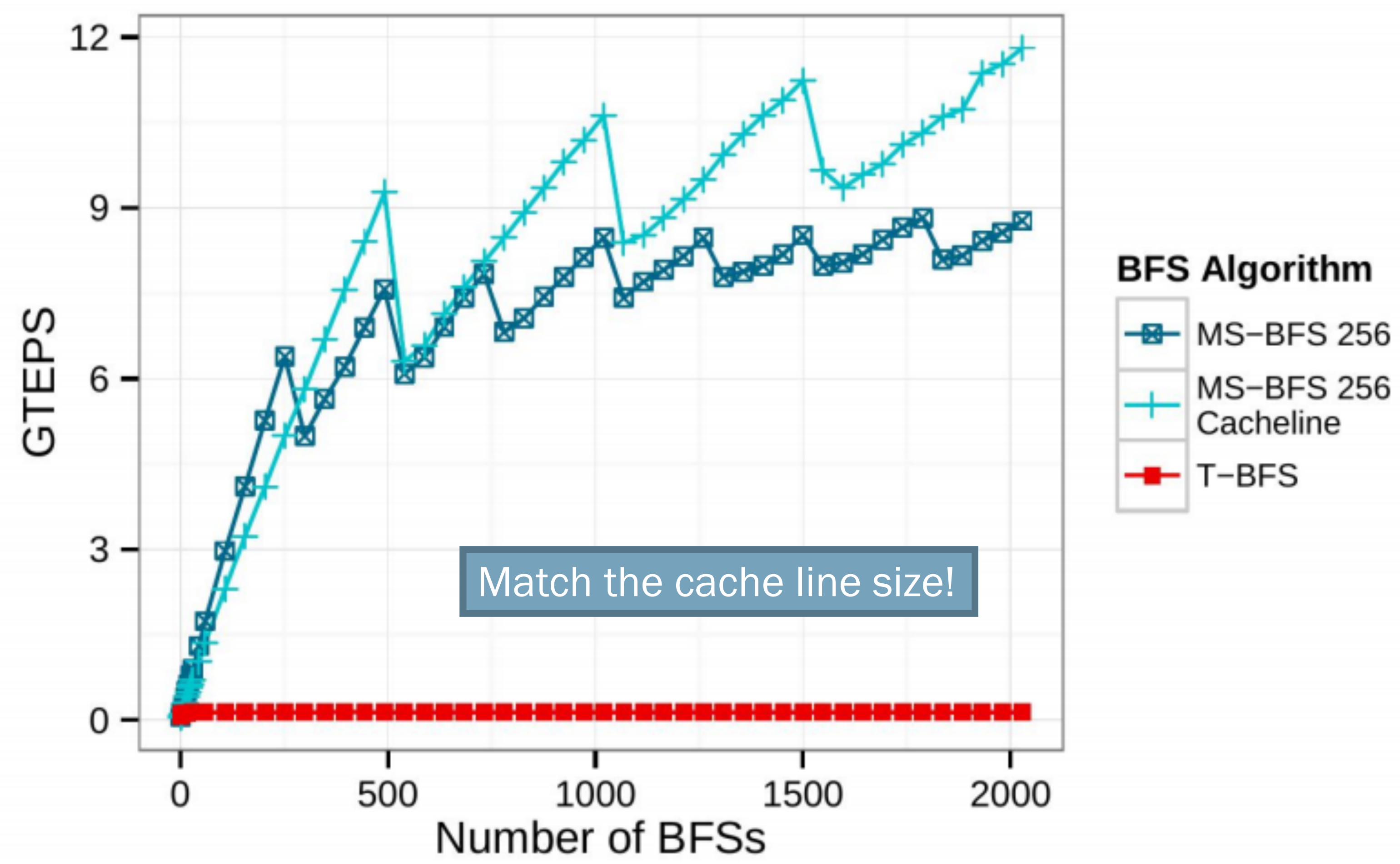
$$
\begin{array}{l}
\text{while } \mathit{visit} \neq \varnothing \\
\quad \text{for } i = 1, \ldots, N \\
\quad\quad \text{if } \mathit{visit}[v_i] = \mathbb{B}_{\varnothing}, \text{ skip} \\
\quad\quad \text{for each } n \in \mathit{neighbors}[v_i] \\
\quad\quad\quad \mathbb{D} \leftarrow \mathit{visit}[v_i] \ \& \sim\mathit{seen}[n] \\
\quad\quad\quad \text{if } \mathbb{D} \neq \mathbb{B}_{\varnothing} \\
\quad\quad\quad\quad \mathit{visitNext}[n] \leftarrow \mathit{visitNext}[n] \mid \mathbb{D} \\
\quad\quad\quad\quad \mathit{seen}[n] \leftarrow \mathit{seen}[n] \mid \mathbb{D} \\
\quad\quad\quad\quad \text{do BFS computation on } n \\
\quad \mathit{visit} \leftarrow \mathit{visitNext} \\
\quad \text{reset } \mathit{visitNext}
\end{array}
$$

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

How wide to make the bitvectors?

- Bitvector width (ω) = 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 improves cache performance

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

- Each cache line in seen[] accessed once per adjacent edge
- Many concurrent BFSs amortize cost of cache line movement.
- Most expensive line is still this random access to seen[]

MS-BFS with "aggregated neighbor processing"

$$
\begin{aligned}
&\text{while } \mathit{visit} \neq \varnothing \\
&\quad \text{for } i = 1, \ldots, N \\
&\quad\quad \text{if } \mathit{visit}[v_i] = \mathbb{B}_\varnothing, \text{ skip} \\
&\quad\quad \text{for each } n \in \mathit{neighbors}[v_i] \\
&\quad\quad\quad \mathit{visitNext}[n] \leftarrow \mathit{visitNext}[n] \mid \mathit{visit}[v_i] \\
&\\
&\quad \text{for } i = 1, \ldots, N \\
&\quad\quad \text{if } \mathit{visitNext}[v_i] = \mathbb{B}_\varnothing, \text{ skip} \\
&\quad\quad \mathit{visitNext}[v_i] \leftarrow \mathit{visitNext}[v_i] \;\&\; {\sim}\mathit{seen}[v_i] \\
&\quad\quad \mathit{seen}[v_i] \leftarrow \mathit{seen}[v_i] \mid \mathit{visitNext}[v_i] \\
&\quad\quad \text{if } \mathit{visitNext}[v_i] \neq \mathbb{B}_\varnothing \\
&\quad\quad\quad \text{do BFS computation on } v_i \\
&\quad \mathit{visit} \leftarrow \mathit{visitNext} \\
&\quad \text{reset } \mathit{visitNext}
\end{aligned}
$$

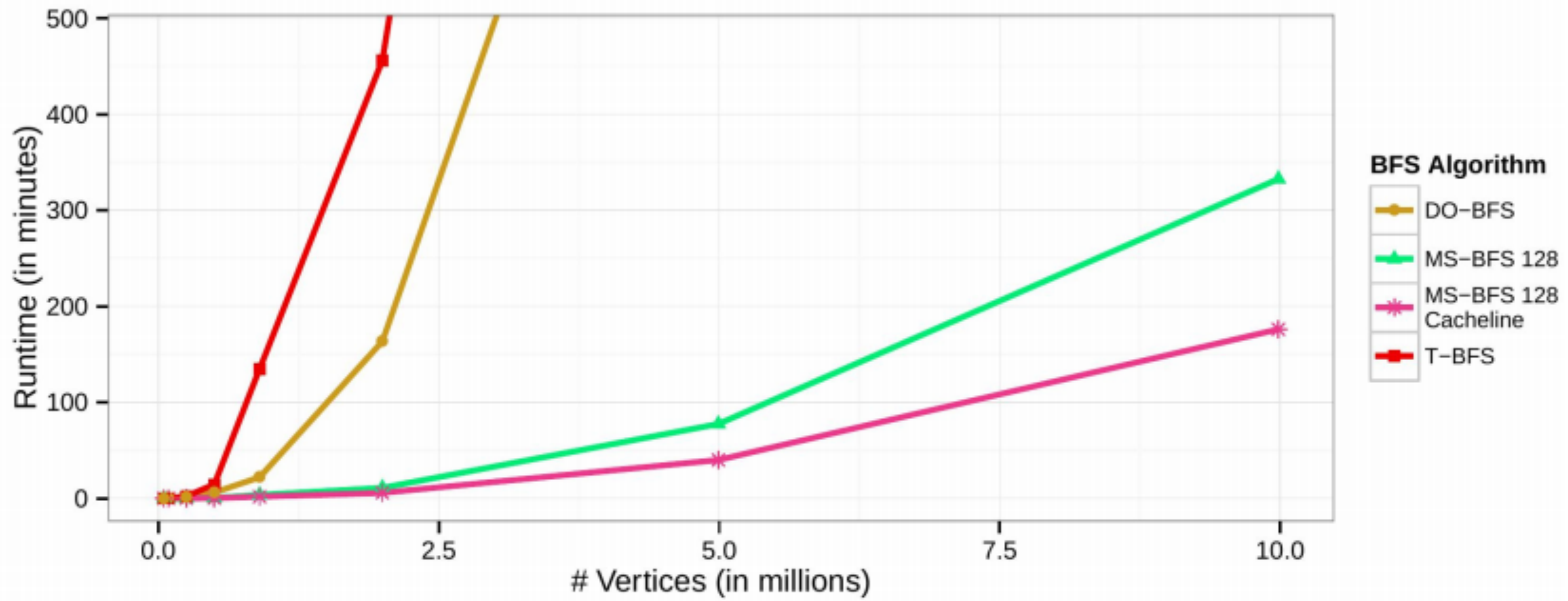
- Defer accesses to seen[], and then do the accesses in scanning fashion, so each entry in seen[] is accessed at most once

MS-BFS: further improvements

- Direction-optimizing
- Explicit prefetching
- Heuristics to decide what groups of BFS tranversals to run together

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

Multi-core scalability?!

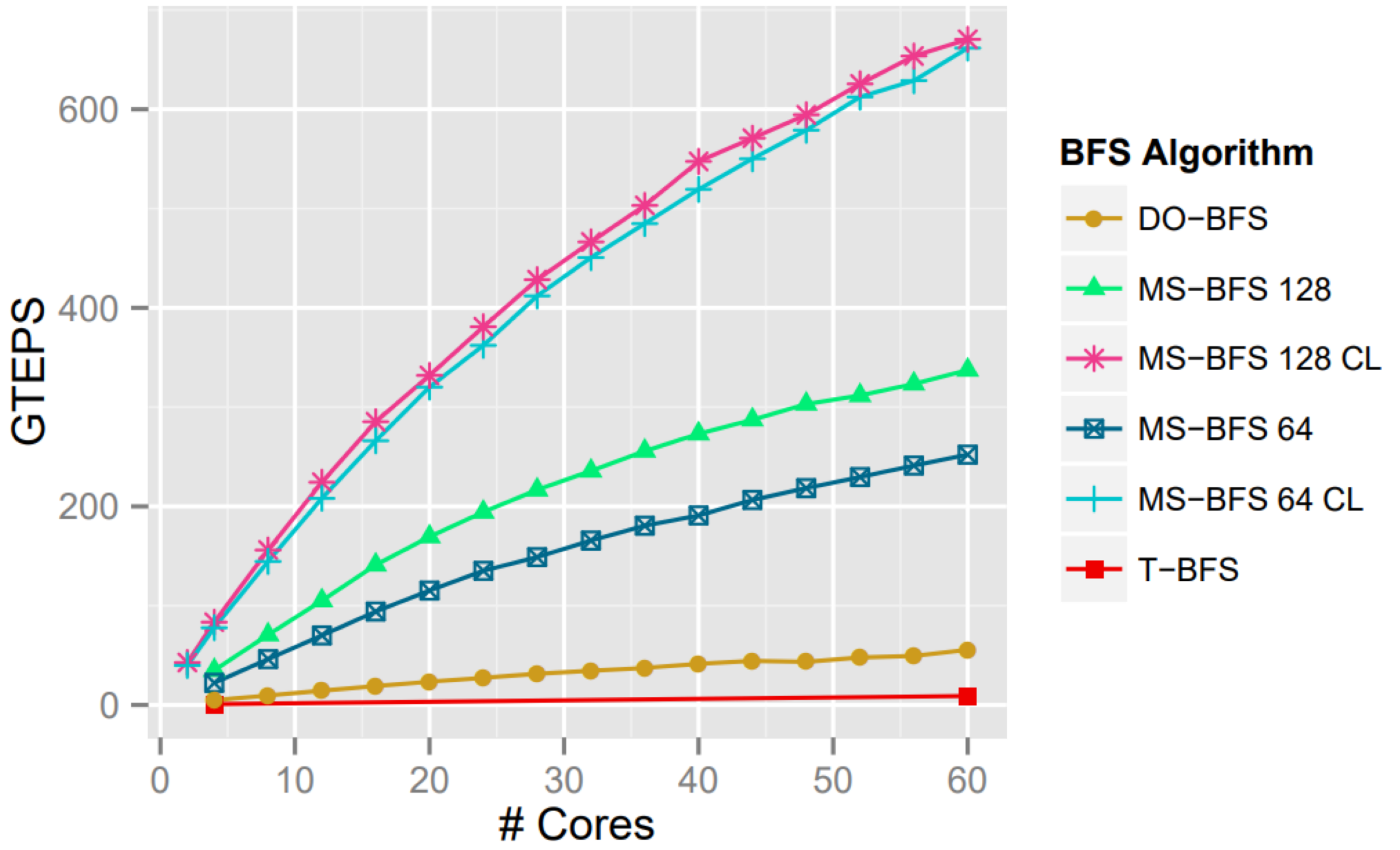


Figure 5: Multi-core scalability results.

Results only for LDBC 1M-vertex graph, which is ~314 MB

Conclusions

- MS-BFS runs multiple BFSs
  - *On the same graph*
  - *Within a single thread*
  - *Amortizes cache line movement cost*
- For low-diameter graphs, a large fraction of vertices are visited each round, so you can amortize cost of traversing graph over many concurrent traversals.
- >10x speedup over direction-optimizing BFS
- Changing random accesses to predictable array scans improves efficiency.

Future work

- Combining parallelism across traversals with parallelism within traversals.
- Alternative architectures:
  - *GPUs should be good at exploiting SIMD-style parallelism?*
- Applications beyond closeness centrality.
- Other graphs. Is there a hybrid approach that works if graphs have moderate diameter?
- Other types of traversals besides BFS. Does it make sense to do multi-source SSSP/"weighted BFS" traversals on weighted graphs?
- Integrating into a graph analytics framework or a graph processing benchmark set?