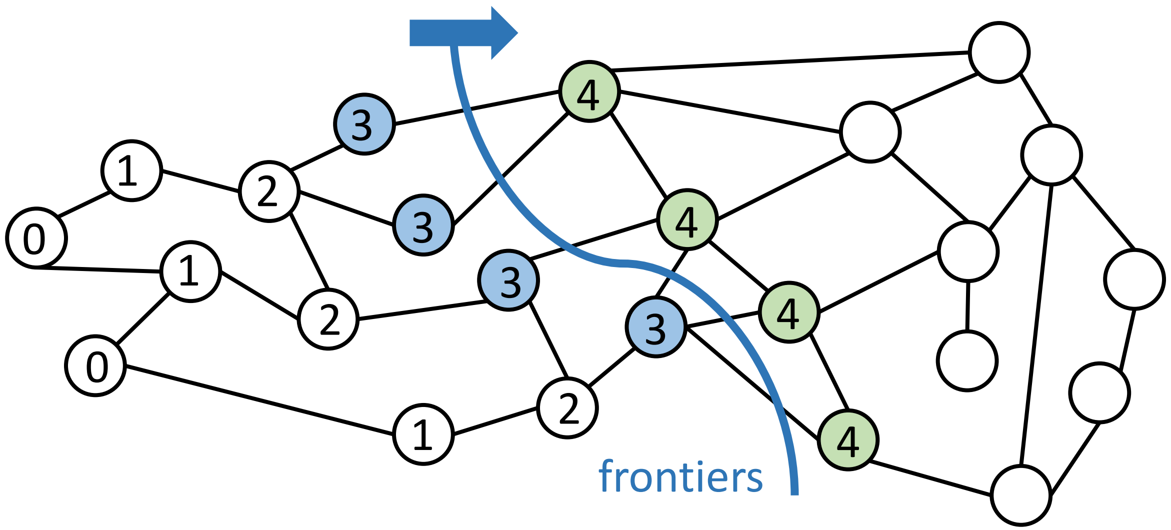
An Efficient Implementation of Parallel Breadth-first Search

Pao-I Chen and Tsung-Wei (TW) Huang
Department of Electrical and Computer Engineering
University of Wisconsin at Madison, Madison, WI
https://tsung-wei-huang.github.io/

Breadth-first Search (BFS) and Algorithm

- **BFS is a fundamental graph traversal algorithm for many applications**
  - Ex: shortest path finding, network analysis, path finding
- **BFS is easy to parallelize due to its level-by-level traversal process**
  - Nodes at level *L* finish first before going to *L+1*
  - Nodes at the same level can run in parallel
  - Implemented via a *frontier*-based framework
    - Guarded by compare-and-swap (CAS) operations



Algorithm 1: Parallel Breadth-First Search (BFS)

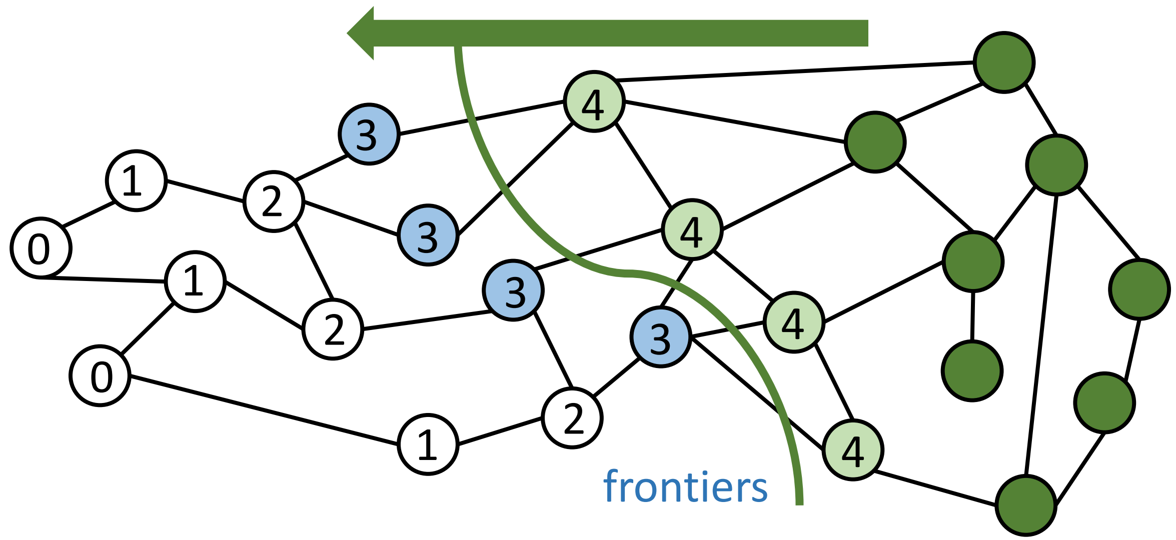
```
Input   : Graph G = (V, E), source vertex s
Output  : Distance array dist[|V|], initialized to ∞
dist[s] ← 0;
Q ← {s};
while Q ≠ ∅ do
    Q_next ← ∅;
    foreach u ∈ Q in parallel do
        foreach v ∈ Neighbors(u) do
            if AtomicCAS(dist[v], ∞, dist[u] + 1) then
                Add v to Q_next;
            end
        end
    end
    Q ← Q_next;
end
```

Bi-directional BFS (BD-BFS) Algorithm

- **Instead of finding next frontiers from current frontiers ("top-down")**
  - Process all vertices in parallel and let each *unexplored* vertex decide whether it can be the next frontier, i.e., with a neighbor at the current frontier
    - Aka "**bottom-up step**"
- **Pros and cons of this bottom-up step:**

☺ High parallelism, early break, no CAS operations

☹ Redundant work (i.e., no neighbors in frontiers)



Algorithm 2: Bottom-up Step

Input : Current frontier queue Q

Output : Next frontier queue Q_{next}

$Q_{\text{next}} \leftarrow \emptyset$;

foreach $u \in V$ **in parallel do**

  foreach $v \in \text{Neighbors}(u)$ **do**

    if $v \in Q$ **then**

      $dist[u] \leftarrow dist[v] + 1$;

      Add u to Q_{next};

      break;

    end

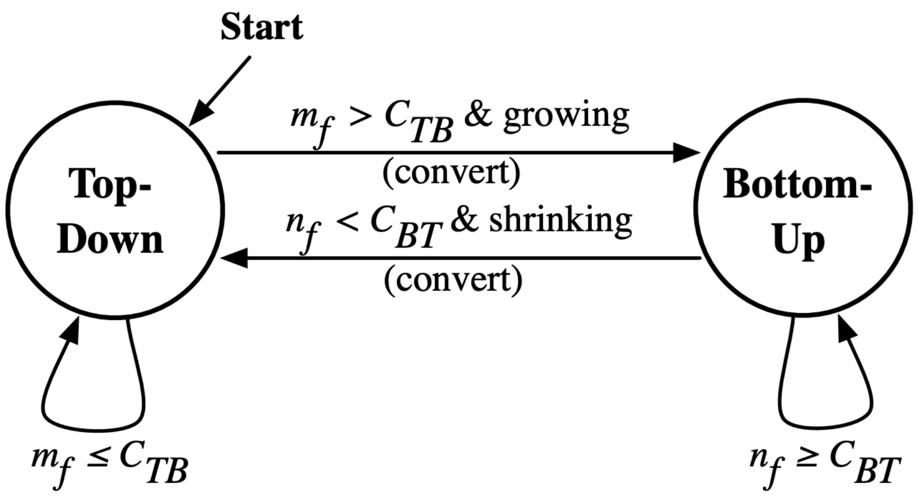
  end

end

[1]S. Beamer, "Direction-optimizing Breadth-First Search," SC'12

Implementation Challenges in BD-BFS

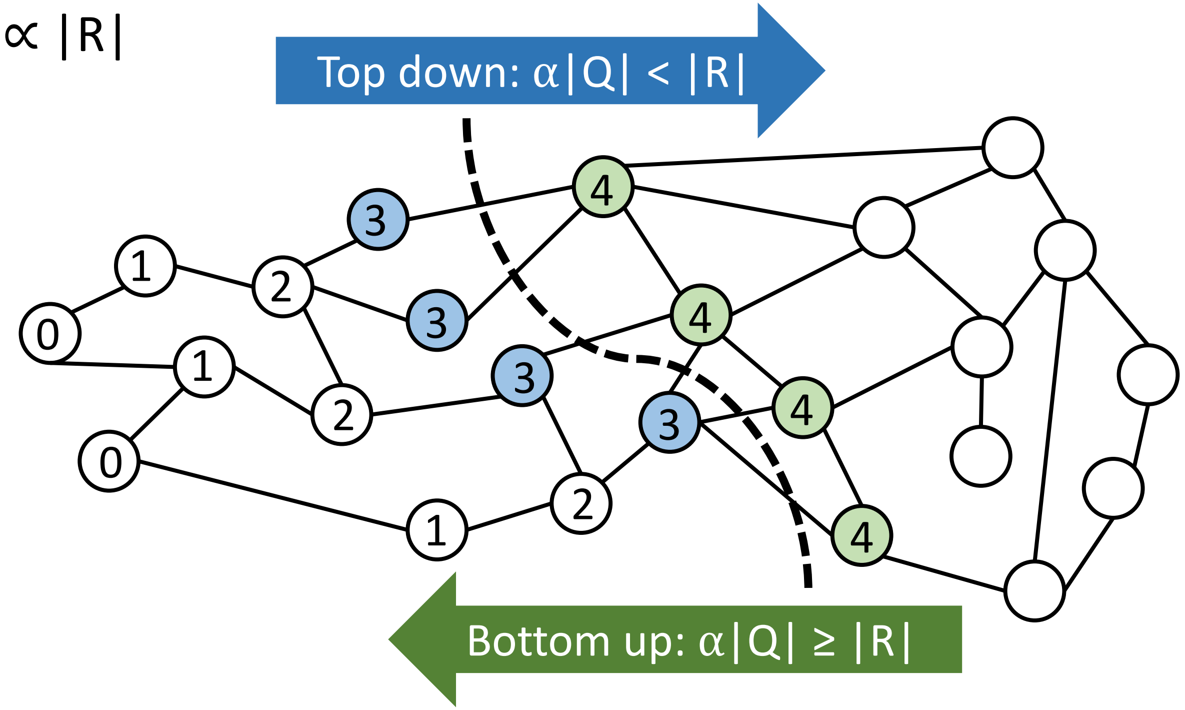
- **BD-BFS relies on carefully tuned parameters to balance the two steps**
  - N_f: the number of current frontiers
  - M_f: the number of edges to check from current frontiers
  - M_u: the number of edges to check from unexplored vertices
  - C_{TB}: user-defined threshold to switch from top-down step to bottom-up step
  - C_{BT}: user-defined threshold to switch from bottom-up step to top-down step
- **Other costly implementation details**[1]
  - Concurrent bitmap for tracking vertex status
  - Sliding window-based queue
  - Iterative parallel reductions for
    - Updating M_f, M_u
    - Extracting unexplored vertices
    - …



[1]GAP benchmark suite: https://github.com/sbeamer/gapbs

The Proposed Algorithm (1/2)

- **A simple yet effective idea – directly estimate the workload of each step**
  - Q: frontier queue, tracking the current frontiers
  - R: remainder queue, tracking the current unexplored vertices
  - α: average edge degree per vertex, $|E|/|V|$
  - Top-down work $\propto |Q| \times \alpha$
  - Bottom-up work $\propto |R|$



The Proposed Algorithm (2/2)

Algorithm 3

Input : Graph $G = (V, E)$, source vertex s, current frontier queue Q, current remainder queue R, next frontier queue Q_{next}, next remainder queue R_{next}

Output: Distance array $dist[|V|]$, initialized to ∞

$R \leftarrow V - \{s\}$;
$Q \leftarrow \{s\}$;
$dist[s] \leftarrow 0$;
$\alpha \leftarrow |E|/|V|$;
while $|Q|$ and $|R|$ **do**
    $Q_{\text{next}} \leftarrow \emptyset$;
    $R_{\text{next}} \leftarrow \emptyset$;
    if $|R| < |Q| \times \alpha$ **then**

Bottom-up step

        foreach $u \in R$ **in parallel do**
            if $dist[u] \neq \infty$ **then**
                bottom_step(u);
                if $dist[u] = \infty$ **then**
                    Add u to R_{next};
                end
            else
                Add u to Q_{next};
            end
        end
    end
    else

Top-down step

        foreach $u \in Q$ **in parallel do**
            foreach $v \in$ Neighbors(u) **do**
                if AtomicCAS($dist[v]$, ∞, $dist[u] + 1$) **then**
                    Add v to Q_{next};
                end
            end
        end
    end
    $Q \leftarrow Q_{next}$;
    $R \leftarrow R_{next}$;
end

Optimization Details

- **We perform parallel traversal only when the queue size is greater than 32**
  - Avoid unnecessary threading overhead when the vertex parallelism is limited
- **We perform lazy initialization and lazy update on *R* whenever needed**
  - Avoid frequent update on *R* as in practice bottom-up steps happen only a few times
- **We perform different scheduling algorithms for bottom-up and top-down steps**
  - Top-down step runs *static scheduling* as frontiers needs to scan all their neighbors
    - chunk size = 4
  - Bottom-up step runs *dynamic scheduling* as unexplored vertices may early-break the scan
    - chunk size = 32
- **We keep per-thread storage for *Q* (frontier queue) and *R* (remainder queue)**
  - Avoid excessive synchronization and contention due to centralized storage

Experimental Results

- **Baseline: BD-BFS, implemented using OpenMP with our optimization strategies**
  - Also removed the bitmap data structure as we didn't observe much performance advantage
  - Original BD-BFS implementation[1] achieved an overall score of about 640M edges/s
- **Our algorithm, implemented using C++ Thread, Taskflow[2], and OpenMP**
  - Achieved nearly 40% performance improvement over the BD-BFS baseline

			Reference	BD-BFS (OpenMP)		Ours (C++ Thread)		Ours (Taskflow)		Ours (OpenMP)	
	|V|	|E|	Time	Time	edges/s	Time	edges/s	Time	edges/s	Time	edges/s
Collaboration Network 1	1.1M	113M	470	5.21	21.20B	4.28	25.82B	4.07	27.09B	3.90	28.31B
Road Network 1	22.1M	30M	3310	210	283.03M	640	90.74M	160	355.93M	160	356.41M
Road Network 2	87M	112.9M	14670	720	199.15M	760	285.76M	560	387.62M	530	407.30M
Social Network	4.9M	85.8M	1060	20.04	4.19B	13.30	6.31B	17.59	4.77B	13.57	6.19B
Synthetic Dense	10M	1B	11870	43.45	22.61B	40.17	24.40B	41.80	23.44B	40.51	24.19B
Synthetic Sparse	10M	40M	1620	130	293.54M	450	86.44M	90.08	435.21M	85.40	459.06M
Web Graph	6.6M	300M	2860	24.92	11.81B	16.44	17.90B	19.90	14.79B	17.38	16.94B
kNN Graph	24.9M	158M	2100	180	876.23M	320	476.68M	130	1.22B	110	1.42B
Score (Geomean)			2042.57	66.87	2.18B	84.64	1.72B	53.02	2.75B	48.31	3.01B

[1] GAP benchmark suite: https://github.com/sbeamer/gapbs [2] Taskflow: https://github.com/taskflow/taskflow.git

Thank you!