Parallel Breadth-First Search Optimization Strategies

Marati Bhaskar Raghavendra Kanakagiri

Indian Institute of Technology Tirupati
{cs24d001, raghavendra}@iittp.ac.in

FastCode Programming Challenge (FCPC), PPoPP'25
September 7, 2026

भारतीय प्रौद्योगिकी संस्थान तिरुपति
IIT TIRUPATI

Presentation Overview

भारतीय प्रौद्योगिकी संस्थान तिरुपति
IIT TIRUPATI

Introduction

Motivation:

1. BFS is a fundamental graph algorithm with wide applications.
2. Large-scale graphs (eg: road networks, web graphs) demand parallel processing.
3. Sequential BFS cannot scale to billions of edges/vertices.

Performance Challenges:

1. Load imbalance: Frontier size varies dramatically between iterations.
2. Synchronization bottlenecks: Concurrent queue access and atomic operations.
3. Memory access pattern: Poor locality leads to cache inefficiency.

भारतीय प्रौद्योगिकी संस्थान तिरुपति
IIT TIRUPATI

Ways of Launching Threads

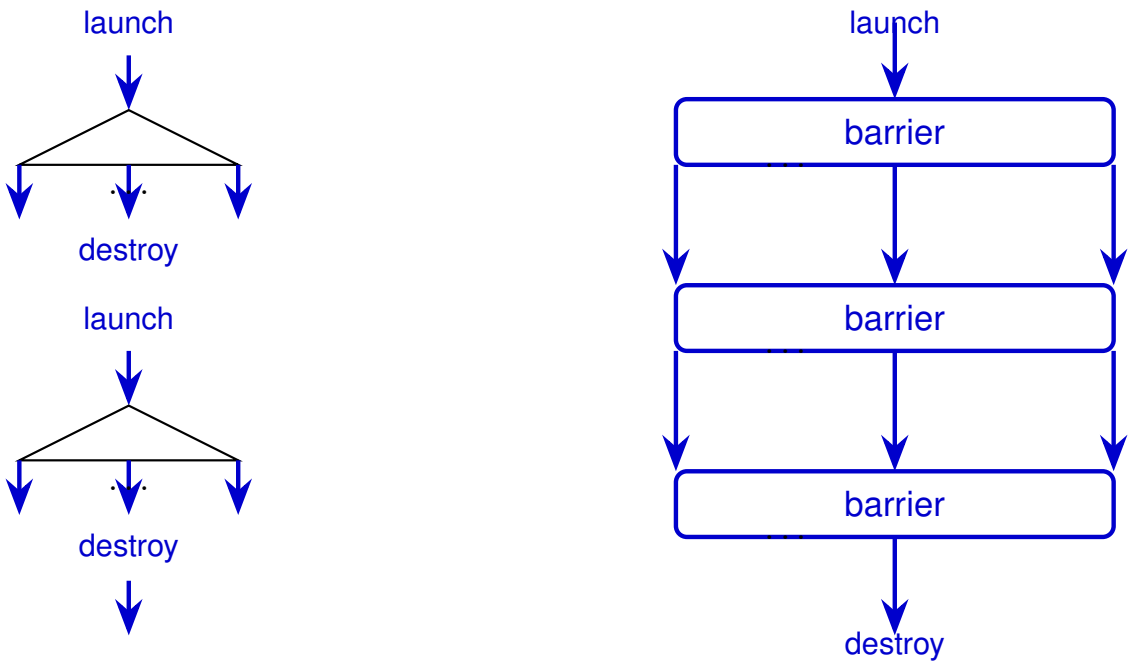

Figure: Launching and destroying threads(left). Launching threads only once(right).

Algorithm 1 BFS-Conventional

```
1: #pragma omp parallel
2: while cfsz ≠ 0 do
3:     #pragma omp for nowait
4:     for all src ∈ frontiers do
5:         for all ngh ∈ neigbor[src] do
6:             /* __sb_CAS : __sync_bool_compare_and_swap */
7:             if __sb_CAS(&dist[ngh], MAX_DIST, dist[src] + 1) then        ▷ atomic update
8:                 lf[lfsz++] = ngh;
9:             end if
10:        end for
11:    end for
12:    /*cfsz = ∑ lfsz, Merge lf[] to form frontiers[] */
13: end while
```

Algorithm 2 BFS-NonAtomic

```
1: #pragma omp parallel
2: while cfsz ≠ 0 do
3:     #pragma omp for nowait
4:     for all src ∈ frontiers do
5:         for all ngh ∈ neigbor[src] do
6:             if dist[src] + 1 < dist[ngh] then          ▷ if ngh is unvisited
7:                 lf[lfsz++] = ngh
8:                 dist[ngh] = dist[src] + 1              ▷ Non-atomic update
9:             end if
10:        end for
11:    end for
12:    /*cfsz = ∑ lfsz, Merge lf[] to form frontiers[] */
13: end while
```

Duplicates Percentage

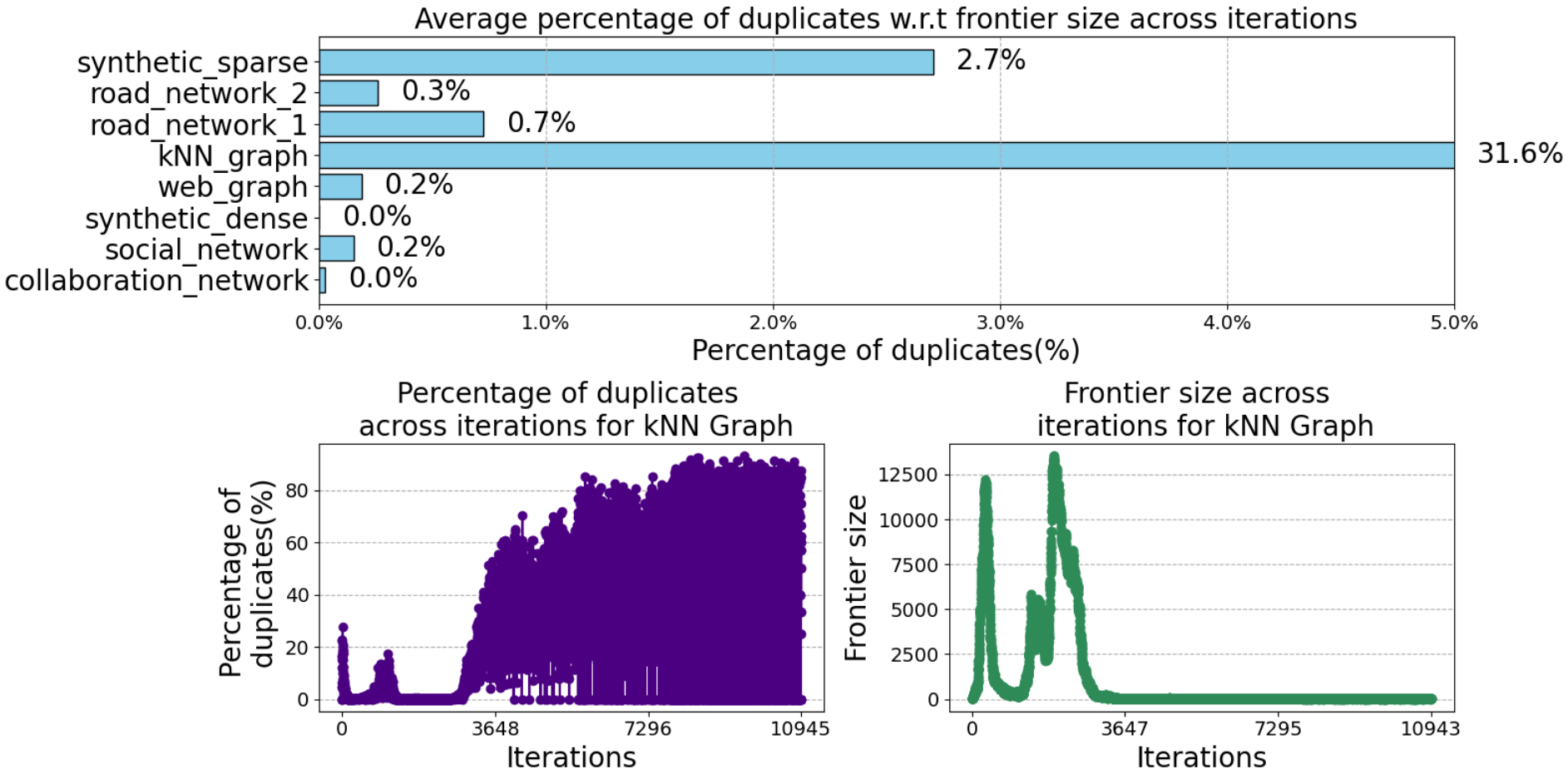

Figure: Analysis of duplicates and frontier size across iterations

Top-Down Vs Bottom-Up

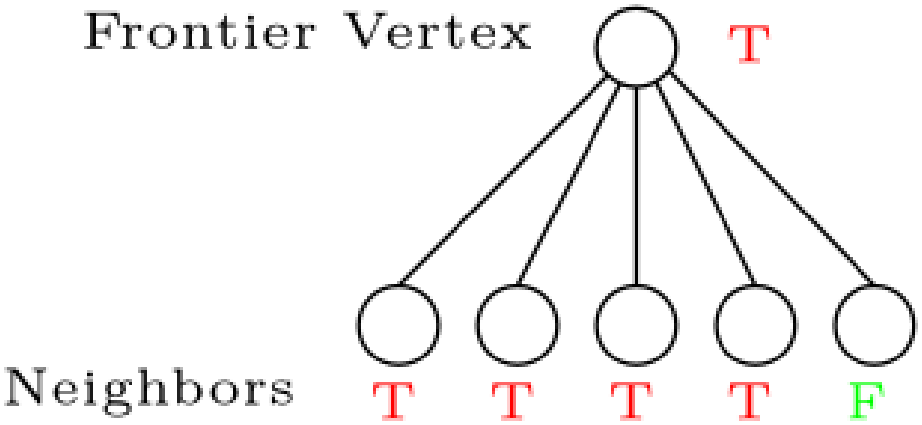

Figure: Top-down Approach

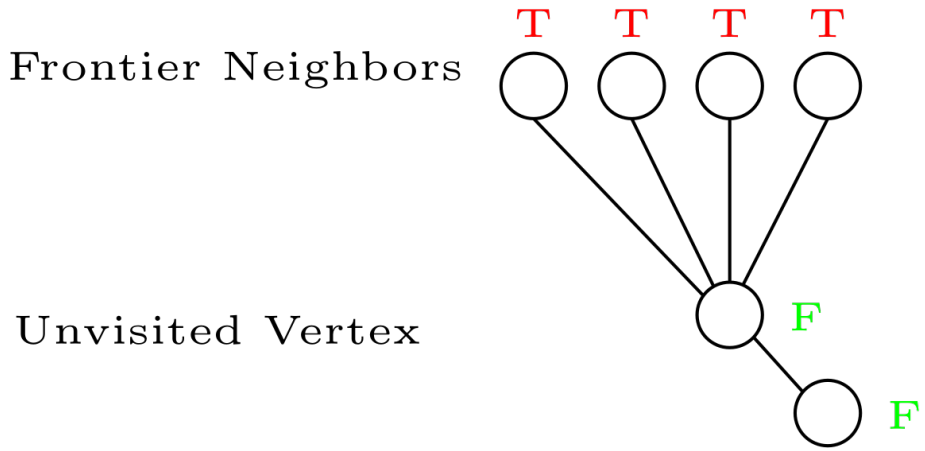

Figure: Bottom-up Approach

BFS-Hybrid†

- Hybrid approach in [1] switches between top-down and bottom-up approaches based on frontier edges, unvisited edges and frontier size.
- When frontier size is large, bottom-up approach reduces edge examination significantly for small diameter graphs.

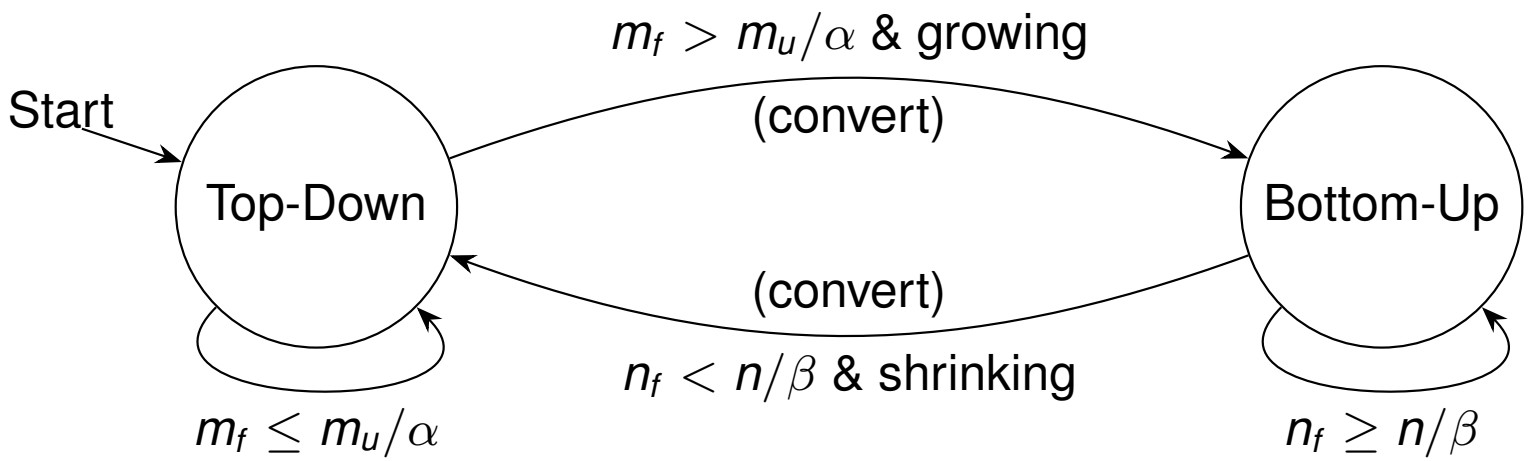

Figure: Switching criteria for $BFS - Hybrid^{\dagger}$ approach in [1]

BFS-Hybrid

- Switching criteria in [1] has to keep track of frontier edges.
- Our switching criteria is solely based on frontier size.

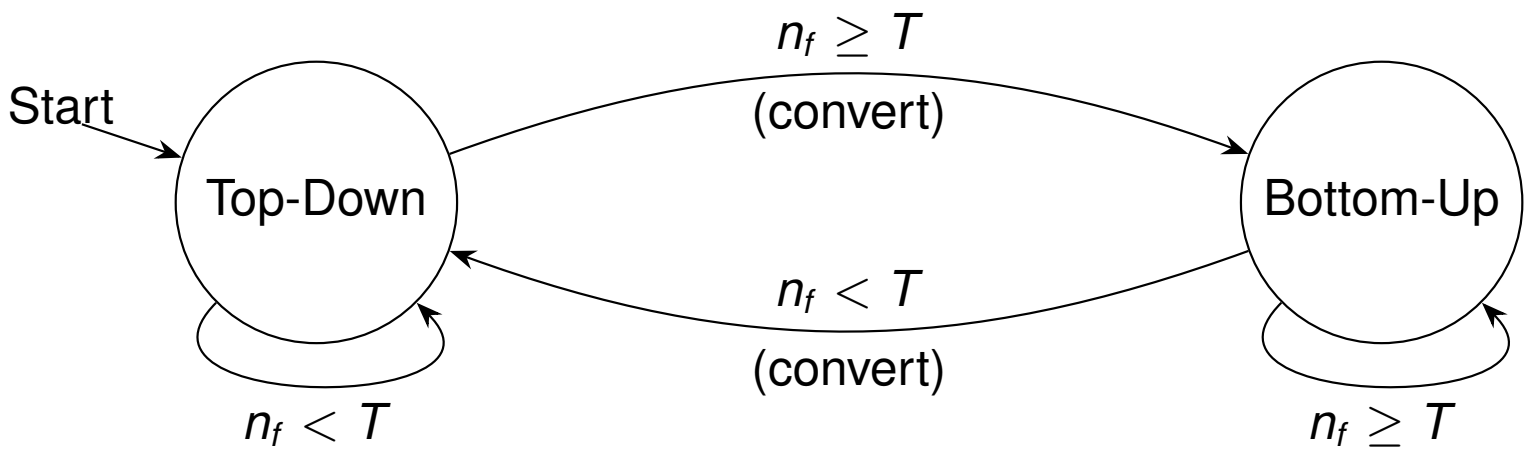

Figure: Our Switching criteria for $BFS - Hybrid$ approach, $T = 0.05 \times N$

Algorithm 3 BFS-Hybrid

```
1:  #pragma omp parallel
2:  while cfsz ≠ 0 do
3:      if is_top_down then
4:          /* Algorithm 1: BFS-NonAtomic */
5:      else
6:          #pragma omp for reduction(+:nxt_fsz)
7:          for all v ∈ V do
8:              if dist[v] == MAX_DIST then                ▷ if v is unvisited
9:                  /* Check if its neighbors are in the current frontier */
10:                 for all ngh ∈ neighbor[v] do
11:                     if curr_front_bm[ngh] then
12:                         dist[v] = depth + 1, nxt_fsz++
13:                         nxt_front_bm[v] = true
14:                         break
15:                     end if
16:                 end for
17:             end if
18:         end for
```

BFS-Hybrid (contd.)

```
19:     end if
20:     #pragma omp barrier
21:     #pragma omp single
22:     /* update curr_frontier, curr_front_bm, cfsz, and nxt_fsz */
23:     is_top_down = (cfsz < 0.05 × N) ? true : false
24:     depth++
25:     #pragma omp barrier
26:     if move_from_TD_to_BU then
27:         /* convert frontier array to curr_front_bm (bitmap) */
28:     else if move_from_BU_to_TD then
29:         /* convert curr_front_bm (bitmap) to curr_frontier (array) */
30:     end if
31:     lfsz = 0
32:     #pragma omp barrier
33: end while
```

BFS-VisitedBitmap

- Prior algorithms used `dist[]` array to check for unvisited vertices.
- Cache friendly data structure like bitmap can be used to achieve the above goal.

Algorithm 4 BFS-VisitedBitmap

```
1: /* Top-down phase: use bitmap to check if a vertex is visited */
2: ...
3: for all ngh ∈ neighbor[src] do
4:     if !visited_bm[ngh] then
5:         dist[dst] = dist[src] + 1
6:         lf[lfsz++] = dst
7:         visited_bm[dst] = true
8:     end if
9: end for
10: ...
```

BFS-VisitedBitmap(contd.)

```
11: // Bottom-up phase
12: ...
13: #pragma omp for nowait reduction(+:nxt_fsz)
14: for all v ∈ V do
15:    if !visited_bm[v] then
16:       for all ngh ∈ neighbor[v] do
17:          if curr_front_bm[ngh] then
18:             dist[v] = depth + 1, nxt_fsz + +
19:             nxt_front_bm[v] = true
20:             visited_bm[v] = true
21:             break
22:          end if
23:       end for
24:    end if
25: end for
26: ...
```

Other optimizations

1. Periodic flushing of local frontiers.
2. Prefetching and Vector instructions for neighbor checking loop in BFS-Hybrid.
3. Reducing number of barriers for large diameter graphs.
4. Launching threads only once.

भारतीय प्रौद्योगिकी संस्थान तिरुपति
IIT TIRUPATI

Evaluation

- BFS-Conventional achieves 22× speedup over serial BFS.

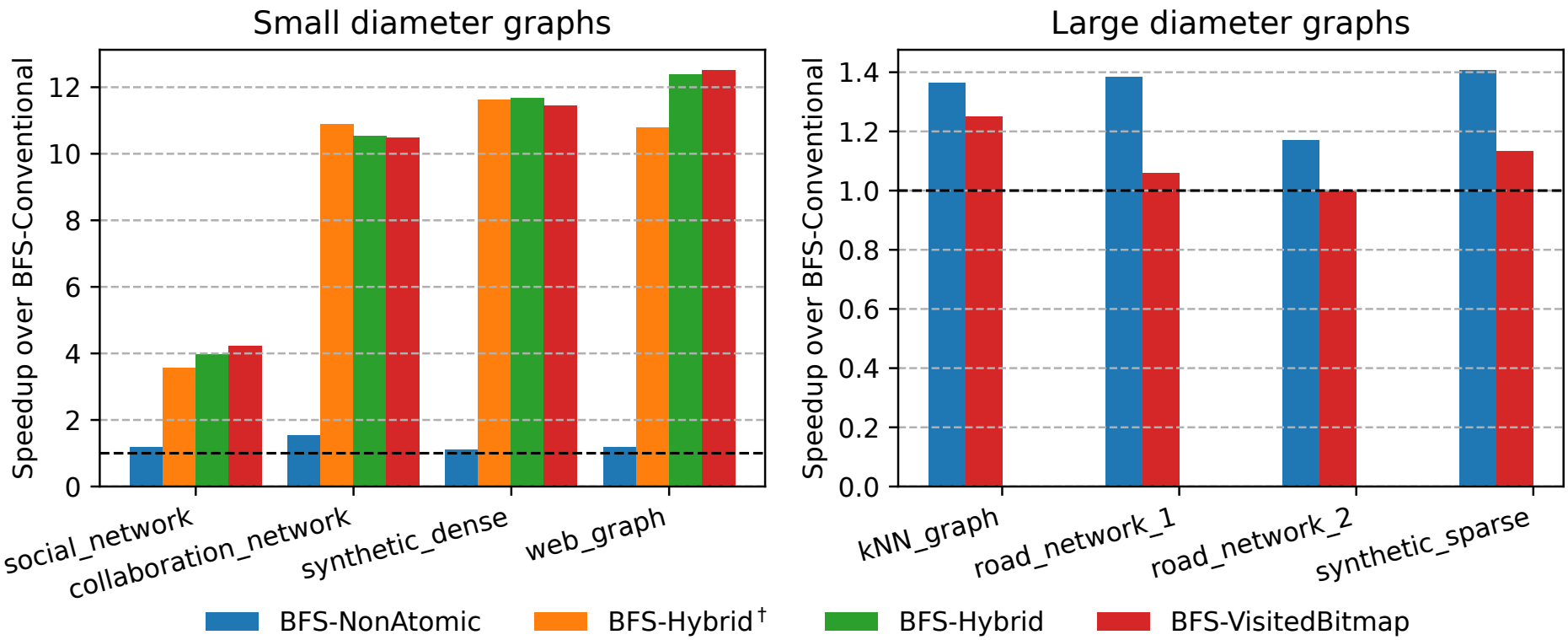

Figure: Performance comparison of BFS optimizations on speedcode [2]

.

Summary

1. We optimize BFS with non-atomic updates for distance values.
2. We analyze opportunities for non-atomic updates and their performance impact.
3. We propose a heuristic to switch between top-down and bottom-up BFS based on frontier size.
4. We use a bitmap-based approach to track visited vertices, reducing memory and improving cache locality.
5. On average, the static scheduling outperforms the dynamic scheduling on the SpeedCode platform[2].

भारतीय प्रौद्योगिकी संस्थान तिरुपति
IIT TIRUPATI

Github link

Figure: Github link

References I

Scott Beamer, Krste Asanović, and David Patterson.
Direction-optimizing breadth-first search.
In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, Washington, DC, USA, 2012. IEEE Computer Society Press.

Tim Kaler, Xuhao Chen, Brian Wheatman, Dorothy Curtis, Bruce Hoppe, Tao B. Schardl, and Charles E. Leiserson.
Speedcode: Software performance engineering education via the coding of didactic exercises.
In *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 391–394, 2024.

भारतीय प्रौद्योगिकी संस्थान तिरुपति
TIRUPATI

Thank You!

Questions? Suggestions? Comments?

भारतीय प्रौद्योगिकी संस्थान तिरुपति
TIRUPATI