Parallel Cluster-BFS and Applications to Shortest Paths

Authors: Letong Wang, Guy Blelloch, Yan Gu, Yihan Sun

Motivation: Multi-Source BFS

Applications:

- **All-Pairs Shortest Paths (APSP)**: Need distances between all vertex pairs
- **Distance Oracles**: Precompute distances for fast query answering
- **Low-diameter Decomposition**: Partition graphs based on distances

Challenges:

- Single BFS: O(m+n)
- Real applications need k = O(n) sources
- Multi-Source BFS: O(nm)

Background and Proposed Method

- **Ligra '13:** Parallel BFS with thread-level parallelism and directional optimization
- **Chan '12:** Cluster-BFS with sequential implementation with bit-parallelism only
    - Cluster-BFS: Running BFS from a cluster of k nearby vertices simultaneously
- **Akiba et al. '12:** Applied sequential cluster-BFS to 2-hop distance oracles, limited to star-shaped clusters (d=2).
    - 2-hop distance oracles: Precompute distances to answer exact shortest path queries via intermediate "hub" vertices
- **Wang et al. '25:** Combine Cluster-BFS with thread-level parallelism

Definitions & Preliminaries

$G = (V, E)$: the input graph. $n = |V|$ and $m = |E|$.

$S = \{s_1, ..., s_k\}$: the source cluster for the BFS.

k : the cluster size, i.e., $k = |S|$.

d : the diameter of the cluster.

w : the length of a word in bits. $w = \Omega(n)$.

D : the diameter of the graph.

$\delta(u, v)$: the shortest distance between u and v.

Table 1: Notations in the paper.

Compute model: Fork and join with binary forking

Unit costs:

- Compare and Swap(p, v_old, v_new)
- Fetch and Or(p, v_new)

Cluster-BFS: Fundamental Observation:

Fact 3.1: Distance Bound Between Nearby Sources

On an unweighted graph, if the distance between vertices s_1 and s_2 is d, then for any vertex $v \in V$:

$$|\delta(s_1,v) - \delta(s_2,v)| \le d$$

Corollary 3.1: Cluster Distance Range

Given a set S of vertices with diameter ≤ d, for any vertex $v \in V$:

$$\max_{s \in S} \delta(s,v) - \min_{s \in S} \delta(s,v) \le d$$

Key Implication for BFS

All sources in cluster S will visit vertex v within at most (d+1) consecutive BFS rounds!

Cluster Distance Representation

Compact Distance Storage:

For vertex v and cluster S:

- **Δ_v** = min_{s∈S} δ(s,v) (smallest distance from any source to v)
- **$S_v[i]$** = {s ∈ S | δ(s,v) = Δ_v + i} for i ∈ [0,d]

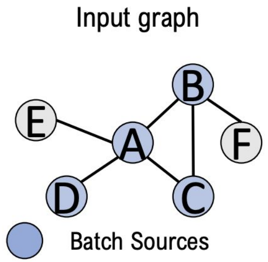
Cluster Distance Vector: ⟨S_v[0..d], Δ_v⟩

Space Efficiency:

- Standard BFS: O(|S|) words per vertex
- Cluster-BFS: O(d) words per |S| vertices when |S| = O(w)

Key elements to highlight:

- Batch set S = {A,B,C,D} with 4-bit representations
- Δ_v tracks minimum distance to each vertex
- $S_v[i]$ uses bit-subsets to track which sources have distance Δ_v + i
- Example: For vertex F, SF[1] = 1010 represents {A,C} have distance Δf + 1 = 2
-



Batch sources	A	B	C	D
Bitwise vector	1	1	1	1

	Δ_v	$S_v[0]$	$S_v[1]$	$S_v[2]$
A	0	1000	0111	0000
B	0	0100	1010	0001
C	0	0010	1100	0001
D	0	0001	1000	0110
E	1	1000	0111	0000
F	1	0100	1010	0001

$\{B\}$ $\{A, C\}$ $\{D\}$

Figure 2: Illustration of bitwise representation. The batch set S is $\{A, B, C, D\}$. 4-bit bit-subsets are used to represent subsets of S. Δ_v is the smallest shortest distance from any vertex in S to v. The subset $S_v[i]$ is defined as $\{s \in S | \delta(s, v) = \Delta_v + i\}$.

Bit Operations Enable Efficiency:

- Union/intersection of source sets via bitwise OR/AND
- O(k/w + 1) work for set operations on k sources

Parallel Cluster-BFS pseudocode

Algorithm 1: Cluster-BFS search from S

Input:
A graph $G = (V, E)$, a cluster $S \subseteq V$ with diameter d
Output:
cluster distance vectors $\langle S_v[0..d], \Delta_v \rangle$ for all $v \in V$.
Maintains:
i: the current round number, initialized to 0
$S_{seen}[\cdot], S_{next}[\cdot]$: array of bit-subset for each $v \in V$
$r[v]$: the lastest round v is in the frontier
$\mathcal{F}_i$: frontier vertices in round i

// Initialization
ParallelForEach $v \in V$ **do**
    $S_{seen}[v] \leftarrow \emptyset$, $S_{next}[v] \leftarrow \emptyset$
    $\Delta_v \leftarrow \infty$
    $r[v] \leftarrow \infty$
for $s \in S$ **do** $S_{seen}[s] \leftarrow \{s\}$
$i \leftarrow 0$
$\mathcal{F}_0 \leftarrow S$
// Traversing
while $\mathcal{F}_i \neq \emptyset$ **do**
    ParallelForEach $u \in \mathcal{F}_i$ **do**
        $S_{new} \leftarrow S_{next}[u] \setminus S_{seen}[u]$
        if $\Delta_u = \infty$ **then** $\Delta_u \leftarrow i$
        $S_u[i - \Delta_u] \leftarrow S_{new}$
        $S_{seen}[u] \leftarrow S_{seen}[u] \cup S_{new}$
    ParallelForEach $u \in \mathcal{F}_i$ **do**
        ParallelForEach $v \in N(u)$ and $i - \Delta_v < d$ **do**
            if FETCH_AND_OR$(S_{next}[v], S_{seen}[u])$
                if COMPARE_AND_SWAP$(r[v], r[v], i)$
                    $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_{i+1} \cup \{v\}$
    $i \leftarrow i + 1$
return $\langle S_v[1..d], \Delta_v \rangle$ for all $v \in V$

Complexity Analysis

Lemma 3.1: Given bit-subset subsets S_1 and S_2 of a set S with size k, we can compute the bit-subset representation of $S_1 \cup S_2$, $S_1 \setminus S_2$ in $O(k/w + 1)$ work and $O(\log(k/w) + 1)$ span

Proof:

- Need $\lceil k/w \rceil$ words to represent k bits
- Each word operation (OR, AND, NOT) takes constant time
- Span is $O(\log(k/w))$ due to binary forking parallelization

Theorem 3.1: Given a set S of k vertices with diameter d, we can compute the cluster distance vector from S to every vertex in V in $O(dm(k/w + 1))$ work and $O((D + d)\log n)$ span

Proof:

- **Work:** Each vertex appears in frontier $\leq d$ times → each edge processed $\leq d$ times → total edge operations = $O(dm)$. Each operation costs $O(k/w + 1)$ by Lemma 3.1
- **Span:** $\leq D + d$ rounds total. Each round: $O(\log n)$ for parallel task generation + $O(\log(k/w))$ for bit operations = $O(\log n)$

Key Insight: When $k = \Theta(w)$, work becomes $O(dm)$ - **asymptotically same as single BFS!**

Applications

Distance Oracle: An index designed to answer shortest distances between vertices efficiently

- Alternative to computing distance on-the-fly (e.g., running BFS)
- Trade preprocessing time/space for fast query response

1. **Exact Distance Oracle (EDO)** - 2-Hop Labeling
   - Select hubs for each vertex and precompute distances; query through hub intersections
   - **PLL Approach:** Processes vertices in predetermined order, running BFS from each to build hub labels with pruning to minimize index size
   - **With C-BFS:** Replace sequential C-BFS in Pruned Landmark Labeling with parallel C-BFS and parallelize pruned BFS phase (batching pruned BFS runs)
2. **Approximate Distance Oracle (ADO)** - Landmark Labeling
   - Select landmarks and precompute distances; query estimates shortest path through nearest landmarks
   - Selection heuristic: Prioritize vertices with high degree
   - **With C-BFS:** Use clusters as landmarks instead of single vertices, processed with parallel C-BFS to get w times more landmarks

Experiments

Content:

- **Machine:** 96-core (192 hyperthreads) Intel Xeon Gold 6252, 1.5TB RAM
- **Implementation:** C++ with ParlayLib for parallelism
- **Datasets:** 18 undirected graphs (social and web networks)
- **Baselines:**
  - Ligra (parallel BFS with thread-level parallelism only)
  - AIY (sequential C-BFS with bit-level parallelism only)

Microbenchmarks for Cluster-BFS

Dataset	Graph Information: n	m	Notes	Seq-BFS Time(s)	Related Work: AIY	Ligra	Parallel C-BFS: Final	Time(s)	Self-Speedup: C-BFS	Ligra
EP	75.9K	811K	Epinions1 [3]	0.18	20.6×	4.02×	102×	0.002	4.39×	9.44×
SLDT	77.4K	938K	Slashdot [25]	0.21	18.8×	3.91×	94.1×	0.002	3.47×	9.55×
DBLP	317K	2.10M	DBLP [44]	0.77	20.3×	6.22×	183×	0.004	10.2×	17.1×
YT	1.13M	5.98M	com-youtube [44]	3.30	22.9×	17.8×	445×	0.007	20.6×	31.6×
SK	1.69M	22.2M	skitter [15, 36]	6.56	21.0×	30.4×	496×	0.013	26.7×	33.1×
IN04	1.38M	27.6M	in_2004 [13, 14]	4.08	20.9×	4.00×	171×	0.024	10.1×	17.8×
LJ	4.85M	85.7M	soc-LiveJournal1 [3]	39.8	23.7×	61.8×	1017×	0.039	47.7×	53.9×
HW	1.07M	112M	hollywood_2009 [14, 36]	18.7	20.9×	89.7×	928×	0.020	32.4×	48.7×
FBUU	58.8M	184M	socfb-uci-uni [34, 36, 41]	268	32.0×	49.6×	973×	0.276	54.4×	52.8×
FBKN	59.2M	185M	socfb-konect [34, 36, 41]	176	27.9×	38.8×	712×	0.247	53.1×	51.8×
OK	3.07M	234M	com-orkut [44]	61.6	19.8×	102×	1119×	0.055	49.0×	65.4×
INDO	7.41M	301M	indochina [11, 13, 36]	38.8	21.9×	12.4×	452×	0.086	25.7×	35.9×
EU	11.3M	521M	eu-2015-host [12–14]	119	23.9×	26.6×	821×	0.145	18.5×	41.3×
UK	18.5M	523M	uk-2002 [13, 14]	91.8	22.7×	30.7×	687×	0.134	42.1×	46.7×
AR	22.7M	1.11B	arabic [13, 14]	147	22.5×	10.7×	461×	0.319	18.0×	33.8×
TW	41.7M	2.41B	Twitter [23]	861	20.6×	157×	856×	1.006	56.3×	60.2×
FT	65.6M	3.61B	Friendster [44]	2084	20.4×	187×	813×	2.563	59.4×	64.6×
SD	89.2M	3.88B	sd_arc [29]	1898	25.0×	80.3×	945×	2.008	55.7×	62.5×
GeoMean				32.0	22.4×	27.0×	500×	0.064	24.8×	35.4×

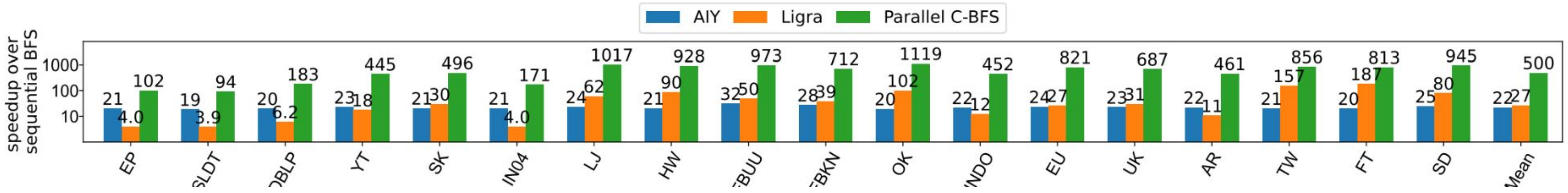
Table 2: Tested graphs and microbenchmarks on different BFS algorithms from a cluster of vertices with size 64. The numbers followed by '×' are speedups, higher is better. Others are running time, lower is better. The columns "AIY", "Ligra" in related work and "Final" show the speedup over the "Seq-BFS". "AIY" is referred to sequential C-BFS from [1], "Ligra" is referred to parallel single BFS [38], and "Final" is referred to our parallel C-BFS.

Parameters:

- **k = 64**
- **d = 2**

Observations:

- **Thread-level Parallelism:** 3.9×–187×, highly graph-dependent (better on larger/dense graphs)
- **Bit-level Parallelism:** Consistent ~20× improvement
- **Synergistic Effect:** Both parallelism types work well together



Influence of number of processors:

- Nearly linear speedup for most of the graphs

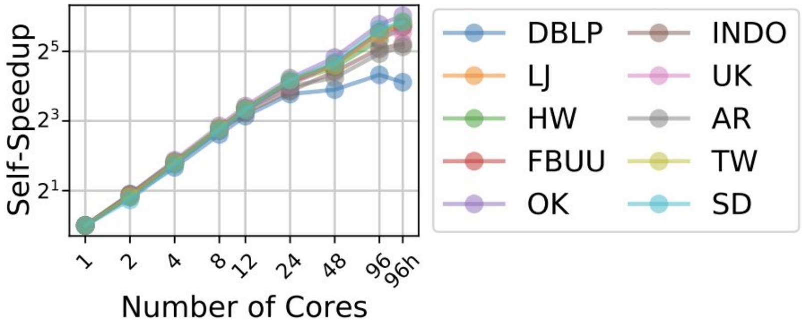


Figure 4: The scalability curve on different number of processors for C-BFS. The y-axis is the self speedup. The C-BFS running on one core is always 1. The x-axis is the number of cores. 96h represents 96 cores with hyperthreads.

Influence of Cluster Diameter d:

- **Performance decreases as d increases** (work ∝ d)
- d=2 provides best overall performance for applications
- d>2 gives flexibility but higher overhead

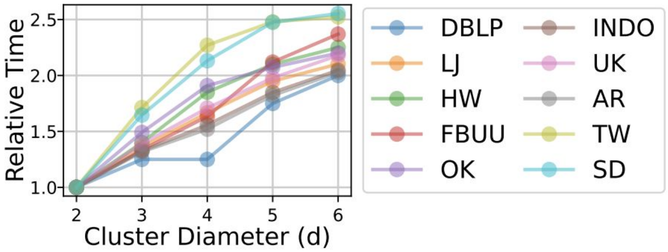


Figure 5: The running time of C-BFS on various cluster diameter d. The y-axis shows the relative running time over $d = 2$. The x-axis shows the cluster diameter d.

Applications Results

2-Hop Distance Oracle

- C-BFS adds slightly more labels than the original AIY due to batch BFS pruning, where vertices in the same batch cannot share labels.
- The overhead is small: ≤ 8% more labels and ≤ 5% increase in index size across all tested graphs.

Performance Improvements:

- **9-36× speedup** over sequential AIY algorithm
- Can process **much larger graphs** than sequential version
- Maintains exact distance guarantees

	r	Alg.	avg. labs	Index Size	Running Time C-BFS	P-BFS	Total
SK	64	AIY	123	2.68	32.9	265	299
		Ours	126	2.71	1.05	15.0	16.2
		Spd			31.3×	17.7×	18.5×
HW	64	AIY	2237	12.2	103	11010	11115
		Ours	2280	12.4	1.26	303	304
		Spd			82.3×	36.4×	36.5×
INDO	64	AIY	323	18.7	246	3740	4051
		Ours	349	19.6	5.63	421	428
		Spd			43.7×	9.02×	9.46×
EU	64	Ours	944	61.0	9.92	1385	1396
LJ	512	Ours	2585	97.7	23.0	2718	2742
AR	256	Ours	989	197	72.7	7690	7767
OK	2048	Ours	6881	198	119	13407	13527

Table 7: Performance on an exact distance oracle based on pruned landmark labeling. "AIY": the sequential implementation from [2]. r: the number of clusters used in C-BFS. Index sizes are in GB. "C-BFS": time for cluster-BFS. "P-BFS": time for pruned BFS. "Spd": speed-up of ours over AIY. For #labels/vertex, index size, and running time, lower is better. See more details in appendix D.1.

Landmark Labelling

Memory Budget Analysis (1024 bytes/vertex):

- **Plain LL:** 1024 single landmarks
- **w=64:** 60 clusters (60×64 = 3840 total landmarks)
- **w=8:** 341 clusters (341×8 = 2728 total landmarks)

Performance Insights:

- **Time improvement:** Consistent across all graphs
- **Accuracy improvement:** More landmarks (even if correlated) help
- **Sweet spot:** w=16 or w=32 provide good time/accuracy balance
- **w=8 vs w=64 trade-off:**
  a. w=8: Better accuracy (more independent clusters)
  b. w=64: Faster preprocessing (fewer total clusters)

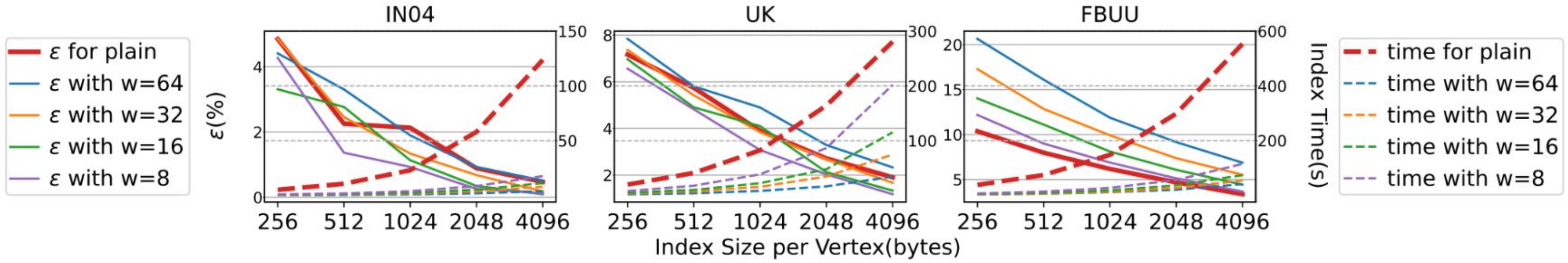
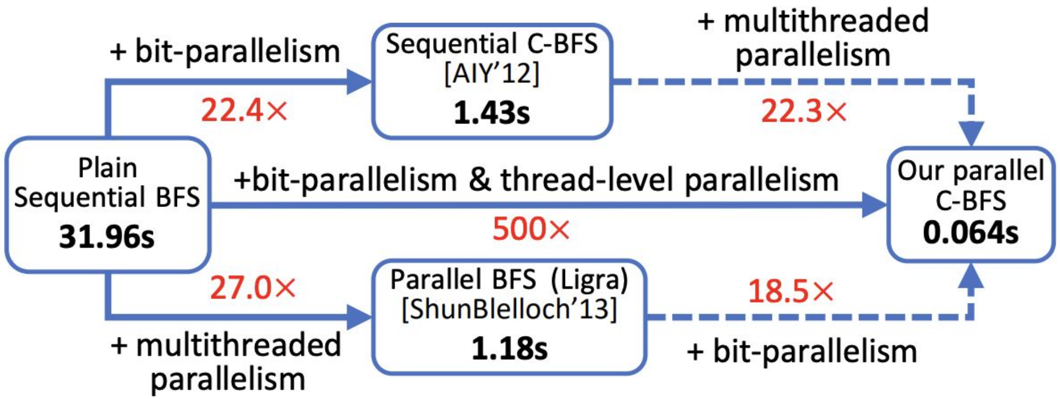


Figure 6: Tradeoffs between index size and distortion/construction time The x-axis is the memory limits per vertex in bytes, and is in log-scale. The y-axis on the left shows the $(1 + \epsilon)$ distortion. The y-axis on the right shows the preprocessing time. For both preprocessing time and distortion, lower is better. For the algorithms compared here, 'plain' is the regular LL, others are the C-BFS-based LL that choose clusters with size w as landmarks.

Summary


+ bit-parallelism
22.4×
Sequential C-BFS
[AIY'12]
1.43s
+ multithreaded parallelism
22.3×
Plain
Sequential BFS
31.96s
+bit-parallelism & thread-level parallelism
500×
Our parallel
C-BFS
0.064s
27.0×
+ multithreaded parallelism
Parallel BFS (Ligra)
[ShunBlelloch'13]
1.18s
18.5×
+ bit-parallelism

Future Directions

1. **Weighted graph extensions** for broader applicability
2. **Dynamic graph support** for evolving graphs
3. **GPU acceleration** for even higher performance
4. **Other applications** that could benefit from Parallel Cluster-BFS

Discussion Questions

Algorithm Design & Extensions

1. What other graph problems could benefit from the cluster-BFS approach?
2. How might this approach be adapted for dynamic graphs where edges are added/removed?
3. When would you prefer this approach over other distance oracle methods like contraction hierarchies?

Scalability & Limitations

4. What factors limit the scalability of this approach to even larger graphs?
5. What are the fundamental bottlenecks that weren't deeply analyzed in the paper?

Paper Quality & Research Context

6. How effective is the paper's flow and explanation of technical concepts?
7. Are the experiments comprehensive enough, or what's missing?
8. Why is there a 10+ year gap between related work, and does this affect the contribution's significance?

Practical Impact

9. Given the limited client base for massive graph processing (mainly tech giants with abundant resources), what are the key challenges for industrial adoption of this approach, and is the implementation complexity justified by the performance gains in real-world scenarios?