Direction-Optimizing Breadth-First Search

Scott Beamer, Krste Asanovic, David Patterson

Presented by Siddhartha Jayanti
Problem

- Speed-Up Parallel BFS Computation
  - Mark Visited Vertices
  - Compute BFS TREE
Breadth First Search Algorithm

**Fig. 1.** Conventional BFS Algorithm

```
function breadth-first-search(vertices, source)
    frontier ← {source}
    next ← {}
    parents ← [-1,-1,...-1]
    while frontier ≠ {} do
        top-down-step(vertices, frontier, next, parents)
        frontier ← next
        next ← {}
    end while
    return tree
```

**Fig. 2.** Single Step of Top-Down Approach

```
function top-down-step(vertices, frontier, next, parents)
    for v ∈ frontier do
        for n ∈ neighbors[v] do
            if parents[n] = -1 then
                parents[n] ← v
                next ← next ∪ {n}
            end if
        end for
    end for
```
Technique

- Reduce EDGE Traversals
  - Worst-Case: $O(n + m)$
  - Best-Case: $O(n)$
- Can’t always do better
  - e.g. Path
  - How about common practical case?
- Motivation
  - Social Network
  - Exponential Law for degrees
Main Concept #1: Frontier
Main Concept #1: Frontier

Frontier at Round 2
Main Concept #2: Classification of Nodes

V  valid parent
P  peer
F  failed child
C  claimed child
Main Concept #2: Classification of Nodes

- **V**: valid parent
- **P**: peer
- **F**: failed child
- **C**: claimed child
Main Concept #3: Social Network Structure
Empirical Justification

In Opportunity Zone:

- Few Claimed Children
- Lots of Failed Children

Fig. 3. Breakdown of edges in the frontier for a sample search on kron27 (Kronecker generated 128M vertices with 2B undirected edges) on the 16-core system.
Empirical Justification

In Ramp Up:
- Lots of Claimed Children

In Opportunity Zone:
- Lots of Failed Children

In Tail Zone:
- Most parents are Valid

Fig. 4. Breakdown of edges in the frontier for a sample search on Kron27 (Kronecker generated 128M vertices with 2B undirected edges) on the 16-core system.
Breadth First Search: Bottom-Up Step

**Fig. 1.** Conventional BFS Algorithm

```plaintext
function breadth-first-search(vertices, source)
    frontier ← \{source\}
    next ← \{
    parents ← [-1,-1,\ldots-1]
    while frontier ≠ \{
        top-down-step(vertices, frontier, next, parents)
        frontier ← next
        next ← \{
    end while
    return tree
```

**Fig. 2.** Single Step of Top-Down Approach

```plaintext
function top-down-step(vertices, frontier, next, parents)
    for \(v \in\) frontier do
        for \(n \in\) neighbors[\(v\)] do
            if parents[\(n\)] = -1 then
                parents[\(n\)] ← \(v\)
                next ← next \cup \{n\}
            end if
        end for
    end for
end for
```
Breadth First Search: Bottom-Up Step

function bottom-up-step(vertices, frontier, next, parents)
    for v ∈ vertices do
        if parents[v] = -1 then
            for n ∈ neighbors[v] do
                if n ∈ frontier then
                    parents[v] ← n
                    next ← next ∪ {v}
                    break
                end if
            end for
        end if
    end for
end for

Fig. 5. Single Step of Bottom-Up Approach
Comparison

Advantages of Top-Down:
- Frontier is small & lots of neighbors

Advantages of Bottom-Up:
- Frontier large compared to remaining vertices
- Can stop search early
- No write-contention
Heuristic

\[ n_f = \# \text{ vertices in frontier} \]

\[ m_f = \# \text{ edges to check from the frontier} \]

\[ m_u = \# \text{ edges to check from unexplored vertices} \]

\( \alpha, \beta \) - tuning parameters

\[ m_f > \frac{m_u}{\alpha} = C_{TB} \]

\[ n_f < \frac{n}{\beta} = C_{BT} \]

Fig. 7. Control algorithm for hybrid algorithm. (convert) indicates the frontier must be converted from a queue to a bitmap or vice versa between the steps. Growing and shrinking refer to the frontier size, and although they are typically redundant, their inclusion yields a speedup of about 10%.
Hybrid-heuristic is robust to tuning $\alpha$

Fig. 8. Performance of hybrid-heuristic on each graph relative to its best on that graph for the range of $\alpha$ examined.
Hybrid-heuristic is robust to tuning $\beta$

Fig. 9. Performance of hybrid-heuristic on each graph relative to its best on that graph for the range of $\beta$ examined.
Performance of Method (dark purple vs. light blue)

Hybrid is 2 to 8 times as fast as original top-down-check algorithm.

Fig. 10. Speedups on the 16-core machine relative to Top-down-check.
Additional Threads Help Speed Up - Hyperthreading Doesn’t
Conclusion

- Works well if **Low Effective Diameter**

- Top Down -> Bottom Up -> Top Down

- High-diameter graphs don’t benefit from bottom-up, but are easier to parallelize

Fig. 12. Breakdown of time spent per search.
Questions

- How important is parallelism for this idea?

- Are there other graph problems that could benefit from this type of thought?
  - Could parallel Dijkstra use this idea?
  - Something else?

- What are experiments that you’d like to see that were missing?

- When might normal BFS be better than the hybrid algorithm?
Main Concept #1: Frontier
Main Concept #1: Frontier