The More the Merrier: Efficient Multi-Source Graph Traversal

Then et al.

Presenter: Sai Sameer Pusapaty
Background

- Lot of information held via graphs
  - Social networks
  - Road Networks
  - Comp bio
- Graph analytics to comprehend relationships
  - Often requires multiple graph traversals (BFS)
  - Graph centrality
  - All pairs shortest paths
- Usually do a BFS from each vertex on the graph
Small World Assumption

- “Distance between any two vertices is very small compared to size of the graph”
- “Number of vertices discovered in each iteration grows rapidly”
- I.e. only a few iterations of BFS to traverse the entire graph
- This assumption is not unreasonable:
  - 92% of Facebook users are connected by only 5 steps
  - (“Four Degrees of Separation”, L. Backstrom et al. 2012)
- Wikis, WWW, gene networks, electrical power grids
BFS (Single Traversal)

- Single source
- Keep track of unexplored neighbors
- Maintain levels of exploration
  - Max num of levels = diameter
  - Small-world assumption
- Optimizations:
  - Parallel BFS
  - Bottom up approach (direction optimized)

\[
\text{Work} = \Theta(n+m)
\]
\[
\text{Depth} = O(\Delta \log m)
\]

Listing 1: Textbook BFS algorithm.

1. Input: \( G, s \)
2. \( \text{seen} \leftarrow \{s\} \)
3. \( \text{visit} \leftarrow \{s\} \)
4. \( \text{visitNext} \leftarrow \emptyset \)
5. 
6. \( \text{while visit} \neq \emptyset \)
7. \( \quad \text{for each } v \in \text{visit} \)
8. \( \quad \quad \text{for each } n \in \text{neighbors}_v \)
9. \( \quad \quad \quad \text{if } n \notin \text{seen} \)
10. \( \quad \quad \quad \quad \text{seen} \leftarrow \text{seen} \cup \{n\} \)
11. \( \quad \quad \quad \quad \text{visitNext} \leftarrow \text{visitNext} \cup \{n\} \)
12. \( \quad \quad \quad \text{do BFS computation on } n \)
13. \( \quad \text{visit} \leftarrow \text{visitNext} \)
14. \( \quad \text{visitNext} \leftarrow \emptyset \)
Motivation for MS-BFS

- Goal is to optimize execution of multiple independent BFSs
  - Common graph analytics would benefit from this
- Related work just on improving single execution of BFS
- Compared to old method of repetitive BFSs traversals...
  - We want better memory locality since the same vertices are discovered and explored -> tldr; fewer cache misses
  - We want better resource management as the # of BFSs increase and # of cores increase
  - We want to avoid synchronization due to its overhead
Overview of MS-BFS

- Very similar to a normal BFS
- Each vertex maintains $seen$, holds BFSs that have already visited it
- $Visit$ contains a tuple of the vertex and BFS that is currently visiting it -- unioned together
- Neighbors not seen before are explored
- **Main Idea:** BFS that share common sub-traversal travel together

Listing 2: The MS-BFS algorithm.

```plaintext
Input: $G, \mathcal{B}, S$

1. $seen_{s_i} \leftarrow \{b_i\}$ for all $b_i \in \mathcal{B}$
2. $visit \leftarrow \bigcup_{b_i \in \mathcal{B}} \{(s_i, \{b_i\})\}$
3. $visitNext \leftarrow \emptyset$
4. 
5. while $visit \neq \emptyset$
6.   for each $v$ in $visit$
7.     $\mathcal{B}_v \leftarrow \emptyset$
8.     for each $(v', \mathcal{B}') \in visit$ where $v' = v$
9.       $\mathcal{B}_v \leftarrow \mathcal{B}_v \cup \mathcal{B}'$
10. for each $n \in neighbors_v$
11.     $D \leftarrow \mathcal{B}_v \setminus seen_n$
12.     if $D \neq \emptyset$
13.         $visitNext \leftarrow visitNext \cup \{(n, D)\}$
14.         $seen_n \leftarrow seen_n \cup D$
15.         do BFS computation on $n$
16.     do BFS computation on $n$
17.     $visit \leftarrow visitNext$
18.     $visitNext \leftarrow \emptyset$
```
An Example

Figure 2: An example of the MS-BFS algorithm, where vertices 3 and 4 are explored once for two BFSs.
Improving with Bit Operations

- In practice set unions and differences are expensive
- Use bit operations instead
- `seen` is a bit-field for each vertex `v`, such that if the `ith` bit is 1, that means the `ith` BFS has already seen `v`
- Similarly `visit` and `visitNext` are set up s.t. If the `ith` bit is 1, then `v` still needs to be explored by the `ith` BFS
- Set operations become binary operations
- Store these three bitfields in arrays for constant time access -> `visit_v = visit[v]`

```
Listing 3: MS-BFS using bit operations.
1  Input: G, B, S
2  for each b_i ∈ B
3      seen[s_i] ← 1 << b_i
4      visit[s_i] ← 1 << b_i
5  reset visitNext
6  while visit ≠ Ø
7      for i = 1, ..., N
8          if visit[v_i] = B_Ø, skip
9              for each n ∈ neighbors[v_i]
10                 D ← visit[v_i] & ~seen[n]
11                   if D ≠ B_Ø
12                      visitNext[n] ← visitNext[n] | D
13                      seen[n] ← seen[n] | D
14                 do BFS computation on n
15      visit ← visitNext
16  reset visitNext
```
An Example

Figure 3: An example showing the steps of MS-BFS when using bit operations. Each row represents the bit field for a vertex, and each column corresponds to one BFS. The symbol X indicates that the value of the bit is 1.
Aggregated Neighbor Processing

- Still some bad memory access
- `seen` has a lot of random accesses which lead to cache misses
- ANP first collects all the vertices needed to be explored in the next level (lines 8-11)
- `seen` is updated in batch
- Improvement include
  - Fewer calls to `seen` (once per discovered vertex)
  - Thus, fewer iterations of BFS computation
  - Memory access is sequential
- Direction Optimized Traversal, prefetching, max sharing heuristic

```
Listing 4: MS-BFS algorithm using ANP.
1  Input: G, B, S
2  for each bᵢ ∈ B
3      seen[ sᵢ ] ← 1 << bᵢ
4  visit[ sᵢ ] ← 1 << bᵢ
5  reset visitNext
6  while visit ≠ Ø
7      for i = 1,...,N
8          if visit[vᵢ] = B₀, skip
9              for each n ∈ neighbors[vᵢ]
10                 visitNext[n] ← visitNext[n] | visit[vᵢ]
11      for i = 1,...,N
12          if visitNext[vᵢ] = B₀, skip
13              visitNext[vᵢ] ← visitNext[vᵢ] & ¬seen[vᵢ]
14              seen[vᵢ] ← seen[vᵢ] | visitNext[vᵢ]
15              if visitNext[vᵢ] ≠ B₀
16                  do BFS computation on vᵢ
17          visit ← visitNext
18  reset visitNext
```
Experimental Results

Figure 4: Data size scalability results.

Figure 5: Multi-core scalability results.
More Results

Figure 6: BFS count scalability results.

Figure 7: Speedup achieved by cumulatively applying different tuning techniques to MS-BFS.
Relative Speedups

<table>
<thead>
<tr>
<th>Graph</th>
<th>T-BFS</th>
<th>DO-BFS</th>
<th>MS-BFS</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDBC 1M</td>
<td>2:15h</td>
<td>0:22h</td>
<td>0:02h</td>
<td>73.8x, 12.1x</td>
</tr>
<tr>
<td>LDBC 10M</td>
<td>*259:42h</td>
<td>*84:13h</td>
<td>2:56h</td>
<td>88.5x, 28.7x</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>*32:48h</td>
<td>*12:50h</td>
<td>0:26h</td>
<td>75.4x, 29.5x</td>
</tr>
<tr>
<td>Twitter (1M)</td>
<td>*156:06h</td>
<td>*36:23h</td>
<td>2:52h</td>
<td>54.6x, 12.7x</td>
</tr>
</tbody>
</table>

*Execution aborted after 8 hours; runtime estimated.
Strengths

- Outperforms T-BFS and DO-BFS on a single core (main goal)
- Scales well with an increasing number of cores
- Generally scales well even as the number of BFSs increase
- Paper provides further improvements which experimentally did well
- Works well on real life graphs
- Can be parallelized naturally (no immediate barriers -- kind of)
Weaknesses

- Some limitations if the number of BFSs increase past register width
  - Paper proposes some alternatives
  - Parallelizing, Using multiple registers, running many instances
  - Performance/Memory tradeoff
- Graph set is limited to those that follow the small world assumption
- Memory overhead with large graphs to store BFS states at each vertex
- Provides the benefit to vertices that multiple BFSs access on the same level
  - No memory if the vertices have already been accessed before
  - Potential for further decrease in computation
Future Work

- Look at parallelizing at the frontier level
- Adapting MS-BFS for distributed environments and GPUs
- Apply it to other graph analytics algorithms
- Testing MS-BFS on various graph types
- New heuristics to maximize sharing
Discussion

- What are some other strengths and weaknesses you see in MS-BFS?
- Can MS-BFS generalize to other graphs besides small-world graphs?
- Do you have your own thoughts for improvement of future extensions?