The More the Merrier: Efficient Multi-Source Graph Traversal

Authors: Manuel Then, Moritz Kaufmann, et al

6.827 Paper Presentation

Presenter: Edmund Williams

February 2022
Table of Contents

1 Problem

2 Solution Intuition

3 Algorithm + walk-through

4 Iteration 2, usage of bitmaps

5 Iteration 3, optimizing cache misses

6 Experimental Data
Problem: How To Make BFS Faster

Previous Ideas

- Parallel BFS implementations
- Bottom-Up approach (Beamer et al.)

New Idea

- Most applications require more than a single BFS traversal
- Instead of making one BFS faster, can we make batches of BFS traversals run faster?
Problem: How To Make BFS Faster

<table>
<thead>
<tr>
<th>Previous Ideas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel BFS implementations</td>
</tr>
<tr>
<td>Bottom-Up approach (Beamer et al.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>New Idea</th>
</tr>
</thead>
<tbody>
<tr>
<td>Most applications require more than a single BFS traversal</td>
</tr>
<tr>
<td>Instead of making one BFS faster, can we make batches of BFS traversals run faster?</td>
</tr>
</tbody>
</table>
Problem: How To Make BFS Faster

Previous Ideas
- Parallel BFS implementations
- Bottom-Up approach (Beamer et al.)

New Idea
- Most applications require more than a single BFS traversal
- Instead of making one BFS faster, can we make batches of BFS traversals run faster?
Problem: How To Make BFS Faster

Previous Ideas

- Parallel BFS implementations
- Bottom-Up approach (Beamer et al.)

New Idea

- Most applications require more than a single BFS traversal
- Instead of making one BFS faster, can we make batches of BFS traversals run faster?
Problem: How To Make BFS Faster

Previous Ideas
- Parallel BFS implementations
- Bottom-Up approach (Beamer et al.)

New Idea
- Most applications require more than a single BFS traversal
- Instead of making one BFS faster, can we make batches of BFS traversals run faster?
Intuition

- Due to the small-world principle most real large graphs have a relatively small diameter compared to their size. Because of this most vertices are explored within a few steps of the BFS traversal.

- Concurrent BFS traversals are likely to have a large overlap of what vertices they are exploring within a single step of a BFS traversal.

- Is there a way to efficiently store this overlap instead of each BFS maintaining their own data structures?

Figure 1: Percentage of vertex explorations that can be shared per level across 512 concurrent BFSs.
Table of Contents

1. Problem

2. Solution Intuition

3. Algorithm + walk-through

4. Iteration 2, usage of bitmaps

5. Iteration 3, optimizing cache misses

6. Experimental Data
Listing 2: The MS-BFS algorithm.

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

$\mathcal{B} = \{b_1, b_2\}$
$S = \{1, 2\}$

$seen_1 = \{b_1\}$
$seen_2 = \{b_2\}$

$visit = \{(1, \{b_1\}\), \{(2, \{b_2\}\)\}$
The More the Merrier: Efficient Multi-Source Graph Traversal

Walk-Through

\[
\mathbb{B} = \{b_1, b_2\} \\
S = \{1, 2\}
\]

1st BFS Level

\[
\begin{align*}
\text{seen}_1 &= \{b_1\} \\
\text{seen}_3 &= \{b_1, b_2\} \\
\text{seen}_2 &= \{b_2\} \\
\text{seen}_4 &= \{b_1, b_2\}
\end{align*}
\]

\[
\text{visit} = \begin{cases} 
(3, \{b_1\}) & B'_3 = \{b_1, b_2\} \\
(3, \{b_2\}) & B'_4 = \{b_1, b_2\} \\
(4, \{b_1\}) \\
(4, \{b_2\})
\end{cases}
\]
Walk-Through

The More the Merrier: Efficient Multi-Source Graph Traversal

February 2022

10 / 18

\[ G \]

\[ V = \{b_1, b_2\} \]
\[ S = \{1, 2\} \]

2nd BFS Level

\[ \text{seen}_1 = \{b_1, b_2\} \quad \text{seen}_4 = \{b_1, b_2\} \]
\[ \text{seen}_2 = \{b_1, b_2\} \quad \text{seen}_5 = \{b_1, b_2\} \]
\[ \text{seen}_3 = \{b_1, b_2\} \quad \text{seen}_6 = \{b_1, b_2\} \]

\[ \text{visit} = \{(5, \{b_1, b_2\}), (6, \{b_1, b_2\}), (1, \{b_2\}), (2, \{b_1\})\} \]
Table of Contents

1. Problem
2. Solution Intuition
3. Algorithm + walk-through
4. Iteration 2, usage of bitmaps
5. Iteration 3, optimizing cache misses
6. Experimental Data
A short coming of Iteration 1 was the overhead of runtime in maintaining the traversal sets ($b_i \in \beta$) when doing set operations. A solution to this was the usage of bitmaps that have constant time operations.

```
Listing 3: MS-BFS using bit operations.
1 Input: $G, B, S$
2 for each $b_i \in B$
3 seen[$s_i$] $\leftarrow 1 << b_i$
4 visit[$s_i$] $\leftarrow 1 << b_i$
5 reset visitNext
6
7 while visit $\neq \emptyset$
8 for $i = 1, \ldots, N$
9 if visit[$v_i$] = $B_\emptyset$, skip
10 for each $n \in$ neighbors[$v_i$]
11 D $\leftarrow$ visit[$v_i$] & ~seen[$n$]
12 if D $\neq B_\emptyset$
13 visitNext[$n$] $\leftarrow$ visitNext[$n$] | D
14 seen[$n$] $\leftarrow$ seen[$n$] | D
15 do BFS computation on $n$
16 visit $\leftarrow$ visitNext
17 reset visitNext
```
# Table of Contents

1. Problem

2. Solution Intuition

3. Algorithm + walk-through

4. Iteration 2, usage of bitmaps

5. Iteration 3, optimizing cache misses

6. Experimental Data
Many neighbors of a single vertex in the visit set are the neighbors of other vertices in the visit set. To avoid exploring the same neighbors multiple times (and possibly multiple cache misses for the same vertex), all neighbors are accumulated first before exploring them and adding them to the visitNext set.
Table of Contents

1 Problem

2 Solution Intuition

3 Algorithm + walk-through

4 Iteration 2, usage of bitmaps

5 Iteration 3, optimizing cache misses

6 Experimental Data
Figure 4: Data size scalability results.
Figure 5: Multi-core scalability results.
### Table 4: Runtime and speedup of MS-BFS compared to T-BFS and DO-BFS.

<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.*