A New Parallel Algorithm for Connected Components in Dynamic Graphs

Tim Kralj
Outline

• Connected Components
• Motivation
• STINGER
• Data Structure
• Parent-Neighbor Relationship
• Initialization
• Results
• Discussion
Connected Component

- Subgraph of original graph
- All vertices connected
- Usually calculated with BFS or DFS
- All vertices connected with paths
Uses of Connected Components

- Betweenness centrality
- Community detection
- Image processing
- Largely used in social networks
STINGER

- Data structure for dynamic graph problems
- Adjacency lists
  - Fast updates
- CSR
  - Low storage component
- Fast insertions
- Good locality
- Parallelism
- Streaming Graphs
Dynamic vs Static

- Dynamic Graph
  - Constantly updating
  - Steam of updates
  - Analysis of continuously changing state

- Static
  - Snapshot of dynamic graph
  - Analysis of a state
  - How most algorithms taught (006 etc)
Parent-Neighbor Relationship

• Memory Requirement of $O(V)$
• Directed subgraph
• Each vertex has list of vertices above “parents”
• List of vertices at same level “neighbors”
• Limit number of parents and neighbors with threshold
  • Helps maintain $O(V)$ instead of $O(V + E)$
Data Structure

- Maintained in real time
- Initialization step

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Size (Elements)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>Component labels</td>
<td>array</td>
<td>$O(V)$</td>
</tr>
<tr>
<td>$\text{Size}$</td>
<td>Component sizes</td>
<td>array</td>
<td>$O(V)$</td>
</tr>
<tr>
<td>$\text{Level}$</td>
<td>Approximate distance from the root</td>
<td>array</td>
<td>$O(V)$</td>
</tr>
<tr>
<td>$\text{PN}$</td>
<td>Parents and neighbors of each vertex</td>
<td>array of arrays</td>
<td>$O(V \cdot \text{thresh}_{\text{PN}}) = O(V)$</td>
</tr>
<tr>
<td>$\text{Count}$</td>
<td>Counts of parents and neighbors</td>
<td>array</td>
<td>$O(V)$</td>
</tr>
<tr>
<td>$\text{thresh}_{\text{PN}}$</td>
<td>Maximum count of parents and neighbors for a given vertex</td>
<td>value</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>$\tilde{E}_I$</td>
<td>Batch of edges to be inserted into graph</td>
<td>array</td>
<td>$O(\text{batch size})$</td>
</tr>
<tr>
<td>$\tilde{E}_R$</td>
<td>Batch of edges to be deleted from graph</td>
<td>array</td>
<td>$O(\text{batch size})$</td>
</tr>
</tbody>
</table>
Initialization

- Uses Parallel BFS
- Start with
  - Level at INF
  - Counter = 0
  - Component size = 0
- Dequeue vertex and add parents and neighbor relationships
  - Only add neighbors if below Threshold
  - All parents will be found before neighbors added
- Atomic compare-and-swap, fetch-and-add
Updates

• Insertions-simple
• Deletions-harder
Insertions

• 2 options:
  • Edge within a connected component
  • Edge joins two components
• Levels of s and d checked
• Update Data structure
Deletions

• Need to check parents of deleted edge
• Search for remaining parent
  • If remains, connection to the root of component must exist-safe
  • If no parents, check for neighbors if levels > 0 (meaning path to the root)
  • If neither is true, deletion was not safe
• Need to check from both s and d of <s,d> deletion
Results

• Varying threshold variable
  • Used for number of parents/neighbors maintained

Figure 2. Average number of unsafe deletes in $PN$ data structure for batches of $100K$ updates as a function of the average degree (x-axis) and $thresh_{PN}$ (bars).

Figure 1. Average number of inserts and deletes in $PN$ array for batches of $100K$ updates for RMAF-22 graphs. The subfigures are for different values of $thresh_{PN}$. Note that the ordinate is dependent on the specific bar chart. The charts for RMAF-21 graphs had very similar structure and have been removed for the sake of brevity.
Results

Figure 4. Speed up of the new algorithm over performing parallel static recomputation after each batch on three different RMAT-22 graphs with each average degree as a function of the number of threads.

Figure 6. Speed up over performing static recomputation after each batch on scale 24 graphs for three graphs at each edge factor using 64 threads.
Mentions of increasing core counts but none of distributed systems, would this be feasible and improve performance?

Use BFS in the initial stage and during steaming, be better if switched to a faster algorithm

Largest graph had $2^{24}$ vertices (16 million) but social graphs much larger

First parallel computing of connected components with processor oblivious runtime