An Efficient Connected Components Algorithm for Massively-Parallel Devices

Jayadharini Jaiganesh & Martin Burtscher
Department of Computer Science
Connected Components

- A Connected Component \( C \) is a subset of vertices such that,
  - All vertices in \( C \) are \textit{reachable} from any vertex in \( C \)
  - \textbf{No edges} between vertices belonging to different components

- Navigation
- Medicine - Cancer and tumor detection
- Biochemistry
  - Protein study
  - Drug discovery
PRIOR WORK
Standard CC Algorithm

- Label Propagation
  - Mark each vertex with unique label
  - Propagate vertex labels through edges
  - Repeat until all vertices in same component have same label

![Diagram of label propagation](image-url)
Parallel CC Algorithm - Shiloach & Vishkin’s

- Each vertex is considered a separate tree
  - Component labelled by its own ID

- Iterates on two operations
  - Hooking
  - Pointer Jumping
Hooking

- Works on edges
- For each edge \((u, v)\), checks if \(u\) and \(v\) have same label
- If not, link higher label to lower label
Pointer Jumping

- Works on vertices
- Replaces a vertex’s label with its parent’s label
- Reduces depth of tree by one
Parallel CC Algorithm - Soman’s

- A variant of Shiloach-Vishkin’s algorithm
- Uses Multiple Pointer Jumping
  - Iteratively performs Pointer Jumping
  - Converts multi-level tree to a single-level tree (star)
  - Reduces tree’s height to one
Parallel CC Algorithm - Groute

- Variant of Soman’s work

- Comprises **Atomic Hooking** and Multiple Pointer Jumping
  - Locks component ID vertex until hooking succeeds
  - No overriding with concurrent hooking operations

- Splits graph into $\frac{2|E|}{|V|}$ edge list segments
  - Enables intermediate pointer jumping
  - Reduces operations in the next segment’s hooking
ECL-CC: OUR ALGORITHM
Our Solution - ECL-CC Algorithm

- Like previous work, it chooses **minimum vertex ID** in each component as component ID to guarantee **uniqueness**

- Comprises three main functions
  - Init, Compute, and Flatten

- Init function
  - Initializes each vertex’s label with a **smaller neighbor ID** if possible
Our Solution - ECL-CC Algorithm (cont.)

- **Compute function**
  - Processes each edge of a vertex so that both ends of edge have same component ID
  - Makes sure that each edge is considered in only one direction
  - Employs Intermediate Pointer jumping

- **Flatten function**
  - A form of Multiple Pointer jumping
ECL-CC - GPU Implementation

- Written in CUDA
- Lock-free implementation based on atomic operations
- Uses double-sided worklist for load balancing
- Uses three compute kernels
  - compute1: $|E| \leq 16$, thread-level parallelism
  - compute2: $16 < |E| \leq 352$, warp-level parallelism
  - compute3: $|E| > 352$, block-level parallelism

```
V_1  V_4  V_5  |  V_7  V_2
16 < |E| \leq 352
|E| > 352
```
Our Solution - ECL-CC\textsubscript{af} Algorithm

- Atomic operations
  - Slower than atomic-free operations
  - Potential bottleneck for future massively parallel devices

- ECL-CC\textsubscript{af} - Synchronous atomic-free version of ECL-CC
- Uses same three functions - Init, Compute, and Flatten
- Repeatedly calls Compute to avoid data races
EVALUATION METHODOLOGY
Machines - GPU

- NVIDIA GeForce GTX Titan X
- NVIDIA Tesla K40

<table>
<thead>
<tr>
<th></th>
<th>Titan X</th>
<th>K40</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>3072</td>
<td>2880</td>
</tr>
<tr>
<td>Global Memory</td>
<td>12 GB</td>
<td>12 GB</td>
</tr>
<tr>
<td>Clock Frequency</td>
<td>1.1 GHz</td>
<td>745 MHz</td>
</tr>
</tbody>
</table>
Machine - CPU

- Machine 1
  - Intel Xeon E5-2687W
  - Hyperthreading

<table>
<thead>
<tr>
<th></th>
<th>Machine 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sockets</td>
<td>2</td>
</tr>
<tr>
<td>Cores</td>
<td>10</td>
</tr>
<tr>
<td>Clock Frequency</td>
<td>3.1 GHz</td>
</tr>
</tbody>
</table>
Input Graphs

- Eighteen graphs
  - 65K to 18M vertices
  - 387K to 523M edges
- Graph types
  - Roadmaps
  - Random graphs
  - Synthetic graphs
  - Internet topology graphs
  - Social network graphs
  - Web-links graphs
RESULTS: ECL-CC$_{af}$
Fastest on 6 graphs and Groute is $1.04x$ faster
Slowdown Relative to ECL-CC_{af} - K40

- Fastest on 8 graphs and Groute is 1.2x faster
RESULTS: ECL-CC
Slowdown Relative to ECL-CC - Titan X

- Fastest on 16 graphs and at least 1.8x faster on average
Fastest on **14** graphs and at least **1.6x** faster on average
Geometric-Mean Slowdown Across Systems

- **Fastest** among all benchmarks across different platforms
ALGORITHM ANALYSIS
Init Versions

- Version 1
  - Label is assigned with the vertex’s own ID
- Version 2
  - Label is assigned with the vertex’s minimum neighbor’s ID
- Version 3
  - Label is set with the ID of the first smaller neighbor
  - Avoids traversing all neighbors
  - Label is set with a better value
  - Used in ECL-CC algorithm
- On average, **1.4x** faster than version 2
Pointer Jumping Versions

- Version 1 - Multiple Pointer Jumping
- Version 2 - Single Pointer Jumping
- Version 3 - No Pointer Jumping (returns end of list)
- Version 4 - Intermediate Pointer Jumping
  - Links every node to second-to-next node
  - Reduces list length by a factor of two
- Used in ECL-CC

Connected Components 29
## Vertex Chain Length

<table>
<thead>
<tr>
<th>No</th>
<th>Graph Name</th>
<th>Vertex degree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>max</td>
</tr>
<tr>
<td>1</td>
<td>2d-2e20</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>amazon0601</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>as-skitter</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>citationCitesen</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>cit-Patents</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>coPapersDBLP</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>delaunay_n24</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>europe_osm</td>
<td>122</td>
</tr>
<tr>
<td>9</td>
<td>in-2004</td>
<td>31</td>
</tr>
<tr>
<td>10</td>
<td>internet</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>kron_g500-logn21</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>r4-2e23</td>
<td>29</td>
</tr>
<tr>
<td>13</td>
<td>rmat16</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td>rmat22</td>
<td>8</td>
</tr>
<tr>
<td>15</td>
<td>soc-livejournal</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>uk-2002</td>
<td>91</td>
</tr>
<tr>
<td>17</td>
<td>USA-NY</td>
<td>43</td>
</tr>
<tr>
<td>18</td>
<td>USA-USA</td>
<td>27</td>
</tr>
</tbody>
</table>
Slowdown Relative to ECL-CC Pointer Jumping

- At least **1.2x to 3.6x** faster than other versions on average
Flatten Versions

- **Version 1 - Intermediate Pointer jumping**
  - Links every node to second-to-next node
  - Current node is linked to end of list
  - Reduces list length by a factor of two

- **Version 2 - Multiple Pointer jumping**
  - Links every node to end of list

- **Version 3 - Pointer jumping**
  - Only current node is linked to end of list
  - Used in ECL-CC
- Flatten’s runtime at least $4\times$ faster on larger graphs - $|V| > 15M$
- On average, $1.2\times$ faster than version 2
SUMMARY
Summary

- **ECL-CC\_af** - Atomic free and synchronous algorithm
  - Iterates over compute kernels to avoid data races
  - Average performance on par with Groute

- **ECL-CC** - Asynchronous CC algorithm
  - Uses optimized version of initialization
  - Employs a **double-sided worklist** & three compute kernels
  - Incorporates Intermediate Pointer jumping
  - Considers each edge in only one direction
  - On average, **1.7x** faster than fastest GPU algorithm
Thank you 😊

Jayadharini Jaiganesh
Texas State University
jayadharini@txstate.edu
Download link
http://cs.txstate.edu/~burtscher/research/ECL-CC/
Algorithm - ECL-CC

- **procedure:** ECL-CC (V, E)
  1. **Init (V, nstat)**
  2. **Compute (V, E, nstat)**
  3. **Flatten (V, nstat)**

- **procedure:** Init (V, nstat)
  1. nstat = {0, ..., |V|-1} //Hold the vertex labels
  2. for each vertex v in V
  3. nstat[v] ← First neighbor smaller than v.
procedure: Compute \((V, E, nstat)\)

1. for each \(v\) in \(V\) { 
2. \(vstat \leftarrow \text{representative} (v, nstat)\) 
3. for each edge \((u, v)\) in \(E\) { 
4. \(\text{if} (v > u) \{\)
5. \(ostat \leftarrow \text{representative} (u, nstat)\)
6. \(\text{if} (vstat < ostat)\)
7. \(nstat[ostat] \leftarrow vstat\)
8. \(\text{else}\)
9. \(nstat[vstat] \leftarrow ostat\)
10. \} 
11. } 
12. }
procedure: Representative \((v, nstat)\)
1. \(curr \leftarrow nstat[v]\)
2. \(\text{if} \ (curr \neq v) \{\)
3. \(prev \leftarrow v\)
4. \(next \leftarrow nstat[curr]\)
5. \(\text{while} \ (curr > next) \{\)
6. \(nstat[prev] \leftarrow next\)
7. \(prev \leftarrow curr\)
8. \(curr \leftarrow next\)
9. \}\)
10. \}\)
Flatten Function

- A form of **pointer jumping**
- Updates the label of all the vertices so that it represents the component ID directly

**procedure**: Flatten (V, nstat)

1. for each vertex v in V {
2.     vstat ← nstat[v]
3.     while (vstat > nstat[vstat])
4.         vstat ← nstat[vstat]
5.     nstat[v] ← vstat
6. }
Algorithm - ECL-CC\textsubscript{af}

- **procedure:** ECL-CC\textsubscript{af} (V, E)

1. Init (V, nstat)
2. reiterate $\leftarrow 1$
3. do
4. if reiterate
5. Compute (V, E, nstat, &reiterate)
6. end if
7. while (!reiterate)
8. Flatten (V, nstat)
Graph Representation

- Compressed Adjacency List (two arrays)
  - neighborlist - concatenation of all adjacency lists
  - neighborindex - starting point of each adjacency list
### Graph Details

<table>
<thead>
<tr>
<th>S. No</th>
<th>Graph Name</th>
<th>No. of Edges (M)</th>
<th>No. of Vertices (M)</th>
<th>Vertex degree</th>
<th>No. of CC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>1</td>
<td>2d-2e20</td>
<td>1.0</td>
<td>4.2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>amazon0601</td>
<td>0.4</td>
<td>4.9</td>
<td>1</td>
<td>2752</td>
</tr>
<tr>
<td>3</td>
<td>as-skitter</td>
<td>1.7</td>
<td>2.2</td>
<td>1</td>
<td>35455</td>
</tr>
<tr>
<td>4</td>
<td>citationCiteseer</td>
<td>0.3</td>
<td>2.3</td>
<td>1</td>
<td>1318</td>
</tr>
<tr>
<td>5</td>
<td>cit-Patents</td>
<td>3.8</td>
<td>33.0</td>
<td>1</td>
<td>793</td>
</tr>
<tr>
<td>6</td>
<td>coPapersDBLP</td>
<td>0.5</td>
<td>30.5</td>
<td>1</td>
<td>3299</td>
</tr>
<tr>
<td>7</td>
<td>delaunay_n24</td>
<td>16.8</td>
<td>100.7</td>
<td>3</td>
<td>26</td>
</tr>
<tr>
<td>8</td>
<td>europe_osm</td>
<td>50.9</td>
<td>108.1</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>in-2004</td>
<td>1.4</td>
<td>27.2</td>
<td>0</td>
<td>21869</td>
</tr>
<tr>
<td>10</td>
<td>internet</td>
<td>0.1</td>
<td>0.4</td>
<td>1</td>
<td>151</td>
</tr>
<tr>
<td>11</td>
<td>kron_g500-logn21</td>
<td>2.1</td>
<td>182.1</td>
<td>0</td>
<td>213904</td>
</tr>
<tr>
<td>12</td>
<td>r4-2e23</td>
<td>8.4</td>
<td>67.1</td>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>rmat16</td>
<td>0.1</td>
<td>1.0</td>
<td>0</td>
<td>569</td>
</tr>
<tr>
<td>14</td>
<td>rmat22</td>
<td>4.2</td>
<td>65.7</td>
<td>0</td>
<td>3687</td>
</tr>
<tr>
<td>15</td>
<td>soc-livejournal</td>
<td>4.8</td>
<td>85.7</td>
<td>0</td>
<td>20333</td>
</tr>
<tr>
<td>16</td>
<td>uk-2002</td>
<td>18.5</td>
<td>523.6</td>
<td>0</td>
<td>194955</td>
</tr>
<tr>
<td>17</td>
<td>USA-NY</td>
<td>0.3</td>
<td>0.7</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>18</td>
<td>USA-USA</td>
<td>23.9</td>
<td>57.7</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>