# An Efficient Connected Components Algorithm for Massively-Parallel Devices

Jayadharini Jaiganesh & Martin Burtscher

Department of Computer Science

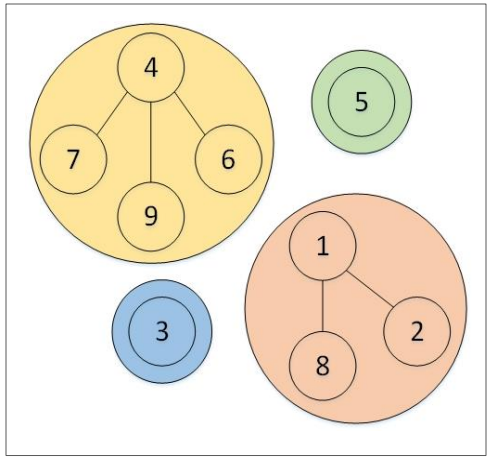
TEXAS STATE UNIVERSITY

*The rising* STAR *of Texas*

# Connected Components

- A Connected Component C is a subset of vertices such that,
  - All vertices in C are **reachable** from any vertex in C
  - **No edges** between vertices belonging to different components

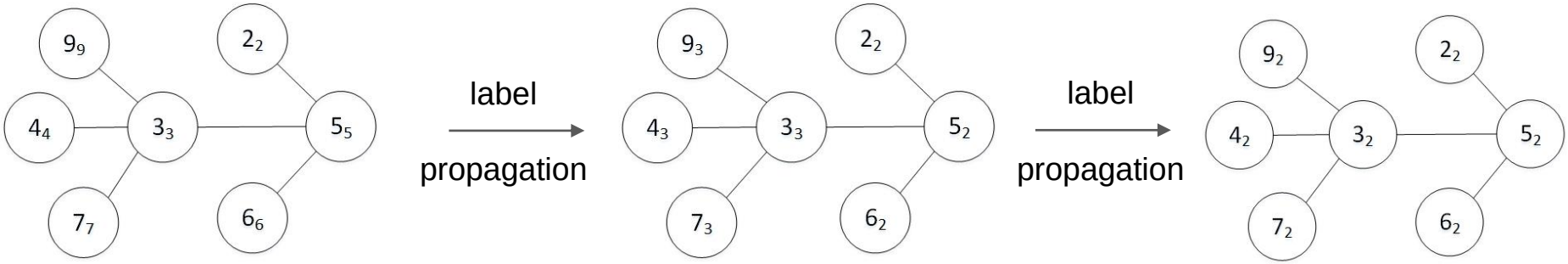
- Navigation
- Medicine - Cancer and tumor detection
- Biochemistry
  - Protein study
  - Drug discovery



# PRIOR WORK

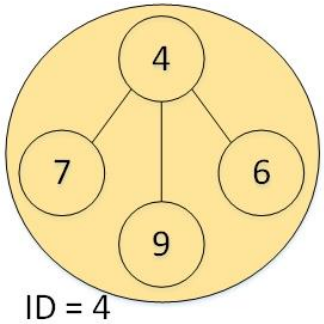
# Standard CC Algorithm

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

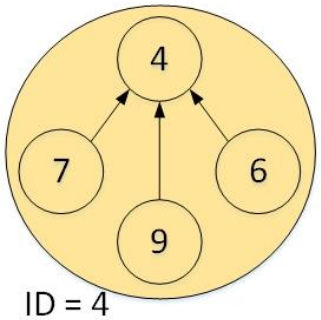


# Parallel CC Algorithm - Shiloach & Vishkin's

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

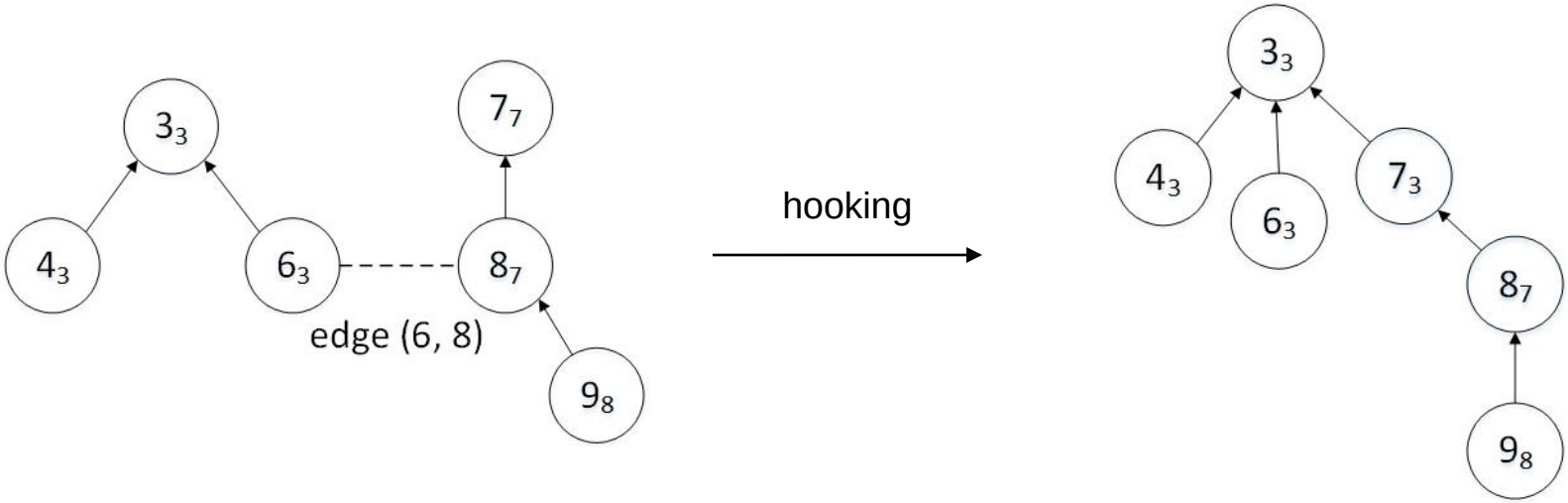



- Iterates on two operations
  - Hooking
  - Pointer Jumping



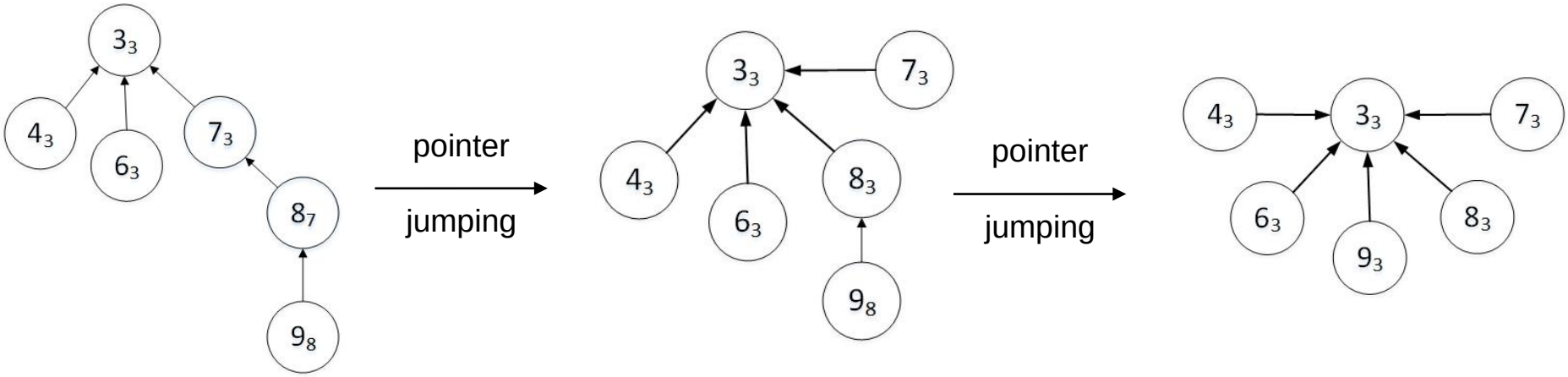
# Hooking

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



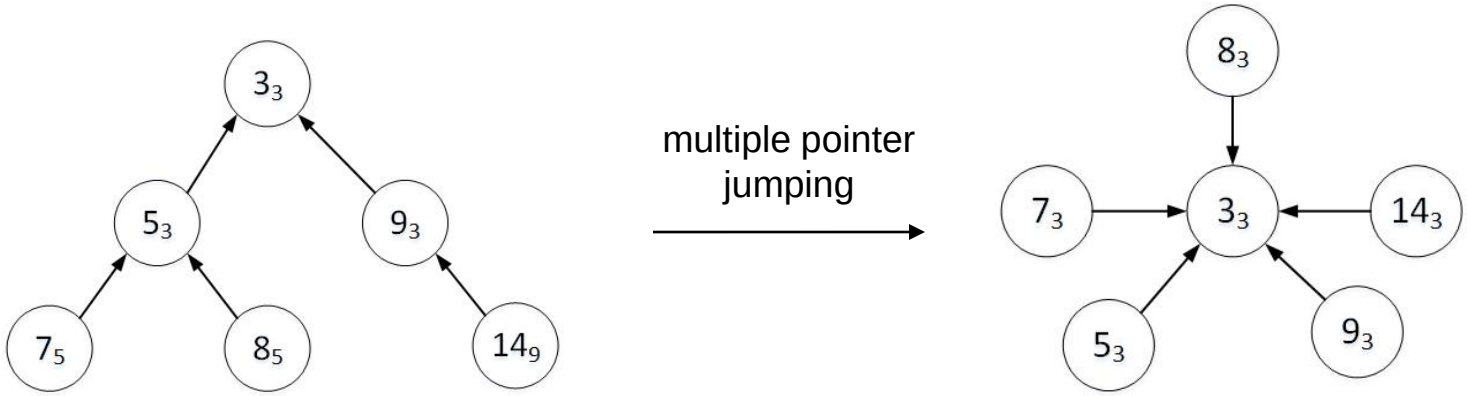
# Pointer Jumping

- 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**



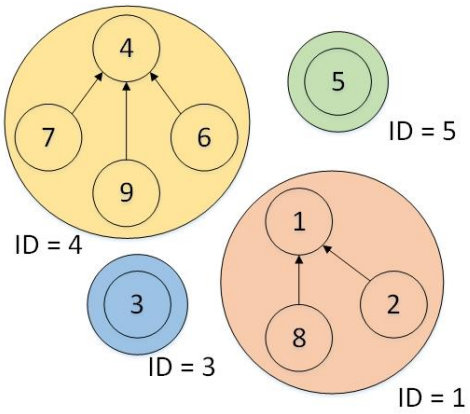
# Parallel CC Algorithm - Groute

- 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 **(2*|E|)/|V|** edge list segments
  - Enables intermediate pointer jumping
  - Reduces operations in the next segment's hooking

# ECL-CC: OUR ALGORITHM

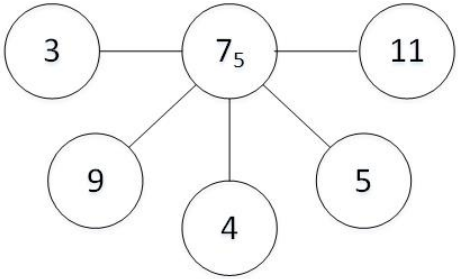
# Our Solution - ECL-CC Algorithm

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



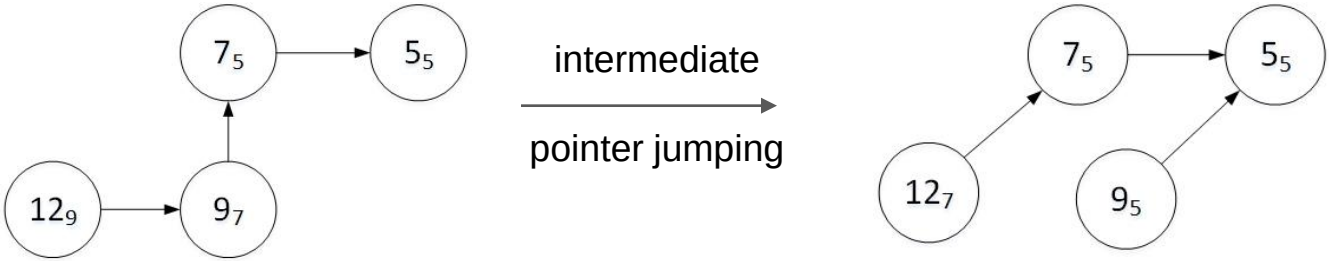

- Comprises three main functions
  - Init, Compute, and Flatten

- 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

# Our Solution - ECL-$CC_{af}$ Algorithm

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

- ECL-$CC_{af}$ - 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

| | **Titan X** | **K40** |
|---|---|---|
| Cores | 3072 | 2880 |
| Global Memory | 12 GB | 12 GB |
| Clock Frequency | 1.1 GHz | 745 MHz |

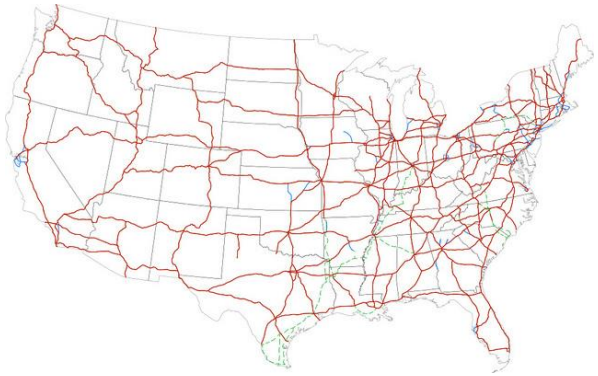
# Machine - CPU

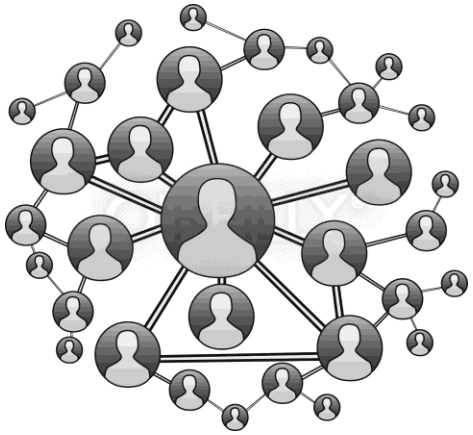
- Machine 1
  - Intel Xeon E5-2687W
  - Hyperthreading


| | **Machine 1** |
|---|---|
| Sockets | 2 |
| Cores | 10 |
| Clock Frequency | 3.1 GHz |

# Input Graphs

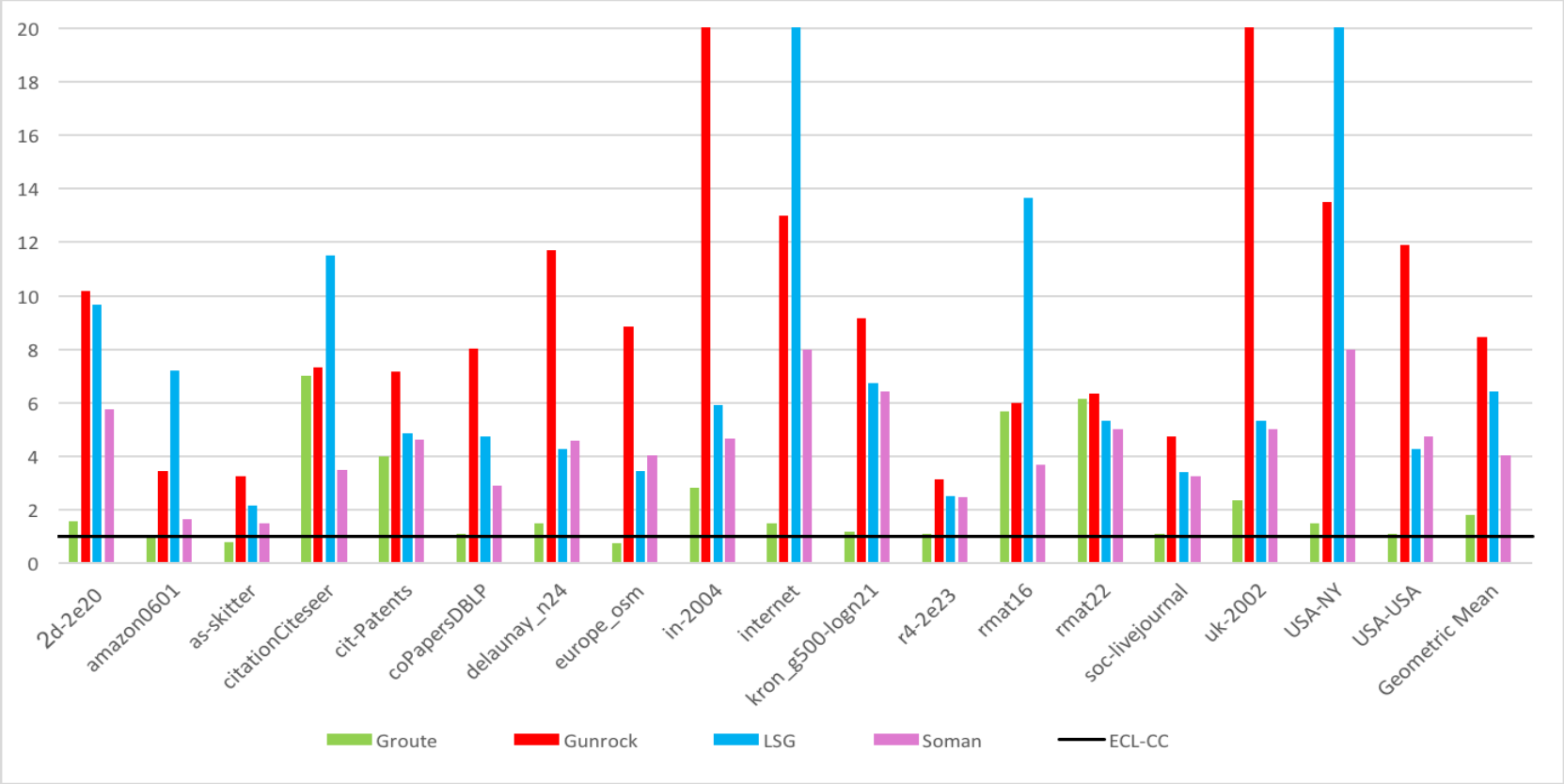
- Eighteen graphs
  - 65K to 18M vertices
  - 387K to 523M edges
- Graph types
  - Roadmaps
  - Random graphs
  - Synthetic graphs
  - Internet topology graphs
  - Social network graphs
  - Web-links graphs





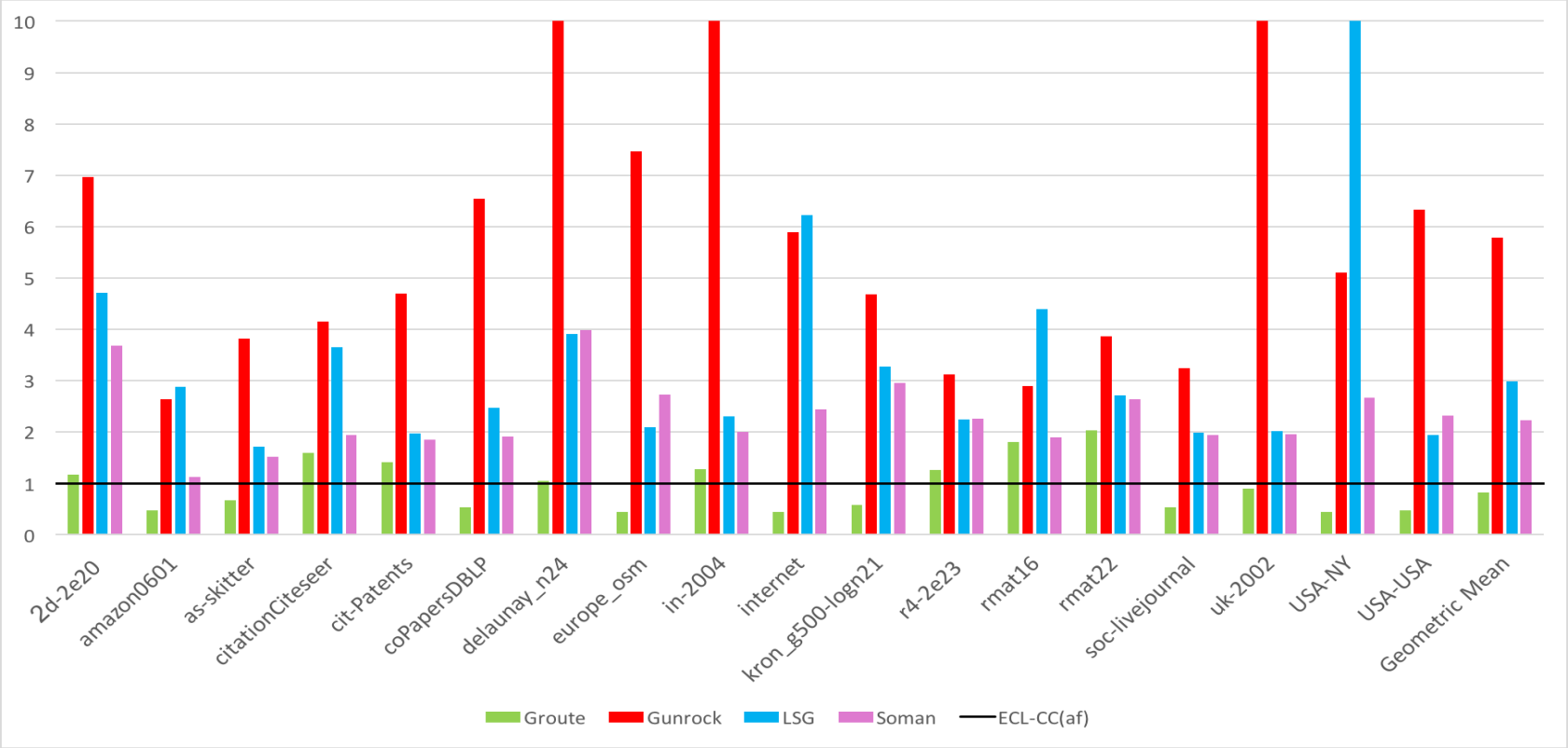
# RESULTS: ECL-$CC_{af}$

# Slowdown Relative to ECL-CC$_{af}$ - Titan X




- Fastest on **6** graphs and Groute is **1.04x** faster

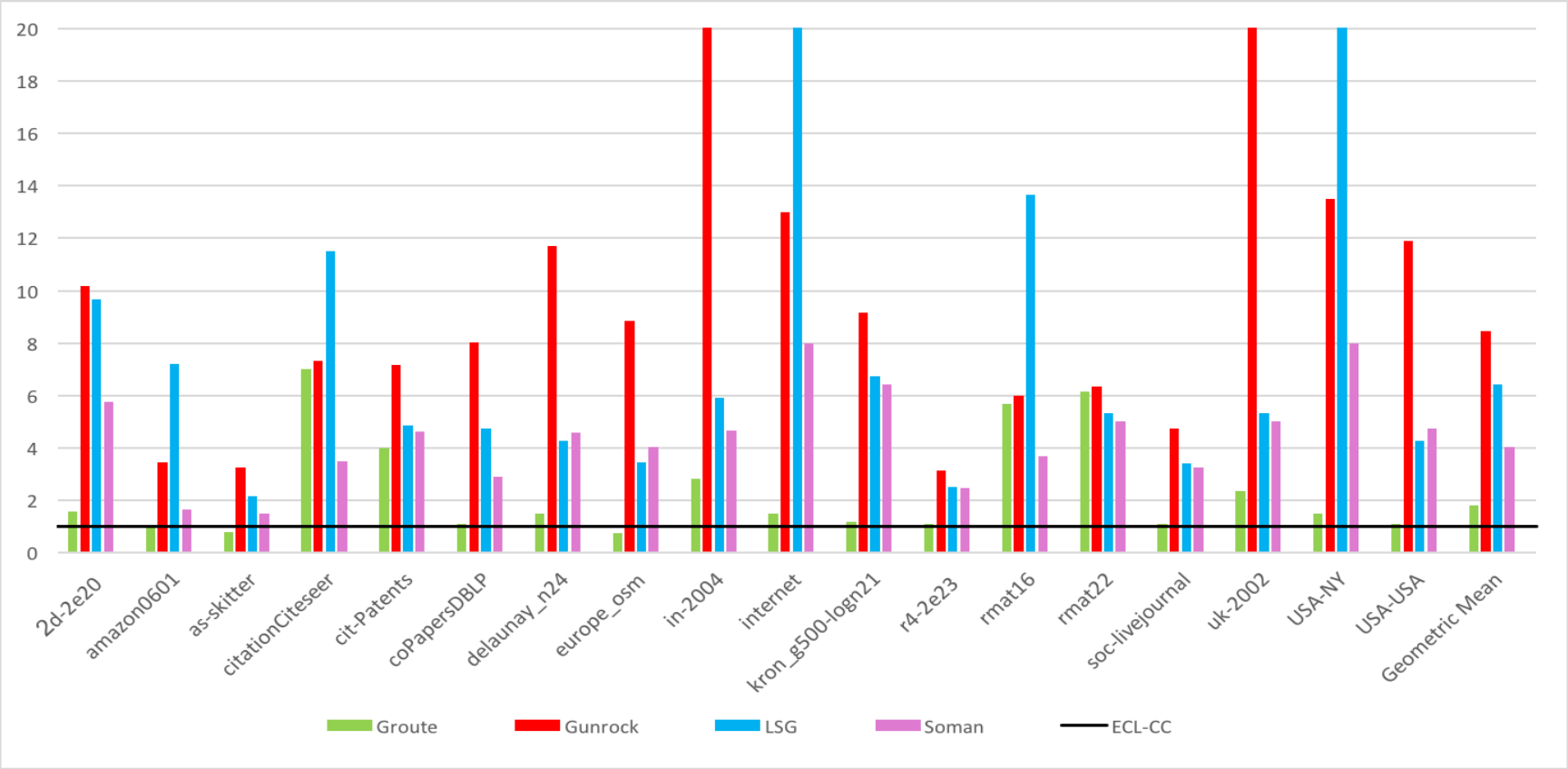
# Slowdown Relative to ECL-CC$_{af}$ - K40




- Fastest on **8** graphs and Groute is **1.2x** faster

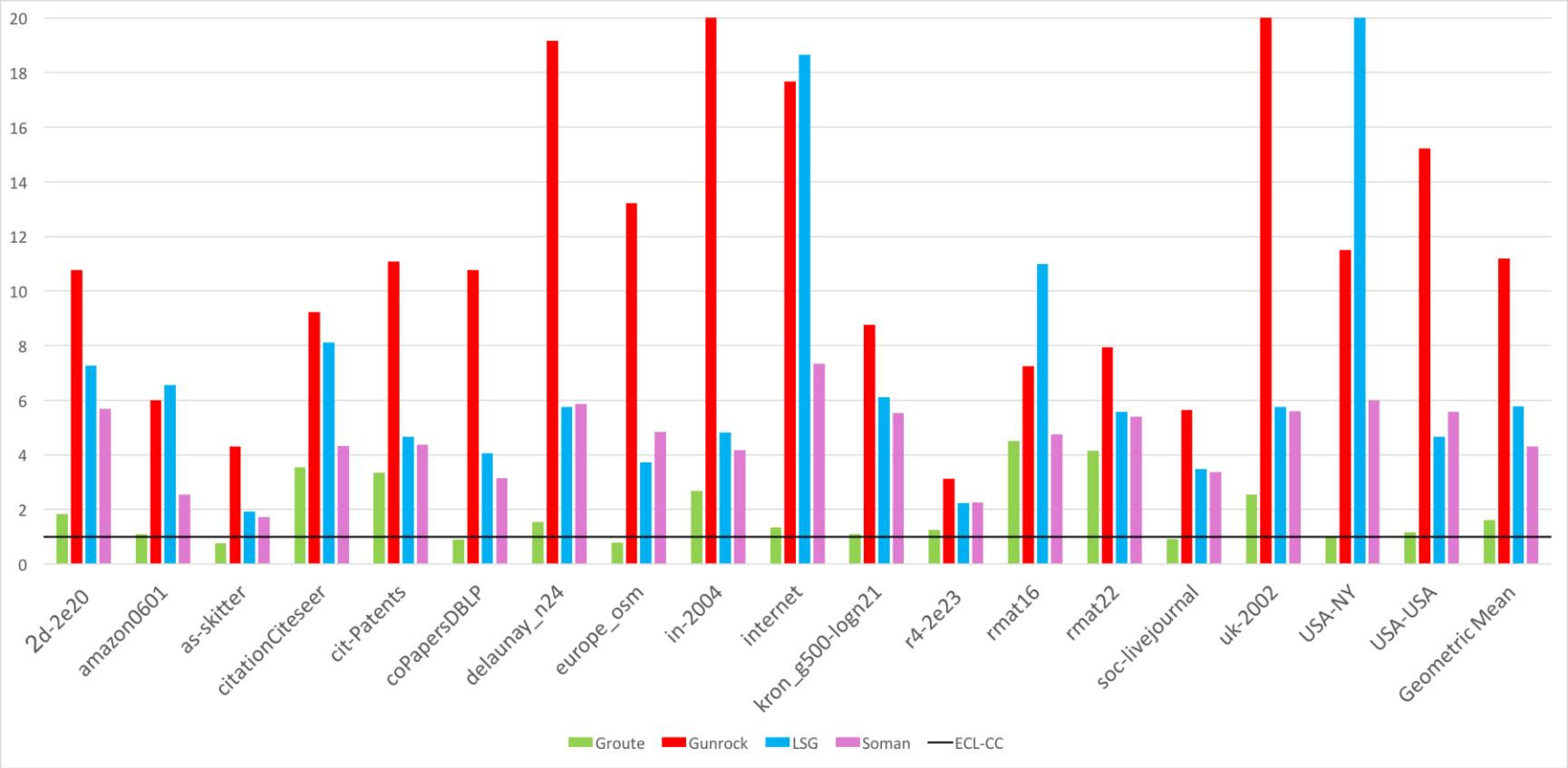
# RESULTS: ECL-CC

# Slowdown Relative to ECL-CC - Titan X



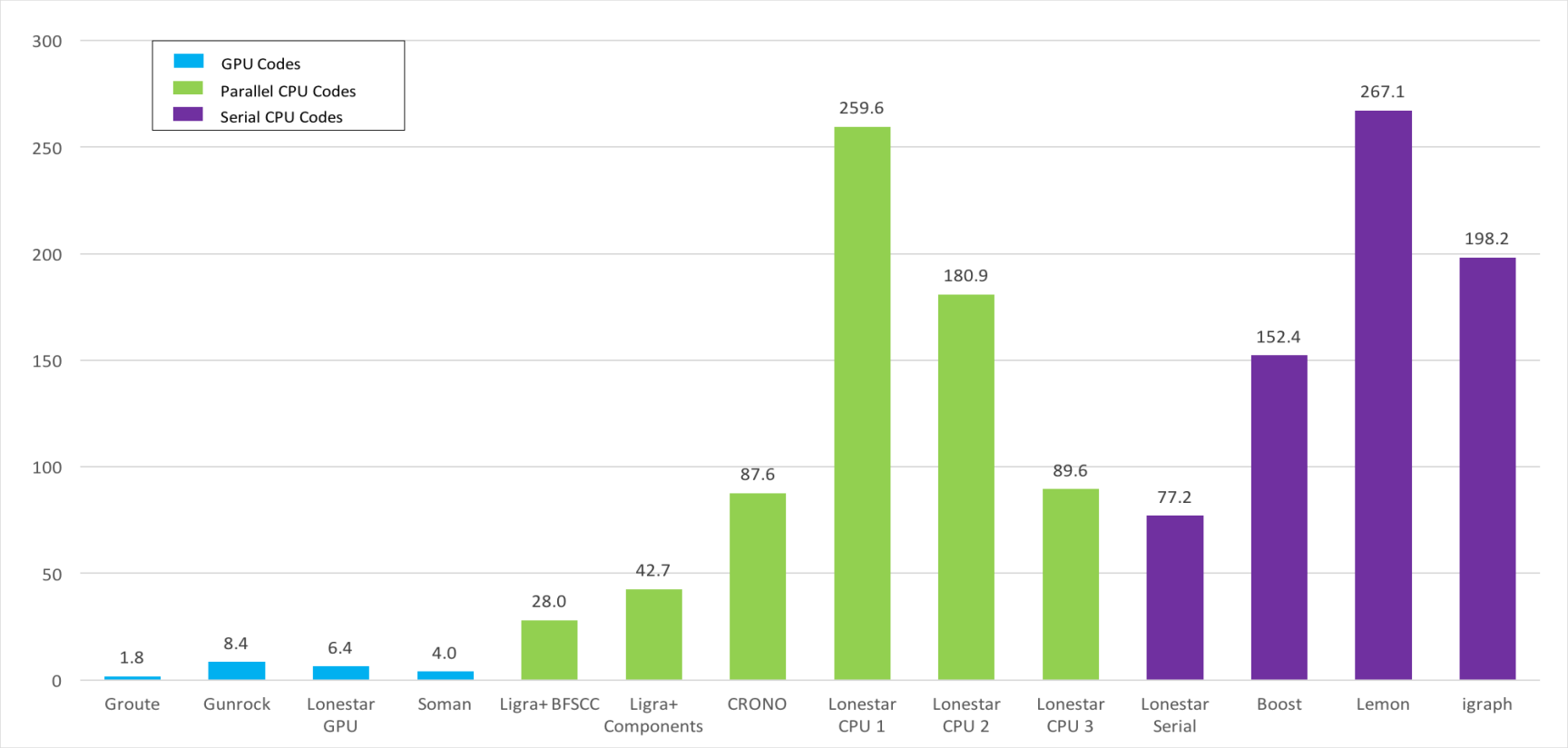

- Fastest on **16** graphs and at least **1.8x** faster on average

# Slowdown Relative to ECL-CC - K40




- Fastest on **14** graphs and at least **1.6x** faster on average

# Geometric-Mean Slowdown Across Systems



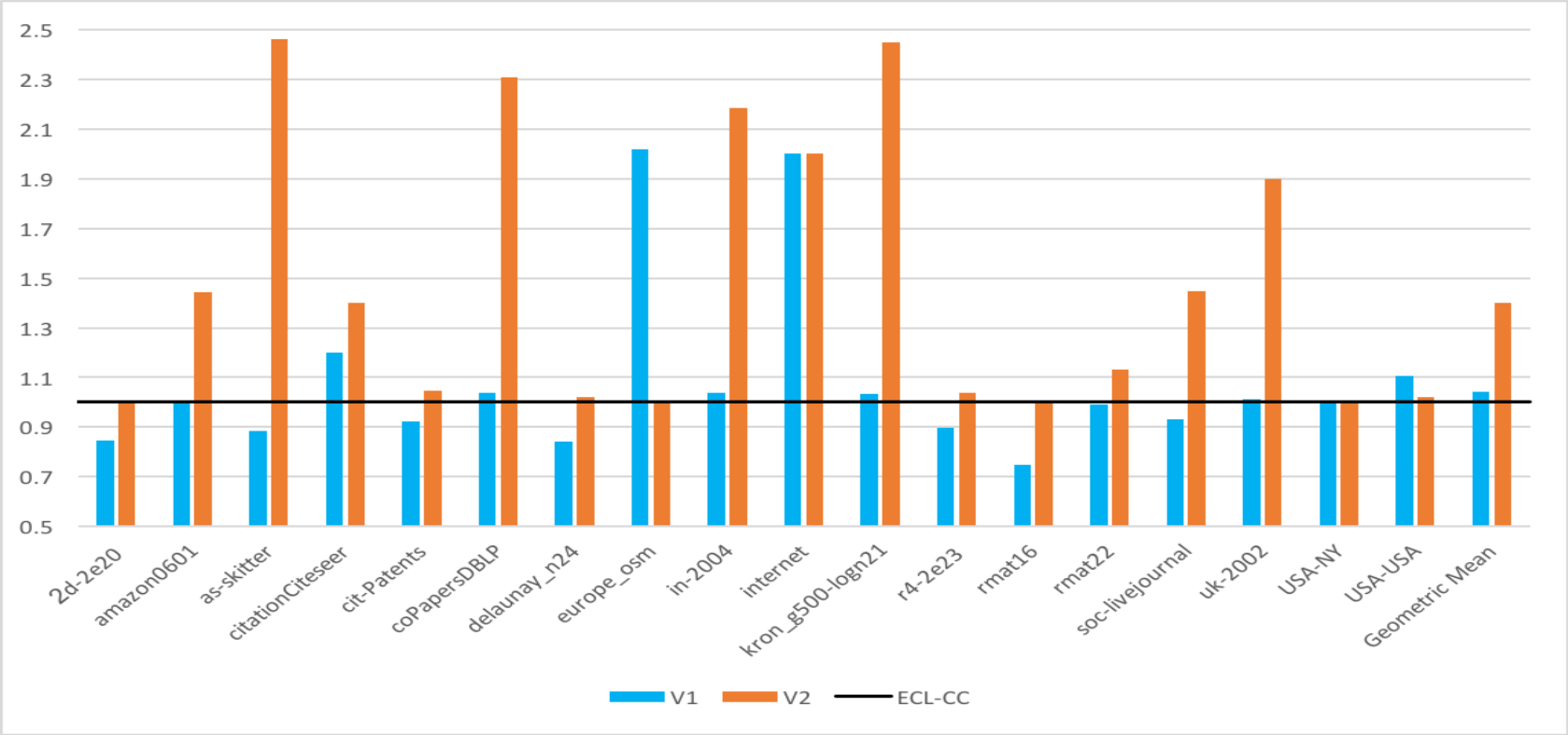

- **Fastest** among all benchmarks across different platforms

# 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

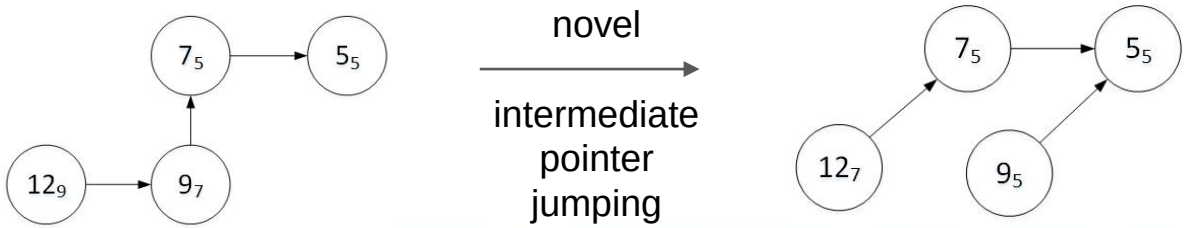
# Slowdown Relative to ECL-CC Init




- On average, **1.4**x faster than version 2

# Pointer Jumping Versions

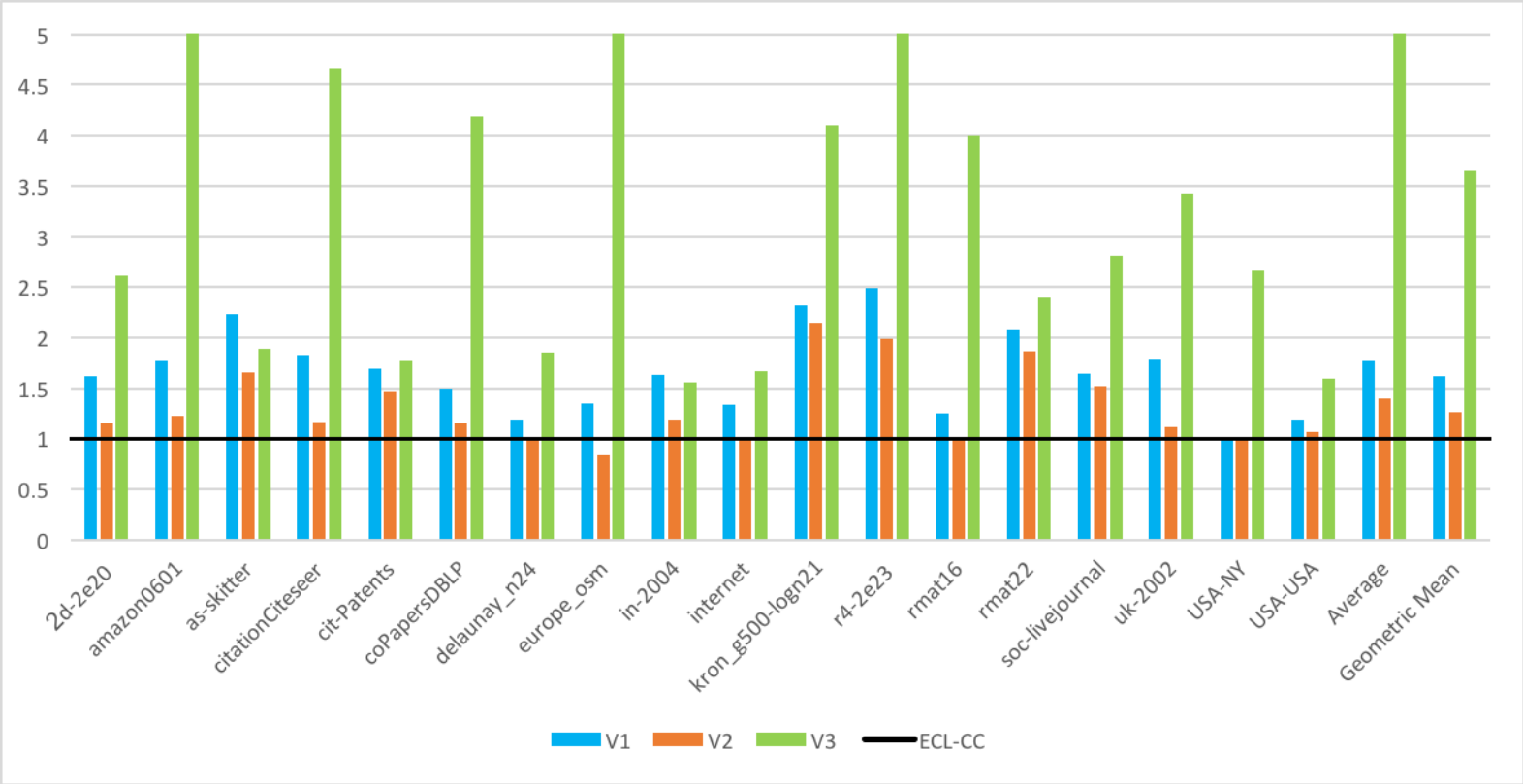
- Version 1 - Multiple Pointer Jumping
- Version 2 - Single Pointer Jumping
- Version 3 - No Pointer Jumping (returns end of list)
- Version 4 - Intermediate Pointer Jumping
  - Links every node to second-to-next node
  - Reduces list length by a factor of two
  - Used in ECL-CC



# Vertex Chain Length

| No | Graph Name | Vertex degree | |
|---|---|---|---|
| | | max | avg |
| **1** | 2d-2e20 | 9 | 1.4 |
| **2** | amazon0601 | 8 | 1.3 |
| **3** | as-skitter | 17 | 1.0 |
| **4** | citationCiteseer | 11 | 1.1 |
| **5** | cit-Patents | 9 | 1.0 |
| **6** | coPapersDBLP | 8 | 1.0 |
| **7** | delaunay_n24 | 13 | 1.4 |
| **8** | europe_osm | 122 | 4.3 |
| **9** | in-2004 | 31 | 1.1 |
| **10** | internet | 10 | 1.5 |
| **11** | kron_g500-logn21 | 6 | 1.0 |
| **12** | r4-2e23 | 29 | 1.3 |
| **13** | rmat16 | 10 | 1.3 |
| **14** | rmat22 | 8 | 1.1 |
| **15** | soc-livejournal | 7 | 1.0 |
| **16** | uk-2002 | 91 | 1.2 |
| **17** | USA-NY | 43 | 2.6 |
| **18** | USA-USA | 27 | 1.6 |

# Slowdown Relative to ECL-CC Pointer Jumping

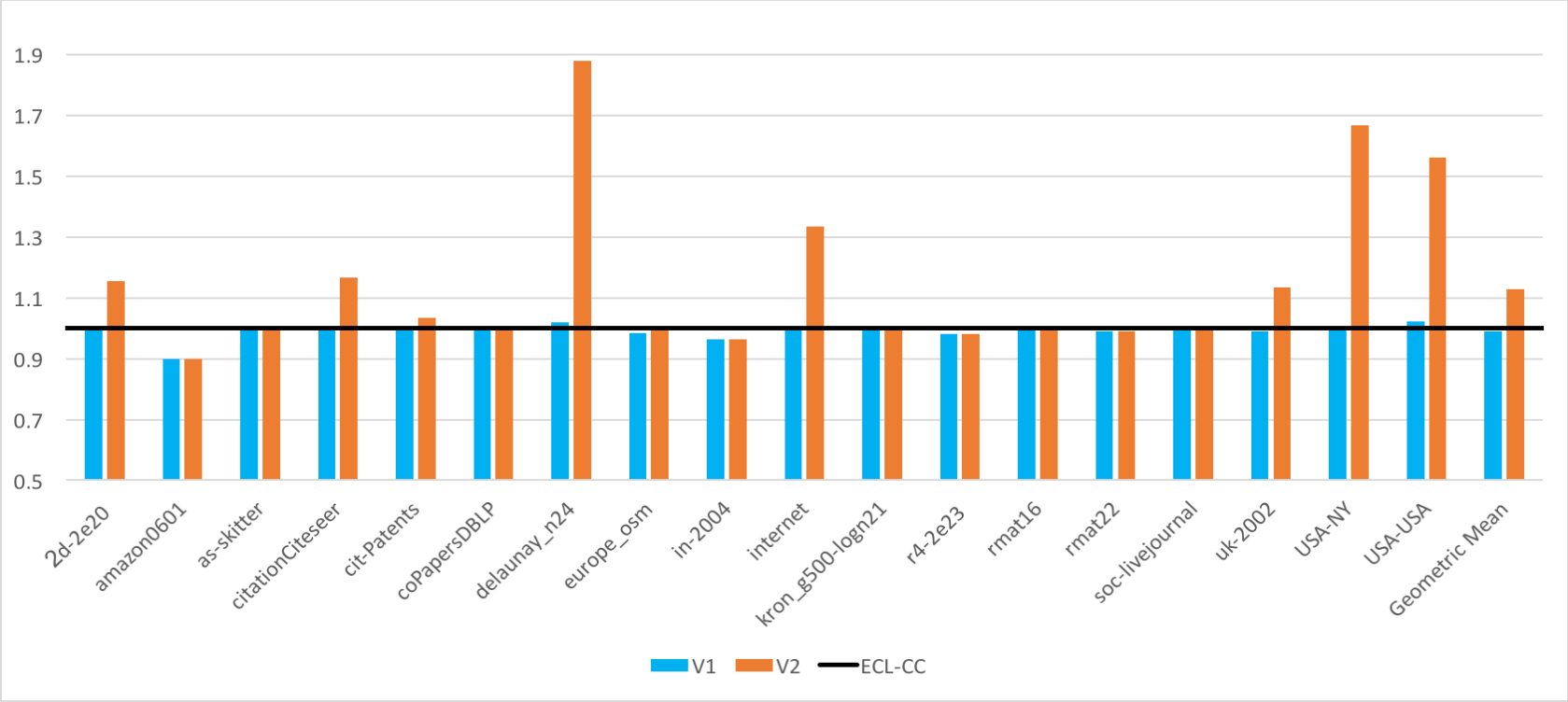



- At least **1.2x to 3.6x** faster than other versions on average

# 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

# Slowdown Relative to ECL-CC Flatten




- Flatten's runtime at least **4x** faster on larger graphs -|V| > 15M
- On average, **1.2x** 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)*

```
for each v in V {
    vstat ← representative (v, nstat)
    for each edge (u, v) in E {
        if (v > u) {
            ostat ← representative (u, nstat)
            if (vstat < ostat)
                nstat[ostat] ← vstat
            else
                nstat[vstat] ← ostat
        }
    }
}
```

- ***procedure:*** *Representative (v, nstat)*

```
curr ← nstat[v]
if (curr != v) {
    prev ← v
    next ← nstat[curr]
    while (curr > next) {
        nstat[prev] ← next
        prev ← curr
        curr ← next
    }
}
```

# 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)*

```
for each vertex v in V {
    vstat ← nstat[v]
    while (vstat > nstat[vstat])
        vstat ← nstat[vstat]
    nstat[v] ← vstat
}
```

# Algorithm - ECL-$CC_{af}$

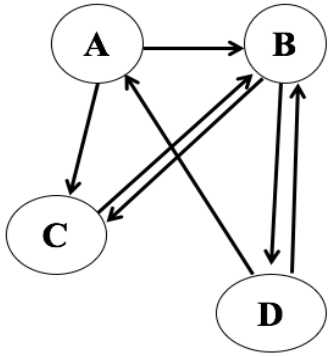
- **procedure:** ECL-$CC_{af}$ (V, E)

```
Init (V, nstat)
reiterate ← 1
do
    if reiterate
        Compute (V, E, nstat, &reiterate)
    end if
while (!reiterate)
Flatten (V, nstat)
```

# Graph Representation

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

Graph




Adjacency Lists

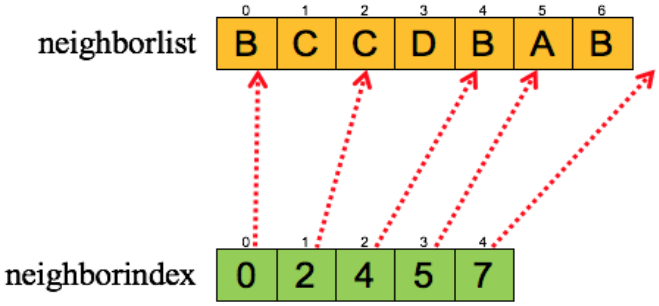
A: B, C

B: C, D

C: B

D: A, B

Compressed Adjacency List



# Graph Details

| S. No | Graph Name | No. of Edges (M) | No. of Vertices (M) | Vertex degree | | | No. of CC |
|---|---|---|---|---|---|---|---|
| | | | | min | max | avg | |
| 1 | 2d-2e20 | 1.0 | 4.2 | 2 | 4 | 3 | 1 |
| 2 | amazon0601 | 0.4 | 4.9 | 1 | 2752 | 12 | 7 |
| 3 | as-skitter | 1.7 | 2.2 | 1 | 35455 | 1 | 756 |
| 4 | citationCiteseer | 0.3 | 2.3 | 1 | 1318 | 8 | 1 |
| 5 | cit-Patents | 3.8 | 33.0 | 1 | 793 | 8 | 3,627 |
| 6 | coPapersDBLP | 0.5 | 30.5 | 1 | 3299 | 56 | 1 |
| 7 | delaunay_n24 | 16.8 | 100.7 | 3 | 26 | 5 | 1 |
| 8 | europe_osm | 50.9 | 108.1 | 1 | 13 | 2 | 1 |
| 9 | in-2004 | 1.4 | 27.2 | 0 | 21869 | 19 | 134 |
| 10 | internet | 0.1 | 0.4 | 1 | 151 | 3 | 1 |
| 11 | kron_g500-logn21 | 2.1 | 182.1 | 0 | 213904 | 86 | 553,159 |
| 12 | r4-2e23 | 8.4 | 67.1 | 2 | 26 | 7 | 1 |
| 13 | rmat16 | 0.1 | 1.0 | 0 | 569 | 14 | 3,900 |
| 14 | rmat22 | 4.2 | 65.7 | 0 | 3687 | 15 | 428,640 |
| 15 | soc-livejournal | 4.8 | 85.7 | 0 | 20333 | 17 | 1,876 |
| 16 | uk-2002 | 18.5 | 523.6 | 0 | 194955 | 28 | 38,359 |
| 17 | USA-NY | 0.3 | 0.7 | 1 | 8 | 2 | 1 |
| 18 | USA-USA | 23.9 | 57.7 | 1 | 9 | 2 | 1 |