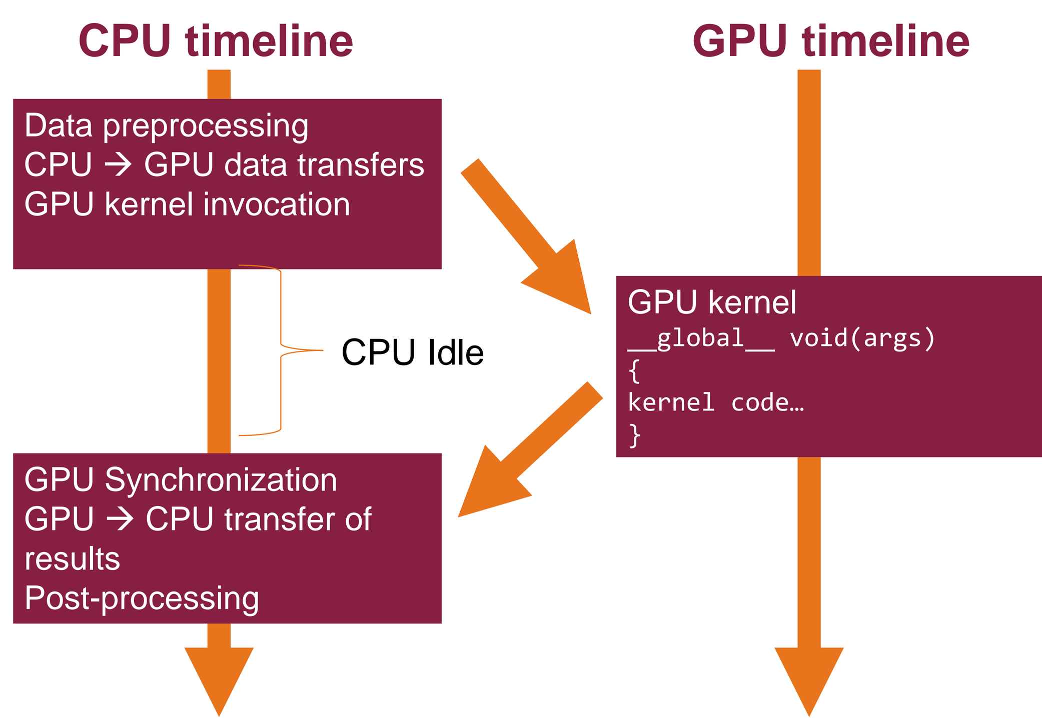
Hybrid CPU-GPU Implementation of Edge-Connected Jaccard Similarity in Graph Datasets

Atharva Gondhalekar, Paul Sathre, Wu-chun Feng | Virginia Tech | Blacksburg, VA, USA | {atharva1, sath6220, wfeng}@vt.edu

Motivation

- **Typical GPGPU computing:**

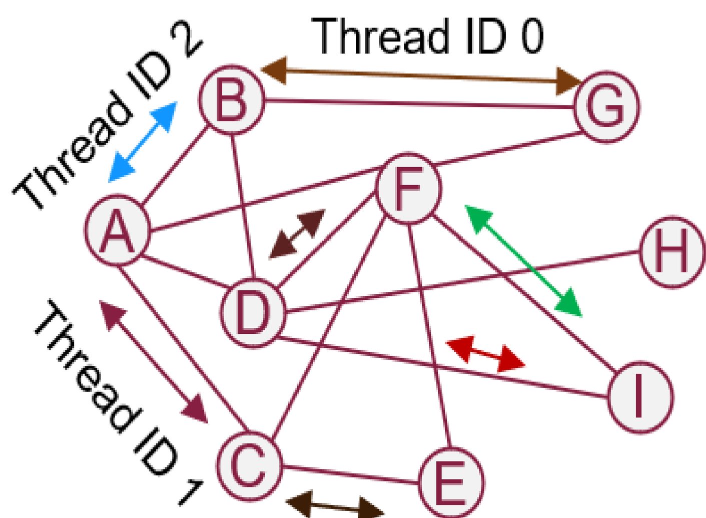


- CPU typically remains idle while GPU is executing the kernel
- Possible to distribute a portion of the workload to the CPU while the GPU kernel is running
- **Prior work on hybrid CPU-GPU workload distribution:**
  Automated distribution for optimal runtime [1] for **regular** workloads
- **This work:**
  Workload distribution for the **irregular** workload (Jaccard similarity on graph datasets)

- **Challenge:**
  - CPU + GPU workload distribution for irregular workloads
- **Solution**:
  - Evaluate hybrid CPU + GPU implementation of Jaccard similarity
  - Identify graph datasets for which CPU + GPU considerably outperforms GPU-only approach

CPU Implementation

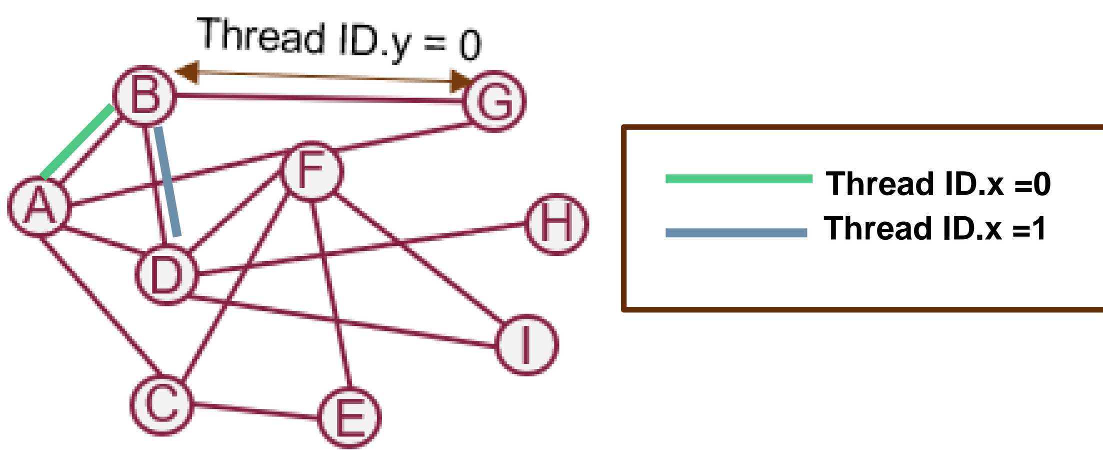
- Edge-centric (EC) parallelism
- Each thread computes JS for a unique edge



- Our implementation uses binary search and two-pointer intersection [5] to compute the set intersection
- Implemented in OpenMP with guided scheduling

GPU Implementation

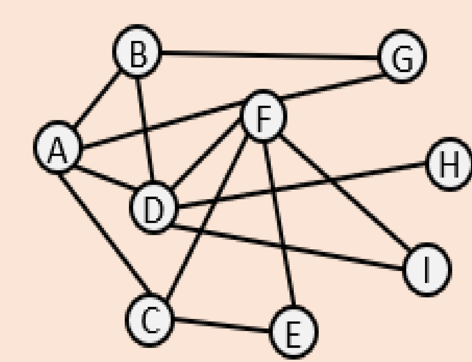
- Uses edge-centric (EC) parallelism
- 2D Set-intersection pair kernel from cuGraph[3]
- Y dimension →. Identify source + destination vertices
- X dimension → Check if each source neighbor is present in destination neighbor list



Jaccard Similarity Computation

Step 1: Graph Representation

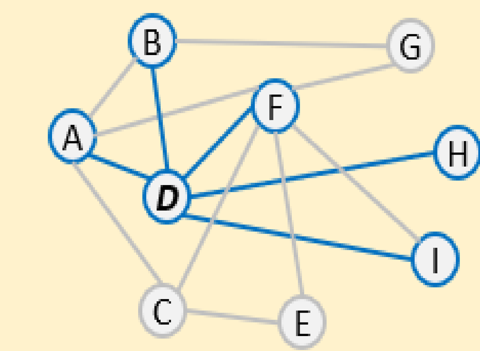
Undirected, Unweighted graph



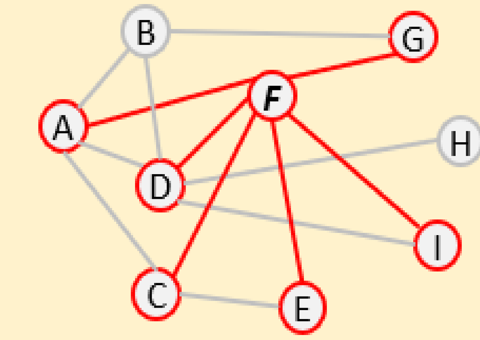
Equivalent Compressed Sparse Row (CSR)
Offset= {0,4,7,10,15,17,23,25,26,28}
Col = {1,2,3,5,0,3,6,0,4,5,0,1,5,7,8,2,5,0,2,3,4,6,8,1,5,3,3,5}

Step 2: Compute Sums of Vertex Neighbors

Unweighted RowSum(D) = 5



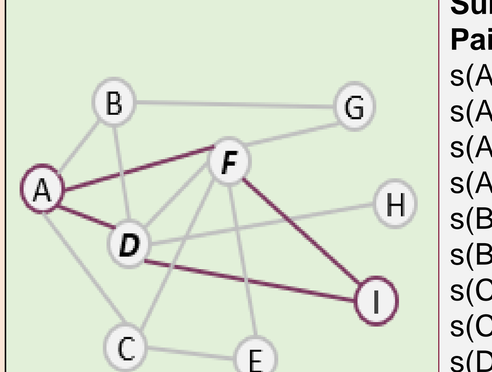
Unweighted RowSum(F) = 6



Row Sum
A 4
B 3
C 3
D 5
E 2
F 6
G 2
H 1
I 2

Step 3: Compute Pair Sums and Intersections

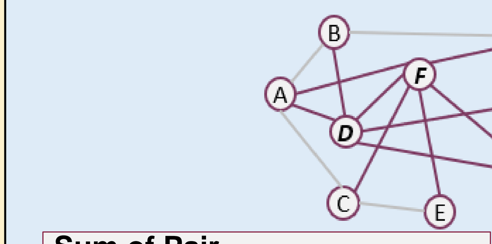
Unweighted Intersection (D,F) = 2



Sums of Pairs	Intersection
s(A,B) 7	i(A,B) 1
s(A,C) 7	i(A,C) 1
s(A,D) 9	i(A,D) 2
s(A,F) 10	i(A,F) 2
s(B,D) 8	i(B,D) 1
s(B,G) 5	i(B,G) 0
s(C,E) 5	i(C,E) 1
s(C,F) 9	i(C,F) 2
s(D,F) 11	i(D,F) 2
s(D,H) 6	i(D,H) 0
s(D,I) 7	i(D,I) 1
s(E,F) 8	i(E,F) 1
s(F,G) 8	i(F,G) 0
s(F,I) 8	i(F,I) 1

Step 4a: Compute Pair Unions on-the-fly

Unweighted Union (D,F) = 9



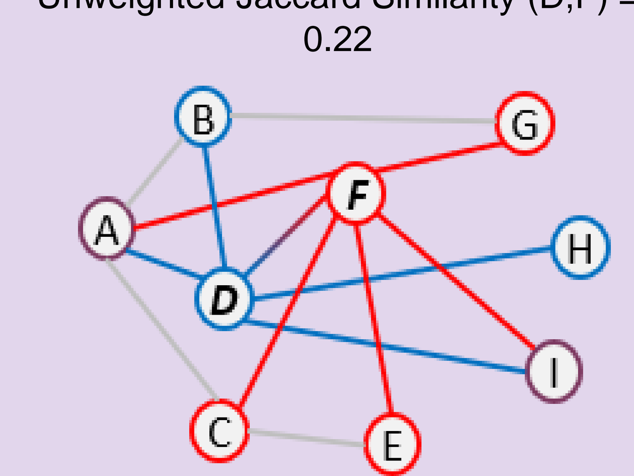
Sum of Pair	
s(A,B) 7	s(C,F) 9
s(A,C) 7	s(D,F) 11
s(A,D) 9	s(D,H) 6
s(A,F) 10	s(D,I) 7
s(B,D) 8	s(E,F) 8
s(B,G) 5	s(F,G) 8
s(C,E) 5	s(F,I) 8

Intersection	
i(A,B) 1	i(C,F) 2
i(A,C) 1	i(D,F) 2
i(A,D) 2	i(D,H) 0
i(A,F) 2	i(D,I) 1
i(B,D) 1	i(E,F) 1
i(B,G) 0	i(F,G) 0
i(C,E) 1	i(F,I) 1

Union
u(A,B) 6
u(A,C) 6
u(A,D) 7
u(A,F) 8
u(B,D) 7
u(B,G) 5
u(C,E) 4
u(C,F) 7
u(D,F) 9
u(D,H) 6
u(D,I) 6
u(E,F) 7
u(F,G) 8
u(F,I) 7

Step 4b: Compute Jaccard Similarity

Unweighted Jaccard Similarity (D,F) = 0.22



Intersection	
i(A,B) 1	i(C,F) 2
i(A,C) 1	i(D,F) 2
i(A,D) 2	i(D,H) 0
i(A,F) 2	i(D,I) 1
i(B,D) 1	i(E,F) 1
i(B,G) 0	i(F,G) 0
i(C,E) 1	i(F,I) 1

Union	
u(A,B) 6	u(C,F) 7
u(A,C) 6	u(D,F) 9
u(A,D) 7	u(D,H) 6
u(A,F) 8	u(D,I) 6
u(B,D) 7	u(E,F) 7
u(B,G) 5	u(F,G) 8
u(C,E) 4	u(F,I) 7

Jaccard Similarity
js(A,B) 0.167
js(A,C) 0.167
js(A,D) 0.286
js(A,F) 0.25
js(B,D) 0.143
js(B,G) 0
js(C,E) 0.25
js(C,F) 0.286
js(D,F) 0.22
js(D,H) 0
js(D,I) 0.167
js(E,F) 0.143
js(F,G) 0
js(F,I) 0.143

Steps 1-4b also discribed in our prior work on FPGA-based Jaccard similarity [2]

Workload Distribution Approach

Source vertex	1	1	1	2	3
Destination vertex	2	3	4	5	6
Intersection size	8	10	5	9	2

Original edge-list:

Perform preprocessing to compute intersection search-size for each vertex pair in the edge list

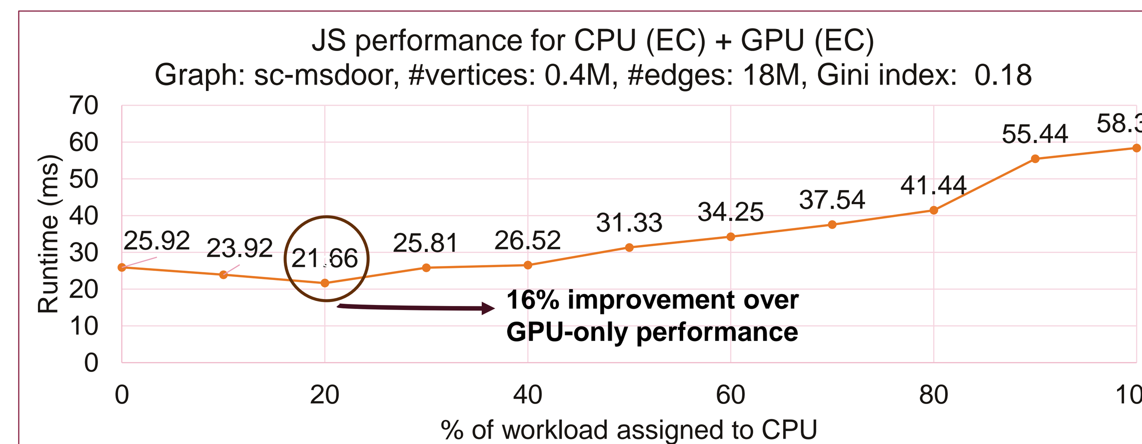
Source vertex	3	1	1	2	1
Destination vertex	6	4	2	5	3
Intersection size	2	5	8	9	10
Mask	0	0	1	1	1

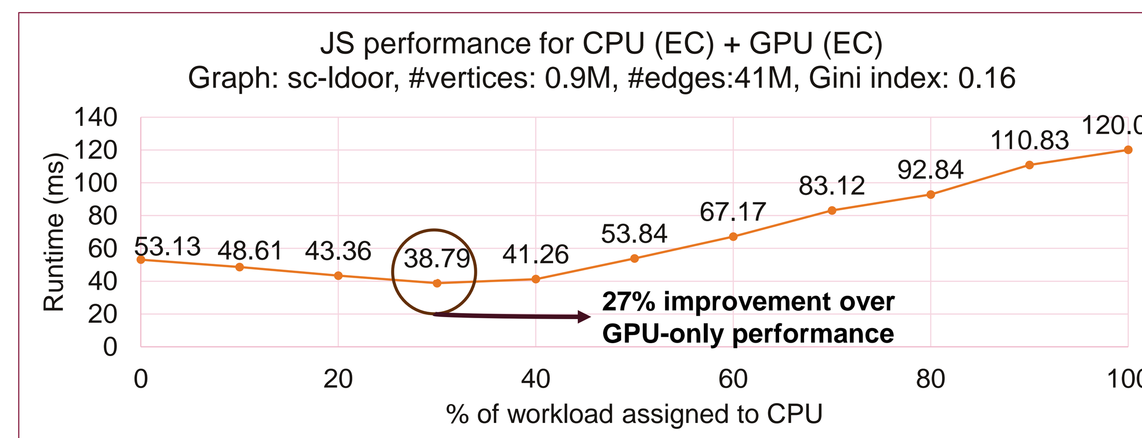
Sort based on the intersection size and assign a mask to each edge (mask = 0 → assign to CPU, 1 → GPU)
Post sorting: Assign a (pre-defined) % of edges to CPU and remaining edges to GPU

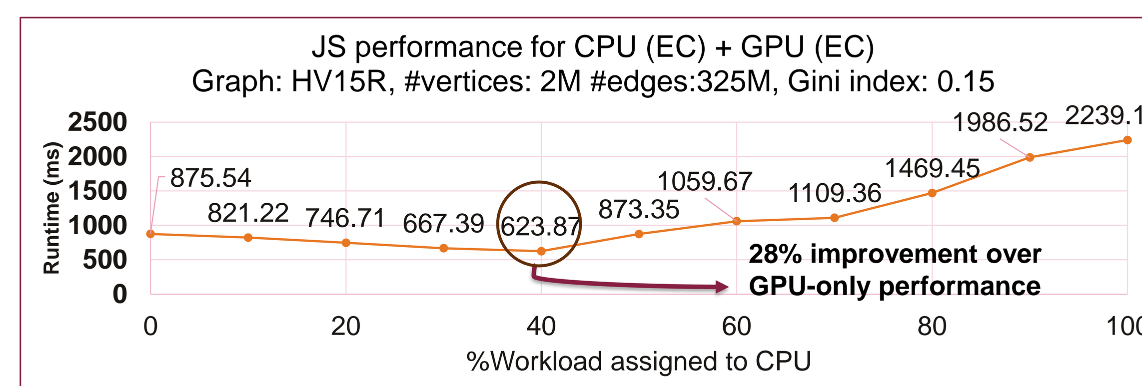
Source vertex	1	1	1	2	3
Destination vertex	2	3	4	5	6
Mask	1	1	0	1	0

Use the original unsorted edge list with the masks assigned in previous and compute Jaccard similarity using hybrid CPU + GPU approach

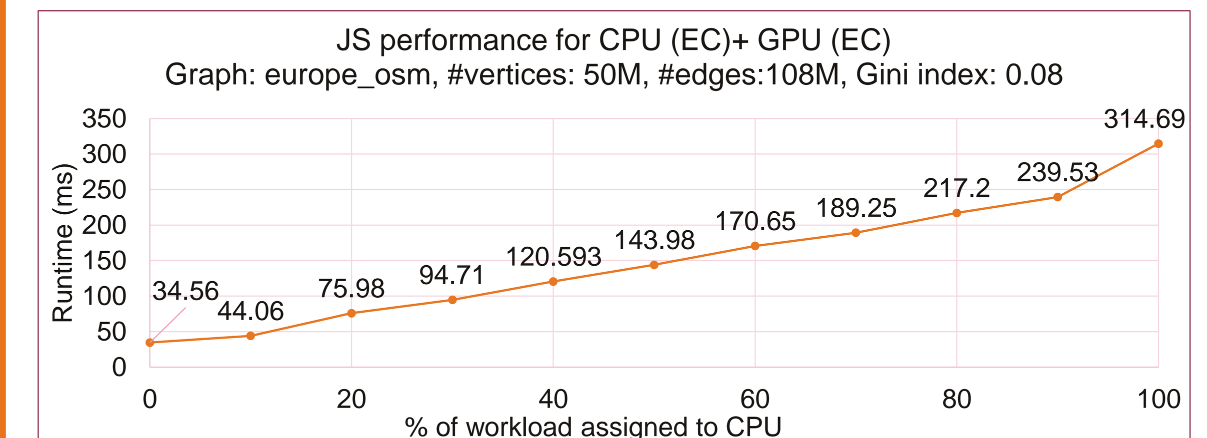
Performance Evaluation of Hybrid CPU-GPU Implementation







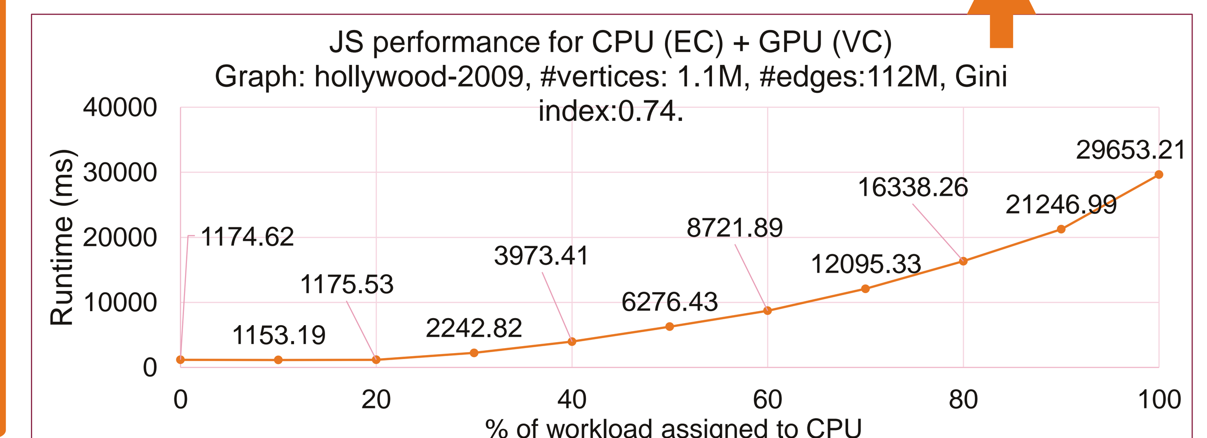
Gini index (GI) → A measure of inequality in the degree distribution.
GI = 0 → even degree distribution.
GI closer to 1→ Few vertices with a very high degree while remaining vertices with low degree



16.4 -28.8% improvement over GPU-only performance of cuGraph intersection pair kernel

Europe_osm: Near-hypersparse graph (avg. degree 2). No improvement over cuGraph intersection pair kernel with hybrid CPU + GPU approach

Hollywood-2009: High avg degree and high GI: Only a marginal improvement over GPU-only performance



Experimental setup for evaluation: CPU → AMD EPYC 7742 64-Core Processor, GPU → NVIDIA A100 @80GB
Graph datasets taken from the network repository[4]

Future Work

- Include the Vertex-centric variant of the intersection kernel from [3] in the implementation
- Automated distribution using classification and regression analysis: Machine learning-based approaches to identify the distribution that yields optimal performance
- Integrate a fast sorting library to reduce preprocessing overhead [6]

References

1. Scogland, T.R.W., Feng, Wc., Rountree, B., de Supinski, B.R. (2014). CoreTSAR: Adaptive Worksharing for Heterogeneous Systems. In: Kunkel, J.M., Ludwig, T., Meuer, H.W. (eds) Supercomputing. ISC 2014. Lecture Notes in Computer Science, vol 8488. Springer, Cham. https://doi.org/10.1007/978-3-319-07518-1_11
2. P. Sathre, A. Gondhalekar and W. -c. Feng, "Edge-Connected Jaccard Similarity for Graph Link Prediction on FPGA," *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, USA, 2022, pp. 1-10, doi: 10.1109/HPEC55821.2022.9926326.
3. cuGraph RAPIDSAI Graph Analytics Library: Jaccard Similarity Computation via Pairwise Intersection - https://github.com/rapidsai/cugraph/blob/branch-23.08/cpp/src/link_prediction/legacy/jaccard.cu#L119
4. Ryan A. Rossi, and Nesreen K. Ahmed 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI.* https://networkrepository.com
5. C. Pearson, M. Almasri, O. Anjum, V. S. Mailthody, Z. Qureshi, R. Nagi,J. Xiong, and W.-m. Hwu, "Update on triangle counting on gpu," in 2019IEEE High Performance Extreme Computing Conference (HPEC), Sep.2019, pp. 1–7.
6. CUB DeviceRadixSort Struct Reference https://nvlabs.github.io/cub/structcub_1_1_device_radix_sort.html

VIRGINIA TECH

SyNeRG