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

Atharva Gondhalekar
atharva1@vt.edu
Dept. of Elec. & Comp. Engg.
Virginia Tech
Blacksburg, Virginia, USA

Paul Sathre
sath6220@cs.vt.edu
Dept. of Computer Science
Virginia Tech
Blacksburg, Virginia, USA

Wu-chun Feng
wfeng@vt.edu
Dept. of Computer Science
Virginia Tech
Blacksburg, Virginia, USA

ABSTRACT

Typical GPU programs consist of four steps: (1) data preparation and preprocessing, (2) host CPU-to-GPU data transfers, (3) execution of one or more GPU kernels, and (4) transfer of results back to the CPU. While the kernel is running on the GPU, the CPU cores often remain idle, waiting on the GPU to finish the kernel execution. It is possible to utilize the idle resources by assigning a portion of the workload to the CPU cores while GPU processes the rest of the workload. In recent years, several frameworks have been presented that perform automated distribution of workload to both CPU and GPU and show performance improvement over GPU-only solutions. While the aforementioned frameworks offer techniques for CPU+GPU workload distribution for regular applications, identifying the CPU+GPU workload distribution that delivers performance improvements over GPU-only solutions for irregular applications remains a difficult problem due in part to the workload imbalance and data-dependent irregular memory access patterns.

This work evaluates a hybrid CPU+GPU implementation of an irregular workload – graph link prediction using the Jaccard similarity index. For the graphs that benefit the most from our hybrid CPU-GPU approach, our implementation delivers a 16.4-28.8% improvement over the state-of-the-art Jaccard similarity implementation from the cuGraph library.

CCS CONCEPTS

• **Computing methodologies** → **Parallel computing methodologies**; **Concurrent computing methodologies**; • **Hybrid CPU-GPU workload distribution**;

KEYWORDS

GPGPU, cuGraph, Irregular Workloads, Hybrid CPU-GPU Workload Distribution, Graph Algorithms, CUDA, OpenMP

1 INTRODUCTION

General-purpose GPU (GPGPU) computing implements an offload model to move expensive computation to the GPU. In many GPGPU computations, the CPU cores remain idle while the GPU programs (known as kernels) are running, leading to CPU core underutilization. In recent years, multiple frameworks have been introduced that offer automated distribution of workloads between CPU(s) and GPU(s) for improved performance over GPU-only solutions. An example of such a framework is CoreTSAR [9], a task-size adapting runtime system that supports work-sharing of loop iterations across heterogeneous resources. While CoreTSAR and similar frameworks offer work-sharing across CPU and GPU for *regular* workloads, distributing the workload for irregular applications remains a challenging task due to the workload imbalance and irregular memory access pattern of such workloads. This work presents a hybrid CPU-GPU implementation of an irregular workload – graph link prediction using the Jaccard similarity index [4]. We evaluate the efficacy of our implementation on a number of real-world graphs with variations in the number of vertices, number of edges, and degree distribution.

The work detailed herein has been supported in part by NSF I/UCRC CNS-1822080 via the NSF Center for Space, High-performance, and Resilient Computing (SHREC) .

2 JACCARD SIMILARITY

In graph analytics and related applications, the Jaccard similarity index can be used to measure the connectedness between two or more vertex pairs to support prediction of new links. The Jaccard similarity between any two sets **A** and **B**, is defined as shown in the Equation (1).

$$\textbf{Jaccard similarity}(\mathbf{A}, \mathbf{B}) = \frac{|\mathbf{A} \cap \mathbf{B}|}{|\mathbf{A} \cup \mathbf{B}|} \tag{1}$$

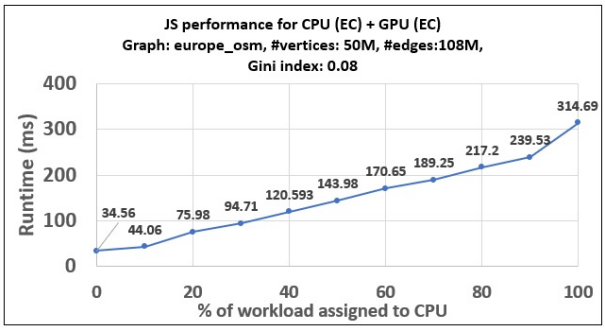
In graph datasets, the Jaccard similarity for any two vertices is computed using the respective neighborhood sets of the vertices. Computing the intersection size of the neighborhood sets is often done using either a binary search, for its good asymptotic performance on unknown data sizes, or a two-pointer approach for more regular memory accesses on smaller data.

2.1 CPU implementation

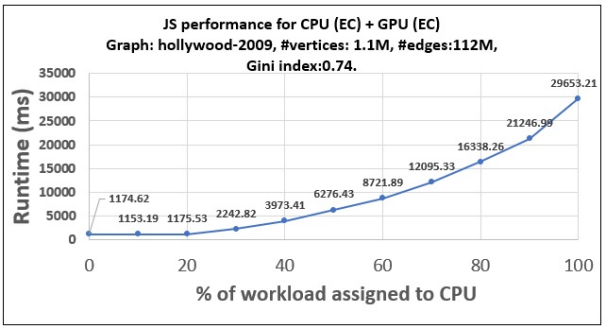
The CPU implementation used in this work uses edge-centric parallelism, where each thread computes the Jaccard similarity score for a unique vertex pair connected by an existing edge. The algorithm terminates when the similarity scores for all edge-connected vertex pairs are computed. Similar to the set intersection approach by Pearson et al. [7], the CPU implementation to compute set intersection selects between the binary search and two-pointer set intersection [7] based on the neighborhood set sizes. We use OpenMP [6] to implement edge-centric Jaccard similarity computation on CPU.

2.2 GPU implementation

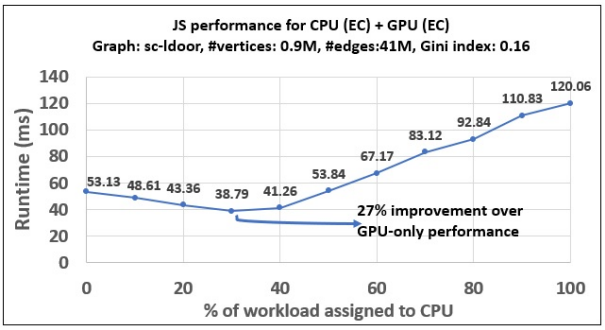
The pairwise intersection kernel [2] from the cuGraph [3] library is used as the GPU implementation for this work. It is an edge-centric kernel with a two-dimensional thread block: threads along one dimension identify the source and destination vertices of an edge and threads along the second dimension check if each source neighbor is present in the destination neighbor list.



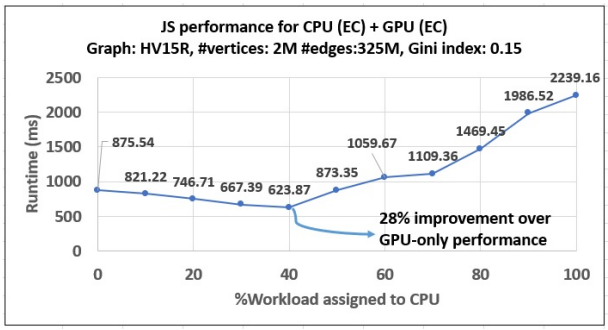
(a) Runtime evaluation for europe_osm graph



(b) Runtime evaluation for hollywood-2009 graph



(c) Runtime evaluation for sc-ldoor graph



(d) Runtime evaluation for HV15R graph

Figure 1: Hybrid CPU-GPU runtime evaluation for Jaccard Similarity (JS)

2.3 Workload distribution

The edge list containing the source and destination vertices is sorted based on the neighborhood sizes of the source and destination vertices. A configurable portion of the edges, starting from the sparser end of the list, is then marked for execution on the CPU while the remaining denser edges are marked for processing on GPU. The edges are then assigned to respective targets but using the original edge list and computed vector masks.

3 EVALUATION

Fig. 1 shows the runtime evaluation of the hybrid CPU-GPU implementation. The graph datasets are taken from the network repository [8]. The CPU used in the evaluation is an AMD EPYC 7742 64-core processor, and the GPU is an NVIDIA A100. We evaluate our implementation on graphs with varied characteristics such as the number of edges, number of vertices, and degree distribution, which is captured by the *Gini index* [5] metric, which lies in the range [0,1). A Gini index closer to 1 indicates that a small number of vertices in the graph have a very high degree, while most of the vertices have a low degree, indicating a workload imbalance. Conversely, a near-zero Gini index indicates an almost even degree distribution. From the evaluation, we observe that some graphs do not benefit from a CPU+GPU implementation, while some graphs show as much as a 28.8% improvement. As a part of the extension to this work, we intend to present a heuristic based on the aforementioned graph properties to determine the following.

(1) Whether a CPU+GPU implementation can outperform GPU-only performance for a given input graph
(2) How to split the workload to achieve performance improvements over GPU-only performance

4 CONCLUSION

This work presents a hybrid CPU+GPU implementation of graph link prediction using the Jaccard similarity index. For the graphs that benefit the most from our hybrid approach, our implementation achieves 16.4%-28.8% improvement over the state-of-the-art Jaccard similarity implementation from the cuGraph library [2].

5 FUTURE WORK

As a subject of future study, we intend to deploy a heuristic-based approach to determine CPU + GPU workload distribution that delivers the optimal performance relative to the GPU-only performance. Such a heuristic would be based on the properties of the input graph, for ex., avg. degree, Gini index, minimum, and maximum degree. We also intend to optimize the preprocessing steps such as sorting using existing optimized libraries such as CUB [1].

REFERENCES

[1] [n.d.]. CUB - Device Radix Sort. GitHub IO. https://nvlabs.github.io/cub/structcub_1_1_device_radix_sort.html
[2] [n.d.]. cuGraph - Legacy Jaccard Similarity via Pairwise Intersection Kernel. GitHub. https://github.com/rapidsai/cugraph/blob/branch-23.08/cpp/src/link_prediction/legacy/jaccard.cu#L119
[3] [n.d.]. cuGraph - RAPIDS Graph Analytics Library. GitHub. https://github.com/rapidsai/cugraph

[4] Paul Jaccard. 1912. The Distribution of the Flora in the Alpine Zone.1. *New Phytologist* 11, 2 (1912), 37–50. https://doi.org/10.1111/j.1469-8137.1912.tb05611.x arXiv:https://nph.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-8137.1912.tb05611.x

[5] Jérôme Kunegis and Julia Preusse. 2012. Fairness on the Web: Alternatives to the Power Law. In *Proceedings of the 4th Annual ACM Web Science Conference* (Evanston, Illinois) *(WebSci '12)*. Association for Computing Machinery, New York, NY, USA, 175–184. https://doi.org/10.1145/2380718.2380741

[6] OpenMP Architecture Review Board. 2008. OpenMP Application Program Interface Version 3.0. http://www.openmp.org/mp-documents/spec30.pdf

[7] Carl Pearson, Mohammad Almasri, Omer Anjum, Vikram S. Mailthody, Zaid Qureshi, Rakesh Nagi, Jinjun Xiong, and Wen-mei Hwu. 2019. Update on Triangle Counting on GPU. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–7. https://doi.org/10.1109/HPEC.2019.8916547

[8] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. http://networkrepository.com

[9] Thomas R.W. Scogland, Wu-Chun Feng, Barry Rountree, and Bronis R. de Supinski. 2015. CoreTSAR: Core Task-Size Adapting Runtime. *IEEE Transactions on Parallel and Distributed Systems* 26, 11 (2015), 2970–2983. https://doi.org/10.1109/TPDS.2014.2365192