HyGraph: High-Performance Graph Processing on Hybrid CPU-GPU Platforms

Stijn Heldens
University of Twente
The Netherlands
s.j.heldens@utwente.nl

Ana Lucia Varbanescu
University of Amsterdam
The Netherlands
a.l.varbanescu@uva.nl

Alexandru Iosup
Delft University of Technology
The Netherlands
a.iosup@tudelft.nl

VU University Amsterdam
The Netherlands
a.iosup@vu.nl

I. Introduction

Graphs are fundamental for describing a set of entities (*vertices*) and the relations (*edges*) between them. Many real-world datasets can be represented as a graph, such as the World Wide Web and social networks. Algorithms that process these datasets are plentiful, examples are product recommendation [1] and discovering "communities" in social networks [2]. Analysis of large-scale graphs is becoming increasingly important in many domains, such as biology, sociology and data mining.

To cope with the ever increasing size of these graph datasets, many different large-scale graph processing systems have been proposed. Most research focuses on systems for distributed clusters (e.g., Giraph [3], GraphPad [4], GraphX [5], and PowerGraph [6]). To increase performance of these systems, research has also focused on graph processing on accelerators such as GPUs (e.g., CuSha [7], Medudsa [8], and Gunrock [9]). Performance can be increased even further by focusing on *hybrid platforms*, i.e, using *both* CPU and GPU *simultaneously* for graph processing. TOTEM [10] is the state-of-the-art in graph processing on hybrid platforms. There has been little research that further explores this idea.

II. Problem Description

To divide a graph over the available resources, TOTEM uses *static* workload partitioning, i.e., each vertex of the graph is assigned to either CPU or GPU. The partitioning policy and the size of the partitions needs to be defined by the user. Each vertex is assigned only once during graph loading and vertices cannot be migrated during execution.

Finding the optimal partitioning is crucial for obtaining performance. An incorrect partitioning can lead to workload imbalance, where one device is overloaded while the other is underutilized. This results in sub-optimal performance.

However, graph algorithms are inherently irregular applications and execution is data-driven. Some combinations of graph dataset and graph algorithm are more suitable for CPUs, while other are more suitable for GPUs. This makes finding a suitable partitioning for TOTEM challenging. The optimal partitioning differ per application, per algorithm, and, for iterative algorithm, even per iteration.

III. Design of HyGraph

We present HyGraph[1] [11], a graph-processing system for hybrid platforms that tackles the problem of workload balancing. Instead of static partitioning, HyGraph uses a smart replication policy which allows for *dynamic* scheduling. Vertices get divided into fixed-sized segments called *blocks*. Each block corresponds to one processing job which can be performed by either the CPU or the GPU in each iteration. A global scheduler assigns jobs to the CPU and the GPU in a dynamic fashion. This automatic dynamic scheduling strategy provides load-balancing and completely removes the need to manually define a partitioning policy.

Besides solving workload imbalance, HyGraph is also highly configurable and offers the following features:

- Support for various execution modes: CPU-only, GPU-only, CPU+GPU, and CPU+multiple GPUs.
- High-level vertex-centric programming model. Each vertex program needs to be define once, from which HyGraph generates code for both CPU and GPU supporting different data structures (vertex/edge-based) and access patterns (push/push).
- Communication to copy data between CPU and GPU is overlapped with computation, thus removing the high overhead of these data transfers.
- Enables GPU acceleration even for datasets which exceeds GPU memory. This is achieved by only uploading as many blocks as fit into GPU memory.
- Customizable scheduler allows for different policies. Both static and dynamic scheduling policies are povided by default.

IV. Experimental Results

We have evaluated HyGraph on the DAS5 [12] for seven real-world and synthetic datasets. The host machine contains dual Intel Xeon E5-2630 CPU (2×8 cores), has 64GB of memory, connects to the GPU via PCIe 3.0 16x, and runs CentOS 6. The GPU is a NVIDIA Kelper K40c with 12GB memory. We have implemented four representative graph algorithms: Breadth-first search (BFS), PageRank (PR), Connected components (CC), and Single-Source Shortest Paths (SSSP).

[1]Source code available at: https://github.com/atlarge-research/HyGraph

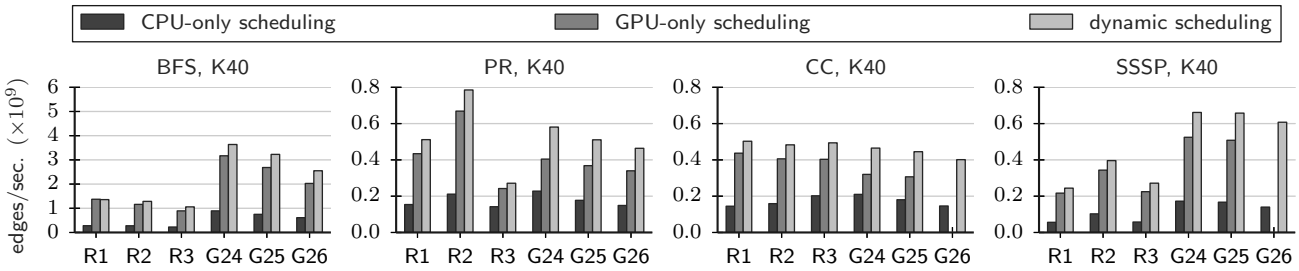

Fig. 1: Performance of HyGraph for different datasets and algorithms for CPU-only, GPU-only, and dynamic scheduling (CPU+GPU). Higher is better. Note differences in scales on vertical axes.

TABLE I: Overview datasets used for evaluation. Storage is the size of the file which stores the edge list in binary format.

ID	Name	Origin	Vertices	Edges	Storage
R1	LiveJournal	Social net.	4.85M	69.0M	0.51GB
R2	Orkut	Social net.	3.07M	117M	0.87GB
R3	Wikipedia	WWW	12.1M	378M	2.8GB
R4	Friendster	Social net.	65.6M	1.81B	16GB
G24	RMAT-24	Synthetic	8.9M	260M	1.9GB
G25	RMAT-25	Synthetic	17.1M	524M	3.9GB
G26	RMAT-26	Synthetic	32.8M	1.05B	7.8GB

Figure 1 compares the performance of HyGraph when using only the CPU, only the CPU, or combing both CPU and GPU using dynamic scheduling. Measurements are reported as billion edges per seconds (EPS), obtained by dividing the graph size by algorithm run-time. Higher values are better.

The results show that using GPU is always faster than using the CPU, average speedup is 2.0x. However, combing both CPU and GPU is even faster than only using the GPU. The decrease in execution time is up to 16.8% for BFS, 37.3% for PR, 32.4% for CC, and 22.7% for SSSP.

Figure 2 shows the performance of HyGraph against state-of-the-art CPU-based (GraphPad), GPU-based (CuSha, Gunrock), and hybrid systems (TOTEM). Measurement are reported as total execution time, lower is better. Results show that HyGraph outperforms all others. The only execption is Cusha for G23, but this system uses a complex data structure and cannot handle large graphs such as G26. For GraphPad, a distributed system, one needs to employ at least 4 machines to outperform HyGraph.

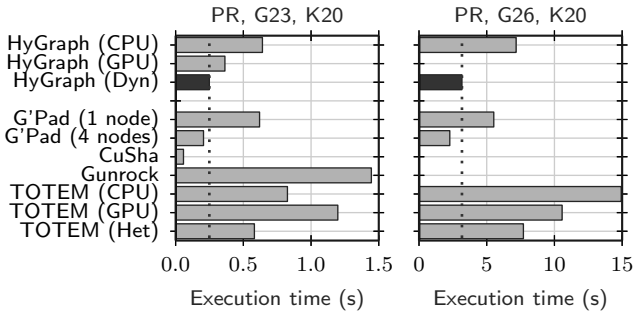

Fig. 2: Performance of HyGraph against state-of-the-art.

V. Conclusions & Future Work

Large-scale graph processing is becoming increasing important in many fields of study. In this work, we present HyGraph: a novel graph processing system that targets hybrid CPU-GPU platforms. HyGraph uses dynamic scheduling to overcome the workload imbalance problems that arise from static partitioning. Overall, our solutions shows good speedups over using only using CPU or only using GPU. It even outperforms many state-of-the-art systems. For the future, we plan to extend our work to support distributed clusters. We also looking into implementing complex graph algorithms, such as community detection and triangle counting.

References

[1] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *WWW2001*. ACM, 2001, pp. 285–295.

[2] S. Fortunato, "Community detection in graphs," *Physics reports*, vol. 486, no. 3, pp. 75–174, 2010.

[3] C. Avery, "Giraph: Large-scale graph processing infrastructure on hadoop," *Proceedings of the Hadoop Summit. Santa Clara*, vol. 11, 2011.

[4] M. J. Anderson, N. Sundaram, N. Satish, M. Patwary, T. L. Willke, and P. Dubey, "GraphPad: optimized graph primitives for parallel and distributed platforms."

[5] R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "Graphx: A resilient distributed graph system on spark," in *GRADES*. ACM, 2013, p. 2.

[6] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *OSDI*, 2012, pp. 17–30.

[7] F. Khorasani, K. Vora, R. Gupta, and L. N. Bhuyan, "CuSha: vertex-centric graph processing on GPUs," in *HPCS*. ACM, 2014, pp. 239–252.

[8] J. Zhong and B. He, "Medusa: Simplified graph processing on gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1543–1552, 2014.

[9] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the GPU," in *SIGPLAN*. ACM, 2016, p. 11.

[10] A. Gharaibeh, E. Santos-Neto, L. B. Costa, and M. Ripeanu, "Efficient large-scale graph processing on hybrid CPU and GPU systems," *CoRR*, vol. abs/1312.3018, 2013.

[11] S. Heldens, A. L. Varbanescu, and A. Iosup, "Dynamic load balancing for high-performance graph processing on hybrid cpu-gpu platforms," in *Proceedings of the Sixth Workshop on Irregular Applications: Architectures and Algorithms*. IEEE Press, 2016, pp. 62–65.

[12] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff, "A medium-scale distributed system for computer science research: Infrastructure for the long term," *Computer*, vol. 49, no. 5, pp. 54–63, 2016.