Automating Vectorized Distributed Graph Computation (Technical Perspective)

Julian Shun
MIT CSAIL
jshun@mit.edu

Graph data, which are used to model relationships between objects of interest, arise in many important applications today. For example, they are used to model human relationships in social networks, links among Web pages on the Internet, transactions in financial networks, interactions among proteins, genes, and molecules in biological/chemical networks, meshes in scientific simulations, particle interactions in physical/quantum systems, relationships among galaxies in astronomical studies, communication in cyber networks, and links in geospatial data. Analyzing the structure and properties of these graphs is an important component of many data analytics pipelines.

With the explosion in the volume of data, graphs have become very large and can contain hundreds of billions of vertices and trillions of edges. It is therefore crucial to design scalable solutions to enable graph analysis to be done efficiently. With the ubiquity of parallel machines today, using parallelism to speed up graph processing has become standard. Over the past several decades, parallel graph algorithms have been proposed for different classes of hardware platforms (CPUs, GPUs, distributed clusters, domain-specific accelerators, etc.). However, due to the complexity of parallel programming, developing efficient algorithms is time-consuming and is only accessible to a small subset of programmers who are experts in parallelism.

To make parallel graph processing accessible to non-experts, high-level programming frameworks have been proposed. These solutions provide high-level APIs that users can use to write parallel graph algorithms without having to deal with most of the complexities of parallel programming. Numerous graph processing frameworks have been proposed over the past two decades, targeting different types of hardware platforms, graph workloads, and graph data.

One kind of graph workload involves running multiple instances of a graph algorithm (e.g., with different source vertices). For example, one may be interested in computing shortest paths or local clusters from a collection of source vertices. A simple approach for tackling such workloads is to run an algorithm for each source vertex independently. However, it is often the case that the different algorithm instances involve some redundant computation, which can be avoided by interleaving the runs together. To this end, several multi-instance graph algorithms have been proposed in the literature, and they show promising speedups over the simple baseline of running the instances independently. However, existing solutions require significant expertise to develop and understand, even when they are part of programming frameworks.

To address this challenge, the following paper [2] presents AutoMI, a framework that simplifies the task of developing distributed multi-instance graph algorithms. AutoMI automatically transforms vertex-centric graph algorithms written in the GAS (Gather-Apply-Scatter) programming model [1] into multi-instance algorithms with SIMD vectorization.

The paper first shows that a naive conversion of a single-instance graph algorithm into a multi-instance graph algorithm with SIMD vectorization can lead to programs that produce incorrect results. Then, it presents the AutoMI framework that ensures correct program generation. AutoMI creates an abstract syntax tree (AST) from the input program and then generates a vectorized multi-instance program according to the AST. The approach, by default, generates code for tracking the visited states of vertices. However, the authors observe that if a program satisfies a certain kind of idempotence, then the tracking information is not required for correctness. This leads to simpler and faster code. The notion of idempotence that is required for bypassing the tracking code is formalized via an algebraic characterization of GAS programs. The authors use AutoMI to generate multi-instance graph algorithms for integer breadth-first search (BFS), boolean BFS, single-source shortest paths, graph coloring, sparse matrix-vector multiplication, personalized PageRank, and collaborative filtering.

Multi-instance graph algorithms have many important applications, and the AutoMI framework is one of the latest additions to the collection of tools available for programming such algorithms. It would be interesting to see how this paper's ideas can be extended to other kinds of graph programs and hardware platforms.

1. REFERENCES

[1] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, page 17–30, 2012.

[2] W. Zhao, Y. Cao, P. Buneman, J. Li, and N. Ntarmos. Automating vectorized distributed graph computation. *Proc. ACM Manag. Data*, 2(6), Dec. 2024.