# UC Davis
## UC Davis Previously Published Works

**Title**
Accelerating GPU betweenness centrality

**Permalink**
https://escholarship.org/uc/item/9tn2q0ks

**Journal**
Communications of the ACM, 61(8)

**ISSN**
0001-0782

**Authors**
McLaughlin, Adam
Bader, David A

**Publication Date**
2018-07-23

**DOI**
10.1145/3230485

Peer reviewed

rh

# Technical Perspective
# Graphs, Betweenness Centrality, and the GPU

By John D. Owens

GRAPHS ARE THE natural data structures to represent relationships, and in our age of big data, graphs are very big indeed. For instance, Facebook's social graph has well over two billion users (vertices in the graph), and their friendships (edges in the graph) may number in the hundreds of billions. How do we make sense of data this large?

If possible, we can gain significant insight into complex problems of interest both to commerce and to science. Through graph data, we may be able to detect anomalies (say, intrusions into a computer network), make recommendations (say, which movie to watch), search a graph for patterns (say, credit card fraud), or detect communities (say, identifying proteins within a cell with similar functionality). Enabling faster graph computation allows us to find answers to these questions more quickly and cheaply.

As the graphics processor (GPU) has become ubiquitous in personal computers, supercomputers, and more recently datacenters, its advantages in raw performance and price-performance have motivated its use in graph computation. A significant body of recent research has demonstrated the performance advantages of GPUs over CPUs on a variety of graph computations. However, the GPU presents several challenges to authors of efficient graph implementations:

▸ To be effective on any problem, GPUs require large, parallel workloads. Thus GPU application authors must identify and expose significant parallelism in their applications. Fortunately, most graph computations allow parallelization over the graph's vertices, and large graphs exhibit more than enough parallelism to make GPUs a viable choice.

▸ However, graphs are particularly challenging because of the load imbalance across vertices. Some vertices have few neighbors, while others have many. A straightforward parallelization that assigns vertices to different processing units means units assigned vertices with few neighbors are idle while waiting for heavily loaded units to finish. The resulting *load balance problem* is perhaps the most significant challenge in writing an efficient graph computation.

▸ GPUs have modest-sized memories, and the largest graphs of interest cannot fit in a single GPU's memory. Distributing work across multiple GPUs faces two problems: efficiently partitioning both the data and computation across the GPUs in a load-balanced way, and structuring the multi-GPU computation so that the resulting communication between GPUs does not become a bottleneck.

The following work by McLaughlin and Bader ably addresses these challenges in the important context of a graph computation called *betweenness centrality* (BC). Centrality metrics on a graph ascertain the most important nodes in that graph. Betweenness centrality—perhaps the most popular centrality metric—does so by counting how many shortest paths in the graph flow through a particular node. For instance, we may wish to know the most important airports in the world. Betweenness centrality would consider every possible pair of airports and compute the fastest route between each pair; airports involved in the fastest routes would then be the most important.

While the straightforward method for computing betweenness centrality (individually compute the shortest paths between all pairs) would be quite expensive for large graphs, the much cheaper formulation of Ulrik Brandes (2001) is the basis for any modern computation of betweenness centrality, including the following paper. Its efficient parallelization on GPUs is a significant challenge and the focus of this work.

In my view, this paper offers three key insights:

1. The authors describe two different GPU parallelizations of betweenness centrality. Their "work-efficient" approach assigns only active vertices to processing units; their "edge-parallel" approach instead assigns edges to processing units.

2. They analyze both methods through the lens of different types of graphs. Large-diameter graphs with a uniform out-degree are well suited for the work-efficient approach, while the edge-parallel approach is a better fit for scale-free (small-world) graphs. The authors show how to choose the right approach at runtime by first sampling the graph to estimate graph diameter and then choosing the better approach to compute BC on the entire graph.

3. They also identify coarser parallelism in the overall computation that allows them to distribute work across multiple GPUs and demonstrate near-linear speedup on a 192-GPU cluster.

These contributions are crucial building blocks for future work on GPU graph computation. For BC, important next steps include incremental computations on mutable graphs and multi-GPU scaling to graphs that do not fit into GPU memory. More broadly, while work in GPU graph analytics today generally focuses on relatively simple graph problems, real-world workloads are more complex. Our community must move toward frameworks that address these more complex problems that deliver both high performance and high-level programmability. ⓒ

**John D. Owens** is the Child Family Professor of Engineering and Entrepreneurship in the Department of Electrical and Computer Engineering at the University of California, Davis, CA, USA.