Gunrock: A High-Performance, Data-Centric Abstraction for GPU Graph Computation

John Owens
Child Family Professor of Engineering and Entrepreneurship
Department of Electrical and Computer Engineering
UC Davis
w/ Yangzihao Wang, Yuechao Pan, Yuduo Wu, Carl Yang, Leyuan Wang, Mohamed Ebeida, Chenshan Shari Yuan, Weitang Liu

Slides at http://preview.tinyurl.com/owens-nv-webinar-160426

Graphs

ted:

ds

are

ls.

API

et

n,

Twitter Dataset 1 Overview

Tweets: 292.7 Million +
Unique Users: 7,619,916
Total Size: 232 GB

Figure 1: Collection profile of Twitter Dataset 1

Twitter Dataset 2 Overview

Tweets: 1 Billion+
Unique Users: 94 Million+
Geolocated Tweets: 31 Million+
Total Size: 146 GB

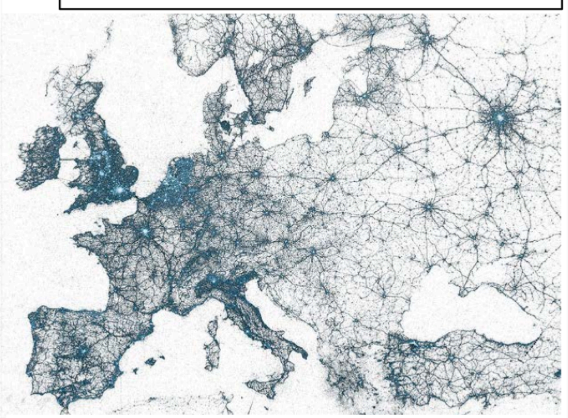


Figure 2: Twitter in Europe

Image obtained from https://blog.twitter.com/2013/geography-tweets-3

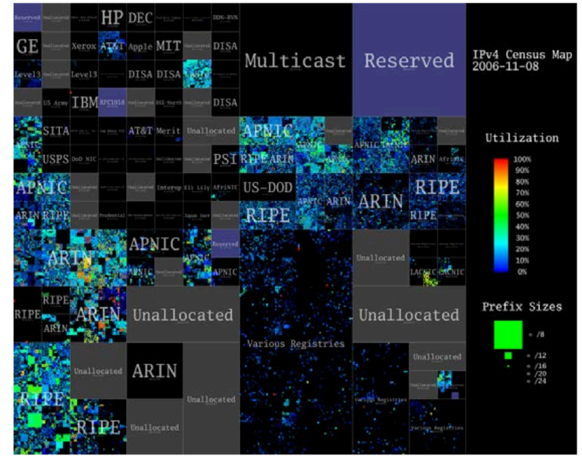


Figure 1: IPv4 Census Map (http://www.caida.org/research/id-consumption/census-map/images/20061108.png)

Data Type	# of Records (Size)	Quick Description
Service Probes	180 billion (5.5 TB)	Results of probes with different formats sent to various service ports of IPv4 addresses.
Reverse DNS	10.5 billion (366 GB)	Results of DNS name requests (reverse lookups) for addresses within the IPv4 space using 16 large DNS Servers.
TCP/IP Fingerprints	80 million (50 GB)	Results of remote OS detection fingerprinting from NMap tool.

Table 1: Net Data Volume

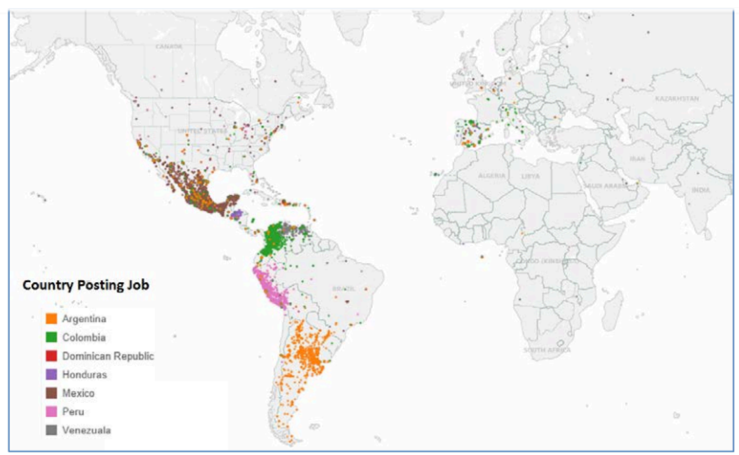


Figure 1: Map of Jobs (Colored by Country)

ckground and Formats: The dataset consists of 119+ Million bs and is about **40 GB** in size. There are approximately 2.1 illion unique jobs in the set as many records are duplicates. To

Data Field	Example
Posted Date	2012-10-23
Location	Capital Federal
Department	Capital Federal

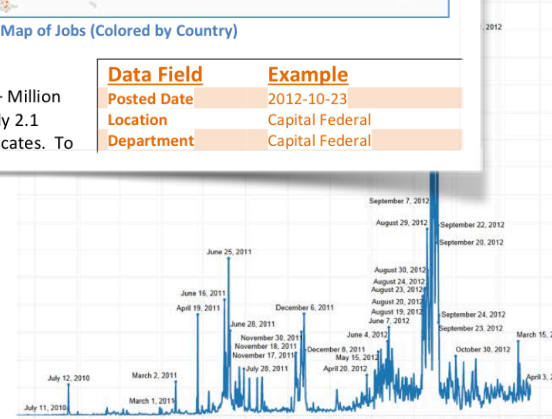


Figure 1: Bitcoin Transactions Over Time

Bitcoin Data Set Overview (May 15, 2013)

# Transactions:	15.8 Million+
# Edges:	37.4 Million +
# Senders:	5.4 Million+
# Receivers:	6.3 Million+
# Bitcoins Transacted:	1.4 Million +

The Ninja Problem

"I believe that in the datacenter, one question is critical: If you can't get to peak performance on GPUs, they basically lose all their value proposition. So how can you get close to peak without becoming an architecture expert and programming/ performance wizard?"

—Anonymous, Large Internet Company, 27 May 2014

Gunrock Genesis

- Summer 2013, DARPA XDATA summer camp
- Focus: to-the-metal GPU graph implementations
- 8 weeks to write (port) betweenness centrality
- Not a sustainable model!

Gunrock: Goals

- Bottom-up: To leverage the highest-performing GPU computing primitives for efficiency.
- Top-down: To be expressive enough to represent a wide variety of graph computations for usability.

Gunrock Status

- Open-source release (Apache 2.0), currently version 0.3
- http://gunrock.github.io/
- Fastest programmable GPU library for graph analytics
  - Superior load-balancing/work distribution
  - More powerful abstraction

Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, and John D. Owens. ***Gunrock: A High-Performance Graph Processing Library on the GPU****. ACM PPoPP 2016. Distinguished Paper. http://escholarship.org/uc/item/6xz7z9k0*

Other programmable GPU frameworks …

- … leverage a *bulk-synchronous* model
- … use CPU abstractions:
  - Pregel (Medusa)
  - GAS (VertexAPI2, CuSha, MapGraph)
- … organize steps of *computation*, with two significant disadvantages:
  - Programming models are not very general
  - Kernels are small and miss opportunities for producer-consumer locality

Gunrock: Programming Model

- Graph represented as CSR (~ sparse matrix)
- **Bulk-synchronous**: series of parallel *steps* (operations) separated by global barriers
- **Data-centric**: All operations are on one or more *frontiers* of active vertices/edges
  - **Advance**: generates a new frontier through visiting the neighbor vertices/edges of elements in the current frontier. Key: Work distribution/load balancing
  - **Filter**: removes elements from frontier via validation test
  - **Compute**: user-defined vertex-centric or edge-centric computations that run in parallel

Gunrock: Programming Model

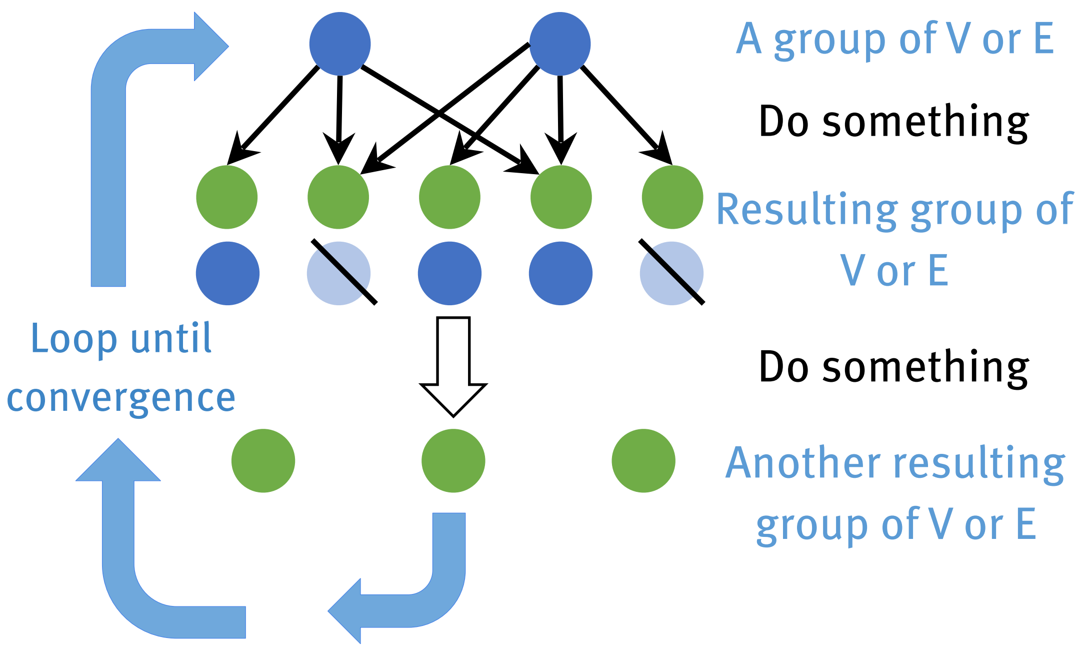
- Graph represented as CSR (~ sparse matrix)
- **Bulk-synchronous**: s s) separated by global
- **Data-centric**: All ope *ers* of active vertices/ed
  - **Advance**: generates a new frontier through visiting the neighbor vertices/edges of elements in the current frontier. Key: Work distribution/load balancing
  - **Filter**: removes elements from frontier via validation test
  - **Compute**: user-defined vertex-centric or edge-centric computations that run in parallel

Considering new operators:

- Global
- Neighborhood
- Sampling
- Frontier-frontier intersection

Gunrock's Data-Centric Abstraction & Bulk-Synchronous Programming

A generic graph algorithm:



- **Data-centric abstraction**
  – *Operations* are defined on a group of vertices or edges = a *frontier*
  – Operations = manipulations of one or more frontiers
- **Bulk-synchronous programming**
  – Operations are done one by one, in order
  – Within a single operation, computing on multiple elements can be done in parallel, without order

Using Gunrock

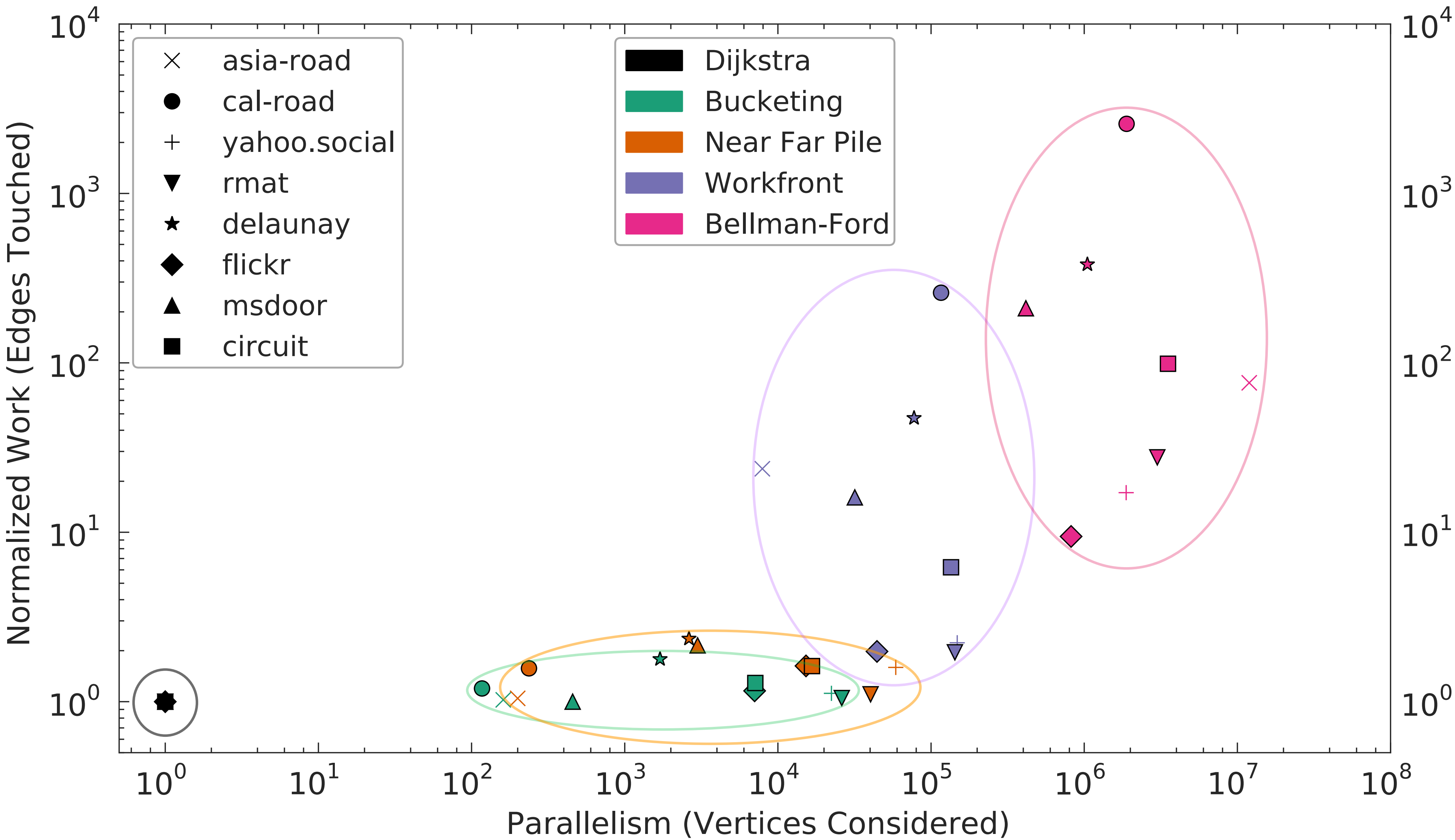
- As a *programmer* …
  - Write your own Gunrock primitives (using advance, compute, filter)
  - Write your own Gunrock operators!

- As an *end-user* …
  - Write an executable that runs Gunrock primitives
  - Link against a Gunrock library that provides Gunrock primitives (C linkage)
    - python
    - Julia

Graph challenges on GPUs

- Efficient parallel algorithms
  - Different balance between brute-force and elegant than on CPUs (next slide)
- Load-balancing due to irregularity
- Moving beyond simple algorithms
- Graph representations
- Scalability (memory constraints)

Algorithm example: SSSP



Andrew Davidson, Sean Baxter, Michael Garland, and John D. Owens. ***Work-Efficient Parallel GPU Methods for Single Source Shortest Paths****. In Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium, May 2014.*

Currently Supported Primitives

- Currently have over 10 graph primitives including:
  - Traversal-based (e.g., BFS, DOBFS, SSSP)
  - Node-ranking (e.g., HITS, SALSA, PageRank)
  - Global (e.g., connected component, MST, triangle-counting)

- LOC under 300 for each primitive, under 10 to use a primitive
- In progress:
  - Graph coloring, Maximal Independent Set
  - Community Detection
  - Subgraph Matching

Industry interest examples

- Twitter: "Who To Follow" service
  - Historically: SALSA ("Stochastic Approach for Link-Structure Analysis")
  - Personalized PageRank generates circle of trust
  - Hubs & authorities, random walks
- Facebook
  - PageRank & Personalized PageRank
  - Label propagation
  - Graph embeddings into R^n so similar nodes are close

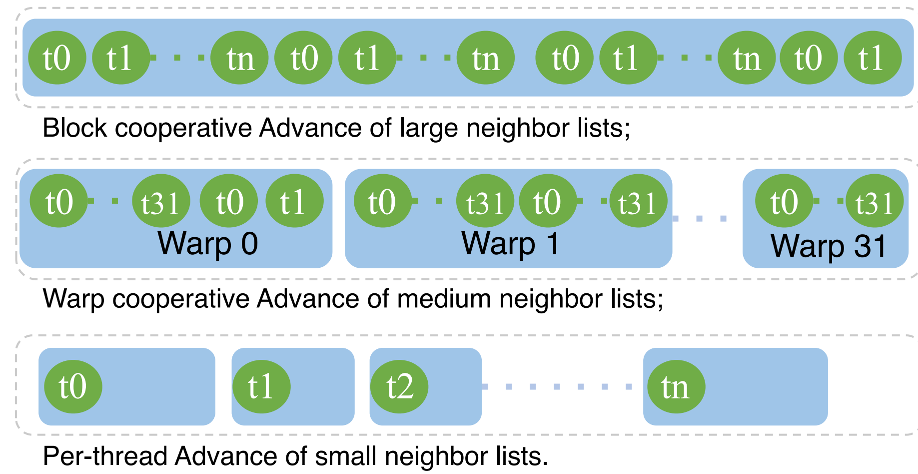
Industry interest examples

- Twitter: "Who To Follow" service
  - Historically: SALSA ("Stochastic Approach for Link-Structure Analysis")
  - Personalized PageRank generates circle of trust
  - Hubs & authorities, random walks
- Facebook
  - PageRank & Personalized PageRank
  - Label propagation
  - Graph embeddings into R^n so similar nodes are close

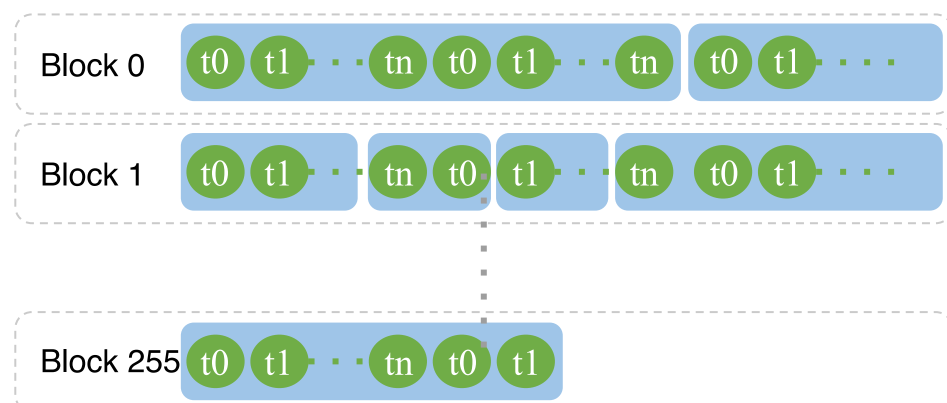
Afton Geil, Yangzihao Wang, and John D. Owens. ***WTF, GPU! Computing Twitter's Who-To-Follow on the GPU****. In Proceedings of the Second ACM Conference on Online Social Networks, COSN '14, pages 63–68, October 2014.*

Load-Balanced Traversal

- Problem: Lots of parallelism across vertices, but each vertex has a different number of neighbors
- Merrill: Depending on size of worklist, vertex work mapped to one {thread, warp, block}
- Davidson: Instead of allocating vertices to threads, allocate edges to threads
  - Requires sorted search to find start and endpts of edges
- Gunrock advantage: Best load-balancing (2–20x over Medusa)



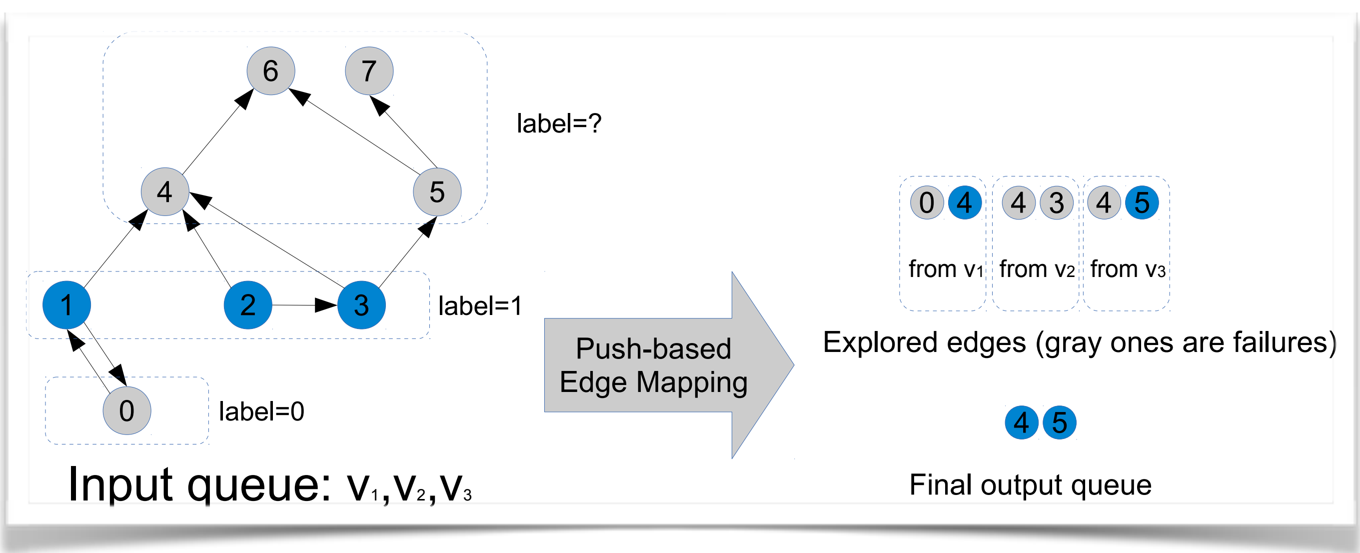
Merrill's per-{thread,warp,CTA} load balance



Davidson's load-balanced partitioning

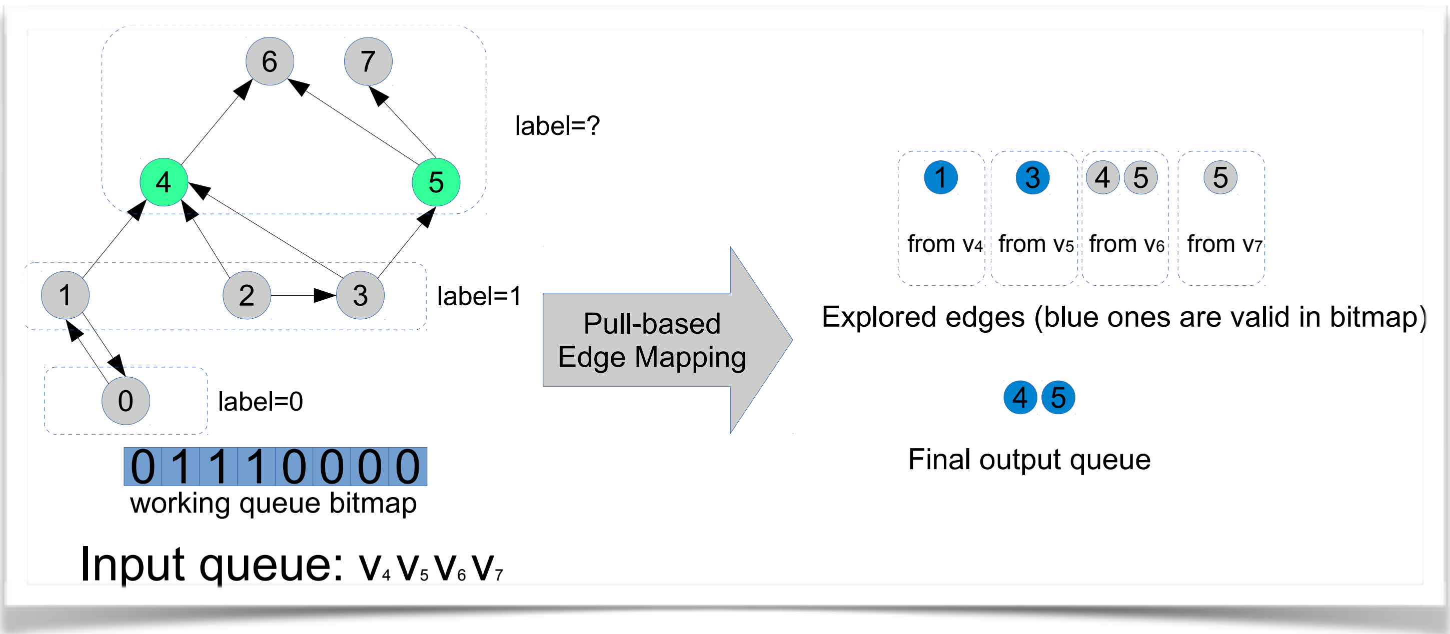
Push vs. Pull (Direction-Optimized Breadth-First Search)

- Normal operation ("push"): vertex frontier visits every outbound edge, generates list of connected unvisited vertices
  - Works great when you're expanding: more new vertices than old
  - Works poorly with few unvisited vertices



Push vs. Pull

- We also support pull: Start with unvisited vertices, check which have inbound edges from frontier



- Difficult to express in compute-focused APIs
- Gunrock: Frontier is unvisited vertices; pull from that frontier

Supporting Priority Queues

- Our SSSP (standalone) implementation compares three data structures for work queues:
  - *Workfront*: All active vertices only (big improvement over Bellman-Ford)
  - *Bucketing*: Closest to delta-stepping: vertices sorted by distance from source, placed into buckets
  - *Near-far*: 2 buckets, “near” and “far”
- Near-far is best: cost of multisplit was too high on GPU (reorganizational overhead: 82% of runtime). (We published a paper on multisplit at PPoPP 2016.)
- Difficult on compute-focused APIs, but in Gunrock we can just have multiple active frontiers, one per bucket

Research Directions: Broader Graph Types

- Bipartite graphs (SALSA, matching, link prediction, personalized PageRank)
- Streaming graphs
- Mutable graphs
  - Graphs that change as a result of the computation (Borůvka minimum-spanning-tree, Delaunay triangulation)
  - Graphs that require modifying the graph to compute (Karger's mincut)
  - In general, **significant data structure challenges**.
    - Is CSR the right format?

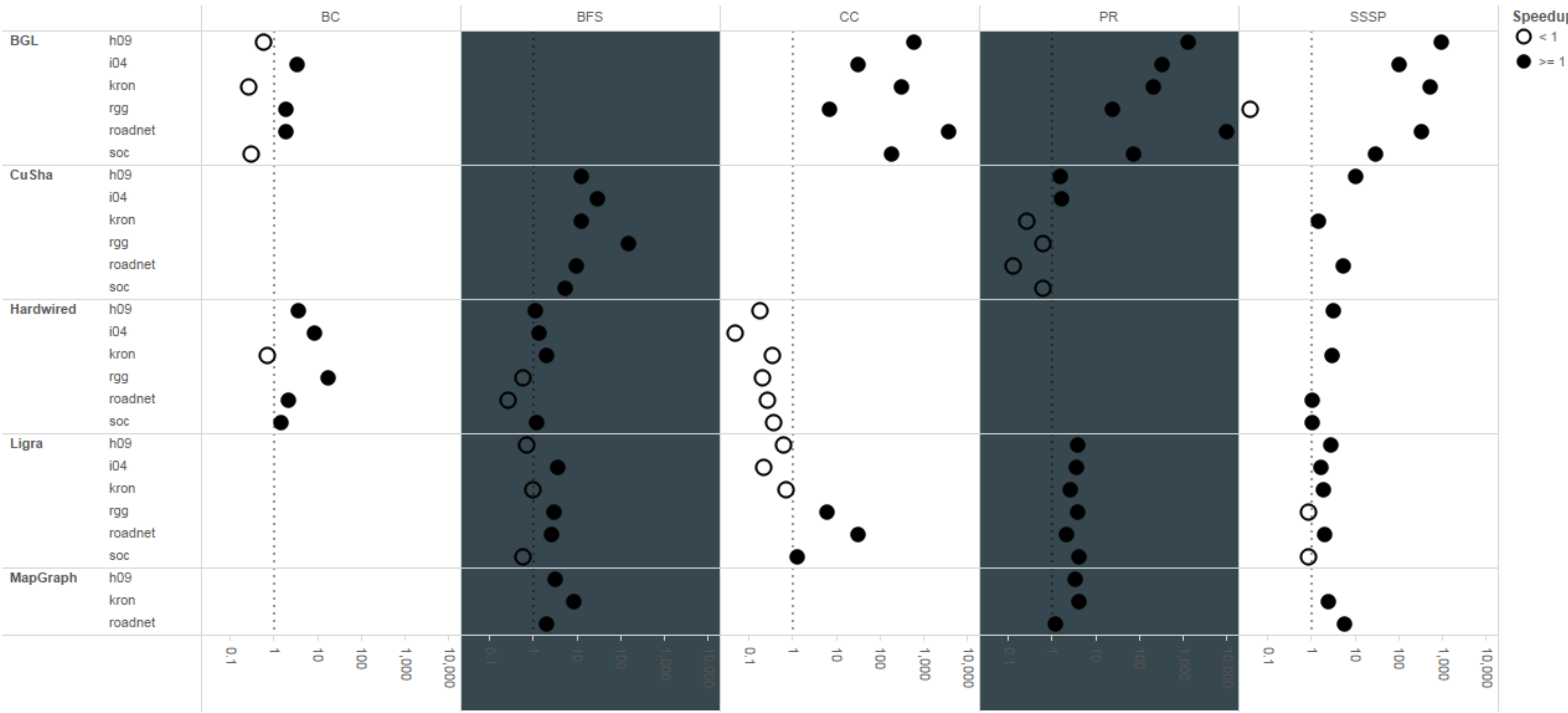
Gunrock vs. nvGRAPH

- Native graph representation
- Custom (but good!) load-balancing
- Open-source
- Write your own primitives
- Which primitives fit into the Gunrock model?

- Matrix-based representation
- Leverages extensive sparse-matrix infrastructure (sparse vector: a challenge!)
- API access only
- Limited set of primitives
- Which primitives fit into the Graph BLAS?

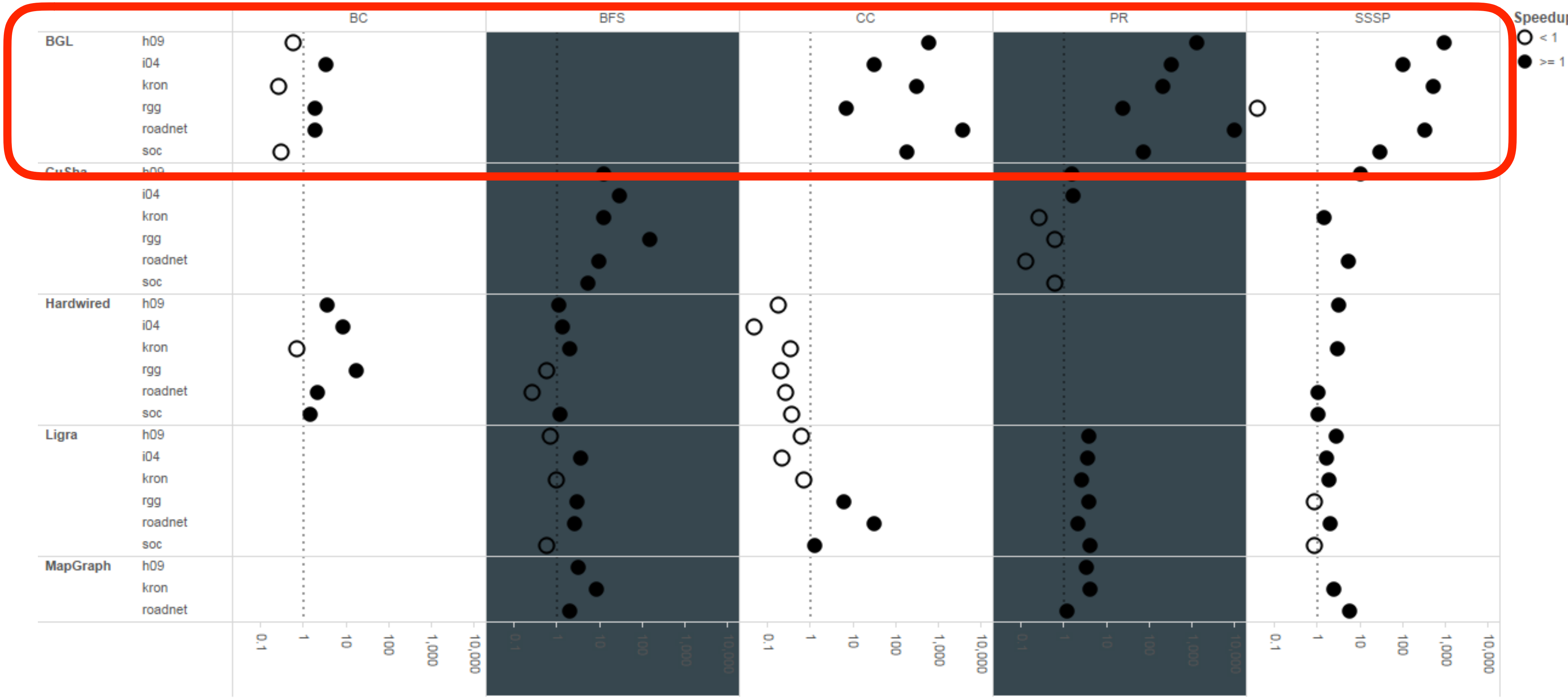
https://developer.nvidia.com/nvgraph

1-GPU Performance Comparison



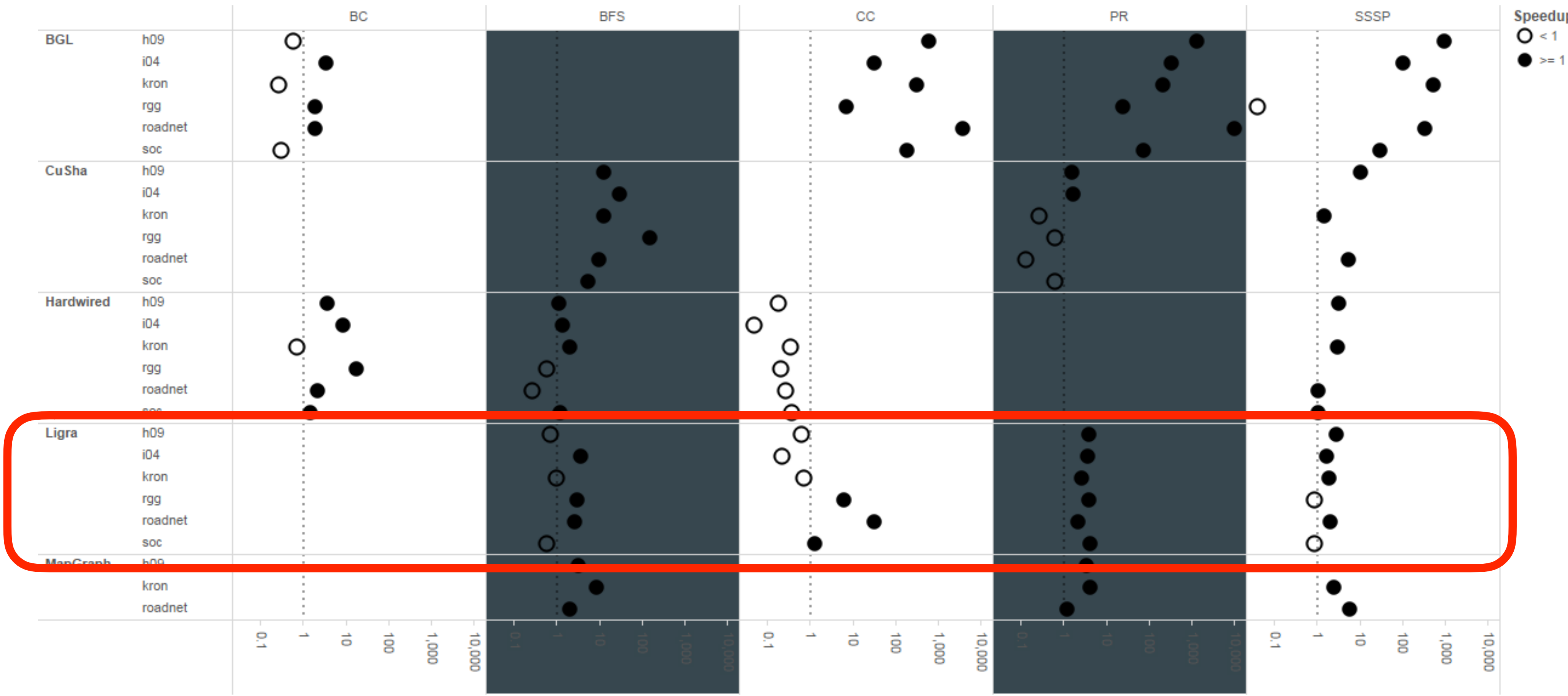
- Each row: single engine on certain dataset, vs. Gunrock
- Black dots/right: Gunrock faster. White dots/left: Gunrock slower

1-GPU Performance Comparison



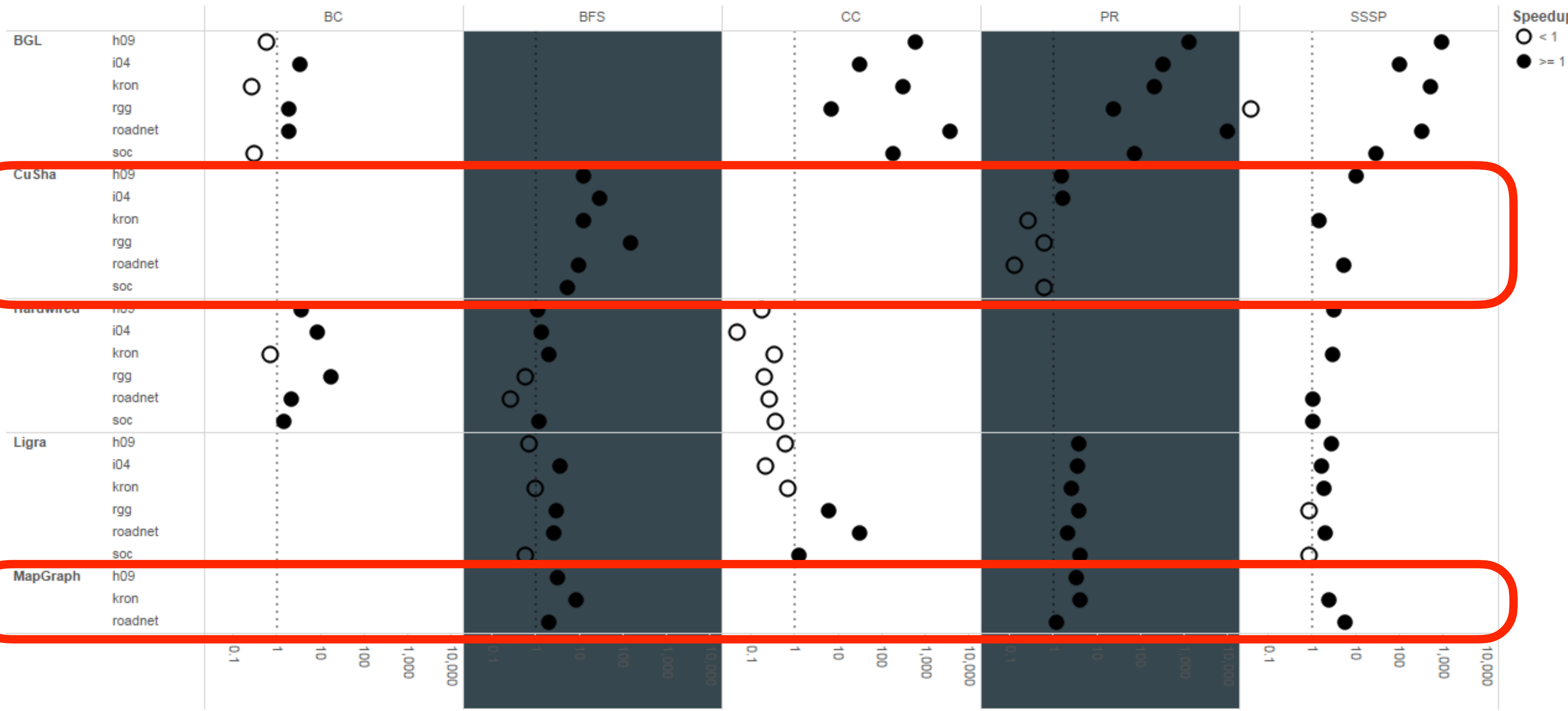
- 10+x faster than single-core CPU (Boost), or PowerGraph

1-GPU Performance Comparison



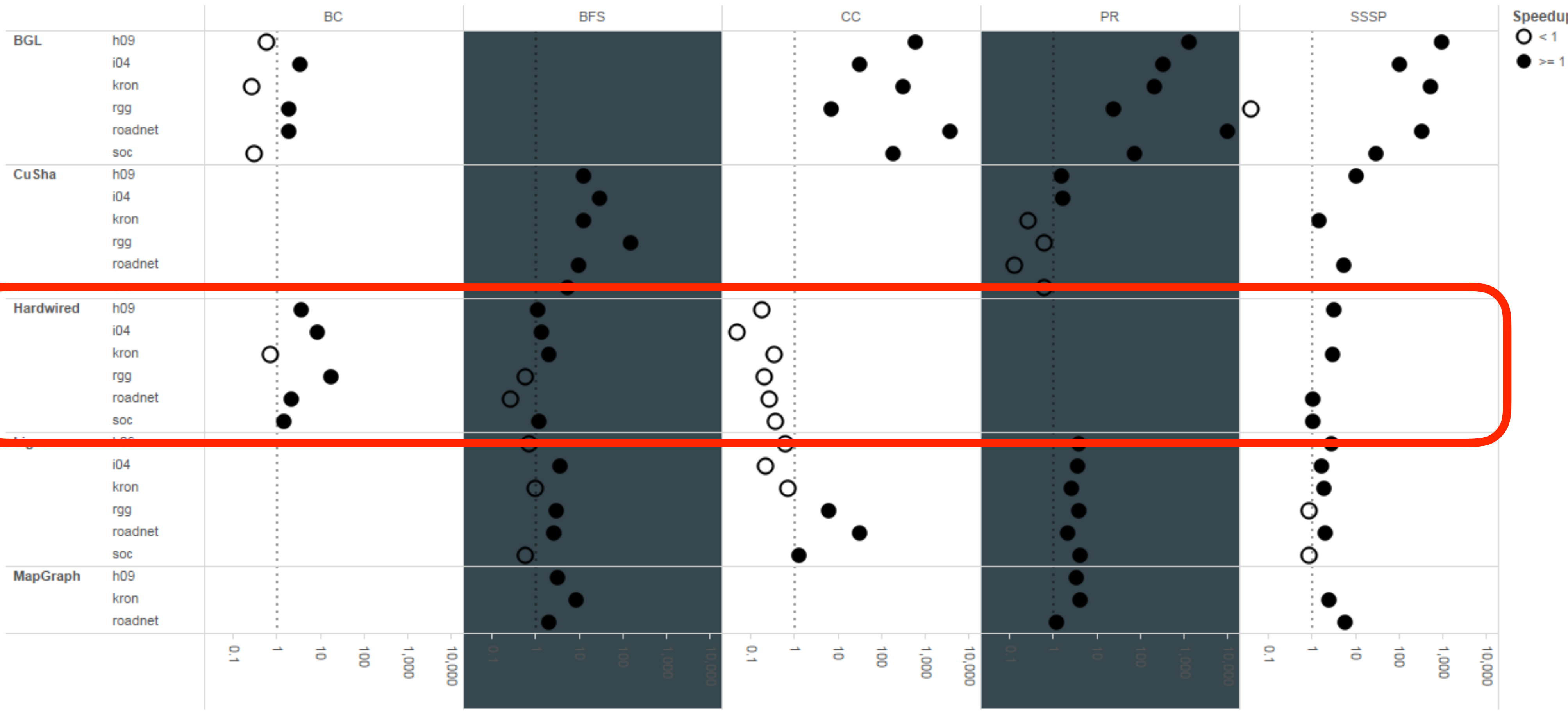
- On par with fastest 2-socket CPU (Ligra) (Gunrock 16 wins, Ligra 8 wins)

1-GPU Performance Comparison



- Fastest of all GPU programmable frameworks (CuSha, MapGraph, Medusa)

1-GPU Performance Comparison



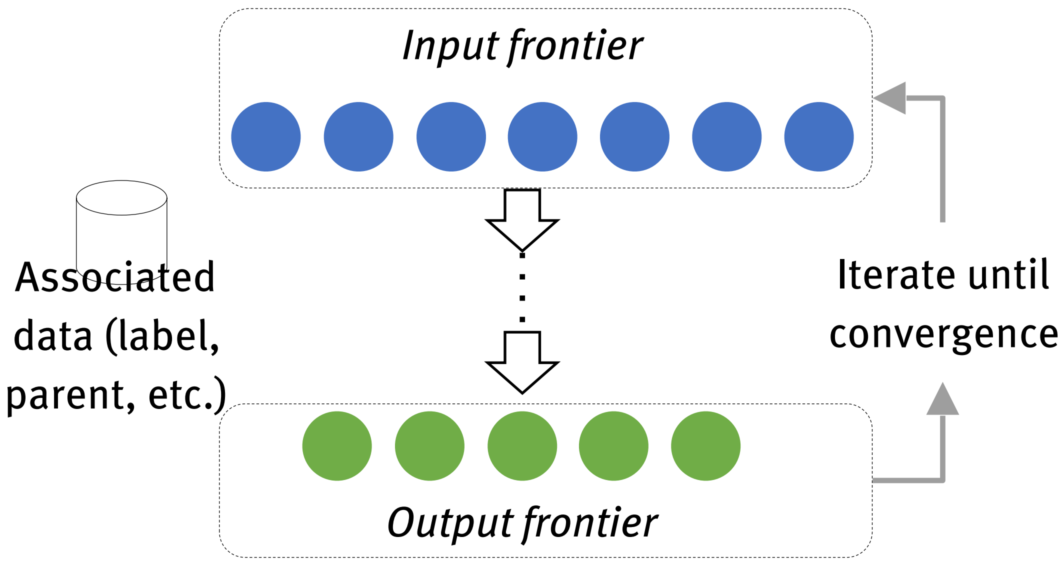
- Competitive with hardwired GPU implementations

Research Directions: Scalability

- Largest memory on a CPU: 5 TB
- On an NVIDIA GPU: 12 GB (GP100: 16 GB)
- Today: Multi-GPU, single node (next!)
- Tomorrow:
  - Out of core?
  - Multi-node?
  - Long term?: heterogeneous single-chip processors

Multi-GPU Framework (for programmers)

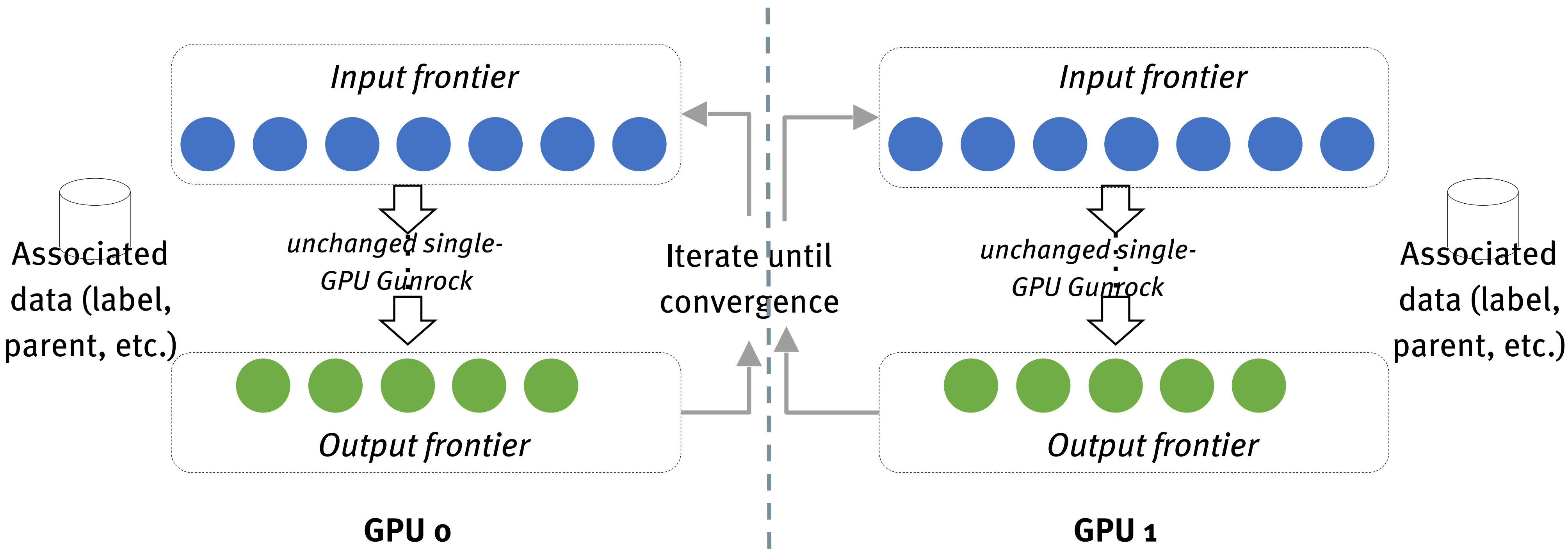
Recap: Gunrock on single GPU



Single GPU

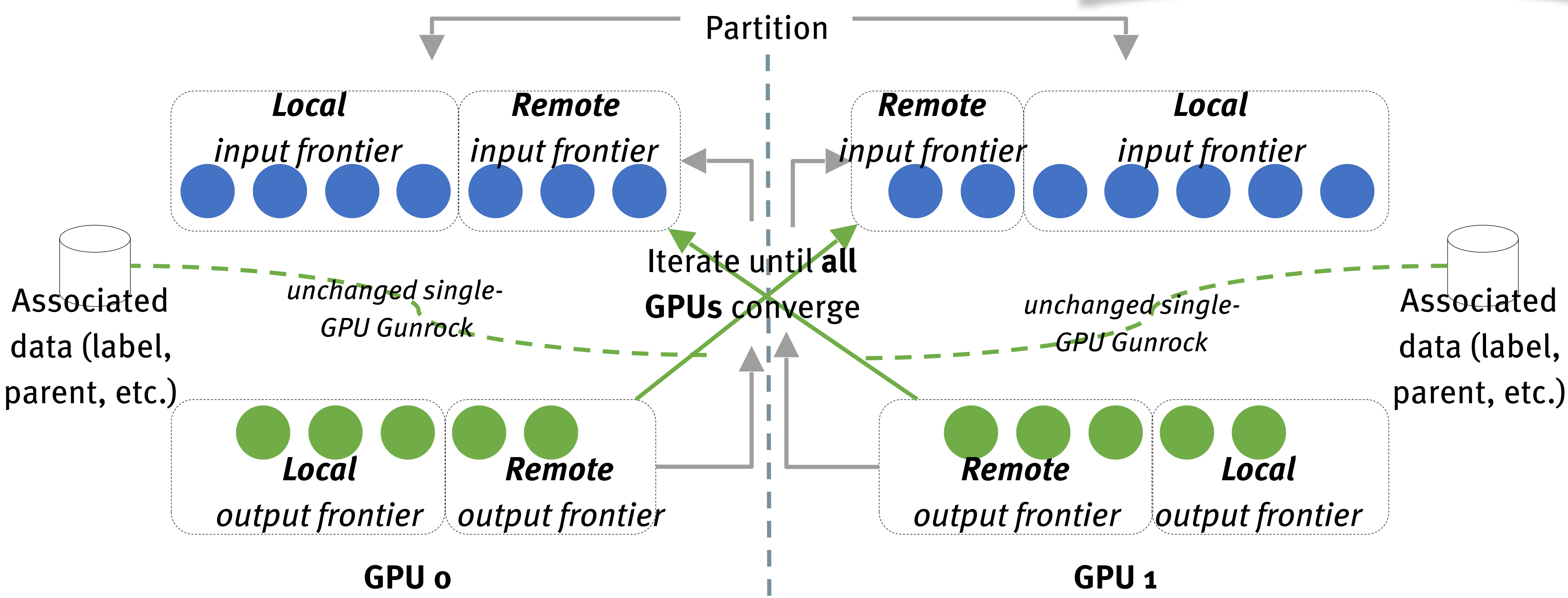
Multi-GPU Framework (for programmers)

Dream: just duplicate the single GPU implementation
Reality: it won't work, but **good try!**



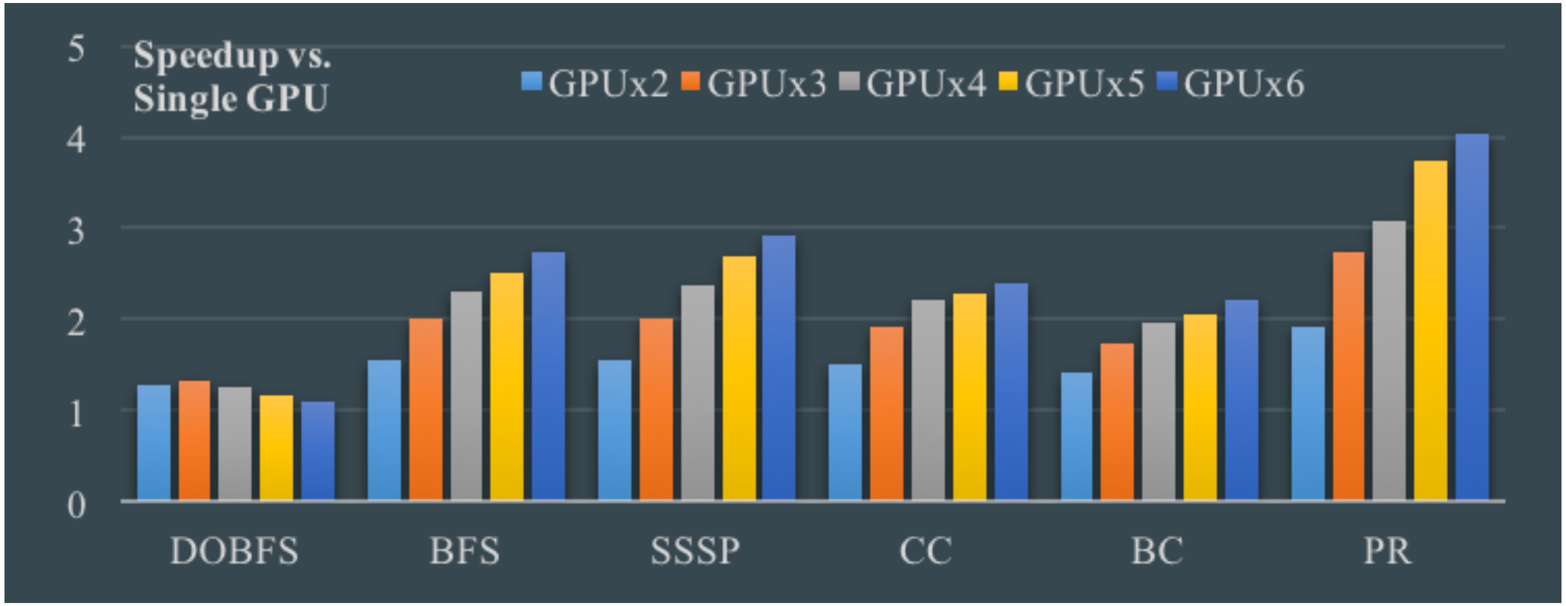
Multi-GPU Framework (for programmers)

Yuechao Pan, Yangzihao Wang, Yuduo Wu, Carl Yang, and John D. Owens. ***Multi-GPU Graph Analytics****. arxiv, abs/1504.04804(1504.04804v2), April 2016.*



Specify (1) how to combine frontiers, (2) what data to communicate, (3) global convergence condition

Results: Multi-GPU Scaling



- Primitives (except DOBFS) get good speedups (averaged over 16 datasets of various types)
  BFS: 2.74x, SSSP: 2.92x, CC: 2.39x, BC: 2.22x, PR: 4.03x using 6 GPUs
- Peak DOBFS performance: 514 GTEPS with rmat_n20_512
- Gunrock can process a graph with 3.6B edges (full-friendster graph, undirected, DOBFS in 339ms, 10.7 GTEPS using 4 K40s); 50 PR iterations on the directed version (2.6B edges) took ~51 seconds

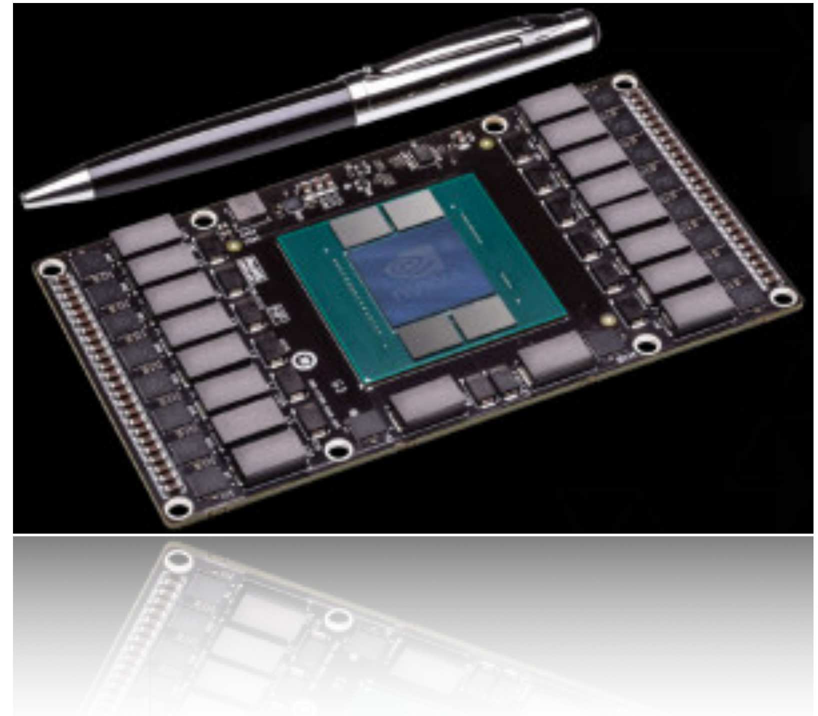
BFS: Multi-GPU Gunrock vs. Others

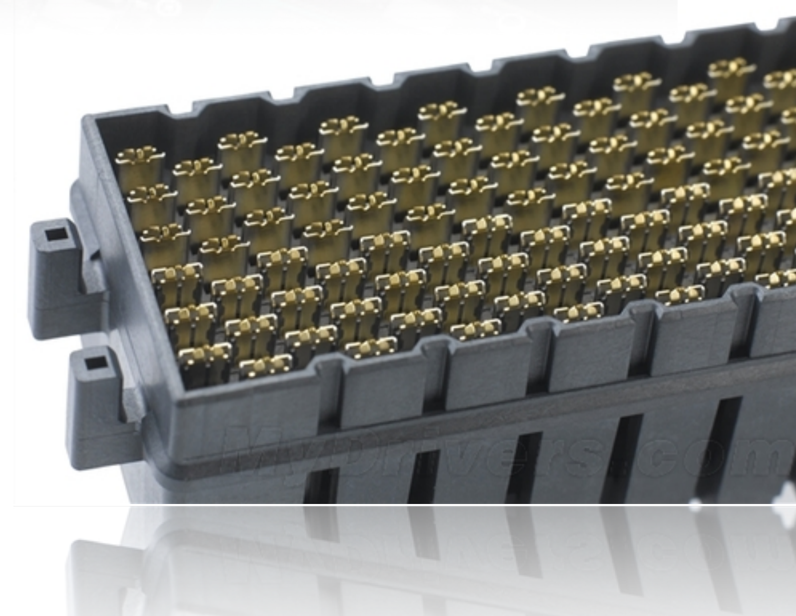
graph	algo	ref.	ref. hw.	ref. perf.	our hw.	our perf.	comp.
com-orkut (3M, 117M, UD)	BFS	Bisson [5]	1×K20X×4	2.67 GTEPS	4×K40	14.22 GTEPS	5.33X
com-Friendster (66M, 1.81B, UD)	BFS	Bisson [5]	1×K20X×64	15.68 GTEPS	4×K40	14.1 GTEPS	0.90X
kron_n23_16 (8M, 256M, UD)	BFS	Bernaschi [4]	1×K20X×4	∼1.3 GTEPS	4×K40	30.8 GTEPS	23.7X
kron_n25_16 (32M, 1.07G, UD)	BFS	Bernaschi [4]	1×K20X×16	∼3.2 GTEPS	6×K40	31.0 GTEPS	9.69X
kron_n25_32 (32M, 1.07G, D)	BFS	Fu [15]	2×K20×32	22.7 GTEPS	4×K40	32.0 GTEPS	1.41X
kron_n23_32 (8M, 256M, D)	BFS	Fu [15]	2×K20×2	6.3 GTEPS	4×K40	27.9 GTEPS	4.43X
kron_n24_32 (16.8M, 1.07G, UD)	BFS	Liu [24]	2×K40×1	15 GTEPS	2×K40	77.7 GTEPS	5.18X
kron_n24_32 (16.8M, 1.07G, UD)	BFS	Liu [24]	4×K40×1	18 GTEPS	4×K40	67.7 GTEPS	3.76X
kron_n24_32 (16.8M, 1.07G, UD)	BFS	Liu [24]	8×K40×1	18.4 GTEPS	4×K80	40.2 GTEPS	2.18X
twitter-mpi (52.6M, 1.96G, D)	BFS	Bebee [3]	1×K40×16	0.2242 sec	3×K40	94.31 ms	2.38X
rmat_n21_64 (2M, 128M, D)	BFS	Merrill [29]	4×C2050×1	8.3 GTEPS	4×K40	23.7 GTEPS	2.86X

- Gunrock generally outperforms other implementations on GPU clusters with 4–64 GPUs on both the real and generated graphs cited in their publications
  - Gunrock's "just-enough" memory allocation: critical!
- 2–5 times faster than Enterprise (Liu and Huang, SC15), a dedicated multi-GPU DOBFS implementation

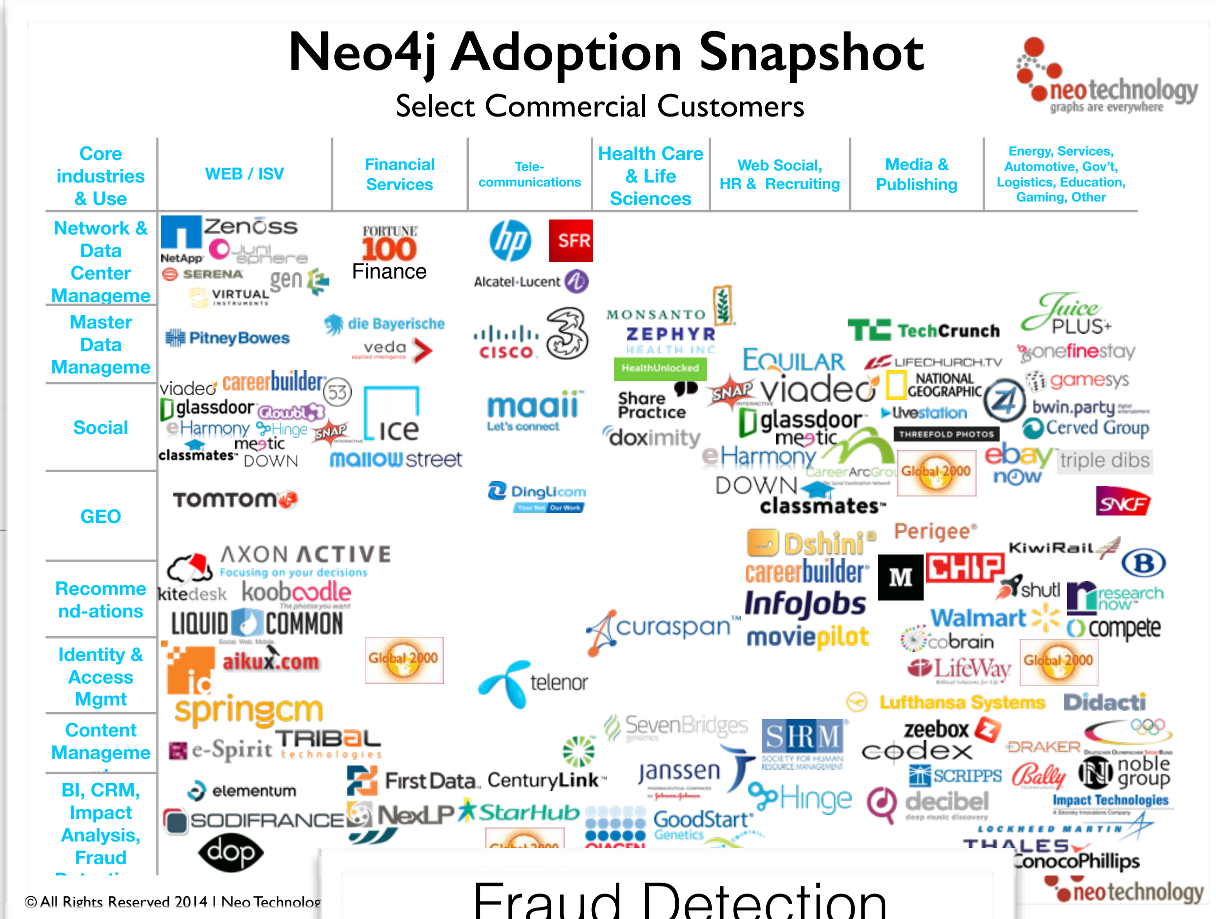
NVIDIA “Pascal” (2016)

- How to scale beyond one node?
  - Scale-out: multiple nodes?
  - Scale-up: out-of-core?
- GP100 has:
  - Stacked memory (720 GB/s)
  - NVLink high-speed CPU-GPU connection (160 GB/s bidirectional)
  - CUDA 8’s unified virtual memory
- On current hardware, we contend mGPU on 1 node is the right building block





Graph Matching



Cypher: Basic Example

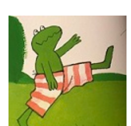
- Declarative query langue with SQL-like clause syntax
- Visual graph patterns
- Tabular results

```
// get node
MATCH (a:Person {id: 0}) RETURN a

// return friends
MATCH (a:Person {id: 0})-->(b) RETURN b

// return friends of friends
MATCH (a:Person {id: 0})--()--(c) RETURN c
```

Fraud Detection

```
MATCH     (accountHolder:AccountHolder)-[]->(contactInformation)
WITH      contactInformation,
          count(accountHolder) AS RingSize
MATCH     (contactInformation)<-[]-(accountHolder),
          (accountHolder)-[r:HAS_CREDITCARD|HAS_UNSECUREDLOAN]->(unsecuredAccount)
WITH      collect(DISTINCT accountHolder.UniqueId) AS AccountHolders,
          contactInformation, RingSize,
          SUM(CASE type(r)
              WHEN 'HAS_CREDITCARD' THEN unsecuredAccount.Limit
              WHEN 'HAS_UNSECUREDLOAN' THEN unsecuredAccount.Balance
              ELSE 0
          END) as FinancialRisk
WHERE     RingSize > 1
RETURN    AccountHolders AS FraudRing,
          labels(contactInformation) AS ContactType,
          RingSize,
          round(FinancialRisk) as FinancialRisk
ORDER BY  FinancialRisk DESC
```

Research Directions: Long Term

- Efficiency
  - Raw peak performance
  - Achieving peak performance with smaller graphs
- More and higher-level algorithms
  - More customers!
- Asynchronous execution
  - Graph coloring 
- Rich data on vertices and edges
- What goes above Gunrock? GraphX, TinkerPop, etc.
- What goes below Gunrock?
  - Beyond CSR
  - Graph BLAS
  - Dynamic graphs

Frog: Asynchronous Graph Processing …
http://grid.hust.edu.cn/xhshi/projects/frog.html

Thanks to …

- Yangzihao Wang, Yuechao Pan, Yuduo Wu, Carl Yang, Leyuan Wang, Mohamed Ebeida, Chenshan Shari Yuan, Weitang Liu (UC Davis)
- Nikolai Sakharnykh, Rob Zuppert, Joe Eaton, Doug Holt, Tom Reed, Ujval Kapasi, Cliff Woolley, Mark Harris, Duane Merrill, Michael Garland, David Luebke, Chandra Cheij (NVIDIA), and the CUDA Fellows program
- Vishal V, Erich Elsen, Guha Jayachandran (Onu)
- DARPA XDATA program & program managers Christopher White and Wade Shen, and Gabriela Araujo
- NSF awards CCF-1017399, OCI-1032859
- UC Lab Fees Research Program Award 12-LR-238449
- Adobe and Grainger Foundation grants
- NVIDIA hardware donations & cluster access

Next steps!

- Feel free to send us questions!
  jowens@ece.ucdavis.edu

- Even better, file Gunrock issues!
  https://github.com/gunrock/gunrock/issues

- Slides from this talk at
  http://preview.tinyurl.com/owens-nv-webinar-160426