GPU GRAPH ANALYTIC PERFORMANCE

Brad Rees, Joe Eaton November 1st, 2018

AGENDA

- Introduction - Brad
- Graph Algorithms - Joe
- Conclusion - Brad

CYBER

Why Graph and Data Science is Important

- First Principle of Cybesecurity
  - Indication of compromise needs to improve as attacks are becoming more sophisticated, subtle, and hidden in the massive volume of data. Combining ***machine learning***, ***graph analytics***, ***and applied statistics*** while integrating these methods with machine learning is essential to reduce false positives, detect threats faster, and empower analysts to be more efficient

- Looking at Insider Threat

HIGH CONSEQUENCE

Insider Threat Cost

$8.76 Million
per incident

Average cost to an organization for an insider-related incident

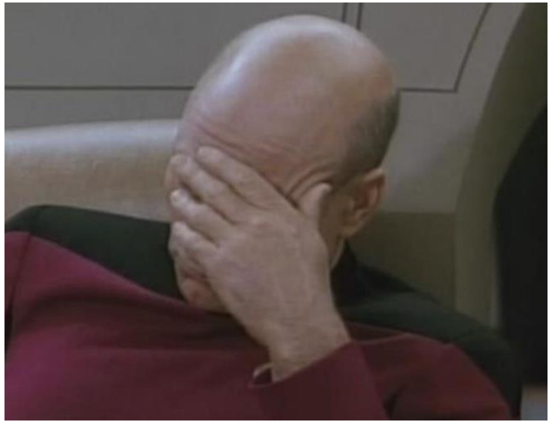
- 2018 Ponemon Institute

https://153j3ttjub71nfe89mc7r5gb-wpengine.netdna-ssl.com/wp-content/uploads/2018/04/ObserveIT-Insider-Threat-Global-Report-FINAL.pdf

HIGH CONSEQUENCE

Insider Threat Lag

191 Days to Detect



Average number of days to detect a data breach

https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=SEL03130WWEN&

MOVING BEYOND CURRENT APPROACHES

Most Popular Method

- Current methods depends on prebuilt signatures run against log files
  - Does not scale
  - Does not handle unseen methodologies

- Alternative Approach – use Graph Analytics

GRAPH ANALYTIC

One Possible Solution

1. Build a User-to-User Activity Graph
   - Property graph with temporal information

2. Compute user behavior changes over time
   - PageRank – changes in user’s importance
   - Jaccard Similarity – changes in relationship to others
   - Louvain – changes in social group
   - Triangle Counting – changes in group density

3. Look for anomalies

WHAT IS NEEDED

- Fast Graph Processing
- Use GPUs (Shameless Marketing)

Can GPUs be used for Graphs?

32GB VOLTA

The First Step

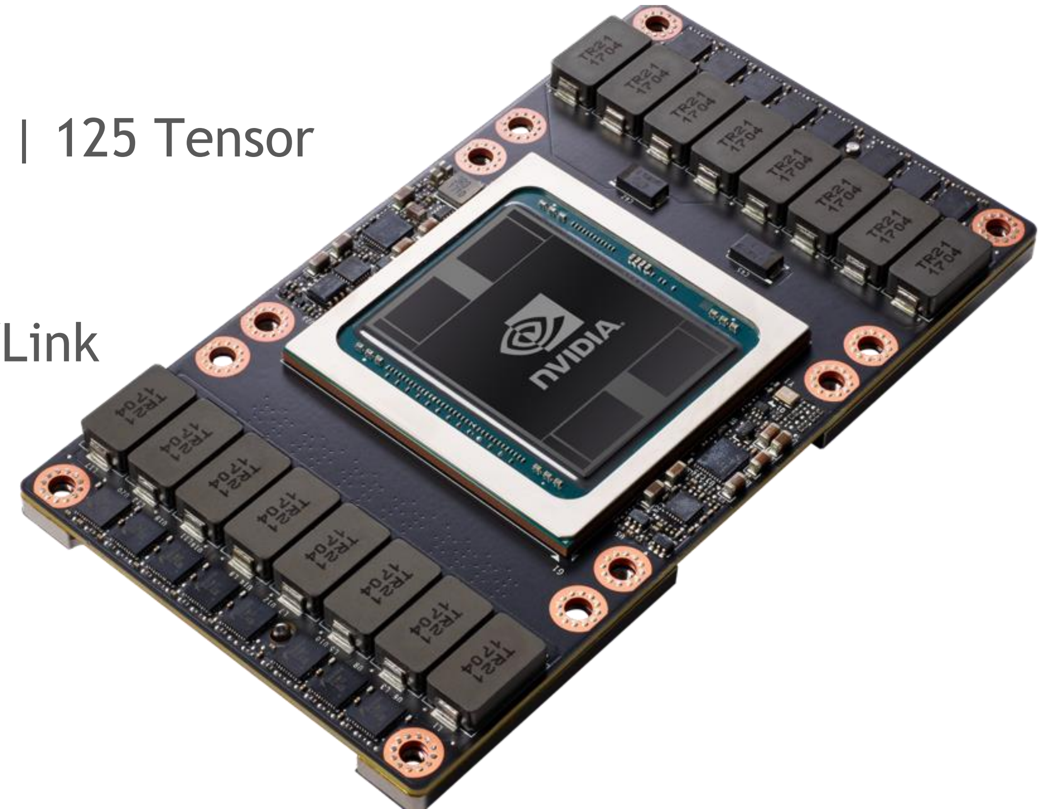
5,120 CUDA cores

640 NEW Tensor cores

7.8 FP64 TFLOPS | 15.7 FP32 TFLOPS | 125 Tensor TFLOPS

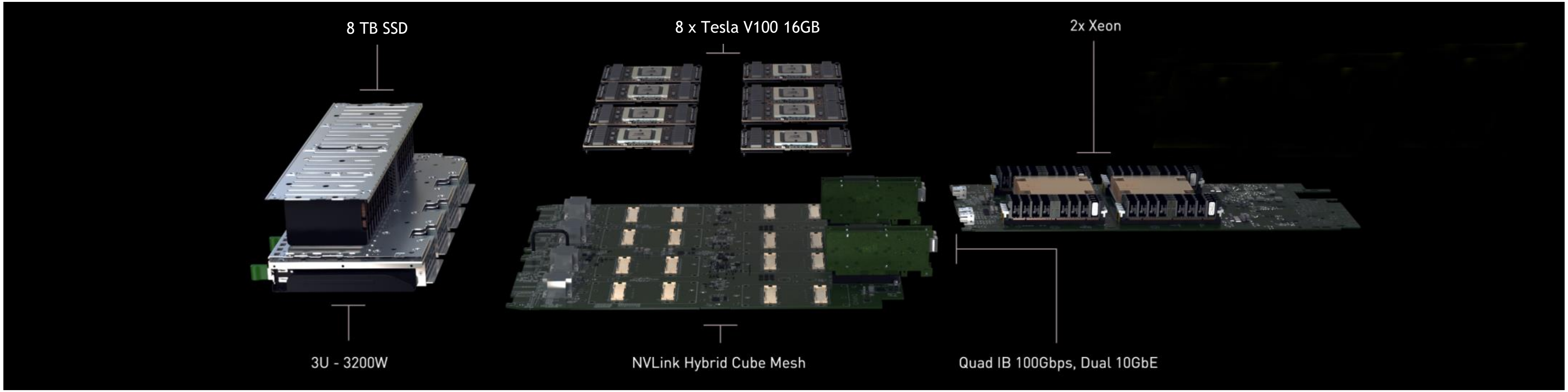
20MB SM RF | 16MB Cache

<u>32GB HBM2</u> @ 900GB/s | 300GB/s NVLink



32GB V100 DGX-1

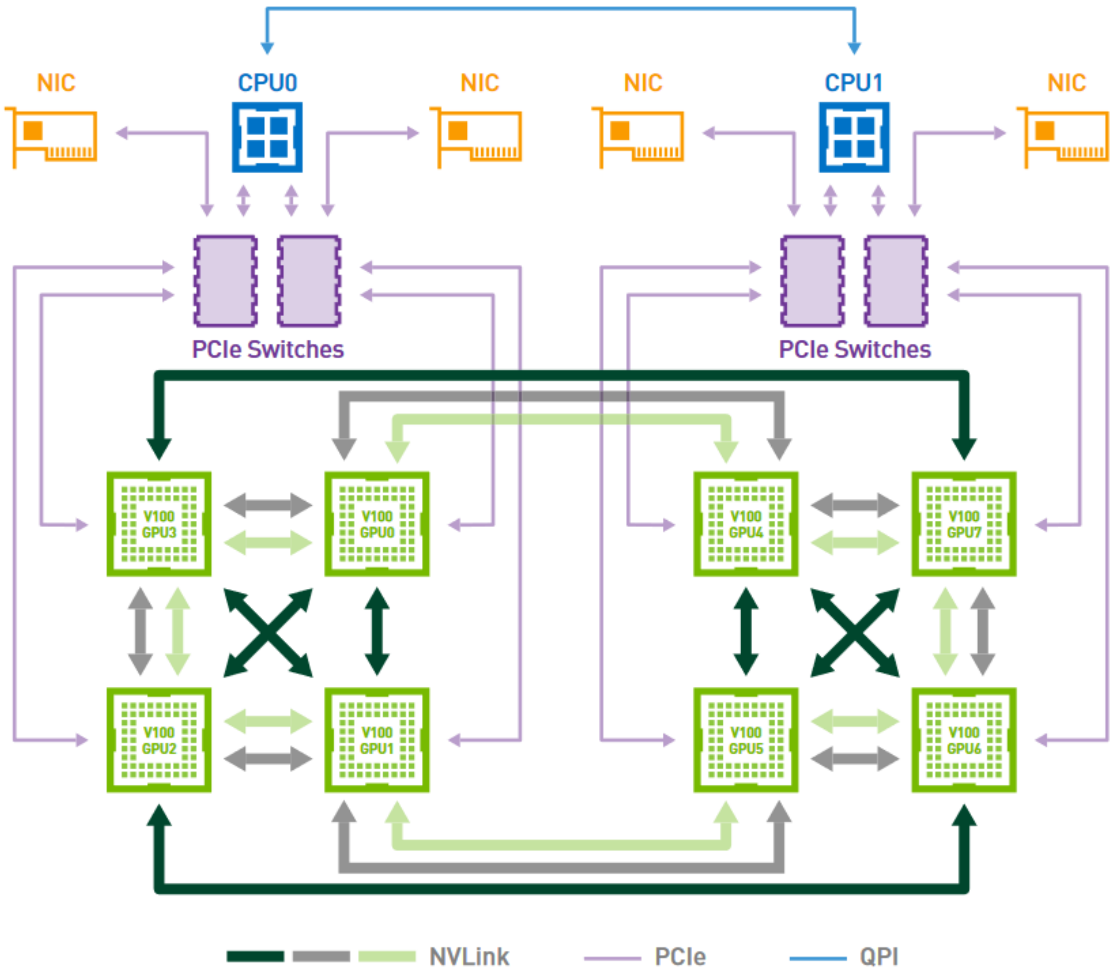
Now with 256GB of GPU Memory



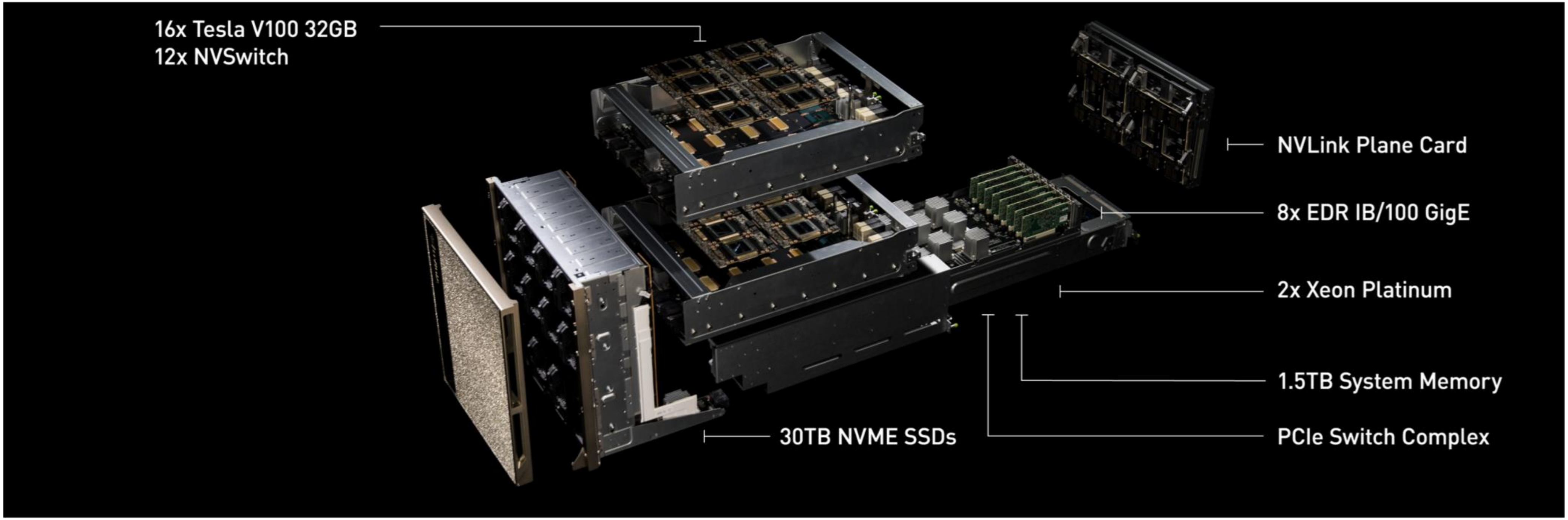
1 PFLOPS | 8x Tesla V100 32GB | 300 GB/s NVLink Hybrid Cube Mesh

2x Xeon | 8 TB RAID 0 | Quad IB 100Gbps, Dual 10GbE | 3U — 3200W

DGX-1 HYBRID-CUBE MESH

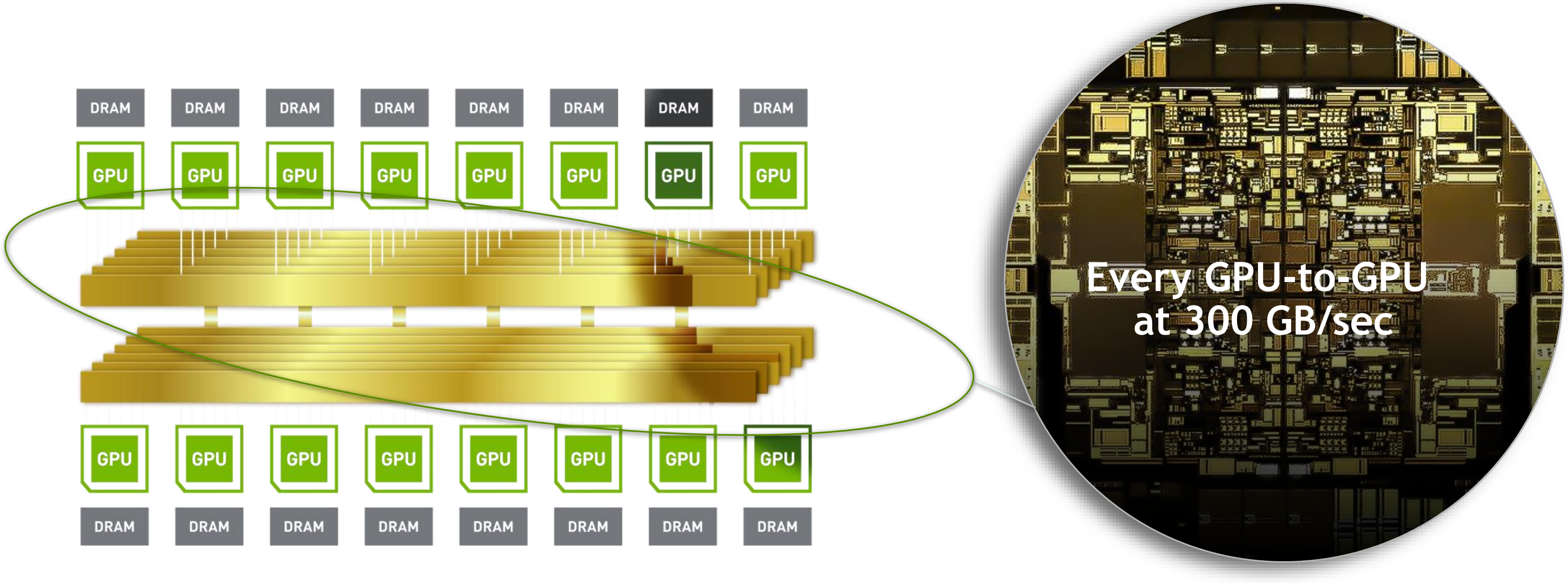


DGX-2



2 PFLOPS | 512GB HBM2 | 16 TB/sec Memory Bandwidth | 10 kW / 160 kg

DGX-2 INTERCONNECT

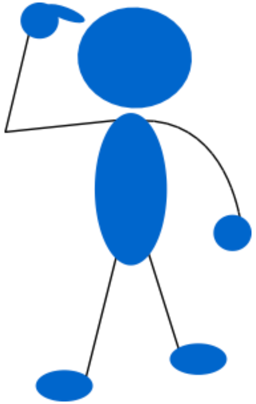


16 Tesla V100 32GB Connected by NVSwitch | On-chip Memory Fabric Semantic Extended Across All GPUs

THE QUESTION IS ..

We now have the computer environment

How well will this work for Graph Analytics?



GRAPH ANALYTIC FRAMEWORKS

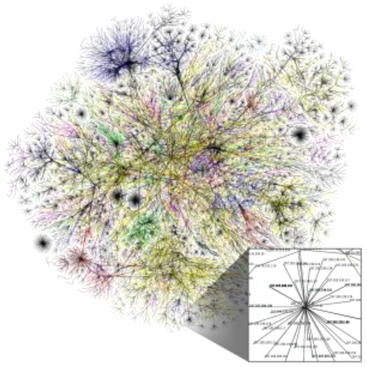
For GPU Benchmarks

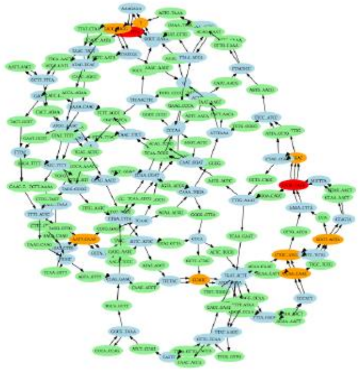
- Gunrock from UC Davis
- Hornet from Georgia Tech (also HornetsNest)
- nvGraph from NVIDIA

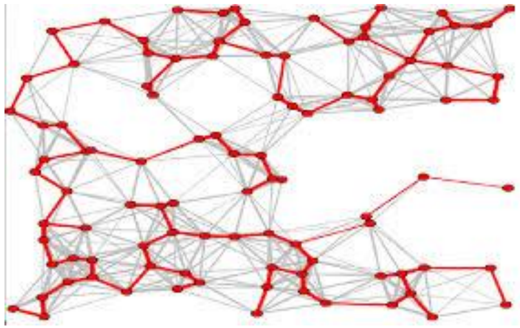
NVGRAPH IN RAPIDS

Easy Onramp to GPU Accelerated Graph Analytics

- GPU Optimized Algorithms
- Reduced cost & Increased performance
- Integration with RAPIDS data preparation and ML methods
- Performance Constantly Improving





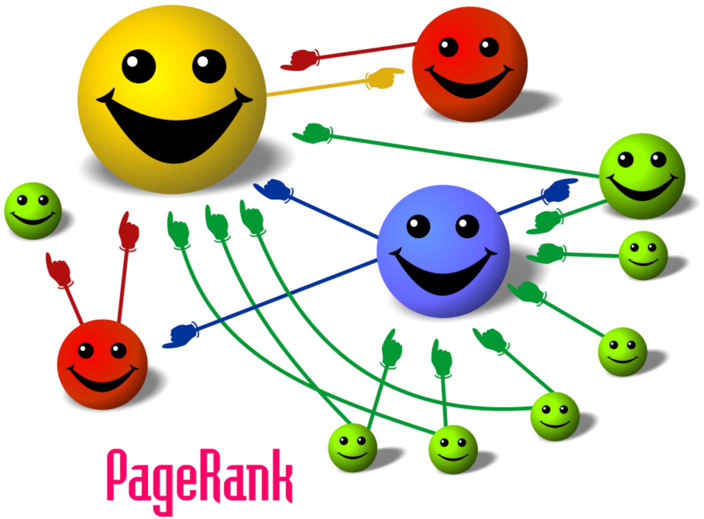


PAGERANK

- Ideal application: influence in social networks
- Each iteration involves computing:

$$y = A\,x$$
$$x = y/norm(y)$$

- Merge-path load balancing for graphs
- Power iteration for largest eigenpair by default
- Implicit Google matrix to preserve sparsity
- Advanced eigensolvers for ill-conditioning



PAGERANK PIPELINE BENCHMARK

Graph Analytics Benchmark

Proposed by MIT LL.

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Apply supercomputing benchmarking methods to create scalable benchmark for big data workloads.

Four different phases that focus on data ingest and analytic processing, details on next slide….

Reference code for serial implementations available on GitHub.
https://github.com/NVIDIA/PRBench

PAGERANK PIPELINE BENCHMARK

4 Stage Graph Analytics Benchmark

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Stage 1 – Generate graph (not timed)

Stage 2 – Read graph from disk, sort edges, write back to disk

Stage 3 – Read sorted edge list, generate normalized adjacency matrix for graph

Stage 4 – Run 20 iterations of Pagerank algorithm (power method)

Stage 2 tests I/O

Stage 3 tests I/O + compute

Stage 4 tests compute

TRIANGLE COUNTING

High Performance Exact Triangle Counting on GPUs
Mauro Bisson and Massimiliano Fatica

Useful for:

- **Community Strength**
- **Graph statistics for summary**
- **Graph categorization/labeling**

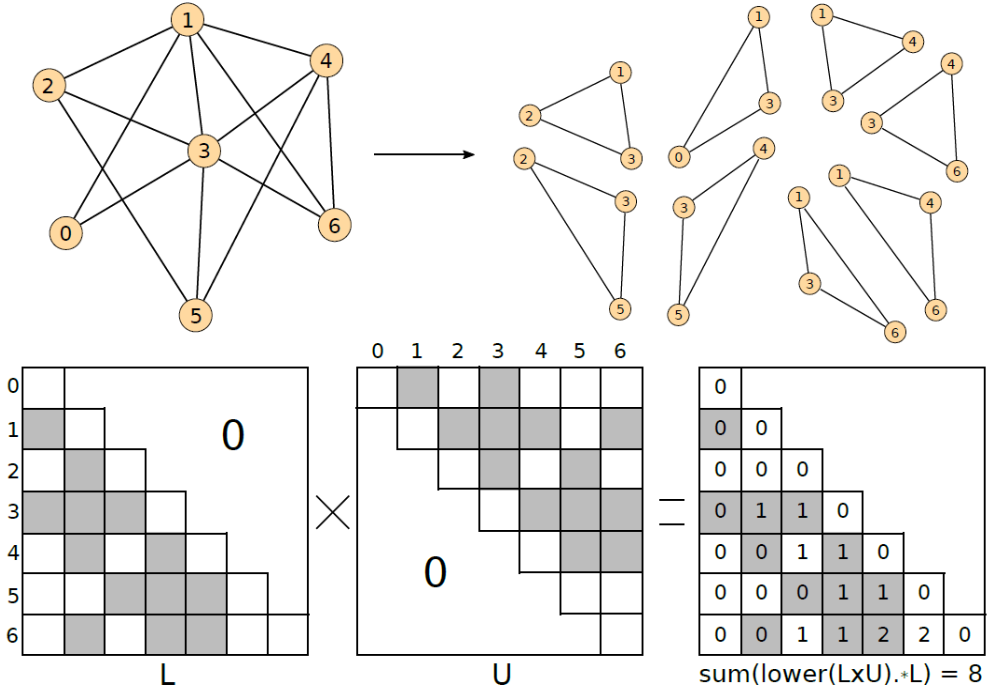


Figure 1. Example of triangle counting via multiplication of the two halves of an adjacency matrix. The sum is restricted to only the grey elements of the original L matrix.

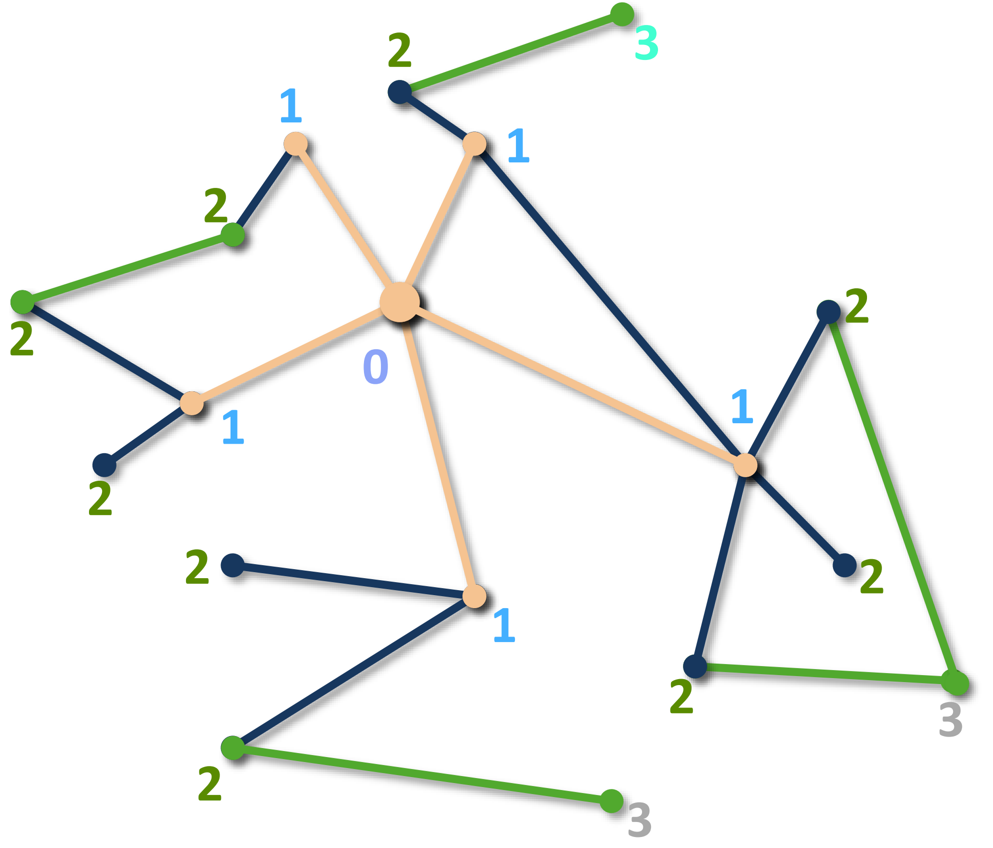
TRAVERSAL/BFS

Common Usage Examples:

Path-finding algorithms:
- Navigation
- Modeling
- Communications Network

Breadth first search
building block
fundamental graph primitive

Graph 500 Benchmark



BFS PRIMITIVE

Load balancing

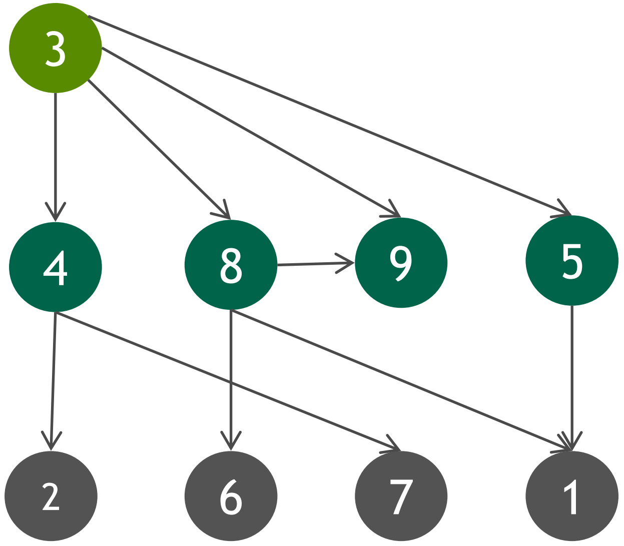
Frontier : 4 8 9 5

Corresponding vertices degree :

2	3	0	1

Exclusive sum :

0	2	5	5	6



k = max (k' such as exclusivesum[k] <= thread_idx)

For this thread :

source = frontier[k]

Edge_index = row_ptr[source] + thread_idx – exclusivesum[k]

Binary search

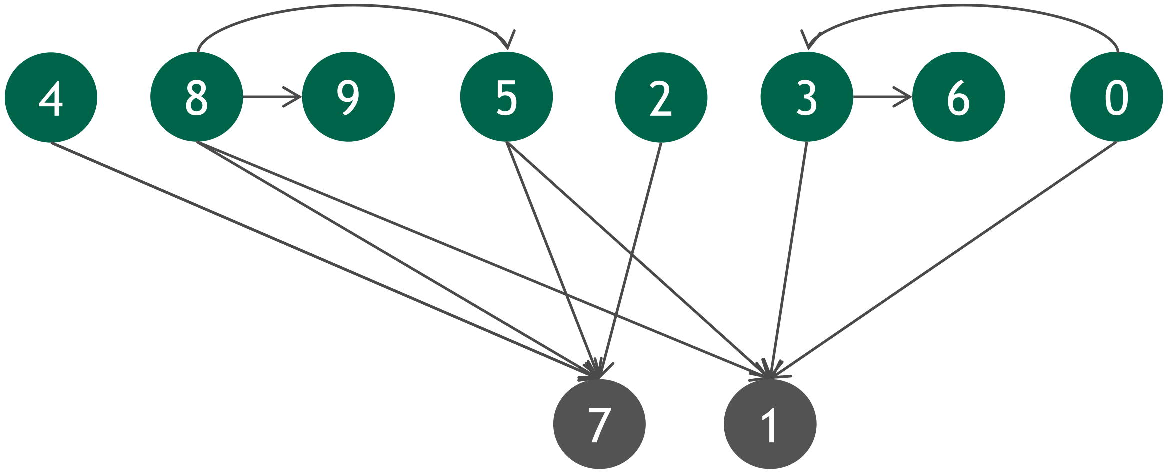
BOTTOM UP

Motivation

- Sometimes it's better for children to look for parents (bottom-up)

Frontier depth=3

Frontier depth=4



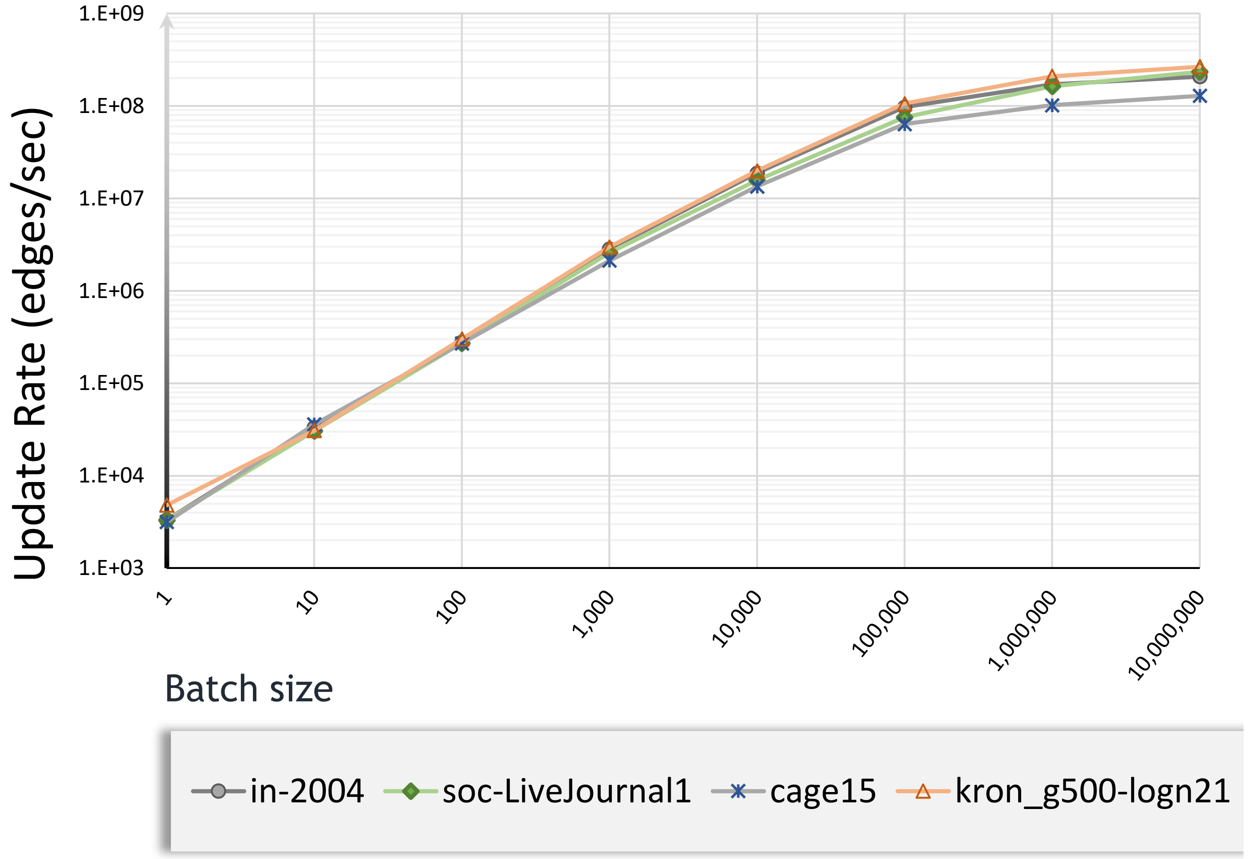
HORNET

- Designed for sparse and irregular data – great for powerlaw datasets
- Essentially a multi-tier memory manager
  - Works with different block sizes -- always powers of two (ensures good memory utilization)
  - Supports memory reclamation
  - Superfast!
- Hornet in RAPIDS: Will be part of cuGraph.
  - Streaming data analytics and GraphBLAS good use cases.
  - Data base operations such as join size estimation.
  - String dictionary lookups, fast text indexing.

HORNET

Performance – Edge Insertion

- Results on the NVIDIA P100 GPU
- Supports over 150M updates per second
    - Checking for duplicates
    - Data movement (when newer block needed)
    - Memory reclamation
- Similar results for deletions

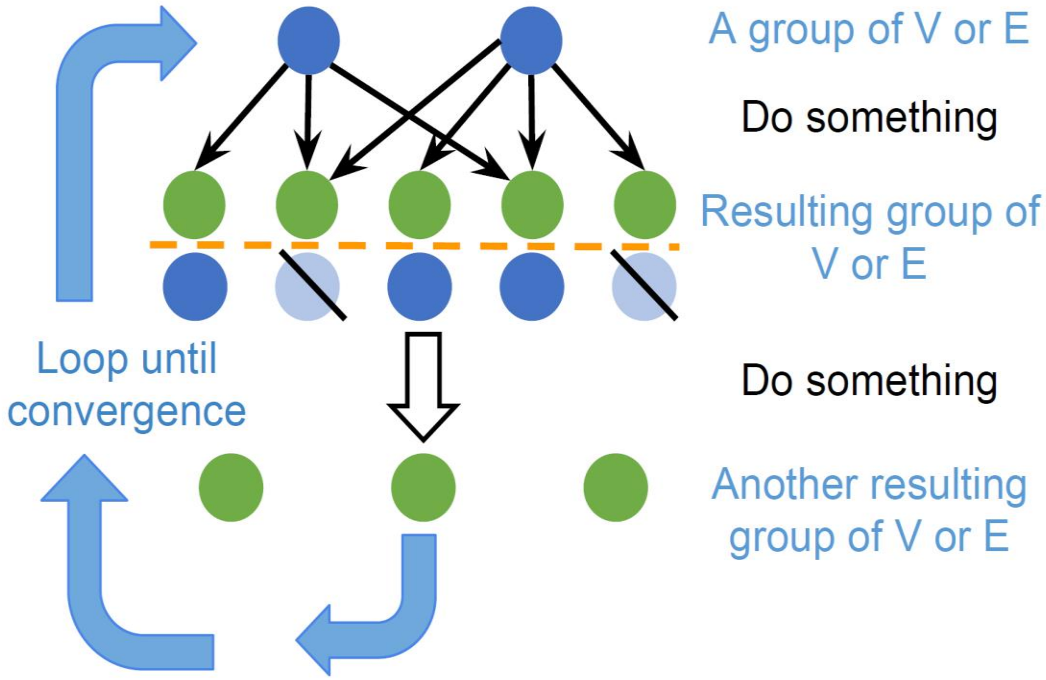


https://gunrock.github.io

- Generality
  - Supports many algorithms
- Programmability
  - Easy to add new methods
- Scalability
  - Multi-GPU support
- Performance
  - Competitive with other GPU frameworks

Programming Model

A generic graph algorithm:



Data-centric abstraction

- Operations are defined on
  a group of vertices or edges $\stackrel{\text{def}}{=}$ a frontier

=> Operations = manipulations of frontiers

Bulk-synchronous programming

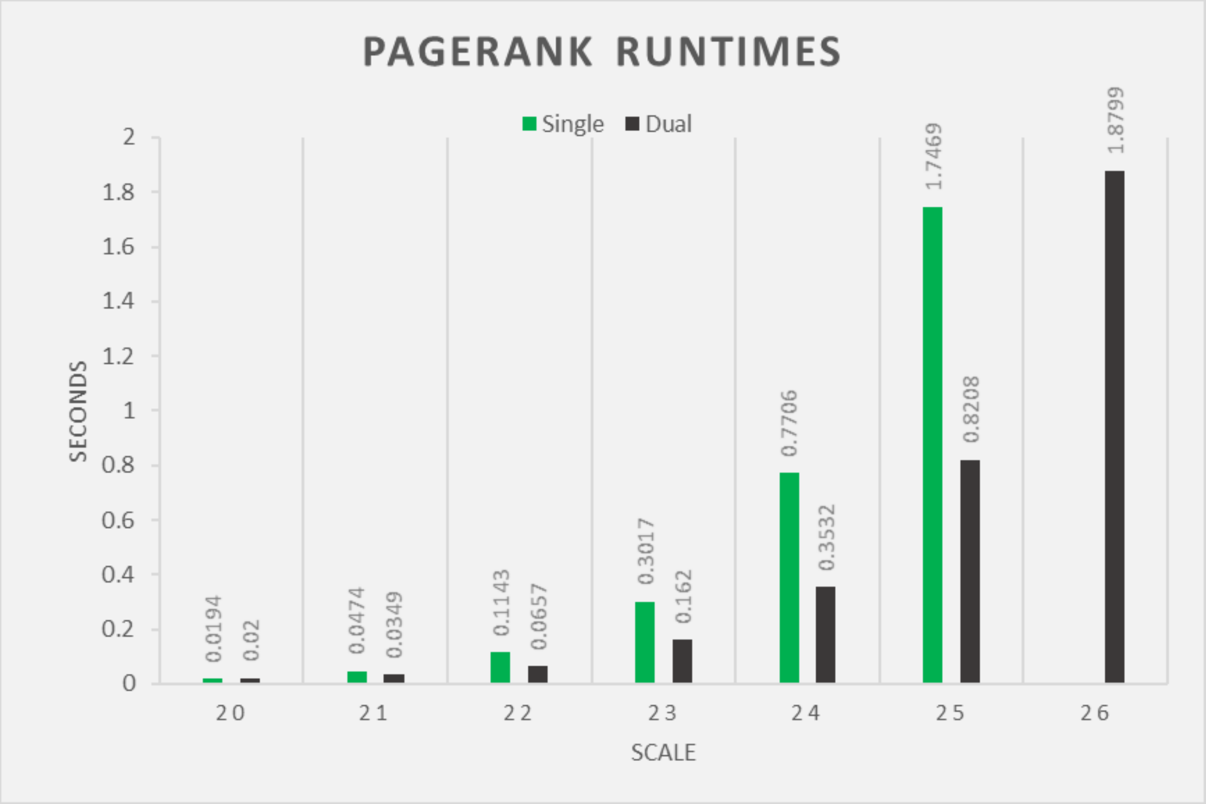
- Operations are done one by one, in order
- Within a single operation, computing on multiple elements can be done in parallel, without order

Yangzihao Wang, Yuechao Pan, Andrew Davidson, Yuduo Wu, Carl Yang, Leyuan Wang, Muhammad Osama, Chenshan Yuan, Weitang Liu, Andy T. Riffel, and John D. Owens.

32GB V100

Single and Dual GPU on Commodity Workstation

RMAT	Nodes	Edges	Single	Dual
20	1,048,576	16,777,216	0.019	0.020
21	2,097,152	33,554,432	0.047	0.035
22	4,194,304	67,108,864	0.114	0.066
23	8,388,608	134,217,728	0.302	0.162
24	16,777,216	268,435,456	0.771	0.353
25	33,554,432	536,870,912	1.747	0.821
26	67,108,864	1,073,741,824		1.880



Scale 26 on a single GPU can be achieved by using Unified Virtual Memory. Runtime was 3.945 seconds
Larger sizes exceed host memory of 64GB

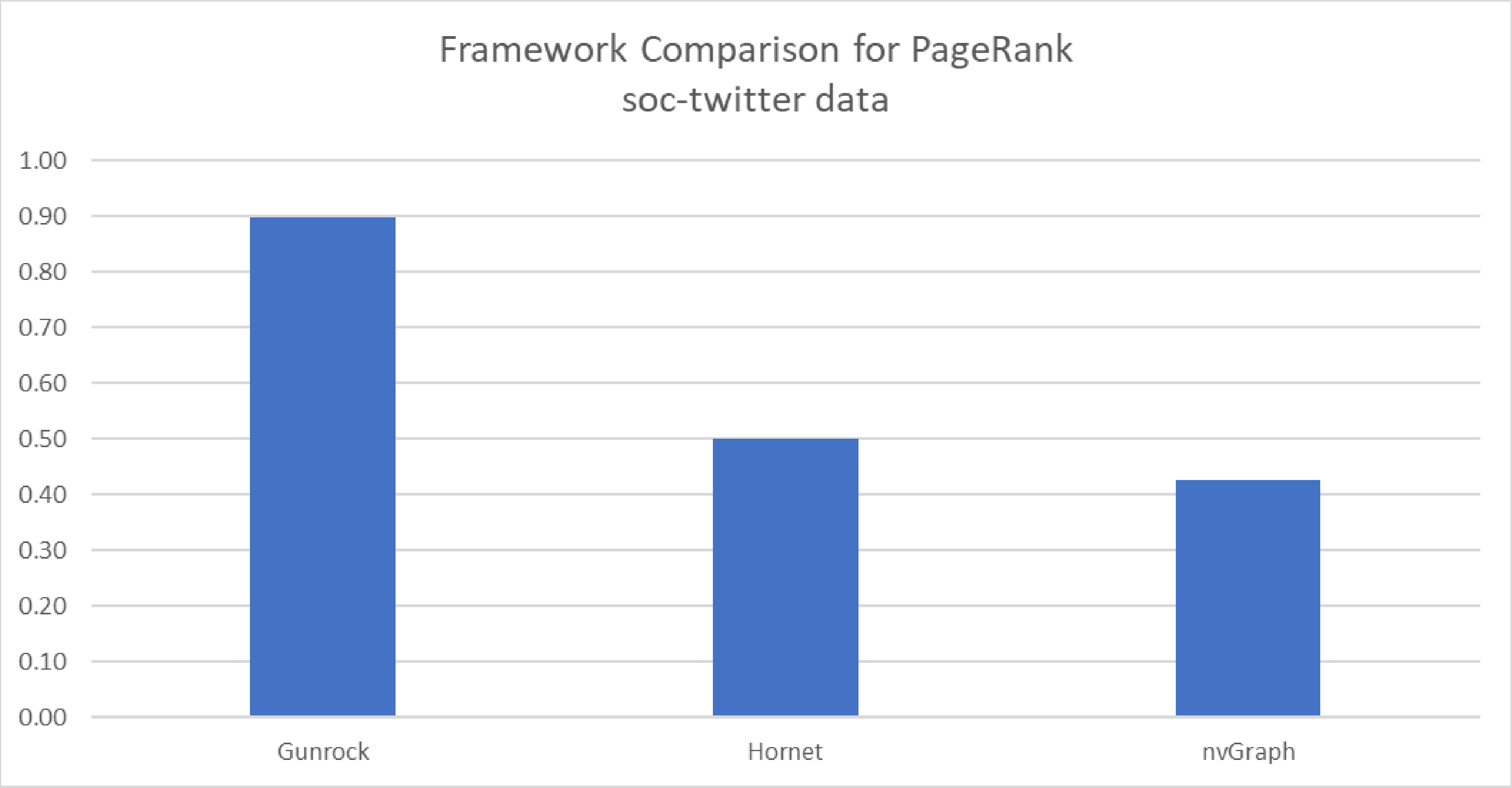
DATASETS

Mix of social network and RMAT

Dataset	Nodes	Edges
soc-twitter-2010	21,297,772	530,051,618
Twitter.mtx	41,652,230	1,468,365,182
RMAT – Scale 26	67,108,864	1,073,741,824
RMAT – Scale 27	134,217,728	2,122,307,214
RMAT - Scale 28	268,435,456	4,294,967,296

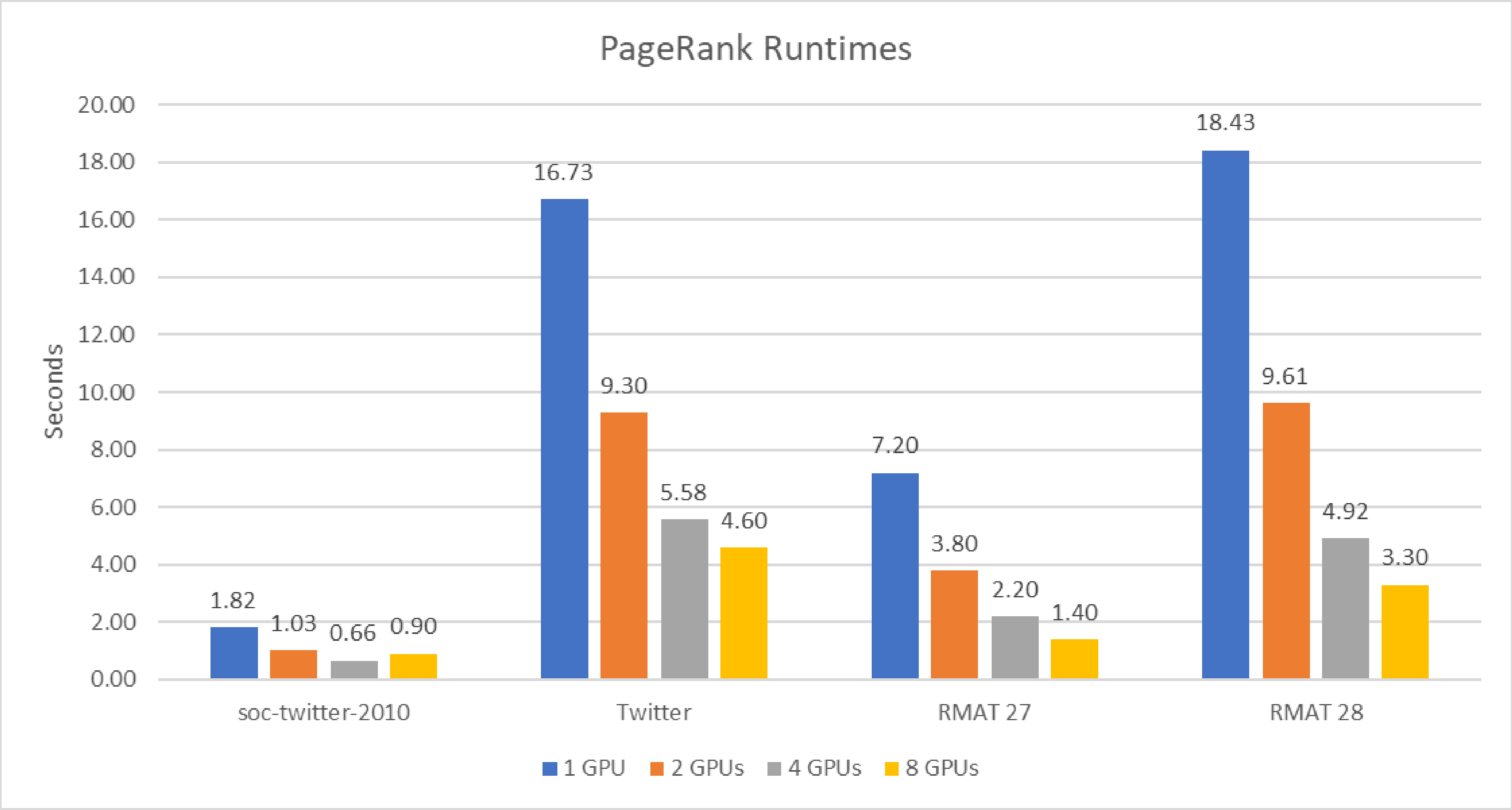
FRAMEWORK COMPARISON

PageRank on DGX-1, Single GPU



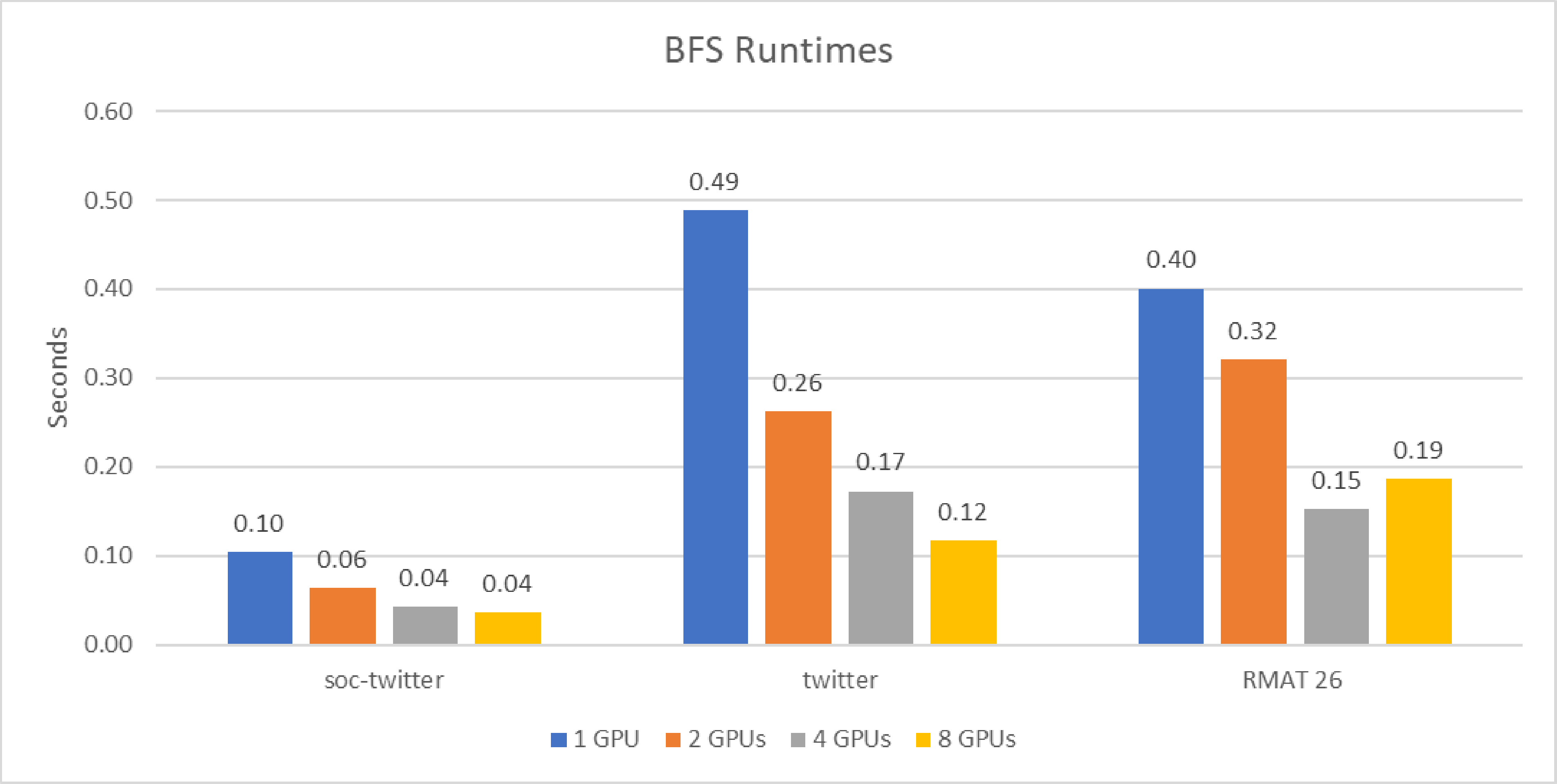
PAGERANK ON DGX-1

Using Gunrock, Multi-GPU

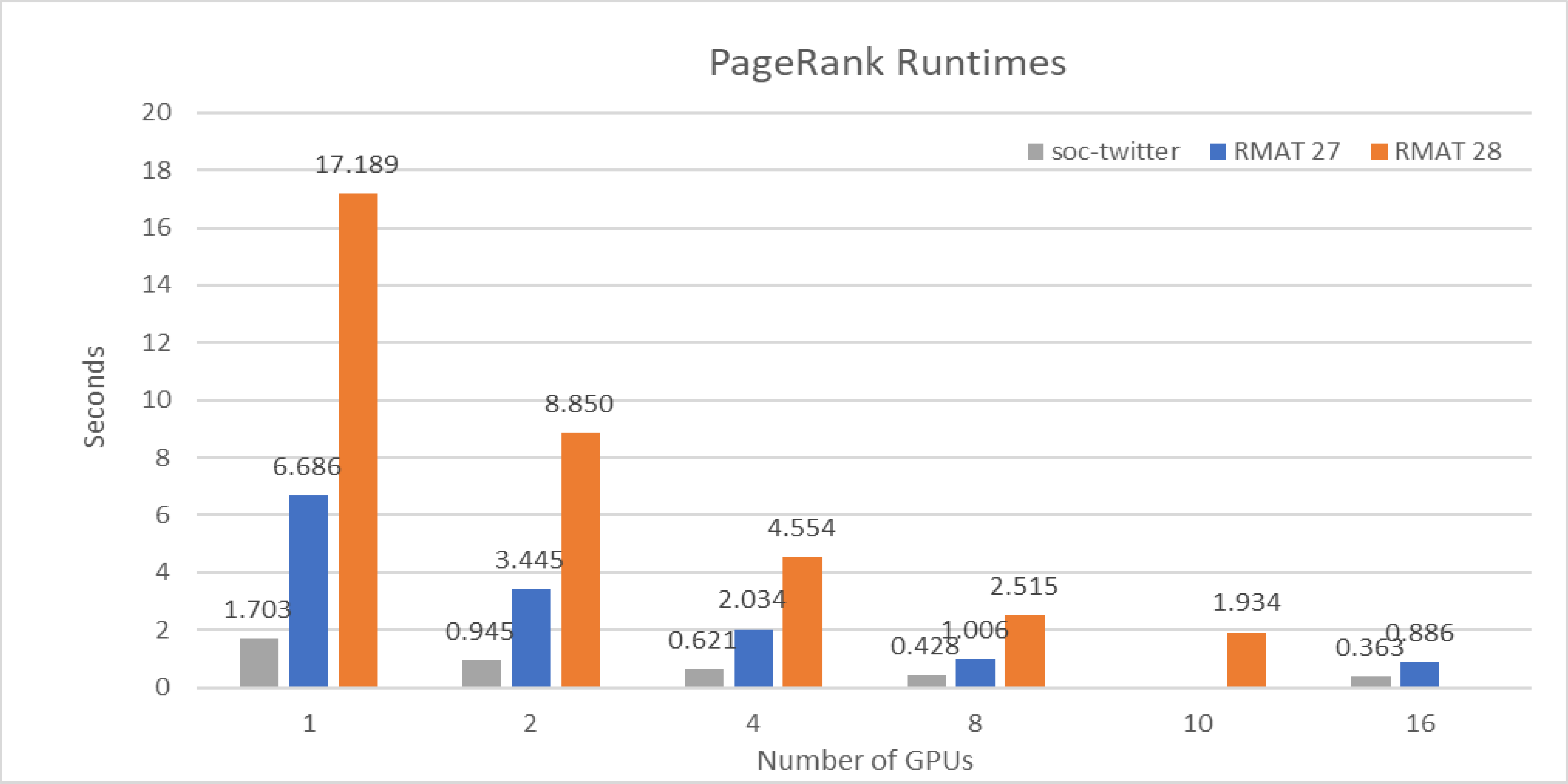


BFS ON DGX-1

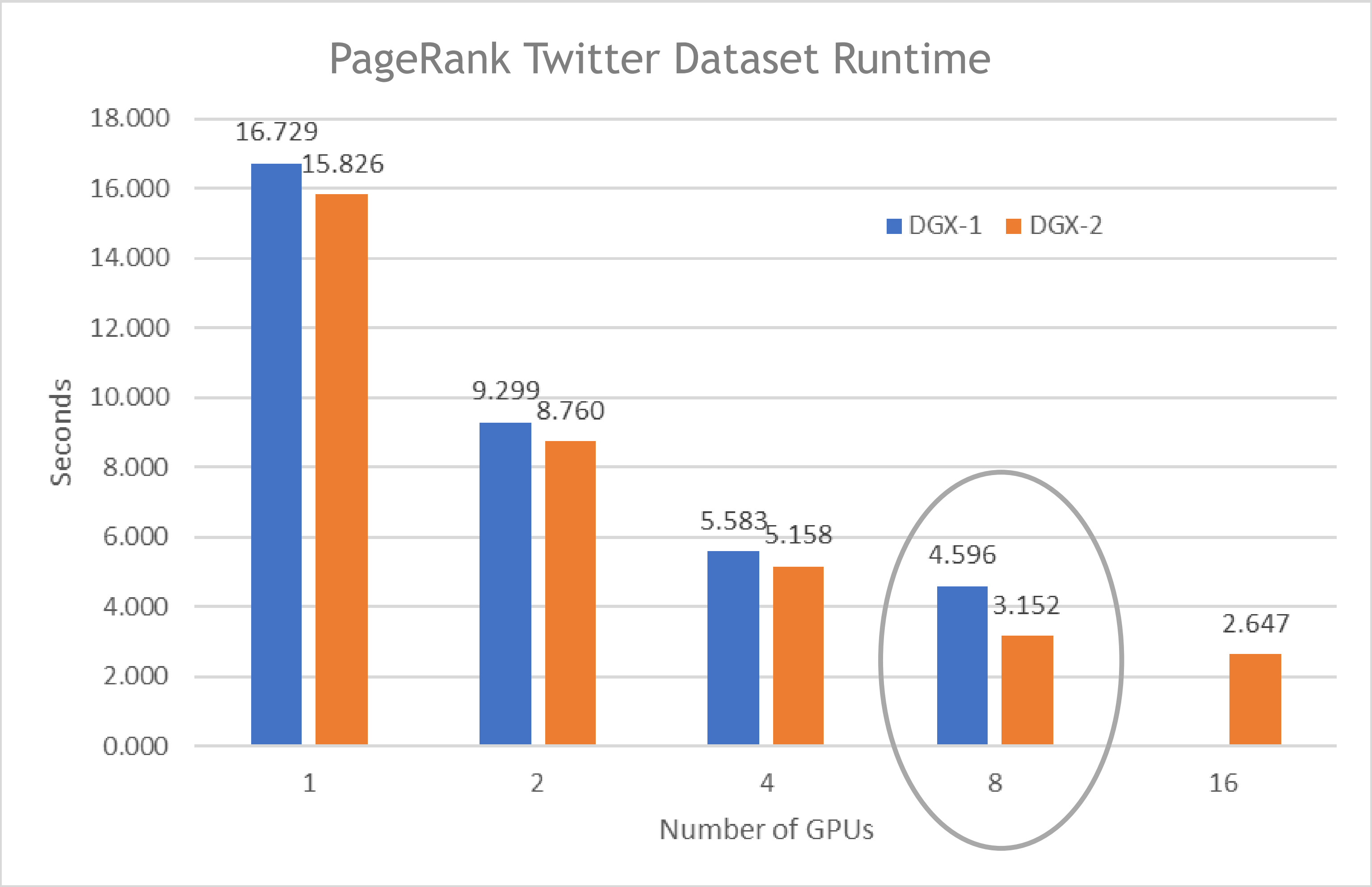
Using Gunrock, Multi-GPU



DGX-2


PageRank Runtimes
Seconds
20
18
16
14
12
10
8
6
4
2
0
soc-twitter
RMAT 27
RMAT 28
17.189
6.686
1.703
8.850
3.445
0.945
4.554
2.034
0.621
2.515
1.006
0.428
1.934
0.886
0.363
1
2
4
8
10
16
Number of GPUs

DGX-1 VS. DGX-2


PageRank Twitter Dataset Runtime
18.000
16.000
14.000
12.000
10.000
8.000
6.000
4.000
2.000
0.000
Seconds
16.729
15.826
9.299
8.760
5.583
5.158
4.596
3.152
2.647
DGX-1
DGX-2
1
2
4
8
16
Number of GPUs

RMAT SCALING, STAGE 4 PR PIPELINE

Near Constant Time Weak Scaling is Real Due to NVLINK

GPU Count	Max RMAT scale	Comp time (sec)	Gedges/sec	MFLOPS	NVLINK Speedup
1	25	1.4052	7.6	15282.90	1.0
2	26	1.3914	15.4	30867.37	1.4
4	27	1.3891	30.9	61838.78	2.8
8	28	1.4103	60.9	121815.46	4.1
16	29	1.4689	117.0	233917.04	8.1

CONCLUSIONS

Can Do Real Graphs on GPUs

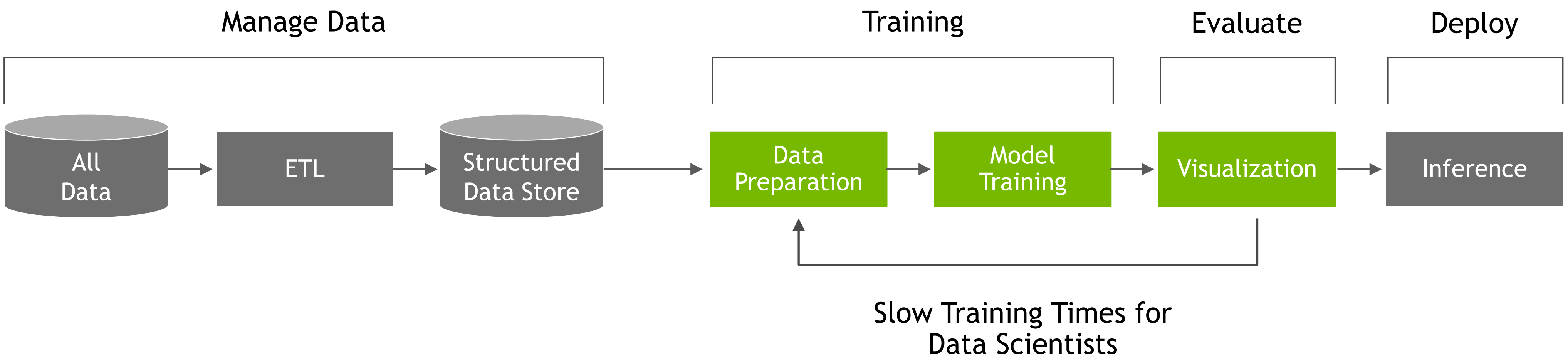
- The benefits are full NVLink connectivity between GPUs is evident with any analytic that needs to share data between GPUs
- DGX-2 is able to handle graphs into the billions of edges
- Frameworks need to be updated to support more than 8 GPUs, some have hardcoded limits due to DGX-1

So what is next?

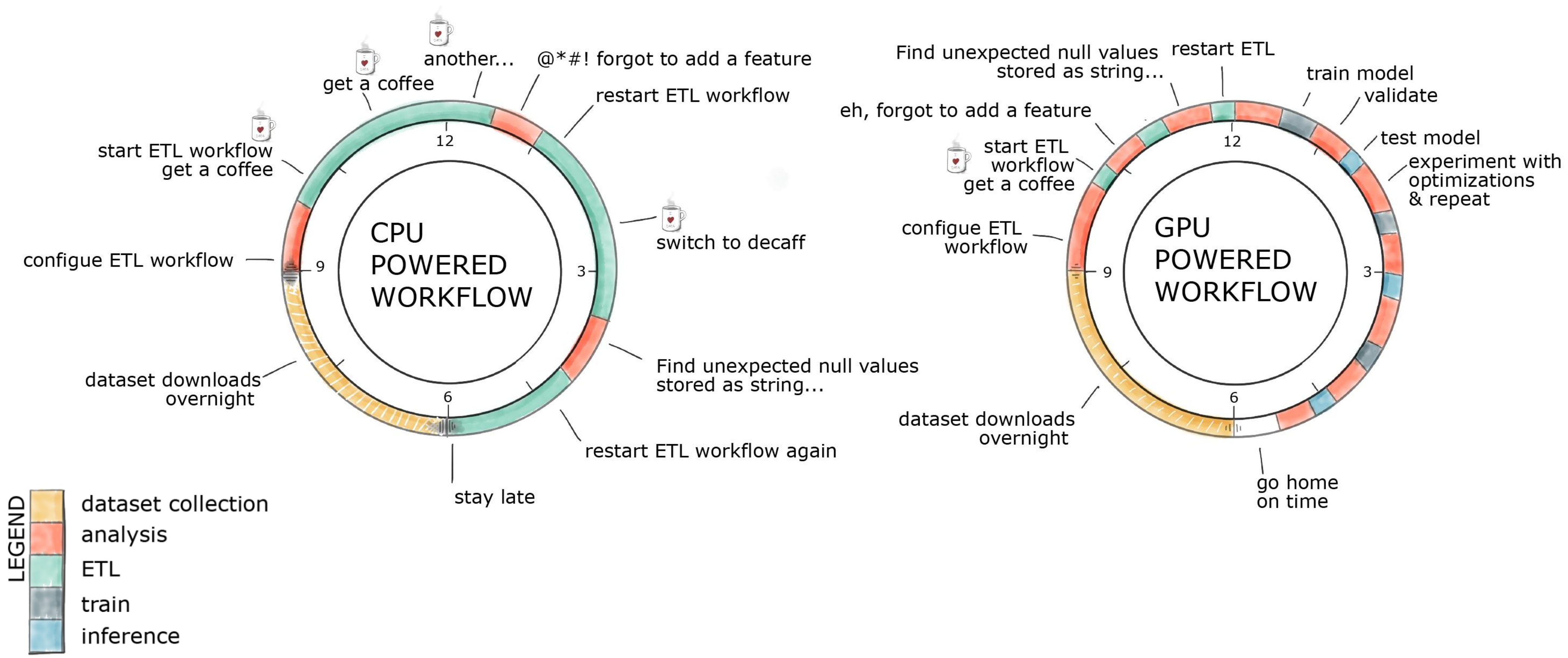
RAPIDS

https://rapids.ai

THE BIG PROBLEM IN DATA SCIENCE

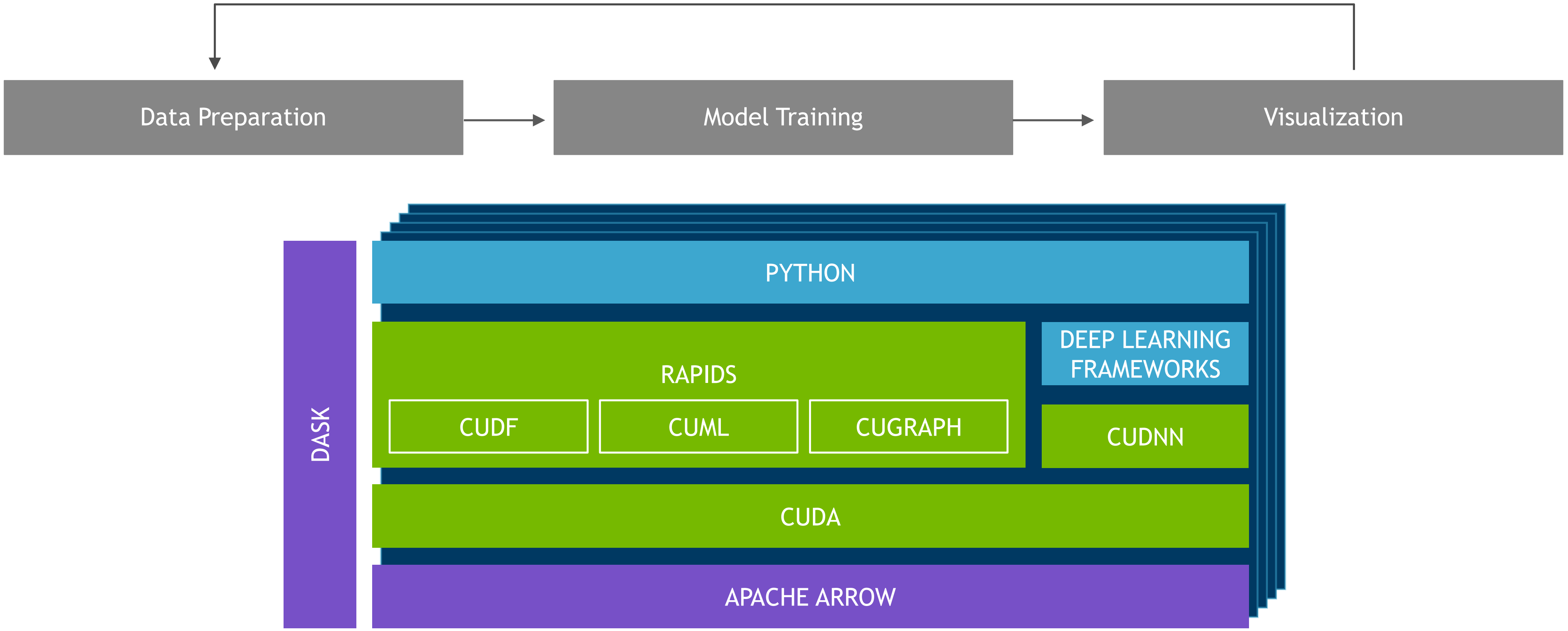


DAY IN THE LIFE OF A DATA SCIENTIST



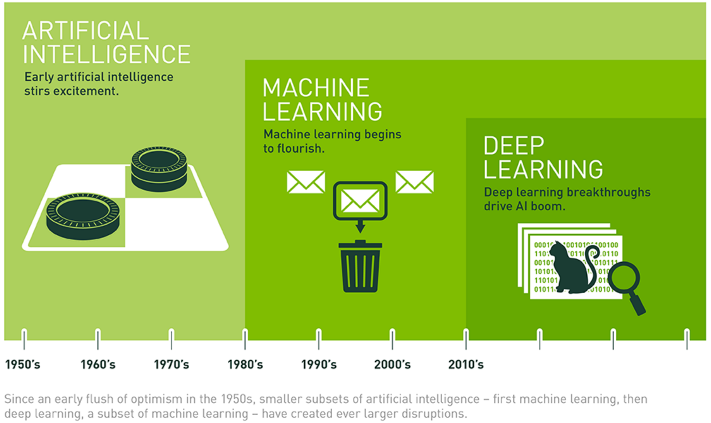
RAPIDS — OPEN GPU DATA SCIENCE

Software Stack



AI LIBRARIES

cuML & cuGraph



Accelerating more of the AI ecosystem

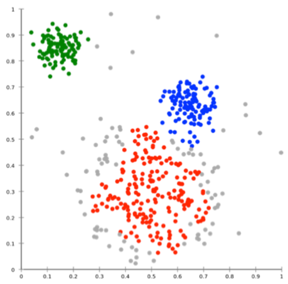
Graph Analytics is fundamental to network analysis

Machine Learning is fundamental to prediction, classification, clustering, anomaly detection and recommendations.

Both can be accelerated with NVIDIA GPU

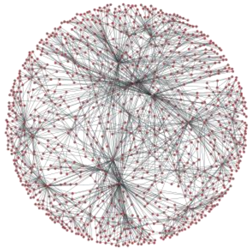
8x V100 20-90x faster than dual socket CPU

Machine Learning



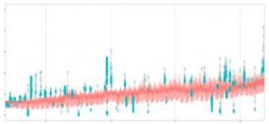
Decisions Trees
Random Forests
Linear Regressions
Logistics Regressions
K-Means
K-Nearest Neighbor
DBSCAN
Kalman Filtering
Principal Components
Single Value Decomposition
Bayesian Inferencing

Graph Analytics



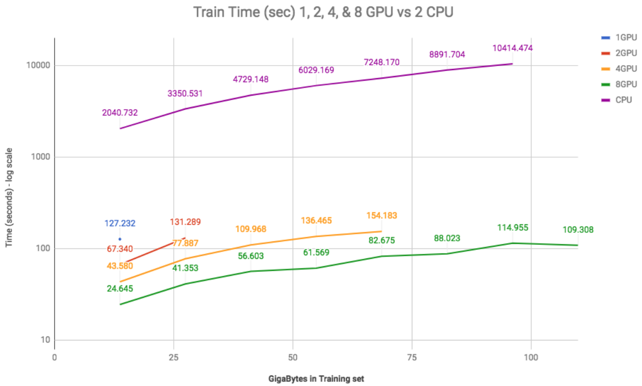
PageRank
BFS
Jaccard Similarity
Single Source Shortest Path
Triangle Counting
Louvain Modularity

Time Series



ARIMA
Holt-Winters

XGBoost, Criteo Dataset, 90x



3 Hours to 2 mins on 1 DGX-1

NVIDIA

cuML — MACHINE LEARNING

GPU Accelerated Scikit-learn + XGBoost Libraries

Dask

Distributed Training: Used for distributed cuML model training

Python API

Language Bindings: Python bindings to C++/CUDA based cuML | Uses cuDF DataFrames as input

cuML

C++/CUDA ML Algorithms: C++/CUDA machine learning algorithms

ml-prims

CUDA ML Primitives: Low level machine learning primitives used in cuML | Linear algebra, statistics, matrix operations, distance functions, random number generation

Dask
Distributed Training

Python API
Language Bindings

cuML
C++/CUDA ML algorithms

ml-prims
CUDA ML primitives

cuML — ROADMAP

Scikit-learn + XGBoost

cuML Algorithms	Available Now	Q4-2018	Q1-2019
XGBoost GBDT	MGMN		
Truncated Singular Value Decomposition (tSVD)	SG		MG
Principal Component Analysis (PCA)	SG		MG
Density-based Spatial Clustering of Applications with Noise (DBSCAN)	SG		MG
XGBoost Random Forest		MGMN	
K-Means Clustering		MG	
Kalman Filter		SG	MG
FAISS K-NN		MG	MGMN
GLM (including Logistic)			MGMN
Time Series			MG
Support Vector Machines			MGMN
Collaborative Filtering			MG
UMAP			MG

SG
Single GPU

MG
Multi-GPU

MGMN
Multi-GPU Multi-Node

Last updated 10.16.18

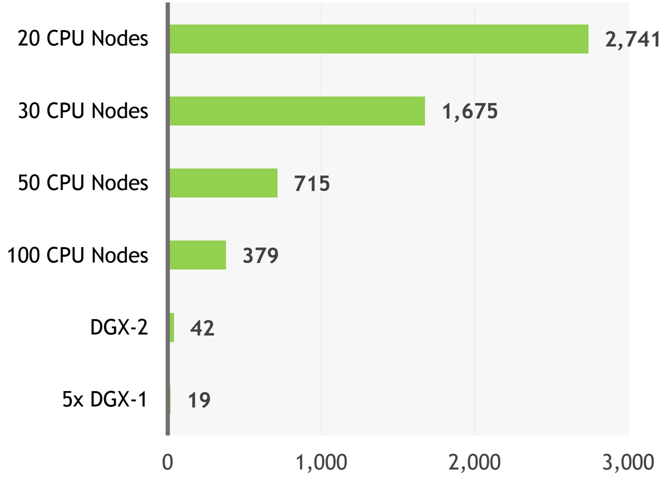
cuGRAPH — GRAPH ANALYTICS

GPU Accelerated Unified Graph Analytics

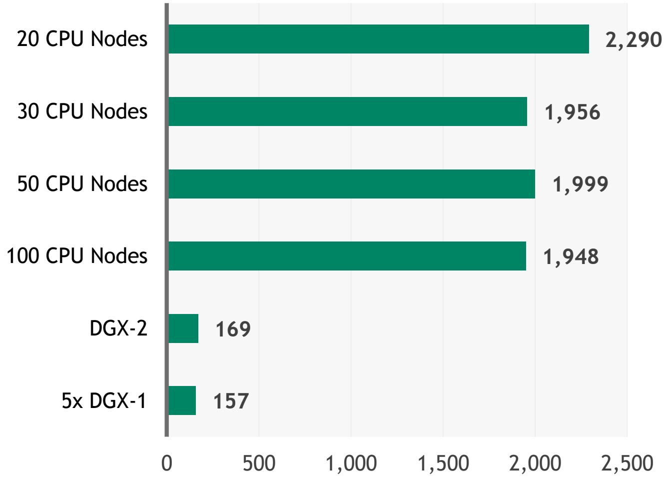
- Unifies the GPU accelerated graph analytics libraries
    - cuGraph = nvGraph + Gunrock + Hornet
    - Single and Multi-GPU versions of graph algorithms
- Available soon …

BENCHMARKS

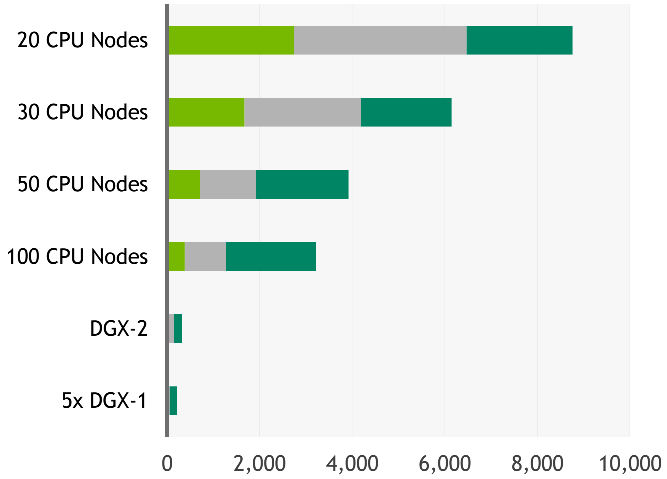
cuIO/cuDF — Load and Data Preparation



cuML — XGBoost



End-to-End



Time in seconds — Shorter is better

cuIO / cuDF (Load and Data Preparation) Data Conversion XGBoost

Benchmark

200GB CSV dataset; Data preparation includes joins, variable transformations.

CPU Cluster Configuration

CPU nodes (61 GiB of memory, 8 vCPUs, 64-bit platform), Apache Spark

DGX Cluster Configuration

5x DGX-1 on InfiniBand network

DOWNLOAD AND DEPLOY

GitHub

NGC

docker

CONDA

python Package Index

Source code, libraries, packages

On-premises

Cloud

NVIDIA®