RAPIDS
The Platform Inside and Out
Release 0.8

Joshua Patterson - GM, Data Science
RAPIDS
End-to-End Accelerated GPU Data Science

Data Preparation → Model Training → Visualization

Dask

cuDF cuIO Analytics  →  cuML Machine Learning  →  cuGraph Graph Analytics  →  PyTorch Chainer MxNet Deep Learning  →  cuXfilter <-> pyViz Visualization

GPU Memory  →  Apache Arrow
Data Processing Evolution
Faster data access, less data movement

Hadoop Processing, Reading from disk

Spark In-Memory Processing

Traditional GPU Processing

25-100x Improvement
Less code
Language flexible
Primarily In-Memory

5-10x Improvement
More code
Language rigid
Substantially on GPU
Data Movement and Transformation

The bane of productivity and performance

- **CPU**
  - **APP A**
  - **Copy & Convert**
  - **APP B**

- **GPU**
  - **APP A**
  - **Copy & Convert**
  - **GPU Data**
  - **Copy & Convert**
  - **GPU Data**

- **APP B**
  - **Read Data**
  - **Load Data**
Data Movement and Transformation

What if we could keep data on the GPU?
Learning from Apache Arrow

- Each system has its own internal memory format
- 70-80% computation wasted on serialization and deserialization
- Similar functionality implemented in multiple projects

- All systems utilize the same memory format
- No overhead for cross-system communication
- Projects can share functionality (e.g., Parquet-to-Arrow reader)

*From Apache Arrow Home Page - https://arrow.apache.org/*
Data Processing Evolution
Faster data access, less data movement

Hadoop Processing, Reading from disk

Spark In-Memory Processing

Traditional GPU Processing

RAPIDS
Faster Speeds, Real-World Benefits

Benchmark
200GB CSV dataset; Data prep includes joins, variable transformations

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

DGX Cluster Configuration
5x DGX-1 on InfiniBand network

Time in seconds (shorter is better)

**cuIO/cuDF - Load and Data Preparation**
- 20 CPU Nodes: 2741
- 30 CPU Nodes: 1675
- 50 CPU Nodes: 715
- 100 CPU Nodes: 379
- DGX-2: 42
- 5x DGX-1: 19

**cuML - XGBoost**
- 20 CPU Nodes: 2290
- 30 CPU Nodes: 1956
- 50 CPU Nodes: 1999
- 100 CPU Nodes: 1948
- DGX-2: 169
- 5x DGX-1: 157

**End-to-End**
- 20 CPU Nodes: 8762
- 30 CPU Nodes: 6148
- 50 CPU Nodes: 3925
- 100 CPU Nodes: 3221
- DGX-2: 322
- 5x DGX-1: 213
Speed, UX, and Iteration
The Way to Win at Data Science

Winners are those who went through "more iterations" of the "loop of progress" — going from an idea, to its implementation, to actionable results. So the winning teams are simply those able to run through this loop "faster".

And this is were Keras gives you an edge.

Visualization & understanding

Results

Infrastructure

Experiment

Software tools

Idea

François Cholet  @fcholet  · Apr 3

We often talk about how following UX best practices for API design makes Keras more accessible and easier to use, and how this helps beginners.

But those who stand to benefit most from good UX "aren’t" the beginners. It's actually the very best practitioners in the world.

François Cholet  @fcholet  · Apr 3

Because good UX reduces the overhead (development overhead & cognitive overhead) to setting up new experiments. It means you will be able to iterate faster. You will be able to try more ideas.

And ultimately, that’s how you win competitions or get papers published.

François Cholet  @fcholet  · Apr 3

So I don’t think it’s mere personal preference if Kaggel champions are overwhelmingly using Keras.

Using Keras means you’re more likely to win, and inversely, those who practice the sort of fast experimentation strategy that sets them up to win are more likely to prefer Keras.

Joshua Patterson  @datametrician  · Apr 3

Replying to @fcholet:

This is the fundamental belief that drives @RAPIDSai, @nvidia GPU infrastructure is fast, people need to iterate quickly, people want a known #python interface. Combine them and you’re off to the races!

All (primary + auxiliary) ML software tools used by top-5 Kaggle teams in each competition (n=120)
Open Source Data Science Ecosystem

Familiar Python APIs

Data Preparation → Model Training → Visualization

- Dask
- Pandas Analytics
- Scikit-Learn Machine Learning
- NetworkX Graph Analytics
- PyTorch Chainer MxNet Deep Learning
- Matplotlib/Seaborn Visualization

CPU Memory
RAPIDS
End-to-End Accelerated GPU Data Science
Dask
RAPIDS
Scaling RAPIDS with Dask

Data Preparation → Model Training → Visualization

Dask

cuDF cuIO Analytics → cuML Machine Learning → cuGraph Graph Analytics → PyTorch Chainer MxNet Deep Learning → cuXfilter <-> pyViz Visualization

GPU Memory

Apache Arrow
Why Dask?

• PyData Native
  • Built on top of NumPy, Pandas Scikit-Learn, etc. (easy to migrate)
  • With the same APIs (easy to train)
  • With the same developer community (well trusted)

• Scales
  • Easy to install and use on a laptop
  • Scales out to thousand-node clusters

• Popular
  • Most common parallelism framework today at PyData and SciPy conferences

• Deployable
  • HPC: SLURM, PBS, LSF, SGE
  • Cloud: Kubernetes
  • Hadoop/Spark: Yarn
Why OpenUCX?

Bringing hardware accelerated communications to Dask

• TCP sockets are slow!

• UCX provides uniform access to transports (TCP, InfiniBand, shared memory, NVLink)

• Python bindings for UCX (ucx-py) in the works

• Will provide best communication performance, to Dask based on available hardware on nodes/cluster
Scale up with RAPIDS

**RAPIDS and Others**

- Accelerated on single GPU
- NumPy -> CuPy/PyTorch/.. 
- Pandas -> cuDF 
- Scikit-Learn -> cuML 
- Numba -> Numba

**PyData**

- NumPy, Pandas, Scikit-Learn, Numba and many more 
- Single CPU core 
- In-memory data
Scale out with RAPIDS + Dask with OpenUCX

**RAPIDS and Others**
Accelerated on single GPU
- NumPy -> CuPy/PyTorch/...
- Pandas -> cuDF
- Scikit-Learn -> cuML
- Numba -> Numba

**PyData**
NumPy, Pandas, Scikit-Learn, Numba and many more
- Single CPU core
- In-memory data

**Dask**
Multi-core and Distributed PyData
- NumPy -> Dask Array
- Pandas -> Dask DataFrame
- Scikit-Learn -> Dask-ML
- ... -> Dask Futures

**RAPIDS + Dask with OpenUCX**
Multi-GPU
- On single Node (DGX)
- Or across a cluster
cuDF
GPU-Accelerated ETL
The average data scientist spends 90+% of their time in ETL as opposed to training models.
ETL - the Backbone of Data Science

libcuDF is...

CUDA C++ Library

- Low level library containing function implementations and C/C++ API
- Importing/exporting Apache Arrow in GPU memory using CUDA IPC
- CUDA kernels to perform element-wise math operations on GPU DataFrame columns
- CUDA sort, join, groupby, reduction, etc. operations on GPU DataFrames

```c
void some_function( cudf::column const* input,
                 cudf::column * output,
                 args...)
{
    // Do something with input
    // Produce output
}
```
ETL - the Backbone of Data Science

cuDF is...

Python Library

- A Python library for manipulating GPU DataFrames following the Pandas API
- Python interface to CUDA C++ library with additional functionality
- Creating GPU DataFrames from Numpy arrays, Pandas DataFrames, and PyArrow Tables
- JIT compilation of User-Defined Functions (UDFs) using Numba
ETL - the Backbone of Data Science

cuDF is not the end of the story
ETL - the Backbone of Data Science

String Support

Now v0.8 String Support:
- Regular Expressions
- Element-wise operations
  - Split, Find, Extract, Cat, Typecasting, etc...
- String GroupBys, Joins

Future v0.9+ String Support:
- Combining cuStrings into libcudf
- Extensive performance optimization
- More Pandas String API compatibility
- Improved Categorical column support
Extraction is the Cornerstone of ETL

cuIO is born

• Follows the APIs of Pandas and provide >10x speedup
• CSV Reader - v0.2, CSV Writer v0.8
• Parquet Reader - v0.7
• ORC Reader - v0.7
• JSON Reader - v0.8
• Avro Reader - v0.9
• HDF5 Reader - v0.10

• Key is GPU-accelerating both parsing and decompression wherever possible

Source: Apache Crail blog: SQL Performance: Part 1 - Input File Formats
ETL is not just DataFrames!
RAPIDS
Building bridges into the array ecosystem

Data Preparation → Model Training → Visualization

Dask

cuDF cuIO Analytics

cuML Machine Learning

cuGraph Graph Analytics

PyTorch Chainer MxNet Deep Learning

cuXfilter <-> pyViz Visualization

GPU Memory

Apache Arrow
Interoperability for the Win

DLPack and __cuda_array_interface__

**PyTorch**

**mpi4py**

**mxnet**

**Numba**

**Chainer**

**CuPy**
Interoperability for the Win

DLPack and __cuda_array_interface__
ETL - Arrays and DataFrames
Dask and CUDA Python arrays

- Scales NumPy to distributed clusters
- Used in climate science, imaging, HPC analysis up to 100TB size
- Now seamlessly accelerated with GPUs
Benchmark: single-GPU CuPy vs NumPy

SVD Benchmark
Dask and CuPy Doing Complex Workflows

![Graph showing performance comparison between Dask and CuPy in SVD benchmark]
Also...Achievement Unlocked:
Petabyte Scale Data Analytics with Dask and CuPy

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single CPU Core</td>
<td>2hr 39min</td>
</tr>
<tr>
<td>Forty CPU Cores</td>
<td>11min 30s</td>
</tr>
<tr>
<td>One GPU</td>
<td>1min 37s</td>
</tr>
<tr>
<td>Eight GPUs</td>
<td>19s</td>
</tr>
</tbody>
</table>

Cluster configuration: 20x GCP instances, each instance has:

- **CPU**: 1 VM socket (Intel Xeon CPU @ 2.30GHz), 2-core, 2 threads/core, 132GB mem, GbE ethernet, 950 GB disk
- **GPU**: 4x NVIDIA Tesla P100-16GB-PCle (total GPU DRAM across nodes 1.22 TB)
- **Software**: Ubuntu 18.04, RAPIDS 0.5.1, Dask=1.1.1, Dask-Distributed=1.1.1, CuPY=5.2.0, CUDA 10.0.130

https://blog.dask.org/2019/01/03/dask-array-gpus-first-steps
ETL - Arrays and DataFrames

More Dask Awesomeness from RAPIDS

https://youtu.be/gV0cykgsTPM

https://youtu.be/R5CiXti_MWo
cuML
Machine Learning
More models more problems

data preparation → model training → visualization

Dask

cuDF cuIO
Analytics

cuML
Machine Learning

cuGraph
Graph Analytics

PyTorch Chainer MxNet
Deep Learning

cuXfilter <-> pyViz
Visualization

GPU Memory

Apache Arrow
Problem

Data sizes continue to grow

Massive Dataset

Histograms / Distributions

Dimension Reduction
Feature Selection

Remove Outliers

Sampling

Better to start with as much data as possible and explore / preprocess to scale to performance needs.

Hours? Days?

Time Increases

Iterate. Cross Validate & Grid Search. Iterate some more.

Meet reasonable speed vs accuracy tradeoff
ML Technology Stack

- Python
- Cython
- cuML Algorithms
- cuML Prims
- CUDA Libraries
- CUDA

- Dask cuML
- Dask cuDF
- cuDF
- Numpy

- Thrust
- Cub
- cuSolver
- nvGraph
- CUTLASS
- cuSparse
- cuRand
- cuBlas
Algorithms
GPU-accelerated Scikit-Learn

Classification / Regression
- Decision Trees / Random Forests
- Linear Regression
- Logistic Regression
- K-Nearest Neighbors
- Kalman Filtering
- Bayesian Inference
- Gaussian Mixture Models
- Hidden Markov Models

Statistical Inference
- K-Means
- DBSCAN
- Spectral Clustering
- Principal Components
- Singular Value Decomposition
- UMAP
- Spectral Embedding
- ARIMA
- Holt-Winters

Clustering
- Implicit Matrix Factorization

Decomposition & Dimensionality Reduction
- Recommendations
- Cross Validation
- Hyper-parameter Tuning

More to come!
from sklearn.datasets import make_moons
import pandas

X, y = make_moons(n_samples=int(1e2),
                  noise=0.05, random_state=0)

X = pandas.DataFrame({'fea%d' % i: X[:, i]
                      for i in range(X.shape[1])})

from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)
dbscan.fit(X)
y_hat = dbscan.predict(X)
GPU-Accelerated Clustering

Code Example

```python
from sklearn.datasets import make_moons
import cudf
X, y = make_moons(n_samples=int(1e2), noise=0.05, random_state=0)
X = cudf.DataFrame({f'fea%d' % i: X[:, i] for i in range(X.shape[1])})

from cuml import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)
dbscan.fit(X)
y_hat = dbscan.predict(X)
```
Benchmarks: single-GPU cuML vs scikit-learn

1x V100 vs 2x 20 core CPU
## Road to 1.0

**October 2018 - RAPIDS 0.1**

<table>
<thead>
<tr>
<th>cuML</th>
<th>Single-GPU</th>
<th>Multi-GPU</th>
<th>Multi-Node-Multi-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Boosted Decision Trees (GBDT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logistic Regression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Forest</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-NN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBSCAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARIMA &amp; Holts-Winters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kalman Filter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t-SNE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principal Components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rapid to 1.0  
June 2019 - RAPIDS 0.8

<table>
<thead>
<tr>
<th>cuML</th>
<th>Single-GPU</th>
<th>Multi-GPU</th>
<th>Multi-Node-Multi-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Boosted Decision Trees (GBDT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logistic Regression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Forest</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-NN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBSCAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARIMA &amp; Holts-Winters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kalman Filter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t-SNE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principal Components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Road to 1.0

**August 2019 - RAPIDS 0.9**

<table>
<thead>
<tr>
<th>cuML</th>
<th>Single-GPU</th>
<th>Multi-GPU</th>
<th>Multi-Node-Multi-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Boosted Decision Trees (GBDT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logistic Regression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Forest</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-NN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBSCAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARIMA &amp; Holts-Winters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kalman Filter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t-SNE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principal Components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Road to 1.0

January 2020 - RAPIDS 0.12?

<table>
<thead>
<tr>
<th>cuML</th>
<th>Single-GPU</th>
<th>Multi-GPU</th>
<th>Multi-Node-Multi-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Boosted Decision Trees (GBDT)</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
</tr>
<tr>
<td>GLM</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
</tr>
<tr>
<td>Random Forest</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
</tr>
<tr>
<td>K-Means</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
</tr>
<tr>
<td>K-NN</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
</tr>
<tr>
<td>DBSCAN</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
</tr>
<tr>
<td>UMAP</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
</tr>
<tr>
<td>ARIMA &amp; Holts-Winters</td>
<td>Green</td>
<td>Green</td>
<td></td>
</tr>
<tr>
<td>Kalman Filter</td>
<td>Green</td>
<td>Green</td>
<td></td>
</tr>
<tr>
<td>t-SNE</td>
<td>Green</td>
<td>Green</td>
<td></td>
</tr>
<tr>
<td>Principal Components</td>
<td>Green</td>
<td>Green</td>
<td></td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td>Green</td>
<td>Green</td>
<td></td>
</tr>
</tbody>
</table>
cuGraph
Graph Analytics

More connections more insights
GOALS AND BENEFITS OF CUGRAPH
Focus on Features an Easy-of-Use

• Seamless integration with cuDF and cuML
  • Python APIs accepts and returns cuDF DataFrames

• Features
  • Extensive collection of algorithm, primitive, and utility functions**
  • Python API:
    • Multiple APIs: NetworkX, Pregel**, Frontier**
    • Graph Query Language**
  • C/C++
    • Full featured C++ API that gives both an easy to use experience, as well as exposing lower-level granular control to developers

• Breakthrough Performance

** On Roadmap
nvGRAPH has been Opened Sourced and integrated into cuGraph. A legacy version is available in a RAPIDS GitHub repo

* Gunrock is from UC Davis
Algorithms

GPU-accelerated NetworkX

Community
- Spectral Clustering
- Balanced-Cut
- Modularity Maximization
- Louvain
- Subgraph Extraction
- Triangle Counting

Components
- Weakly Connected Components
- Strongly Connected Components

Link Analysis
- Page Rank
- Personal Page Rank

Link Prediction
- Jaccard
- Weighted Jaccard
- Overlap Coefficient

Traversal
- Single Source Shortest Path (SSSP)
- Breadth First Search (BFS)

Structure
- COO-to-CSR
- Transpose
- Renumbering

Query Language

Multi-GPU

Utilities

More to come!
WHAT’S IN CUGRAPH
Current Single GPU Algorithms - as of v0.8

- COO-to/from-CSR Conversion
- Renumbering
- PageRank
- Personal PageRank
- Jaccard (1 and 2-hop)
- Weighted Jaccard
- Overlap Coefficient
- Single Source Shortest Path (SSSP)
- Breadth First Search (BFS)
- Triangle Counting (TC)
- Subgraph Extraction
- Spectral Clustering
  - Balanced-Cut
  - Modularity Maximization
- Louvain
- Graph functions
  - Size, Order, Degree
  - Get two hop neighbors

MG - Multi-GPU

SG - Single GPU
Louvain Single Run

G = cugraph.Graph()
G.add_edge_list(gdf["src_0"], gdf["dst_0"], gdf["data"]) df, mod = cugraph.nvLouvain(G)

Louvain returns:
cudf.DataFrame with two names columns:
louvain["vertex"]: The vertex id.
louvain["partition"]: The assigned partition.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>preferentialAttachment</td>
<td>100,000</td>
<td>999,970</td>
</tr>
<tr>
<td>caidaRouterLevel</td>
<td>192,244</td>
<td>1,218,132</td>
</tr>
<tr>
<td>coAuthorsDBLP</td>
<td>299,067</td>
<td>299,067</td>
</tr>
<tr>
<td>dblp-2010</td>
<td>326,186</td>
<td>1,615,400</td>
</tr>
<tr>
<td>citationCiteseer</td>
<td>268,495</td>
<td>2,313,294</td>
</tr>
<tr>
<td>coPapersDBLP</td>
<td>540,486</td>
<td>30,491,458</td>
</tr>
<tr>
<td>coPapersCiteseer</td>
<td>434,102</td>
<td>32,073,440</td>
</tr>
<tr>
<td>as-Skitter</td>
<td>1,696,415</td>
<td>22,190,596</td>
</tr>
</tbody>
</table>

Performance Speedup: cuGraph vs NetworkX
# PageRank Performance

**Performance Using PageRank Pipeline Benchmark**

Running on V100 32G

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
<th>Single GPU</th>
<th>Dual GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,048,576</td>
<td>16,777,216</td>
<td>0.019</td>
<td>0.20</td>
</tr>
<tr>
<td>2,097,152</td>
<td>3,354,432</td>
<td>0.047</td>
<td>0.035</td>
</tr>
<tr>
<td>4,194,304</td>
<td>134,217,728</td>
<td>0.302</td>
<td>0.162</td>
</tr>
<tr>
<td>16,777,216</td>
<td>268,435,456</td>
<td>0.771</td>
<td>0.353</td>
</tr>
<tr>
<td>33,554,432</td>
<td>536,870,912</td>
<td>1.747</td>
<td>0.821</td>
</tr>
<tr>
<td>67,108,864</td>
<td>1,073,741,824</td>
<td>1.747</td>
<td>1.880</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>33,554,432</td>
<td>536,870,912</td>
<td>1.405</td>
</tr>
<tr>
<td>67,108,864</td>
<td>1,073,741,824</td>
<td>1.389</td>
</tr>
<tr>
<td>134,217,728</td>
<td>2,122,307,214</td>
<td>1.389</td>
</tr>
<tr>
<td>268,435,456</td>
<td>4,294,967,296</td>
<td>1.410</td>
</tr>
<tr>
<td>536,870,912</td>
<td>8,589,934,592</td>
<td>1.468</td>
</tr>
</tbody>
</table>

Single and Dual GPU on Commodity Workstation (PCIe)

Single DGX-2 (16 NVLinked GPUs)

** Benchmarking code. Final produce performance might differ
<table>
<thead>
<tr>
<th>cuGraph</th>
<th>Single-GPU</th>
<th>Multi-GPU</th>
<th>Multi-Node-Multi-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jaccard and Weighted Jaccard</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page Rank</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Personal Page Rank</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSSP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BFS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triangle Counting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subgraph Extraction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Katz Centrality</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Betweenness Centrality</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connected Components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Louvain</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral Clustering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InfoMap</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Cores</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Road to 1.0
### August 2019 - RAPIDS 0.9

<table>
<thead>
<tr>
<th>cuGraph</th>
<th>Single-GPU</th>
<th>Multi-GPU</th>
<th>Multi-Node-Multi-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jaccard and Weighted Jaccard</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page Rank</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Personal Page Rank</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSSP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BFS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triangle Counting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subgraph Extraction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Katz Centrality</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Betweenness Centrality</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connected Components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Louvain</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral Clustering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InfoMap</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Cores</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Road to 1.0

**January 2020 - RAPIDS 0.12?**

<table>
<thead>
<tr>
<th>cuGraph</th>
<th>Single-GPU</th>
<th>Multi-GPU</th>
<th>Multi-Node-Multi-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jaccard and Weighted Jaccard</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page Rank</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Personal Page Rank</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSSP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BFS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triangle Counting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subgraph Extraction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Katz Centrality</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Betweenness Centrality</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connected Components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Louvain</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral Clustering</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InfoMap</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Cores</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
UPCOMING
(MG == Multi-GPU, SG == Single-GPU)

• Up Next: 0.9
  • MG PageRank
  • MG COO-to-CSR
  • MG Degree Computation
  • Strongly Connected Component
  • Initial Hornet integration (cuHornet)
  • Draft Pregel/BSP versions
    • bicliques
  • Utility functions

• Next Steps: 0.10
  • Initial Gunrock Integration
  • Subgraph Matching
  • Initial graphBLAS Integration
  • cuHornet Analytics
    • Katz Centrality
    • K-Cores
  • Utilities
    • Symmetrization
    • Transpose

• The Future: 0.10 +
  • Graph query language: OpenCypher
  • MG BSF
  • Expanded community detection
  • Dynamic Graph support
  • Betweenness Centrality
  • More Multi-GPU analytics
Community
Ecosystem Partners

CONTRIBUTORS

ADOPTERS

OPEN SOURCE
Building on top of RAPIDS
A bigger, better, stronger ecosystem for all

nuclio

High-Performance
Serverless event and
data processing that
utilizes RAPIDS for GPU
Acceleration

Streamz

blazingSQL

GPU accelerated SQL
game built on top of
RAPIDS

Distributed stream
processing using
RAPIDS and Dask
Deploy RAPIDS Everywhere
Focused on robust functionality, deployment, and user experience

Integration with major cloud providers
Both containers and cloud specific machine instances
Support for Enterprise and HPC Orchestration Layers
Getting Started
RAPIDS Docs
New, improved, and easier to use

https://docs.rapids.ai
RAPIDS Docs
Easier than ever to get started with cuDF

10 Minutes to cuDF

Modeled after 10 Minutes to Pandas, this is a short introduction to cuDF, geared mainly for new users.

```python
[s1]: import os
import numpy as np
import pandas as pd
import cudf
cpy.random.seed(12)

# Portions of this were borrowed from the
# cuDF cheatsheet, existing cuDF documentation,
# and 10 Minutes to Pandas.
# Created November, 2018.

Object Creation

Creating a Series.

```python
[s2]: s = cudf.Series([1,2,3,None,4])
print(s)
```

Creating a DataFrame by specifying values for each column.
RAPIDS
How do I get the software?

- [https://github.com/rapidsai](https://github.com/rapidsai)
- [https://anaconda.org/rapidsai/](https://anaconda.org/rapidsai/)
- [https://hub.docker.com/r/rapidsai/rapidsai/](https://hub.docker.com/r/rapidsai/rapidsai/)
Join the Movement
Everyone can help!

APACHE ARROW
https://arrow.apache.org/
@ApacheArrow

RAPIDS
https://rapids.ai
@RAPIDSAI

Dask
https://dask.org
@Dask_dev

GPU Open Analytics Initiative
http://gpuopenanalytics.com/
@GPUOAI

Integrations, feedback, documentation support, pull requests, new issues, or code donations welcomed!