RAPIDS
End-to-End Accelerated GPU Data Science
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
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 (eg, 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

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

5-10x Improvement
More code
Language rigid
Substantially on GPU

50-100x Improvement
Same code
Language flexible
Primarily on GPU
Faster Speeds, Real-World Benefits

**cuIO/cuDF - Load and Data Preparation**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 CPU Nodes</td>
<td>2741</td>
</tr>
<tr>
<td>30 CPU Nodes</td>
<td>1675</td>
</tr>
<tr>
<td>50 CPU Nodes</td>
<td>715</td>
</tr>
<tr>
<td>100 CPU Nodes</td>
<td>379</td>
</tr>
<tr>
<td>DGX-2</td>
<td>42</td>
</tr>
<tr>
<td>5x DGX-1</td>
<td>19</td>
</tr>
</tbody>
</table>

**cuML - XGBoost**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 CPU Nodes</td>
<td>2290</td>
</tr>
<tr>
<td>30 CPU Nodes</td>
<td>1956</td>
</tr>
<tr>
<td>50 CPU Nodes</td>
<td>1999</td>
</tr>
<tr>
<td>100 CPU Nodes</td>
<td>1948</td>
</tr>
<tr>
<td>DGX-2</td>
<td>169</td>
</tr>
<tr>
<td>5x DGX-1</td>
<td>157</td>
</tr>
</tbody>
</table>

**End-to-End**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 CPU Nodes</td>
<td>8762</td>
</tr>
<tr>
<td>30 CPU Nodes</td>
<td>6148</td>
</tr>
<tr>
<td>50 CPU Nodes</td>
<td>3925</td>
</tr>
<tr>
<td>100 CPU Nodes</td>
<td>3221</td>
</tr>
<tr>
<td>DGX-2</td>
<td>322</td>
</tr>
<tr>
<td>5x DGX-1</td>
<td>213</td>
</tr>
</tbody>
</table>

---

**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)*
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

Software tools

Results

Infrastructure

Experiment

Idea

@fcholet

François Chollet

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.

@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.

@fcholet

APR 3

So I don’t think it’s mere personal preference if Kaggle 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 RAPIDS. @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!
RAPIDS Core
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

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
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

- **Easy Migration**: Built on top of NumPy, Pandas, Scikit-Learn, etc.
- **Easy Training**: With the same APIs
- **Trusted**: With the same developer community

Deployable

- **HPC**: SLURM, PBS, LSF, SGE
- **Cloud**: Kubernetes
- **Hadoop/Spark**: Yarn

Easy Scalability

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

Popular

- Most common parallelism framework today in the PyData and SciPy community
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

### RAPIDS + Dask with OpenUCX
Multi-GPU
- On single Node (DGX)
- Or across a cluster

### 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
GPU Accelerated data wrangling and feature engineering

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
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

```cpp
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
Benchmarks: single-GPU Speedup vs. Pandas

- cuDF v0.9, Pandas 0.24.2

Running on NVIDIA DGX-1:
- GPU: NVIDIA Tesla V100 32GB
- CPU: Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz

Benchmark Setup:
- DataFrames: 2x int32 columns key columns, 3x int32 value columns
- Merge: inner
- GroupBy: count, sum, min, max calculated for each value column
ETL - the Backbone of Data Science

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

String Support

**Current v0.9 String Support**
- Regular Expressions
- Element-wise operations
  - Split, Find, Extract, Cat, Typecasting, etc...
- String GroupBys, Joins
- Categorical columns fully on GPU

**Future v0.10+ String Support**
- Combining cuStrings into libcudf
- Extensive performance optimization
- More Pandas String API compatibility
- JIT-compiled String UDFs
Extraction is the Cornerstone
culIO for Faster Data Loading

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

• GPU Direct Storage integration in progress for bypassing PCIe bottlenecks!

• 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 configurations.](image)
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>

![3.2 PETABYTES IN LESS THAN 1 HOUR](https://blog.dask.org/2019/01/03/dask-array-gpus-first-steps)

3.2 PETABYTES IN LESS THAN 1 HOUR
Distributed GPU array | parallel reduction | using 76x GPUs

<table>
<thead>
<tr>
<th>Array size</th>
<th>Wall Time (data creation + compute)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2 PB (20M x 20M doubles)</td>
<td>54 min 51 s</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-PCIe (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
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 cuIOAnalytics
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

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

Histograms / Distributions

Dimension Reduction
Feature Selection

Remove Outliers

Sampling

Time Increases

Hours? Days?

Meet reasonable speed vs accuracy tradeoff

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

- Inference
  - Random forest / GBDT inference

- Clustering
  - K-Means
  - DBSCAN
  - Spectral Clustering

- Decomposition & Dimensionality Reduction
  - Principal Components
  - Singular Value Decomposition
  - UMAP
  - Spectral Embedding

- Time Series
  - Holt-Winters
  - Kalman Filtering

Key:
- ● Preexisting
- ● NEW for 0.9

More to come!
RAPIDS matches common Python APIs

CPU-Based Clustering

```python
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({f'fe{a}%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)
```
RAPIDS matches common Python APIs

GPU-Accelerated Clustering

```python
from cuml import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)
dbscan.fit(X)
y_hat = dbscan.predict(X)
```

```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'f{str(i)}': X[:, i] for i in range(X.shape[1])})
```
Benchmarks: single-GPU cuML vs scikit-learn

1x V100 vs 2x 20 core CPU

Rows:
- 1M
- 2M
- 4M
cuML’s Forest Inference Library accelerates prediction (inference) for random forests and boosted decision trees:

- Works with existing saved models (XGBoost and LightGBM today, scikit-learn RF and cuML RF soon)
- Lightweight Python API
- Single V100 GPU can infer up to 34x faster than XGBoost dual-CPU node
- Over 100 million forest inferences per sec (with 1000 trees) on a DGX-1

**Forest Inference**

**Taking models from training to production**
# 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>Holt-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
March 2020 - RAPIDS 0.14

<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; Holt-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>
cuGraph
Graph Analytics

More connections more insights

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
GOALS AND BENEFITS OF CUGRAPH

Focus on Features and User Experience

Breakthrough Performance

• Up to 500 million edges on a single 32GB GPU
• Multi-GPU support for scaling into the billions of edges

Seamless Integration with cuDF and cuML

• Property Graph support via DataFrames

Multiple APIs

• Python: Familiar NetworkX-like API
• C/C++: lower-level granular control for application developers

Growing Functionality

• Extensive collection of algorithm, primitive, and utility functions
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 (Multi-GPU)
- Personal Page Rank

Link Prediction
- Jaccard
- Weighted Jaccard
- Overlap Coefficient

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

Structure
- COO-to-CSR (Multi-GPU)
- Transpose
- Renumbering

Query Language

Multi-GPU

Utilities

More to come!
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
## Multi-GPU PageRank Performance

PageRank portion of the HiBench benchmark suite

<table>
<thead>
<tr>
<th>HiBench Scale</th>
<th>Vertices</th>
<th>Edges</th>
<th>CSV File (GB)</th>
<th># of GPUs</th>
<th>PageRank for 3 Iterations (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huge</td>
<td>5,000,000</td>
<td>198,000,000</td>
<td>3</td>
<td>1</td>
<td>1.1</td>
</tr>
<tr>
<td>BigData</td>
<td>50,000,000</td>
<td>1,980,000,000</td>
<td>34</td>
<td>3</td>
<td>5.1</td>
</tr>
<tr>
<td>BigData x2</td>
<td>100,000,000</td>
<td>4,000,000,000</td>
<td>69</td>
<td>6</td>
<td>9.0</td>
</tr>
<tr>
<td>BigData x4</td>
<td>200,000,000</td>
<td>8,000,000,000</td>
<td>146</td>
<td>12</td>
<td>18.2</td>
</tr>
<tr>
<td>BigData x8</td>
<td>400,000,000</td>
<td>16,000,000,000</td>
<td>300</td>
<td>16</td>
<td>31.8</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 (Weak and Strong)</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

**March 2020 - RAPIDS 0.14**

<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 (Weak and Strong)</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>
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

blazingSQL
GPU accelerated SQL engine built on top of RAPIDS

Streamz
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
5 Steps to getting started with RAPIDS

1. **Install** RAPIDS on using [Docker](https://docs.docker.com/), [Conda](https://docs.conda.io/en/latest/), or [Colab](https://colab.research.google.com/)
2. **Join** our community conversations on [Slack](https://join.slack.com), [Google](https://developers.google.com), and [Twitter](https://twitter.com)
3. **Explore** our [walk through videos](https://www.youtube.com), [blog content](https://www.example.com/blog), our [github](https://github.com), the [tutorial notebooks](https://www.example.com/notebooks), and our [examples workflows](https://www.example.com/workflows),
4. **Build** your own data science workflows.
5. **Contribute** back. Don’t forget to ask and answer questions on [Stack Overflow](https://stackoverflow.com)
Easy Installation
Interactive Installation Guide

RAPIDS RELEASE SELECTOR

RAPIDS is available as conda packages, docker images, and from source builds. Use the tool below to select your preferred method, packages, and environment to install RAPIDS. Certain combinations may not be possible and are dimmed automatically. Be sure you've met the required prerequisites above and see the details below.

METHOD
- Conda
- Docker + Examples
- Docker + Dev Env
- Source

RELEASE
- Stable (0.9)
- Nightly (0.10a)

PACKAGES
- cuDF
- cuML
- cuGraph
- All Packages

LINUX
- Ubuntu 16.04
- Ubuntu 18.04
- CentOS 7

PYTHON
- Python 3.6
- Python 3.7

CUDA
- CUDA 9.2
- CUDA 10.0

COMMAND
conda install -c rapidsai -c nvidia -c numba -c conda-forge -c anaconda \ cudf=0.9 cuml=0.9 cugraph=0.9 python=3.6 anaconda cudatoolkit=9.2

COPY COMMAND
Join the Conversation

Google Groups | Docker Hub | Slack Channel | Stack Overflow
Explore: RAPIDS Github

https://github.com/rapidsai
Explore: RAPIDS Docs
Improved and easier to use!

https://docs.rapids.ai
Explore: RAPIDS Code and Blogs

Check out our code and how we use it

https://github.com/rapidsai

https://medium.com/rapids-ai

RAPIDS Release 0.8: Same Community New Freedoms
Making more friends and building more bridges to more ecosystems. It’s now easier than ever to get started with RAPIDS.

Josh Patterson
3d 19 · 7 min read

gQuant—GPU Accelerated Examples for Quantitative Analysis Tasks
A simple trading strategy backtest for 5000 stocks using GPUs and getting 20X speedup

Yi Dong
3d 26 · 6 min read

NVIDIA GPUs and Apache Spark, One Step Closer
RAPIDS XGBoost4j-Spark Package Now Available

Karthikeyan Rajendran

When Less is More: A Brief Story about XGBoost Feature Engineering
A glimpse into how a Data Scientist makes decisions about feature engineering on an XGBoost machine

Nightly News: CI produces latest packages
Release code early and often. Stay current on latest features with our nightly conda and container releases.

Financial data modeling with RAPIDS.
See how RAPIDS was used to place 17th in the Banco Santander Kaggle Competition

Joel Liu
3d 3 · 5 min read
Explore: Notebooks Contrib

Notebooks Contrib Repo has tutorials and examples, and various E2E demos. RAPIDS Youtube channel has explanations, code walkthroughs and use cases.

<table>
<thead>
<tr>
<th>Folder</th>
<th>Notebook Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intro_tutorials</td>
<td>05_Introduction_to_Dask_cuDF</td>
<td>This notebook shows how to work with cuDF DataFrames distributed across multiple GPUs using Dask.</td>
</tr>
<tr>
<td>intro_tutorials</td>
<td>06_Introduction_to_Supervised_Learning</td>
<td>This notebook shows how to do GPU accelerated Supervised Learning in RAPIDS.</td>
</tr>
<tr>
<td>intro_tutorials</td>
<td>07_Introduction_to_XGBoost</td>
<td>This notebook shows how to work with GPU accelerated XGBoost in RAPIDS.</td>
</tr>
<tr>
<td>intro_tutorials</td>
<td>08_Introduction_to_Dask_XGBoost</td>
<td>This notebook shows how to work with Dask XGBoost in RAPIDS.</td>
</tr>
<tr>
<td>intro_tutorials</td>
<td>09_Introduction_to_Dimensionality_Reduction</td>
<td>This notebook shows how to do GPU accelerated Dimensionality Reduction in RAPIDS.</td>
</tr>
<tr>
<td>intro_tutorials</td>
<td>10_Introduction_to_Clustering</td>
<td>This notebook shows how to do GPU accelerated Clustering in RAPIDS.</td>
</tr>
</tbody>
</table>

Intermediate Notebooks:

<table>
<thead>
<tr>
<th>Folder</th>
<th>Notebook Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>examples</td>
<td>DBSCAN_Demo_FULL</td>
<td>This notebook shows how to use DBSCAN algorithm and its GPU accelerated implementation present in RAPIDS.</td>
</tr>
<tr>
<td>examples</td>
<td>Dask_with_cuDF_and_XGBoost</td>
<td>In this notebook we show how to quickly setup Dask and train an XGBoost model using cuDF.</td>
</tr>
</tbody>
</table>
Contribute Back

Issues, feature requests, PRs, Blogs, Tutorials, Videos, QA...bring your best!
Getting Started
RAPIDS Docs
New, improved, and easier to use

https://docs.rapids.ai
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
[1]:
import os
import numpy as np
import pandas as pd
import cudf
np.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
[2]:
s = cudf.Series([1,2,3,None,4])
print(s)

0    1
1    2
2    3
3    4

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

- https://github.com/rapidsai
- https://anaconda.org/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!
THANK YOU

Joshua Patterson  @datametrician
joshuap@nvidia.com