RAPIDS
The Platform Inside and Out
Release 0.14

Joshua Patterson - Senior Director, RAPIDS Engineering
Why GPUs?
Numerous hardware advantages

- Thousands of cores with up to ~20 TeraFlops of general purpose compute performance
- Up to 1.5 TB/s of memory bandwidth
- Hardware interconnects for up to 600 GB/s bidirectional GPU <--> GPU bandwidth
- Can scale up to 16x GPUs in a single node

Almost never run out of compute relative to memory bandwidth!
RAPIDS
End-to-End GPU Accelerated Data Science

Data Preparation → Model Training → Visualization

Dask

cuDF, cuIO: Analytics
cuML: Machine Learning
cuGraph: Graph Analytics
PyTorch, TensorFlow, MxNet: Deep Learning
cuxfilter, pyViz, plotly: 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
Data Movement and Transformation

What if We Could Keep Data on the GPU?

[Diagram showing data movement and transformation between CPU and GPU applications and data.]
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)

Source: 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

5-10x Improvement
More Code
Language Rigid
Substantially on GPU

50-100x Improvement
Same Code
Language Flexible
Primarily on GPU

25-100x Improvement
Less Code
Language Flexible
Primarily In-Memory
Speed, UX, and Iteration
The Way to Win at Data Science
RAPIDS 0.14 Release Summary
What’s New in Release 0.14?

- **cuDF** drops libcudf legacy code, greatly improves API documentation, and adds bug fixes as well as performance and memory usage optimizations.

- **cuML** machine learning library adds support for flexible data formats (inputs from Pandas, NumPy, cuDF or GPU) for all models, weighted K-means, and MNMG coordinate descent.

- **cuGraph** major improvement to the Louvain algorithm, new Betweenness Centrality, added Force Atlas 2, finished the big refactoring effort.

- **UCX-Py** adds InfiniBand support as well as Multi-Node Multi-GPU NVLink/InfiniBand tests.

- **BlazingSQL** now supports out-of-core query execution, which enables queries to operate on datasets dramatically larger than available GPU memory.

- **cuSignal** major refactoring and improvements to documentation and sample notebooks.

- **cuSpatial** adds fast quadtree building and major documentation and API improvements.

- **cuxfilter** adds Dask cuDF support, groupby optimizations, chart autoscaling and other general improvements.

- **RMM** adds fast logging of allocations/deallocations and an uninitialized vector container.
RAPIDS Everywhere
The Next Phase of RAPIDS

Exactly as it sounds—our goal is to make RAPIDS as usable and performant as possible wherever data science is done. We will continue to work with more open source projects to further democratize acceleration and efficiency in data science.
RAPIDS Core
Open Source Data Science Ecosystem
Familiar Python APIs
RAPIDS
End-to-End Accelerated GPU Data Science

Data Preparation → Model Training → Visualization

Dask

cuDF cuIO Analytics
cuML Machine Learning
cuGraph Graph Analytics
PyTorch, TensorFlow, MxNet Deep Learning
cuxfilter, pyViz, plotly Visualization

GPU Memory

Apache Arrow
RAPIDS

Scaling RAPIDS with Dask

Data Preparation → Model Training → Visualization

Dask

cuDF, cuIO, Analytics

cuML, Machine Learning

cuGraph, Graph Analytics

PyTorch, TensorFlow, MxNet, Deep Learning

cuxfilter, pyViz, plotly, Visualization

GPU Memory

Apache Arrow
Why Dask?

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

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

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)
- Alpha Python bindings for UCX (ucx-py)
- Will provide best communication performance, to Dask based on available hardware on nodes/cluster

```bash
conda install -c conda-forge -c rapidsai \ cudatoolkit=<CUDA version> ucx-proc=*=gpu ucx ucx-py
```
**Benchmarks: Distributed cuDF Random Merge**

**cuDF v0.14, UCX-PY 0.14**

- Running on NVIDIA DGX-2:
  - GPU: NVIDIA Tesla V100 32GB
  - CPU: Intel(R) Xeon(R) CPU 8168 @ 2.70GHz

- Benchmark Setup:
  - DataFrames: Left/Right 1x int64 column key column, 1x int64 value columns
  - Merge: Inner
  - 30% of matching data balanced across each partition
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

**Scale Up / Accelerate**
- RAPIDS

**Scale Out / Parallelize**
- DASK
cuDF
RAPIDS
GPU Accelerated Data Wrangling and Feature Engineering

Data Preparation → Model Training → Visualization

Dask

cuDF, cuIO
Analytics

cuML
Machine Learning

cuGraph
Graph Analytics

PyTorch, TensorFlow, MxNet
Deep Learning

cuxfilter, pyViz, plotly
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 Technology Stack

- Python
- Cython
- cuDF C++
- CUDA Libraries
- CUDA
- Dask cuDF
- cuDF
- Pandas
- Thrust
- Cub
- Jitify
ETL - the Backbone of Data Science

libcuDF is...

CUDA C++ LIBRARY

- Table (dataframe) and column types and algorithms
- CUDA kernels for sorting, join, groupby, reductions, partitioning, elementwise operations, etc.
- Optimized GPU implementations for strings, timestamps, numeric types (more coming)
- Primitives for scalable distributed ETL

```cpp
std::unique_ptr<table> gather(table_view const& input,
                                 column_view const& gather_map, ...)
{
  // return a new table containing
  // rows from input indexed by
  // gather_map
}
```
ETL - the Backbone of Data Science

cuDF is...

- 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 v0.13, Pandas 0.25.3

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

- Benchmark Setup:
  - RMM Pool Allocator Enabled
  - 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

Data Preparation → Model Training → Visualization

Dask

cuDF cuIO
Analytics

cuML Machine Learning

cuGraph Graph Analytics

PyTorch, TensorFlow, MxNet
Deep Learning

cuxfilter, pyViz, plotly
Visualization

GPU Memory

Apache Arrow
ETL - the Backbone of Data Science
String Support

**CURRENT V0.14 STRING SUPPORT**
- Regular Expressions
- Element-wise operations
  - Split, Find, Extract, Cat, Typecasting, etc...
- String GroupBys, Joins, Sorting, etc.
- Categorical columns fully on GPU
- Native String type in libcudf C++

**FUTURE V0.15+ STRING SUPPORT**
- Further performance optimization
- JIT-compiled String UDFs
- Tokenizers
  - SubWord, ngrams

---

![Graph comparing Pandas and Cuda Strings performance for various string operations](image-url)
Extraction is the Cornerstone
cuIO for Faster Data Loading

- Follow Pandas APIs and provide >10x speedup
- CSV Reader, CSV Writer
- Parquet Reader, Parquet Writer
- ORC Reader, ORC Writer
- JSON Reader
- Avro Reader
- GPU Direct Storage integration in progress for bypassing PCIe bottlenecks!
- Key is GPU-accelerating both parsing and decompression

```
import pandas, cudf

time len(pandas.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))
CPU times: user 25.9 s, sys: 3.26 s, total: 29.2 s
Wall time: 29.2 s
2]: 12748986

time len(cudf.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))
CPU times: user 1.59 s, sys: 372 ms, total: 1.96 s
Wall time: 2.12 s
3]: 12748986

tdu -hs data/nyc/yellow_tripdata_2015-01.csv
1.9G data/nyc/yellow_tripdata_2015-01.csv
```

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, TensorFlow, MxNet
Deep Learning

cuxfilter, pyViz, plotly
Visualization

GPU Memory

Apache Arrow
Interoperability for the Win

- Real-world workflows often need to share data between libraries.
- RAPIDS supports device memory sharing between many popular data science and deep learning libraries.
- Keeps data on the GPU—avoids costly copying back and forth to host memory.
- Any library that supports DLPack or __cuda_array_interface__ will allow for sharing of memory buffers between RAPIDS and supported libraries.
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
cuML
Machine Learning
More Models More Problems

Data Preparation → Model Training → Visualization

Dask

cuDF, cuIO, Analytics

cuML, Machine Learning

cuGraph, Graph Analytics

PyTorch, TensorFlow, MxNet, Deep Learning

cuxfilter, pyViz, plotly, Visualization

GPU Memory

Apache Arrow
Problem
Data Sizes Continue to Grow

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

Time Increases
Hours? Days?


Meet Reasonable Speed vs Accuracy Trade-off
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
RAPIDS Matches Common Python APIs

CPU-based Clustering

```
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'f{idx}': X[:, idx]
                      for idx 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)
```
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({'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)
Algorithms

GPU-accelerated Scikit-Learn

- Decision Trees / Random Forests
- Linear/Lasso/Ridge Regression
- Logistic Regression
- K-Nearest Neighbors
- Support Vector Machine Classification
- Random Forest / GBDT Inference
- K-Means
- DBSCAN
- Spectral Clustering
- Principal Components
- Singular Value Decomposition
- UMAP
- Spectral Embedding
- T-SNE
- Holt-Winters
- Seasonal ARIMA

Key:
Preexisting | NEW or enhanced for 0.14

More to come!
Benchmarks: Single-GPU cuML vs Scikit-learn
cuML’s Forest Inference Library accelerates prediction (inference) for random forests and boosted decision trees:

- Works with existing saved models (XGBoost, LightGBM, scikit-learn RF 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
XGBoost + RAPIDS: Better Together

- RAPIDS 0.14 comes paired with XGBoost 1.1
- XGBoost now builds on the GoAI interface standards to provide zero-copy data import from cuDF, cuPY, Numba, PyTorch and more
- Official Dask API makes it easy to scale to multiple nodes or multiple GPUs
- gpu_hist tree builder delivers huge perf gains
  Memory usage when importing GPU data decreased by 2/3 or more
- New objectives support Learning to Rank on GPU

All RAPIDS changes are integrated upstream and provided to all XGBoost users – via pypi or RAPIDS conda

XGBoost speedup on GPUs comparing a single NVIDIA V100 GPU to a dual 20-core Intel Xeon E5-2698 server
RAPIDS Integrated into Cloud ML Frameworks

Accelerated machine learning models in RAPIDS give you the flexibility to use hyperparameter optimization (HPO) experiments to explore all variants to find the most accurate possible model for your problem.

With GPU acceleration, RAPIDS models can train 40x faster than CPU equivalents, enabling more experimentation in less time.

The RAPIDS team works closely with major cloud providers and OSS solution providers to provide code samples to get started with HPO in minutes

https://rapids.ai/hpo
HPO Use Case: 100-Job Random Forest Airline Model

Huge speedups translate into >7x TCO reduction

Based on sample Random Forest training code from cloud-ml-examples repository, running on Azure ML. 10 concurrent workers with 100 total runs, 100M rows, 5-fold cross-validation per run.

GPU nodes: 10x Standard_NC6s_v3, 1 V100 16G, vCPU 6 memory 112G, Xeon E5-2690 v4 (Broadwell) - $3.366/hour
CPU nodes: 10x Standard_DS5_v2, vCPU 16 memory 56G, Xeon E5-2673 v3 (Haswell) or v4 (Broadwell) - $1.017/hour
Road to 1.0 - cuML
June 2020 - RAPIDS 0.14

<table>
<thead>
<tr>
<th>cuML</th>
<th>Single-GPU</th>
<th>Multi-Node-Multi-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Boosted Decision Trees (GBDT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear Regression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logistic Regression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Forest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-NN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBSCAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Holt-Winters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARIMA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T-SNE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principal Components</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Road to 1.0 - cuML

Later in 2020 - RAPIDS 1.0

<table>
<thead>
<tr>
<th>cuML</th>
<th>Single-GPU</th>
<th>Multi-Node-Multi-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Boosted Decision Trees (GBDT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear Regression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logistic Regression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Forest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-NN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBSCAN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Holt-Winters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARIMA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T-SNE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principal Components</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Graph Analytics

More Connections, More Insights

Data Preparation → Model Training → Visualization

Dask

cuDF cuIO
Analytics

cuML
Machine Learning

cuGraph
Graph Analytics

PyTorch,
TensorFlow, MxNet
Deep Learning

cuxfilter, pyViz,
plotly
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
Graph Technology Stack

Python

Cython

cuGraph Algorithms

Prims  cuGraphBLAS*  cuHornet

CUDA Libraries

CUDA

Dask cuGraph
Dask cuDF
cuDF
Numpy

Thrust
Cub
cuSolver
cuSparse
cuRand
Gunrock*

+cuGraphBLAS is still in development and will be ready late 2020

* Gunrock is from UC Davis
**Algorithms**

**GPU-accelerated NetworkX**

<table>
<thead>
<tr>
<th>Community</th>
<th>Spectral Clustering - Balanced Cu and Modularity Maxim Louvain (redone for 0.14) Ensemble Clustering for Graphs KCore and KCore Number Triangle Counting K-Truss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
<td>Weakly Connected Components Strongly Connected Components</td>
</tr>
<tr>
<td>Link Analysis</td>
<td>Page Rank (Multi-GPU) Personal Page Rank Jaccard Weighted Jaccard Overlap Coefficient</td>
</tr>
<tr>
<td>Link Prediction</td>
<td>Single Source Shortest Path (SSSP) Breadth First Search (BFS)</td>
</tr>
<tr>
<td>Traversal</td>
<td>Katz Betweenness Centrality (redone in 0.14)</td>
</tr>
<tr>
<td>Centrality</td>
<td></td>
</tr>
</tbody>
</table>

**Graph Classes**

- Subgraph Extraction

**Structure**

- Force Atlas 2

**Utilities**

- Renumbering
- Auto-Renumbering
- GPU-accelerated NetworkX

**Graph Classes**

- Subgraph Extraction
# Benchmarks: Single-GPU cuGraph vs NetworkX

## Performance Speedup cuGraph vs NetworkX

<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>20,491,458</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>

### Diagram:

- **Louvain**
- **PageRank**
- **BFS**
- **SSSP**

The diagram illustrates the performance speedup of cuGraph compared to NetworkX for different datasets, showing the speedup ratios for various graph algorithms and datasets.
## 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># of CPU Threads</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></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></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></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></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></td>
<td>31.8</td>
</tr>
<tr>
<td>BigData x8</td>
<td>400,000,000</td>
<td>16,000,000,000</td>
<td>300</td>
<td>800*</td>
<td></td>
<td>5760*</td>
</tr>
</tbody>
</table>

*BigData x8, 100x 8-vCPU nodes, Apache Spark GraphX ⇒ 96 mins!"
### Road to 1.0

**December 2019 - RAPIDS 0.14**

<table>
<thead>
<tr>
<th>cuGRAPH</th>
<th>Single-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Rank</td>
<td></td>
</tr>
<tr>
<td>Personal Page Rank</td>
<td></td>
</tr>
<tr>
<td>Katz</td>
<td></td>
</tr>
<tr>
<td>Betweenness Centrality</td>
<td></td>
</tr>
<tr>
<td>Spectral Clustering</td>
<td></td>
</tr>
<tr>
<td>Louvain</td>
<td></td>
</tr>
<tr>
<td>Ensemble Clustering for Graphs</td>
<td></td>
</tr>
<tr>
<td>K-Truss &amp; K-Core</td>
<td></td>
</tr>
<tr>
<td>Connected Components (Weak &amp; Strong)</td>
<td></td>
</tr>
<tr>
<td>Triangle Counting</td>
<td></td>
</tr>
<tr>
<td>Single Source Shortest Path (SSSP)</td>
<td></td>
</tr>
<tr>
<td>Breadth-First Search (BFS)</td>
<td></td>
</tr>
<tr>
<td>Jaccard &amp; Overlap Coefficient</td>
<td></td>
</tr>
<tr>
<td>Force Atlas 2</td>
<td></td>
</tr>
</tbody>
</table>
# Road to 1.0

June 2020 - RAPIDS 1.0

<table>
<thead>
<tr>
<th>cuGRAPH</th>
<th>Single-GPU</th>
<th>Multi-Node-Multi-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Rank</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Personal Page Rank</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Katz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Betweenness Centrality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral Clustering</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Louvain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ensemble Clustering for Graphs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Truss &amp; K-Core</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connected Components (Weak &amp; Strong)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triangle Counting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single Source Shortest Path (SSSP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breadth-First Search (BFS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jaccard &amp; Overlap Coefficient</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Force Atlas 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hungarian Algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leiden</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
cuSpatial Technology Stack
BREAKTHROUGH PERFORMANCE & EASE OF USE
- Up to 1000x faster than CPU spatial libraries
- Python and C++ APIs for maximum usability and integration

GROWING FUNCTIONALITY
- Extensive collection of algorithm, primitive, and utility functions for spatial analytics

SEAMLESS INTEGRATION INTO RAPIDS
- cuDF for data loading, cuGraph for routing optimization, and cuML for clustering are just a few examples
# cuSpatial

## 0.14 and Beyond

<table>
<thead>
<tr>
<th>Layer</th>
<th>0.14 Functionality</th>
<th>Functionality Roadmap (0.15/0.16)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High-level Analytics</strong></td>
<td>C++ Library w. Python bindings enabling distance, speed, trajectory similarity, trajectory clustering</td>
<td>Symmetric segment path distance</td>
</tr>
<tr>
<td><strong>Graph Layer</strong></td>
<td>cuGraph</td>
<td>cuGraph</td>
</tr>
<tr>
<td><strong>Query Layer</strong></td>
<td>Spatial Window</td>
<td>Nearest Neighbor, Spatiotemporal range search and joins</td>
</tr>
<tr>
<td><strong>Index Layer</strong></td>
<td></td>
<td>Quadtree</td>
</tr>
<tr>
<td><strong>Geo-operations</strong></td>
<td>Point in polygon (PIP), Haversine distance, Hausdorff distance, lat-lon to xy transformation</td>
<td>ST_distance and ST_contains</td>
</tr>
<tr>
<td><strong>Geo-Representation</strong></td>
<td>Shape primitives, points, polylines, polygons</td>
<td>Fiona/Geopandas I/O support and object representations</td>
</tr>
</tbody>
</table>
## cuSpatial

### Performance at a Glance

<table>
<thead>
<tr>
<th>cuSpatial Operation</th>
<th>Input Data</th>
<th>cuSpatial Runtime</th>
<th>Reference Runtime</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Point-in-Polygon Test</strong></td>
<td>1.3+ million vehicle point locations and 27 Region of Interests</td>
<td>1.11 ms (C++) 1.50 ms (Python)</td>
<td>334 ms (C++, optimized serial) 130468.2 ms (python Shapely API, serial) [Nvidia Titan V] [Intel i7-7800X]</td>
<td>301X (C++) 86,978X (Python)</td>
</tr>
<tr>
<td><strong>Haversine Distance Computation</strong></td>
<td>13+ million Monthly NYC taxi trip pickup and drop-off locations</td>
<td>7.61 ms (Python) [Nvidia T4]</td>
<td>416.9 ms (Numba) [Nvidia T4]</td>
<td>54.7X (Python)</td>
</tr>
<tr>
<td><strong>Hausdorff Distance Computation (for clustering)</strong></td>
<td>52,800 trajectories with 1.3+ million points</td>
<td>13.5s [Quadro V100]</td>
<td>19227.5s (Python SciPy API, serial) [Intel i7-6700K]</td>
<td>1,400X (Python)</td>
</tr>
</tbody>
</table>
cuSignal
Unlike other RAPIDS libraries, cuSignal is purely developed in Python with custom CUDA Kernels written with Numba and CuPy (notice no Cython layer).
cuSignal - Selected Algorithms

GPU-accelerated SciPy Signal

- Convolution
  - Convolve/Correlate
  - FFT Convolve
  - Convolve/Correlate 2D
  - Resampling - Polyphase, Upfirdn, Resample
  - Hilbert/Hilbert 2D
  - Wiener
  - Firwin
- Filtering and Filter Design
- Waveform Generation
- Window Functions
- Spectral Analysis
- Wavelets
- Peak Finding
- More to come!
Speed of Light Performance - V100

*timeit* (7 runs) rather than *time*. Benchmarked with ~1e8 sample signals on a DGX Station

<table>
<thead>
<tr>
<th>Method</th>
<th>Scipy Signal (ms)</th>
<th>cuSignal (ms)</th>
<th>Speedup (xN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fftconvolve</td>
<td>27300</td>
<td>85.1</td>
<td>320.8</td>
</tr>
<tr>
<td>correlate</td>
<td>4020</td>
<td>47.4</td>
<td>84.8</td>
</tr>
<tr>
<td>resample</td>
<td>14700</td>
<td>45.9</td>
<td>320.2</td>
</tr>
<tr>
<td>resample_poly</td>
<td>2360</td>
<td>8.29</td>
<td>284.6</td>
</tr>
<tr>
<td>welch</td>
<td>4870</td>
<td>45.5</td>
<td>107.0</td>
</tr>
<tr>
<td>spectrogram</td>
<td>2520</td>
<td>23.3</td>
<td>108.1</td>
</tr>
<tr>
<td>convolve2d</td>
<td>8410</td>
<td>9.92</td>
<td>847.7</td>
</tr>
</tbody>
</table>

Learn more about cuSignal functionality and performance by browsing the [notebooks](#).
Seamless Data Handoff from cuSignal to PyTorch >=1.4
Leveraging the __cuda_array_interface__ for Speed of Light End-to-End Performance

```
from numba import cuda
import cupy as cp
import torch
from cusignal import resample_poly

# Create CuPy Array on GPU
gpu_arr = cp.random.randn(100_000_000, dtype=cp.float32)

# Polyphase Resample
resamp_arr = resample_poly(gpu_arr, up=2, down=3, window=('kaiser', 0.5))

# Zero Copy to PyTorch
torch_arr = torch.as_tensor(resamp_arr, device='cuda')

# Confirm Pointers
print('Resample Array: ', resamp_arr.__cuda_array_interface__['data'])
print('Torch Array: ', torch_arr.__cuda_array_interface__['data'])

Resample Array: (140516096213000, False)
Torch Array: (140516096213000, False)
```

Enabling Online Signal Processing with Zero-Copy Memory
CPU <-> GPU Direct Memory Access with Numba’s Mapped Array

```
import numpy as np
import cupy as cp
from numba import cuda

# Create CPU/GPU Shared Memory, similar to numpy.zeros()
N = 2**18
shared_arr = cusignal.get_shared_mem(N, dtype=np.complex64)
print('CPU Pointer: ', shared_arr.__cuda_array_interface__['data'])

CPU Pointer: (1398985170817640, False)

# Load Shared Array with Numpy Array
shared_arr = np.random.randn(N) + 1j*np.random.randn(N)

# Perform CPU FFT
cpu_fft = np.fft.fft(shared_arr)
8 ms ± 80.2 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

# Perform GPU FFT

gpu_fft = cp.fft.fft(shared_arr)
cp.cuda.Device(0).synchronize()
866 µs ± 38 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```
Cyber Log Accelerators
CLX
Cyber Log Accelerators

Built using RAPIDS and GPU-accelerated platforms

Targeted towards Senior SOC (Security Operations Center) Analysts, InfoSec Data Scientists, Threat Hunters, and Forensic Investigators

*Notebooks* geared towards info sec and cybersecurity data scientists and data engineers

*SIEM integrations* that enable easy data import/export and data access

Workflow and I/O *components* that enable users to instantiate new use cases while

Cyber-specific *primitives* that provide accelerated functions across cyber data types
CLX Technology Stack

CLX Applications / Use Cases

Python

Cython

RAPIDS

GPU Packages

CUDA Libraries

CUDA

Security Products and SIEMs
CLX Contains Various Use Cases and Connectors

Example Notebooks Demonstrate RAPIDS for Cybersecurity Applications

<table>
<thead>
<tr>
<th>CLX</th>
<th>Type</th>
<th>Proof-of-Concept</th>
<th>Stable</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGA Detection</td>
<td>Use Case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network Mapping</td>
<td>Use Case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asset Classification</td>
<td>Use Case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phishing Detection</td>
<td>Use Case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Alert Analysis</td>
<td>Use Case</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Splunk Integration</td>
<td>Integration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLX Query</td>
<td>Integration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cyBERT</td>
<td>Log Parsing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU Wordpiece Tokenizer</td>
<td>Pre-Processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accelerated IPv4</td>
<td>Primitive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accelerated DNS</td>
<td>Primitive</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
cyBERT
AI Log Parsing for Known, Unknown, and Degraded Cyber Logs

Provide a flexible method that does not use heuristics/regex to parse cybersecurity logs

Built using RAPIDS and PyTorch, tested with a variety of language models (including BERT)

Parsing with micro-F1 and macro-F1 > 0.999 across heterogeneous log types with a validation loss of < 0.0048

Second version ~160x (min) faster than first version due to creation of the first all-GPU wordpiece tokenizer that supports non-truncation of logs/sentences
GPU SubWord Tokenizer
Fully On-GPU Pre-Processing for BERT Training/Inference

Only wordpiece tokenizer that supports non-truncation of logs/sentences

Returns encoded tensor, attention mask, and metadata to reform broken logs

Supports stride/overlap

Ready for immediate pipelining into PyTorch for inference

Up to 270x faster than production-ready Hugging Face wordpiece tokenizer (Python version)

SubWord Tokenizer Speedup Comparison

CPU numbers ran on 2x Intel Xeon E5 v4 @ 2.2 GHz. GPU numbers ran on 1x NVIDIA Tesla V100
CLX Query
Query Long-Term Data Store Directly from Splunk

```
| cli query="SELECT domain, count(*) as cnt FROM main.dga GROUP BY domain ORDER BY cnt DESC LIMIT 10000"| table domain, cnt |
```

```
| cli query="SELECT domain, count(*) as cnt FROM main.dga GROUP BY domain ORDER BY cnt DESC LIMIT 10000"| table domain, cnt |
```

10,000 results (10/29/19 1:00:00.000 PM to 10/30/19 1:31:21.000 PM)  No Event Sampling  

New Search

```
| cli query="SELECT domain, count(*) as cnt FROM main.dga GROUP BY domain ORDER BY cnt DESC LIMIT 10000"| table domain, cnt |
```

10,000 results (10/29/19 1:00:00.000 PM to 10/30/19 1:31:21.000 PM)  No Event Sampling  

New Search
Visualization
cuXfilter
GPU Accelerated Cross-Filtering

STREAMLINED FOR NOTEBOOKS

Cuxfilter allows you to visually explore your cuDF dataframes through fully cross-filtered dashboards in less than 10 lines of notebook code.

MINIMAL COMPLEXITY & FAST RESULTS

Select from vetted chart libraries, pre-designed layouts, and multiple themes to find the best fit for a use case, at speeds typically 20x faster per chart than Pandas.

SEAMLESS INTEGRATION WITH RAPIDS

Cuxfilter is designed to be easy to install and use with RAPIDS. Learn more about our approach here.

https://docs.rapids.ai/api/cuxfilter/stable
pyViz Integrations
GPU Accelerated Python Visualizations

ACCELERATED PYTHON VIZ
The RAPIDS focus on python makes it straightforward to integrate GPU acceleration for most python visualization libraries. We are currently collaborating with bokeh, plotly, and datashader, among others.

HIGHLIGHT: DATASHADER INTEGRATION
Datashader's ability to visualize billions of points makes it well suited for GPU acceleration. It now includes cuDF integration as well as other enhanced capabilities that offer between a 10-120x speed up. More details here.

https://datashader.org/
Plotly Dash
GPU Accelerated Visualization Apps with Python

PLOTLY & RAPIDS PARTNERSHIP
Plotly Dash now has RAPIDS acceleration, enabling the ease-of-use development and deployment of Dash apps, with the speed of RAPIDS. Find out more on RAPIDS.ai and Plotly’s announcement.

300 MILLION DATAPoint CENSUS EXAMPLE
Interact with data points of every individual in the United States, in real time, with the 2010 Census visualization. Get the code on GitHub and read about details here.

https://github.com/rapidsai/plotly-dash-rapids-census-demo
Community
Ecosystem Partners

CONTRIBUTORS
- Anaconda
- CapitalOne
- CuPy
- Chainer
- UNROCK
- Quansight
- Walmart

ADOPTERS
- Booz Allen Hamilton
- CapitalOne
- Databricks
- Graphistry
- H2O.ai
- IBM
- Iguzio
- Inria

- Kinectica
- MAPR
- Omniscale
- Preferred Networks
- PyTorch
- Uber
- Ursa Labs
- Walmart

OPEN SOURCE
- Allen
- CapitalOne
- CuPy
- Dask
- GAl
- Nuclio
- Numba
- learn
- XGBoost
Building on Top of RAPIDS
A Bigger, Better, Stronger Ecosystem for All

NVTabular
ETL library for recommender systems building off of RAPIDS and Dask

blazingSQL
GPU accelerated SQL engine built on top of RAPIDS

Streamz
Distributed stream processing using RAPIDS and Dask
NVTabular
ETL library for recommender systems building off of RAPIDS and Dask

A HIGH LEVEL API BUILDING UPON DASK-CUDF
NVTabular’s high level API allows users to think about what they want to do with the data, not how they have to do it or how to scale it, for operations common within recommendation workflows.

ACCELERATED GPU DATALOADERS
Using cuIO primatives and cuDF, NVTabular accelerates dataloading for PyTorch & Tensorflow, removing I/O issues common in deep learning based recommender system models.
BlazingSQL
GPU-accelerated SQL engine built with RAPIDS

BLAZING FAST SQL ON RAPIDS
- Incredibly fast distributed SQL engine on GPUs—natively compatible with RAPIDS!
- Allows data scientists to easily connect large-scale data lakes to GPU-accelerated analytics
- Directly query raw file formats such as CSV and Apache Parquet inside Data Lakes like HDFS and AWS S3, and directly pipe the results into GPU memory.

NEW FOR 0.14: OUT OF CORE EXECUTION
- Users no longer limited by available GPU memory
- 10TB workloads on a single Tesla V100 (32GB)!
from blazingsql import BlazingContext
import cudf

bc = BlazingContext()

bc.s('bsql', bucket_name='bsql', access_key_id='<access_key>', secret_key='<secret_key>')

bc.create_table('orders', s3://bsql/orders/)

gdf = bc.sql('select * from orders').get()
cuStreamz
Stream processing powered by RAPIDS

ACCELERATED KAFKA CONSUMER
Ingesting Kafka messages to cudf is increased by roughly 3.5 - 4X over standard cpu ingestion. Streaming TCO is lowered by using cheaper VM instance types.

CHECKPOINTING
Streaming job version of “where was I?” Streams can gracefully handle errors by continuing to process a stream at the exact point they were before the error occurred.

Kafka throughput
Single Node

- **No Checkpointing**
  - streamz: 134
  - cuStreamz: 489

- **With Checkpointing**
  - streamz: 96
  - cuStreamz: 396

megabytes
Join the Conversation

GOOGLE GROUPS
https://groups.google.com/forum/#!forum/rapidsai

DOCKER HUB
https://hub.docker.com/r/rapidsai/rapidsai

SLACK CHANNEL
https://rapids-goai.slack.com/join

STACK OVERFLOW
https://stackoverflow.com/tags/rapids
**Contribute Back**

Issues, Feature Requests, PRs, Blogs, Tutorials, Videos, QA...Bring Your Best!

### cuml
**cuML - RAPIDS Machine Learning Library**
- machine-learning
- gpu
- machine-learning-algorithms
- coda
- nvidia

| C++ | Apache-2.0 | 111 | 608 | 106 (25 issues need help) | 31 | Updated 9 minutes ago |

### cudf
**cudF - GPU DataFrame Library**
- anaconda
- gpu
- arrow
- machine-learning-algorithms
- h2o
- coda
- pandas

| Cuda | Apache-2.0 | 250 | 1,699 | 325 (6 issues need help) | 41 | Updated 31 minutes ago |

### notebooks-contrib
**RAPIDS Community Notebooks**
- Jupyter Notebook
- Apache-2.0
- 56
- 70
- 10 (1 issue needs help) | 8 | Updated 40 minutes ago

---

**Getting Started with cuDF (RAPIDS)**

---

**Comparison CPU vs GPU @rapidsai to project 100 million x,y points to lat/lon to 0.01mm accuracy. CPU 1 core c 65 mins, multicore c 13 mins, GPU #RAPIDSAI 2 seconds. I optimised the code since previous run. Dell T7910 Xeon E5-2640v4x2/NVIDIA Titan Xp cc @NvidiaAI @marc_stampfli**

---

**How GPU Computing literally saved me at work?**

---

**Darren Ramsook**

Jun 9 - 3 min read
Getting Started
5 Steps to Getting Started with RAPIDS

1. Install RAPIDS on using Docker, Conda, or Colab.

2. Explore our walk through videos, blog content, our github, the tutorial notebooks, and our example workflows.

3. Build your own data science workflows.

4. Join our community conversations on Slack, Google, and Twitter.

5. Contribute back. Don’t forget to ask and answer questions on Stack Overflow.
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.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>Preferred</th>
<th>Advanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conda</td>
<td>Docker + Examples</td>
<td>Docker + Dev Env</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RELEASE</th>
<th>Stable (0.14)</th>
<th>Nightly (0.15a)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PACKAGES</th>
<th>cuDF</th>
<th>cuML</th>
<th>cuGraph</th>
<th>cuSignal</th>
<th>cuSpatial</th>
<th>cuFilter</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>LINUX</th>
<th>Ubuntu 16.04</th>
<th>Ubuntu 18.04</th>
<th>CentOS 7</th>
<th>RHEL 7</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>PYTHON</th>
<th>Python 3.6</th>
<th>CUDA 10.0</th>
<th>CUDA 10.1.2</th>
<th>CUDA 10.2</th>
</tr>
</thead>
</table>

**NOTE:** Ubuntu 16.04/18.04 & CentOS 7 use the same `conda install` commands.

**COMMAND**

```
conda install -c rapidsai -c nvidia -cconda-forge 
-c defaults rapids=0.14 python=3.6
```

[https://rapids.ai/start.html](https://rapids.ai/start.html)
RAPIDS Docs
New, Improved, and Easier to Use

https://docs.rapids.ai
RAPIDS Docs

Easier than Ever to Get Started with cuDF

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
Explore: RAPIDS Github

https://github.com/rapidsai
### 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>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>Dash with cuDF and XGBoost</td>
<td>In this notebook we show how to quickly setup Dash and train an XGBoost model using cuDF.</td>
</tr>
</tbody>
</table>

RAPIDS Youtube channel:
- **Introduction to XGBoost**: 114 views - 1 month ago
- **Introduction to RAPIDS**: 533 views - 1 month ago
- **Introduction to Dash XGBoost**: 94 views - 1 month ago
- **Accelerated with RAI Walmart uses RAPID**: 402 views - 1 month ago
RAPIDS
How Do I Get the Software?

GITHUB
https://github.com/rapidsai

ANAconda
https://anaconda.org/rapidsai/

NGC

DOCKER
https://hub.docker.com/r/rapidsai/rapidsai/
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
Join the Movement
Everyone Can Help!

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

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
THANK YOU

Joshua Patterson
joshuap@nvidia.com
@datametrician