RAPIDS
The Platform Inside and Out
Release 0.15

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
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

HDFS Read  | Query  | HDFS Write | HDFS Read  | ETL  | HDFS Write | HDFS Read  | ML Train
--- | --- | --- | --- | --- | --- | --- | ---

HDFS Read  | Query  | ETL  | ML Train
--- | --- | --- | ---

HDFS Read  | GPU Read  | Query  | CPU Write  | GPU Read  | ETL  | CPU Write  | GPU Read  | ML Train
--- | --- | --- | --- | --- | --- | --- | --- | ---
Data Movement and Transformation
The Bane of Productivity and Performance

Read Data

Copy & Convert

Copy & Convert

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)

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
Lightning-fast performance on real-world use cases
Up to 350x faster queries; Hours to Seconds!

TPCx-BB is a data science benchmark consisting of 30 end-to-end queries representing real-world ETL and Machine Learning workflows, involving both structured and unstructured data. It can be run at multiple “Scale Factors”.

- SF1 - 1GB
- SF1K - 1 TB
- SF10K - 10 TB

RAPIDS results at SF1K (2 DGX A100s) and SF10K (16 DGX A100s) show GPUs provide dramatic cost and time-savings for small scale and large-scale data analytics problems

- SF1K 37.1x average speed-up
- SF10K 19.5x average speed-up (7x Normalized for Cost)
Speed, UX, and Iteration
The Way to Win at Data Science

Francois Chollet: @fchollet 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.

Francois Chollet: @fchollet Apr 3
Because good UX reduces the 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.

Francois Chollet: @fchollet 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 @fchollet
This is the fundamental belief that drives @RAPIDSInc. @nvidia's 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 0.15 Release Summary
What’s New in Release 0.15?

- **CUDA 11 and Ampere** support added across all libraries.
- **cuDF** adds unsigned integer and timedelta (duration) dtypes support as well as experimental support for a List dtype, Kafka as a supported data source, Groupby.rolling() support, and additional NLP preprocessing functions.
- **cuML** machine learning library adds scikit-learn compatible NLP preprocessors, initial AutoARIMA support, MNMG kNN classification and regression wrappers, accelerated multi-class forest inference, SVM sample weights and probability prediction, and an experimental Incremental PCA model for out-of-core fitting.
- **cuGraph** initial version of the Leiden algorithm, added Edge Betweenness Centrality, improved Louvain and the ECG analytics, and added the HITS algorithm.
- **UCX-Py** adds reusable endpoints, Cython optimizations, code clean up, and bug fixes.
- **BlazingSQL** now supports out-of-core query execution, which enables queries to operate on datasets dramatically larger than available GPU memory.
- **cuSignal** adds multi-point kalman filter, improved embedded GPU support, and performance optimizations.
- **cuSpatial** adds point-to-nearest polyline queries and accelerated point-in-polygon via quadtree.
- **cuxfilter** will have a delayed release, but now has large graph support. See Notice RGN4 in our docs for details.
- **RMM** deprecates the CNMeM, adds per-thread default stream support, adds `binning_memory_resource`.
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.
Open Source Data Science Ecosystem

Familiar Python APIs

- Pandas Analytics
- Scikit-Learn Machine Learning
- NetworkX Graph Analytics
- PyTorch, TensorFlow, MxNet Deep Learning
- Matplotlib Visualization

Dask

Data Preparation → Model Training → 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, TensorFlow, MxNet Deep Learning
-> cuxfilter, pyViz, plotly Visualization

Apache Arrow
GPU Memory
RAPIDS
Scaling RAPIDS with Dask

Data Preparation → Model Training → Visualization

Dask

cuDF, cuIO, cuML, cuGraph, PyTorch, TensorFlow, MxNet, cuxfilter, pyViz, plotly

Apache Arrow → GPU Memory
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

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

RAPIDS + DASK WITH OPENUCX
Multi-GPU
On single Node (DGX)
Or across a cluster

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

Scale Up / Accelerate

Scale Out / Parallelize
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

Apache Arrow

GPU Memory
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...

**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

```python
In [2]: # Read in the data. Notice how it decompresses as it reads the data into memory.
gdf = cudf.read_csv('/opt/ibm/rapids/datasets/black-friday.csv')

In [3]: # Taking a look at the data. We use "to_pandas()" to get the pretty printing.
gdf.head().to_pandas()

Out[3]:

<table>
<thead>
<tr>
<th>User_ID</th>
<th>Product_ID</th>
<th>Gender</th>
<th>Age</th>
<th>Occupation</th>
<th>City_Category</th>
<th>Stay_In_Current_City_Years</th>
<th>Marital_Status</th>
<th>Product_Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10000001</td>
<td>F</td>
<td>0-17</td>
<td>A</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10000001</td>
<td>P</td>
<td>0-17</td>
<td>A</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10000001</td>
<td>P</td>
<td>0-17</td>
<td>A</td>
<td>2</td>
<td>0</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10000001</td>
<td>P</td>
<td>0-17</td>
<td>A</td>
<td>2</td>
<td>0</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10000002</td>
<td>M</td>
<td>55+</td>
<td>C</td>
<td>16</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

In [6]: # Grabbing the first character of the years in city string to get rid of plus sign, and converting to int
gdf["city_years"] = gdf.Stay_In_Current_City_Years.str.get(0).str.toi()

In [7]: # Here we can see how we can control what the value of our dummies with the replace method and turn strings to int

gdf["City_Category"] = gdf.City_Category.str.replace('A', '1')
gdf["City_Category"] = gdf.City_Category.str.replace('B', '2')
gdf["City_Category"] = gdf.City_Category.str.replace('C', '3')
gdf["City_Category"] = gdf["City_Category"].str.stoi()
```
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

Apache Arrow → GPU Memory
ETL - the Backbone of Data Science

String Support

CURRENT V0.15 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++
- NLP Preprocessors
  - Tokenizers, Normalizers, Edit Distance, Porter Stemmer, etc.

FUTURE V0.16+ STRING SUPPORT
- Further performance optimization
- JIT-compiled String UDFs
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

```python
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
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
12748986
du -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, cuML, cuGraph
Analytics, Machine Learning, Graph Analytics

PyTorch, TensorFlow, MXNet
Deep Learning

cuxfilter, pyViz, plotly
Visualization

Apache Arrow
GPU Memory
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

![Graph showing SVD Benchmark results with Dask and CuPy.]
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.

Meet Reasonable Speed vs Accuracy Trade-off

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({'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)
```
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/ElasticNet Regression
Logistic Regression
K-Nearest Neighbors
Support Vector Machine Classification and Regression
Naive Bayes
Random Forest / GBDT Inference (FIL)

Text vectorization (TF-IDF / Count)
Target Encoding
Cross-validation / splitting

K-Means
DBSCAN
Spectral Clustering
Principal Components
Singular Value Decomposition
UMAP
Spectral Embedding
T-SNE

Holt-Winters
Seasonal ARIMA / Auto ARIMA

Key:
Preexisting | NEW or enhanced for 0.15

More to come!

Cross Validation
Hyper-parameter Tuning

Inference
Preprocessing

Classification / Regression

Clustering
Decomposition & Dimensionality Reduction

Time Series
Benchmarks: Single-GPU cuML vs Scikit-learn (WIP)

1x V100 vs. 2x 20 Core CPUs (DGX-1, RAPIDS 0.15)
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 comes paired with XGBoost 1.1 (as of 0.14)
- 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 0.15**

<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

### 2020 EOY Plan

<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
Spectral Clustering - Balanced Cu and Modularity Maxim
Louvain and Leiden
Ensemble Clustering for Graphs
KCore and KCore Number
Triangle Counting
K-Truss

Community

Weakly Connected Components
Strongly Connected Components

Components

Page Rank (Multi-GPU)
Personal Page Rank
HITS

Link Analysis

Jaccard
Weighted Jaccard
Overlap Coefficient

Link Prediction

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

Traversal

Katz
Betweenness Centrality (Vertex and Edge)

Centrality

Algorithms
GPU-accelerated NetworkX

Graph Classes
Subgraph Extraction

Structure

Re-numbering
Auto-Re-numbering
Force Atlas 2

Utilities

Community
Benchmarks: Single-GPU 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>Dblp-2010</td>
<td>326,186</td>
<td>1,615,400</td>
</tr>
<tr>
<td>coPapersCiteseeer</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>
## 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></td>
<td>800*</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 0.15**

<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>
<tr>
<td>Leiden</td>
<td></td>
</tr>
</tbody>
</table>
## Road to 1.0

### 2020 EOY Plan

<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
cuSpatial Technology Stack

- Python
- Cython
- cuSpatial
- cuDF C++
- Thrust
- CUDA
cuSpatial

**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
### 0.15 and Beyond

<table>
<thead>
<tr>
<th>Layer</th>
<th>0.15 Functionality</th>
<th>Functionality Roadmap (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, Nearest polyline</td>
<td>Nearest Neighbor, Spatiotemporal range search and joins</td>
</tr>
<tr>
<td><strong>Index Layer</strong></td>
<td>Quadtree</td>
<td></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>Point-in-Polygon Test</td>
<td>1.3+ million vehicle point locations and 27 Region of Interests</td>
<td>1.11 ms (C++)</td>
<td>334 ms (C++, optimized serial)</td>
<td>301X (C++)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.50 ms (Python)</td>
<td>130468.2 ms (python Shapely API, serial)</td>
<td>86,978X (Python)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Nvidia Titan V]</td>
<td>[Intel i7-7800X]</td>
<td></td>
</tr>
<tr>
<td>Haversine Distance Computation</td>
<td>13+ million Monthly NYC taxi trip pickup and drop-off locations</td>
<td>7.61 ms (Python)</td>
<td>416.9 ms (Numba)</td>
<td>54.7X (Python)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Nvidia T4]</td>
<td>[Nvidia T4]</td>
<td></td>
</tr>
<tr>
<td>Hausdorff Distance Computation</td>
<td>52,800 trajectories with 1.3+ million points</td>
<td>13.5s</td>
<td>19227.5s (Python SciPy API, serial)</td>
<td>1,400X (Python)</td>
</tr>
<tr>
<td>(for clustering)</td>
<td></td>
<td>[Quadro V100]</td>
<td>[Intel i7-6700K]</td>
<td></td>
</tr>
</tbody>
</table>
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

Filtering and Filter Design
- Resampling - Polyphase, Upfirdn, Resample
- Hilbert/Hilbert 2D
- Wiener
- Firwin

Waveform Generation
- Chirp
- Square
- Gaussian Pulse

Window Functions
- Kaiser
- Blackman
- Hamming
- Hanning

Spectral Analysis
- Periodogram
- Welch
- Spectrogram

More to come!

Peak Finding

Wavelets
## 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

```python
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

```python
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.__array_interface__['data'])
print('GPU Pointer: ', shared_arr.__cuda_array_interface__['data'])

CPU Pointer: (198090170837448, False)
GPU Pointer: (198090170837640, False)

# Load Shared Array with Numpy Array
shared_arr = np.random.randn(N) + 3j*np.random.randn(N)

# Perform CPU FFT
cpu_fft = np.fft.fft(shared_arr)

# Perform GPU FFT
gpu_fft = cp.fft.fft(cp.asarray(shared_arr))
cp.cuda.Device().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 engineer
- 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

Log Parsing with Inserted Values

Log Parsing with Missing Values
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
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
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 & 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
blazingSQL
CapitalOne
CuPy
Chainer
GUNROK
QUANSIGHT
Walmart

ADOPTERS
Booz Allen Hamilton
CapitalOne
databricks
graphistry
H2O.ai
IBM
iguazio
Inria
kinetica
MAPR
omnisci
Preferred Networks
PyTorch
Saturn Cloud
Uber
URSA Labs
Walmart

OPEN SOURCE
ARROW
blazingSQL
CuPy
DASK
GOAI
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 primitives 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.

NOW WITH 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.
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**
CumML - RAPIDS Machine Learning Library

- Machine-learning
- Cuda
- Nvidia

**cudf**
CuDF - GPU DataFrame Library

- Anaconda
- Arrow
- Cuda
- Pandas

**notebooks-contrib**
RAPIDS Community Notebooks

- Jupyter Notebook
- Apache

---

**Getting Started with cuDF (RAPIDS)**

- John Murray
- @MurrayData

Comparison CPU vs GPU @rapidsai to project 100 million x,y points to lat/lon to 0.01nm 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
# RAPIDS Notices

Communicate and document changes to RAPIDS for contributors, core developers, users, and the community

![Table](https://docs.rapids.ai/notices)

## Notice Details

<table>
<thead>
<tr>
<th>Notice</th>
<th>Title</th>
<th>Topic</th>
<th>RAPIDS Version</th>
<th>Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDN 1</td>
<td>'dask-xgboost' is deprecated in v0.15 &amp; removed v0.16</td>
<td>Library Deprecation</td>
<td>v0.15</td>
<td>26 August 2020</td>
</tr>
<tr>
<td>RGN 1</td>
<td>Stable/Release Branch Renaming to 'main' in v0.15</td>
<td>Git Repo Change</td>
<td>v0.15</td>
<td>26 August 2020</td>
</tr>
<tr>
<td>RGN 2</td>
<td>v0.15 No CUDA 11 Release for 'ctx'</td>
<td>Release Change</td>
<td>v0.15</td>
<td>26 August 2020</td>
</tr>
<tr>
<td>RGN 4</td>
<td>v0.15 Release Delay for 'cuxfilter'</td>
<td>Release Change</td>
<td>v0.15</td>
<td>26 August 2020</td>
</tr>
<tr>
<td>RN 2</td>
<td>EOL Python 3.6 &amp; CUDA 10.0 in v0.14</td>
<td>Platform Support Change</td>
<td>v0.14 &amp; v0.15</td>
<td>13 July 2020</td>
</tr>
<tr>
<td>RN 3</td>
<td>Support for Python 3.8 in v0.15</td>
<td>Platform Support Change</td>
<td>v0.15</td>
<td>17 July 2020</td>
</tr>
<tr>
<td>RN 4</td>
<td>Support for CUDA 11.0 in v0.15</td>
<td>Platform Support Change</td>
<td>v0.15</td>
<td>26 August 2020</td>
</tr>
</tbody>
</table>

[https://docs.rapids.ai/notices](https://docs.rapids.ai/notices)
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.

NOTICES
- RAPIDS repos will rename stable/release branches in v0.15
- Python 3.6 & CUDA 10.0 EOL in v0.14
- Release changes to ckl and cufilter in v0.15
- RAPIDS dask-xgboost library is deprecated in v0.15

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

RELEASE
- Legacy (0.14)
- Stable (0.15)
- Nightly (0.16a)

PACKAGES
- All Packages
- cuDF
- cuML
- cuGraph
- cuSignal
- cuSpatial
- cufilter

LINUX
- Ubuntu 16.04
- Ubuntu 18.04
- CentOS 7
- RHEL 7

PYTHON
- Python 3.6 (0.14 only)
- Python 3.7
- Python 3.8 (0.15/0.16 only)

CUDA
- CUDA 10.0 (0.14 only)
- CUDA 10.2
- CUDA 11.0 (0.15/0.16 only)

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

COMMAND
conda install --rapidsai --nvidia --conda-forge
--defaults rapids=0.15 python=3.7

https://rapids.ai/start.html
RAPIDS Docs
Up to Date and Easy 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

<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>Dark_with_cuDF_and_XGBoost</td>
<td>In this notebook we show how to quickly setup Dark and train an XGBoost model using cuDF.</td>
</tr>
</tbody>
</table>

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