RAPIDDS

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

HDFS Read

Query

HDFS Read

ETL

HDFS Read

ML Train

HDFS Read

Query

ETL

ML Train

HDFS Read

Query

ETL

ML Train

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

APP A

Data Movement Diagram:
- Read Data from CPU to GPU
- Copy & Convert
- Load Data from GPU to CPU
- Copy & Convert

GPU

APP B

CPU

APP A

APP B

Copy & Convert

Read Data

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 (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

<table>
<thead>
<tr>
<th>HDFS Read</th>
<th>Query</th>
<th>HDFS Write</th>
<th>HDFS Read</th>
<th>ETL</th>
<th>HDFS Write</th>
<th>HDFS Read</th>
<th>ML Train</th>
</tr>
</thead>
</table>

Spark In-Memory Processing

<table>
<thead>
<tr>
<th>HDFS Read</th>
<th>Query</th>
<th>ETL</th>
<th>ML Train</th>
</tr>
</thead>
</table>

Traditional GPU Processing

<table>
<thead>
<tr>
<th>HDFS Read</th>
<th>GPU Read</th>
<th>Query</th>
<th>CPU Write</th>
<th>GPU Read</th>
<th>ETL</th>
<th>CPU Write</th>
<th>GPU Read</th>
<th>ML Train</th>
</tr>
</thead>
</table>

RAPIDS

<table>
<thead>
<tr>
<th>Arrow Read</th>
<th>Query</th>
<th>ETL</th>
<th>ML Train</th>
</tr>
</thead>
</table>

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
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
Faster Speeds, Real-World Benefits

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

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

### DGX Cluster Configuration
- 5x DGX-1 on InfiniBand network

---

**Time in seconds (shorter is better)**

- **cuIO/cuDF (Load and Data Prep)**
- **Data Conversion**
- **XGBoost**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 CPU Nodes</td>
<td></td>
</tr>
<tr>
<td>30 CPU Nodes</td>
<td></td>
</tr>
<tr>
<td>50 CPU Nodes</td>
<td></td>
</tr>
<tr>
<td>100 CPU Nodes</td>
<td></td>
</tr>
<tr>
<td>DGX-2</td>
<td></td>
</tr>
<tr>
<td>5x DGX-1</td>
<td></td>
</tr>
<tr>
<td>cuIO/cuDF - Load and Data Preparation</td>
<td>2741</td>
</tr>
<tr>
<td>Data Conversion</td>
<td>1675</td>
</tr>
<tr>
<td>XGBoost</td>
<td>715</td>
</tr>
<tr>
<td>CPU Cluster</td>
<td>3925</td>
</tr>
<tr>
<td>DGX Cluster</td>
<td>213</td>
</tr>
<tr>
<td>End-to-End</td>
<td>8762</td>
</tr>
</tbody>
</table>
Faster Speeds, Real-World Benefits
Even Better with A100 and RAPIDS 0.17

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

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGX-2 RAPIDS v0.2</td>
<td>42</td>
</tr>
<tr>
<td>DGX-2 RAPIDS v0.10</td>
<td>37</td>
</tr>
<tr>
<td>16x A100 RAPIDS v0.17</td>
<td>32</td>
</tr>
</tbody>
</table>

**XGBoost Machine Learning**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGX-2 RAPIDS v0.2</td>
<td>169</td>
</tr>
<tr>
<td>DGX-2 RAPIDS v0.10</td>
<td>147</td>
</tr>
<tr>
<td>16x A100 RAPIDS v0.17</td>
<td>75</td>
</tr>
</tbody>
</table>

**End-to-End**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGX-2 RAPIDS v0.2</td>
<td>322</td>
</tr>
<tr>
<td>DGX-2 RAPIDS v0.10</td>
<td>209</td>
</tr>
<tr>
<td>16x A100 RAPIDS v0.17</td>
<td>122</td>
</tr>
</tbody>
</table>

Time in seconds (shorter is better)
- **cuIO/cuDF (Load and Data Prep)**
- **Data Conversion**
- **XGBoost**

**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

**A100 Cluster Configuration**
16 A100 GPUs (40GB each)
Lightning-fast performance on real-world use cases

GPU Big Data Benchmark (GPU-BDB) is a data science benchmark derived from TPCx-BB\(^1\), consisting of 30 end-to-end queries representing real-world ETL and Machine Learning workflows. It involves both structured and unstructured data. The benchmark starts with reading data from disk, performs common analytical and ML techniques (including NLP), then writes results back to disk to simulate a real world workflow.

Results at 10TB scale show RAPIDS’ performance increasing over time, while TCO continues to go down. The recently announced DGX-A100 640GB is perfectly suited to data science workloads, and lets us do more work in almost half as many nodes as the DGX-A100 320GB (6 nodes vs 10) for even better TCO.

Continuous Improvement

- 2.8x performance, almost a third the nodes, and cheaper to boot—in <1 year
- BlazingSQL at 10TB showing 25% improvement compared to Dask over TCP
- Q27 faster and more accurate with hugging Face

---

1: GPU-BDB is derived from the TPCx-BB benchmark and is used for internal performance testing. Results from GPU-BDB are not comparable to TPCx-BB.
cuDF
RAPIDS
GPU Accelerated data wrangling and feature engineering

Data Preparation → Model Training → Visualization

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

Dask
Apache Arrow
GPU Memory
cuDF — ANALYTICS
GPU DataFrame Library Built on Apache Arrow

Libraries

**Dask-cuDF**: Distributed Computing using Dask; Support for multi-GPU, multi-node

**cuDF**: Python bindings for libcudf (Pandas like API for DataFrame manipulation)

**libcudf**: CUDA C++ Apache Arrow GPU DataFrame and operators (Join/Merges, GroupBys, Sort, Filters, etc.)

---

### cuIO — FILE I/O
**Direct File Loading to cuDF**

<table>
<thead>
<tr>
<th>Availability</th>
<th>Supported File Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now</td>
<td>CSV, Parquet, ORC, JSON</td>
</tr>
</tbody>
</table>
Extraction is the Cornerstone

cuDF I/O 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 v0.12
- ORC Reader - v0.7, ORC Writer v0.10
- JSON Reader - v0.8
- Avro Reader - v0.9

- GPUDirect Storage integration in progress for bypassing PCIe bottlenecks!

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

```python
1]: import pandas, cudf
def 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

def 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

du -sh data/nyc/yellow_tripdata_2015-01.csv
1.96 data/nyc/yellow_tripdata_2015-01.csv
```

Source: Apache Crail blog: [SQL Performance: Part 1 - Input File Formats](#)
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
cuDF String Support

Current v0.16 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.
- Further performance optimization
- JIT-compiled String UDFs
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.
cuML
Machine Learning

More models more problems

Data Preparation → Model Training → Visualization

Dask

cuDF cuIO
Analytics
cuML
Machine Learning
cuGraph
Graph Analytics
PyTorch Chainer MxNet
Deep Learning

GPU Memory

cuXfilter <> pyViz
Visualization

Apache Arrow
cuML — MACHINE LEARNING
GPU Accelerated Scikit-learn + XGBoost Libraries

Dask
**Distributed Training:** Used for distributed cuML model training

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

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

**ml-prims**
**CUDA ML Primitives:** Low level machine learning primitives used in cuML
- Linear algebra, statistics, matrix operations, distance functions, random number generation
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'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)
```
RAPIDS matches common Python APIs

GPU-Accelerated Clustering

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(dict(('fea%d' % i : X[:, i]
                         for i in range(X.shape[1]))))

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

1x V100 vs. 2x 20 Core 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 per sec (with 1000 trees) on a DGX-1 for large (sparse) or dense models
• RAPIDS works closely with the XGBoost community to accelerate GBDTs on GPU
• XGBoost can seamlessly load data from cuDF dataframes and cuPy arrays
• Dask allows XGBoost to scale to arbitrary numbers of GPUs
• With the `gpu_hist` tree method, a single GPU can outpace 10s to 100s of CPUs
• RAPIDS comes paired with XGBoost 1.1 (as of 0.14)
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 25x 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

### RAPIDS 0.17 - December 2020

<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>
SHAP Explainability

GPUTreeSHAP for XGBoost

- **SHAP** provides a principled way to explain the impact of input features on each prediction or on the model overall - critical for interpretability

- SHAP has often been too computationally-expensive to deploy for large-scale production

- RAPIDS ships with GPU-accelerated SHAP for XGBoost with speedups of 20x or more (demo code available in the XGBoost repo)

- RAPIDS 0.17 includes experimental [Kernel and Permutation explainers](#) for black box models
cuGraph
Graph Analytics

More connections more insights

Data Preparation → Model Training → Visualization

Dask

cuDF, cuIO
Analytics

Dask

cuML
Machine Learning

Dask

cuGraph
Graph Analytics

Dask

PyTorch, Chainer, MxNet
Deep Learning

Dask

cuXfilter <-> pyViz
Visualization

Apache Arrow

GPU Memory
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
## 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>

*BigData x8, 100x 8-vCPU nodes, Apache Spark GraphX ⇒ 96 mins!
## Road to 1.0 - cuGraph
### RAPIDS 0.17 - December 2020

<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>
HOW TO GET STARTED AND EXAMPLES
RAPIDS
How do I get the software?

https://github.com/rapidsai
https://anaconda.org/rapidsai/

https://hub.docker.com/r/rapidsai/rapidsai/
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
- Release changes to clx and cuXfilter in v0.15
- RAPIDS dask-xgboost library is deprecated in v0.15
- Python 3.6 & CUDA 10.0 EOL in v0.14

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
- cuXfilter

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 10.1.2
- CUDA 10.2
- CUDA 11.0 (0.15/0.16 only)

COMMAND

conda install -c rapidsai -c nvidia -c conda-forge \
- c default -c rapids=0.15 python=3.7

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

https://github.com/rapidsai
# Notebook Examples

https://github.com/rapidsai/notebooks

## RAPIDS PREREQUISITES

<table>
<thead>
<tr>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA Pascal GPU architecture or better</td>
</tr>
<tr>
<td><strong>CUDA</strong> 10.1, 10.2, 11.0 &amp; compatible NVIDIA driver</td>
</tr>
<tr>
<td>Ubuntu 16.04/18.04, CentOS 7</td>
</tr>
<tr>
<td><strong>Docker CE v19.03+</strong></td>
</tr>
<tr>
<td><strong>nvidia-docker v2+</strong></td>
</tr>
</tbody>
</table>
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-gpai.slack.com/join

- **STACK OVERFLOW**
  https://stackoverflow.com/tags/rapids

Contribute back: Issues, Feature Requests, PRs, Blogs, Tutorials, Videos, QA
Dask + GPUs
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

Apache Arrow

GPU Memory
Dask

What is Dask?

• Distributed compute scheduler built to scale Python

• Scales workloads from laptops to supercomputer clusters

• Extremely modular: disjoint scheduling, compute, data transfer and out-of-core handling

• Multiple workers per node allow easier one-worker-per-GPU model
## Why Dask?

<table>
<thead>
<tr>
<th>PyData Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>Easy Migration:</strong> Built on top of NumPy, Pandas, Scikit-Learn, etc.</td>
</tr>
<tr>
<td>• <strong>Easy Training:</strong> With the same APIs</td>
</tr>
<tr>
<td>• <strong>Trusted:</strong> With the same developer community</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deployable</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>HPC:</strong> SLURM, PBS, LSF, SGE</td>
</tr>
<tr>
<td>• <strong>Cloud:</strong> Kubernetes</td>
</tr>
<tr>
<td>• <strong>Hadoop/Spark:</strong> Yarn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Easy Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Easy to install and use on a laptop</td>
</tr>
<tr>
<td>• Scales out to thousand-node clusters</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Popular</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Most common parallelism framework today in the PyData and SciPy community</td>
</tr>
</tbody>
</table>
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)
  [https://github.com/rapidsai/ucx-py](https://github.com/rapidsai/ucx-py)

• Will provide best communication performance, to Dask based on available hardware on nodes/cluster
OpenUCX
Dask Array SVD + CuPy Experiment with and without UCX
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
Development Environment

Jupyter Lab - Dask Extension - NVDashboard Extension
THANK YOU!