Distributed Multi-GPU Computing with Dask, CuPy and RAPIDS
Outline

• Interoperability / Flexibility
• Acceleration (Scaling Up)
• Distribution (Scaling Out)
Clustering

Code Example

```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({'fe%dd%i' % (i, 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)
What is RAPIDS?
New GPU-Accelerated Data Science Pipeline

• Suite of open source, end-to-end data science tools
• Built on CUDA
• Unifying framework for GPU data science
• Pandas-like API for data preparation
• Scikit-learn-like API for machine learning
RAPIDS
End-to-End GPU-Accelerated Data Science

Data Preparation ➔ Model Training ➔ Visualization

- cuDF cuIO Analytics
- cuML Machine Learning
- cuGraph Graph Analytics
- PyTorch Chainer MxNet Deep Learning
- cuXfilter <> Kepler.gl Visualization

GPU Memory ➔ Apache Arrow
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 Science Workflow with RAPIDS
Open Source, GPU-Accelerated ML Built on CUDA

DATA → cuDF → cuML → VISUALIZE → PREDICTIONS

- cuDF: Data preparation / wrangling
- cuML: ML model training
- VISUALIZE: Dataset exploration
- PREDICTIONS: Predictions
Ecosystem Partners

Community Contributors

[Logos of various companies including Anaconda, BlazingDB, Gunrock, NVIDIA, Quansight, and others]

Ecosystem Partners

[Logos of additional companies including Chainer, Databricks, graphistry, H2O.ai, IBM, Iguazio, Inria, MapR, Omnisci, PyTorch, Uber, and others]

Walmart
ML Technology Stack

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

- Dask cuML
- Dask cuDF
- cuDF
- Numpy

- Thrust
- Cub
- nvGraph
- cuBLAS
- cuRand
- cuSolver
- cuSparse
- CUTLASS
High-Level APIs

- Data Parallelism
- Model Parallelism

- Python
  - Dask Multi-GPU ML

- CUDA/C++
  - ML Algorithms
  - ML Primitives
  - Multi-Node / Multi-GPU Communication
    - Host 1: GPU1, GPU2, GPU3, GPU4
    - Host 2: GPU1, GPU2, GPU3, GPU4
UMAP

Dimensionality reduction technique now on GPU

Uniform Manifold Approximation and Projection (UMAP) is a dimension reduction technique that can be used for visualization similarly to t-SNE, but also for general non-linear dimension reduction.

- Fast
- General purpose dimension reduction
- Scales beyond what most t-SNE packages can manage
- Often preserves global structure better than t-SNE
- Supports a wide variety of distance functions
- Supports adding new points to an existing embedding via the standard scikit-learn transform method
- Supports supervised and semi-supervised dimension reduction
- Has solid theoretical foundations in manifold learning


UMAP
GPU vs CPU

GPU: 10.5 seconds

CPU: 100 seconds
Dask
What is Dask and why does RAPIDS use it for scaling out?

• 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
Distributing Dask

Distributed array from many arrays

NumPy Array

Dask Array
Combine Dask with CuPy

Distributed GPU array from many GPU arrays
NumPy Array Function (NEP-18)

Interoperability of NumPy-like Libraries

• Function dispatch mechanism
• Allows using NumPy as a high-level API
• NumPy-like arrays need only to implement \_\_array_function\_\_
Dask SVD Example
Interoperability of NumPy-like Libraries

In [1]: import dask, dask.array
   ...: import numpy

In [2]: x = numpy.random.random((1000000, 1000))
   ...: dx = dask.array.from_array(x, chunks=(10000, 1000), asarray=False)

In [3]: u, s, v = numpy.linalg.svd(dx)

In [4]: %%time
   ...: u, s, v = dask.compute(u, s, v)
CPU times: user 39min 4s, sys: 47min 31s, total: 1h 26min 35s
Wall time: 1min 21s
Dask+CuPy SVD Example
Interoperability of NumPy-like Libraries

In [1]: import dask, dask.array
   ...: import numpy
   ...: import cupy

In [2]: x = cupy.random.random((1000000, 1000))
   ...: dx = dask.array.from_array(x, chunks=(10000, 1000), asarray=False)

In [3]: u, s, v = numpy.linalg.svd(dx)

In [4]: %%time
   ...: u, s, v = dask.compute(u, s, v)
CPU times: user 34.5 s, sys: 17.6 s, total: 52.1 s
Wall time: 41 s
NumPy Array Function (NEP-18)

Protocol Limitations

- Universal functions - `__array_ufunc__` already addresses those
- `numpy.array()` and `numpy.asarray()` - will require their own protocol
- Dispatch for methods of any kind - e.g., `numpy.random.RandomState()`
uarray

Alternative to __array_function__

- Generic multiple-dispatch mechanism
- Intended to address shortcomings of NEP-18
- [https://uarray.readthedocs.io/](https://uarray.readthedocs.io/)
uarray

CuPy Example

In [1]: import uarray as ua
       ...: import unumpy as np
       ...: import unumpy.cupy_backend as cupy_backend

In [2]: with ua.set_backend(cupy_backend):
       ...:     a = np.ones((2, 2))
       ...:     print(np.sum(a))
       ...:     print(type(a))
       ...:     print(type(np.sum(a)))
4.0
<class 'cupy.core.core.ndarray'>
<class 'cupy.core.core.ndarray'>
uarray
Dask+CuPy Example

In [1]: import uarray as ua
   ...: import unumpy as np
   ...: import unumpy.cupy_backend as cupy_backend
   ...: import unumpy.dask_backend as dask_backend

In [2]: with ua.set_backend(cupy_backend), ua.set_backend(dask_backend):
   ...:     a = np.ones((2, 2))
   ...:     print(np.sum(a).compute())
   ...:     print(type(a))
   ...:     print(type(np.sum(a).compute()))

4.0
<class 'dask.array.core.Array'>
<class 'numpy.float64'>  # currently
<class 'cupy.core.core.ndarray'>  # expected – Dask will need to support uarray for this to work!
Python CUDA Array Interface
Interoperability for Python GPU Array Libraries

• GPU array standard

• Allows sharing GPU array between different libraries

• Native ingest and export of __cuda_array_interface__ compatible objects via Numba device arrays in cuDF

• Numba, CuPy, and PyTorch are the first libraries to adopt the interface:
  • https://numba.pydata.org/numba-doc/dev/cuda/cuda_array_interface.html
  • https://github.com/cupy/cupy/releases/tag/v5.0.0b4
  • https://github.com/pytorch/pytorch/pull/11984
Interoperability for the Win

DLPack and __cuda_array_interface__
Challenges: Communication

OpenUCX

- TCP sockets are slow!
- UCX provides uniform access to transports (TCP, InfiniBand, shared memory, NVLink)
- Python bindings for UCX (ucx-py) in the works https://github.com/rapidsai/ucx-py
- Will provide best communication performance, to Dask according to available hardware on nodes/cluster
Challenges: Communication

OpenUCX Performance - Before and After
Benchmark: single-GPU CuPy vs NumPy

Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>GPU Speedup Over CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elementwise</td>
<td>270</td>
</tr>
<tr>
<td>FFT</td>
<td>210</td>
</tr>
<tr>
<td>Array Slicing</td>
<td>190</td>
</tr>
<tr>
<td>Stencil</td>
<td>150</td>
</tr>
<tr>
<td>Sum</td>
<td>66</td>
</tr>
<tr>
<td>Matrix Multiplication</td>
<td>11</td>
</tr>
<tr>
<td>SVD</td>
<td>18</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>17</td>
</tr>
</tbody>
</table>

Array Size
- 800MB
- 8MB

Benchmarks: single-GPU cuML vs scikit-learn
SVD Benchmark

![Graph showing SVD Benchmark results for different configurations]

- **Dask - Single DGX-1 (CPU only - 80 threads)**
- **Dask + CuPy - Single DGX-1 (1 Tesla V100)**
- **Dask + CuPy - Single DGX-1 (8 Tesla V100)**
- **Dask + CuPy - Dual DGX-1 (2x8 Tesla V100)**

The graph illustrates the compute time (in seconds) vs. the number of rows and columns for each configuration.
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 up and out with RAPIDS and Dask

**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 + RAPIDS**
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**
## Road to 1.0
### October 2018 - RAPIDS 0.1

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

**June 2019 - RAPIDS 0.8**

<table>
<thead>
<tr>
<th>cuML</th>
<th>Single-GPU</th>
<th>Multi-GPU</th>
<th>Multi-Node-Multi-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Boosted Decision Trees (GBDT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logistic Regression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Forest (regression)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-NN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBSCAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARIMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kalman Filter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Holts-Winters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principal Components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Road to 1.0
### Q4 - 2019 - RAPIDS 0.12?

<table>
<thead>
<tr>
<th>cuML</th>
<th>Single-GPU</th>
<th>Multi-GPU</th>
<th>Multi-Node-Multi-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient Boosted Decision Trees (GBDT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logistic Regression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Forest (regression)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-Means</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-NN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBSCAN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARIMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kalman Filter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Holts-Winters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principal Components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singular Value Decomposition</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Road to 1.0
Focused on robust functionality, deployment, and user experience

Integration with every major cloud provider
Both containers and cloud specific machine instances
Support for Enterprise and HPC Orchestration Layers
RAPIDS

How do I get the software?

• [https://github.com/rapidsai](https://github.com/rapidsai)
• [https://anaconda.org/rapidsai/](https://anaconda.org/rapidsai/)
• [https://hub.docker.com/r/rapidsai/rapidsai/](https://hub.docker.com/r/rapidsai/rapidsai/)
Additional Reading Material

• Python, Performance and GPUs (Matthew Rocklin): https://towardsdatascience.com/python-performance-and-gpus-1be860ffd58d?ncid=so-tw-n2-96487&linkId=100000006881312


• uarray update: API changes, overhead and comparison to __array_function__ (Hameer Abbasi): https://labs.quansight.org/blog/2019/07/uarray-update-api-changes-overhead-and-comparison-to-__array_function__/