Data sizes continue to grow

Prototyping and production diverge

Large-scale cluster

- Spark, Hadoop
- High throughput
- Full data

Workstation

- Python
- Fast iteration
- Small data subset

Challenges!

- “Tools gap” - rewriting Python or R code to Spark/Hadoop jobs to scale to cluster
- High latency on cluster leads to slower iteration
- Small data subsets on workstation make it hard to build realistic models

RAPIDS on GPU

- RAPIDS + Dask
- Consistent tools: Workstation or Cluster
- High throughput / low latency
- Full data or large subsets
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?
Data Processing Evolution

Faster data access, less data movement

Hadoop Processing, Reading from disk

Spark In-Memory Processing

Traditional GPU Processing

RAPIDS

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

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

50-100x Improvement
Same code
Language flexible
Primarily on GPU
RAPIDS
Scale up and out with accelerated GPU data science

Data Preparation \rightarrow Model Training \rightarrow Visualization

Dask

cuDF cuIO Analytics \uparrow cuML Machine Learning \uparrow cuGraph Graph Analytics \uparrow PyTorch Chainer MxNet Deep Learning \uparrow cuXfilter <> pyViz Visualization

Apache Arrow \rightarrow GPU Memory
Scale up and out with accelerated GPU data science

Data Preparation → Model Training → Visualization

Dask

Pandas API

cuDF cuIO Analytics

cuML Machine Learning

cuGraph Graph Analytics

PyTorch Chainer MxNet Deep Learning

cuXfilter <-> pyViz Visualization

sklearn API

NetworkX API

Apache Arrow

GPU Memory
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
Faster Speeds, Real-World Benefits

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

- 20 CPU Nodes: 2741 seconds
- 30 CPU Nodes: 1675 seconds
- 50 CPU Nodes: 715 seconds
- 100 CPU Nodes: 379 seconds
- DGX-2: 42 seconds
- 5x DGX-1: 19 seconds

**cuML - XGBoost**

- 20 CPU Nodes: 2290 seconds
- 30 CPU Nodes: 1956 seconds
- 50 CPU Nodes: 1999 seconds
- 100 CPU Nodes: 1948 seconds
- DGX-2: 169 seconds
- 5x DGX-1: 157 seconds

**End-to-End**

- 20 CPU Nodes: 8762 seconds
- 30 CPU Nodes: 6148 seconds
- 50 CPU Nodes: 3925 seconds
- 100 CPU Nodes: 3221 seconds
- DGX-2: 322 seconds
- 5x DGX-1: 213 seconds

**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

**DGX Cluster Configuration**

- 5x DGX-1 on InfiniBand network
Dask + cuML
Machine Learning at Scale
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
- `cuXfilter <-> pyViz`
  Visualization

Apache Arrow → GPU Memory
Algorithms

GPU-accelerated Scikit-Learn

- Classification / Regression:
  - Decision Trees / Random Forests
  - Linear Regression
  - Logistic Regression
  - K-Nearest Neighbors
- Inference:
  - Random forest / GBDT inference
- Clustering:
  - K-Means
  - DBSCAN
  - Spectral Clustering
  - Principal Components
  - Singular Value Decomposition
  - UMAP
  - Spectral Embedding
- Decomposition & Dimensionality Reduction
- Time Series:
  - Holt-Winters
  - Kalman Filtering

Cross Validation

Hyper-parameter Tuning

More to come!
RAPIDS matches common Python APIs

GPU-Accelerated Clustering

```python
from sklearn.datasets import make_moons
import cudf

X, y = make_moons(n_samples=int(1e2),
                  noise=0.05, random_state=0)
X = cudf.DataFrame({f'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 CPU
Why Dask?

• **PyData Native**
  • Built on top of NumPy, Pandas Scikit-Learn, etc. (easy to migrate)
  • With the same APIs (easy to train)
  • With the same developer community (well trusted)

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

• **Popular**
  • Most common parallelism framework today at PyData and SciPy conferences

• **Deployable**
  • HPC: SLURM, PBS, LSF, SGE
  • Cloud: Kubernetes
  • Hadoop/Spark: Yarn
Using Dask in Practice

Familiar Python APIs

```python
import pandas as pd
df = pd.read_csv("data-*.csv")
df.groupby(df.user_id).value.mean()
```

```python
import dask_cudf
def = dask_cudf.read_csv("data-*.csv")
df.groupby(df.user_id).value.mean().compute()
```

Dask supports a variety of data structures and backends
A quick demo...
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/)
AI is more than Model Training

Identify and define business problem
Capture business requirements

Modeling & performance evaluation
Model development

Deploy the solution at scale
Promote user adoption
Continuous model improvement

EXPLORATION

Exploratory data analysis
Acquire, prepare & enrich data
Preliminary ROI analysis

FINDINGS

Deliver Insights & Recommendations
Measure business effectiveness & ROI
Promote business enablement

Source: Adapted from DellTech Data science solutions learnings & presentation
A vision for end-to-end ML journey – Fluid and Flexible

Dell Tech AI offerings Cloud to In-house, One vendor with no Lock-in

- Direct & Consultative Sales
- System Integrator partners
- Dell Tech consulting services

SOFTWARE PARTNERS ECO-SYSTEM

Enterprise ISV ML Software
Accelerator Virtualization and pooling
Open Source Deep Learning Frameworks

SOFTWARE PARTNERS DATA PREPARATION

Public cloud storage
Elastic cloud storage
Dell EMC Enterprise Storage

CLOUD FOUNDATION

Public cloud & distributed edge
Hosted Private cloud
Hybrid Cloud Appliances
Ready Bundles for ML and DL
Infrastructure

Pivotal
BOOMI CONNECTORS

PARTNER SOFTWARE DATA PREPARATION
### AI and Deep Learning Workstation Use Cases

**Data Science Sandboxes**

**Production Development**

<table>
<thead>
<tr>
<th>NVIDIA GPU CLOUD (NGC) Container Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow, Caffe2, Apache SparkML, Rapids, mxnet, pyTorch, Microsoft Cognitive Services</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU 1-3 x</th>
<th>Precision WS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy and stage</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU 1-3 x</th>
<th>Precision WS</th>
</tr>
</thead>
<tbody>
<tr>
<td>In place access</td>
<td></td>
</tr>
</tbody>
</table>

Isilon Storage
Pre-verified GPU-accelerated Deep Learning Platforms

White papers, best practices, performance tests, dimensioning guidelines

NVIDIA GPU CLOUD (NGC)

TensorFlow, Caffe2, Spark, ML, Rapids, mxnet, pyTorch

Ready Solutions
Currently pre-verified
To be verified in H2 2019

Isilon F, H and A nodes as appropriate
ECS Object Store

Isilon the foundation data lake for AI platforms
THANK YOU

Ramesh Radhakrishnan
TODO insert email / Twitter

John Zedlewski @zstats
jzedlewski@nvidia.com
Join the Movement
Everyone can help!

APACHE ARROW
https://arrow.apache.org/
@ApacheArrow

RAPIDS
https://rapids.ai
@RAPIDSAI

Dask
https://dask.org
@Dask_dev

GPU Open Analytics Initiative
http://gpuopenanalytics.com/
@GPUOAI

Integrations, feedback, documentation support, pull requests, new issues, or code donations welcomed!
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
GPU-Accelerated ETL
The average data scientist spends 90+% of their time in ETL as opposed to training models.
ETL - the Backbone of Data Science

cuDF is...

Python Library

- A Python library for manipulating GPU DataFrames following the Pandas API
- Python interface to CUDA C++ library with additional functionality
- Creating GPU DataFrames from Numpy arrays, Pandas DataFrames, and PyArrow Tables
- JIT compilation of User-Defined Functions (UDFs) using Numba
Benchmarks: single-GPU Speedup vs. Pandas

GPU (cuDF) Speedup vs. CPU (Pandas)

- groupby: 32 vs. 6.7
- merge: 62 vs. 24
- sort: 32 vs. 15

Shape
- (100M rows, 2 columns)
- (10M rows, 2 columns)

cuDF v0.8, Pandas 0.23.4
Running on NVIDIA DGX-1:

GPU: NVIDIA Tesla V100 32GB
CPU: Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
Eliminate data extraction bottlenecks

cuIO under the hood

- Follow Pandas APIs and provide >10x speedup
- CSV Reader - v0.2, CSV Writer v0.8
- Parquet Reader - v0.7, Parquet Writer v0.10
- ORC Reader - v0.7, ORC Writer v0.10
- JSON Reader - v0.8
- Avro Reader - v0.9
- GPU Direct Storage integration in progress for bypassing PCIe bottlenecks!
- Key is GPU-accelerating both parsing and decompression wherever possible

Source: Apache Crail blog: SQL Performance: Part 1 - Input File Formats
cuML
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