RAPIDS
Accelerating GPU Data Science End-to-End

Data Preparation → Model Training → Visualization

Dask

cuDF, cuIO
Analytics

cuML
Machine Learning

cuGraph
Graph Analytics

PyTorch,
TensorFlow, MxNet
Deep Learning

cuXfilter <-> pyViz
Visualization

Apache Arrow
GPU Memory
## DATA SCIENCE API

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>GPU/RAPIDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Manipulation</td>
<td>Pandas</td>
<td>cuDF</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>scikit-learn</td>
<td>cuML</td>
</tr>
<tr>
<td>Graph Analysis</td>
<td>NetworkX</td>
<td>cuGraph</td>
</tr>
<tr>
<td>Scalability</td>
<td>Dask</td>
<td>Dask</td>
</tr>
</tbody>
</table>
**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

### cuIO/cuDF - Load and Data Preparation

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 CPU Nodes</td>
<td>2741</td>
</tr>
<tr>
<td>30 CPU Nodes</td>
<td>1675</td>
</tr>
<tr>
<td>50 CPU Nodes</td>
<td>715</td>
</tr>
<tr>
<td>1000 CPU Nodes</td>
<td>379</td>
</tr>
<tr>
<td>DGX-2</td>
<td>42</td>
</tr>
<tr>
<td>5x DGX-1</td>
<td>19</td>
</tr>
</tbody>
</table>

### XGBoost Machine Learning

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 CPU Nodes</td>
<td>2290</td>
</tr>
<tr>
<td>30 CPU Nodes</td>
<td>1956</td>
</tr>
<tr>
<td>50 CPU Nodes</td>
<td>1999</td>
</tr>
<tr>
<td>1000 CPU Nodes</td>
<td>1948</td>
</tr>
<tr>
<td>DGX-2</td>
<td>169</td>
</tr>
<tr>
<td>5x DGX-1</td>
<td>157</td>
</tr>
</tbody>
</table>

### End-to-End

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 CPU Nodes</td>
<td>8762</td>
</tr>
<tr>
<td>30 CPU Nodes</td>
<td>6148</td>
</tr>
<tr>
<td>50 CPU Nodes</td>
<td>3925</td>
</tr>
<tr>
<td>1000 CPU Nodes</td>
<td>3221</td>
</tr>
<tr>
<td>DGX-2</td>
<td>322</td>
</tr>
<tr>
<td>5x DGX-1</td>
<td>213</td>
</tr>
</tbody>
</table>

**Time in seconds (shorter is better)**
- cuIO/cuDF (Load and Data Prep)
- Data Conversion
- XGBoost
## ROAD TO 1.0

**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>
Single-cell RNA-seq

Read Counts

<table>
<thead>
<tr>
<th></th>
<th>Cell 1</th>
<th>Cell 2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gene 1</td>
<td>18</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Gene 2</td>
<td>1010</td>
<td>506</td>
<td></td>
</tr>
<tr>
<td>Gene 3</td>
<td>0</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>Gene 4</td>
<td>22</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Compare gene expression profiles of single cells

Gene 1
Cell 1

Single cell gene expression
CHALLENGES

Single-cell analysis pipelines need to be fast, exploratory and interactive.

Exploring, analyzing and re-analyzing in real time.
Clustering and sub-clustering cells.
Trying different algorithms and parameters.
### Towards Interactive Tertiary Analysis

#### End-to-End RAPIDS Single Cell Analysis Pipeline

<table>
<thead>
<tr>
<th>Preprocess</th>
<th>Reduce</th>
<th>Visualize</th>
<th>Cluster</th>
<th>Differential Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear regression</td>
<td>PCA</td>
<td>KNN+UMAP</td>
<td>Louvain</td>
<td>Logistic regression</td>
</tr>
<tr>
<td>cuDF/cuML</td>
<td>cuML</td>
<td>cuML</td>
<td>cuGraph</td>
<td>cuML</td>
</tr>
</tbody>
</table>

#### “Fast scRNA” Pipeline x20 (UMAP-Louvain)

- Linear regression: cuDF/cuML
- PCA: cuML
- KNN+UMAP: cuML
- Louvain: cuGraph
- Logistic regression: cuML

#### “Slow scRNA” Pipeline x80 (tSNE-kMeans)

- Linear regression: cuDF/cuML
- PCA: cuML
- t-SNE: cuML
- k-Means: cuML
- Logistic regression: cuML
## RAPIDS & Scanpy Single-Cell RNA-seq Workflow

**Copyright © 2020, NVIDIA CORPORATION. All rights reserved.**

NVIDIA CORPORATION and its licensors retain all intellectual property and proprietary rights in and to this software, related documentation and any modifications thereof. Any use, reproduction, disclosure or distribution of this software and related documentation without an express license agreement from NVIDIA CORPORATION is strictly prohibited.

This notebook demonstrates a single-cell RNA-seq analysis workflow that begins with preprocessing a count matrix of size _n_genomes_ x _n_cells_ and results in a visualization of the clustered cells for further analyses. For demonstration purposes, we use a dataset of ~70k human lung cells from Treaklin et al. 2020 [https://www.biorxiv.org/content/10.1101/2020.03.10.924022](https://www.biorxiv.org/content/10.1101/2020.03.10.924022) and label cells using the AC12 and IMPRESS3 genes.

To install requirements: conda install -c conda-forge -c rapidai cudatoolkit=11.2 cudf=0.13 coul=0.13 scanpy=1.13 paramep=0.8* dask=2.1.1* distributed=2.1.1* jupyterlab=3.0

### Import requirements

```python
In [1]:
import scanpy as sc
import numpy as np
import anndata
import cudf
import pandas as pd
from sc.raw import count_matrix
from sc.raw import features
from sc.raw import layers
from sc.raw import expression
import warnings
warnings.filterwarnings('ignore', 'deprecated')

In [2]:
from dask.distributed import Client
cli = Client('localhost:8786')
cli
```

### Load and Prepare Data

We have saved our count matrix data as a sparse matrix in `h5ad` format. This is much faster to load than a dense CSV file. To convert your CSV file into an `h5ad` file with a sparse count matrix, see [cav_to_sparse_h5ad.py](https://github.com/clara-parabricks/rapids-single-cell-examples).

```python
In [5]:
# Add path to input file here.
input_file = "honey_blon_fly_10x_INXS_sparse.h5ad"

In [6]:
data_load_start = time.time()

In [7]:
@time
adata = sc.read(input_file)
CPU times: user 3.21 s, sys: 973 ms, total: 4.19 s
Wall time: 4.05 s

In [8]:
adata.shape
Out[8]:

```

We maintain the index of unique cells and genes in our dataset:

```python
In [9]:
@time
cells = cudf.Series(adata.obs_names)
genes = cudf.Series(adata.var_names)
CPU times: user 778 ms, sys: 433 ms, total: 1.21 s
Wall time: 1.94 s

In [10]:
@time
sparse_gpu_array = cp.sparse.csr_matrix(adata.X)
CPU times: user 213 ms, sys: 311 ms, total: 523 ms
Wall time: 570 ms

Verify the shape of the resulting sparse matrix:

```python
In [11]:
sparse_gpu_array.shape
Out[11]:
```
RAPIDS integrated within SCNAPY

Scanpy - Single-Cell Analysis in Python

```
In [ ]: %time
    sc.pp.neighbors(adata, n_neighbors=n_neighbors, n_pcs=knn_n_pcs, method='rapids')
```

The UMAP function from Rapids is also integrated into Scanpy.

```
In [ ]: %time
    sc.tl.umap(adata, min_dist=umap_min_dist, spread=umapspread, method='rapids')
```

Finally, we use the Louvain algorithm for graph-based clustering, once again using the rapids option in Scanpy.

```
In [ ]: %time
    sc.tl.louvain(adata, flavor='rapids')
```

We plot the cells using the UMAP visualization, using the Louvain clusters as labels.
PERFORMANCE DEMONSTRATION

~70,000 human lung cells

t-SNE + k-Means pipeline

UMAP + Louvain pipeline
ACCELERATION BENCHMARKS

Acceleration

All runtimes are given in seconds.

<table>
<thead>
<tr>
<th>Step</th>
<th>CPU runtime (16 core AMD EPYC 7571)</th>
<th>GPU runtime (Tesla V100 32 GB)</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessing</td>
<td>324.35</td>
<td>68.49</td>
<td>4.7x</td>
</tr>
<tr>
<td>PCA</td>
<td>16.2</td>
<td>1.59</td>
<td>10.2x</td>
</tr>
<tr>
<td>t-SNE</td>
<td>166</td>
<td>1.95</td>
<td>85.1x</td>
</tr>
<tr>
<td>k-means (single iteration)</td>
<td>7.3</td>
<td>0.11</td>
<td>66.4x</td>
</tr>
<tr>
<td>KNN</td>
<td>23</td>
<td>5.18</td>
<td>4.4x</td>
</tr>
<tr>
<td>UMAP</td>
<td>78</td>
<td>0.98</td>
<td>80x</td>
</tr>
<tr>
<td>Louvain clustering</td>
<td>13.6</td>
<td>0.25</td>
<td>54.4x</td>
</tr>
<tr>
<td>Differential Gene Expression</td>
<td>45.1</td>
<td>18.9</td>
<td>2.4x</td>
</tr>
</tbody>
</table>
Analyze 1 million cells in 11 minutes

<table>
<thead>
<tr>
<th>AWS Instance</th>
<th>m5a.12xlarge</th>
<th>p3.8xlarge</th>
<th>Acceleration</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU/GPU type</td>
<td>Intel Xeon Platinum 8000, 48 vCPUs</td>
<td>V100-16GB</td>
<td></td>
</tr>
<tr>
<td>Preprocessing</td>
<td>4033</td>
<td>323</td>
<td>12.5</td>
</tr>
<tr>
<td>PCA</td>
<td>34</td>
<td>20.6</td>
<td>1.7</td>
</tr>
<tr>
<td>t-SNE</td>
<td>5417</td>
<td>41</td>
<td>132.1</td>
</tr>
<tr>
<td>k-Means clustering</td>
<td>106</td>
<td>2.1</td>
<td>50.5</td>
</tr>
<tr>
<td>KNN</td>
<td>585</td>
<td>53.4</td>
<td>11.0</td>
</tr>
<tr>
<td>UMAP</td>
<td>1751</td>
<td>20.3</td>
<td>86.3</td>
</tr>
<tr>
<td>Louvain clustering</td>
<td>597</td>
<td>2.5</td>
<td>238.8</td>
</tr>
<tr>
<td>End-to-end</td>
<td>13002</td>
<td>672.7</td>
<td>19.3</td>
</tr>
<tr>
<td>Price/hr ($)</td>
<td>2.064</td>
<td>12.24</td>
<td></td>
</tr>
<tr>
<td>Cost ($)</td>
<td>7.455</td>
<td>2.287</td>
<td>3.3</td>
</tr>
</tbody>
</table>
GPU-accelerated cell browsing

Real-time, interactive selection and analysis

Point-and-click clustering and exploration of selected cells

Export data from selected cells for analysis
Many more spaces you could leverage RAPIDS

**Cheminformatics** - searching, screening, and organizing large chemical databases

https://github.com/NVIDIA/cheminformatics

**GWAS**

https://github.com/STRIDES-Codes/GPU-GWAS

Workflow:

1. VCF
2. Parquet
3. Variant dataframe
4. Filtered variant DF
5. Sample data frame
6. Sample annotated dataframe
7. cuDF
8. Regression model output
9. Visualize p-values
10. Sample annotations txt
11. cuDF csv reader

VariantWorks VCFIO
cuDF pqt reader
Filter alleles
PCA on genotype column
Merge DF
RAPIDS
Get Started

github.com/rapidsai  anaconda.org/rapidsai  ngc.nvidia.com/rapidsai  hub.docker.com/rapidsai

RAPIDS RELEASE SELECTOR

RAPIDS is available as conda packages, docker images, and from source, built. Use the tab below to select your preferred method, packages, and versions to install RAPIDS. Certain combinations may not be possible and are dimmed automatically. Be sure you've installed the required Miniconda before.

METHOD
Conda (Preferred)

RELEASE
R裸 (Recommended)

PACKAGES
All Packages

GPU
Ubuntu: 16.04

CPU
Python: 3.6


COPY CONFIGURE

 DETAILS BELOW
加入 NVIDIA 開發者計畫

請掃描左邊 QR code 加入 NVIDIA 開發者計畫，活動結束前完成註冊，即可換取精美小禮物。

請向 NVIDIA 工作人員出示主旨為「Your application for the program NVIDIA Developer Program is approved」的email。

developer.nvidia.com/developer-program
THANK YOU