AI APPLICATION BOOST WITH NVIDIA RAPIDS ACCELERATION

EXPERT ACADEMY

COURSE CONTENT

- Introduction to GPU, CUDA and other concepts
- Benefits of using a GPU
- Introduction to NVIDIA’s solutions for GPU acceleration

RAPIDS

- cuDF – equivalent to pandas
- cuML – equivalent to sklearn
- Development of a complete project
- Dask + cuDF + cuML

REQUIREMENTS

- Programming Logic
- Basic Python programming
- Basic knowledge about Machine Learning

WHAT IS A GPU?

- **GPUs** (*Graphics Processing Units*) are specialized processors designed originally to accelerate graphics and computational processing.
- Unlike CPUs *(Central Processing Unit)*, GPUs have thousands of cores that can execute tasks simultaneously, making them highly efficient for parallel tasks.
- They were originally developed to render graphics in games and computer graphics applications, but their potential has extended to other areas, such as Data Science and Machine Learning.
- The GPUs are highly efficient in processing large volumes of data and complex calculations, due to its parallel architecture and mass execution capabilities.

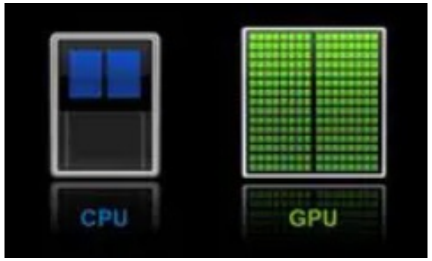


Image Source: NVIDIA

GPU IN AI APPLICATIONS

- GPUs have become essential tools for accelerating machine learning algorithms and performing intensive calculations in various areas, such as scientific research, data analysis and computer simulations.
- GPUs are widely used in high-performance computing platforms, data centers and cloud computing, thus providing a huge boost in processing and performance for various applications.
- GPU acceleration has been especially relevant for modern artificial intelligence applications, where complex and data-intensive algorithms are more common.

GPUs vs. CPUs

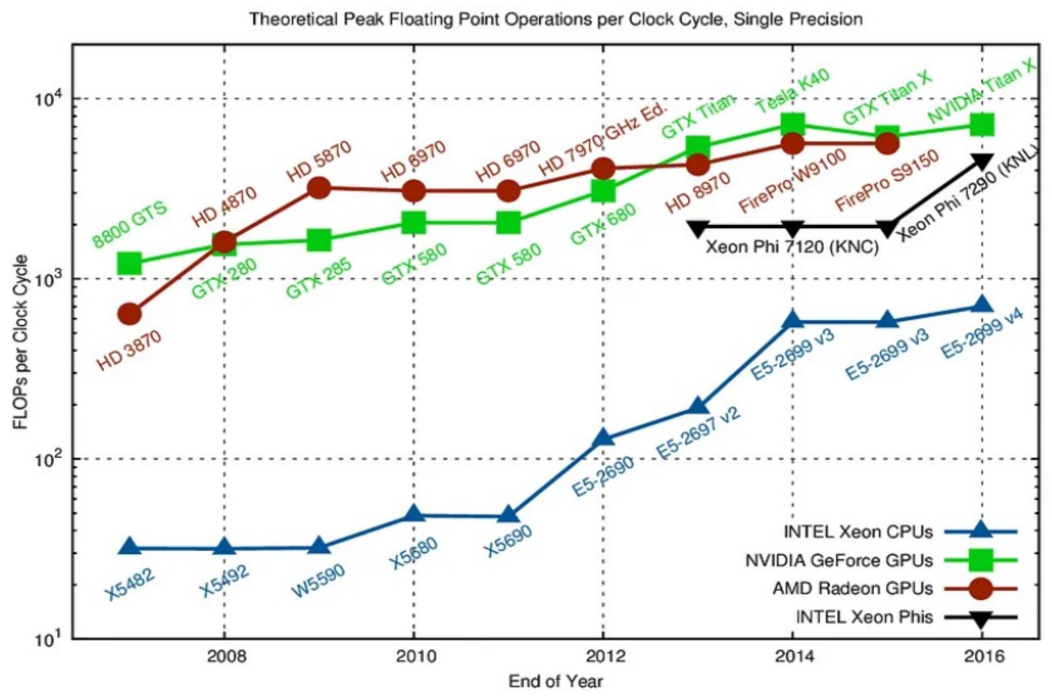
- CPU is composed of just a few cores with lots of cache memory that can handle a few software threads at a time.
- On the other hand, a GPU is made up of hundreds of cores that can handle thousands of threads simultaneously, that is, thousands of operations at the same time.
- However, CPUs certainly are still essential. While CPUs are recommended for serial processing, GPUs are best for parallel processing.

For AI applications:

- GPU: accelerates processing by simultaneously running multiple tasks across thousands of cores, providing speed and efficiency for data-intensive algorithms.
- CPU: Although it is designed for serial tasks, the CPU can also run AI applications, but on a smaller scale. It is best suited for sequential and complex tasks.

GPUs vs. CPUs

GPUs *(red/green)* can theoretically perform 10 to 15 times more operations than CPUs *(blue)*.



Source: fast.ai

BENEFITS OF GPU FOR AI APPLICATIONS

- The main advantage of GPUs is the parallelism: they enable parallel execution of tasks, allowing multiple calculations to be processed simultaneously.
- By using GPUs, it is possible to accelerate the training and inference of machine learning models and neural networks.
- The ability to run tasks in parallel on GPUs reduces the time needed to train models and analyze the results.

BENEFITS OF GPU FOR AI APPLICATIONS

- GPUs offer better performance for real-time image processing and computer vision algorithms.
- As the demand for real-time analytics is constantly growing, GPUs provide the speedup needed to quickly respond to events and make timely decisions.
- GPU acceleration makes it possible to handle increasingly larger and complex datasets. They are able to boost the scientific research and the discovery of insights in diverse areas of knowledge.

REQUIREMENTS FOR USING GPU IN YOUR PROJECT

- The system and the GPU need to be compatible
- The programming language and its libraries need to have GPU support
- Adapt the code to use GPU
- It is also important to test and evaluate the code's performance to know how viable it is to transfer to GPU.

GPUs

Some tools are specifically designed to use GPUs:

- APIs that can directly access the GPU – e.g. CUDA and OpenCL
- Implementations of popular algorithms such as convolutions and matrix operations.
- Functions to move data from CPU to GPU – and vice-versa.

CUDA

- **CUDA** (*Compute Unified Device Architecture*) is a parallel computing platform and programming model developed by NVIDIA.
- Allows the development of programs using languages such as C/C++.
- It was designed to use the power of GPU parallel processing so it can perform tasks that require high computational performance.
- CUDA provides a development toolkit with compilers and debuggers to assist in programming with the GPU.

CUDA

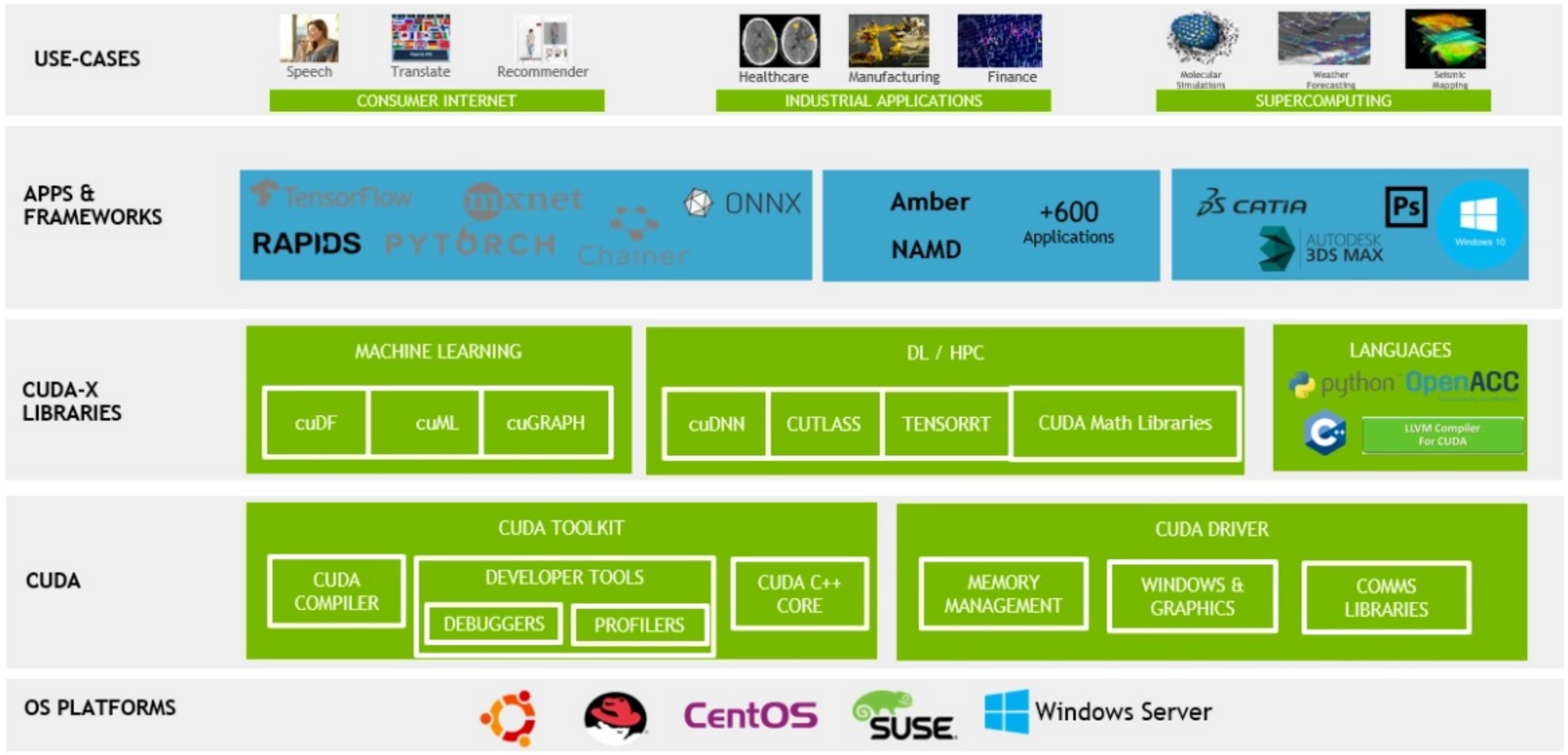


Image source: NVIDIA

NVIDIA'S SOLUTIONS FOR GPU ACCELERATION

- NVIDIA currently offers some solutions that provide significant acceleration for data processing and video analysis.
- These solutions can allow a significant increase in performance and efficiency, which opens possibilities for high-impact applications in various sectors.

RAPIDS

- Offers a broad ecosystem of GPU libraries used to accelerate Data Science and Machine Learning projects.
- Enables the use of GPUs to accelerate common tasks such as loading, manipulating, analyzing and visualizing data.

DeepStream

- Real-time video analytics platform.
- Designed to process and manage video streams in real-time, allowing to accelerate advanced computer vision techniques.

RAPIDS - NVIDIA

WHAT IS RAPIDS?

- RAPIDS is a suite of open-source libraries that enable end-to-end execution of data science and machine learning pipelines entirely on GPUs.

RAPIDS

RAPIDS: Libraries for End to End GPU Data Science
https://rapids.ai

- It provides significant speedups with an API that mirrors the most popular Data Science and Machine Learning libraries for Python, like Pandas and scikit-learn.

- RAPIDS eliminates the need to know the technical details or learn how to deal with parallelism and GPU-specific languages – e.g. CUDA or OpenCL.

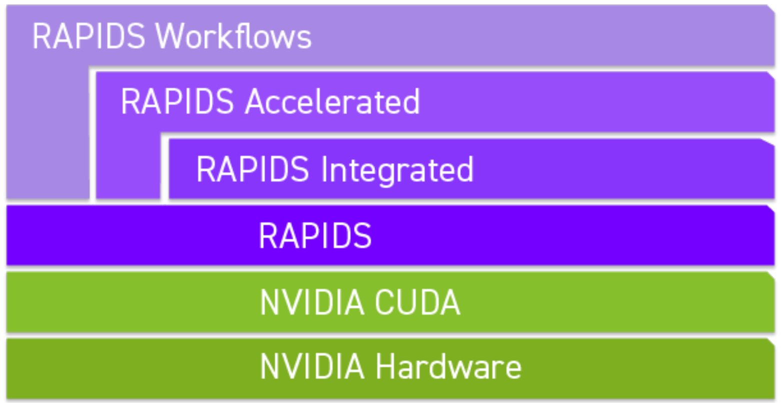
BENEFITS OF RAPIDS

- Easy and familiar integration
- Productivity boost
- Increase in speed and accuracy
- Cost reduction
- Open source

BENEFITS OF RAPIDS

- In many situations, the complexity and cost of working with a GPU is in the process of getting data out of host memory and putting into GPU memory.
- RAPIDS performs this process in a very optimized way, leaving the data in the GPU cache.
- The more we can process using the GPU, the more acceleration we will have.
- By using RAPIDS, there is a potential of 100x to 200x speedup on more powerful GPUs (the speedup factor vary a lot between different GPU models).

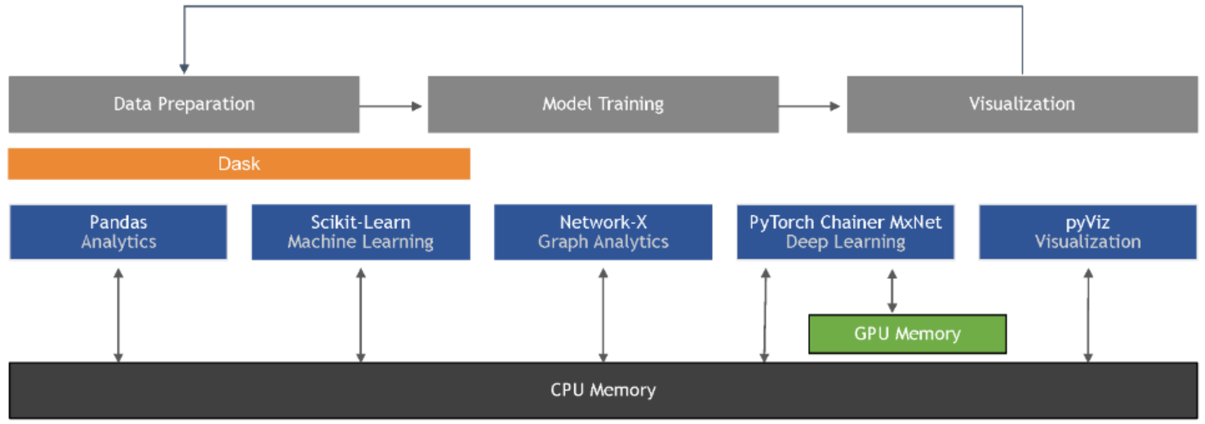
DATA SCIENCE AND MACHINE LEARNING ECOSYSTEM



Source: https://rapids.ai

TRADITIONAL ECOSYSTEM VS. RAPIDS ECOSYSTEM

Traditional Data Science Ecosystem



RAPIDS

Ecosystem

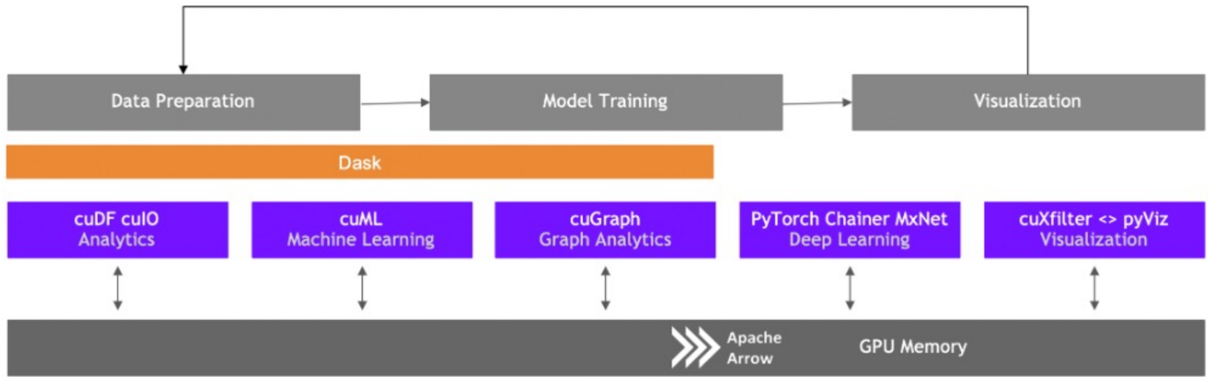


Image source: https://rapids.ai/

APACHE ARROW

- RAPIDS works connected with Apache Arrow, inside GPU memory.
- Apache Arrow provides an efficient and interoperable columnar memory format. It transforms a table into columnar data.
- It means that the data can be accessed faster.
- It improves the performance and efficiency for reading, writing and manipulating data operations.
- Apache Arrow contributes to GPU-accelerated data processing in an efficient and scalable way, reducing data conversion overhead and improving the efficiency.

APACHE ARROW
A cross-language development platform for in-memory analytics

https://arrow.apache.org

RAPIDS APIs

RAPIDS is able to reimplement famous Python libraries to work efficiently with GPUs, offering more efficiency and speed than would be possible with a CPU

- **cuDF** – is a DataFrame library for loading, joining, aggregating, filtering, and otherwise manipulating data. It was designed to be an alternative to **Pandas**.
- **cuML** – is a ML library that uses CUDA to run Machine Learning algorithms on GPUs. It is an alternative to **scikit-learn**.
- **cuGraph** – graph analysis library. It is an alternative to **NetworkX.**
- **cuSpatial** – provides functionalities to spatial computation. It is an alternative to **GeoPandas** and **Shapely**
- **cuSignal** – based on and extends **scipy.signal**
- **cuCIM** – alternative to **scikit-image**.

More APIs: https://github.com/rapidsai *(which may still be released in the near or distant future)*

INTEGRATION TOOLS

RAPIDS Can be integrated with:

- Dask – Used to workloads across several GPUs.
- Apache Spark – fully integrated GPU acceleration through Spark RAPIDS to accelerate workflows without the need to change the code, thus increasing performance and reducing costs.
- Dask SQL – integrated with Rapids SQL, an open source lib that brings SQL functionalities to Dask; allows you to write SQL queries to work with large distributed datasets, easily scaling to large volumes of data.
- XGBoost – open source library for decision tree-based ML algorithms.
- *Complete ecosystem: consult the official documentation.*

Currently, RAPIDS has integration with hundreds of different services, ranging from open source to commercial software. The list grows every year. Some are official integrations (developed by the RAPIDS team) and others from the community.

ENTERPRISE USE CASES

Walmart

– implemented XGBoost to increase forecast accuracy and save money

Image source: Sundry Photography /Shutterstock

AT&T

– Powered several data science operations which then resulted in gains in efficiency

Amazon

– implemented Graph Neural Networks (GNN) for applications including drug discovery, recommender systems, fraud detection, and cybersecurity

Amazon and NVIDIA have built a partnership on developing the Enterprise Deep Graph Library (DGL) for large-scale GNNs

CONTENT

Libraries that can be easily integrated with common data science pipelines:

- **cuDF**
- **cuML**
- **cuPy**

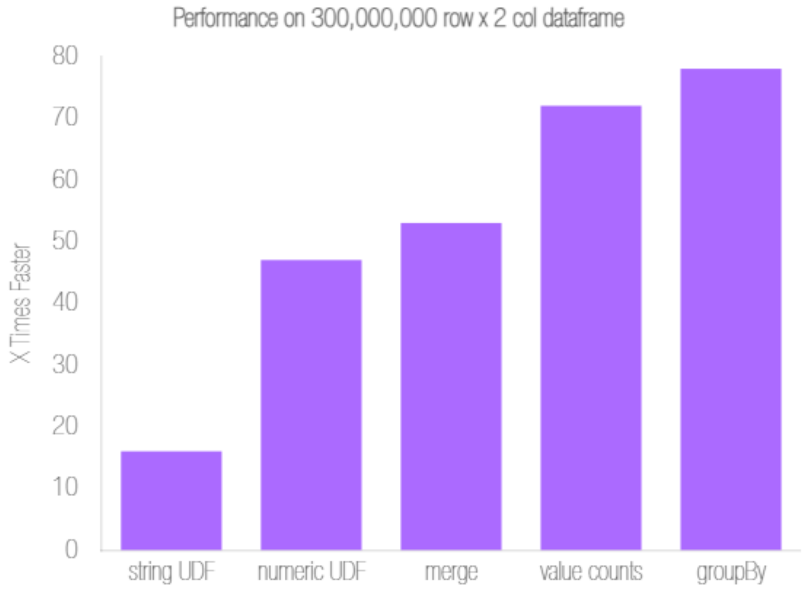
cuDF

- **cuDF** is a package in the RAPIDS ecosystem that allows you to easily migrate your Pandas workflows from CPU to GPU.
- It was built based on the Apache Arrow columnar memory format. It is a GPU DataFrame library used to load, join, aggregate, filter and manipulate data.
- The calculations can take advantage of the immense parallelization provided by GPUs.
- By providing a Pandas-like API, developers and data scientists can easily accelerate their workflows without going into the details of CUDA programming.

cuDF - COMPARISON

- The operations are almost the same as Pandas and can easily replace Pandas operations in a traditional data science pipeline.
- While results may vary slightly on different GPUs, it should be clear that GPU acceleration can make a significant difference.
- We can get much faster results with the same code.

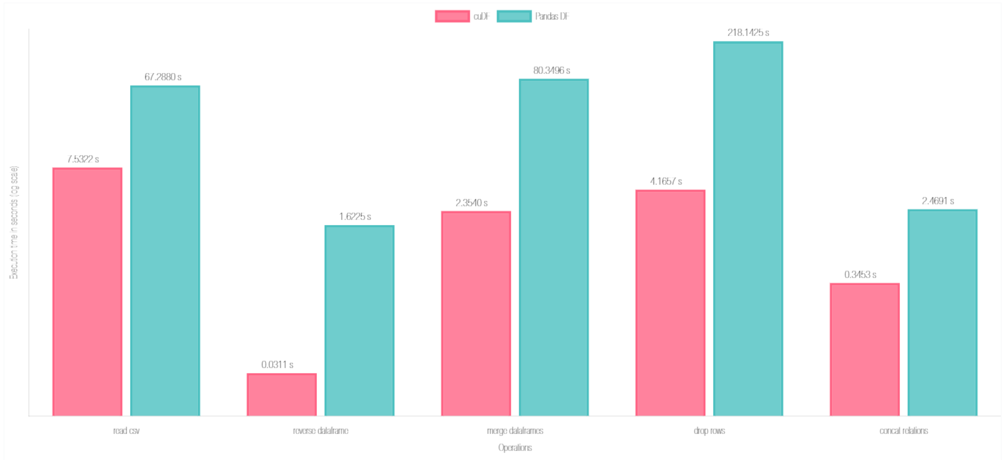
Benchmark on:
- AMD EPYC 7642 (using 1x 2.3GHz CPU core) with 512GB;
- NVIDIA A100 80GB (1x GPU) with pandas v1.5 and cuDF v23.02.



Source: rapids.ai

cuDF - COMPARISON

- Another comparison can be seen in the chart on the right. The *California road network* dataset *(Leskovec 2009)* was used



Operation	**cuDF (s)**	**Pandas DF (s)**	**Speedup**
Read CSV	7.532238	67.287993	8.9x
Reverse DF	0.031103	1.622508	52.2x
Merge DFs	2.354040	80.349599	34.1x
Drop column and rows	4.165711	218.142479	52.4x
Concat DFs	0.345340	2.469050	7.1x

Source: Ahmedur Shovon

PANDAS vs. cuDF

Create a DataFrame and add the values of a specific column

Pandas code

```
import pandas as pd
```

```
df = pd.DataFrame()
df['id'] = [0, 1, 2, 2, 3, 3, 3]
df['val'] = [float(i + 10) for i in range(7)]
print(df)
```

```
soma = df['val'].sum()
print(soma)
```

```
91.0
```

cuDF code

```
import cudf
```

```
df = cudf.DataFrame()
df['id'] = [0, 1, 2, 2, 3, 3, 3]
df['val'] = [float(i + 10) for i in range(7)]
print(df)
```

```
soma = df['val'].sum()
print(soma)
```

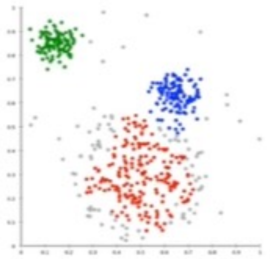
```
91.0
```

cuML

- **cuML** is a suite of fast and GPU-accelerated machine learning algorithms designed for data science and analytical tasks. It is based on scikit-learn.
- Supports a wide range of algorithms such as: regression, classification and clustering.
- cuML takes advantage of the parallel processing power of GPUs for fast performance with high scalability.
- Reduces the time required to develop and deploy Machine Learning models.
- By increasing the training speed we can increase the productivity of data scientists and ML engineers.

cuML

- Some of the methods and algorithms supported by cuML:



Classification / Regression	Decision Trees / Random Forests Linear/Lasso/Ridge/LARS/ElasticNet Regression Logistic Regression K-Nearest Neighbors (exact or approximate) Support Vector Machine Classification and Regression Kernel regression Naive Bayes
Inference	Random Forest / GBDT Inference (FIL)
Preprocessing / Metrics	Text vectorization (TF-IDF / Count) Target, Label and OneHot Encoding Scaling and Imputers Cross-validation / splitting 22+ metrics
Clustering Decomposition & Dimensionality Reduction	K-Means DBSCAN and HDBScan Spectral Clustering Principal Components (including iPCA) Singular Value Decomposition UMAP Spectral Embedding T-SNE
Time Series	Holt-Winters Seasonal ARIMA / Auto ARIMA
Cross Validation	
Hyper-parameter Tuning	

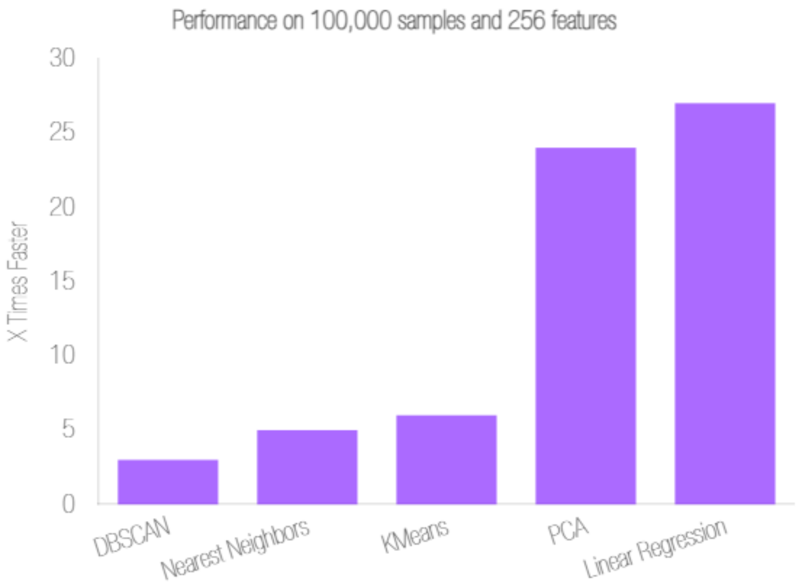
- Note: this list can easily become outdated because RAPIDS is frequently launching new updates, so the support for new methods may be included in next versions.

cuML – COMPARISON

- cuML brings considerable speedups to ML modeling with an API that matches scikit-learn
- The GPU-based implementations can be 10-50x faster than CPU.
- Like cuDF, when we change to cuML it is possible to obtain results much faster and still using the same code (or very similar).

Benchmark on:

- AMD EPYC 7642 (using 1x 2.3GHz CPU core) with 512GB;
- NVIDIA A100 80GB (1x GPU) with scikit-learn v1.2 and cuML v23.02.



Source: rapids.ai

SCIKIT-LEARN vs. cuML

Implementing a Linear Regression for Empirical Data

Scikit-learn code

```
import sklearn
from sklearn.linear_model import LinearRegression
```

```
linear_regression = LinearRegression()
```

```
linear_regression.fit(np.expand_dims(x, 1), y_noisy)
```

```
inputs = np.linspace(start=-5, stop=5, num=1000000)
```

```
outputs = linear_regression.predict(np.expand_dims(inputs, 1))
```

cuML code

```
import cuml
from cuml.linear_model import LinearRegression as LinearRegressionGPU
```

```
linear_regression_gpu = LinearRegressionGPU()
```

```
linear_regression_gpu.fit(cp.expand_dims(cp.array(df['x']),1), y_noisy)
```

```
inputs = np.linspace(start=-5, stop=5, num=1000000)
df_cudf = cudf.DataFrame({'inputs': inputs})
```

```
outputs_gpu = linear_regression_gpu.predict(df_cudf[['inputs']])
```

EXPERT ACADEMY

RAPIDS AND DASK

- **Dask** is a flexible open-source library for parallel computing in Python that makes scaling out your workflow smooth and simple.
- It is used for distributed processing, allowing this functionality to be easily integrated with RAPIDS.
- On CPU, Dask uses Pandas (NumPy) to perform operations in parallel on DataFrame (array) partitions.
- Since the library abstracts all complex processes, it is not necessary to know about parallelism to implement it.
- The algorithm is responsible for the distribution, synchronization and management.

RAPIDS AND DASK – WHEN TO USE

- If you want to distribute your workflow across multiple GPUs.
- If you have more data than you can fit in memory on a single GPU. For example, when you are dealing with a dataset larger than the available GPU memory.
- When you want to analyze data spread across many files at once.

RAPIDS AND DASK

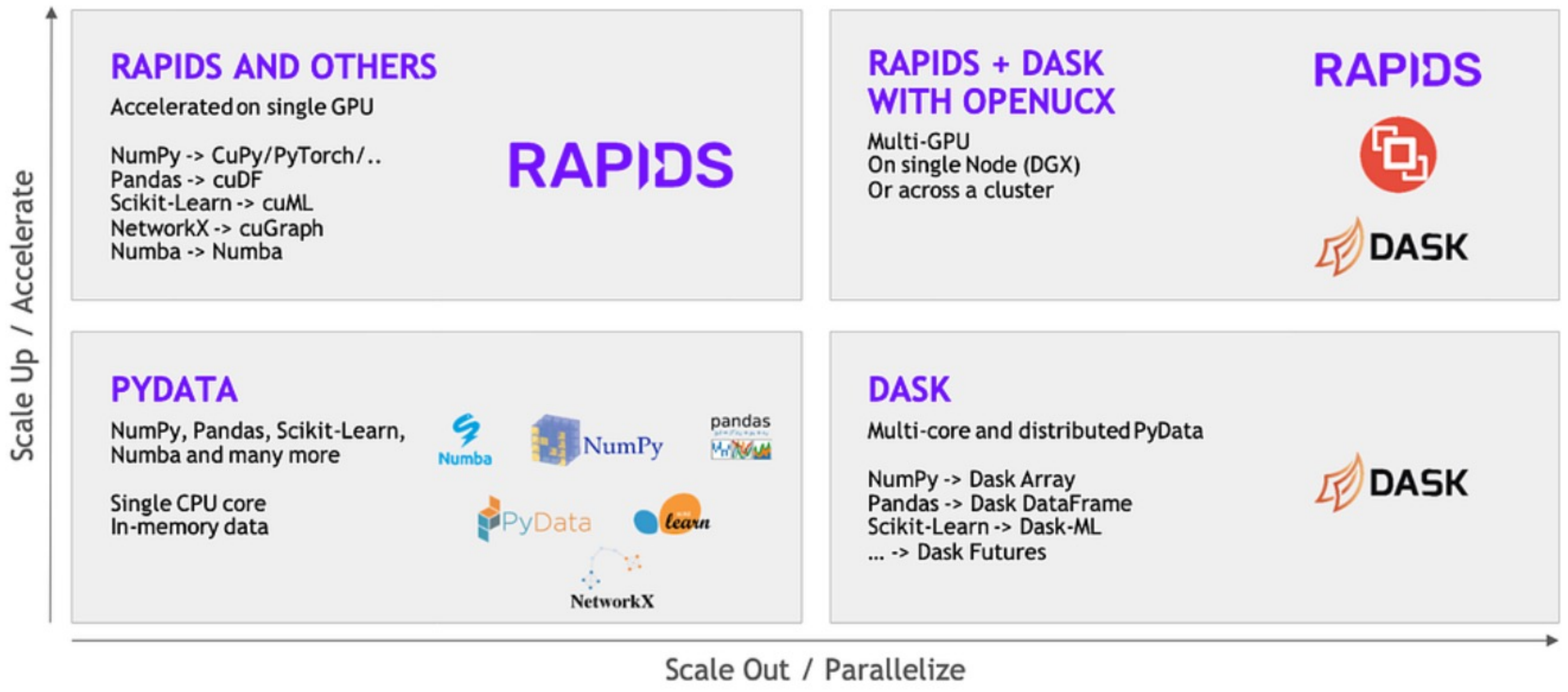
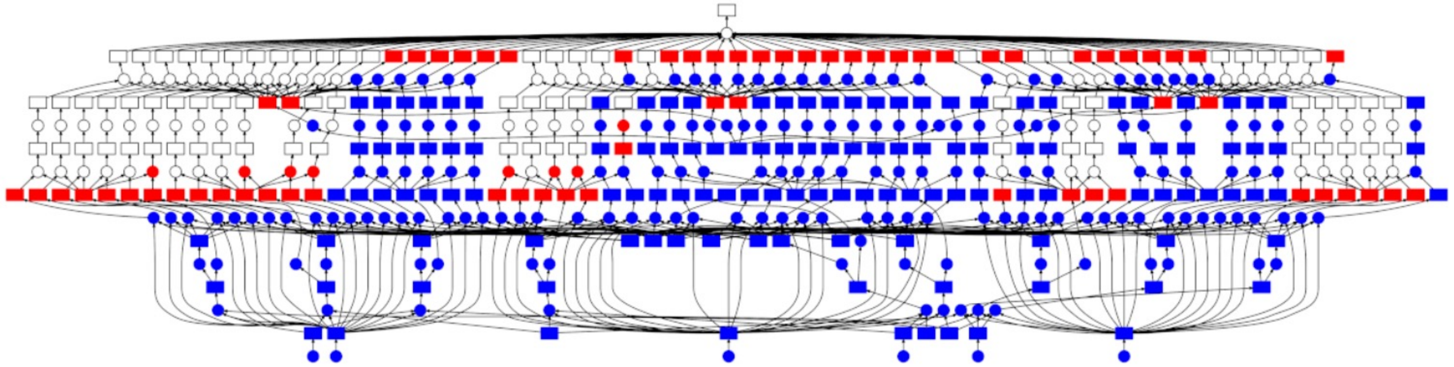


Image source: https://docs.rapids.ai/overview

DASK

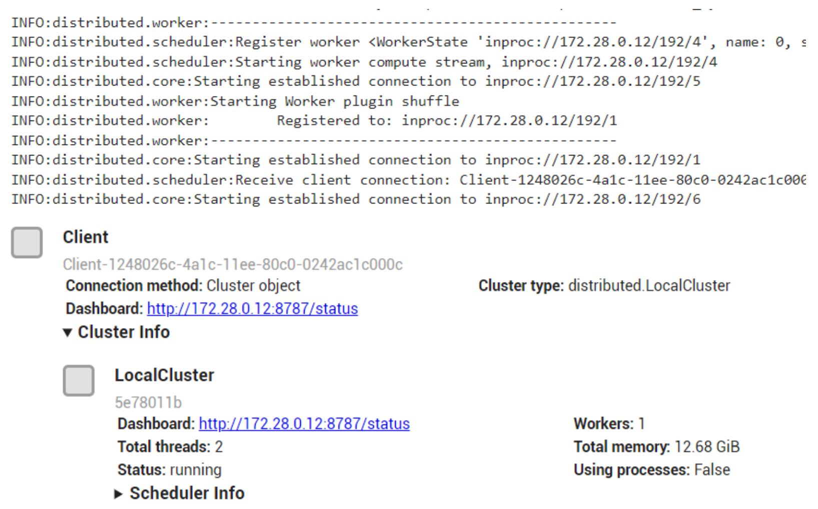
- Dask allows to distribute the work by creating a Directed Acyclic Graph (DAG) representation of your code at runtime. This graph is sent to a scheduler that allocates individual tasks to worker processes. These worker processes can be on a single machine, allowing the use all CPU cores or across many machines. It is possible to scale to hundreds or even thousands of CPU cores.



Source: https://dask.org/

DASK

- Dask operates by creating a cluster composed of a Client and several Workers.
- The client is responsible for scheduling the work
- Workers are responsible for the effective execution of the work. They have two functions:
  - Calculate tasks as indicated by the scheduler.
  - Store and provide computed results to other workers or clients.



```
INFO:distributed.worker:-------------------------------------------------
INFO:distributed.scheduler:Register worker <WorkerState 'inproc://172.28.0.12/192/4', name: 0, s
INFO:distributed.scheduler:Starting worker compute stream, inproc://172.28.0.12/192/4
INFO:distributed.core:Starting established connection to inproc://172.28.0.12/192/5
INFO:distributed.worker:Starting Worker plugin shuffle
INFO:distributed.worker:        Registered to: inproc://172.28.0.12/192/1
INFO:distributed.worker:-------------------------------------------------
INFO:distributed.core:Starting established connection to inproc://172.28.0.12/192/1
INFO:distributed.scheduler:Receive client connection: Client-1248026c-4a1c-11ee-80c0-0242ac1c00
INFO:distributed.core:Starting established connection to inproc://172.28.0.12/192/6
```

Client

Client-1248026c-4a1c-11ee-80c0-0242ac1c000c

Connection method: Cluster object **Cluster type:** distributed.LocalCluster

Dashboard: http://172.28.0.12:8787/status

▼ **Cluster Info**

LocalCluster

5e78011b

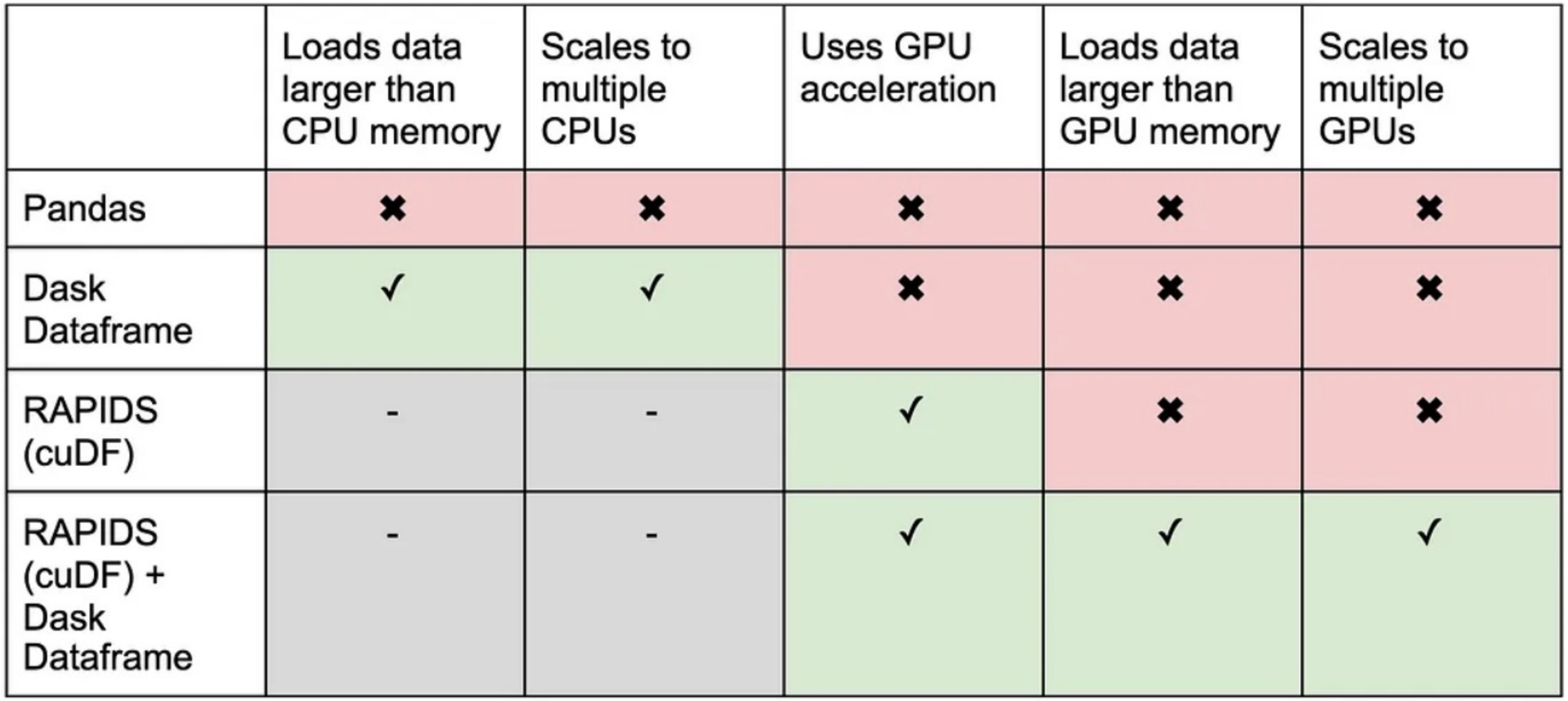
Dashboard: http://172.28.0.12:8787/status **Workers:** 1

Total threads: 2 **Total memory:** 12.68 GiB

Status: running **Using processes:** False

▶ **Scheduler Info**

DASK AND RAPIDS



	Loads data larger than CPU memory	Scales to multiple CPUs	Uses GPU acceleration	Loads data larger than GPU memory	Scales to multiple GPUs
Pandas	✖	✖	✖	✖	✖
Dask Dataframe	✓	✓	✖	✖	✖
RAPIDS (cuDF)	-	-	✓	✖	✖
RAPIDS (cuDF) + Dask Dataframe	-	-	✓	✓	✓

Source: RAPIDS AI – Medium

EXPERT ACADEMY

REFERENCES

Introduction

- https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/
- https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/
- https://www.datanami.com/2019/03/22/how-walmart-uses-gpus-for-better-demand-forecasting
- https://www.nvidia.com/en-us/on-demand/session/gtcfall22-a41235/
- https://www.nvidia.com/en-us/on-demand/session/gtcfall22-a41386/

RAPIDS

- https://rapids.ai/ecosystem/#overview
- cuDF - https://docs.rapids.ai/api/cudf/stable/
- cuML - https://docs.rapids.ai/api/cuml/stable/

Dask

- https://docs.dask.org/en/stable/gpu.html
- https://medium.com/rapids-ai