Presentation

Boosting Python performance with GPU-accelerated libraries

Nguyen Tiet Nguyen Khoi
Data Scientist Trainee

Content

1/ Review: Parallel programming with GPU & CUDA

2/ Numba for Python: CPU-accelerated & GPU-accelerated.

3/ Demo 1: Numba example - CPU & GPU acceleration

4/ Dask & RAPIDS (cudf, cuml): Introduction and comparison to Numpy, Pandas, and sklearn

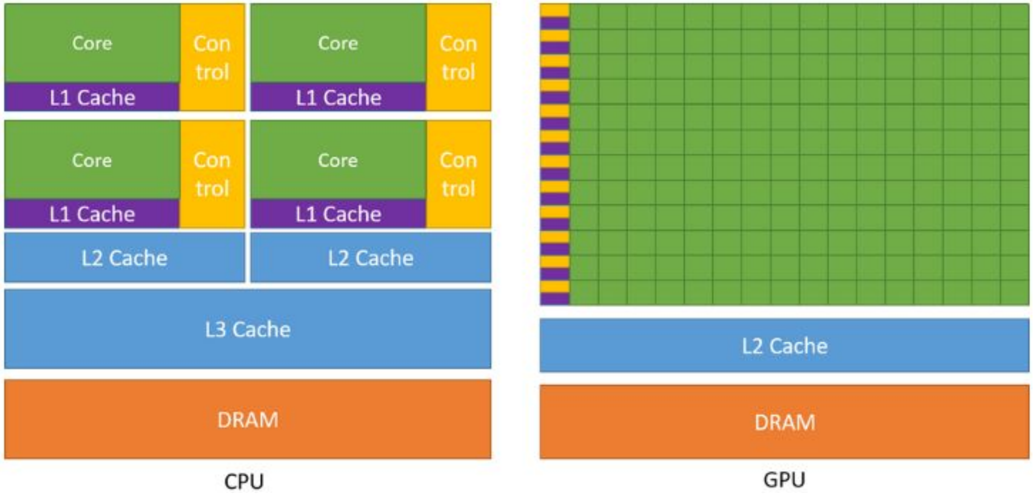
5/ Demo 2: Dask, cudf, cuml

Review: Parallel programming with GPU/CUDA

GPU enables parallel processing that speed-up several tasks.

From ~ 10-1000x faster than pure Python.

CUDA is a parallel computing platform and programming model for GPU-accelerated program.



Review: Parallel programming with GPU/CUDA

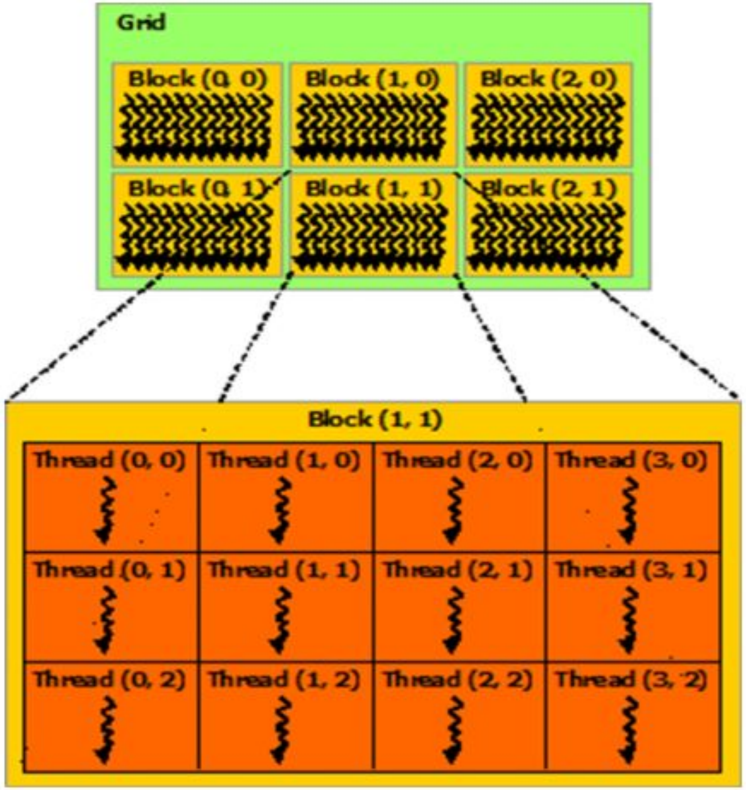
Components inside a CUDA-supported GPU: Grids -> Blocks -> Threads.

Maximum threads per block is **1024**.

Maximum blocks per grid is **65535** blocks in a **single 1-dimensional grid**.

There can be 2D & 3D grids as well.

A GPU-accelerated function to be executed is called a **kernel**.



Review: Parallel programming with GPU/CUDA

GPU-accelerated Python libraries

Compilation with Numba

A Python to LLVM compiler

JIT compiles numeric Python code to C speeds

We can have for loops again!

Parallelism with Dask

A Dynamic task scheduler

Runs Python task graphs on distributed hardware

MPI. But easier and slower
Spark. But more flexible and without the JVM!

GPUs - RAPIDS, CuPy

CUDA-backed GPU libraries

Like NumPy/Pandas/Scikit-Learn, but backed by CUDA code

Python helped you to forget C
Now you can forget CUDA too!

UCX

High Performance Networking

Provides interfaces and routing to high performance networking libraries like InfiniBand and NVLink

Because once computation is fast we need to focus on everything else

Numba for Python: Introduction

Switching down to C/C++/CUDA in these cases can be challenging, especially for Python developers.

=> **Numba** is the solution.

Numba can be applied for both CPU & GPU speed-up.

Installation:

```
$ conda install cudatoolkit
```

or:

```
$ pip install cuda-python
```

Numba for Python: Introduction

Important terms:

- *host*: the CPU
- *device*: the GPU
- *host memory*: the system main memory
- *device memory*: onboard memory on a GPU card
- *kernels*: a GPU function launched by the host and executed on the device
- *device function*: a GPU function executed on the device which can only be called from the device (i.e. from a kernel or another device function)

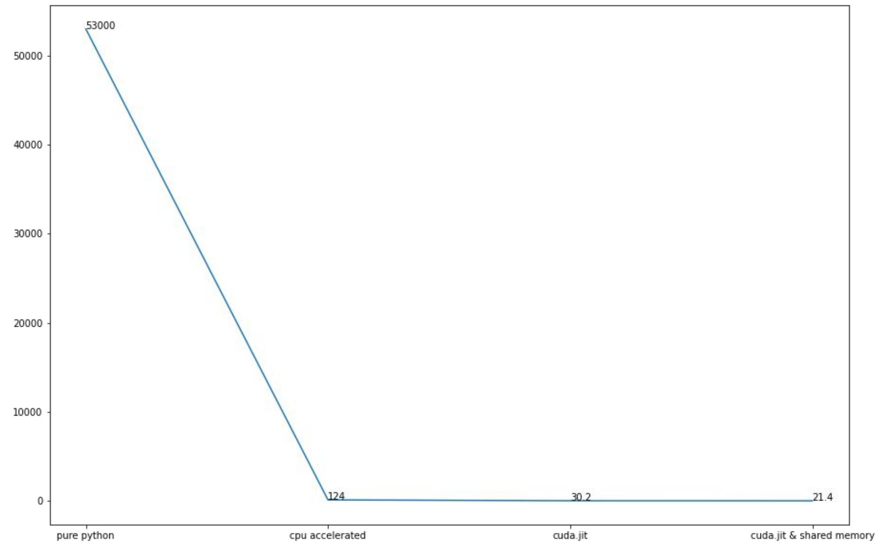
Numba for Python: Introduction

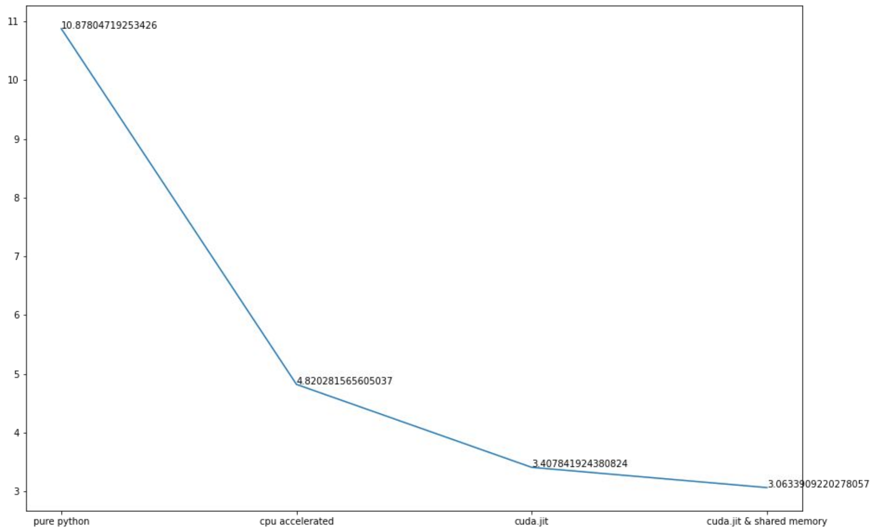
Numba supports 3 kinds of GPU memory:

- Global device memory (the large, relatively slow off-chip memory that’s connected to the GPU itself).
- On-chip shared memory.
- Local memory.

Initialization and assignment with **local memory** is faster than **device memory**.

Numba for Python: Performance comparison





Matrix multiplication (in ms)

Numba for Python: Performance comparison

The performance of each implementation is:

Pure Python: **55.3 ms**

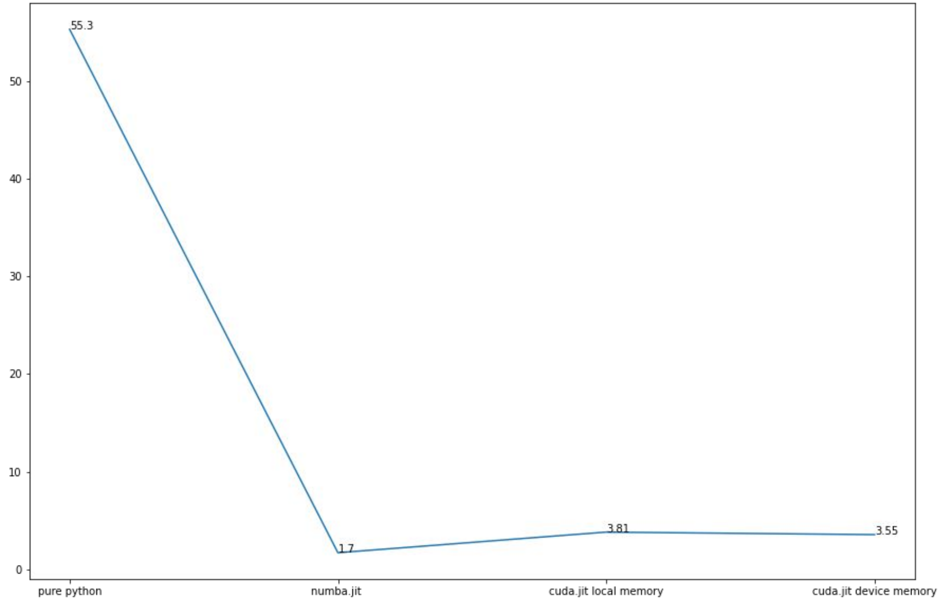
Python with CPU-accelerated: **1.7 ms**

Cuda.jit using local memory with copy_to_host: **3.93 ms**

Cuda.jit using device memory with copy_to_host: **3.55 ms**

Cuda.jit using local memory: **331 µs**

Cuda.jit using device memory: **83.8 µs**



Quick-sort (in ms)

Demo 1

Numba example: CPU & GPU acceleration

Dask & RAPIDS (cudf, cuml): Introduction

Rising trend for Python modules:

- Similar syntax & APIs for all Python modules.
- Programmers can change and work with different data-type from different libraries.
- Example: matmul with a numpy matrix and a pytorch tensor.

-> Dask & RAPIDS were built on top of Numpy, Pandas, Sklearn to speed-up the Python program by enabling parallelization.

RAPIDS DASK

Dask & RAPIDS (cudf, cuml): Introduction

Dask vs. RAPIDS

Dask allows you to **scale out** your workload onto **multiple processors and multiple machines**.

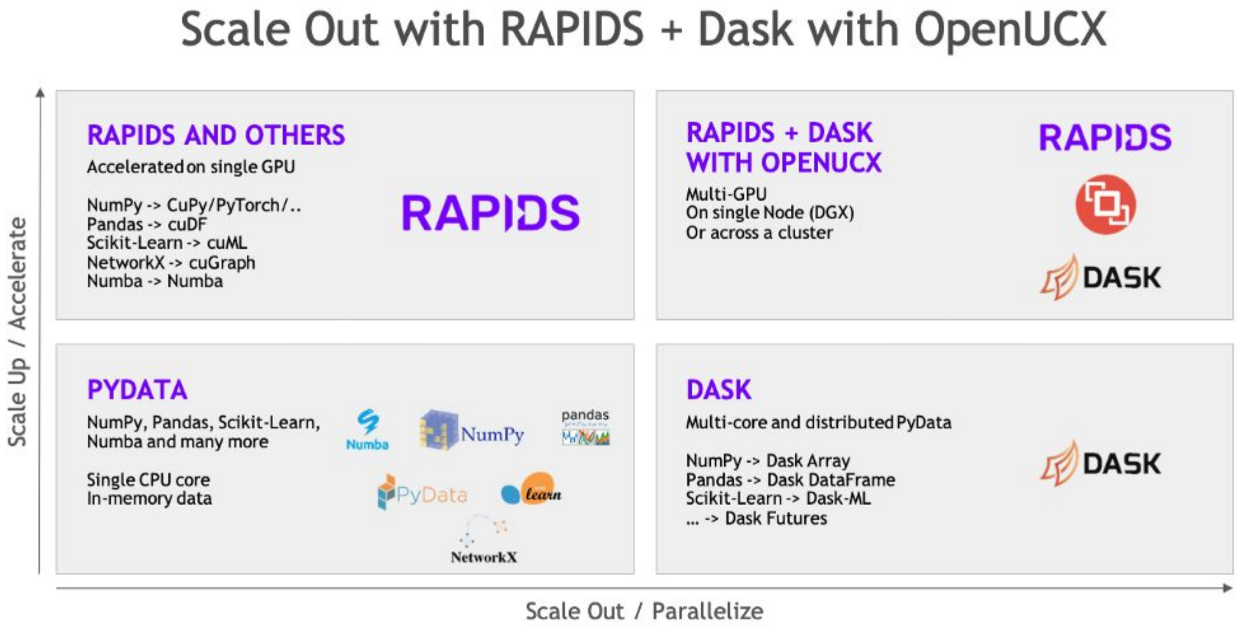
Dask creates an abstract representation of your code and then distributing that onto a cluster.

RAPIDS allows you to **scale up** your workload by **reimplementing low-level parts** of **common open source APIs**.

Run **RAPIDS** on NVIDIA GPUs giving you faster execution.

Dask & RAPIDS (cudf, cuml): Introduction

Dask vs. RAPIDS:



Dash & RAPIDS (cudf, cuml): Introduction

Dask vs. 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	-	-	✓	✓	✓

Dask: Parallel task scheduler for Python

Dask enables parallel computing in Python.

- Focuses on **scaling out** and **deployability**.
- Can scale out to thousand-node clusters.
- Easy to install and use.
- Easy to deploy on Hadoop/Spark, Kubernetes, etc.

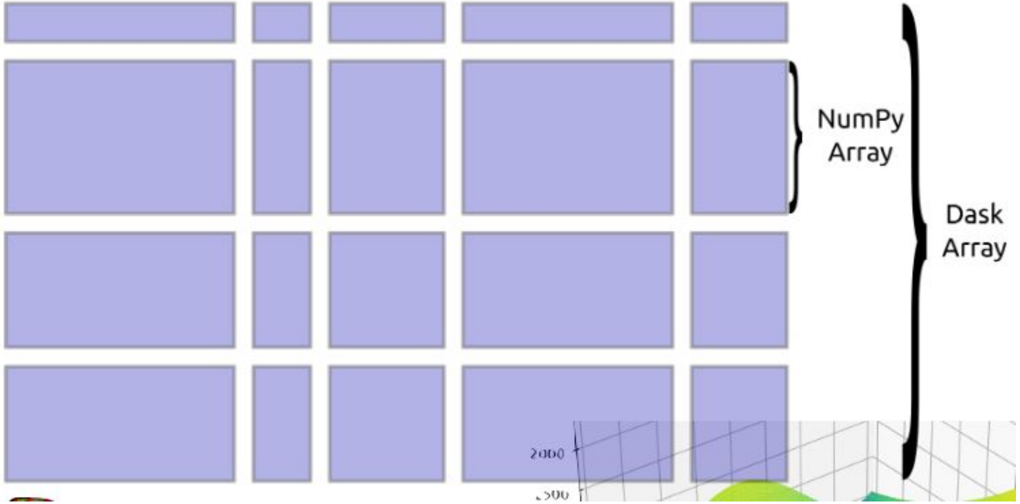
Dash: Parallel task scheduler for Python

Parallel NumPy:

Same API as Numpy.

One Dask array is built from multiple Numpy arrays.

```
import dask.array as da
x = da.from_hdf5(...)
x + x.T - x.mean(axis=0)
```



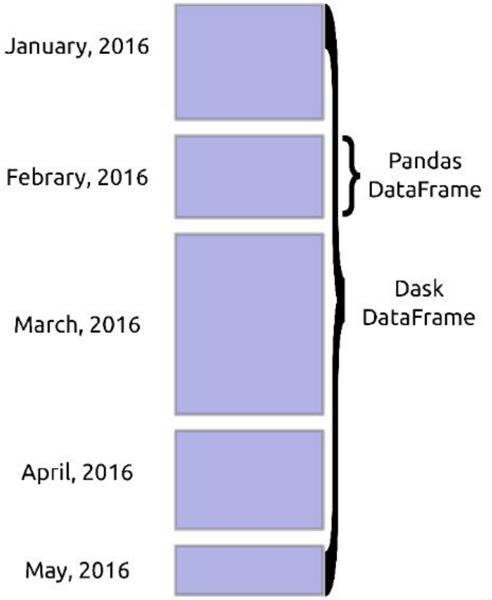
Dash: Parallel task scheduler for Python

Parallel Pandas:

Same API as Pandas.

One Dask dataframe is built from multiple Pandas dataframe..

```
import dask.dataframe as dd
df = dd.read_csv(...)
df.groupby('name').balance.max()
```



Dash: Parallel task scheduler for Python

Parallel Sklearn:

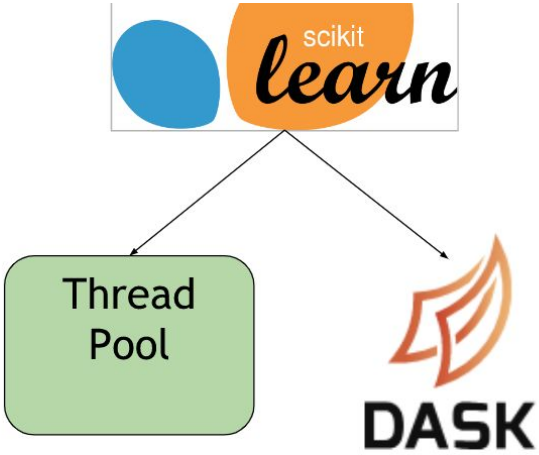
Same API as Sklearn.

Same code, just wrap with a decorator.

Allowing scaling onto clusters.

Available for most Sklearn where joblib is used.

```
from scikit_learn.externals import joblib
with joblib.parallel_backend('dask'):
    estimator = RandomForest()
    estimator.fit(data, labels)
```



RAPIDS: Collection of GPU-accelerated libraries

RAPIDS includes Python libraries:

- Cudf: Pandas-alike
- Cuml: Sklearn-alike

RAPIDS store data on GPU memory (different from Sklearn & Pandas)

Computation using GPU.

```
import cudf
gdf = cudf.read_csv("/path/to/my/data-*.csv")  # Your CSV data must be
smaller than your GPU memory
```

Performance comparison

Importing data (2m rows x 12 cols):

- Pandas: **18s**
- Dask (dask.dataframe): **88ms**
- RAPIDS (cudf): **78ms**

Perform clustering and K-selection (2m rows x 12 cols):

- Sklearn: **3min 24s**
- RAPIDS (cuml): **10.7s**

Demo 2

Dask & RAPIDS examples

Thank you for listening!