Week 13.3: GPU Frameworks

DS-GA 1004: Big Data

This week

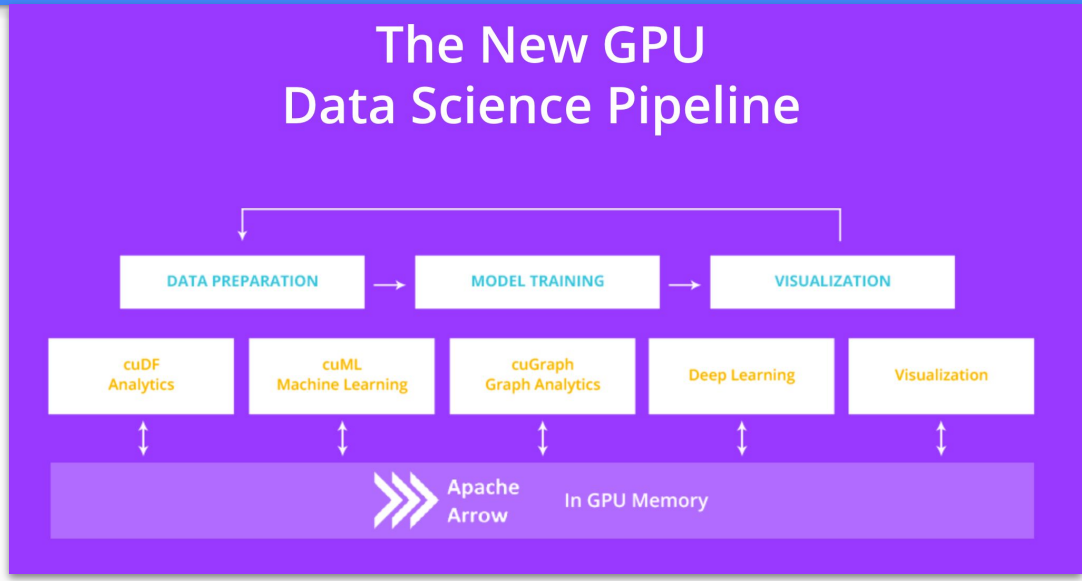
- Graphics processing units (GPUs)
- GPGPUs and CUDA
- **Software frameworks**

What tools should I use?

- Deep learning?
  - CUDA/CuDNN ⇐ (TensorFlow | PyTorch | pick your favorite…)
- Traditional ML and DataFrame processing
  - **RAPIDS**/cuDF
- Custom analysis / writing your own kernels
  - Raw cuda (if you like writing C)
  - **numba** (if you like writing Python)

RAPIDS

- Uses **Arrow** for in-memory **column-oriented storage** on the GPU
- Implements dataframe operations on the GPU as well
- Minimizes overall data transfer
- Integrates well with Dask



https://github.com/rapidsai/cudf

Numba example

```
from numba import cuda
import numpy as np
```

```
@cuda.jit
def axpy(r, a, x, y):
        i = cuda.grid(1)    # 1-dimensional grid
        if i < len(r):
                r[i] = a * x[i] + y[i]
```

```
N = 10000
a = 5.0
x = np.arange(N).astype(np.int32)
y = np.random.randint(100, size=N).astype(np.int32)
r = np.zeros_like(x)

threads = 256
blocks = (N // threads) + 1
axpy[blocks, threads](r, a, x, y)
```

- Looks similar to cuda, but written in Python
- Numba **compiles to GPU code** at run-time
- Can be a good option if you know exactly what you want to do in GPU terms, but don't want to deal with nvcc or C++

In general, numba is a just-in-time compiler and alternate implementation of the numpy API.

By default it compiles to LLVM, and is a great way to make Python/numpy code go faster without much effort!

Life of a Numba Kernel - with - output.ipynb

GPU evolution / resources available on NYU HPC (Greene cluster)

Year	Product	# Cores	RAM (GB)	# FLOPS (SP)	Memory bandwidth (GB/sec)	**# In Greene Cluster**
1997	RIVA 128	-	0.004	-	1.6	
2006	GeForce 8800	128	1	250 - 350G	64	
2009	Fermi	512	6	1.5T	192	
2013	Kepler (K40)	2880	12	4.2-5T	288	
2014	Kepler (K80)	4992	2x 12	5.5-8.7T	2x 240	
2016	GTX 1080	2560	8-11	8-10T	320	
2016	Pascal (P40)	3840	24	11T	345	
2016	Pascal (P100)	3584	16	~10T	732	
2017	Volta (V100)	5120	32	14T	900	76
2018	Turing (RTX8000)	4608+	48	16.3T	672	292

https://sites.google.com/nyu.edu/nyu-hpc/hpc-systems/greene/hardware-specs

Summary

- GPU architectures provide massive parallelism for simple operations
- Many of the programming ideas we've seen in other contexts show up in CUDA programming
  - Independent mapper programs
  - Local memory access
  - Alternating **serial** and **parallel** processes
  - **Minimizing communication!**
- For jobs too big for a single CPU, but not big enough for a cluster, GPUs can be a cost-effective solution