Sören Metje

GPU Computing with Python

Table of contents

Motivation

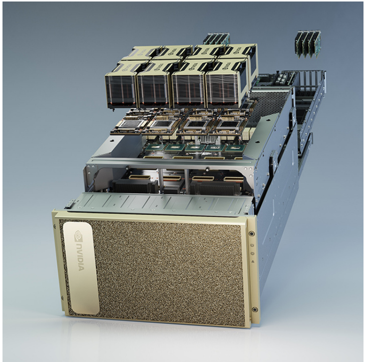
Why Python?

- High-level programming
- Compatible with many platforms and systems
- Many high quality frameworks and libraries
- Widely distributed in many different domains
- Big community

Motivation

Why GPU Computing?

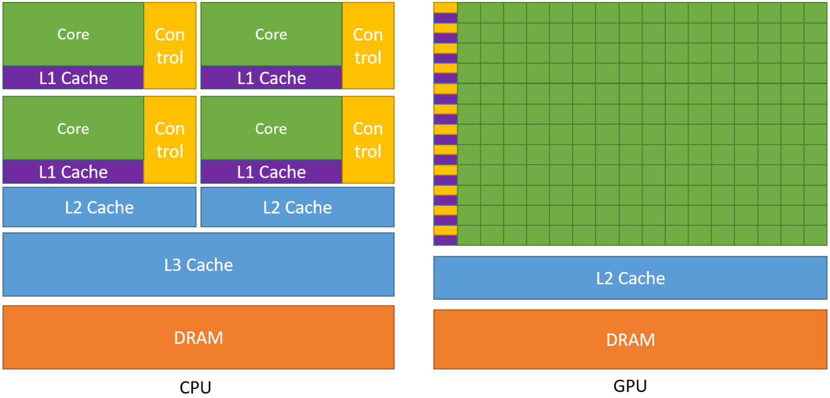
- Reduce wall-clock time
- Achieve higher cost-efficiency



Source: NVIDIA - GPU-Accelerated Google Cloud [NVId]

Outline

GPU Architecture



Source: CUDA Toolkit Documentation [NVIa]

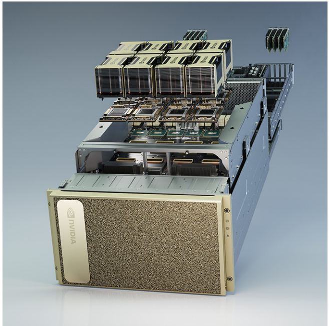
Use Cases

When to use GPU Computing?

- Large data set available
- Parallel processing possible
- Use cases: Fluid dynamics, Image processing, Deep leaning, ...

When **not** to use GPU Computing?

- Data set is too small
- Data set is too big (exceeds GPU memory size)
- Large amount of small sequential operations



Source: NVIDIA - GPU-Accelerated Google Cloud [NVId]

CUDA

Definition

NVIDIA CUDA (Compute Unified Device Architecture)
is a **parallel computing platform**
and programming model for general computing on **GPUs**.

- Initial release: June 23, 2007
- Gives access to the GPU's virtual instruction set
- Enables execution of compute kernels
- Accessible through frameworks, libraries, and compiler directives
- Closed source

CUDA Compute Kernel

Definition

A compute kernel is a **function** compiled for accelerators (such as GPUs).

C++

```cpp
__global__ void VecAdd(float* A, float* B, float* C) {
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main() {
    // ...
    VecAdd<<<1, N>>>(A, B, C);  // blocks per grid, threads per block
}
```

Compute Kernel Limitations

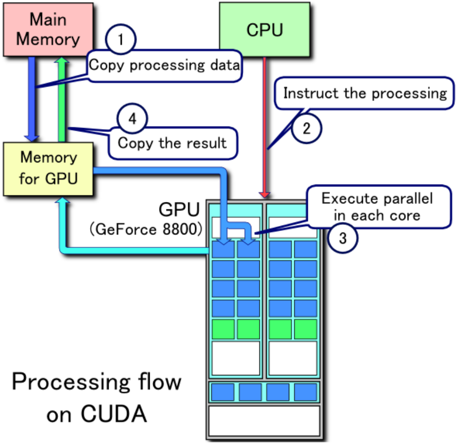
- Allowed operations: basic math operations, if / else, for / while loops
- Can not explicitly return a value
- Write results to passed array

C++

```cpp
__global__ void VecAdd(float* A, float* B, float* C) {
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main() {
    // ...
    VecAdd<<<1, N>>>(A, B, C);  // blocks per grid, threads per block
}
```

CUDA Processing Flow



Source: Wikipedia - CUDA [Wika]

AMD ROCm

Definition

AMD ROCm (Radeon Open Compute) is a software stack for **GPU programming**.

- "NVIDIA CUDA for AMD GPUs"
- Initial release: November 14, 2016
- Available on GitHub (open-source)
- Supported by:
  - PyTorch
  - TensorFlow
  - CuPy
- Not as widely supported as NVIDIA CUDA
- Gaining traction in the TOP500

TOP 500 List

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	1,110,144	151.90	214.35	2,942
4	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096

Source: TOP 500 List - June 2022 [TOP]

Outline

Python Frameworks for GPU Computing

Computing Frameworks

- Numba
- CuPy
- Scikit-cuda
- RAPIDS
- Triton (presented by Dimitris Oikonomou on 2022-05-19)
- ...

Deep Learning Frameworks

- PyTorch
- TensorFlow
- Keras
- ...

Python Frameworks for GPU Computing

Computing Frameworks

- Numba
- CuPy
- Scikit-cuda
- RAPIDS
- Triton (presented by Dimitris Oikonomou on 2022-05-19)
- ...

Deep Learning Frameworks

- PyTorch
- TensorFlow
- Keras
- ...

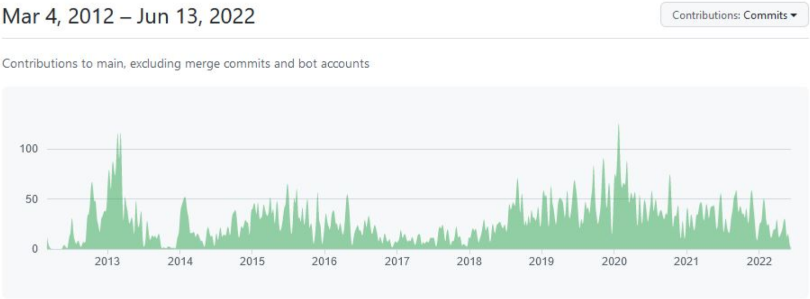
Numba

Definition

Numba is a **just-in-time compiler** for numerical functions in Python.

- Translates Python functions to optimized machine code at runtime
- Compiles code for CPU and GPU
- Supports
  - NVIDIA GPUs: CUDA
  - ~~AMD GPUs: ROCm~~ (deprecated)
- Available on GitHub (open-source)

Numba - Development



Source: GitHub - Numba [Gita]

Numba - Programming Approaches

2 Approaches for GPU Programming

- Universal functions
- CUDA Kernels

Numba - Universal Functions

Definition

A universal function (or ufunc for short) is a **function** that operates on arrays in an element-by-element fashion.

Python

```python
from numba import vectorize

@vectorize(['float32(float32, float32)'])
def add_ufunc(x, y):
    return x + y

n = 100000
a = np.arange(n).astype(np.float32)
b = 2 * a
out = add_ufunc(a, b)
```

Numba - Universal Functions on GPU

Python

```python
from numba import vectorize

@vectorize(['float32(float32, float32)'], target='cuda')
def add_ufunc(x, y):
    return x + y
```

1. Compile CUDA kernel
2. Allocate GPU memory
3. Copy data to the GPU
4. Executed CUDA kernel
5. Copy result back to the CPU
6. Return the result

Numba - CUDA Kernels

Python

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

@cuda.jit
def add_kernel(x, y, out):
    start, stride = cuda.grid(1), cuda.gridsize(1)  # 1 = one dim. thread grid
    for i in range(start, x.shape[0], stride):
        out[i] = x[i] + y[i]

x = np.arange(100000).astype(np.float32)
y = np.arange(100000).astype(np.float32)
out = np.empty_like(x)
add_kernel[30, 128](x, y, out)  # blocks per grid, threads per block
print(out)  # [0.0 2.0 4.0 ... 1.99998e+05]
```

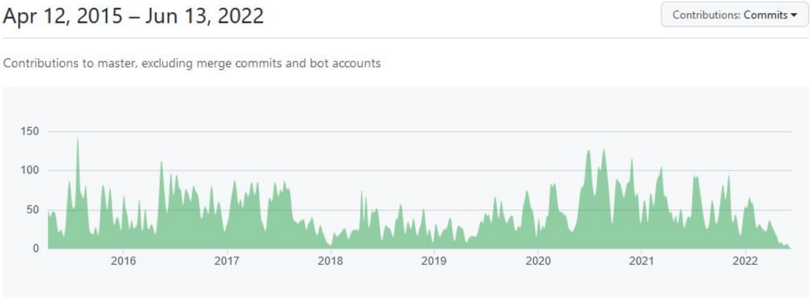
CuPy

Definition

CuPy is a NumPy/SciPy-compatible **array library** for GPU-accelerated computing.

- "NumPy for GPU computing"
- Provides various math operations
- Supports
  - NVIDIA GPUs: CUDA
  - AMD GPUs: ROCm (experimental)
- Available on GitHub (open-source)

CuPy - Development



Source: GitHub - CuPy [Gitb]

CuPy - Math Functions

Python

```python
import cupy as cp

x = cp.arange(100000)
y = x * 2

out1 = cp.add(x, y)  # array([     0,      3,      6, ..., 299997])
out2 = cp.sum(x)  # array(704982704)
out3 = cp.linalg.norm(x)  # array(18257281.65280911)
# ...
# All common NumPy functions are supported by CuPy
```

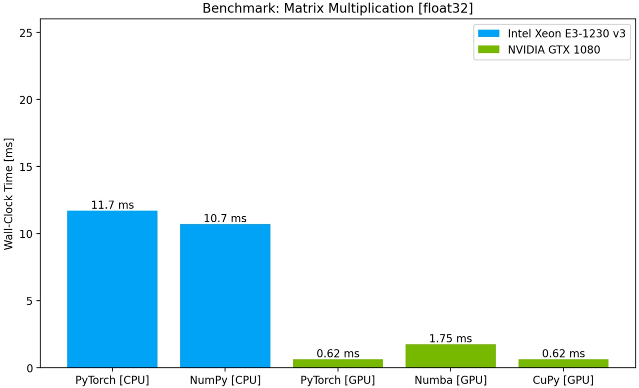
CuPy - CUDA Kernels

Python

```python
import cupy as cp

add = cp.RawKernel(r'''
extern "C" __global__ void add(const int* p, const int* q, int* z) {
    int tid = blockDim.x * blockIdx.x + threadIdx.x;
    z[tid] = p[tid] + q[tid];
 }
''', 'add')
x = cp.arange(100000, dtype=int)
y = cp.arange(100000, dtype=int)
out = cp.zeros(100000, dtype=int)
add((250, 1), (1024, 1), (x, y, out))  # blocks per grid, threads per block
print(out)  # array([      0       2       4 ... 199994 199996 199998])
```

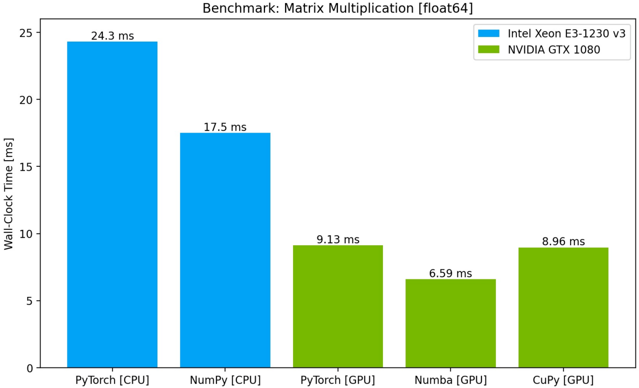
Benchmark - Matrix Multiplication



Wall-clock time in milliseconds for computing result of fast matrix multiplication. Mean of 100 loops and 4 runs.
Multiplied a 1024x2048 matrix and a 2048x512 matrix of float32.
Measured time does not count in time to copy matrices to GPU and result matrix back from GPU.

Code for benchmarking is available on GWDG GitLab

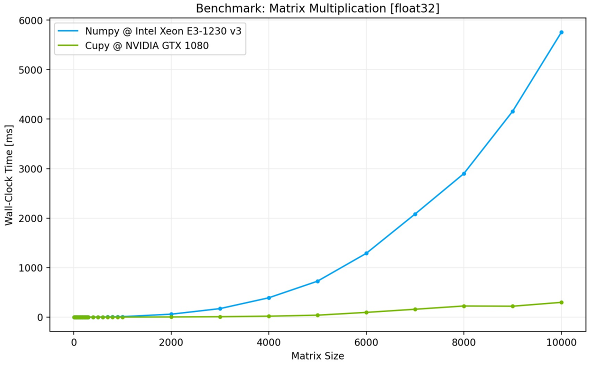
Benchmark - Matrix Multiplication Float64



Wall-clock time in milliseconds for computing result of fast matrix multiplication. Mean of 100 loops and 4 runs.
Multiplied a 1024x2048 matrix and a 2048x512 matrix of float64.
Measured time does not count in time to copy matrices to GPU and result matrix back from GPU.

Code for benchmarking is available on GWDG GitLab

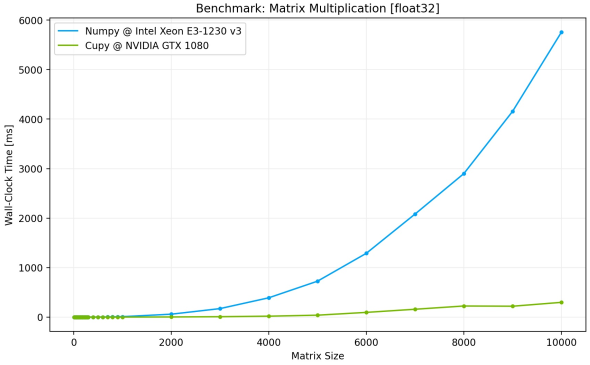
Benchmark - Matrix Multiplication



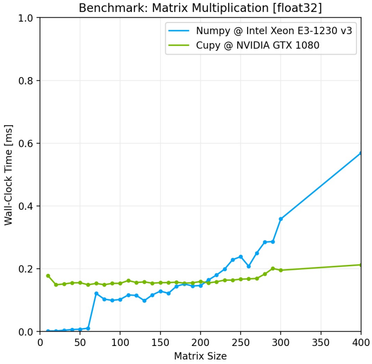
Wall-clock time in milliseconds for computing result of fast matrix multiplication. Mean of 10 loops and 3 runs. Multiplied 2 square matrices of float32 with given size. Not measured time to copy matrices to and from GPU.

Code for benchmarking is available on GWDG GitLab

Benchmark - Matrix Multiplication



Wall-clock time in milliseconds for computing result of fast matrix multiplication. Mean of 10 loops and 3 runs.
Multiplied 2 square matrices of float32 with given size. Not measured time to copy matrices to and from GPU.



Code for benchmarking is available on GWDG GitLab

Outline

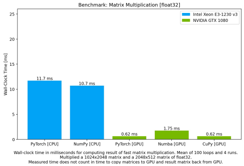
New Trends

What’s new?

- GPUs get more powerful
- More complex models and computations
- New high-level Python frameworks provide features for various use cases
- ROCm (open-source) is gaining traction

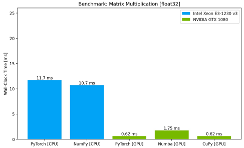
Summary

- **GPUs**
  - achieve massive data parallelism
  - reduce **wall-clock time**
  - increase **cost efficiency**



Summary

- **GPUs**
  - achieve massive data parallelism
  - reduce **wall-clock time**
  - increase **cost efficiency**

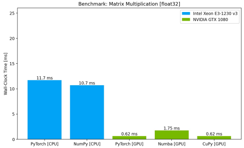


Wall-clock time in milliseconds for computing result of fast matrix multiplication. Mean of 100 loops and 4 runs.
Multiplied a 1024x2048 matrix and a 2048x512 matrix of float32.
Measured time does not count in time to copy matrices to GPU and result matrix back from GPU.

- **Numba** Advantages
  - Enables implementing **own universal functions & kernels** in Python running on GPU

Summary

- **GPUs**
  - achieve massive data parallelism
  - reduce **wall-clock time**
  - increase **cost efficiency**



Wall-clock time in milliseconds for computing result of fast matrix multiplication. Mean of 100 loops and 4 runs.
Multiplied a 1024x2048 matrix and a 2048x512 matrix of float32.
Measured time does not count in time to copy matrices to GPU and result matrix back from GPU.

- **Numba** Advantages
  - Enables implementing **own universal functions & kernels** in Python running on GPU
- **CuPy** Advantages
  - Easily move **existing NumPy code** towards GPU computing
  - Directly use NumPy-style **array operations** and execute them on GPU
  - Great starting-point to **learn** about GPU computing

References I

- Avimanyu Bandyopadhyay. *Hands-On GPU Computing with Python*. Packt Publishing Ltd, 2019. ISBN: 9781789341072.
- NVIDIA. *Programming Guide :: CUDA Toolkit Documentation*. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.
- NVIDIA. *PTX ISA :: CUDA Toolkit Documentation*. URL: https://docs.nvidia.com/cuda/parallel-thread-execution/index.html.
- NVIDIA. *CUDA Zone - Library of Resources | NVIDIA Developer*. URL: https://developer.nvidia.com/cuda-zone.
- NVIDIA. *GPU-Accelerated Google Cloud*. URL: https://www.nvidia.com/en-us/data-center/gpu-cloud-computing/google-cloud-platform/.
- Wikipedia. *CUDA*. URL: https://en.wikipedia.org/w/index.php?title=CUDA&oldid=1089899218.
- Wikipedia. *ROCm*. URL: https://en.wikipedia.org/w/index.php?title=ROCm&oldid=1085428014.
- Wikipedia. *Compute kernel*. URL: https://en.wikipedia.org/w/index.php?title=Compute_kernel&oldid=1008367491.
- Microsoft. *VISC: Virtual Instruction Set Computing*. URL: https://www.youtube.com/watch?v=xM9vEOHf6nI.

References II

- GitHub. *Numba*. URL: `https://github.com/numba/numba`.
- Anaconda. *Numba: A High Performance Python Compiler*. URL: `https://numba.pydata.org/`.
- Anaconda. *Numba documentation*. URL: `https://numba.readthedocs.io/en/stable/index.html`.
- Villoro. *Villoro - numba*. URL: `https://villoro.com/post/numba`.
- GitHub. *CuPy*. URL: `https://github.com/cupy/cupy`.
- Preferred Networks, Inc. and Preferred Infrastructure, Inc. *CuPy NumPy & SciPy for GPU*. URL: `https://docs.cupy.dev/en/stable/index.html`.
- NumPy. *NumPy documentation*. URL: `https://numpy.org/doc/stable/index.html`.
- Mindfire Solutions. *Python: 7 Important Reasons Why You Should Use Python*. URL: `https://medium.com/@mindfiresolutions.usa/python-7-important-reasons-why-you-should-use-python-5801a98a0d0b`.
- Marko Aleksic. *CPU Vs. GPU: A Comprehensive Overview*. URL: `https://phoenixnap.com/kb/cpu-vs-gpu`.

References III

TOP500. *TOP 500 List*. URL: `https://www.top500.org/lists/top500/list/2022/06/`.

Amazon Web Services. *Amazon EC2 P3 instances*. URL: `https://aws.amazon.com/ec2/instance-types/p3/?nc1=h_ls`.

Google. *Google Colab*. URL: `https://colab.research.google.com/`.

GPU as a Service

Instance Size	GPUs - Tesla V100	GPU Peer to Peer	GPU Memory (GB)	vCPUs	Memory (GB)	Network Bandwidth	EBS Bandwidth	On-Demand Price/hr*	1-yr Reserved Instance Effective Hourly*	3-yr Reserved Instance Effective Hourly*
p3.2xlarge	1	N/A	16	8	61	Up to 10 Gbps	1.5 Gbps	$3.06	$1.99	$1.05
p3.8xlarge	4	NVLink	64	32	244	10 Gbps	7 Gbps	$12.24	$7.96	$4.19
p3.16xlarge	8	NVLink	128	64	488	25 Gbps	14 Gbps	$24.48	$15.91	$8.39
p3dn.24xlarge	8	NVLink	256	96	768	100 Gbps	19 Gbps	$31.218	$18.30	$9.64

Example Pricing: Amazon EC2 P3 instances. Source: AWS [Ama]

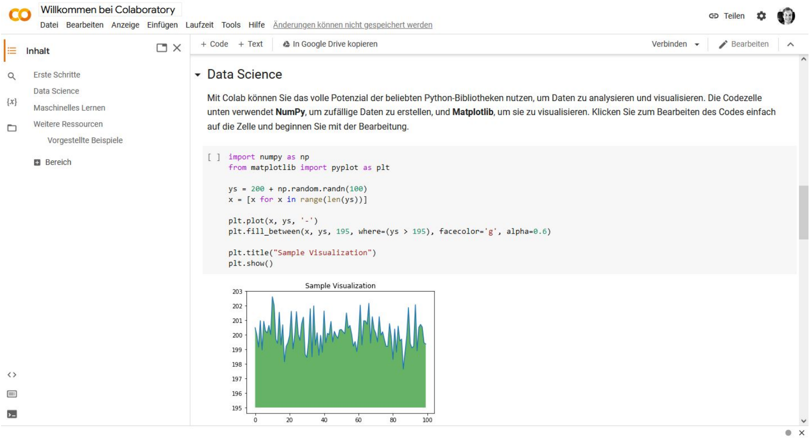
Advantages

- Scalable resources
- On-demand pricing

Disadvantages

- Expense for large periods of time
- Data confidentiality not guaranteed depending on vendor

GPU as a Service



Google Colab Jupyter Notebooks. Source: Google [Goo]

NVIDIA PTX

Definition

NVIDIA PTX (Parallel Thread Execution) is a low-level parallel thread execution **virtual machine** and instruction set architecture (ISA)

- Exposes the GPU as a data-parallel computing device
- Interprets compiled code (analog to Java byte code interpreted by JVM)

Advantage

- Achieves portability of source code among multiple NVIDIA GPUs