HPS

Sadaf Shafi

GPU Computing with Python

Performance comparison of CUDA and CUDA Python

Table of Contents

Introduction

- Many GPU programming frameworks exist, especially for Deep Learning, and most of them still evolving.
- One chooses a framework based on ease of use and implementation versus the efficiency of the frameworks.
- A thorough comparison of these frameworks is needed, which highlights the strengths and weaknesses of these frameworks.
- In this presentation we explore the performance comparison of CUDA and CUDA Python

Hardware and Software

CPU (Central Processing Unit)

- **Architecture:** x86_64
- **CPU op-mode(s):** 32-bit, 64-bit
- **CPU(s):** 2
- **Model name:** Intel(R) Xeon(R) CPU @ 2.20GHz

GPU (Graphics Processing Unit)

- **Name:** Tesla T4
- **Driver Version:** 535.104.05
- **CUDA Version:** 12.2
- **Memory:** 15360 MiB

Hardware and Software (contd)

Memory

- **Total Memory:** 12 GiB
- **Used Memory:** 1.1 GiB
- **Free Memory:** 6.7 GiB
- **Shared Memory:** 1.0 MiB
- **Buffer/Cache:** 4.9 GiB
- **Available Memory:** 11 GiB

Software and Versions

- **NVIDIA CUDA Compiler Driver (nvcc)**
  - Version: 12.2.140
  - Build Date: August 15, 2023
  - Release: 12.2
- **Numba**
  - Version: 0.58.1

CPU vs GPU: Comparison

- **Core Count**
  - **CPU:** Fewer, powerful cores (2 to 16)
  - **GPU:** Many, simpler cores (hundreds to thousands)
- **Task Handling**
  - **CPU:** Best for tasks with low parallelism and high complexity
  - **GPU:** Best for tasks with high parallelism and repetitive computations
- **Performance**
  - **CPU:** Optimized for single-threaded performance and latency-sensitive tasks
  - **GPU:** Optimized for high-throughput, data-parallel tasks
- **Architecture**
  - **CPU:** Complex control logic, designed for a wide range of tasks
  - **GPU:** Simple control logic, specialized for parallel processing

CPU vs GPU: Comparison (Contd..)

- **Optimization**
  - **CPU:** Latency optimized (low latency in processing tasks)
  - **GPU:** Throughput optimized (high throughput for large data sets)
- **Applications**
  - **CPU:** General computing (e.g., operating systems, word processing, web browsing)
  - **GPU:** Graphics rendering, machine learning, scientific simulations

CPU Simulation Images

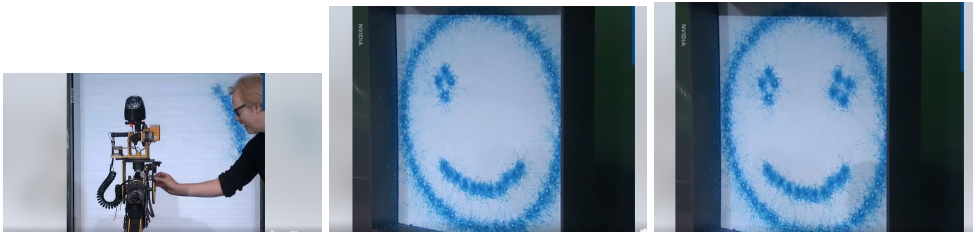


Figure: CPU Simulation Images

GPU Simulation Image

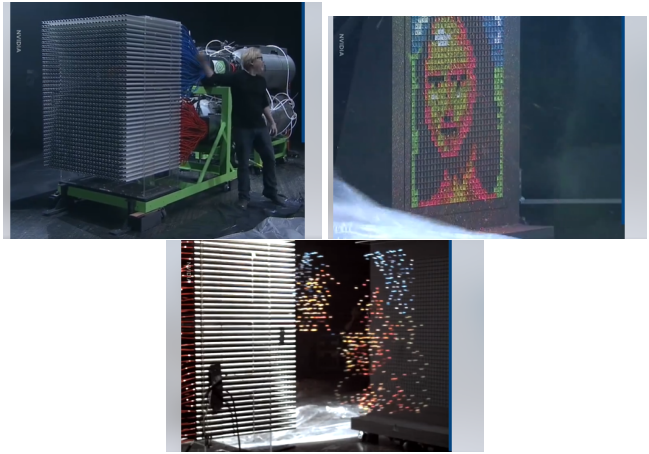


Figure: GPU Simulation Images

Challenges for these Experiments

Limited memory in Google Colab

- After pushing certain limits of experiments, the system would crash, e.g., generating a billion numbers to sort, having a matrix of dimensions 20x20 to multiply

Got access to HPC at GWDG

- Needed VPN or presence in the office
- Had no GPU in them
- Got new account created for me

Challenges for these Experiments (continued)

Installation of CUDA

- NVCC
- Needed Sudo access for installation of required libraries

Went back to Colab and simplified the experiments

- Multiplication of numbers are integers (1s) not floats anymore
- Kept expanding the input until the system crashed

Famous GPU Frameworks(Wang et al.,)

Framework	Core language	CUDA support
Caffe/Caffe2	C++	Yes
TensorFlow	C++	Yes
Theano	Python	Yes
Torch	Lua	Yes
CNTK	C++	Yes
MXNet	Small C++ core library	Yes
MatConvNet	C++	Yes
Deeplearning4j	Java	Yes
Neon	Python	Yes

CUDA

- A general-purpose parallel computing platform and programming model that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU. (NVIDIA,)
- Developed and Introduced by NVIDIA in 2006.
- CUDA comes with a software environment that allows developers to use C++ as a high-level programming language

CUDA Python

- CUDA Python provides Cython/Python wrappers for CUDA driver and runtime APIs. (NVIDIA,)
- CUDA Python provides uniform APIs and bindings for inclusion into existing toolkits and libraries to simplify GPU-based parallel processing for HPC, data science, and AI.
- The goal of CUDA Python is to unify the Python ecosystem with a single set of interfaces that provide full coverage of and access to the CUDA host APIs from Python. (NVIDIA,)

Why is Performance Comparison needed ?

- **Performance Optimization**:
  - CUDA C/C++ offers superior performance through close-to-hardware optimization.
- **Ease of Development**:
  - CUDA Python provides simpler and faster development with libraries like Numba and CuPy.
- **Trade-off Evaluation**:
  - Balance development ease and execution efficiency based on application needs.
- **Application Requirements**:
  - Determine if Python's productivity gains justify potential performance losses.

Why Performance Comparison is needed? [Contd..]

Rapid Prototyping:

- Python's simplicity accelerates the prototyping and testing phases.

Hybrid Approaches:

- Leverage Python for high-level orchestration and CUDA C/C++ for critical performance sections.

Scalability:

- Assess how each approach scales with problem size and GPU capabilities.

Community and Ecosystem:

- Consider the extensive support and resources available for Python development.

Related Work

- There are some performance comparisons between the two frameworks but they don't cover all the different domains we aim to cover.
- Fernandes et al. Explores the two frameworks for only 4th order Runge-Kutta method (Fernandes et al., "Comparative study of CUDA-based parallel programming in C and Python for GPU acceleration of the 4th order Runge-Kutta method")
- Oden et al. draw a comparison for matrix operations, reduction operations etc. (Di Domenico, Lima, and Cavalheiro, "NAS Parallel Benchmarks with Python: a performance and programming effort analysis focusing on GPUs")
- Askar et al. compare them in the context of Monte Carlo Radiation Transport (Askar et al., "Exploring Numba and CuPy for GPU-Accelerated Monte Carlo Radiation Transport")

Performance Comparison

- In this project we will do a more comprehensive comparison of the two frameworks, especially over the Artificial Intelligence domain
- Program Selection for Comparison (performance comparison between CUDA Python and CUDA):
  - Matrix Multiplication
  - Sorting Algorithms
  - Machine Learning Model Training

Experiment Design

Training an ML Model on MNIST Dataset

- Logistic Regression Algorithm
- Data points used: 60, 600, 6000, 60000

Sorting Numbers

- Merge Sort
- Numbers used: 10, 100, 1000, 10000 and so on

Experiment Design (continued)

Multiplying Matrices

- Dimensions: 2^1, 2^2, 2^3, 2^4, 2^5 and so on

Metrics Calculated (in percentage)

- CPU Utilization
- GPU Utilization
- Memory Utilization
- GPU Memory Utilization
- Time taken to run the code (and/or compilation)

Experiment Design (continued)

Execution Frequency

- We run the algorithm for each step 3 to 5 times and then get the average of the values

Note: All the inputs in both algorithms were exactly the same, an attempt to keep all the variables constant except for the one which needs to be compared

Results : Sorting for CUDA

Framework	Comp time	Exec time	GPU Usage	CPU Usage	CPU Ram	GPU RAM	Lines of Code	No of samples
CUDA	2.51	0.1	0	46	5.7	0	90	10
CUDA	2.61	0.11	0	63.3	5.65	0	90	100
CUDA	2.61	0.12	0	96.7	6.1	0	90	1000
CUDA	2.61	0.10	7	67.9	6.16	0	90	10000
CUDA	2.61	0.29	35	73.3	6.88	0	90	100000
CUDA	3.42	0.22	45	97.5	6.9	0	90	1000000
CUDA	2.61	0.28	74	63	7.2	1.0	90	10000000
CUDA	2.61	0.67	100	53.3	7.28	0	90	100000000
CUDA	2.5	1.3	100	73.3	36	50	90	1000000000

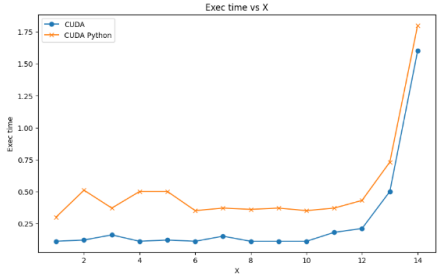
Since we have so many such tables, we therefore focus on few plots worth attention in this presentation

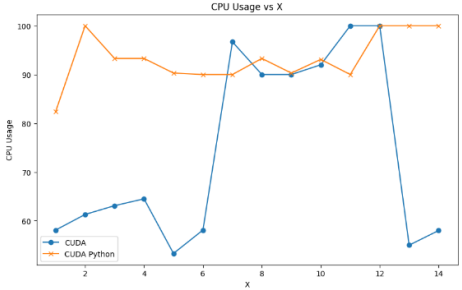
Results: Matrix Multiplication

Observations

- Execution time for CUDA Python is higher
- CUDA Python uses way more memory as compared to CUDA right from the beginning
- CPU usage drops in CUDA when GPU comes to work
- GPU utilization is more in case of CUDA Python

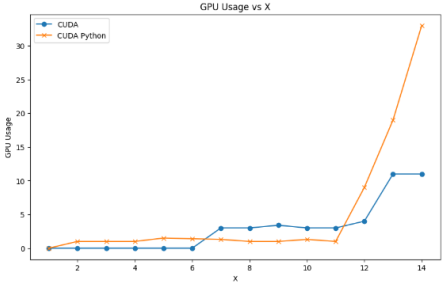
Results: Matrix Multiplication (Images)

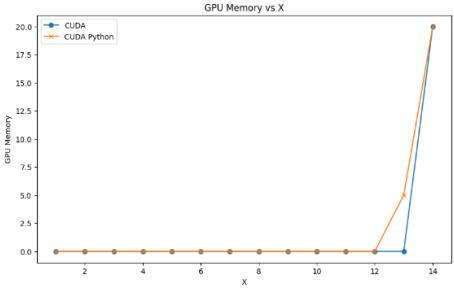




-> see how CUDA Python is consuming more resources than CUDA right from the beginning

Results: Matrix Multiplication (Images Continued)





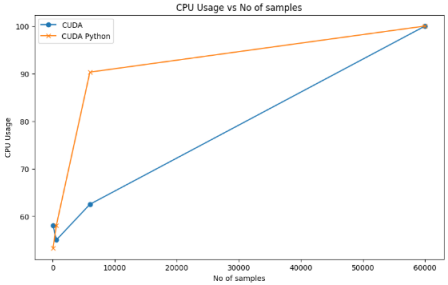
-> See how the results here are comparable and close in both of the frameworks

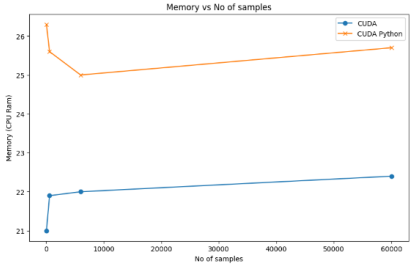
Results: ML Training

Observations

- CPU usage is more in case of CUDA Python
- GPU usage is quite comparable
- Memory Consumption is more in case of CUDA Python

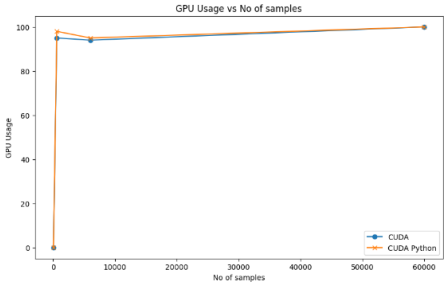
Results: ML Training (Images)





-> Again how CUDA Python is way higher right from the beginning in its usage in resources

Results: ML Training (Images Continued)



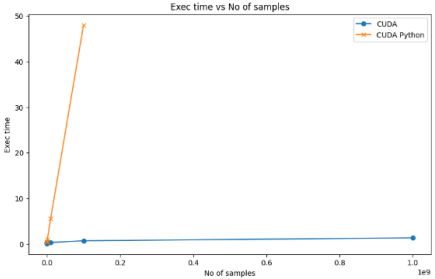
-> Again the results are comparable

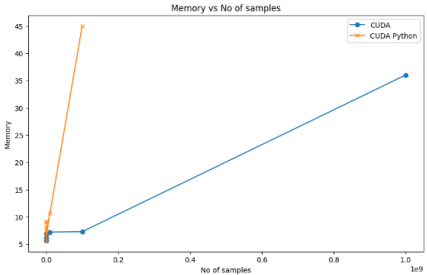
Results: Sorting

Observations

- Execution time ramps up as data increases and eventually crashes for CUDA Python
- Memory consumption abruptly increases and the system crashes in case of CUDA Python
- CUDA goes a step further and then eventually terminates instead of crashing
- CUDA is also memory efficient as compared to CUDA Python

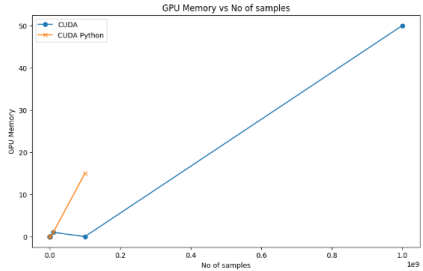
Results: Sorting (Images)





-> Again CUDA Python goes higher in resource consumption

Results: Sorting (Images Continued)



-> Dosen’t last long in the experiment

Summary

- In this project we experiment and compare the performance of the two most important Frameworks for GPUs
- CUDA forms the bedrock for all the major deep-learning frameworks while as CUDA Python is gaining an overwhelming popularity among the Python Community
- We compare some of the most common types of algorithms' performance with these frameworks, i.e. Matrix Multiplication, Sorting and Deep Learning Models training.
- we also see the utilisation of different resources by these frameworks.

References

Askar, Tair et al. “Exploring Numba and CuPy for GPU-Accelerated Monte Carlo Radiation Transport”. In: *Computation* 12.3 (2024), p. 61.

Di Domenico, Daniel, Joao VF Lima, and Gerson GH Cavalheiro. “NAS Parallel Benchmarks with Python: a performance and programming effort analysis focusing on GPUs”. In: *The Journal of Supercomputing* 79.8 (2023), pp. 8890–8911.

Fernandes, Davi F. et al. “Comparative study of CUDA-based parallel programming in C and Python for GPU acceleration of the 4th order Runge-Kutta method”. In: *Nuclear Engineering and Design* 421 (2024), p. 113050.

NVIDIA. Accessed: 2024-05-25. 2023. URL: `https://docs.nvidia.com/cuda/index.html`.

— .Accessed: 2024-05-25. 2023. URL: `https://nvidia.github.io/cuda-python/motivation.html`.

— .Accessed: 2024-05-25. 2023. URL: `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`.

Wang, Zhaobin et al. In: *Archives of Computational Methods in Engineering* (2019), pp. 1–24.