**Challenge #13: Benchmarking Different SAXPY Problem Sizes (CUDA)**

**Objective**

To evaluate how SAXPY (Single-Precision A·X Plus Y) kernel performance scales with increasing vector sizes on a CUDA-enabled GPU, using **Google Colab** and **CUDA C/C++**.

**Learning Goals**

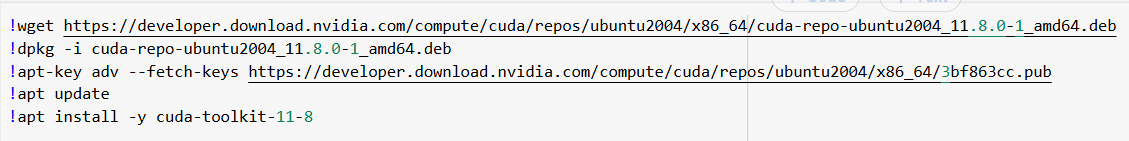
* Set up and verify a **CUDA development environment**
* Understand and modify **CUDA SAXPY** kernel
* Measure execution time across varying input sizes
* Differentiate between **kernel-only execution** and **full pipeline time**
* Visualize trends in performance scaling

**Prompts Used**

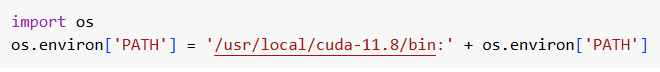
* “How to set up a CUDA development environment in Google Colab?”
* “Can you modify this SAXPY CUDA code to benchmark for N=2^15 to 2^25?”
* “How do I measure kernel-only vs total execution time using cudaEvent?”
* “Can you help visualize the results using Matplotlib?”
* “Give documentation for Challenge #13: Benchmarking SAXPY”

**Step-by-Step Process**

* **Step 1**: Set Up GPU Environment in Google Colab
  + Changed runtime: Runtime → Change runtime type → GPU
  + Verified GPU:  
    !nvidia-smi
* **Step 2:** Installed CUDA Toolkit 11.8



* Step 3: Updated PATH



* Step 4: Modified SAXPY CUDA Code
  + Looped from N = 2^15 to 2^25
  + Measured:
    - **Total execution time**: Allocation + Copy + Kernel + Copy-back
    - **Kernel-only time**: GPU compute only
  + Used cudaEventRecord() to profile both
* Step 5: Compiled and Ran the CUDA Program



* Step 6: Captured Output & Visualized with Matplotlib
  + Parsed printed timings
  + Created grouped bar chart comparing total vs kernel times

**Results (Sample Output)**

| **N = 2^x** | **Total Time (ms)** | **Kernel Time (ms)** |
| --- | --- | --- |
| 15 | 12.041 | 11.667 |
| 16 | 0.439 | 0.013 |
| 17 | 0.679 | 0.016 |
| 20 | 3.298 | 0.052 |
| 23 | 23.497 | 0.380 |
| 25 | 92.901 | 1.523 |

✅ **Max error for all cases = 0.000000**

**Visualization**

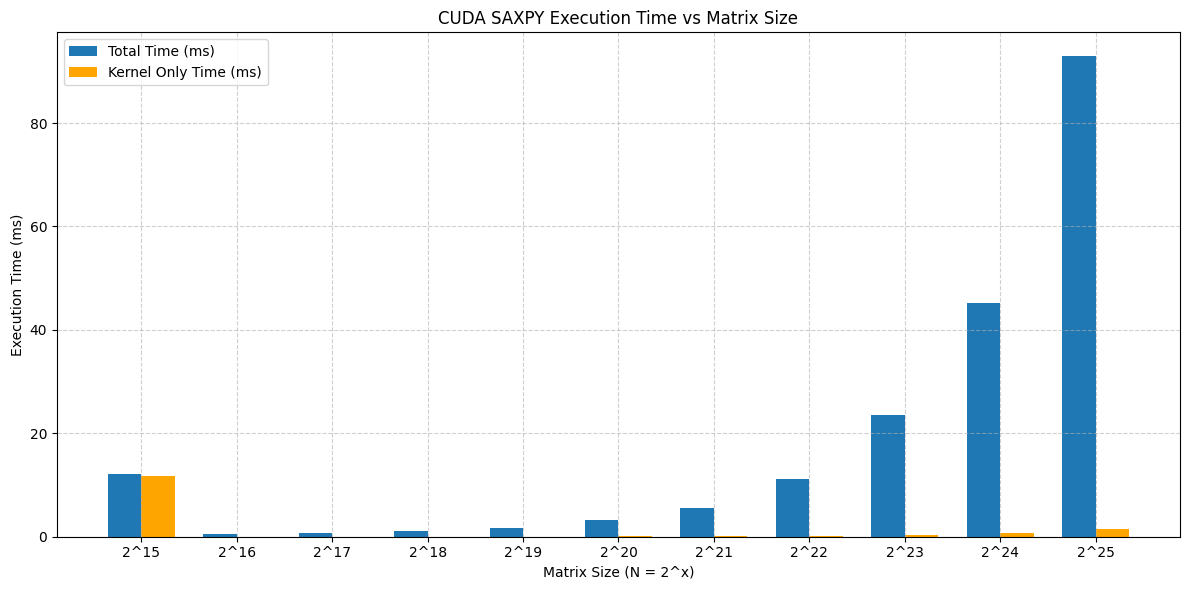
We used the following Matplotlib snippet:

plt.bar(x - bar\_width/2, total\_times, label='Total Time')

plt.bar(x + bar\_width/2, kernel\_times, label='Kernel Time', color='orange')

What We Observed

* **Kernel time scales linearly** with N
* **Total time increases faster**, especially at lower N, due to:
  + Memory allocation latency
  + Device-host memory transfers
* Kernel execution is very **efficient**, while memory ops introduce overhead



**Conclusion**

This challenge successfully demonstrated how GPU workloads scale with input size. We:

* Set up a GPU-ready CUDA environment in Google Colab
* Wrote and profiled a SAXPY CUDA kernel
* Analyzed kernel performance in isolation from memory overhead
* Gained insight into **CUDA memory bottlenecks** and how to benchmark GPU code effectively