EE 533- CUDA LAB

Name: Chaitanya Joshi
USC ID: 4991436817 (joshic@usc.edu)
GitHub repo: https://github.com/chaijosh/sp2026_ee533_CUDAlab1

System specifications:

- CPU- 11th Gen Intel(R) Core(TM) i5-11320H @ 3.20GHz
- GPU- NVIDIA GeForce RTX 3050 Laptop GPU
  - Interface: PCI bus 1, device 0, function 0
  - Max Dedicated GPU memory- 4.0 GB
  - Max Shared GPU memory- 7.9 GB
  - Max total GPU Memory- 11.9 GB
- OS- WSL (Ubuntu 24.04.3) over Windows 11
- C compiler- GCC 13.3.0
- CUDA compiler- CUDA/NVCC V13.1.115

PART 1: Comparison of execution times of Matrix Multiplication using CPU and various GPU kernels

Different C/CUDA codes for matmult operation were tested and their execution times were recorded.

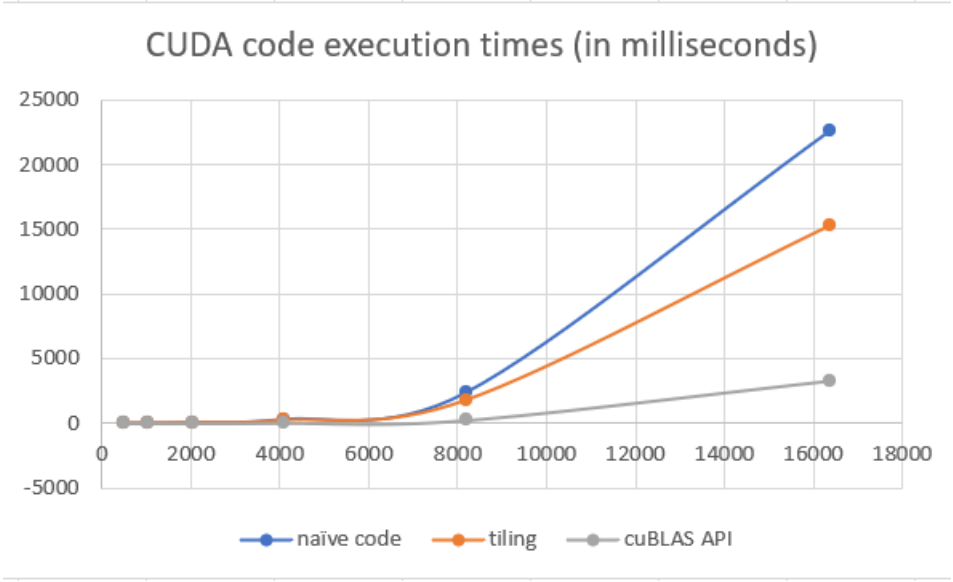
- CPU execution times were measured using std::clock() function (present in time.h header)
- GPU only execution times were also measured using cudaEventCreate() and cudaEventRecord() functions [1]
- This was done as CPU execution may not always be accurate due to:
  - CPU-GPU synchronization delays
  - PCIe interface delays

Below programs were executed 5 times and average execution times were recorded for various matrix sizes (N):

- CPU execution: /main/
- GPU execution (Naïve code): /main/
- CPU execution (Shared memory tiling): /main/
- CPU execution (cuBLAS library call): /main/

Observations:

Code	N=512		N=1024		N=2048		N=4096		N=8192		N=16384	
	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU	CPU	GPU
matrix_cpu.c	315.0928		5167.459		115769.1		1021593		8987671		TLE	
matrix_gpu.cu	1.6566	2.471763	6.3946	6.920045	45.9314	46.45742	293.6342	293.89	2346.252	2345.365	22638.8	22625.51
matrix_gpu_tiled.cu	1.3064	2.017146	5.144	5.830656	35.3316	35.85297	217.3456	217.762	1759.368	1758.763	15298.99	15291.12
matrix_gpu_cuBLAS.cu	0.3584	0.217293	0.78	0.635699	8.1266	4.360602	58.4496	30.82895	473.021	237.825	6574.019	3295.156



Analysis Questions

1. How does performance change as matrix size increases?

- We see that CPU execution time goes very high (~17 mins for N=4096, ~2.5 hrs for N=8192). Since matrix multiplication time complexity approaches $O(N^3)$, this observation is expected.
- We wee that GPU code is significantly faster, with N=16384 computation happening in ~25s even for naïve CUDA code.
- Tiled and cuBLAS codes appear to be much faster than naïve code as they utilize more resources and do It smartly

2. At what point does the GPU significantly outperform the CPU?

- Even at N=32, the CUDA codes run ~1.3 times faster than CPU. \
- This effect is more pronounced, with N=512 showing 100-1000x improvements than CPU code.

3. How much speedup is gained by tiling optimization vs. naïve CUDA?

- Speedup = ~1.3 based on values shown in table

4. How close is your optimized kernel to cuBLAS performance?

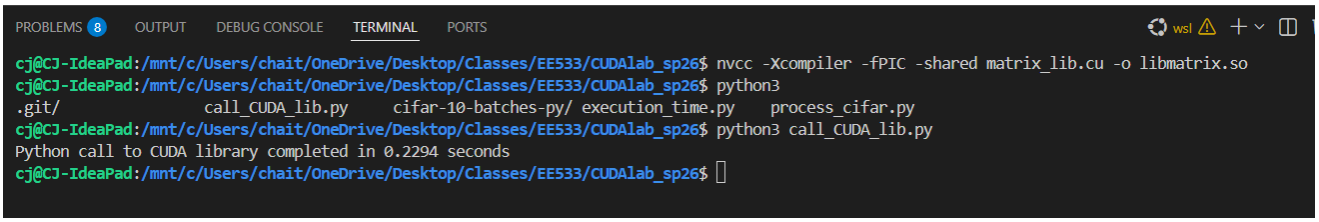
- cuBLAS code is significantly faster than tiled code.
- For N=512-8192, speedup ~8x
- For higher N (=16384), even cuBLAS code starts saturating, giving 4x improvement over tiled code.

5. Why might cuBLAS still outperform hand-written kernels?

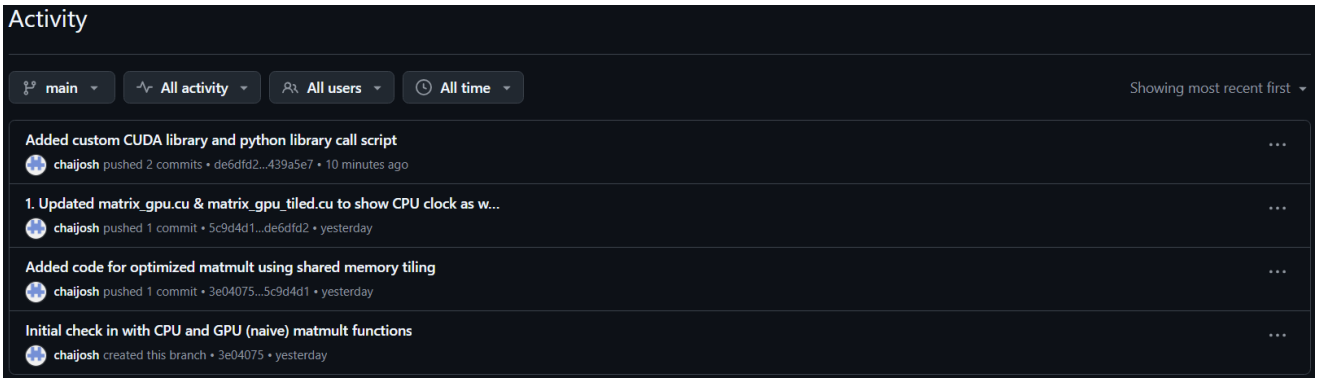
- cuBLAS lib calls can be specific to the CUDA library we installed, i.e. catered to the particular GPU architecture we have. handwritten kernel is more generic implementation without knowing the exact GPU architecture & uArch.
- lib call can access all DPU/Tensor cores efficiently as it is written in highly optimized assembly. Also, during compilation, lib calls are optimized to have the best scheduling, memory/register access, resource utilization etc.

Part 2: Creating a Shared Library and Using it in Python

- CUDA library: /main/ matrix_lib.cu
- Python call: /main/call_CUDA_lib.py



Github commit history:



References:

[1] CUDA toolkit documentation (https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EVENT.html)