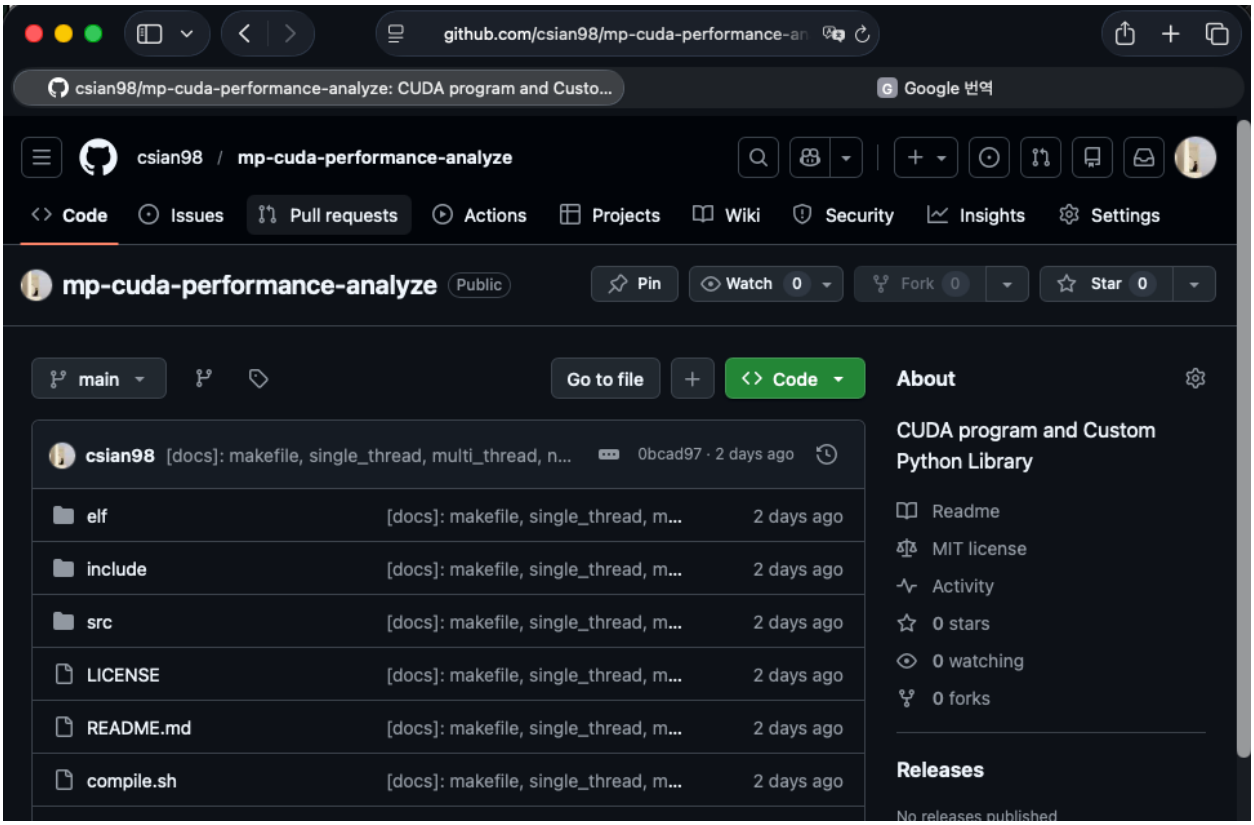
USCID: 5023184813
Group 15

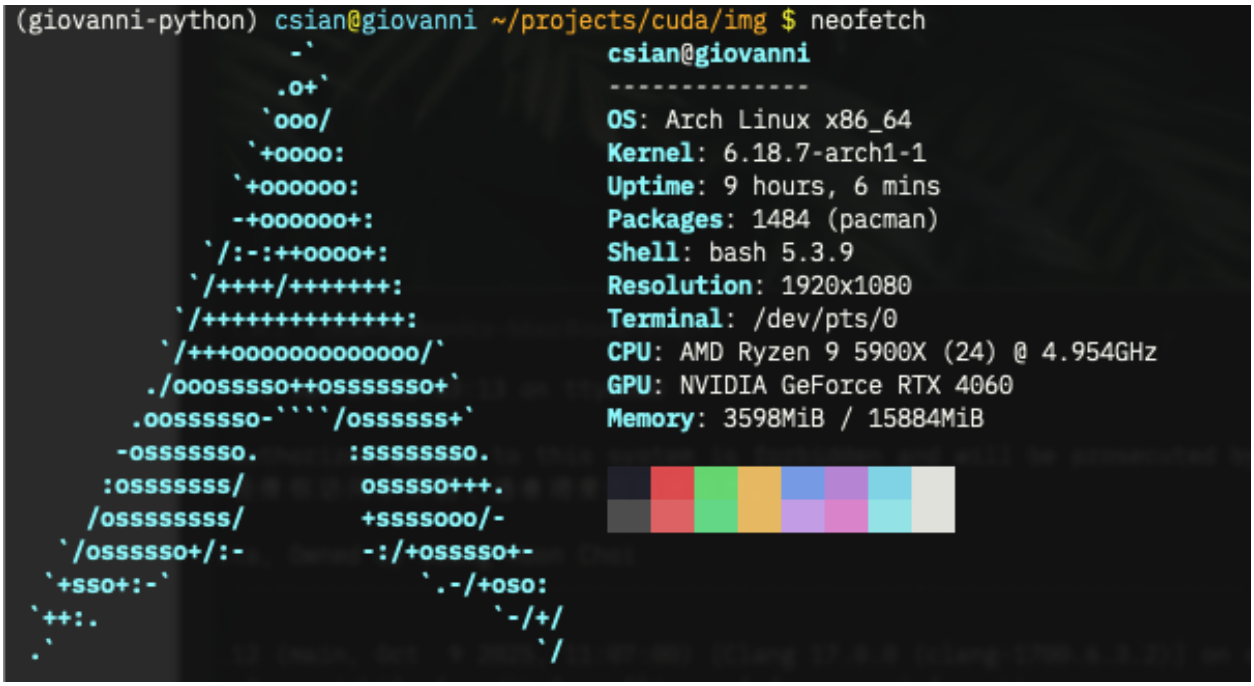
Laboratory Assignment 3

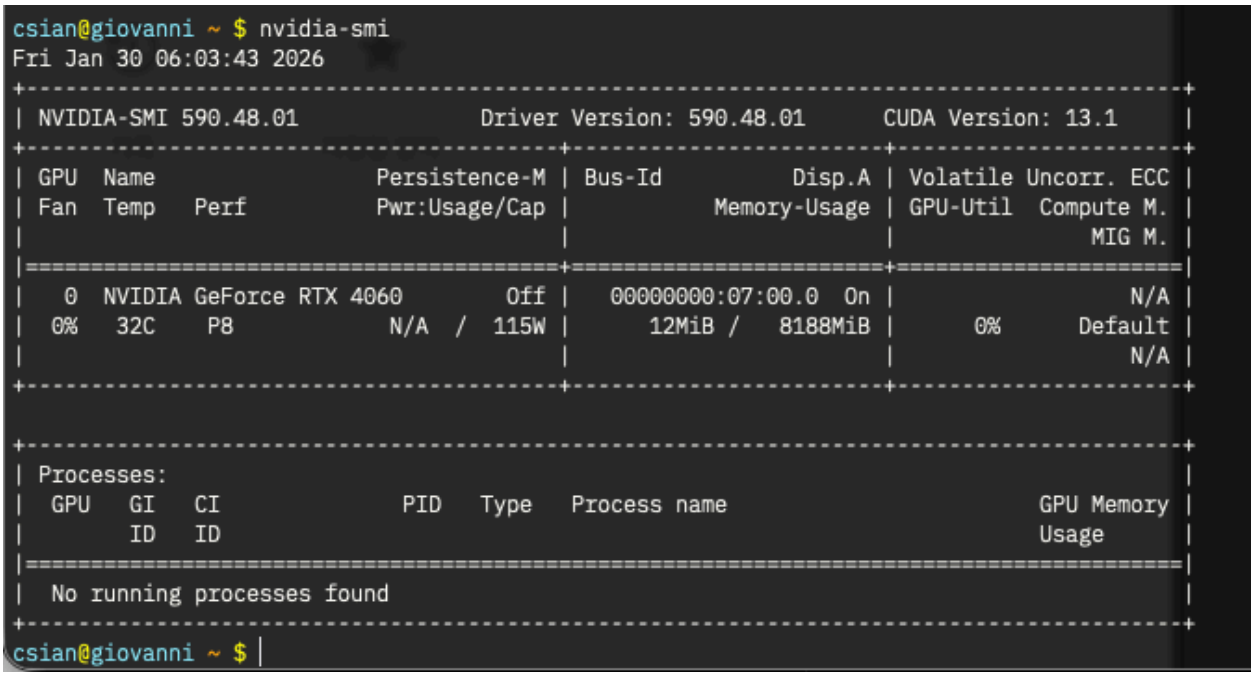
Group 15: Jeong Hoon Choi, Viraj Bansalm, Chih-Yun Pai



Jeong Hoon Choi: https://github.com/csian98/mp-cuda-performance-analyze
Viraj Bansalm: https://github.com/SlyShot42/cuda-matmul-conv-bench.git
Chih-Yun Pai: https://github.com/Howard-Pai/DSCI-560-Data-Science-practicum/tree/main/Lab3

This week, we will each use our own repositories and share all the repository URLs. The example code below uses Jeong Hoon Choi's GitHub code.





All processes were run on my local device with Nvidia GeForce RTX 4060

```
csian@giovanni ~/projects/cuda $ cat process_speed_check.cu
/**
 * @File                process_speed_check.cu
 * @brief               single thread, multi thread and SIMT GPU
 * @author              Jeong Hoon (Sian) Choi
 * @version     1.0.0
 * @date                2024-05-19
 */

/* Copyright (C)
 * 2024 - Jeong Hoon (Sian) Choi
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 */

#include <cuda.h>
```

1 Matrix Multiplication

A significant portion of the code was written by referencing process_speed_check.cu, which I had written previously.
https://github.com/csian98/sian/blob/main/usages/process_speed_check.cu

```
csian@giovanni ~/projects/cuda $ ls
build             compile.sh      elf  include  LICENSE   matrix_multiplication.py  README.md
compile_flags.txt  convolution.py  img  lib      makefile  process_speed_check.cu    src
csian@giovanni ~/projects/cuda $ make all
make: Nothing to be done for 'all'.
csian@giovanni ~/projects/cuda $
```

All code can be compiled using the makefile, and the executable files will be stored in the elf directory.

```
csian@giovanni ~/projects/cuda/elf $ ls
convolution.elf  multi_thread      naive_cuda.elf     single_thread
cuBLAS.elf       multi_thread.elf  optimize_cuda.elf  single_thread.elf
csian@giovanni ~/projects/cuda/elf $
```

CPU: single_thread, multi_threads

```
csian@giovanni ~/projects/cuda/elf $ ./single_thread.elf
####
Single Thread Matrix Multiplication
####

n: 1024, k: 1024, m: 1024
#              Timer Report              #
#  The number of timer = 1
####
single thread : 3240.13 milli seoncds
csian@giovanni ~/projects/cuda/elf $ ./multi_thread.elf
####
Multi Thread Matrix Multiplication
####

n: 1024, k: 1024, m: 1024
number of threads: # 24
#              Timer Report              #
#  The number of timer = 1
####
multi thread : 244.221 milli seoncds
```

GPU(CUDA): naive_cuda.elf, optimize_cuda.elf

```
csian@giovanni ~/projects/cuda/elf $ ./naive_cuda.elf
####
Naive CUDA Matrix Multiplication
####

n: 1024, k: 1024, m: 1024
#              Timer Report                #
#  The number of timer = 1
####
navie cuda : 108.671 milli seoncds
csian@giovanni ~/projects/cuda/elf $ ./optimize_cuda.elf
####
Optimize CUDA Matrix Multiplication
####

n: 1024, k: 1024, m: 1024
#              Timer Report                #
#  The number of timer = 1
####
optimize cuda : 106.869 milli seoncds
```

cuBLAS: cuBLAS.elf

```
csian@giovanni ~/projects/cuda/elf $ ./cuBLAS.elf
####
cuBLAS Matrix Multiplication
####

n: 1024, k: 1024, m: 1024
#              Timer Report                #
#  The number of timer = 1
####
cuBLAS : 139.92 milli seoncds
```

Implementation	N=512	N=1024	N=2048	N=4096
single_thread.elf	412.977 ms	3449.16 ms	28816.2 ms	336982 ms
multi_thread.elf	26.7683 ms	224.861 ms	1751.99 ms	33731.4 ms
naive_cuda.elf	97.9309 ms	104.937 ms	179.876 ms	689.199 ms
optimize_cuda.elf	99.5198 ms	106.23 ms	194.67 ms	809.963 ms
cuBLAS.elf	136.078 ms	127.684 ms	213.883 ms	778.369 ms

Analysis Questions:
1. How does performance changes as matrix size increases?
CPU calculation increasing O(n^3) exponentially, but GPU calculation time more depend on the copy and load data between host and device.
2. At what point does the GPU significantly outperform the CPU?
In the case of single-threaded execution, the CUDA code consistently took more time than the non-cuda code, while in the case of multi threaded execution, CUDA took less time starting from a size of 1024.
3. How much speedup is gained by tiling optimization vs naive CUDA?
In the size range from 512 to 4096, the naive CUDA implementation consistently took less time. Personally, I believe that bottlenecks in the thread synchronization process likely caused it to take a similar or even longer amount of time.
4. How close is your optimized kernel to cuBLAS performance?
The optimized CUDA and cuBLAS code showed similar levels of computational speed in almost all cases.
5. Why might cuBLAS still outperform hand-written kernels?
I believe that, like many optimation codes, cuBLAS reduces computation time by selecting the calculation method based on the input size.
For example, it likely uses a naive_cuda method for small N and an optimized_cuda method for large N.

```
(giovanni-python) csian@giovanni ~/projects/cuda $ python matrix_multiplication.py
Python call to CUDA library cost time: 0.1033
(giovanni-python) csian@giovanni ~/projects/cuda $
```

Using ctypes to call CUDA C++ code from Python

2 Image Convolution

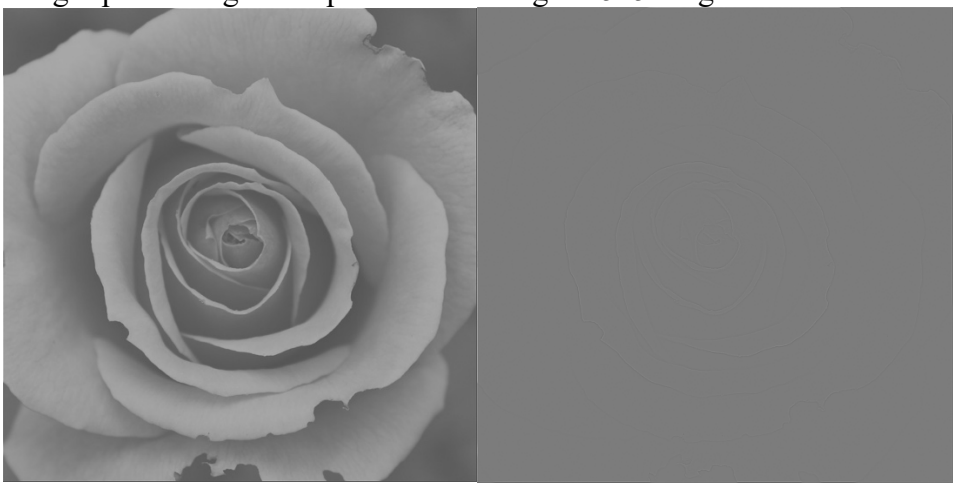
USCID: 5023184813
Group 15

Regarding the original image, after processing the grayscale uint8 data, image processing was performed on the 2048x2048 pixel image (center cropped).

```
(giovanni-python) csian@giovanni ~/projects/cuda $ python convolution.py
(giovanni-python) csian@giovanni ~/projects/cuda $
```

```
(giovanni-python) csian@giovanni ~/projects/cuda/img $ ls
edge_detection.png  original.png  sharpen.png
(giovanni-python) csian@giovanni ~/projects/cuda/img $
```

Image processing was performed using a 5x5 edge detection filter and a sharpen filter.



Although the edges are not clearly visible in the edge detection result due to the small filter size, when the contrast is adjusted to change the brightness and darkness, it can be confirmed that the edges are being extracted.

USCID: 5023184813
Group 15