Report - cuda lab

Yuhan Liang

9363204003

1 github link (source codes, use Makefile to compile):

https://github.com/yuhanlia1/cuda-lab2.git

2 graph: (use the ./scripts/run_benchmarks.sh and ./scripts/plot_results.py to verify)

Fig.1 matrix mult:

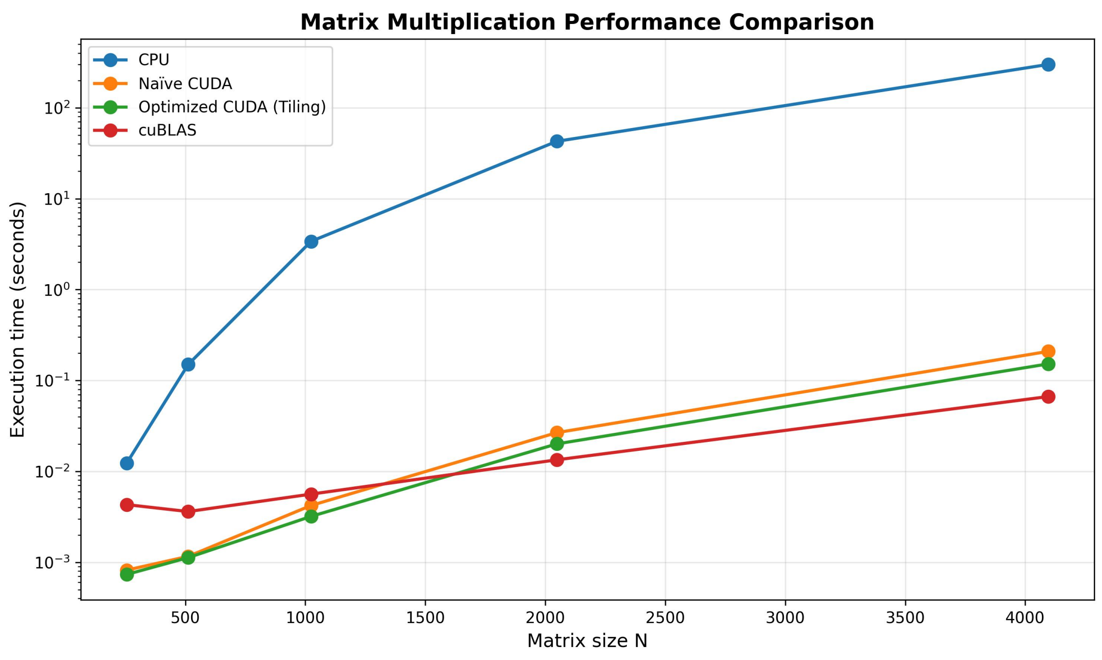


Fig.2 convolution

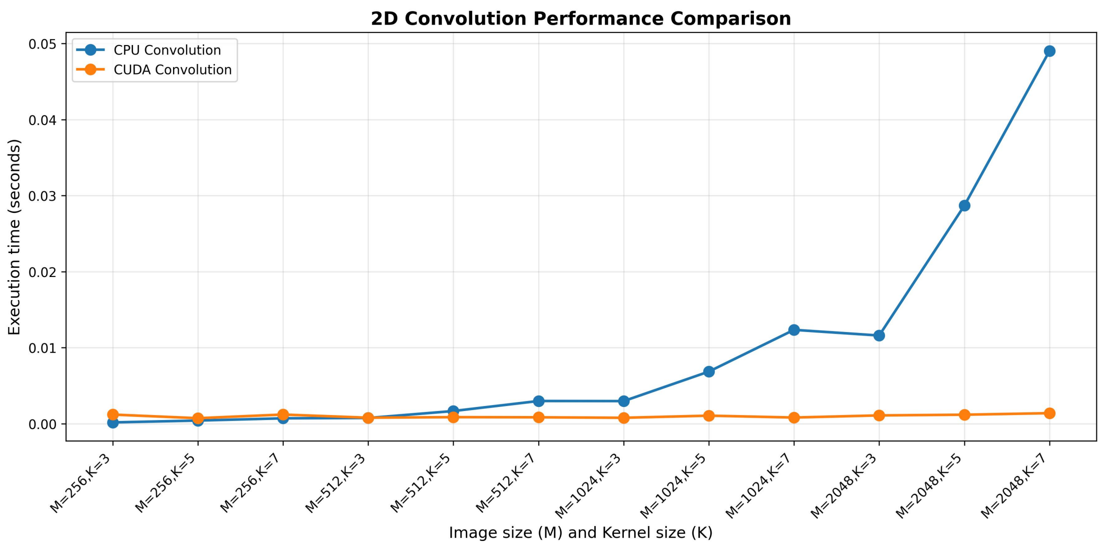


Fig.3 speedup

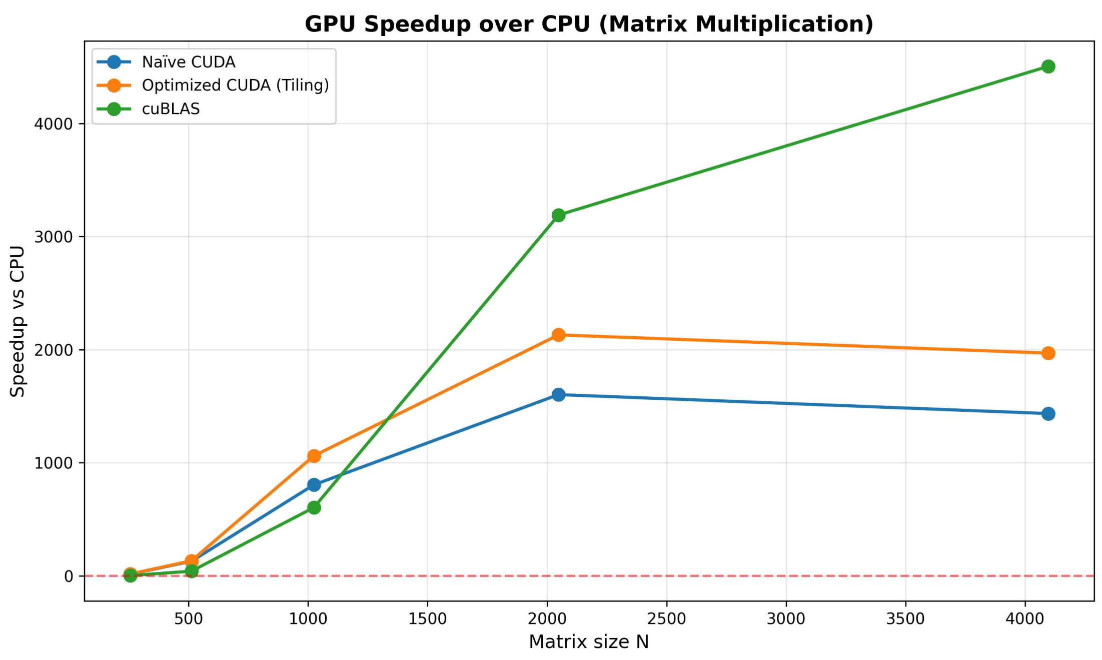


Table.1 dataset (without the scripts, made by hand)

Implementation	N=512	N=1024	N=2048	N=4096
CPU(C)	0.150000	3.391378	42.727917	299.540570
Naive CUDA	0.001161	0.004213	0.026660	0.208726
Opt CUDA	0.001125	0.003196	0.020054	0.152211
cuBLAS	0.003612	0.005617	0.013393	0.066493
shared lib on .py	0.002100	0.005700	0.025400	0.172100

Unit: seconds

3 report and analysis :

1 Observations on performance scaling:

a.Matrix mult:

Figure 1 shows the execution time of matrix multiplication as the matrix size increases. **Theoretically**, the time complexity of matrix multiplication on a CPU is O(N^3), and experimental results show this reality(Y-asix is log scaling). **In contrast**, all GPU implementations scale much better. The naïve CUDA kernel significantly outperforms the CPU version even for moderate matrix sizes, while the optimized tiled kernel further reduces execution time by improving memory locality. The cuBLAS implementation provides the best overall scalability for large matrices.

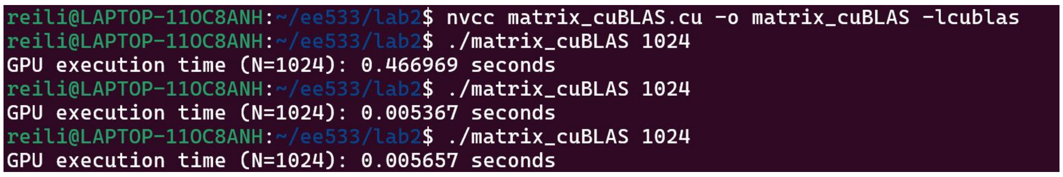
b.Convolution:

Figure 2 illustrates the performance of 2D convolutions under varying image dimensions (M) and kernel sizes (K). **Theoretically**, the execution time of convolutions exhibits an approximate O(M^2) growth trend. **As** image dimensions increase, CUDA-based convolutions progressively demonstrate significant advantages.

2 Analysis of overhead associated with using GPU/DPU accelerator

Firstly, GPUs don't offer advantages for data that cannot fully leverage parallel computing, which we can observe the Fig.3 where the speedup rate almost equals 1.0 at the beginning, because the matrix size is too small to use the GPU performance.

Secondly, GPU has a warm-up phrase at the first-time test. This start-up time also cause a huge timing consumption. As you can see the screenshot below, the GPU need a longer time to warm-up (about = (0.466969 - 0.0055) seconds).



```
reili@LAPTOP-11OC8ANH:~/ee533/lab2$ nvcc matrix_cuBLAS.cu -o matrix_cuBLAS -lcublas
reili@LAPTOP-11OC8ANH:~/ee533/lab2$ ./matrix_cuBLAS 1024
GPU execution time (N=1024): 0.466969 seconds
reili@LAPTOP-11OC8ANH:~/ee533/lab2$ ./matrix_cuBLAS 1024
GPU execution time (N=1024): 0.005367 seconds
reili@LAPTOP-11OC8ANH:~/ee533/lab2$ ./matrix_cuBLAS 1024
GPU execution time (N=1024): 0.005657 seconds
```

3 Reflections on optimization trade-offs

For moderate matrix sizes (about 1k), the hand-optimized tiled kernel (matrix_opt.cu) performed comparably to or slightly better than cuBLAS, likely due to lower overhead and problem-specific tuning. This indicates that further optimization requires careful control over hyperparameter settings such as tile size selection. This will inevitably increase the complexity of the code.

4 List and description of your shared library for Python

A CUDA shared library was created to expose GPU-accelerated functions to Python using the ctypes interface. The library includes an optimized tiled matrix multiplication kernel and a CUDA-based 2D convolution function.

As shown in the Table.1, in terms of execution time, Python code always runs slower than C++ code, even when calling dynamic libraries compiled with CUDA.

4 Some screenshot of running log:

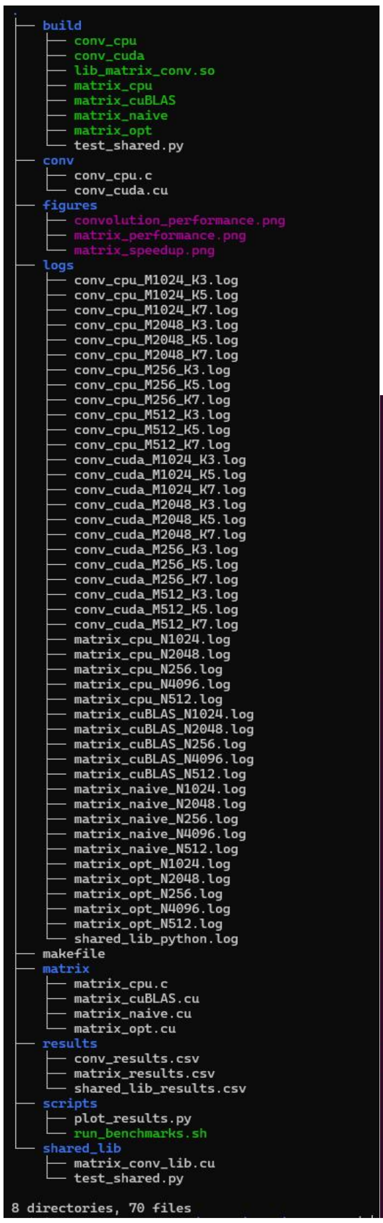
./conv: save the convolution sources code

./matrix: save the matrix source code

./shared_lib: save the shared library source code

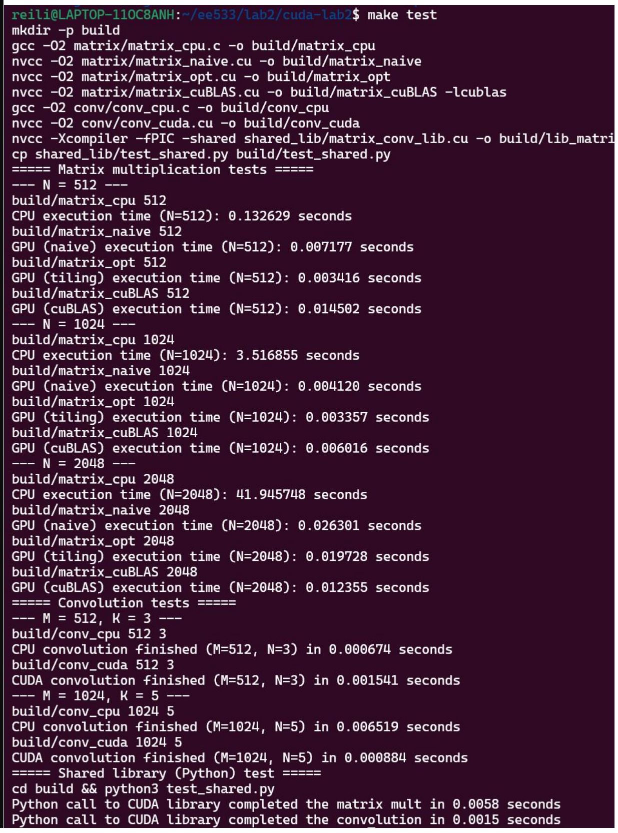
./scripts: it includes data test

./figure, ./build, ./log, ./results: some dir save the results after scripts run



```
.
├── build
│   ├── conv_cpu
│   ├── conv_cuda
│   ├── lib_matrix_conv.so
│   ├── matrix_cpu
│   ├── matrix_cuBLAS
│   ├── matrix_naive
│   ├── matrix_opt
│   └── test_shared.py
├── conv
│   ├── conv_cpu.c
│   └── conv_cuda.cu
├── figures
│   ├── convolution_performance.png
│   ├── matrix_performance.png
│   └── matrix_speedup.png
├── logs
│   ├── conv_cpu_M1024_K3.log
│   ├── conv_cpu_M1024_K5.log
│   ├── conv_cpu_M1024_K7.log
│   ├── conv_cpu_M2048_K3.log
│   ├── conv_cpu_M2048_K5.log
│   ├── conv_cpu_M2048_K7.log
│   ├── conv_cpu_M256_K3.log
│   ├── conv_cpu_M256_K5.log
│   ├── conv_cpu_M256_K7.log
│   ├── conv_cpu_M512_K3.log
│   ├── conv_cpu_M512_K5.log
│   ├── conv_cpu_M512_K7.log
│   ├── conv_cuda_M1024_K3.log
│   ├── conv_cuda_M1024_K5.log
│   ├── conv_cuda_M1024_K7.log
│   ├── conv_cuda_M2048_K3.log
│   ├── conv_cuda_M2048_K5.log
│   ├── conv_cuda_M2048_K7.log
│   ├── conv_cuda_M256_K3.log
│   ├── conv_cuda_M256_K5.log
│   ├── conv_cuda_M256_K7.log
│   ├── conv_cuda_M512_K3.log
│   ├── conv_cuda_M512_K5.log
│   ├── conv_cuda_M512_K7.log
│   ├── matrix_cpu_N1024.log
│   ├── matrix_cpu_N2048.log
│   ├── matrix_cpu_N256.log
│   ├── matrix_cpu_N4096.log
│   ├── matrix_cpu_N512.log
│   ├── matrix_cuBLAS_N1024.log
│   ├── matrix_cuBLAS_N2048.log
│   ├── matrix_cuBLAS_N256.log
│   ├── matrix_cuBLAS_N4096.log
│   ├── matrix_cuBLAS_N512.log
│   ├── matrix_naive_N1024.log
│   ├── matrix_naive_N2048.log
│   ├── matrix_naive_N256.log
│   ├── matrix_naive_N4096.log
│   ├── matrix_naive_N512.log
│   ├── matrix_opt_N1024.log
│   ├── matrix_opt_N2048.log
│   ├── matrix_opt_N256.log
│   ├── matrix_opt_N4096.log
│   ├── matrix_opt_N512.log
│   └── shared_lib_python.log
├── makefile
├── matrix
│   ├── matrix_cpu.c
│   ├── matrix_cuBLAS.cu
│   ├── matrix_naive.cu
│   └── matrix_opt.cu
├── results
│   ├── conv_results.csv
│   ├── matrix_results.csv
│   └── shared_lib_results.csv
├── scripts
│   ├── plot_results.py
│   └── run_benchmarks.sh
└── shared_lib
    ├── matrix_conv_lib.cu
    └── test_shared.py

8 directories, 70 files
```



```
reili@LAPTOP-11OC8ANH:~/ee533/lab2/cuda-lab2$ make test
mkdir -p build
gcc -O2 matrix/matrix_cpu.c -o build/matrix_cpu
nvcc -O2 matrix/matrix_naive.cu -o build/matrix_naive
nvcc -O2 matrix/matrix_opt.cu -o build/matrix_opt
nvcc -O2 matrix/matrix_cuBLAS.cu -o build/matrix_cuBLAS -lcublas
gcc -O2 conv/conv_cpu.c -o build/conv_cpu
nvcc -O2 conv/conv_cuda.cu -o build/conv_cuda
nvcc -Xcompiler -fPIC -shared shared_lib/matrix_conv_lib.cu -o build/lib_matri
cp shared_lib/test_shared.py build/test_shared.py
===== Matrix multiplication tests =====
--- N = 512 ---
build/matrix_cpu 512
CPU execution time (N=512): 0.132629 seconds
build/matrix_naive 512
GPU (naive) execution time (N=512): 0.007177 seconds
build/matrix_opt 512
GPU (tiling) execution time (N=512): 0.003416 seconds
build/matrix_cuBLAS 512
GPU (cuBLAS) execution time (N=512): 0.014502 seconds
--- N = 1024 ---
build/matrix_cpu 1024
CPU execution time (N=1024): 3.516855 seconds
build/matrix_naive 1024
GPU (naive) execution time (N=1024): 0.004120 seconds
build/matrix_opt 1024
GPU (tiling) execution time (N=1024): 0.003357 seconds
build/matrix_cuBLAS 1024
GPU (cuBLAS) execution time (N=1024): 0.006016 seconds
--- N = 2048 ---
build/matrix_cpu 2048
CPU execution time (N=2048): 41.945748 seconds
build/matrix_naive 2048
GPU (naive) execution time (N=2048): 0.026301 seconds
build/matrix_opt 2048
GPU (tiling) execution time (N=2048): 0.019728 seconds
build/matrix_cuBLAS 2048
GPU (cuBLAS) execution time (N=2048): 0.012355 seconds
===== Convolution tests =====
--- M = 512, K = 3 ---
build/conv_cpu 512 3
CPU convolution finished (M=512, N=3) in 0.000674 seconds
build/conv_cuda 512 3
CUDA convolution finished (M=512, N=3) in 0.001541 seconds
--- M = 1024, K = 5 ---
build/conv_cpu 1024 5
CPU convolution finished (M=1024, N=5) in 0.006519 seconds
build/conv_cuda 1024 5
CUDA convolution finished (M=1024, N=5) in 0.000884 seconds
===== Shared library (Python) test =====
cd build && python3 test_shared.py
Python call to CUDA library completed the matrix mult in 0.0058 seconds
Python call to CUDA library completed the convolution in 0.0015 seconds
```