

Assignment 2: Naive CUDA GEMM Implementation

Objective

The goal of this assignment is to implement a **naive Generalized Matrix Multiplication (GEMM)** kernel using CUDA, without using high-level CUDA libraries such as cuBLAS or cuDNN. The implementation must rely solely on **global memory** and support optional transposition of input matrices.

Implementation Overview

A CUDA kernel was implemented to compute:

$$C \leftarrow \alpha \cdot op(A) \cdot op(B) + \beta \cdot C$$

where:

- $A \in \mathbb{R}^{m \times k}$
- $B \in \mathbb{R}^{k \times n}$
- $C \in \mathbb{R}^{m \times n}$
- $op(A)$ and $op(B)$ may be either the original matrix or its transpose
- α and β are scalar coefficients

Each CUDA thread computes one element of the output matrix C . The kernel uses a **2D grid of 2D thread blocks**, and all memory accesses are performed using **global memory only**, as required.

No optimizations such as shared memory, tiling, or loop unrolling were applied.

Supported GEMM Variants

The kernel supports all required combinations:

- $C \leftarrow \alpha AB + \beta C$
- $C \leftarrow \alpha A^T B + \beta C$
- $C \leftarrow \alpha AB^T + \beta C$
- $C \leftarrow \alpha A^T B^T + \beta C$

The output matrix C is updated **in place**.

Correctness Verification

A CPU reference implementation of GEMM was used to verify correctness. The GPU results are compared element-wise against the CPU output, and mismatches are reported.

Compilation

The code was compiled using the NVIDIA CUDA compiler:

```
nvcc -O2 -std=c++17 gemm_naive.cu -o gemm
```

Execution Environment

The local development machine is equipped with an **Intel® Arc™ 130V GPU**, which is **not CUDA-capable**. CUDA requires an **NVIDIA GPU and NVIDIA driver** to execute kernels.

At runtime, the program checks for a CUDA-capable device using `cudaGetDeviceCount()`. Since no NVIDIA GPU is present, the program exits gracefully with an informative message indicating that no CUDA device is available. This behavior is expected and confirms correct runtime error handling.

The implementation is expected to run correctly on any system with a CUDA-capable NVIDIA GPU.

Conclusion

This assignment successfully implements a naive CUDA GEMM kernel that:

- Uses only global memory
- Supports optional transposition of input matrices
- Updates the output matrix in place
- Includes proper error handling and correctness verification

The code adheres strictly to the assignment requirements and demonstrates correct use of CUDA's programming model.