Week 2 Assignment

Parallel Thinking & CUDA Programming Basics

GPU Programming using CUDA and Triton (WiDS'25)

Abhineet Agarwal
Dept. of Electrical Engineering, IIT Bombay

Overview

This assignment introduces hands-on CUDA programming. You will write your first CUDA kernels, manage GPU memory, and explore how different grid and block configurations affect correctness and scalability.

The focus this week is **correctness and understanding the execution model**, not performance optimization. Performance tuning will begin in Week 3.

Submission Instructions

Submit a single PDF summarizing your work, along with source files. Place your submission in the following folder:

```
week2/
  assignment.pdf
  vector_add.cu
  multiply_scale.cu
  relu.cu
```

1 Task 1: Implement Basic CUDA Kernels

Implement the following CUDA kernels from scratch:

1. Vector addition: $\mathbf{c} = \mathbf{a} + \mathbf{b}$
2. Elementwise multiply-and-scale: $\mathbf{c} = \alpha \cdot \mathbf{a} \odot \mathbf{b}$
3. ReLU activation: $y = \max(x, 0)$

For each kernel:

- Write a complete `.cu` program
- Allocate GPU memory using `cudaMalloc`
- Copy input data to the device using `cudaMemcpy`
- Launch the kernel with a reasonable grid and block configuration
- Copy the result back to the host
- Verify correctness against a CPU (NumPy or C++) implementation

You may use single-precision floating point (`float`) throughout.

Correctness Requirements

- All kernels must produce numerically correct results
- Bounds checking must be implemented to avoid out-of-range memory access. Without this check, threads beyond the array size will cause undefined behavior. (Check the first chapter of Operating Systems: Three Easy Pieces to get to know better why this happens.)
- Code should compile using `nvcc` without warnings

2 Task 2: Grid and Block Configuration Exploration

For the vector addition kernel, experiment with different problem sizes and block dimensions.

Input Sizes

Test the following vector sizes:

$$n \in \{10^3, 10^5, 10^7\}$$

Block Sizes

Test at least the following block dimensions:

$$\texttt{blockDim.x} \in \{32,\ 128,\ 256,\ 512\}$$

For each configuration:

- Compute the required grid size
- Record the total number of threads launched
- Verify correctness of the output

Briefly answer:

- What happens if the total number of threads exceeds n?
- Why is bounds checking required inside the kernel?
- Which configurations failed or behaved unexpectedly, if any?
- Based on Task 2, when might you choose a smaller block size vs. a larger one? What trade-offs do you anticipate (we'll explore these in Week 3)?

3 Task 3 (Optional): First Look at Timing

This task is optional but encouraged.

Measure execution time for the vector addition kernel using either:

- CUDA events (`cudaEvent_t`), or
- Python timing if embedding the kernel in a PyTorch/CuPy workflow

Record timing results for different block sizes and input sizes. Do **not** optimize yet. Summarize the results.

Notes

- You may work locally or on Google Colab
- You may reference CUDA samples, but all submitted code must be your own
- Clear, well-structured code.