Week 4 Assignment

Real GPU Kernels for Machine Learning & Scientific Compute

GPU Programming using CUDA and Triton (WiDS'25)
Abhineet Agarwal
Dept. of Electrical Engineering, IIT Bombay

The goal this week is to design, implement, and benchmark a meaningful GPU kernel, and to understand the performance trade-offs involved.

Submission Instructions

Submit a single PDF summarizing your design, implementation, and analysis, along with source files.

Your submission folder should look like:

```
week4/
  assignment.pdf
  kernel_naive.cu
  kernel_optimized.cu
  correctness_check.py
```

1 Task 1: Implement a Real GPU Kernel

Implement the following kernels in CUDA:

- **Softmax** (numerically stable) or **Dense matrix multiplication (GEMM)**
- **Port a PyTorch operation to CUDA**
  (e.g., elementwise op, reduction, normalization, or custom layer)

Requirements

- Implement a correct CUDA kernel using global memory only
- Match the mathematical behavior of a CPU or PyTorch reference
- Use appropriate grid and block dimensions
- Include bounds checking

Place this implementation in `kernel_naive.cu`.
Note: Correctness is mandatory. Performance comes later.

2 Task 2: Kernel Optimization & Memory-Aware Design

Optimize your kernel with the goal of improving performance on the GPU.

Possible techniques include (but are not limited to):

- Shared memory reuse
- Improved memory access patterns
- Reducing redundant global memory loads
- Kernel restructuring or fusion

Important: Unlike Week 3, you are not required to use shared memory. You must *decide* whether shared memory is beneficial for your kernel.

If shared memory is used:

- Explain what data is reused
- Explain how synchronization is handled

If shared memory is not used:

- Explain why it does not provide a benefit

Place the optimized implementation in `kernel_optimized.cu`.

3 Task 3: Benchmarking and Comparison

Benchmark the following implementations:

- CPU or PyTorch baseline
- Naive CUDA kernel
- Optimized CUDA kernel

Benchmarking Requirements

- Use consistent input sizes
- Run multiple iterations and report average runtime
- Clearly state GPU model and environment

Analysis

In your PDF, include:

- Runtime table or plot
- Speedup factors
- Identification of the dominant bottleneck (memory or compute)

4 Task 4: Reading Production GPU Code (Conceptual)

Answer the questions below based on these two repos:

- `tiny-cuda-nn` (NVIDIA)
- `FlashAttention`

Answer briefly (2–3 sentences each):

- What aspects of memory access or reuse stand out?
- What design choices differ from your implementation?

Notes

- You may use Google Colab or a local GPU
- You may reference CUDA documentation and public repositories
- All submitted code must be your own