Week 5 Assignment: CUDA vs Triton Softmax

Abhishek Upadhya 24B1309

1 Task 1: Triton Softmax Implementation

The chosen kernel is a row-wise, numerically stable softmax. For each row, the maximum element is subtracted before exponentiation to avoid numerical overflow, followed by normalization using the sum of exponentials. In Triton, each program instance processes one row of the input tensor using block-level parallelism.

Correctness was verified by comparing the Triton output against PyTorch's reference softmax implementation. The maximum absolute error observed was 7.45×10^{-9}, confirming numerical equivalence.

2 Task 2: Benchmarking and Performance Comparison

2.1 Experimental Setup

- GPU: NVIDIA GTX 1650 Ti
- Frameworks: PyTorch, Triton
- Input tensor: 2D tensor with softmax applied row-wise
- Timing method: wall-clock time with CUDA synchronization
- Metric: average runtime over multiple iterations

2.2 Results

Implementation	Time (ms)	Relative Speed
CUDA / PyTorch Softmax	0.1979	1.00×
Triton Softmax	0.1936	1.02×

Table 1: Runtime comparison between CUDA (PyTorch) and Triton softmax implementations.

The Triton implementation achieves performance comparable to the optimized CUDA-based PyTorch softmax on the tested input size, despite using a higher-level abstraction and minimal manual tuning.

3 Task 3: CUDA vs Triton — Design Reflection

The softmax kernel is easier to express in Triton due to its high-level handling of indexing, launch configuration, and memory access. Compared to CUDA, Triton provides less explicit control over warp-level behavior, shared memory usage, and fine-grained synchronization. For research and rapid prototyping, Triton is preferable due to faster development and readability, while CUDA remains better suited for scenarios requiring maximum performance tuning and architectural control.

4 Conclusion

This work demonstrates that Triton can achieve competitive performance with significantly reduced implementation complexity. While CUDA offers unmatched low-level control, Triton provides a productive alternative for writing correct and efficient GPU kernels in a research setting.