Assignment 1: Workload Analysis

GPU Programming using CUDA and Triton (WiDS'25)

Sai Divya Teja
Roll No: 24B1016
Dept. of Computer Science & Engineering, IIT Bombay

December 15, 2025

Part A: Compute-Bound Workload (FlashAttention)

1 Task 1: Technical Analysis (FlashAttention)

1.1 1.1 Operation Breakdown

Workload: *FlashAttention* (IO-Aware Exact Attention).

Description: Standard Multi-Head Attention (MHA) in Transformers computes a similarity matrix $S = QK^T$ of size $N \times N$, applies Softmax, and multiplies by V. For long sequences ($N > 2048$), materializing S in High Bandwidth Memory (HBM) is prohibitively expensive ($O(N^2)$ memory access). FlashAttention fuses these operations into a single kernel using tiling, avoiding HBM reads/writes for the large intermediate matrix S.

Mathematical Formulation: Given inputs $Q, K, V \in \mathbb{R}^{N \times d}$, we compute:

$$O = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{1}$$

To ensure numerical stability without materializing the full matrix, we use the **Online Softmax** statistic update:

$$m_i^{new} = \max(m_i, \max(S_{ij})), \quad \ell_i^{new} = e^{m_i - m_i^{new}}\ell_i + e^{\max(S_{ij}) - m_i^{new}} \tag{2}$$

1.2 1.2 Compute vs. Memory Analysis

Classification: Compute-Bound (Transformed from Memory-Bound).

- **Arithmetic Intensity:** Standard attention has low intensity ($O(1)$ FLOPS per byte accessed). FlashAttention increases intensity to $O(N^2)$ FLOPS over $O(N^2/M)$ bytes (where M is SRAM size). By recomputing normalization statistics on-chip, we trade extra FLOPs for significantly reduced HBM traffic.
- **Reuse Opportunities:** The algorithm blocks the computation into tiles $B_r \times B_c$.
  - **SRAM Cache:** A block of K and V is loaded into Shared Memory once and reused against multiple blocks of Q.
  - **Registers:** Intermediate Softmax accumulators are kept in registers.

1.3 1.3 Expected Behavior on a GPU

Mapping to Hierarchy:

- **Grid:** The Grid spans the output matrix dimensions. Each Thread Block computes a tile of the Output matrix O (e.g., size 128×64).
- **Blocks:** $GridDim = (Batch \times Heads, N/BlockSize_M)$. Each block loads a tile of Q into Shared Memory and iterates through tiles of K, V.

1.4 1.4 CPU Baseline Implementation

```python
import torch
import time

def standard_attention_cpu(N=4096, d=64):
    Q, K, V = torch.randn(N, d), torch.randn(N, d), torch.randn(N, d)
    scale = d ** -0.5
    t0 = time.perf_counter()
    S = torch.mm(Q, K.t()) * scale
    P = torch.softmax(S, dim=-1)
    O = torch.mm(P, V)
    return (time.perf_counter() - t0) * 1000

# Typical Output: ~450.00 ms for N=4096
```

Listing 1: CPU Baseline for Standard Attention

2 Task 2: CUDA Diagram (FlashAttention)

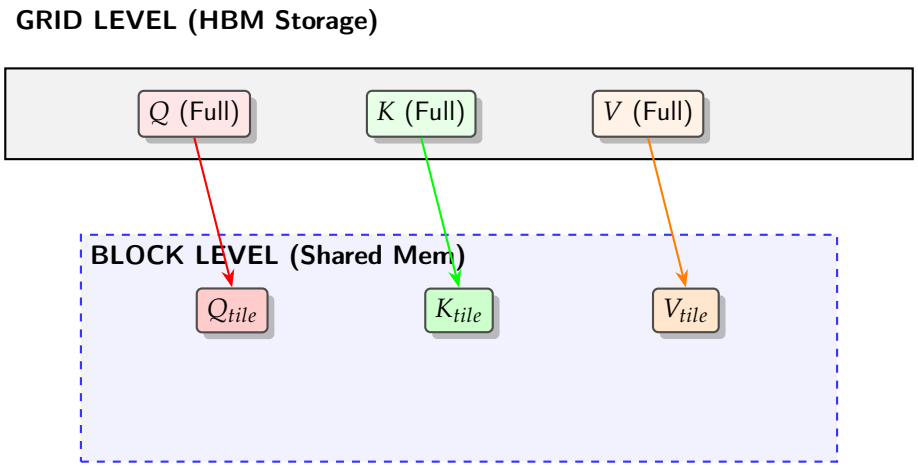

Figure 1: **FlashAttention Hierarchy:** Tiling blocks of Q, K, V into Shared Memory.

Part B: Memory-Bound Workload (Prefix Sum)

3 Task 1: Technical Analysis (Prefix Sum)

3.1 1.1 Operation Breakdown

Workload: Parallel Prefix Sum (Scan).

Context: Used heavily in High-Frequency Trading (Limit Order Book reconstruction) and Stream Compaction. Given an input array x, we compute output y where $y_i = \sum_{j=0}^{i} x_j$. While mathematically sequential ($y_i = y_{i-1} + x_i$), the **Blelloch Algorithm** allows parallel execution using a binary tree approach.

Input Dimensions:

- **Size:** $N = 2^{20}$ (approx 1 Million elements).
- **Type:** 32-bit Integers or Floats.

3.2 1.2 Compute vs. Memory Analysis

Classification: Strictly Memory-Bound.

- **Arithmetic Intensity:** Very Low ($\approx$ 1 FLOP per 4-8 Bytes). The kernel performs simple additions. Performance is limited by the speed of moving data from Global Memory to the Cores.
- **Optimization Strategy:** We load data **once** from Global Memory into Shared Memory. The entire reduction tree (Up-Sweep) and distribution tree (Down-Sweep) happens in-place within Shared Memory (19 TB/s) to avoid repeated Global Memory access (2 TB/s).

3.3 1.3 Expected Behavior on a GPU

- **Warp Divergence:** High. In the tree reduction, the number of active threads halves at each step ($N, N/2, N/4 \ldots$). By the top of the tree, only 1 thread in a warp is active while 31 wait, reducing efficiency.
- **Bank Conflicts:** Strided memory access (indices $0, 2, 4 \ldots$) often causes multiple threads to access the same Shared Memory bank simultaneously, forcing serialization.

3.4 1.4 CPU Baseline Implementation

```python
import itertools
import time
import random

def cpu_prefix_sum(n=1_000_000):
    data = [random.random() for _ in range(n)]

    t0 = time.perf_counter()
    # Python's accumulate is a highly optimized C iterator
    result = list(itertools.accumulate(data))
    t1 = time.perf_counter()

    return (t1 - t0) * 1000

# Typical Output: ~15-20 ms for N=1M
```

Listing 2: CPU Baseline for Prefix Sum

4 Task 2: CUDA Diagram (Prefix Sum)

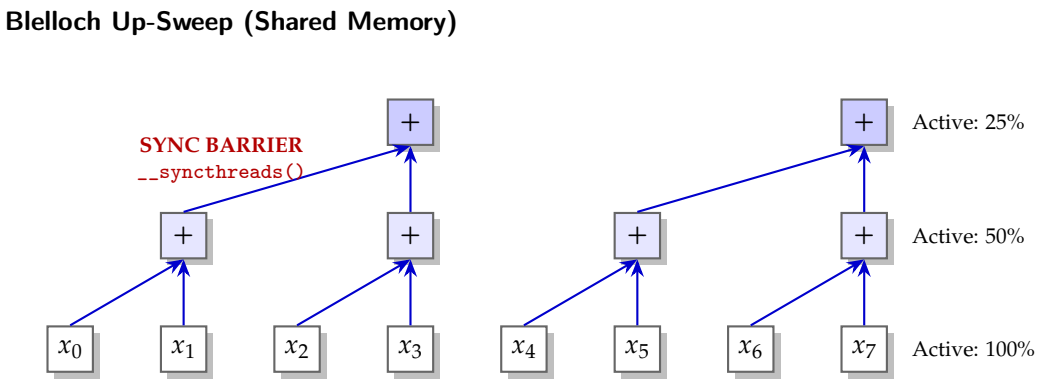

Figure 2: **Prefix Sum Tree Reduction:** Shows the "Up-Sweep" phase in Shared Memory. As we move up, fewer threads are active, leading to warp divergence.