Assignment 3 Report - Andrew Chan

All experiments were conducted on an NVIDIA A100 GPU. Matrix dimensions were m = k = n = 10,000 using single-precision floating point values. For Task 1, 10 independent matrix multiplications were executed per run. Kernel execution time was measured using CUDA events and throughput was reported in GFLOPs using:

GFLOPs = $2 * m * k * n * \text{NUM\_MULS}/(\text{execution time in seconds} * 10^9)$

Task 1 - Batched Matrix Multiplication with Streams

In Task 1 a shared memory tiled matrix multiplication kernel was implemented using 16×16 thread blocks. Each thread block will load tiles of matrices, A and B, into shared memory and computes partial dot products to reduce global memory traffic.

Two execution models were being evaluated:

1. **Baseline (Sequential Execution)**
   Each multiplication performs:
   - Host-to-device copy
   - Kernel execution
   - Device-to-host copy
     All operations are executed sequentially.
2. **Streamed Execution**
   Ten CUDA streams were created (one per multiply).
   Asynchronous memory copies (`cudaMemcpyAsync`) and kernel launches were issued in separate streams to allow overlap of memory transfers and computation.

Results:

```
[Baseline] 10 multiplies, total time: 6898.080 ms, throughput: 2899.36 GFLOPs
[Streams]  10 multiplies, total time: 6728.680 ms, throughput: 2972.35 GFLOPs
```

Task 2 – AoS vs SoA Grayscale Conversion

For task 2 we were tasks with comparing the performance of two grayscale image conversion layouts on the GPU:

1. an Array-of-Structures (AoS) layout using `uchar3`, and
2. a Structure-of-Arrays (SoA) layout where the R, G, and B channels are stored in separate arrays.

The goal is to quantify how memory layout affects memory coalescing and overall kernel performance.

- Image size: 2048 × 2048 pixels (RGB, 8-bit channels).
- AoS kernel: each thread reads a single `uchar3` from global memory and writes one grayscale

output byte.
- SoA kernel: host code converts the input `uchar3` array into three separate `uchar` arrays (R, G, B); the kernel reads from the three arrays and writes the grayscale result.
- Both kernels use the same luminance weights:
  `gray = 0.21 * R + 0.72 * G + 0.07 * B`.
- Thread configuration: 16×16 threads per block with a 2D grid covering the full image.

The GPU timing is measured with CUDA events around the grayscale kernels only.

```
AoS execution time:   253.84 ms
SoA execution time:   0.048 ms
```

Task 3 – Shared-Memory Image Blurring

In Task 3, a 2D image blur filter was implemented for four different blur radius (R = 1, 2, 4, 8).
Each blur computes the average of a (2R+1) × (2R+1) neighborhood around every pixel, with wrap-around boundary conditions so that accesses beyond the image edges wrap to the opposite side.

- Image size: 2048 × 2048, RGB, 8-bit channels stored as `uchar3`.
- Thread configuration: 16 × 16 threads per block, 2D grid covering the full image.
- For a given radius R, each block loads a shared-memory tile of size `(BLOCK_DIM + 2R) × (BLOCK_DIM + 2R)` that includes the interior pixels and the region.
- Global coordinates are wrapped with modular indexing so there are no dark borders or special-case branches at the image edges.
- For each output pixel, the kernel accumulates the sum of R, G, and B values over the local window and multiplies by a constant weight `1 / ((2R+1)^2)` to compute the blurred pixel.

As the blur radius increases, the output image becomes more heavily smoothed and fine details are eventually removed.

Conclusion

This assignment showed how memory layout, memory hierarchy, and execution scheduling affected GPU performance.