WiDS Week 3

Task 1: Global Memory Access Patterns

For a vector with a size of 1.6M elements, the execution times are as follows:

	Coalesced	Non-coalesced
Execution Time	1.2 ms	30.2 ms

Both kernels performed the same operation on the same amount of data. The non-coalesced version was approximately 25 times slower than the coalesced version.

For the coalesced kernel, consecutive threads within a warp access consecutive memory locations. Thus, all 32 memory requests in a warp can be combined into a smaller number of global memory transactions. In the non-coalesced kernel, consecutive threads access memory elements separated by a stride. The GPU is unable to combine these requests and has to make 32 separate global memory transactions instead of one. This leads to a large increase in execution time. The observed 25x slowdown directly reflects the loss of coalescing efficiency. Without any global memory access patterns, the GPU is unable to merge memory requests from threads in a warp.

Task 2: Shared Memory Optimization

For a vector with a size of 1.6M elements, the execution times are as follows:

	Baseline Kernel	Shared memory
Execution Time	1.07 ms	1.56 ms

Both kernels performed the following operation:

$$b[i] = \sum_{k=-5}^{5} a[i+k]$$

In theory, the global kernels need to load 11 elements from the global memory each whereas in the case of the shared memory kernel, each kernel loads only one element from the global memory onto the shared memory on average. Therefore, shared memory should be faster than the baseline kernel by at least 11 times. However, this is not true.

In practice, the baseline kernel also loads memory onto the L1 cache which is a considerable optimization. Each kernel does not load 11 elements from the global memory. Most of the elements are already present in the L1 cache. Therefore, the baseline kernel is already highly optimized. In case of the shared memory, data is first loaded onto the shared memory and then loaded by the thread again for computation which is worse than just using the cache. It is quite difficult to find an example where using a shared memory is faster as modern GPUs are highly optimised and use caches super efficiently. One could argue that the cache is the shared memory.

Shared memory improves performance only when it reduces global memory traffic or latency that the hardware cache cannot already hide. Shared memory helps when reuse is real and caches are insufficient or unpredictable.

Shared memory in this example shouldn’t suffer from bank conflicts. A bank conflict is a performance penalty that happens when multiple threads in the same warp access different addresses that live in the same shared-memory bank at the same time. Here, consecutive threads map to consecutive banks. Thus, there aren’t any bank conflicts.

Task 3: CPU Baseline

Baseline was submitted in week 1 itself.

Execution time: 1323 s