Lecture 19 Summary

This lecture is about CUDA Reduction. Parallel reduction is common and important data parallel primitive. It uses tree-based approach within each thread block. But there is a problem about communication between thread blocks for partial result. However, CUDA has no global synchronization because it is expensive to build in hardware for GPUs with high processor count. It would also force programmer to run fewer blocks to avoid deadlock, which may reduce overall efficiency. The solution is decomposition into multiple kernels. Kernel launch serves as a global synchronization point and it has negligible HW overhead and low SW overhead. The optimization goal is to achieve peak bandwidth of 86.4 GB/s for G80 GPU. The first method is interleaved addressing. In this method, each thread loads one element from global to shared memory, then do reduction in shared memory and write result to global memory. The problem for this method is that highly divergent warps are inefficient and % operator is very slow. The improved method is to replace divergent branch in inner loop with stride index and non-divergent branch. This makes a step speedup of 2.33x and a cumulative speedup of 2.33x. The second method is sequential addressing. Likely, it replaces stride indexing in inner loop of the improved method with reversed loop and threadID-based indexing. This makes a step speedup of 2.01x compared with the improved method and a cumulative speedup of 4.68x. A problem for above methods is that half of the threads are idle on first loop iteration, which is wasteful. To improve, they halve the number of blocks and replace single load with two lads and first add of the reduction. This makes a step speedup of 1.78x and a cumulative speedup of 8.34x. But it is still far from the peak bandwidth. Therefore, a likely bottleneck is instruction overhead and the strategy is unrolled loops for the last 6 iterations. This achieves a step speedup of 1.8x and a cumulative speedup of 15.01x. An improved method, completely unrolled, achieves a step speedup of 1.41x and a cumulative speedup of 21.16x. The seventh method, to improve the performance lose caused by algorithm, Multiple adds / Thread is used and results in a step speedup of 1.42x and a cumulative speedup of 30.04x.