EECS468 Lab 4 Journal

Adam Pollack and Gregory Leung

Part A:

Setting DEFAULT_NUM_ELEMENTS to 16777216 and MAX_RAND to 3
CPU Time: 46.263 (ms)
GPU Time: 18.827000 (ms)
Speedup: 2.457269x

Part B:

Our code defines the number of blocks as DEFAULT_NUM_ELEMENTS / BLOCK_SIZE. We keep a consistent block size of 1024 and we allocate the appropriate number of blocks based on input size. Each block computes a 1024 element chunk of the array and we use an “auxiliary array” to keep track of the running sum of all the previous chunks. We minimized shared memory bank conflicts by having each thread compute only one element of the chunk of the input. Each thread inherently accesses a separate bank. Our optimizations are listed below.

No Optimizations: Initial Solution

GPU Time: 1051.034058 (ms)
Speedup: 0.046707x

Goal: To develop a solution using UpSweep and DownSweep to compute Parallel Prefix Scan correctly.
Kernel: Two kernels: UpSweepKernel (performs a reduction on the input) and DownSweepKernel (finishes the scan).
Notes: Very naive implementation of handling large arrays. Used an auxiliary array to handle the combining of chunks of the input but have to pull values from this array repeatedly in the DownSweepKernel. Used nvprof and found that DownSweepKernel slows computation down substantially, further supporting the previous statement.
Hours to Complete: 10 hours

Optimization 1: AuxArrayScan

GPU Time: 18.827000 (ms)
Speedup: 2.457269x

Goal: Eliminate slowdown due to repeated accesses of auxArray
Kernel: UpSweepKernel, AuxArrayScan, and DownSweepKernel

Notes: Added a third kernel to compute the sum scan of the auxiliary array. This kernel does so using a simple serial computation.
Hours to Complete: 5 hours

Using nvprof on Optimization 1:
- UpSweepKernel: 7.22ms
- DownSweepKernel: 8.05ms
- AuxArrayScan: 3.67ms

Part C:

GFLOPS Calculations
GPU (Calculated based on program):
- UpSweepKernel: 1,023 FLOP/Block = 1,023 * 16,384 = 16,760,832
- AuxArrayScan: 16,384 (number of blocks)
- DownSweepKernel: 1,023 FLOP/Block + numElements = 1,023 * 16,384 + 16,777,216 = 33,538,048
- Total Floating Point Operations: 50,315,264
- GFLOPS = 50,315,264 / 0.018827s = 2.6725 GFLOPS

GPU (Theoretical)
- 8 SMs * 192 CUDA cores per SM * 1.058 GHz = 1,625.088 GFLOPS

CPU (Calculated based on program):
- Total Floating Point Operations: 2 * numElements = 2 * 16,777,216 = 33,554,432
- GFLOPS = 33,554,432 / 0.046263s = 0.7253 GFLOPS

CPU (Theoretical)
- 8 CPUs * 4 cores/CPU * 3.6GHz * 2 threads/core = 230.4 GFLOPS

Notes:
- Our GPU calculation has a GFLOPS than the CPU calculation
- We have a much lower GFLOPS than the theoretical value of the GPU because the algorithm itself doesn’t have complete utilization of the hardware
- The CPU calculation is only using one thread on one core on one CPU and it has to deal with hardware scheduling so it is significantly slower than the maximum theoretical value.