GPU — hw 2

Anirudhan J. Rajagopalan
N18824115
ajr619

October 25, 2016

Q1

1. 262144 (1024 * 256). Since number of blocks is 1024 and number of threads per block is 256.

2. There can be a maximum of 32 threads in a warp. In this case, since the number of threads is divisible by 32, all blocks has 32 threads in a warp.

3. 256. Given by ELEMENT_N.

4. Each thread does three global memory loads. One each for reading d_A (A[tx]), and d_B (B[tx]). And one for writing to d_C.

5. We will have a total of 1277 shared memory access per block.

   There are 256 threads in a block. For each thread we do one write in line 21 and one read in line 30. This gives total of 256 + 256 = 512 access without considering the code inside the for loop.

   The statement within the for loop concerning shared memory can be expanded as accumResult[tx] = accumResult[tx] + accumResult[tx + stride]. Here we peroform two reads and one write for a total of three shared memory access. We can clearly see that the number of shared memory acces within the for loop gets halved at each iteration from 128 to 1 for a total of 255*3 = 765 shared memory access.

   Thus adding both the values we get 765 + 256 + 256 = 1277 memory access.

6. 5 iterations will have branch divergence. We know that the statement in the true path of the if condition is executed by 128, 64, 32, 16, 8, 4, 2, 1 threads in every iteration. We also know that threads execute as a warp which is a set of 32 threads. The first three iterations wont have branch divergence as all threads take the same path. But the last 5 threads will have branch divergence as they have less than 32 threads taking the if and else paths.

7. There are three places where we access global memory. We cannot eliminate the first two of these access as they have to be done to fetch the values requried for computation. We can trade the last global memory access for branch divergence.

   We can use only thread 0 to copy the value from shared memory to global memory. This will require an if condition to check for the thread index which will involve branch divergence in the first warp. This eliminates $255 \times 1024 = 261120$ global memory access.

Q2

There are three main reasons why two different warps of the same block can take different amount of time.

1. Branch divergence — Suppose a warp has half of the threads taking one path and the rest of the threads take a different path, we know that the different paths are executed sequentially. This means that a thread inside a warp with branch divergence will take more time than the one without.

2. Memory Locality or Cache misses — When a thread accesses values that are not local or are not cached by L1 cache or shared memory.

3. Non uniform memory access — When a thread has to access memory differently based on the position it belongs to.

Q3 — Shared memory vs L1 cache

Shared memory is software managed and should be done explicitely by the programmer whereas L1 cache is managed by the hardware and is managed automatically by the hardware.

Shared memory should be allocated and evicted explicitely (__shared__ keyword). L1 cache is evicted/replaced based on the access pattern of the code.

Q4 — Can memory be coalesced

Yes. Depending on the branch taken by the set of threads inside a warp, a warp inside a block can find coalesced memory whereas another warp might find non-coalsced memory.

Q5

Given that the thread blocks should be a squre. Which means, we can have a maximum of $32 \times 32 = 1024$ threads in a block. Any other dimensions of the block will reduce the number of threads available within the block.

Since we have 32×32 blocks we will have a grid of size $\lceil \frac{400}{32} \rceil \times \lceil \frac{900}{32} \rceil = 13 \times 29$.