Jaidon Lybbert
EEP524
Homework 4
11/16/2022

HW4 GPU Compute CUDA Questions

1. For the tiled matrix-matrix multiplication kernel discussed in Lectures 3, 4 & 5, if we use a 32X32 tile, what is the reduction of memory bandwidth usage for input matrices A and B?

   C. With the untiled approach, each thread reads one row of A and one column of B. The total number of reads is width(A) * height(B) * height(A) * width(B). In the tiled approach, all threads in the same block share their reads from memory where applicable. The reads from A are shared between threads in the same row of A. The reads from B are shared between threads in the same columns of B. So, there is a 1/32 reduction in reads from A, and a 1/32 reduction in reads from B. So, the overall reduction in memory bandwidth usage is 1/32.

2. For the tiled single-precision matrix multiplication kernel as discussed in Lectures 3, 4 &5, assume that the tile size is 32X32 and the system has a DRAM burst size of 128 bytes. How many DRAM bursts will be delivered to the processor as a result of loading one A-matrix tile by a thread block?

   B. A single precision floating point value is 4 bytes. The width of the tile is 4*32=128 bytes. Each value in a row is in contiguous memory, so one DRAM burst will load one row of the tile. Since there are 32 rows in the tile, 32 DRAM bursts will be needed to load one tile.

3. Assume a tiled matrix multiplication that handles boundary conditions only on the right edge and lower (bottom) edges. Assume that we use 32X32 tiles to process square matrices of 1,000x1,000. Within EACH thread block, what is the maximal number of warps that will have control divergence due to handling boundary conditions for loading input matrix M (see L4 slides 31-35) tiles throughout the kernel execution?

   A. There are ceil(1000/32)=32 tiles needed to cover the input matrices horizontally and vertically, with 1000%32=8 unused threads overhanging at the right and bottom boundaries. The bottom boundary condition doesn't result in control divergence, because the entire overhanging rows (warps) are in agreement about not doing anything. The maximum number of diverging warps will happen at the right boundary (except for the bottom right tile) where all 32 rows (warps) are overhanging the edge, and every one of the 32 warps has control divergence.

4. We are to process a 600X800 (800 pixels in the x or horizontal direction, 600 pixels in the y or vertical direction) picture with the PictureKernel(). That is m's value is 600 and n's value is 800.

```
__global__ void PictureKernel(float* d_Pin, float* d_Pout, int n, int m) {
    // Calculate the row # of the d_Pin and d_Pout element to process
    int Row = blockIdx.y*blockDim.y + threadIdx.y;
    // Calculate the column # of the d_Pin and d_Pout element to process
    int Col = blockIdx.x*blockDim.x + threadIdx.x;
    // each thread computes one element of d_Pout if in range
    if ((Row < m) && (Col < n)) {
        d_Pout[Row*n+Col] = 2*d_Pin[Row*n+Col];
    }
}
```

   Assume that we decided to use a grid of 16X16 blocks. That is, each block is organized as a 2D 16X16 array of threads. How many warps will be generated during the execution of the kernel?

   C. The image is ceil(600/16)=38 blocks high with 600%16=8 rows of overhang at the bottom boundary. The image is ceil(800/16)=50 blocks wide with 800%16=0 columns of overhand at the right boundary. Each block contains ceil(16*16/32)=8 warps. The total number of warps is 38*50*8=15,200.

5. In Question 14, how many warps will have control divergence?

   D. All warps overhanging the right boundary will have control divergence. In this layout, that is 0 warps. Since warps occupy two rows of the block, any warps which are split at the bottom boundary will have divergence. In this layout, that is also 0 warps. The total diverging warps is 0.

6. In Question 14, if we are to process an 800x600 picture (600 pixels in the x or horizontal direction and 800 pixels in the y or vertical direction) picture, how many warps will have control divergence?

   G. All warps overhanging the right boundary will have control divergence. In this layout, that is ceil(800/2)=400 warps. Since warps occupy two rows of the block, any warps which are split at the bottom boundary will have divergence. In this layout, that is 0 warps. The total diverging warps is 400.

7. In Question 14, if we are to process a 799x600 picture (600 pixels in the x direction and 799 pixels in the y direction), how many warps will have control divergence?

   i. All warps overhanging the right boundary will have control divergence. In this layout, that is ceil(799/2)=400 warps. Since warps occupy two rows of the block, any warps which are split at the bottom boundary will have divergence. In this layout, that is 38 warps. However, we counted the bottom right warp twice, so the total diverging warps is

400+38-1=437 warps.

8. Assume the following simple matrix multiplication kernel

```
__global__ void MatrixMulKernel(float* M, float* N, float* P, int Width)
{
    int Row = blockIdx.y*blockDim.y+threadIdx.y;
    int Col = blockIdx.x*blockDim.x+threadIdx.x;
    if ((Row < Width) && (Col < Width)) {
        float Pvalue = 0;
        for (int k = 0; k < Width; ++k) {Pvalue += M[Row*Width+k] * N[k*Width+Col];}
        P[Row*Width+Col] = Pvalue;
    }
}
```

   If we launch the kernel with a block size of 16X16 on a 1000x1000 matrix, how many warps will have control divergence?

   N. All warps overhanging the right boundary will have control divergence. In this layout, that is ceil(1000/2)=500 warps. Since warps occupy two rows of the block, any warps which are split at the bottom boundary will have divergence. In this layout, that is (1000%2)*ceil(1000/16)=0 warps. The total number of diverging warps is 500.

9. A CUDA device's SM (streaming multiprocessor) can take up to 1,536 threads and up to 8 thread blocks. Which of the following block configurations would result in the largest number of threads in each SM?

   S. We can calculate the unused threads in each case with the equation 1536-(threads per block)(8) and 1536%(threads per block) if the first calculation is negative for each block size.

   1536 - 64*8 = 1024 unused threads
   1536 - 128*8 = 512 unused threads
   1536%512 = 0 unused threads
   1536-1024 = 512 unused threads

   The best case scenario is a block size of 512, which results in 0 unused threads.

10. Assume that we want to use each thread to calculate two (adjacent) output elements of a vector addition. Assume that variable i should be initialized with the index for the first element to be processed by a thread. Which of the following should be used for such initialization to allow correct, coalesced memory accesses to these first elements in the following statement?

```
if(i<n) C[i] = A[i] + B[i];
```

A. In order for the memory access to be coalesced, threads must access contiguous memory. Option A results in loading block-width-sized chunks of the vectors, with each successive block skipping one block-width chunk for the first access.

11. Continuing from Question 20, what would be the correct statement for each thread to process the second element?

    D. To process the next element, each thread will index the element with the same threadIdx.x, but in the following block-width-sized chunk of the vector which was skipped in the first load.

12. Assume the following simple matrix multiplication kernel

```
__global__ void MatrixMulKernel(float* M, float* N, float* P, int Width)
{
  int Row = blockIdx.y*blockDim.y+threadIdx.y;
  int Col = blockIdx.x*blockDim.x+threadIdx.x;
  if ((Row < Width) && (Col < Width)) {
     float Pvalue = 0;
     for (int k = 0; k < Width; ++k) {Pvalue += M[Row*Width+k] * N[k*Width+Col];}
     P[Row*Width+Col] = Pvalue;
   }
 }
```

    Based on the coalesced access judging criterion in Lecture 5 (slide 35), which of the following is true?

    C. M[Row*Width+k] is not coalesced but N[k*Width+Col] and P[Row*Width+Col] both are

    Threads in the same row, but consecutive columns will simultaneously load contiguous memory from the N matrix, and write to contiguous memory in P. However, they will simultaneously load the same elements from M, not contiguous elements.