UCRIVERSIDE UNIVERSITY OF CALIFORNIA

CS/EE 217

Midterm

ANSWER ALL QUESTIONS

TIME ALLOWED 60 MINUTES

Question	Possible Points	Points Scored
1	24	
2	32	
3	20	
4	24	
Total	100	

Question 1] [24 Points]

Given a GPGPU with 14 streaming multiprocessor each supporting up to 1536 threads in up to 8 blocks. Each block can have up to 1024 threads. The GPU has 500 GFLOP peak performance. Assume further that the DRAM system has 8 channels, and can provide up to 150 GB/s.

For any three of the following, explain how it could harm performance and possible ways the program can be modified to reduce this effect. **Please be specific.**

(a) The application needs to access global memory to get one floating point value for every operation.

How this harms performance:

Technique/change that could reduce this effect:

(b) Memory accesses are not coalesced.

How this harms performance:

Technique/change that could reduce this effect:

(c) Control divergence:

How this harms performance:

Technique/change that could reduce this effect:

(d) Suboptimal block size selection in grid decomposition

How this harms performance:

Technique/change that could reduce this effect:

(e) **Question 2] [32 Points]**

We would like to launch a matrix multiplication kernel to multiply an 80X96 matrix A with a 96X40 matrix B with the simple matrix multiplication kernel using 16X16 thread blocks. Answer the following questions:

(a) (6 points) How many blocks will be launched if each thread is responsible for one element?

(b) (6 points) How many blocks if each thread is responsible for four elements?

(c) (10 points) Assume we use tiled with case (b) where each thread is responsible for four elements. For a non-boundary tile what is the ratio of global memory accesses in the tiled version to that in the simple matrix multiplication?

(d) (10 points) What are the considerations in selecting the tile size for matrix multiplication?

Question 3] [20 points]: Consider a tiled 3D convolution kernel on data of size 512x512x512. Assume mask size is 5x5x5. Assume that the size of the output set in the tile is 8x8x8.

(a) We considered two implementations that load the full input set into shared memory: the first had one thread per output set element, with some of the threads also helping in loading the halo elements. The second had one thread per input element. How many threads does each configuration need? Are they both feasible?

(b) What is the reduction in the number of global memory accesses per full tile compared to a non-tiled implementation?

Question 4 [24 points]

Work inefficient Prefix Sum kernel

```
__shared__ float   XY[BLOCK_SIZE];
int i = blockIdx.x * blockDim.x + threadIdx.x;
//load into shared memory
if (i < InputSize) { XY[threadIdx.x] = X[i];}
//perform iterative scan on XY
for (unsigned int stride = 1; stride <= threadIdx.x; stride
*=2) {
        __syncthreads();
        if(i>=stride)
           in1 = XY[threadIdx.x – stride];
        __syncthreads();
        if(i>=stride)
          XY[threadIdx.x]+=in1;
}
```

Work efficient Prefix Sum kernel

```
// XY[2*BLOCK_SIZE] is in shared memory
for(int    stride=1;    stride    <=    BLOCK_SIZE;
stride=stride*2)
   {
      int index = (threadIdx.x+1)*stride*2 - 1;
      if(index < 2*BLOCK_SIZE)
         XY[index] += XY[index-stride];
      __syncthreads();
   }
//post reduction step
…
```

(a) Explain work efficiency in the context of the two prefix sum implementations above

(b) For the work inefficient kernel, assuming a BLOCK_SIZE of 1024 threads, how many warps will have control divergence at the step when stride is 16.

(c) For the work efficient kernel, assume that we have 2048 elements (each block has BLOCK_SIZE=1,024 threads) in each section and warp size is 32, how many warps in each block will have control divergence during the reduction tree phase iteration where stride is 16?