CSCI-GA.3033-008

Graphics Processing Units (GPUs): Architecture and Programming

Fall 2013 – Dec 17th, 2013 (90 minutes)

NAME: **ID:**

- This exam contains 5 questions with a total of 30 points.
- The exam is open book/notes but no electronic devices.
- If you have to make assumptions to continue solving a problem, state your assumptions clearly.
- You answer on the question sheet. You can use extra white papers if you want.

1. [6 points] As a CUDA programmer, how does knowing about the concept of warps help you, especially that warps are transparent to the programmer?

2.[6 points] Suppose we have a compute bound application with enough parallelism. Discuss the pros and cons of each the following two strategies: (i) more blocks per gird and less threads per block (ii) more threads per block but less blocks per grid. Assume the total number of threads is fixed.

3. [8 points] Suppose an NVIDIA GPU has 8 SMs. Each SM has 32 SPs, but a single warp is only 16 threads. The GPU is to be used to add two arrays element-wise. Assume that the number of array elements is 2^{24}. Let t denote the amount of time it takes one thread (yes, just one) to perform the entire calculation on the GPU. The kernel code is shown below (num_threads is the total number of threads in the whole GPU):

```
__device__ void prob(int array_size) {
        int tid = threadIdx.x + blockIdx.x * blockDim.x;
        for ( int i=tid; i<array_size; i += num_threads )
                result[i] = a[i] + b[i];
        }
```

(*a*) What is the amount of time it takes if we use one block of 16 threads?

(b) What is the amount of time it takes if we use two blocks of 8 threads each?

(c) Justify why the above two answers are similar/different.

(d) Assume that 256 threads are enough to keep all SPs in the SM busy all the time. What is the amount of time it would take to perform the computation for one block of 1024 threads? Justify.

(e) Repeat question (d) above but with two blocks of 512 threads each.

4. [6 points] Given the following code (assume balls are already in GPU global memory and the variables balls_per_thread and delta are defined elsewhere) :

```
struct Ball {
    float position;
    float velocity;
};
struct Ball* balls;  /*
__device__ void update(float delta) {
        int start=(threadIdx.x+blockIdx.x*blockDim.x)* balls_per_thread;
         int stop = start + balls_per_thread;
         for ( int i=start; i<stop; i++ )
                balls[i].position += delta * balls[i].velocity;
}
```

If we assume that all threads are assigned to one block, explain **two** enhancements to speed-up the above code. You don't need to write code but show the parts that need enhancement, explain why they need enhancement, and what is your fix, and why your fix is actually better than the original.

5. [4 points] The line of code below checks for a special case to avoid calling an expensive square root. Describe a situation in which it makes sense for CUDA to do that, and a different situation when it makes no sense (meaning it would be faster to do the square root all the time). Assume that 50% of the time d is equal to 1.

```
if ( d == 1 ) s = 1; else s = sqrt(d);
```