CSCI-GA.3033-004

Graphics Processing Units (GPUs): Architecture and Programming

Fall 2014 – (90 minutes)

NAME: **ID:**

- This exam contains 7 questions with a total of 20 points.
- The exam is open book/notes.
- 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. **(3 points) A block is not assigned to an SM before it is given all the resources it needs beforehand.**
   a. What are these resources?

      registers - shared memory - slots in SM scheduler

   b. What is the advantage of doing so?

      zero-time scheduling

2. **(3 points) What is wrong with that piece of kernel code? How to deal with it if we need to syncthreads both in the if-body and else-body (i.e. ho to change that code yet preserve the semantic)?**

```
if {
   ….
__syncthreads() ;
}
else {
     ….
__syncthreads() ;
}
```

   May result in deadlock if some threads in the warp go to the if-part and others in the else-part.

   Move _syncthreads() outside the if-else part.

3. (2 points) Briefly explain why CUDA code optimized for one GPU might run inefficiently when executed on another model of GPU (assuming warp size is the same).

Different number and sizes of SMs
Different size and architecture of shared memory (e.g. separate or with L1 cache)
Existence and non-existence of L2 cache

4. (2 points) A kenel launch is non-blocking from the host perspective. If we have two kernels: kernelA and kernelB. The first kernel produces some results and leave them in the device global memory. The second kernel uses these results to do more calculations. Given the following piece of code at the host:

```
….
kernelA <<<a, b,>>>(arg1);
kernelB <<<a, b>>> (arg1);
```

Does kernelB face a problem of starting execution before the results generated by kernelA are ready? Justify.

No, both are assigned to the default stream and stream commands are executed in order.

5. (2 points) The fact that instructions in the warp are executed in lockstep makes branch divergence a big performance-loss problem. Then why GPU designers insist on this lockstep design?

To amortize the cost of fetching/decoding instructions, and reduce hardware needed for non-computational purposes.

6. (4 points) Suppose that your code needs to deal with large amount of data that does not fit in the device memory, explain briefly what you can do to overcome this problem. Assume that applications are usually NOT embarrassingly parallel (i.e. there may be dependencies that may need to be taken into account).

Tiling

Using streams to move data back-and-forth between CPU and GPU to get new data

Make the best use of shared memory

7. (4 points) For the following piece of code, indicate for each data structure (a, b, c, d, e, and N) the best place for it in the memory hierarchy of a GPU, and justify your answer. We will assume that the code can be modified to reflect your choices (you do not need to re-write the code).

```
#define N  512
float a[N], b[N], c[N][N], d[N], e[N];

__global  compute(float *a,  float *b, float *c, float *d, float *e) {
        float temp;
        int index= blockIdx.x*blockDim.x+ threadIdx.x;
        int j;

        for(j =0; j<N; j++){
                temp= (c[index][j] + b[j])*d[e[index]];
                a[index] =a[index]– temp;
        }
}
```

a: global memory and shared memory (read and write)
b, c: constant memory (if small) or global memory and make use of shared memory
d: accessed randomly (i.e. cannot be coalesced) so better stay in global memory
e: in global memory and fetched into a register