CSCI-UA.0480-003

Parallel Computing

Final Exam

Spring 2018 - May 14th (90 minutes)

NAME: **ID:**

- This exam contains 10 questions with a total of 50 points in **five pages.**
- If you have to make assumptions to continue solving a problem, state your assumptions clearly.

1. For the following code snippet:

```
for (i = 0; i < N - 2; i++) {
        a[i] += a[i + 2] + 5;
        x += a[i];
}
```

a. [2 points] Identify the loop-carried dependencies existent.

You need to look at what the sequential version does. It uses old versions of A[] elements to generate new versions. So, $a[i]_{new} = a[i+2]_{old} + 5$

b. [7 points] Write a parallel version of the code in OpenMP with the dependencies removed.

```
#pragma omp parallel for reduction (+:x)
for (i = 0; i < N - 2; i++) {
        int temp = a[i+2];
        #pragma omp barrier
        a[i] += temp + 5;
        x += a[i];
}
```

2. [6 points] State three reasons why knowing about the concept of warps helps you writing more efficient code. [Note: the question did NOT ask about the definition of warps!]

- Choose a number of threads per block to be multiple of warp size to increase utilization.
- Be careful about branch divergence.
- Make accessing memory more friendly for coalescing because memory coalescing happens per warp.

3. [4 points] For each one of the scenarios indicated in the table regarding GPUs: specify whether there can be a race condition, and why.

Scenario	Potential race condition (Y/N)?	Justify
two threads belonging to the same warp	Y	Even though threads in the same warp execute in lock-step, two threads can access the same memory location.
two threads belonging to the same block but different warps	Y	They can access the same shared or global memory location.
two threads belonging to different blocks but the same kernel	Y	They can access the same global memory location.
two threads belonging to the different kernels but same application	Y	They can access the same global memory location.

4. Suppose we have an array A of N elements. In sequential version we have this loop:

```
for (i = 0; i <N; i++) {
     // do something independent of other iterations with element A[i]
  }
```

a. [3 points] This array is divided among P processes in MPI. Modify the for-loop above such that each process is responsible of the same number of entries in A, except the last process in case N is not divisible by P. Assume each process knows its *processID* and has a variable *numprocs* that contains number of processes. Do not include anything except the for-loop-modification that will be executed by each process.

```
for (i = processID; i <N; i+=numprocs) {
     // do something independent of other iterations with element A[i]
  }
```

b. [3 points] Assume now we use OpenMP, and that each iteration's computations increase with "i". How can we parallelize this for-loop to ensure load-balancing?

```
#pragma omp parallel for schedule(dynamic,1) //guided is also acceptable
for (i = 0; i <N; i++) {
     // do something independent of other iterations with element A[i]
  }
```

c. [4 points] Finally, assume we will execute these iterations on a GPU using two blocks and N is even. Each thread will be responsible of one element of A.

- Assume the kernel is called compute, fill in the blank:

__global__ compute<<< ____2_______, ______N/2________>>>(void);

- What is the ID used by each thread to access its element of A?

blockIdx.x*blockDim.x + threadIdx.x

5. [3 points] When, in OpenMP, do we need to use tasks instead of sections? [The question did NOT ask for definitions of tasks and sections.].

If we do not know beforehand how many sections do we need, then we have to use tasks.

6. [3 points] When, in OpenMP, do we need to use sections instead of tasks? [The question did NOT ask for definitions of tasks and sections.].

Tasks require synchronization and single execution at the beginning. So, it is slower than sections. If we know beforehand the number of tasks needed, then sections is better.

7. [3 points] Why is it sometimes needed, in OpenMP, to use locks instead of critical sections?

To allow parallel execution of several threads onto different critical sections. We can do that by given critical sections a label. But sometimes we do not know beforehand how many critical sections we will have (e.g. accessing queues of different threads and we do not know the number of threads at compile time).

8. [3 points] In an OpenMP program, all parts of the code not covered by #pragma will be executed by all threads. Discuss whether this statement is true or false, and justify.

This statement is false. Code not covered by pragma is executed by a single thread.

9. [3 points] We know that CUDA does not allow synchronization among threads in different blocks. Suppose CUDA allows this. State one potential problem that may arise.

Not all blocks of the grid are executing at the same time. Some are waiting to be scheduled. This can be a cause of deadlock.

10. For a vector addition with CUDA, assume that the vector length is 4000, each thread calculates one output element, and the thread block size is 512 threads. The programmer configures the kernel launch to have a minimal number of blocks to cover all output elements. Answer the following questions and to get full credit, show all steps.

a. [3 points] How many threads will be in the grid? From that, how many warps will have branch

a. 4096
ceil(4000/512)*512 = 8 * 512 = 4096. Another way to look at it is the minimal multiple of 512 to cover 4000 is 512*8 = 4096.

b. [3 points] Based on your answer in a, how many warps will have branch divergence due to checking the boundaries of the vector (i.e. to ensure that it is not accessing data outside the vector)?

None, 4000 threads (i.e. exactly 125 warps) will have the boundary condition true, and the remaining 3 warps (for elements 4001 to 40096) will have the condition false.