CSCI-UA.0480-003

Parallel Computing

Final Exam

Spring 2019 - May 15th (90 minutes)

Last Name: **First Name:** **NetID:**

- This exam contains 6 questions with a total of 50 points in **six pages.**
- If you have to make assumptions to continue solving a problem, state your assumptions clearly.

1. For the following code snippet (questions a to f are independent from each other):

```
1.    #pragma omp parallel num_threads(8)
2.    {
3.          #pragma omp for schedule(static,1) nowait
4.          for(i=0;i<N;i++){
5.                a[i] = ....
6.          }
7.
8.          #pragma omp for schedule(static,1)
9.           for(i=0;i<N;i++){
10.         ... = a[i]
11.         }
12.   }
```

a. [6 points] For each one of the following situations, indicate whether there will be a possibility of race condition or now.

Scenario	**Race Condition (Y/N)**
The code as it is above	N
We remove "nowait" from line 3	N
We keep the "nowait" in line 3 but change the schedule in line 3 to (dynamic, 1)	Y
We keep the "nowait" in line 3 but change the schedule in line 8 to (dynamic, 1)	Y
We remove the "nowait" from line 3 and put it in line 8	N
We remove the whole line 1 and use clause "**omp parallel for**" in both lines 3 and 8 and remove the "nowait" of line 3	N

b. [2 points] How many threads exist at line 7 (that empty line between the two for-loops)?

8

c. [2 points] How many threads exist just after line 12?

1

d. [2 points] Suppose we totally remove line 1. Then we substitute line 3 with:
#pragma omp parallel for num_threads (8) nowait
and we leave line 8 unchanged. The loop iterations of the second for-loop will be distributed among how many threads?

1

e. [3 points] Suppose we just remove line 1 and leave everything else unchanged. Also suppose the current processor has a total of 4 cores. How many threads will execute the first for-loop? And how many will execute the second for-loop?

First for-loop: 1

Second for-loop: 1

f. [2 points] Suppose we remove both lines 3 and 8 and keep everything else unchanged, what will change in the execution? If nothing changes, just say nothing.

All the 8 threads will execute all iterations of both for-loop redundantly.

g. [2 points] What is the difference between the code shown in this problem and the same code but without line 1 and with both lines 3 and 8 using:
#pragma omp parallel for num_threads (8) ?

In the original code, the 8 threads are created once and used in both for-loops. In the second case, threads are created for the first for-loop, then deleted after the iterations, then created again for the second for-loop.

2. [4 points] In a GPU, can the number of threads in a block be more than the number of SPs inside the SM? If yes, then how can this block be executed in the SM? If no, then does the programmer need to know the number of SPs per SM to be able to launch the kernel? [Note: Do not write the definitions of SM, SP, thread, or blocks because no credits will be given to that.]

Yes, because the block is divided into warps and available SPs need to execute the threads in a warp in parallel not the threads in the whole block.

3. [6 points] We know that all NVIDIA GPUs are using warps of 32 threads. However, if we take a snapshot during the execution of a specific warp, we may not find 32 threads executing, but less. Specify three scenarios that can cause this to happen.

- **Scenario 1:** Programmer picked a number of threads per block that is not divisible by 32 and hence the last warp has less than 32 threads.

- **Scenario 2:** There is a branch divergence and hence some threads are executing the if-part and some other the else part.

- **Scenario 3:** Threads reach a non-coalesced memory access instruction and some threads received their data before the others.

4. If we have two kernels: kernelA and kernelB. The first kernel produces some results and 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);

a. [2 points] Since kernel launches are non-blocking from host perspective, aren't we facing a problem of kernelB starting execution before kernelA finishes? Justify.

No, this cannot happen because both kernels are in the same default stream. Hence, they are executed by the device in order.

b. [2 points] We said that kernelA produces results that kernelB uses. Where can kernelA store these results so they can be accessed by kernelB? Explain in 1-2 sentences why your suggestion works.

The first kernel can leave the result in the global memory of the device hence it will remain there will the end of the application and can be accessed by any kernel of the application.

c. [2 points] Can we make two kernels execute on the same GPU at the same time? If yes, explain how. If not, explain why not.

Yes, but launching them from two different streams.

5. Suppose we have the following part of a kernel that will be executed by two threads: T1 and T2. Assume we have only those two threads for this problem.

```
__global__  int x; // x will be stored in the global memory

….. part of code of the kernel to calculate thread ID: tid ….

__shared__  int y = 0; // initialized to zero
int z;

if (tid == T2)     { y = 1; }

if (y == 1)  { z = 1; }
else  { z = 2; }

__syncthreads();

if (tid == T1)    x = z;
```

What will be the final value (or possible values if there is more than one) of x if:

a. [2 points] T1 and T2 are in the same warp.

1

b. [2 points] T1 and T2 are in the same block but not in the same warp.

1 or 2

c. [2 points] T1 and T2 are in two different blocks.

2

d. [4 points] Fill in the table with the number of copies of x, y, and z in each scenario.

Scenario: T1 & T2	copies of x	copies of y	copies of z
same warp	1	1	2
same block but not warp	1	1	2
2 different blocks	1	2	2

6. [5 points] Circle the correct answer among the choices given. If you circle more than one answer, you will lose the grade of the corresponding question.

a. L2 cache miss in the GPU always results in coalesced global memory access.
1. This statement is true. 2. This statement is false.
3. Depends on the warp size. 4. Depends on the global memory size.

b. Reduction operations in MPI take care of critical sections on the programmer's behalf.
1. This statement is true. 2. This statement is false.
3. Depends on the communicator size.

c. Suppose we have 4 processes in MPI. After the execution of MPI_finalize() and before exiting the program
1. All four processes end 2. All four processes end except process 0
3. All four processes still exist

d. Amdah's law
1. Gives upper bound of the speedup 2. Gives lower bound of the speedup

e. I have written my last name, first name, and NetID at the top of the first page of this exam.
1. True 2. False 3. Depends on L2 cache size!