CSCI-UA.0480-051: Parallel Computing
Final Exam (Dec 21st, 2021)
Total: 100 points

Important Notes- READ BEFORE SOLVING THE EXAM

- **If you perceive any ambiguity in any of the questions, state your assumptions clearly and solve the problem based on your assumptions. We will grade both your solutions and your assumptions.**
- **This exam is take-home.**
- **The exam is posted on Brightspace, at 2pm EST of Dec 21st.**
- **You have up to 23 hours and 55 minutes from 2pm EST of Dec 21st to submit your answers on Brightspace (in the assignments section).**
- **You are allowed only one submission, unlike assignments and labs.**
- **Your answers must be very focused. You may be penalized for wrong answers and for putting irrelevant information in your answers.**
- **You must upload one pdf file.**
- **Your answer sheet must have a cover page (as indicated below) and one problem answer per page (e.g., problem 1 in separate page, problem 2 in another separate page, etc.).**
- **This exam has 4 problems totaling 100 points.**
- **The very first page of your answer is the cover page and must contain:**
  - **You Last Name**
  - **Your First Name**
  - **Your NetID**
  - **Copy and paste the honor code showed in the rectangle at the bottom of this page.**

Honor code (copy and paste the whole text shown below, in the rectangle, to the first page of your exam)

- You may use the textbook, slides, and any notes you have. But you may not use the internet.
- You may NOT use communication tools to collaborate with other humans. This includes but is not limited to Google-Chat, Messenger, E-mail, etc.
- Do not try to search for answers on the internet. It will show in your answer and you will earn an immediate grade of 0.
- Anyone found sharing answers or communicating with another student during the exam period will earn an immediate grade of 0.
- **"I understand the ground rules and agree to abide by them. I will not share answers or assist another student during this exam, nor will I seek assistance from another student or attempt to view their answers."**

Problem 1

a. [10 points] There is a problem with the following code. Please explain the problem in 1-2 lines. Can we fix this problem? If yes, show how (either by writing few lines of code or by explaining what you will do.). If no, explain why not.

```
#pragma omp parallel for
for (int i = 1; i < N; i++)
{
 A[i] = B[i] – A[i – 1]; //A and B are arrays of int
}
```

The code as above has loop carried dependency.

We can fix this problem

A[1] = B[1] - A[0]
A[2] = B[2] - A[1] = B[2] – B[1] + A[0]
A[3] = B[3] - A[2] = B[3] -B[2] + B[1] - A[0]
So the loop body will be:

```
factor = 1  //before the outer for-loop but after the program
{  int sum = 0;
   for(j = i; j >0; j--)
      { sum += factor*B[j]; factor = -factor; }
  A[i] = (i%2 == 0? sum - A[0]: sum + A[0]);
}
```

b. [5] Hyperthreading core can execute several instructions at the same time. Superscalar core can also execute several instructions at the same time. What is the main difference then between the two? The answer must not exceed 1-2 lines.

The instructions executed in a purely superscalar core must come from the same thread, while those executing on a hyperthreading core can come from different threads.

c. [5] Does a hyperthreading core need to be also superscalar? Explain.

Yes, superscalar means the ability to execute several instructions at the same time. Hyperthreading core executes several instructions at the same time and can come from different threads. So, it needs to have superscalar capability.

d. [5] Does a superscalar core need to have branch prediction? Explain.

No, it does not. Branch prediction leads to better performance if the prediction is correct. A superscalar without branch prediction will still work but will stall whenever it faces a conditional branch instruction will the condition is evaluated in the execution phase.

Problem 2

Assume the following three processes in MPI and the operations they do at each time step of the program. Also assume that the reduction operation is MPI_MIN. The ranks of the three processes (0, 1, and 2) are their ranks in MPI_COMM_WORLD, and all the APIs called in the table below use MPI_COMM_WORLD.

Time	**Process 0**	**Process 1**	**Process 2**
0	x = 6; y = 7;	x = 8; y = 9;	x = 5; y = 6;
1	MPI_Allreduce(&x, &y, …)	MPI_Allreduce(&y, &x, …)	MPI_Allreduce(&x, &y, …)
2	*non-MPI code not affecting x and y*	MPI_Send(&y, 1, MPI_INT, 2, 0, …)	MPI_Recv(&y, 1, MPI_INT, 1, 0, …)
3	MPI_Allreduce(&y, &x, …)	MPI_Allreduce(&x, &y, …)	MPI_Allreduce(&x, &y, …)

Fill in the following table with values of x and y in each process:

[6] After Step 1:

	Process 0	Process 1	Process 2
x	6	5	5
y	5	9	5

[6] After Step 2:

	Process 0	Process 1	Process 2
x	6	5	5
y	5	9	9

[6] After Step 3:

	Process 0	Process 1	Process 2
x	5	5	5
y	5	5	5

a. [5] Suppose we have three communicators, including MPI_COMM_WORLD. How many ranks does each of the above three processes have? Explain in no more than two lines.

Each process will have two ranks because the split API creates two disjoint communicators with the same name.

b. [7] If we have a quad-core processor, can these three processes use all the cores at the same time? Justify in 1-2 lines. Assume each one of the four cores has superscalar capability but no hyperthreading capability. Of course, the code shown above for each process is just a small part of the full code of the process (i.e. the full code of each process is bigger than what is shown in this problem).

Yes, if one ore more of the processes are multithreaded.

Problem 3

Suppose we have the following code snippet. Assume all variables and arrays have been declared and initialized earlier.

```
1.      #pragma omp parallel
2.      {
3.
4.              for (i = 1 ; i < M; i += 2 )
5.              {
6.                  D[i] = x * A[i] + x * B[i];
7.              }
8.
9.              #pragma omp for
10.             for (i = 0; i < N; i++)
11.             {
12.                 C[i] = k * D[i];
13.             }
14.     } // end omp parallel
```

a. [10] Assume M = N. What do we have to write in line 3 above to get as much performance as possible from the above code? You must not have race condition and at the same time get the best performance. No need to write any explanations, just the line of code.

#pragma omp for nowait

b. [10] Repeat problem a, but assume M = 2N.

#pragma omp for nowait

c. [5] Based on your answer in b, do we need to write anything in line 8 to make the code even faster? If no, then just say no. If yes, then write the line you will include in line 8.

No.

d. [5] If M = 16, N = 8, and the multicore on which we run the code has 16 cores. How many threads do you think OpenMP runtime will create when executing line 1? Justify in no more than two lines. Assume no other processes/threads are using the multicore except the above code.

With static analysis, the compiler will realize that we have eight iterations in each loop. So, it will create eight threads.

Problem 4

[15] Choose one answer only from the multiple choices for each problem. If you pick more than one answer, you will lose the grade of the corresponding problem.

1. Suppose a block, when launching a GPU kernel, is composed of 33 threads. How many warps will be created for this block?
a. 1 b. 2 c. 4 d.5 e. Depends on the total number of blocks.

2. Which of the following is NOT an advantage of warps?
a. reduces complexity of the GPU hardware
b. reduces penalty of thread divergence
c. decouples the dependence of the number of threads in a block on the number of SPs in the SM.

3. If we have a local variable in the GPU kernel, it will surely go to registers.
a. True b. False c. Depends on the compute capability of the GPU.

4. More than one block *from two different kernels* can be assigned to the same SM at the same time.
a. True b. False c. Depends on the compute capability of the GPU.

5. If a critical section in OpenMP looks like **{ x += b;}** what is the best mutual exclusion technique? Assume no other critical sections exist in the program.
a. lock b. critical section c. atomic

6. If a critical section in OpenMP looks like **{ x += c; y+=b; }** what is the best mutual exclusion technique? Assume no other critical sections exist in the program.
a. lock b. critical section c. atomic

7. Sometimes, in OpenMP, we cannot know the number of critical sections in a program by just looking at the source code.
a. True b. False

8. Suppose we launch a kernel as follows: kernelname <<<4,4>>>(argument list …);
And suppose the kernel has a global variable x. How many instances of x will be created?
a. 16 b. 4 c. 8 d. 1 e. 2

9. An algorithm of O(n) can be faster than an algorithm of $O(n^2)$.
a. True b. False

10. An algorithm of $O(n^2)$ can be faster than an algorithm of O(n).
a. True b. False

11. Even though MPI is a distributed memory programming model, it can still run on a multicore processor, which is a shared memory hardware.
a. True b. False

12. Even though OpenMP is a shared memory programming model, it can still run on a distributed memory hardware system.
a. True b. False

13. Even though locks are faster than critical sections in OpenMP, sometimes we must use critical sections.
a. True b. False

14. Non-determinism exists in MPI, OpenMP, and CUDA.
a. True b. False

15. The less the synchronization points in your parallel program, the less sensitive your program will be to load-imbalance.
a. True b. False