CSCI-UA.0480-051: Parallel Computing
Final Exam (May 12th, 2022)
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 May 12th.**
- **You have up to 23 hours and 55 minutes from 2pm EST of May 12th 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, cover 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.
- 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. [5] A programmer is developing a program for a company. The programmer has parallelized 90% of the program. The programmer has been informed that the problem size will remain the same no matter how many cores will be used. What is the expected speedup on 10 cores? Show your steps. Make any needed assumptions if any. Round the speedup to the nearest single digit after the floating point.

[2] We will use Amdahl's law: Sp = 1/[F + (1-F)/p]
[2] We know that (1-F) = 0.9 and that p =10. So, it is easy to solve for Sp.
[1] Sp = 1/(0.1 + 0.9/10) = 5.3

b. [5] A programmer was able to write a parallel program, running on four cores, and obtain a speedup of 2. What is the fraction of the sequential part of this program? Show all the steps to get full credit. Assume the cores are superscalar but not with hyperthreading capability.

[2] We will also use Amdahl's law: Sp = 1/[F + (1-F)/p]
[2] We know that Sp = 2 and that p =4. So, it is easy to solve for F.
[1] 2 = 1/(F + (1-F)/4) → F = 0.3333

c. [15] State three reasons as to why coherence protocol has a negative effect on performance in a shared-memory machine.

- [5] Extra communication to tell other caches to invalidate a block and to receive acknowledgments from them that they have invalidated.
- [5] Delaying the core that wants to write to its cache till other caches acknowledge they have invalidated.
- [5] Extra cache misses for the caches that invalidated their block in case they want to access that block again.

Problem 2

Suppose we have 4 processes. Each one has an array of int called A. Assume these are the only processes in a communicator called NEWCOMM. Each array consists of four int, and initialized as follows:

	Initial values in each process's array			
Process 0	1	8	9	8
Process 1	2	7	10	15
Process 2	3	6	11	14
Process 3	4	5	12	13

a. [7] Each of the four processes has another array of four int called B. Array B in each process are not initialized. There is a function call that all processes made. The result of this call is that the content of arrays B changed to the following values. Write down that call with all its arguments. It is one line not more.

	Values in each process's array B after the call			
Process 0	0	0	8	8
Process 1	0	0	8	8
Process 2	0	0	8	8
Process 3	0	0	8	8

MPI_Allreduce(A, B, 4, MPI_INT, MPI_BAND, NEWCOMM);
[1] [1] [1] [1] [1] [1] [1]

b. [3] What is the *minimum* number of communicators that each one of these four processes can be a member of? Justify.

Two communicators [1], because MPI_COMM_WORLD contains all processes and cannot be deleted [2].

c. [5] Suppose we execute the above processes on a machine with 16 cores, each one is 2-way hyperthreading. Is there any chance these four processes will fully use all the cores simultaneously? Explain.

Yes [2], if one or more process is multithreaded and the total number of threads (on all processes) is the same or more than the total number of logical cores (16 x 2 = 32) [3].

Problem 3

a. [5] We know that statements: **#pragma omp single** and **#pragma omp master** both make its following statement be executed by only one thread. Give a scenario where we *must* use the **#pragma omp master** and not the other one.

In most MPI implementations, only process 0 (master process) can get input from the user from stdin. So, we will have to use the "master" version if we need to get inputs from the user using stdin.

b. [5] Given the following nested-loops, is it more beneficial to parallelize (using OpenMP), the outer loop? or the inner loop? And why?

```
for (i=1; i < n; i++) {
  for (j = 0; j < n; j++) {
     v[j][i] = (v[j][i-1] + v[j][i] + v[j][i+1])/4);
  }
}
```

We must parallelize the inner loop [2], because there is inter-loop dependencies in outer loop but not in inner loop [3].

c. [20] For each one of the following scenarios, state whether you will use atomic, critical, or locks in OpenMP to deal with critical sections. If there are several possible choices in a scenario, pick the one that gives the best performance. If there are several possible solutions for a scenario, each of which gives the same performance as the others, then write all of them. Be as specific as possible. No justifications are needed for your choices.

Scenario	What will you use?
Several critical sections, each one consists of several lines of code, and we know the number of critical sections at programming time.	#pragma omp critical (name) [3] We better give a name to each critical section to avoid performance loss. [2]
Several critical sections, each one consists of one line of code in the form of "variable++" (where variable is any variable name), and we know the number of critical sections at programming time.	#pragma omp atomic [5]
We don't know the exact number of critical sections at programming time, but we know them at execution time.	Locks [5]
Only one critical section that consists of several lines of code	#pragma omp critical [5]

If the full command is no written (i.e. #pragma omp …) no penalty is applied.

Problem 4

Suppose we have the following code snippet. The programmer launches the kernel with: **printing<<<3,2>>>( );**

```
__global__ printing(){

        __shared__ int x = 7;
        if(blockIdx.x > 0)
                printf("%d", blockIdx.x);
        else
                printf("x");
        __synchthreads();
        printf("%d", threadIdx.x);
}
```

a. [12 points] For each one of the following statements, specify whether it is a "possible" output or "not possible" output (i.e. can never be printed on the screen no matter how many times we execute the kernel). No need to justify your choice, just write (a. , b. , …) and next to each one ("possible", or "not possible").

a. 770100010001 not possible
b. 000100011101 not possible
c. 701020711121 possible
d. 777000000111 not possible
e. 000000011177 not possible
f. 001077010001 not possible

b. [6 points] If we remove the __shared__ keyword, which ones of your answers above will change? Justify.

None, they will stay the same [4]. Whether X is per thread or shared among the threads of each block, its value does not change [2].

c. [6 points] If we remove the __synchthreads(), which ones of your answers (in part a) above will change? Justify.

None, they will stay the same [4].
All the threads in the block are in the same warp and hence execute in a lockstep and do not need external synchronization. Also, 5 out of the 6 outputs in the previous question are not possible anyway. [2: one reason is enough to get the two points]

d. [6 points] How many warps are created when the kernel is launched? Explain.

Three warps [4].
One per block because each block has less than 32 threads [2].