CSCI-UA.0480-051: Parallel Computing Midterm Exam (March 14th, 2024)

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 the beginning of the March 14th lecture (2 pm EST).
- You have up to 24 hours to submit on Brightspace (i.e., until March 15th, 2 pm EST), in the same way as you submit an assignment. However, unlike assignments, you can only submit once.
- Your answers must be very focused. You may be penalized for giving wrong answers and for putting irrelevant information in your answers.
- Your answer sheet must be organized as follows:
  - The very first page of your answer must contain only:
    - Your Last Name
    - Your First Name
    - Your NetID
    - Copy and paste the honor code shown in the rectangle at the bottom of this page.
  - In your answer sheet, answer one problem per page. The exam has eight main problems, each one must be answered in a separate page.
- This exam consists of 8 problems, with a total of 100 points.
- Your answers can be typed or written by hand (but with clear handwriting). It is up to you. But you must upload one pdf file containing all your answers.

Honor code (copy and paste to the first page of your exam)

- You may use the textbook, slides, the class recorded lectures, the information in the discussion forums of the class on Brightspace, 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.
- You cannot use LLMs such as chatGPT, Gemini, Bard, 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, communicating with another student, searching the internet, or using prohibited tools (as mentioned above) 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] Suppose we have a core with only superscalar execution (i.e., no pipelining or hyperthreading). Will this core benefit from having a larger instruction cache? Justify your answer in 1-2 lines.

b. [10] Can a single process be executed on a distributed memory machine? If yes, explain how, in 1-2 lines. If not, explain why not.

c. [10] Can several threads, belonging to different processes, be executed on a shared memory machine and get the same performance as when executed on a multicore with the same number of cores? If yes, explain how, in 1-2 lines. If not, explain why not.

d. [6] If we have a two-way superscalar core, how many execution units for integer arithmetic operations would be ideal to get the best performance? Justify.

Problem 2

Suppose we have the following DAG that represents different tasks and their dependencies. The DAG is a simple linear chain: Task A -¿ Task B -¿ Task C -¿ Task D.

The following table shows the execution time of each task if we execute it on a core of type X and if we execute it on a core of type Y. Each CPU type is optimized for some type of operations.

Task	CPU Type X (ms)	CPU Type Y (ms)
A	10	15
B	5	8
C	12	10
D	7	6

a. [10] What is the minimum number of CPUs of each type that we need to get the highest speedup over sequential execution on CPU of type X? Show which CPU will execute which task(s) and calculate the final speedup.

b. [10] Repeat the problem above but using CPU of type Y.

c. [10] Suppose we can improve the algorithm to reduce the execution time of task C on CPU type X by 5ms. How does this impact the optimal speedup achievable using only CPU type X?

Problem 3

Suppose that MPI_COMM_WORLD consists of the four processes 0, 1, 2, and 3, and suppose the following code is executed after MPI has been initialized:

```
int x, y;
MPI_Status status;
if (my_rank == 0) {
  x = 10;
  MPI_Send(&x, 1, MPI_INT, 1, 1, MPI_COMM_WORLD);
} else if (my_rank == 1) {
  MPI_Recv(&x, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &status);
  y = x * 2;
  MPI_Send(&y, 1, MPI_INT, 2, 2, MPI_COMM_WORLD);
} else if (my_rank == 2) {
  MPI_Recv(&y, 1, MPI_INT, 1, 2, MPI_COMM_WORLD, &status);
  printf("Process 2 received: %d\n", y);
} else {
  //Process 3 does nothing
}
```

a. [9 points] What will be the values of x and y for each of the processes 0, 1, and 2 after executing the above code?

Process	x	y
P0		
P1		
P2		
P3		

b. [5] Explain the role of the tag in the MPI_Send and MPI_Recv functions in this code.
c. [5] What will happen if we remove the `else` block for process 3?
d. [5] What will happen if we change the tag in `MPI_Send(x, 1, MPI`$_INT, 1, 1, MPI_COMM_WORLD)$; $to 2$?

Problem 4

a. [10] Explain Amdahl's Law and its implications for the scalability of parallel programs.
b. [10] Describe a scenario where adding more processors to a parallel program does not improve performance, or even decreases it.

Problem 5

a. [12] Compare and contrast OpenMP and MPI. Discuss their strengths and weaknesses in relation to different types of parallel programming problems.

Problem 6

Consider a parallel program that sorts a large array of integers.
a. [8] Describe a parallel sorting algorithm suitable for this task.
b. [7] Discuss the factors that would affect the performance of your chosen algorithm.

Problem 7

a. [10] Explain the concept of false sharing in shared-memory programming and how it can negatively impact performance.
b. [10] Describe techniques to mitigate the effects of false sharing.

Problem 8

Suppose you have a parallel program that needs to compute a global sum of many partial sums from different processors.
a. [10] Design an efficient algorithm using MPI to compute this global sum, minimizing communication overhead.
b. [5] Analyze the communication complexity of your algorithm.