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 (2pm EST).
- You have up to 24 hours to submit on Brightspace (i.e. till March 15th 2pm 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 below.
  - In your answer sheet, answer one problem per page. The exam has forty-nine main problems, each one must be answered in a separate page.
- This exam consists of 49 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 the same process, be executed on a shared memory machine and get the same performance as when executed on a distributed memory machine? If yes explain how, in 1-2 lines. If not, explain why not. Assume each node of the distributed machine has one CPU only.
d. [6] If we have a two-way superscalar core, how many fetch units do we need to get the best performance? Justify.

Problem 2

Suppose we have the following DAG that represents different tasks and their dependencies. [The DAG shows tasks A, B, C, D, E, F, G. A has no dependencies. B depends on A. C depends on A. D depends on B and C. E depends on D. F depends on D. G depends on E and F.]

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 core of type Y. Each CPU type is optimized for some type of operations. That is, one type of CPU is not always faster than the other type for all tasks. You can ignore communication overhead among tasks. [The table shows execution times for each task on CPU types X and Y. Example: Task A: X=2, Y=3; Task B: X=1, Y=2; etc.]

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] Using better algorithm/programming, we can enhance the DAG a bit by adding an arrow. Adding an arrow means more dependency and potential worse performance, however, in some cases, it can lead to better performance. If you were allowed to add one arrow from the above DAG, which one would you add? And why?

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 (`my_rank` contains the rank of the executing process):

```
int x, y, z;
switch(my_rank) {
case 0:
x=1; y=2; z=3;
MPI_Send(&x, 1, MPI_INT, 1, 1, MPI_COMM_WORLD);
MPI_Recv(&z, 1, MPI_INT, 3, 2, MPI_COMM_WORLD, &status);
break;
case 1:
x=4; y=5; z=6;
MPI_Recv(&x, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &status);
MPI_Send(&y, 1, MPI_INT, 2, 3, MPI_COMM_WORLD);
break;
case 2:
x=7; y=8; z=9;
MPI_Recv(&y, 1, MPI_INT, 1, 3, MPI_COMM_WORLD, &status);
MPI_Send(&x, 1, MPI_INT, 3, 4, MPI_COMM_WORLD);
break;
case 3:
x=10; y=11; z=12;
MPI_Recv(&x, 1, MPI_INT, 2, 4, MPI_COMM_WORLD, &status);
MPI_Send(&z, 1, MPI_INT, 0, 2, MPI_COMM_WORLD);
break;
}
```

a. [9 points] What will be the values of x, y, and z for each of the 4 processes after executing the above code?

Process	x	y	z
P0			
P1			
P2			
P3			

b. [5] Is there a possibility that the communication among the 4 processes executes out of order? If yes, explain the reason. If not, why not?
c. [5] What will happen if we execute the above code with: `mpiexec {n 2`
d. [5] What will happen if we remove the `break` statement in case 0?

Problem 4

a. [5] If we have an application with high communication overhead. Does it have good scalability (i.e. as we keep increasing the number of cores, do we see speedup)? Assume the problem size is big enough.
b. [5] If we have two threads that have the same type and number of computations. And we assign each thread to a core. Does this necessarily mean we have load balance? Explain.

Problem 5

...

Problem 6

Problem 7

Problem 8

Problem 9

Problem 10

Problem 11

Problem 12

Problem 13

Problem 14

Problem 15

Problem 16

Problem 17

Problem 18

Problem 19

Problem 20

Problem 21

Problem 22

Problem 23

Problem 24

Problem 25

Problem 26

Problem 27

Problem 28

Problem 29