CSCI-UA.0480-051: Parallel Computing
Midterm Exam (Mar 9th, 2023)
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, assignments section, at the beginning of the Mar 9th lecture.**
- **You have up to 23 hours and 59 minutes from the beginning of the Mar 9th lecture to submit on Brightspace (in the assignments section). That is, you must submit before 2pm Friday March 10th.**
- **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 a 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 3 problems totaling 100 points.**
- **The very first page of your answer is the cover page and must ONLY 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 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 G-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: 2 points each] State five reasons why two threads, executing the same piece of code but on different data, starting execution at the same time on two different cores may not finish at the same time.

- One thread can have more cache misses.
- One thread can have more page faults.
- Memory access and cache access can be non-uniform (NUMA and NUCA) because the data accessed by the threads are different.
- One thread may be accessing more data than the other.
- The two cores executing the threads may be different. Or, if the cores are similar, one of them can get warmer and hence reduces its voltage and frequency, and hence, its speed.
- Even though the two threads execute the same piece of code, the data can lead one of the threads into a different path of execution (due to if-else, switch case, condition in for-loop, etc.)

Any other reason that makes sense is counted as correct.

b. [8] Suppose we have a sixteen-core processor. Fill-in the following table for each scenario.

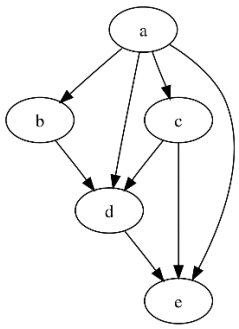
If each one of the 16 cores is	**The maximum number of *threads* that can execute simultaneously on the whole processor (i.e. on all the cores together) is:**	**The maximum number of *processes* that can execute simultaneously on the whole processor (i.e. on all the cores together) is:**
Only pipeline	16	16
Superscalar with no hyperthreading and each core has eight execution units	16	16
Two-way hyperthreading without branch prediction	32	32
Two-way hyperthreading with branch prediction	32	32

c. [7] "Dealing with critical sections among different processes, in MPI for example, can bring the performance down". Is this statement true or false? Justify.

The statement is false. [2]
Because processes are not sharing data. So, there are no critical sections with MPI. [5]

Problem 2

Assume we have the following task flow graph where every node is a task and an arrow from a task to another means dependencies. For example, task B cannot start before task A is done.



The following table shows the time taken by each task in cycles as well as the total number of machine code instructions executed by each task. An arrow, which indicates dependence, implies a communication between the two tasks.

Task	Total time taken (in cycles)	#instructions executed
A	10	100
B	5	50
C	15	40
D	10	80
E	5	30

a. [8] Cite all spans in the above DAG., and for each one, specify the number of tasks in the span, total number of instructions in the span, and total time (in cycles) of the span.

[2] There is only one span: A→C→D→E
[2] This span has four tasks.
[2] Totaling 250 instructions.
[2] And taking a total of 40 cycles.

b. [13] Calculate the CPI for each task in the DAG. Then specify which task is the best in terms of CPI. Show all calculations to get full credit.

[1] CPI = Cycles per instruction = total # cycles / total # instructions
[2] CPI_A= 10 / 100 = 0.1
[2] CPI_B= 5 / 50 = 0.1
[2] CPI_C= 15 / 40 = 0.375
[2] CPI_D= 10 / 80 = 0.125
[2] CPI_E= 5 / 30 = 0.17
[2] Since CPI is a lower the better measurement, the best task in terms of CPI is: A and B

c. [5] Suppose we have four identical cores. What is the best assignment of tasks to cores to achieve the best speedup relative to sequential execution? Use as few cores as possible to get the highest possible speedup. That is, you may or may not have to use all the available cores. Your answer must be like (task x assigned to core 1 and tasks y assigned to core 2, etc.). Assume the cores have a fixed frequency, that does not change, of 4 GHz.

We want to get the most parallelism and at the same time reduce communication among cores. Remember that a dependence between two tasks means there is a communication between them.
[1] The best assignment is to use two cores only.
[2] Core 0: tasks a, b, d, and e
[2] Core 1: task c
You can visualize the communication by seeing how many arrows need to go from a core to another when you assign the tasks from the DAG.
There are other possible correct assignments.

d. [5] What is the efficiency achieved using the assignment you specified in question c above? Show the steps.

Since we check the efficiency of using two cores, we will use the following definition of efficiency:
[1] Efficiency = speedup (with two cores) / 2
[2] Speedup (2) = time sequential / time parallel = 45 / 40 = 1.125
[2] Efficiency = 1.125 / 2 = 0.56

e. [4] We know that a four-way hyperthreading core can execute four threads at the same time. With four cores each one is just a superscalar (i.e., with no hyperthreading) we can also execute four threads at the same time. State two conditions that make the four-way hyperthreading core performs better than the four cores. Assume we have four threads.

- [2] The four threads need to communicate a lot of data among themselves. If they are in the same core, they can do that faster than if they are in different cores.
- [2] The four threads are not competing for the execution unit in the hyperthreading core (e.g. one thread needs integer operation, the other floating point, the third a lot of I/O, …). If this condition is not present, the contention on the execution units may cause performance degradation.

Problem 3

A programmer is developing a program for a company. The programmer has parallelized 80% of the program. For this problem 3, assume the number of threads = number of cores. Also, assume each core is just pipelined with no superscalar or hyperthreading capabilities.

a. [10] 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 8 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 as there is no other way to get the speedup given the above information.
[8] Speedup(8) = 1/(F + (1-F)/8) = 1/(0.2 + 0.8/8) = 3.3

b. [10] Keeping the assumption that the problem size will remain fixed, if we increase the number of cores from 8 to 16, what will be efficiency with 16 and with 8? What is your interpretation of the change of efficiency?

[1] Efficiency(p) = Speedup(p)/p
[3] With p = 8 Efficiency(8) = 3.3/8 = 0.417
[3] With p = 16 Efficiency(16) = speedup(16)/16 = [1/(0.2 + (0.8/16))]/16 = 0.25
[3] With 16 cores the efficiency drops. Because the problem size remains the same. And, by increasing the number of cores and number of threads, we are giving less work to each one. This means we are not making the best use of the resources.

c. [10] For the 20% of the program that the programmer couldn't parallelize, is there anything that we can do to make those 20% a bit faster? If yes, specify how. If not, explain why not.

The 20% is the sequential code. The programmer cannot parallelize that.
[1] There is a solution though.
The only way to go is to get a superscalar [3] core with large number of execution units[3] and high frequency[3] to execute this part of the code fast.

d. [10] Since Amdahl's law is just an approximation. Then, we may wonder why it is useful for. Specify two scenarios showing the usefulness of Amdahl's law.

- [5] The law can give us the theoretical upper bound of speedup. This is needed at an early stage of design to see whether the required speedup can be achieved or not.
- [5] If we have a specific algorithm to solve a problem, the law can help us make a decision whether to use a different algorithm with higher parallel portion, or to keep the current algorithm but use more cores.