CSCI-UA.0480-051: Parallel Computing
Midterm Exam (Oct 19th, 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 the beginning of the Oct 19th lecture.**
- **You have up to 23 hours and 55 minutes from the beginning of the Oct 19th lecture to submit 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 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 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.**

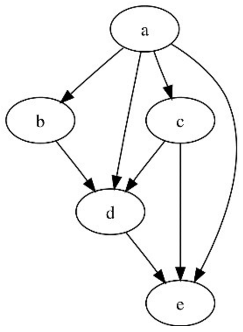
Honor code (copy and paste to the first page of your exam)

- 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."**

Note: square brackets indicate the points of each question (or part of a question).

Problem 1

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.



Suppose we have two types of cores: type 1 and type 2. The following table shows the time taken by each task, in nano seconds, if executed on a core of type 1 and if executed on a core of type 2.

Task	Time Taken on core type 1	Time Taken on core type 2
a	5	5
b	10	5
c	15	20
d	5	5
e	10	5

a. If we use all cores of type 1, what will be the span (indicate tasks and total time) of the DAG?

a→c→d→e [3]
with total time 5+15+5+10 = 35 [2]

b. If we use all cores of type 2, what will be the span (indicate tasks and total time) of the DAG?

a→c→d→e [3]
with total time 5+20+5+5 = 35 [2]

c. What will be the smallest number of cores, of any type, that gives the best speedup compared to using a single core of type 1? You can use a mix of any cores (e.g. two cores of type 1 and one core of type 2, etc). In your solution, indicate which task will run on which core, the total number of cores you will use for each type, the total execution times (for parallel version and the sequential version running on core of type 1), and the speedup (relative to sequential execution on core of type 1).

We need to think using two facts:

- We can have only two tasks to be executed in parallel b and c.
- We need to pick the fastest core for each task.

Core of type 2 is the faster, or equal, to core of type 1 in all tasks except task c. Therefore, we can pick two cores: one of type 1 and the other of type 2. [4]

Given that, the execution will be as follows: [4]

- a: on either of them →5
- b and c in parallel: task c on core type 1 (15) and on core type 2 (5) → 15
- d: on either → 5
- e: on type 2 → 5

Making the total execution time: 5+15+5+5 = 30 [3]
Sequential execution time on core of type 1 = 5+10+15+5+10 = 45 [2]
Speedup = 45/30 = 1.5 [2]

d. Suppose we use only cores of type 1. What is the smallest number of cores to get the highest speedup? Calculate that speedup. Then, if you are allowed to remove only one arrow from the DAG, while keeping the DAG a legal one, what will be that arrow to give a better speedup than the one you just calculated? If there are several solutions, pick one solution, and calculate the new speedup. If there are no solutions, state so, and give no more than two lines of explanation as to why there is no solution.

We need two cores of type 1 → execution = 5+15+5+10 = 35 [3]
Speedup = 45/35 = 1.3 [1]

If we remove the arrow c→d then the DAG is still legal [1]
The time taken with two cores will then be: [3]

- a : 5
- (b followed by d on one core) in parallel with (d on another core): 15
- e: 20

- The new speedup is then: 45/30 = 1.5 [2]

Problem 2

a. [5] For each one of the following designs, indicate whether it is SISD, SIMD, MISD, or MIMD. No justification needed. Note: If the design fits more than one category, then pick the more general one. For example, MIMD can execute as SIMD if all the instructions are the same. The more general is MIMD so pick MIMD not SIMD.

1. (MIMD) single core with superscalar capability but without speculative execution and without hyperthreading technology
2. (MIMD) single core with superscalar capability and speculative execution but without hyperthreading technology
3. (MIMD) single core with superscalar capability, speculative execution, and hyperthreading technology
4. (MIMD) multicore chip where each core has pipelining only
5. (MIMD) multicore chip where some cores have pipelining only while other cores have superscalar capability without hyperthreading technology

b. [5] Indicate whether each statement below is true (T) or false (F). No justification needed.

1. (T) We can have a core with superscalar capability but without speculative execution.
2. (F) In a core with hyperthreading technology, instructions can come from different threads belonging to the same process but not from threads belonging to different processes.
3. (F) MISD is implemented in real life in GPUs even though multicore can do the same job.
4. (T) If we have two processes, with one thread each, executing on a shared memory machine, there is no coherence overhead that can result from their execution.
5. (F) If we have two processes, with multiple threads each, executing on a shared memory machine, there is no coherence overhead that can result from their execution.

c. If we have a multicore processor with eight cores, and two-way hyperthreading each, what is the largest number of *processes* that can execute *at the same time*? Explain your answer with no more than two sentences.

[2] The maximum = 16 processes

[3] This multicore can execute 16 threads at the same time (8 cores x 2-way hyperthreading each). So, if each process has one thread, then we can execute 16 processes in parallel.

Problem 3

a. Suppose you have an algorithm with four tasks that can be executed in parallel.

1. [6] What are the characteristics of those tasks that can make you decide to implement the program as one process with four threads as opposed to four processes with one thread each? State two characteristics to get full credit.

- Tasks are sharing a lot of data.
- Each task is not doing a lot of work to justify the extra overhead of creating a process (which is much higher than creating a thread).

2. [4] With four parallel tasks, it may seem that four cores will give the best speedup over sequential code. However, there may be cases where less than four cores can give the same speedup as the four cores. Give a brief description of such a case in no more than 2-3 lines.

If one of the tasks is taking much longer time than the others, for example it is taking as much as the total of the other three, then the other three cores will finish much earlier and stay idle. In which case, two cores may be enough.

b. If we have two implementations of the same algorithm and we found that one implementation has higher efficiency than the other. Does that always mean that the implementation with higher efficiency will always be faster? Justify in two sentences.

[2] No, higher efficiency does not imply higher speedup.
[2] A single core will always have an efficiency of 100% yet its speedup is 1. With more cores, we can get higher speedup while the efficiency may decrease.

c. We have two different implementations of the same algorithm. The first implementation has one million instructions where half of them are floating point instructions, and the other half are integer instructions. The second implementation has two million instructions where all of them are integer instructions. Suppose a floating-point instruction takes 10 cycles while an integer instruction takes one cycle. We execute these two implementations on a 2GHz machine. Assume single core processor, where this core is SISD.

1. What is the MIPS of each implementation?

MIPS = #instructions in millions / total time in seconds
total time in seconds = total # of cycles * cycle time = total number of cycles / frequency

[3] Implementation 1:
#instructions in million = 1
total time in sec = $(5*10^5*10 + 5*10^5*1)/(2*10^9) = 2.75*10^{-3}$
MIPS = $1/(2.75*10^{-3})$= 363.3

[3] Implementation 2:
#instructions in million = 2
total time in sec = $2*10^6/2*10^9=10^{-3}$
MIPS = $2/(10^{-3})$ = 2000 MIPS

2. What is the CPI of each implementation?

CPI = total number of cycles / total number of instructions

[3] Implementation 1: CPI = $(5*10^5*10 + 5*10^5*1)/1000000 = 5.5$
[3] Implementation 2: CPI = $(2*10^6)/2000000 = 1$

3. What is the total execution time of each implementation?

ET = IC*CPI*CT = #cycles/frequency

[3] Implementation 1: ET = $(5*10^5*10 + 5*10^5*1)/(2*10^9) = 2.75*10^{-3}$ seconds
[3] Implementation 2: ET = $(2*10^6)/ (2*10^9) = 10^{-3}$ seconds

4. State which implementation is better based on each of the following measurements: instruction count, MIPS, CPI, and execution time.
[2 each]

Measurement	**Impl. 1**	**Impl. 2**	**Which is better**
IC	10^6	$2*10^6$	1
MIPS	363.3	2000	2
CPI	5.5	1	2
ET	$2.75*10^{-3}$ seconds	10^{-3} seconds	2

Problem 4

For problem 4, assume we have one communicator: MPI_COMM_WORLD.
Suppose we have four processes. Each process has two arrays, of five integers each, A and B. B is declared as int B[5] but is not initialized and A is initialized as follows:
process 0: int A[5] = {1, 2, 3, 4, 5};
process 1: int A[5] = {11, 12, 13, 14, 15};
process 2: int A[5] = {6, 7, 8, 9, 10};
process 3: int A[5] = {16, 17, 18, 19, 20};

[3, with -0.5 for each mistake]
MPI_Allreduce(A, B, 5, MPI_INT, MPI_SUM, MPI_COMM_WORLD);

a. Write one MPI command that, if executed by all the four processes, the B arrays at each process will be:
process 0: int B[5] = {34, 38, 42, 46, 50};
process 1: int B[5] = {34, 38, 42, 46, 50};
process 2: int B[5] = {34, 38, 42, 46, 50};
process 3: int B[5] = {34, 38, 42, 46, 50};

[3, with -0.5 for each mistake]
MPI_Allreduce(A, B, 5, MPI_INT, MPI_SUM, MPI_COMM_WORLD);

b. Write one MPI command that, if executed by all the four processes, the A arrays at each process will be:
process 0: int A[5] = {6, 7, 8, 9, 10};
process 1: int A[5] = {6, 7, 8, 9, 10};
process 2: int A[5] = {6, 7, 8, 9, 10};
process 3: int A[5] = {6, 7, 8, 9, 10};

[3, with -0.5 for each mistake]
MPI_Bcast(A, 5, MPI_INT, 2, MPI_COMM_WORLD);

c. [4] Why MPI_Bcast() does not have an argument for the tag in a way similar to MPI_send()?

The tags are needed in case messages are received out of order from the same sender, in case the sender sends several messages, using MPI_Send() to the same destination. But MBI_Bcast() is blocking. So, even if one process executes several MPI_Bcast(), the second one won't be executed till the first is done, etc.