CSCI-UA.0480-051: Parallel Computing
Midterm Exam (Mar 17th, 2021)
Total: 100 points

Important Notes- READ BEFORE SOLVING THE EXAM

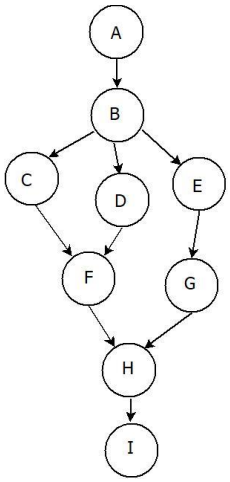
- **If you perceive any ambiguity in any of the questions, state your assumptions clearly and solve the problem based on your assumptions. We will grade both your solutions and your assumptions.**
- **This exam is take-home.**
- **The exam is posted, on NYU classes, at the beginning of the Mar 17th lecture.**
- **You have up to 23 hours and 55 minutes from the beginning of the Mar 17th lecture to submit on NYU classes (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 5 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."**

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.



The following table shows the time taken by each task in nano seconds.

Task	Time Taken
A	3
B	2
C	5
D	5
E	5
F	20
G	25
H	5
I	10

a. *Without* using the DAG method, that is without using T1/span, determine the minimum number of cores needed to get the highest performance [2 points]? Then, specify which tasks will execute on which core to get the best performance [6 points]. Finally, calculate the speedup given the number of cores you calculated [2 points]?

We only need two cores.

Core 0 will do tasks: A, B, C, D, F
Core 1 will do tasks: E, G, H, I
[Tasks A, B, H, and I can be assigned to either cores with the same performance]

To get speedup, we need T(1), that is with one core, and T(2).
T(1) = 3+2+5+5+5+20+25+5+10 = 80
T(2) = 3+2+5+25+5+10 = 50
Speedup = T(1)/T(2)=80/50=1.6

b. Calculate the span [2 points]. Which tasks form the span [5 points]? If there are more than one solution, please state all of them.

Span = 50
Span = A, B,E, G, H, I

c. [8 points] Fill-up the following table given the number of cores to execute the above DAG.

#cores →	**1**	**2**	**4**	**8**
speedup	1	1.6	1.6	1.6
efficiency	1	0.8	0.4	0.2

Problem 2

Answer the following questions about technology and hardware design.

a. [4 points] Superscalar capability can make some instructors execute out-of-order (for example, instruction 3 may execute before instruction 2), why is that?

Suppose we have four execution units and we have five instructions in the issue phase of the pipeline. If instruction 2 depends on the result of instruction 1, the issue phase will not send instruction 2 for execution till instruction 1 is done. To make use of the available execution units, instructions 1, 3, 4, and 5 (assuming they are independent from each other) will be issued for execution. When instruction 1 is done, then instruction 2 will be issued for execution.

b. [4 points] If we have a multicore processor with four cores, is there a possibility to execute more than four threads *at the same time*? If yes, explain how. If no, explain why not.

[2] Yes
[4] If each core that a hyperthreading technology.

c. [4 points] We have shared-memory systems and distributed memory-systems. With shared memory, it is easier for threads to exchange data. Then, why do we build distributed-memory systems?

As the number of cores increases (we can have thousands or even millions of cores in some machines) the shared memory will become a very severe bottleneck. This is why distributed memory systems are built where every few cores, in a node, share memory. But different nodes have different memory.

d. [4 points] In shared-memory system, as the number of nodes increases, how does this affect the overhead of coherence protocols? Justify your answer.

[2] The overhead of coherence increases.
[2] Because more cores means more caches, which means more messages for invalidation, etc. This leads to more bandwidth and more cache misses because the caches that invalidate a block and then need to access it will have a cache miss and will need to get that block again.

e. [4 points] We saw four different categories of architecture in Flynn's taxonomy. What is the classification of single core with hyperthreading capability? Justify your choice.

[2] If is a MIMD.
[2] Because more than one instruction can be executing in parallel and each instruction can be working with different data.

f. [4 points] In distributed-memory system, as the number of nodes increases, how does this affect the overhead of coherence protocols? Justify your answer.

[2] No effect whatsoever.
[2] Because in distributed memory systems there is no coherence needed because processes are not sharing the memory.

Problem 3

Answer the following questions regarding parallel programming techniques.

a. [8 points] If we look at a distributed-memory system with four nodes and found four threads executing in parallel (i.e., *at the same time*), how many processes do we have in the whole system? List all the possibilities and justify your choices. Assume each node consists of s single core with no superscalar or hyperthreading capabilities. That is, each core in the node executes one instruction at a time only.

[2] We have four processes in the system.
[6] Each process has a single thread. The four processes execute on four different nodes. So, we have four threads executing on the four nodes.

b. [6 points] Having too many threads (i.e. way more than the available cores) in an application may not be a good idea. State three reasons for that.

- Dividing the work among too many threads means each thread will do very little work. This cannot justify the overhead of starting, terminating, and keeping track of each thread state.
- Too many threads sharing fixed hardware resources is a source of congestion and hence performance degradation.
- Having too many software threads means that the operating system will have to schedule them on a round-robin fashion, giving each thread a time-slice after which it is suspended, and another thread is scheduled. These suspension-restoration operations are expensive.

[Other reasons that make sense will also be counted as correct.].

c. [6 points] Having too few threads, less than or equal to the number of available cores, in an application is not a good idea either. State three reasons explaining why. Assume the problem size is big enough.

- Creating a thread has an overhead. We need parallelism that overcomes this overhead. With too few threads we may not get enough parallelism and hence your parallel code can have worse performance than sequential code.
- The trend is to have more cores but simpler cores. Too few threads means each thread will be doing a lot of work sequentially. If that thread is assigned to a weak core, the performance will be affected.
- Too few threads means we may not be making the best use of the available hardware (i.e., the cores). We are paying for the cores and not using them.

[Other reasons that make sense will also be counted as correct.].

Problem 4

A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: Class A, Class B, and Class C, and they require one, two, and three cycles (respectively).
The first code sequence has 10 instructions: 2 of A, 6 of B, and 2 of C
The second sequence has 8 instructions: 4 of A, 2 of B, and 2 of C.

a. [4 pt] What is the CPI of the first sequence? Show all your calculations.
b. [4pt] What is the CPI of the second sequence? Show all your calculations.
c. [2 pt] Which sequence is faster based on CPI?
d. [5pts] Which sequence is faster based on ET? Show all the steps.

a. CPI = # cycles/#instructions = 2(1) + 6(2) + 2(3) / 10 = 2

b. CPI = 4(1) + 2(2) + 2(3) / 8 = 14/8 = 1.75

c. The second sequence is better based on CPI.

d. ET = #cycles * CT.
[2] For 1st sequence ET = 20*CT [2] For 2nd sequence ET = 14*CT
[1]So 2nd sequence is better.

Problem 5

Answer the following questions regarding MPI.

a. [6 points] If each process in a communicator call MPI_Reduce twice. That is, two reduction operations one after the other. Can some of the processes finish the first reduction operation and move to the next one before the other processes finish theirs? Justify.

[2] This cannot happen.
[4] Because reduction operation is a collective call and is blocking. So, the process will be blocked until all others processes are done with the first reduction before they all move to the next one.

b. [4 points] Why do MPI processes not suffer from cache coherence overhead?

Because each process has its own virtual address space. Therefore, processes are not sharing memory.

c. [6 points] Do Reduction operations in MPI take care of critical sections on the programmer's behalf? Justify.

[2] No they do not.
[4] Because there are no critical sections in MPI due to the fact that processes are not sharing the memory among each other.