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

Important Notes- READ BEFORE SOLVING THE EXAM

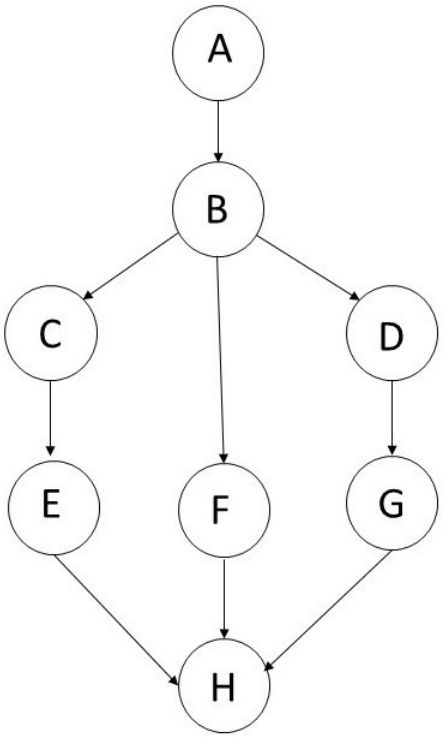
- **If you perceive any ambiguity in any of the questions, state your assumptions clearly and solve the problem based on your assumptions. We will grade both your solutions and your assumptions.**
- **This exam is take-home.**
- **The exam is posted on NYU classes, at 2pm EST of May 17th.**
- **You have up to 23 hours and 55 minutes from 2pm EST of May 17th to submit your answers 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 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 the whole text shown below 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

Suppose we have the following DAG, where each node represents an instruction. The table shows the amount of time taken by each instruction. Assume we are talking about machine level instructions, not high-level instructions.



Instruction	Time
A	5
B	5
C	10
D	10
E	5
F	10
G	10
H	5

a. [5] Suppose we execute the above DAG on a single core processor. This core executes one instruction at a time and does not support neither hyperthreading nor superscalar capabilities. You can neglect the effect of pipelining too. How long does this core take to finish executing the whole program? To get full credit, show the steps you did to reach the final answer.

It will simply execute one instruction at a time, satisfying the dependencies among instructions. The total amount of time is the sum of all instructions = 60

b. [5] Repeat the above problem but assume we have four cores. Each one of these cores is like the core described in question "a" above.

Still, it will take 60 time units because this program is not yet parallelized. [You can assume that the program is parallel. But then, you must specify the threads.]

c. Suppose we have four cores like problem "b" above. We want to aggregate the instructions in the DAG to form threads. Indicate:

- [3] How many threads you will form?
- [2] Give each thread a unique ID starting from 0.
- [3] Which instructions go into which thread, to get the best speedup?

- [2] Calculate that speedup and show all steps.

Three threads
Thread 0: A, B, F, H → 25 time units
Thread 1: C, E → 15
Thread 2: D, G → 20

Time taken with three threads = [A+B] + max[(C, E), (D, G), (F)] + [H]
= 10 + 20 + 5 = 35
Speedup = 60/35 = 1. 7

d. [5] What is the minimum number of threads that you can form, assuming four-core processor, that gives you the highest efficiency? Show all steps. [Hint: Assume Efficiency = speedup/#threads].

Efficiency = speedup / #threads
If we make #threads = 1 then speedup = 1 and Efficiency = 100%

Problem 2

Assume the following three processes in MPI and the operations they do at each time step of the same program. Also assume that the reduction operation is MPI_SUM.

Time	Process 0	Process 1	Process 2
0	x = 6; y = 7;	x = 8; y = 9;	x = 5; y = 6;
1	MPI_Allreduce(&x, &y, …)	MPI_Allreduce(&y, &x, …)	MPI_Allreduce(&x, &y, …)
2	MPI_Allreduce(&y, &x, …)	MPI_Allreduce(&x, &y, …)	MPI_Allreduce(&x, &y, …)

[6] Fill in the following table with values of x and y in each process:
After Step 1:

	Process 0	Process 1	Process 2
x	6	20	5
y	20	9	20

[6] After Step 2:

	Process 0	Process 1	Process 2
x	45	20	5
y	20	45	45

a. [4] Suppose each one of the above processes is parallelized with OpenMP with two threads. So, we have three processes and each one is parallelized with two threads. Will the values you calculated above change? If yes, specify how (no need to recalculate, just specific what factors affect them). If not, explain why not.

The values will not change. Because the MPI APIs work based on processes and is not affected by which thread of the process calls the API as long as the threads are from the same process.

b. [4] Suppose that the original program, the one with three processes but no OpenMP, is executed on a single-core processor. Will the values you calculated above change? If yes, specify how (no need to recalculate, just specific what factors affect them). If not, explain why not.

No, the values will not change. The MPI processes will work the same way regardless of the number of cores. The number of cores affects the performance but not the correctness of the code.

c. [6] If, for some reason, one of the three processes crashes after step 1 but before executing step 2. What will happen to the rest of the program? Justify.

The program will end in a deadlock. Because the MPI_Allreduce in step 2 will miss the call from the crashed process. Hence the other two process will be blocked forever.

Problem 3

Suppose we have the following code snippet is running on a four-core processor:

```
1. #pragma omp parallel for num_threads(8)
2. for(int i = 0; i <N; i++){
3.     for(int j = i; j < N; j++){
4.         array[i*N + j] = (j-i)!;
5.     }
6. }
```

(a) [5 points] How many iterations will each thread execute from the outer loop (line 2)? Justify

The number of iterations will be divided evenly among the threads. We have 8 threads and N iterations. So, each thread will execute N/8 iterations.

(b) [5 points] Will each thread executes the same number of iterations of the inner loop (line 3)? Explain.

No, because the number of iterations of the inner loop depends on the value of the loop index of the outer loop. Each thread will be a different set of loop index values of the outer loop.

(c) [5 points] Is there a possibility that (line 4) is a critical section? Justify.

No, because the elements accessed by each thread depends on i and j. Each thread will have a distinct combination of i and j.

(d) [24 points, 3 for each entry] For each one of the following scenarios, indicate how many threads will be created and what is the maximum number of threads that can execute in parallel given the code shown above. The number of cores is fixed as indicated at the beginning of this problem.

Scenario	**#threads created**	**Maximum #threads in parallel**
Cores do not support hyperthreading but have superscalar capability	8	4
Cores do not support hyperthreading and do not have superscalar capability	8	4
Each core is two-way hyperthreading and supports superscalar capability	8	8
Each core is four-way hyperthreading and supports superscalar capability	8	8

Problem 4

[10] GPU Questions:
Choose one answer only from the multiple choices for each problem. If you pick more than one answer, you will lose the grade of the corresponding problem.

1. Suppose a block is composed of 50 threads. How many warps will be created for this block?
a. 1 b. 2 c. 4 d.5 e. Depends on the total number of blocks.

2. A block size cannot be smaller than a warp size.
a. True b. False c. Depends on the compute capability of the GPU.

3. If we have two local variables in the kernel, they will surely go to registers.
a. True b. False c. Depends on the compute capability of the GPU.

4. More than one block can be assigned to the same SM at the same time.
a. True b. False c. Depends on the compute capability of the GPU.

5. If you organize the block as 3D, the performance will always be worse than 1D or 2D blocks.
a. True b. False

6. The program has full control over how many warps will be created for a specific kernel.
a. True b. False

7. Suppose we launch a kernel as follows: kernelname <<<8,8>>>(argument list …);
And suppose the kernel has a local variable x. How many instances of x will we have?
a. 8 b. 16 c. 64 d. Depends on whether the block is 1D, 2D, or 3D.

8. Suppose we launch a kernel as follows: kernelname <<<8,8>>>(argument list …);
And suppose the kernel has a shared variable x. How many instances of x will we have?
a. 8 b. 16 c. 64 d. Depends on whether the block is 1D, 2D, or 3D.

9. If we do not use if-else in our kernel, then we will surely not suffer from branch divergence.
a. True b. False c. We cannot tell.

10. We can have more than one warp executing in the same SM at the same time.
a. True b. False