CSCI-UA.0480-051: Parallel Computing
Final Exam – Fall 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, in the assignments section, at 2pm EST on Dec 21st.**
- **You have up to 24 hours from 2pm EST of Dec 21st to submit your answers 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 giving wrong answers and for putting irrelevant information in your answers.**
- **You must upload one pdf file.**
- **Your answer sheet must have a cover page (as indicated below) and one problem answer per page (that is, problem 1 on a new page, problem 2 on another new page, etc.).**
- **This exam has 4 problems totaling 100 points.**
- **The very first page of your answer is the cover page and must contain ONLY:**
  - **You Last Name**
  - **Your First Name**
  - **Your NetID**
  - **Copy and paste the honor code shown in the rectangle at the bottom of this page.**

Honor code (copy and paste the whole text shown below, in the rectangle, to the first page, cover page, of your exam)

- You may use the textbook, slides, zoom recordings, and any notes you have.
- You may NOT use communication tools to collaborate with other humans.
- 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. [6 points] Suppose we have a single core. Based on Flynn classification, in what category does it fit (SISD, SIMD, MISD, or MIMD)? Justify your answer. Make any assumptions you think are needed. Assume there are no caches at all in that system.

This single core can be in one of three designs:

- [1] Just pipelining: It will execute on instruction at a time. → [1] MIMD
- [1]Superscalar: It can execute several instructions at a time each with its distinct input → [1]MIMD
- [1] Hyperthreaded: It can execute several instructions at a time each with its distinct input →[1] MIMD

b. [6] Repeat the above problem but assume there is L1 cache of size 32KB with that core.

Exactly same answer as above. Cache has no effect on this classification.

c. [12] We know that coherence has a negative effect on performance. State three reasons as to why.

- [4] Coherence requires messages to be sent to other caches to invalidate, etc. This uses bandwidth, which is a scarce resource.
- [4] Coherence will delay a core writing to its cache until other caches invalidate. This delay affects performance.
- [4] Coherence causes caches to invalidate blocks. This increases cache misses.

Problem 2

Answer the following questions about MPI.

a. [6] Suppose we have an MPI program where we have four processes. MPI_Comm_Split() is called once. How many ranks can each process have after that call? If there are several answers, specify them all and justify your answer.

- [1] A process can have two ranks: [2] the original and the rank of the newly created communicator.
- [1] A process can have one rank [2] if that process calls the split with color: MPI_UNDEFINED, in which case it won't be added to any newly created communicator.

b. For the following piece of code (assume very large number of cores):

```
int main(){
  int globalnum = 0;
  int numprocs, rank;
  int i = 0;

  MPI_Init(NULL, NULL);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  #pragma omp parallel for reduction(+:globalnum)
  for( i = 0; i < 2+rank ; i++)
    {
        globalnum ++;
        …rest of loop body …
     }

  MPI_Finalize();
    …
}
```

We execute the above code with: **mpirun -n 5 ./progname**

1. [5] What is the maximum number of threads we will end up having in the whole system? Explain.

[2] 20 threads will be created.
[3] The number of threads depends on the loop iterations of each process.
process 0: 2 iterations
process 1: 3 iterations
process 2: 4 iterations
process 3: 5 iterations
process 4: 6 iterations

2. [5] Just before executing MPI_Finalize(), how many instances of globalnum do we have in the system? Justify.

[2] Five instances.
[3] Each process has its own copy of globalnum.

3. [5] Just after executing MPI_Finalize(), and before exiting main() how many instances of globalnum do we have in the system? Justify

Exactly same answer as above.

c. [9] Suppose we have three processes. Each process has an array p of integers of 3 elements as follows. Process 0 has [1, 2, 3], process 1 has [4, 5, 7], and process 2 has [7, 8, 9]. Suppose all the processes execute the following API:
MPI_Reduce(p, q, 3, MPI_INT, MPI_BAND, 1, MPI_COMM_WORLD);
Where p is a pointer to the array and q is a pointer to a receiving array. Assume the receiving array q is initially [0, 0, 0] for all processes. After executing the above function, what will be the content of q for each process.

Let's see p in binary for all processors:
process 0: [0001, 0010, 0011]
process 1: [0100, 0101, 0111]
process 2: [0111, 1000, 1001]

Only process 1 array q will be modified (as indicated in the reduction command) the other q arrays will be unmodified.
So, q arrays will be:
[3] process 0: [0, 0, 0] ← unchanged
[3] process 1: [0, 0, 1]
[3] process 2: [0, 0, 0] ← unchanged

Problem 3

For the following code snippet (questions a to d are independent from each other):

```
1.    #pragma omp parallel num_threads(16)
2.    {
3.        #pragma omp for schedule(static,2) nowait
4.        for(i=0; i<N; i++){
5.            a[i] = ....
6.        }
7.
8.        #pragma omp for schedule(static,2)
9.         for(i=0; i<N; i++){
10.       ... = a[i]
11.       }
12.   }
```

a. [6 points] For each one of the following situations, indicate whether there will be a possibility of race condition or not.

[1 point each entry]

Scenario	**Race Condition (Y/N)**
The code as it is above	N
We remove "nowait" from line 3	N
We keep the "nowait" in line 3 but change the schedule in line 3 to (dynamic, 2)	Y
We keep the "nowait" in line 3 but change the schedule in line 8 to (dynamic, 2)	Y
We remove the "nowait" from line 3 and put it in line 8	N
We remove the whole line 1 and use clause **"#pragma omp parallel for**" in both lines 3 and 8 and remove the "nowait" of line 3	N

b. [7 points] How many threads exist at line 7 (that empty line between the two for-loops)? Explain how you reached that number in no more than two lines.

16 threads
These are the number of threads created at line 1 and they stay alive till line 12.

c. [7 points] How many threads exist just after line 12? Explain how you reached that number in no more than two lines.

Only on thread
After line 12, we are out of the parallel region and back to sequential.

d. [5 points] Suppose the above code executes on a multicore that has 16 cores and each core is superscalar and not hyperthreading. If N = 16, how many threads can execute in parallel on this multicore, assuming no other processes are running on this multicore? Justify your answer.

[3] With N = 16 and (static, 2) the loop iterations are distributed as follows:
t0 (iterations 0, 1) t1(2, 3) t2(4, 5) t3(6, 7) t4(8, 9) t5(10, 11) t6(12, 13) t7(14, 15)

[2] This means that even though 16 threads exist, only eight threads will be executing.

Problem 4

Answer the following questions related to GPUs and CUDA.

a. [7] Suppose we have two kernels: kernelA and kernel in the same program. The program launches kernelA. When kernelA finishes, it generates data to be used by kernel. What is the best way for kernelA to send data to kernelB? Justify.

[2] kernelA can leave the results in the device's global memory.
[5] When kernelB is launched, it will find the data in the global memory. That is because the data in the global memory has a lifetime of the whole application. This is better than returning the result back to the host and then resending it to the device for kernelB.

b. [5] Can we make two kernels, from the same program, execute on the same GPU at the same time? If yes, explain how. If not, explain why not.

[2] Yes, we can.

[3] We can write a multithreaded program, in OpenMP for example, and each thread launches a kernel to the same device.

c. [9] If the warp concept did not exist, what would be the implications on GPU programming? State three implications and, for each one, explain in 1-2 lines.

- [3] Without warp each SP needs to have its own front end. With warps, every several SPs share the same front-end, but without it, this cannot be done.
- [3] The scheduling inside the SM must be modified so that it schedules each thread execution to SPs instead of scheduling warps.
- [3] Because each SP will have its own front-end, there will be less space in the SM for a lot of SPs. This means the number of SPs per SM will decrease.