CSCI-UA.0480-003

Parallel Computing

Exam 2 (Final)

Spring 2020

[Total: 100 points]

Important Please Read Very Carefully:

- You must submit a pdf file.
- Read the honor code in the assignment page of the exam at NYU classes. You will need to copy/paste it in your submission as indicated in the bullet below.
- The first page of your submission must contain:
  - Last Name:
  - First Name:
  - NetID (NOT the long university ID)
  - The honor code
  - Put the date and your name under the honor code as a substitution to a signature
- Start your answer from the second page.
- The name of the pdf file that you will submit is your netID.pdf
- You must start submission at least few minutes before the end time. The submission will be closed at 3pm EST sharp. Of course, you can submit early if you want. But be careful that re-submissions are not allowed.
- If you are confused by a problem or you think something is missing in the question, make any assumptions that make you progress in the solution. We will grade both your assumption and the solution.

1. Suppose that MPI_COMM_WORLD consists of the four processes 0,1, 2, and 3, and suppose the following code snippet is executed (my_rank contains the rank of the executing process in MPI_COMM_WORLD) [Note: The code shown here is just part of a bigger program. It is not a full-fledged program.].

```
int x, y, z;
MPI_Comm COMM;
int new_rank;

MPI_Comm_Split(MPI_COM_WORLD, my_rank%2,  my_rank,  &COMM);
MPI_Comm_rank(COMM, &new_rank);

switch(new_rank) {

case 0:
     x=11; y=12; z=10;
     MPI_Reduce(&x, &y, 1, MPI_INT, MPI_SUM, 0, COMM);
     MPI_Allreduce(&y, &z, 1, MPI_INT, MPI_SUM,  COMM);
     break;

case 1:
     x=3; y=8; z=5;
     MPI_Reduce(&x, &z, 1, MPI_INT, MPI_SUM, 0, COMM);
     MPI_Allreduce(&y, &x, 1, MPI_INT, MPI_SUM, COMM);
     break;

default:
     x=8; y=9; z=11;
     MPI_Reduce(&z, &y, 1, MPI_INT, MPI_SUM, 0, COMM);
     MPI_Allreduce(&x, &y, 1, MPI_INT, MPI_SUM, COMM);
     MPI_Bcast(&y, 1, MPI_INT, 1, COMM);
     break;
}
```

a. [5 points] Is there a possibility of deadlock in the above code? If yes, describe the scenario that leads to the deadlock. If not, prove that all collective calls called by the processes will not be blocked forever.

No, no deadlock. There are two new communicators with the same name COMM. One has the original processes 0 and 2 (they will be 0 and 1 respectively in the new communicator). The other has the original processors 1 and 3 (they will be 0 and 1 respectively in the new communicator). All the processes in each communicator will call the collective functions.

b.[12 points] For each one of the four processes (IDs 0, 1, 2, 3, and 4 in the original MPI_COMM_WORLD) what will be the values of x, y, and z after the execution of the above code.

	P0	**P1**	**P2**	**P3**
x	11	11	22	22
y	14	14	8	8
z	22	22	5	5

c. [5 points] How many processes will execute the “default” part of the switch case? What are their process IDs in the original MPI_COMM_WORLD? Justify your answer.

No process will execute it .In the two communicators called COMM, each of the two processes in each communicator will have the rank 0 and 1.

d. [4 points] Can a process have more than one ID (i.e. multiple ranks)? If yes, describe a scenario. If not, explain why not.

Yes, if it is part of several communicators.

e. [4 points] After the execution of the above code, how many communicators exist? What are they?

Three communicators: MPI_COMM_WORLD that has four processes and two COMMs each of which has two processes.

2. [5 points] Can two processes share the same cache memory? Justify your answer.

Yes, if they are assigned to the same core.

---

3. [10 points] If we run two processes on a single core, we expect that the sequential version of the program will be faster than the parallel version where two MPI processes run on that single core. Describe *two scenarios* where two MPI processes running on a single core give a *better* performance than the sequential program.

- Scenario 1: One process is I/O bound and the other is computation bound. If the I/O bound is waiting for I/O the other process is making progress. In the sequential version, if there is an I/O operation, the whole process waits.
- Scenario 2: Two computation intensive processes but they are using different type of computations (e.g. one does integer computations while the other is doing floating point). In that case, they can make use of the hyperthreading technology without having contention. On the other hand, the sequential version will be using one process of the two, or more, hyperthreading slots.

---

4. [10 points] Suppose we have two MPI processes and we run them on a processor with two cores. Describe two scenarios where we can get better performance if we run these two processes on a processor with four cores instead of two.

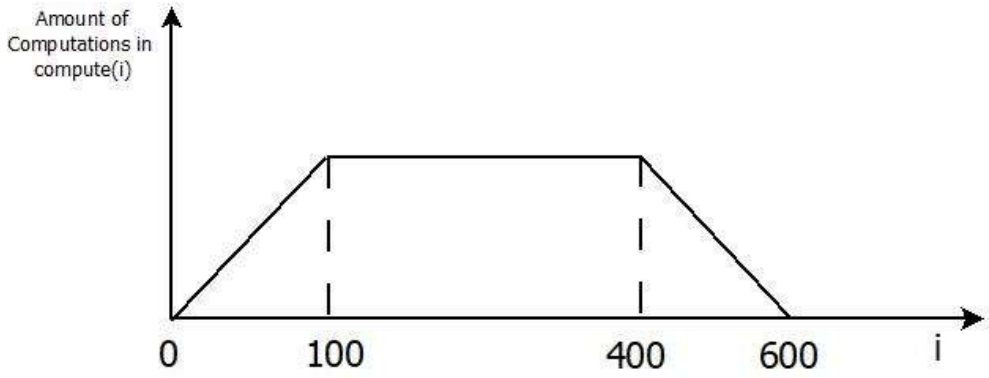
- Scenario 1: Each process is multithreaded and hence can make use of multiple cores.
- Scenario 2: If a core becomes overly hot, it will reduce its voltage and frequency to reduce heat. This makes the core run slower. With four cores, and if the OS supports that, the process can migrate to one of the empty cores and execute at full speed.

---

5. [10 points] Suppose we have the following code snippet for OpenMP

```
#pragma omp parallel for
for( i = 0; i < 600; i++)
        compute(i);
```

The following figure shows the amount of computations done in compute(i) for each value of i. The code as indicated above uses the default schedule. Modify the code to use better schedule(s) given the information about computations shown in the figure. Add 1-2 lines explaining your logic for picking the schedule(s) that you used.



There are three ranges: [0,100[then [100, 400[then [400, 600[
The middle range has same amount of computations for all i's, hence the default schedule is fine.
For the other two, they may require either static with a small chunk size or a dynamic one.

```
#pragma omp parallel for schedule (static,1)
for( i = 0; i < 100; i++)
        compute(i);
```

```
#pragma omp parallel for
for( i = 100; i < 400; i++)
        compute(i);
```

```
#pragma omp parallel for schedule(static, 1)
for( i = 400; i < 600; i++)
        compute(i);
```

Note: A different small chunk size is considered correct too.

---

6. [5 points] Assume we are multiplying an 8000x8000 matrix A with vector y. Each element of the matrix and vector is double float. We are parallelizing this multiplication operation using four threads in OpenMP, such that thread 0 will be responsible for the first 2000 elements of the solution vector, thread 1 for the following 2000 elements, and so on. Also assume that each thread will execute on a separate core. Each core has its own private L1 cache. If the cache line has a size 64 bytes. Is it possible for false sharing to occur at any time between threads 0 and 2? Explain

* double float → 8 bytes
* 1 cache block = 64 → can hold 8 elements of the array
* Thread 0 will be responsible for elements from 0 to 1999.
* If we assume that the last element with thread 0 will be at the beginning of a cache block, then the rest of the cache block will be the seven elements 2000 to 2006 because cache block brings contiguous memory bytes and the array is stored contiguously in memory.
* The first element for thread 2 is element 4000 → can never have false sharing with thread 0.

7. [5 points] We have seen that threads in a warp in CUDA execute instructions in a lockstep. Despite that, there may be scenarios where threads belonging to the same warp can finish before other threads belonging to that same warp. Describe one such scenarios.

- One possible scenario: if-else in kernel and there are no code after the if-else, so the threads which finished executing the if-part will be done while the others are still executing the else-part.
- Another scenario: An if without else. Those threads with false condition will be done while the rest will be executing the if part.

---

8. For the following vector multiplication kernel and the corresponding kernel launch code, answer each of the questions below. Assume Ad, Bd, and Cd have been declared earlier. For each question below, show the steps you used to reach your answer.

```
1  __global__ void vecMultKernel (float* Ad, float* Bd, float* Cd, int n)
2  {
3        int i = threadIdx.x + (blockDim.x * blockIdx.x * 2);
4
5        if (i < n) { Cd[i] = Ad[i] * Bd[i]; }
6                i += blockDim.x;
7        if (i < n) { Cd[i] = Ad[i] * Bd[i]; }
8  }
9
10 int vectMult (float* A, float* B, float* C, int n)
11 {
12       /*  n  is the length of arrays A, B, and C.
13       int size = n * sizeof (float);
14       cudaMalloc ((void **)&Ad, size);
15       cudaMalloc ((void **)&Bd, size);
16       cudaMalloc ((void **)&Cd, size);
17       cudaMemcpy (Ad, A, size, cudaMemcpyHostToDevice);
18       cudaMemcpy (Bd, B, size, cudaMemcpyHostToDevice);
19
20       vecMultKernel<<<ceil (n / 2048), 1024>>> (Ad, Bd, Cd, n);

21       cudaMemcpy (C, Cd, size, cudaMemcpyDeviceToHost);
22 }
```

a. [3 points] If the number of elements n of the A, B, and C arrays is 10,000 elements each, how many warps are there in each block? Show your calculations to get full credit.

Each block has 1024 threads (line 20) → 1024/32 = 32 warps. 1024 is divisible by 32.

b. [3 points] If the number of elements n of the A, B, and C arrays becomes 100,000 elements each, how many warps are there in each block? Show your calculations to get full credit.

Same answer as above because, based on line 20, the larger problem size increases the number of blocks not the size of an individual block.

c. [3 points] What is the CGMA of line 5? Show how you calculate it. You can disregard the computations involved in condition evaluation "if (i < n)".

Three global memory access to Ad, Bd, and Cd and only one computation (multiplication). So CGMA = 1/3

d. [4 points] Explain the effect of using shared memory in the above code to reduce global memory access.

No effect whatsoever because threads are not sharing any data actually.

e. [4 points] If n = 2048, how many warps will suffer from branch divergence in lines 5 and 7? Explain.

n = 2048 → We need only one block → Each thread responsible for two elements from Cd[0] to Cd[2048] → No branch divergence because there are no elements outside the range.

f. [3 points] If another kernel is launched in that same program. Will that kernel be able to access the data in Cd? Justify your answer.

Yes, the data in the global memory:

- **can be viewed by any kernel in the process**
- **has a lifetime of the whole application.**

---

9. [5 points] We know that CUDA does not allow synchronization among threads in different blocks. Suppose CUDA allows this. State one potential problem that may arise and explain clearly.

If there is a synchronization among all blocks of a kernel, a barrier for example, then all blocks must reach the barrier before they can proceed. In many cases we have more blocks than can be executed simultaneously by SMs → several blocks will be waiting till some blocks currently being executed finish and release the resources → Those executing blocks will get stuck at the barrier and those waiting for execution will never reach the barrier because they cannot execute → deadlock.