COMS BC 3159 - S23: Problem Set 1

Introduction: The following exercises will explore computer architecture / systems and parallel programming on the CPU and GPU. In addition to solving the problems found below, you will also need to complete the coding part of the assignment, found in the Github repo.

Finally, we'd like to remind you that all work should be yours and yours alone. This being said, in addition to being able to ask questions at office hours, you are allowed to discuss questions with fellow classmates, provided 1) you note the people with whom you collaborated, and 2) you **DO NOT** copy any answers. Please write up the solutions to all problems independently.

Without further ado, let's jump right in!

Collaborators:

Am I Thread Safe?

Problem 1 (4 Points):

Below you will find some partially completed code for a multi-threaded C++ program that is intended to calculate the average value of an array of integers.

(a) What code needs to be put in the two #TODO# blocks in lines 20 and 21 in order to efficiently divide memory across the threads?

(b) Is this code thread safe? That is, will it produce the correct output? Why or how can you fix it?

```
1  #include <iostream>
2  #include <thread>
3  #include <vector>
4
5  const int N = 100000;
6  const int N_threads = 10;
7  int data[N];
8  int sum = 0;
9  std::thread threads[N_threads];
10
11 void sum_part(int *array, int start, int end) {
12     for (int i = start; i < end; i++) {sum += array[i];}
13 }
14
15 int main() {
16     // Initialize array with random values
17     for (int i=0; i<N; i++) {array[i] = rand();}
18
19     for (int i=0; i<N_threads; i++) {
20         int start = #TODO#
21         int end = #TODO#
22         // launch the sum_part function in a thread
23         threads[i] = std::thread(sum_part,
24                                  data, start, end);
25     }
26
27     // join all of the threads
28     for (int i=0; i<N_threads; i++) {threads[i].join();}
29
30     // output the average
31     std::cout << "Avg: " << (double)sum/N << std::endl;
32     return 0;
33 }
```

Solution 1:

(a)

(b)

Problem 2 (6 Points + 1 Bonus):

Below you will find some partially completed code for a GPU program that is intended to sum all of the numbers from 1 to N. However, there are a few mistakes with the code that will make it not run correctly or produce the correct output. Please list all the changes that need to be made to fix this code.

Hint: You should find at least 3 major issues for full credit. One bonus point if you find all of the major issues.

```
1  #include <iostream>
2  #include <cuda_runtime.h>
3
4  const int N = 100;
5
6  // add 1 to all values in an array
7  __global__ void increment_kernel(int *data, int N) {
8      int idx = threadIdx.x;
9      if (idx < N) {data[idx] += 1;}
10 }
11
12 int main() {
13     int *h_data = (int *)malloc(N*sizeof(int));
14     int *d_data; cudaMalloc(&d_data, N * sizeof(int));
15
16     // initialize the data
17     for (int i =0; i < N; i++){h_data[i] = i;}
18
19     // apply the kernel
20     increment_kernel<<<10, 10>>>(d_data, N);
21     cudaDeviceSynchronize();
22
23     // sum the resulting array
24     int sum = 0;
25     for (int i = 0; i < N; i++) {
26         sum += d_data[i];
27     }
28     std::cout << "Sum: " << sum << std::endl;
29
30     free(h_data); cudaFree(d_data);
31     return 0;
32 }
```

Solution 2:

Computer Architecture and Systems

Problem 3 (6 Points):

Assume that when you access memory from RAM that you load in 64 bytes of memory into the cache. Also assume that loading from RAM to cache and then into registers takes a total of 26ns. Future loads of this data from the cache into registers takes only 2ns. Finally assume that the result of all computations loads data back into cache and registers for free (aka it takes 0ns). For the following code below how long would it take to load data into registers during the runtime of the function? Please write your answer as a function of N.

(a) First, consider the case where this program is run in one thread.

(b) Second, consider the case where the for loop in lines 3-5 is parallelized across 4 threads evenly distributing the work. Ignore thread or synchronizations overheads. Can you describe the least efficient memory access pattern for the four threads? How long would it take? Is this faster or slower than the serial case?

(c) Conversely in the 4 thread case, can you describe the most efficient memory access pattern? How long would it take? Is this faster or slower than the serial case?

Hint: integers are 4 bytes in size!

```
1 void sum_even_vector(int *array) {
2     int sum = 0;
3     for (int i = 0; i < N; i += 2) {
4         sum += array[i];
5     }
6     return sum;
7 }
```

Solution 3:

1.

2.

3.

Problem 4 (4 Points)

Are the following statements true or false? Please explain why in 1-3 sentences.

1. A sorting algorithm which divides an array into four parts and has each part sorted on a different CPU thread using a different sorting algorithm (quick sort, merge sort, bubble sort, etc.) is an example of a MIMD computational pattern.
2. An algorithm that compute the sum of two arrays using a serial loop on a single thread is an example of a SIMD computational pattern.
3. All threads launched in a computational grid can access the same shared memory.
4. You cannot fully control which core a thread runs on in a CPU, nor can you fully control which SM a thread block runs on in a GPU.

Solution 4:

1.
2.
3.
4.