COMS BC 3159 - S23: Problem Set 1

Introduction: The following exercises will explore computer architecture / systems and GPU parallel programming. 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:
AI Tool Disclosure:

Computer Architecture and Systems

Problem 1 (6 Points):

For this problem we will be considering the following function which sums every other value in an array:

```
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 }
```

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 all computations and loads of data back into cache and registers are free (aka it takes 0ns). For the following code above 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 launch 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!

Solution 1:

(a)

(b)

(c)

Problem 2 (4 Points)

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

(a) A sorting algorithm which divides an array into four parts and has each part sorted on a different physical computer using a different sorting algorithm (quick sort, merge sort, bubble sort, etc.) is an example of a MIMD computational pattern.

(b) 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.

(c) All threads launched in a computational grid (aka all of the threads across all of the blocks launched as a part of a single GPU kernel) can access the same shared memory.

(d) You cannot control which SM a thread block runs on in a GPU.

Solution 2:

(a)

(b)

(c)

(d)

CUDA Programming

Problem 3 (6 Points + 1 Bonus):

Below you will find a short completed CUDA 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 four of the major issues (the last one is a bit more subtle).

```
#include <iostream>
#include <cuda_runtime.h>

const int N = 100;

// add 1 to all values in an array
__global__
void increment_kernel(int *d_data, int N) {
    int idx = threadIdx.x;
    if (idx < N) {d_data[idx] += 1;}
}

__host__
int main() {
    int *h_data = (int *)malloc(N*sizeof(int));
    int *d_data; cudaMalloc(&d_data, N * sizeof(int));

    // initialize the data
    for (int i =0; i < N; i++){h_data[i] = i;}

    // apply the kernel
    increment_kernel<<<10, 10>>>(d_data, N);
    cudaDeviceSynchronize();

    // sum the resulting array
    int sum = 0;
    for (int i = 0; i < N; i++) {
        sum += d_data[i];
    }
    std::cout << "Sum: " << sum << std::endl;

    free(h_data); cudaFree(d_data);
    return 0;
}
```

Solution 3:

Problem 4 (4 Points):

Below is another short CUDA program that is intended to efficiently compute the sum of an array.

(a) If you made no changes to this code, not only would it not produce the correct answer, but it would never finish running. Why?

(b) This code has a number of missing or misplaced calls to synchronization primitives. Please describe where and how they should be added / changed / moved. *Hint: You should be able to find at least 2 places to add / change / move them.*

```
#include <stdio.h>
#include <cuda_runtime.h>

__global__
void reduce(int *d_input, int *d_output, int n) {
    // define our shared (cache) memory
    extern __shared__ int s_data[];

    // Load data into shared memory from RAM
    for (int i = threadIdx.y; i < n; i++){
        s_data[i] = d_input[i];
    }

    // Reduction in shared memory
    while(n > 1){
        for(int i=threadIdx.x; i<n/2; i+=blockDim.x){
            s_data[i] += s_data[i + n/2];
            __syncthreads();
        }
        // account for odd length arrays
        if (n%2 && threadIdx.x==0){
            s_data[0] += s_data[n-1];
        }
        // update tree layer
        n = n/2;
    }
    // save back to RAM from shared
    if (threadIdx.x==0) {
        d_output[0] = s_data[0];
    }
}
```

```
__host__
int main() {
    const int n = 64;

    // initialize the values to 1
    int h_vals[n];
    for(int i = 0; i < n; i++){h_vals[i] = 1;}
    // copy onto the GPU
    int nInt = n*sizeof(int);
    int *d_vals; cudaMalloc((void**)&d_vals,nInt);
    cudaMemcpy(d_vals,h_vals,nInt,
        cudaMemcpyHostToDevice);
    // allocate memory for the sum
    int h_sum; int *d_sum;
    cudaMalloc((void**)&d_sum, sizeof(int));

    // run the kernel
    int nThreads = 256;
    reduce<<<1,nThreads,nThreads*sizeof(int)>>>(
        d_vals,d_sum,n);
    cudaDeviceSynchronize();

    // copy the result back and display
    cudaMemcpy(&h_sum,d_sum,sizeof(int),
        cudaMemcpyDeviceToHost);
    printf("%d\n",h_sum);

    // free vars and exit
    cudaFree(d_vals); cudaFree(d_sum); return 0;
}
```

Solution 4:

(a)

(b)