15-418/618 Spring 2021
Exercise 3

Assigned:	Wed., Feb. 24
Due:	Wed., Mar. 3, 11:00 pm

Overview

This exercise is designed to help you better understand the lecture material and be prepared for the style of questions you will get on the exams. The questions are designed to have simple answers. Any explanation you provide can be brief—at most 3 sentences. You should work on this on your own, since that's how things will be when you take an exam.

You will submit an electronic version of this assignment to Canvas as a PDF file. For those of you familiar with the LaTeX text formatter, you can download the template and configuration files at:

http://www.cs.cmu.edu/~418/exercises/config-ex3.tex
http://www.cs.cmu.edu/~418/exercises/ex3.tex

Instructions for how to use this template are included as comments in the file. Otherwise, you can use this PDF document as your starting point. You can either: 1) electronically modify the PDF, or 2) print it out, write your answers by hand, and scan it. In any case, we expect your solution to follow the formatting of this document.

Problem 1: GPU Programming

Your friends have been hearing that you now know how to program a GPU and take advantage of all its capabilities. Eager to bask in your knowledge, they come to you with different problems that they think might benefit from GPU parallelism.

One such friend heard that GPUs can launch thousands of parallel "threads." They want to identify the indices of a large array of n elements in an even larger array (of size m) using binary search. They suggest to you that every thread on the GPU will be searching for one element in the array, so that with say, 1024 threads, they'll be able to search get near $1024\times$ performance.

A. First off, how do you explain that threads on a CPU are not the same as threads on a GPU? (What's the difference between the implementation of p_threads and CUDA threads?) Give at least 3 important distinctions.

- CPU threads are not actual hardware implementations they are programming abstractions.
- GPU threads are more light-weight compared to CPU threads because of the hardware implementation. Thus switching between threads are more costly on CPU.
- GPU threads run as groups(warp) while CPU threads run independently.

B. Still convinced of their binary search's potential, they write this kernel function.

```
// Array S and R are of size n
// Array A is of size m, with its elements ordered.
__global__ void cudaBinarySearch(int n, int m, int *S, int *A, int *R) {
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i >= n) return;

    int search = S[i];
    int left = 0, right = m - 1, middle = 0;
    R[i] = -1;  // In case not found
    // Binary Search
    while (left <= right) {
      middle = left + (right - left) / 2;
      if (search < A[middle])
        right = middle - 1;
      else if (search > A[middle])
        left = middle + 1;
      else
        R[i] = middle;
        break;
    }
}
```

After testing it out, they find that they get very little performance gain relative to the number of threads that they are running. Sullenly, they come back to you and ask where they went wrong. Give at least two reasons for this code's poor performance.

- The code is not parallelized. The threads are looking for value in serialized way inside while loop.
- The code is not optimized for GPU. The code is not using shared memory for arrays S and R.

Problem 2: Task Assignment for OCEAN

Recall that we discussed a version of the OCEAN application that uses a static assignment strategy where it assigns blocks of contiguous rows to threads. In this problem, we will ask you to compare that static approach with another task assignment approach for OCEAN (one that is dynamic). Similar to exercise 2, as you discuss the likely differences in performance between these approaches, please relate your answers to the **three goals** for task assignment (i.e. **balance the workload, maximize locality, and minimize extra work**) as well as discussing the likely impact on the components of **per-thread execution profiles** (like the ones seen on slides 12 and 22 in Lecture 9).

A. **Static versus Coarse-Grained Dynamic Assignment for OCEAN**
Consider a different hypothetical implementation of OCEAN that uses distributed task queues to dynamically assign individual rows to threads at runtime. Each task in the task queues would represent an entire row of the matrix. Assume that equal numbers of tasks would be assigned to each task queue at the start of a time step (with the same initial partitioning as the static assignment: i.e. contiguous blocks of rows), and that task stealing would be used whenever a processor runs out of work. Using the metrics above, provide a qualitative discussion of the likely performance differences between these two versions of OCEAN (i.e. static versus coarse-grained dynamic).

Wokload balance would be better due to dynamic runtime assignment.
However, locality would be worse due to the fact that the threads are not working on contiguous rows.
Extra work would be more due to the fact that the threads need to steal work from queues.