CS121 Problem Set 4

Due: 23:59, April 27, 2021

1. Submit your solutions to Gradescope (www.gradescope.com).
2. In "Account Settings" in Gradescope, set FULL NAME to your Chinese name and enter your STUDENT ID.
3. If you submit handwritten solutions, write neatly and submit a clear scan.
4. When submitting your homework, be sure to match each of your solutions to the corresponding problem number.

1) We discussed barrier synchronization as a way to ensure all threads reach a a point in code before any of them progress beyond the point. The figure below shows a barrier for N threads implemented using locks. Explain clearly how this code works. What should the initial values of `count` and the `arrival` and `departure` locks be? Why is it necessary to use two lock variables?

```
void barrier()
{
    set_lock(arrival);
    count++;
    if (count < N)
        unset_lock(arrival);
    else
        unset_lock(departure);
    set_lock(departure);
    count--;
    if (count > 0)
        unset_lock(departure);
    else
        unset_lock(arrival);
}
```

2) Using one lock and one condition variable, develop an alternative barrier implementation and write it using threads pseudocode. Compare your solution with that in the figure above. What should the initial values be? Why is only one lock needed?

3) GPU computing has been used to speed up many real-world applications. However, not all applications are suitable for GPU acceleration. Consider the following operations / applications and decide whether each is suitable for GPU acceleration or not. Briefly justify your answer. Assume all the input data are initially in the main memory.

a. Matrix multiplications on two matrices A and B, each of size 32000 by 32000.

b. Matrix multiplications on two matrices A and B, each of size 32 by 32.

c. Binary search on a sorted array with 1 billion elements.
d. Binary search on a sorted array with 1 thousand elements.

4) Compared to CPU threads, GPU threads are designed to be "light-weight". Describe some potential disadvantages associated with increasing the amount of work done in each CUDA thread (e.g. using loop unrolling) and thereby decreasing the total number of threads used.