15-418/618 Spring 2021
Exercise 6

Assigned:	Fri., Mar. 26
Due:	Fri., Apr. 2, 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 configuation files at:

http://www.cs.cmu.edu/~418/exercises/ex6.tex
http://www.cs.cmu.edu/~418/exercises/config-ex6.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: Synchronization

You are tasked with creating a program that computes the sum of a randomly selected subset of the elements of a large array using multiple threads. The basic strategy is to use the following:

```
// global variables shared by all threads

int values[N];  // assume N is very large
int sum = 0;

////////////////////////////////////////////////////////
// per thread logic (assume thread_id, num_threads are defined as expected)

for (int i=thread_id; i<N; i+=num_threads) {
  if (random() & 0x1 == 1)
    sum += values[i];
}
```

A. You find the documentation for the GCC atomic builtins and decide to implement the addition using `__sync_fetch_and_add`. Describe how you would modify the above code to do this.

```
__sync_fetch_and_add(&sum, values[i]);
```

I would change the line sum+=values[i] to the above line. This will ensure that the sum is updated atomically.

B. Your boss comes to you and says “Great job, but we also need to keep track of how many times variable `sum` is updated, so that we can compute the average. Fix the following code:”

```
// global variables shared by all threads

int values[N];  // assume N is very large
int count = 0;
int sum = 0;

////////////////////////////////////////////////////////
// per thread logic (assume thread_id, num_threads are defined as expected)

for (int i=thread_id; i<N; i+=num_threads) {
  if (random() & 0x1 == 1) {
    sum += values[i];
    count++;
  }
}
```

Using *only one* fetch-and-add in your thread routine, and with no other synchronization primitives, show how you could do this. You may introduce additional global and local variables, and other computations.

```
// global variables shared by all threads

int values[N];  // assume N is very large
int count = 0;
int sum = 0;
int_prev_sum = 0;


////////////////////////////////////////////////////////
// per thread logic (assume thread_id, num_threads are defined as expected)
for (int i=thread_id; i<N; i+=num_threads) {


  if (random() & 0x1 == 1) {

    prev_sum = __sync_fetch_and_add(&sum, values[i]);
    if((prev_sum != sum  && values[i]!=0 )|| (prev_sum == sum && values[i]==0)){
      count++;
    }
    // this check ensures previous block is executed and since there are
    no other operations between two operations,
    count  value will be updated atomically as well.
  }


}
```

Problem 2: Interconnection Network

Recall that *bisection bandwidth* is a metric for interconnect performance where you perform an imaginary representative cut through the middle of the interconnect (dividing the set of processors in half) and then add up the number of wires that cross over that cut. Roughly speaking, bisection bandwidth helps to characterize the potential communication bandwidth when half of the machine is attempting to communicate with the other half of the machine. In this problem, we would like for you to characterize how the bisection bandwidth is expected to scale with the number of processors (P) for each of the following types of interconnects. Please be sure to **explain your answers** (we want more than just a formula).

A. How does bisection bandwidth scale with P for a **bus**?

All answers answered as follows. Think the network visually, cut it half, and count the number of wires that cross the cut.
Same because the bus is a single wire and bandwith doesn't increase with number of processors.

B. How does bisection bandwidth scale with P for a **crossbar**?

$\sqrt{n}/4$ because the crossbar is a 2D array of wires. When cutting in half the formula becomes $(n/2) * (n/2)$.

C. How does bisection bandwidth scale with P for a **hypercube**?

Hypercube has $log(P)$ dimensions.Bisection bandwith formula becomes $P/2$.

D. **THIS QUESTION IS UNLRELATED TO THE PREVIOUS ONES**. Consider a parallel version of the 2D grid solver problem from class. The implementation uses a 2D tiled assignment of grid cells to processors. (Recall the grid solver updates all the red cells of the grid based on the value of each cellâ€™s neighbors, then all the black cells). Since the grid solver requires communication between processors, you choose to buy a computer with a crossbar interconnect. Your friend observes your purchase and suggests there there is another network topology that would have provided the same performance at a lower cost. What is it? (Why is the performance the same?)

Mesh network. Cost is lower than the crossbar $O(N^2)$ vs $O(N)$. Latency is same because of application spesific layout. Grid solver only requires communication between neighbors.