Full Name:

Andrew Id:

15-418/618 Spring 2019
Exercise 4

Assigned:	Mon., March 18
Due:	Fri., March 22, 11:59 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 Gradescope as a PDF file. For those of you familiar with the LaTeX text formatter, you can download the template, configuation, and figure files at:

http://www.cs.cmu.edu/~418/exercises/config-ex4.tex
http://www.cs.cmu.edu/~418/exercises/ex4.tex
http://www.cs.cmu.edu/~418/exercises/figs/mellanox.pdf
http://www.cs.cmu.edu/~418/exercises/figs/mellanox-big.pdf

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: Memory Consistency

Assume the following program segments are executed on three processors of a multiprocessor machine. Initially before execution, all variables are equal to 0.

P1	P2	P3
`E1a: A = 1`	`E2a: u = A`	`E3a: v = B`
	`E2b: B = 1`	`E3b: w = A`

A. There are totally 8 possible final states of `u`, `v`, `w` as listed. For each one, indicate whether it is valid or invalid under a sequential consistency model.

Case	Final states	Valid? (Y/N)
Case 1:	`u` = 0 `v` = 0 `w` = 0	
Case 2:	`u` = 0 `v` = 0 `w` = 1	
Case 3:	`u` = 0 `v` = 1 `w` = 0	
Case 4:	`u` = 0 `v` = 1 `w` = 1	
Case 5:	`u` = 1 `v` = 0 `w` = 0	
Case 6:	`u` = 1 `v` = 0 `w` = 1	
Case 7:	`u` = 1 `v` = 1 `w` = 0	
Case 8:	`u` = 1 `v` = 1 `w` = 1	

B. Choose one of the final states that you think is invalid under sequential consistency. Prove that it is invalid. (Refer to the operations by the event labels `E1a`, `E2b`, etc.)

C. To maintain the same possible final states (as in sequential consistency) under weaker memory consistency model, one way is to add fences that guarantee that all reads/writes prior to the fence complete before any read/write after the fence. There are 8 possible placements to add fences:

P1	P2	P3
$Fence_1$	$Fence_2$	$Fence_3$
`E1a: A = 1`	`E2a: u = A`	`E3a: v = B`
$Fence_4$	$Fence_5$	$Fence_6$
	`E2b: B = 1`	`E3b: w = A`
	$Fence_7$	$Fence_8$

To minimize the number of necessary fences (while maintaining sequential consistency), where would you place the fences? List a minimal set of fences, and argue that removing any one of these could lead to a consistency violation.

D. Suppose you could use also use a storage fence or a load fence. Could you replace any of the full memory fences in your list with one of these?

Problem 2: Interconnection Networks

This problem concerns *fat-tree networks*, as discussed in the lecture on interconnection networks:

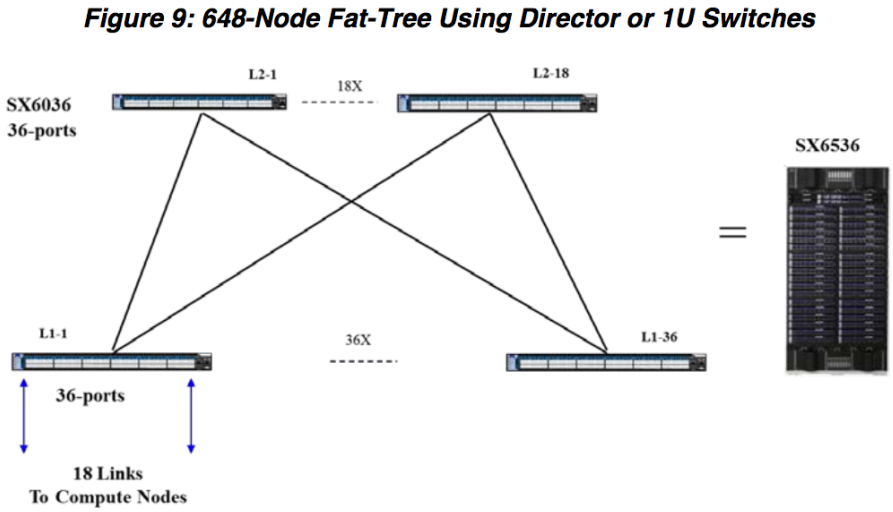
http://www.cs.cmu.edu/~418/lectures/15_interconnects.pdf.

These networks were originally proposed in 1985 by Charles Leiserson, a CMU PhD alumnus. These are a form of *indirect network*, meaning that the network is constructed separately from the computing elements.

The illustrations shown in Figure 1 are taken from the document "Deploying an HPC Cluster with Mellanox InfiniBand Solutions", published December 5, 2018 by Mellanox Technologies, a leading manufacturer of high-performance interconnection networks. In this document, they describe how to take their switches, each having 36 ports, to create various fat-tree network configurations.

We will characterize these networks by two parameters: the *switch connectivity* k, where each switch has k ports (i.e., $k = 36$ for the Mellanox switch), and the number of *levels*, l. We will use the notation $N(k, l)$ to denote a *maximal* network with switch connectivity k and l levels. By maximal, we mean that it is the largest network that can be constructed with those switches and that many levels. We use the notation $P(k, l)$ to indicate the number of *external ports* provided by a network of type $N(k, l)$. These ports can either be connected to computing elements or to other switches in order to form larger networks.

(a) Two-level, 648-node network



(b) Three-level, 1944-node network

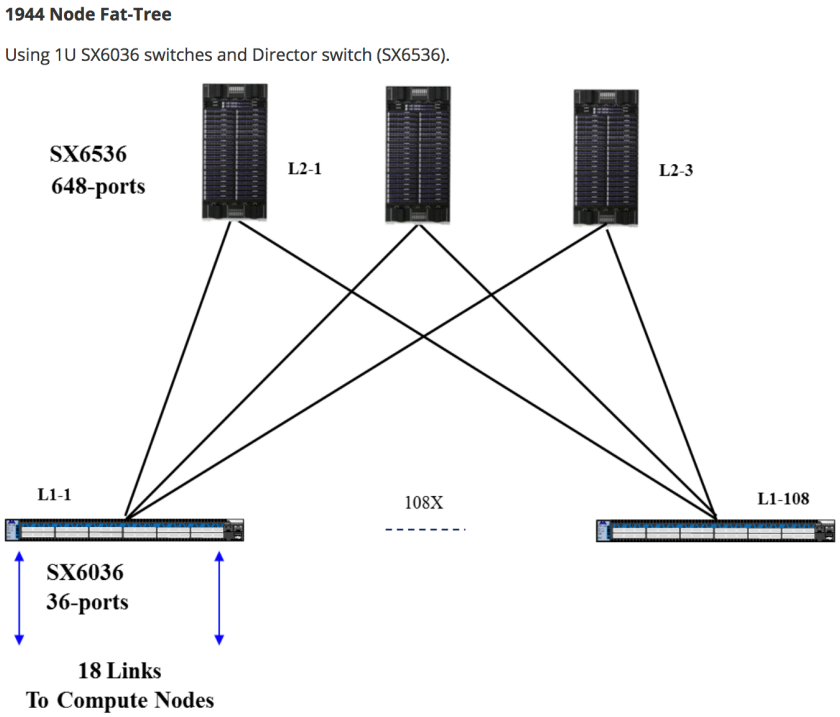


Figure 1: Mellanox Fat-Tree Networks

As the figure suggests (note that (a) forms a building block of (b)), these networks can be defined recursively:

- Network $N(k, 1)$ consists of a single switch, with all k ports being external.
- Network $N(k, l)$ is constructed by assembling $k/2$ *subnetworks*, each of type $N(k, l-1)$ along the top. Along the bottom, the network has an additional $P(k, l-1)$ switches. For each switch along the bottom, $k/2$ of its ports connect to the external ports of the subnetworks (one port per subnetwork), while the other $k/2$ ports serve as external ports for the new network.

You can see that Figure 1(b) follows the general scheme for constructing a network with $l = 3$ levels, but it is *not* maximal. Instead, it uses only $18/6 = 3$ level-2 subnetworks along the top, and $P(18, 2)/6 = 108$ switches along the bottom. Each switch along the bottom has six connections to each subnetwork along the top. By varying the number of switches along the top to any number that evenly divides $k/2$, they can provide a number of different configurations.

A. Slide 35 of the lecture shows a network of type $N(4, 3)$, and slide 36 shows one of type $N(6, 3)$. However, it's a bit difficult to see the recursive structure in these illustrations. Identify the $k/2$ subnetworks of type $N(k, 2)$ for $k \in \{4, 6\}$ in these two illustrations.

B. Give a formula for $P(k, l)$.

C. What is $P(36, 3)$?

D. Summit, the world's fastest computer, contains 4608 nodes connected by a Mellanox switch network. Although the exact details of this network are hard to come by, this article states that the system has 256 racks, each containing 18 compute nodes plus a Mellanox switch. Those switches would then form the bottom layer of the network. It also states that the network has three levels and is nonblocking.

(1) Assuming the upper two levels of the network are also based on 36-port switches, conjecture how these might be designed. (**Hint:** perhaps some of the switch ports don't get used.)

(2) What is the total number of switches in your proposed network?

Problem 3: 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.

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;



////////////////////////////////////////////////////////
// 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) {




  }


}
```