15-418/618, Spring 2018
Exercise 1

Assigned:	Mon., Mar. 19
Due:	Fri., Mar. 23, 1:30 pm

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 a template file at:

http://www.cs.cmu.edu/∼418/exercises/ex1-answer.tex

Instructions for how to use this template are included as comments in the file. Otherwise, you can use the solution template:

http://www.cs.cmu.edu/∼418/exercises/ex1-answer.pdf

You can either: 1) electronically modify the PDF file, 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 the solution template.

Problem 1: Problem Scaling

This problem is a three-dimensional extension of the grid problem described in the lecture on workload-driven performance evaluation:

http://www.cs.cmu.edu/∼418/lectures/09_perfeval.pdf.

Consider an iterative solver that operates on a three-dimensional grid of size $N \times N \times N$. It requires N iterations to reach convergence. We refer to the parameter N as the *problem size*. Increasing N by a factor of 2 increases the number of grid elements $8\times$ and the number of iterations $2\times$.

The state of the system is represented as a three-dimensional array of values `state`. The function `update` computes a new value of the state based on the current state. For each element `state[x][y][z]`, `update` computes the new value as a function of its current value and that of its six neighbors:

Self: `state[x][y][z]`

West: `state[x-1][y][z]`

East: `state[x+1][y][z]`

North: `state[x][y-1][z]`

South: `state[x][y+1][z]`

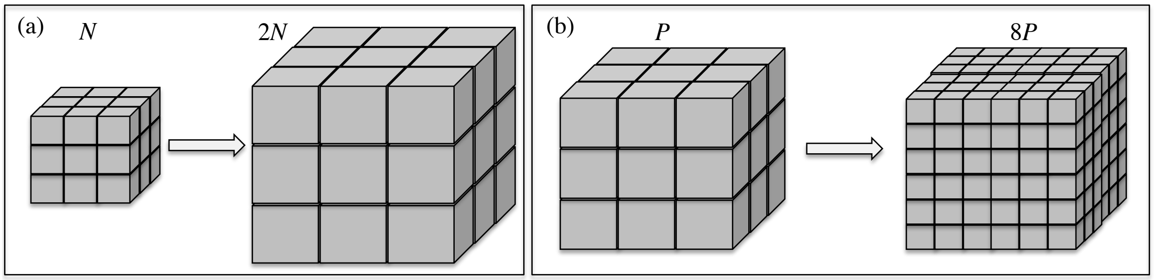


Figure 1: Scaling a problem by increasing doubling the grid size N (a), or by increasing the number of processors 8× (b).

Down: `state[x][y][z-1]`

Up: `state[x][y][z+1]`

(You don't need to know the exact function.)

The overall operation can then be described in a pseudo-C notation as:

```
// For each iteration
for (iter = 0; iter < N; iter++) {
   // Main computation
   for all x, y, z in 0..N
       nstate[x][y][z] = update(state, x, y, z);
   for all x, y, z in 0..N
       state[x][y][z] = nstate[x][y][z]
}
```

In mapping the computation onto P processors, the state array is split into P cubic blocks, each containing N^3/P array elements. Each step of the loop labeled "Main computation" involves having each processor 1) communicate the boundary values with its (up to six) neighbors, and 2) computing the update function for all N^3/P of its assigned elements. The program uses M gigabytes of memory per processor and runs in total time T.

1. Consider the following two scaling possibilities, yielding a problem of size N' running on P' processors, as illustrated in Figure 1.

   (a) Double the value of N, while holding P constant: $N' = 2N$, $P' = P$

   (b) Hold N constant, while increasing P by 8×: $N' = N$, $P' = 8P$

   Fill in the table below showing how the amounts of computation and communication would scale *per processor* and *per iteration*. That is, how much would the required computation and communication scale for each processor when performing the loop labeled "Main computation"? (For example, 2× would indicate growth by a factor of two.)

	Computation	Communication
(a): $N' = 2N$, $P' = P$		
(b): $N' = N$, $P' = 8P$		

2. Based on these specific cases, give formulas for how the computation, communication, and arithmetic intensity would scale (per processor and per iteration) as functions of N and P. (By way of reference, the formulas for the two-dimensional version were given in slide 7 of the lecture.)

Computation	Communication	Arithmetic Intensity

3. Suppose we have a machine with $8P$ processors, each identical to the original P processors. We consider three scaling possibilities, yielding a new problem of size N' requiring total time T' and M' gigabytes per processor:

   Problem scaling: $N' = N$. Solve the same problem faster. Goal is to minimize T'

   Memory scaling: $M' = M$. Make full use of the increased total memory. Goal is to maximize N'.

   Time scaling: $T' = T$. Solve a bigger problem in the same amount of time. Goal is to maximize N'.

   Fill in the following table with formulas indicating the problem size N', the per-processor memory requirement M', the ideal total time T', and the change in arithmetic intensity. (By way of reference, this information for the two-dimensional version was given in the slide 24 of the lecture.)

Scaling Type	N'	M'	T'	Arith. Intensity
Problem	N			
Memory		M		
Time			T	

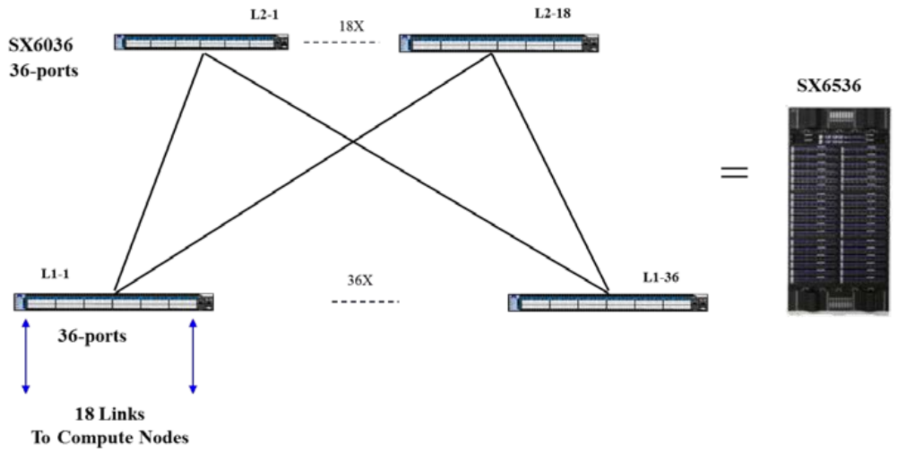
Problem 2: Interconnection Networks

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

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

(a) Two-level, 648-node network

Figure 9: 648-Node Fat-Tree Using Director or 1U Switches



(b) Three-level, 3888-node network

Figure 12: 3888-Node Fat-Tree Using Director and 1U Switches

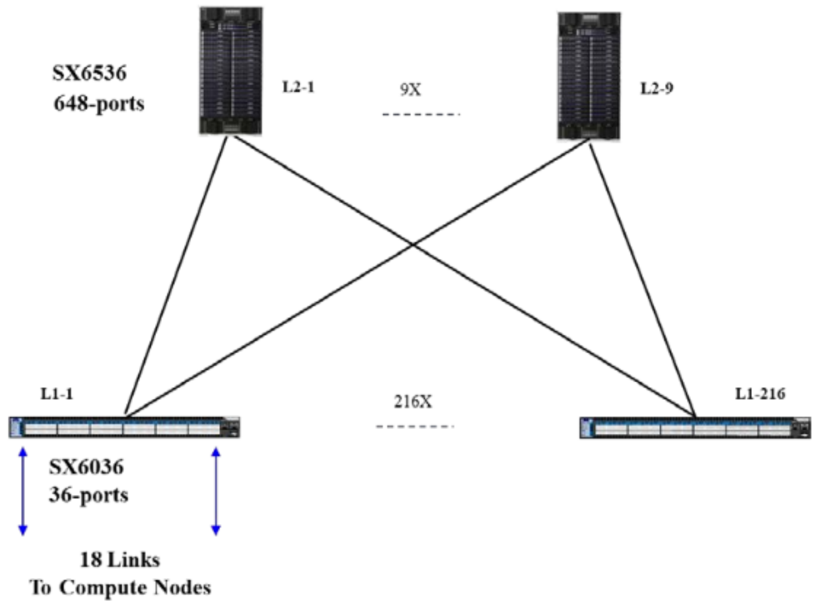


Figure 2: Mellanox Fat-Tree Networks

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 2 are taken from the document "Deploying HPC Cluster with Mellanox InfiniBand Interconnect Solutions, Reference Design," Rev 1.3, published in January, 2017 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 $2k$ ports (i.e., $k = 18$ 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 $n(k, l)$ 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.

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

You can see that Figure 2(b) follows the general scheme for constructing a network with $l = 3$ levels, but it is *not* maximal. Instead, it uses only $18/3 = 6$ level-2 subnetworks (the figure incorrectly indicates 9) along the top, and $n(18, 2)/3 = 216$ switches along the bottom. Each switch along the bottom has three connections to each subnetwork along the top. By varying the number of switches along the top to any number that evenly divides k, they can provide a number of different configurations.

1. Give a formula for $n(k, l)$.
2. What is $n(18, 3)$?
3. Give a formula for the number of switches required to construct network $N(k, l)$.
4. How many switches are in network $N(18, 3)$?