CpSc 418 **Homework 3** Due: Feb. 15, 2018, 1:00pm
Early Bird: Feb. 14, 2018, 1:00pm

44 points.

Please submit your solution using the handin program. Submit your solution as
cs418 hw3
Your submission should consist of one file:

- `hw3.pdf` or `hw3.txt` a PDF of plain ascii text file with your solutions.

1. **Tiling** (10 points)
   Many physical simulation problems are solved using *finite element methods*. This means that we model some two-dimensional or three-dimensional region using a grid of points. For simplicity, we will consider a two-dimensional, $N \times M$ grid. The simulation has a value, i.e. a number, for each of the NM points on the grid. At each step of the simulation, the value at point (i, j) is updated using the values of points $(i-1, j)$, $(i, j-1)$, (i, j), $(i+1, j)$, and $(i, j+1)$. While there are usually special cases at the boundaries, we'll ignore those to keep this question simple.

   A simple way to make a parallel version of this problem is to divide the grid into $K \times K$ *tiles*. Assume that K is a factor of both N and M. For example, if $N = 1600$, $M = 1000$, and $K = 100$, we would have 160 tiles. Tile (u, v) would have the points for $100*u \leq i < 100*(u+1)$ and $100*v \leq j < 100*(v+1)$. The computations for each tile are performed on a separate processor – if we have T tiles, we assume that the problem is running on T parallel processors. For each step of the simulation, the process for a tile will

   - Send the points on its edges to its neighbours. In particular, it will send a four different message with K points to its north, south, east, and west neighbours.
   - Receive messages from its four neighbours. Assume that the total cost to send and receive a message with K points is $\lambda + K$. This is the communication cost.
   - Update the values for its point. Assume that the cost to update a single point is 1. This is the computation cost.

   For simplicity, assume that the total time for a simulation step is the time for communication plus the time for computation – i.e. we don't get to overlap communication with computation.

   (a) (**2 points**) How many tiles are there if $N = 1600$, $M = 1000$, $K = 100$, and $\lambda = 1000$?

   (b) (**4 points**) What is the speed-up if $N = 1600$, $M = 1000$, $K = 100$, and $\lambda = 1000$?

   (c) (**4 points**) What is the parallel efficiency if $N = 1600$, $M = 1000$, $K = 100$, and $\lambda = 1000$?

2. **Hypercube Bisection** (18 points)
   Consider a d-dimensional hypercube. Such a hypercube has 2^d processors. Let $N = 2^d$. We identify the processors by their indices, I, with $0 \leq I < N$. For example a four-dimensional hypercube has processors with indices 0, 1, ... 15. Each processor has d bi-directional links. Processor I has links to processors I `bxor` 2^K, for $0 \leq K < d$, where `bxor` indicates bit-wise exclusive-or (`bxor` is the Erlang operator. In C, Java, or Python, use `^`) For our four-dimensional hypercube example, processor 3 has links to:

```
3 bxor 1 =  2,    3 bxor 2 =  1
3 bxor 4 =  7,    3 bxor 8 = 11
```

   (a) (**2 points**) What are the neighbours of processor 6?

(b) (**2 points**) Describe a path from processor 3 to processor 14.
Notes: your path should traverse three links. There is more than one correct answer to this question.

```
% blend_bits(I, J, K) ->
%      the K least-significant bits of J bitwise OR'd with
%      all but the K least-significant bits of I
%      Example: blend_bits(2#010110, 2#011101, 3) -> 2#010101
blend_bits(I, J, K) ->
  Mask = -1 bsl K,
  (I band Mask) bor (J band bnot Mask).
```

Note that `2#010110` is Erlang syntax for an integer written in binary.

We can send a message from processor I to processor J in d time-steps. For $0 \leq K < d$, on step $K<$ we send the message from processor `blend_bits(`I`, `J`, `K`)` to processor `blend_bits(`I`, `J`, `$K+1$`)`. In particular, if the K^{th} bits of I and J are the same, then we do nothing on the K^{th} time-step. If the K^{th} bits of I and J different, then we send the message along the dimension-K link from processor `blend_bits(`I`, `J`, `K`)` to processor `blend_bits(`I`, `J`, `$K+1$`)`. This is essentially *dimension-ordered routing*, but dimension-ordered routing skips the steps where I and J agree in their K^{th} bits. Dimension-ordered routing can save a few steps, but the remaining parts of this question are a bit simpler when we can assume that a message can be sent from I to J in d time steps.

Using the routing algorithm described above, consider sending a message from processor `22=2#010110` to processor `29=2#011101` in a 6-dimensional hypercube. The sequence of steps is:

```
step 0:  2#010110 → 2#010111, % 22 → 23
step 1:  2#010111 → 2#010101, % 23 → 21
step 2:  2#010101 → 2#010101, % no change
step 3:  2#010101 → 2#011101, % 21 → 29
step 4:  2#011101 → 2#011101, % no change
step 5:  2#011101 → 2#011101, % no change
```

(c) (**2 points**) Write the sequence of steps to send a message from node `45=2#101101` to node `25=2#011001`.

Let C be any integer with $0 \leq C < N = 2^d$. Let's say that for each processor, $0 \leq I < N$, processor I sends a message to processor I `bxor` C. All messages can be delivered using the algorithm described above with d steps. The key observation is that at step K, if bit K of C is a 1, the processor I sends a message to processor I `bxor` 2^K, and processor I `bxor` 2^K sends a message to processor I. If bit K of C is a 0, then no messages are sent on that step.

As an example, consider a four-dimensional hypercube, and let $C = 7 =$ `2#0111`. On step 0:

processors 0 and 1 exchange messages;
processors 2 and 3 exchange messages;
processors 4 and 5 exchange messages;
processors 6 and 7 exchange messages;
processors 8 and 9 exchange messages;
processors 10 and 11 exchange messages;
processors 12 and 13 exchange messages;
processors 14 and 15 exchange messages.

(d) (**2 points**) For a four-dimensional hypercube with $C = 7$, which processors exchange messages on step 1.

We've showed above that for any $0 \leq C < N = 2^d$, each processor I can send a message to processor I `bxor` C and the total time to deliver all N messages is d. Furthermore, each processor can send another message at each step. While each message will take d units of time to be delivered, messages that are start on different steps will be sent over different links. The hypercube allows each processor to send d messages (one on each link) and receive d messages (one on each link) in each step. All of the routing and managing of links is done by hardware in parallel by hardware in the processor's network interface. All of this means that every processor, I, can send one message to processor I `bxor` C every time-step.

That's a lot of set-up that I included so you'd have some examples of routing on a hypercube. We are now ready to prove the main result for this problem.

(e) (**10 points**) Let X and Y be disjoint subsets of $\{0, 1, \ldots N-1\}$ where X and Y each have $N/2$ elements. Show that for any choice of X and Y, there must be a C such that if each processor I sends a message to processor I `bxor` C, then there must be at least $N/4$ messages sent from processors in X to processors in Y and at least $N/4$ messages sent from processors in Y to processors in X.
Hint: This problem is asking you to argue about the choice of C that maximizes the number of messages sent from X to Y. You can do this by determining the total number of such messages summed over *all* possible choices for C. What is the average? The maximum must be at least as large as the average.

We have now shown that for any partition, X and Y, there is some choice of C such that at least $N/4$ messages are sent from X to Y and $N/4$ messages are sent from Y to X. We also showed that such messages can be sent every time-step. Thus, $O(N)$ message must cross between X and Y at each time step. This holds for any choice of X and Y. Thus, the bisection width of the hypercube is $O(N)$. In class, we showed that in three-dimensional space, any network topology whose bisection width is greater than $\Omega(N^{2/3})$ becomes asymptotically "all wire"; in other words, the fraction of the total volume that is occupied by processors goes to zero as the number of processors goes to infinity. We conclude that hypercube processors are asymptotically all wire. This explains why hypercubes were popular for machines with a few hundred or a few thousand processors, but have not seen widespread use for larger supercomputers.

3. **Bitonic Sort** (16 points)
In class we described the bitonic sorting algorithm assuming that N was even, and that we could recursively divide the sorting and merging tasks into subproblems of size $N/2$ until we get tasks of size 2 that can be solved using single compare-and-swap elements. This requires that N is a power of 2. In this problem, we will generalize the approach to arbitrary values of N.

The key place where we assumed that N is even is in the lemma:

> **Bitonic lemma:** Let X be a bitonic sequence consisting of 0s and 1s of length N where N is even. Let
> $$\begin{aligned} Z_i &= \min(X_i, X_{i+\frac{N}{2}}), \quad 0 \leq i < \tfrac{N}{2} \\ &= \max(X_{i-\frac{N}{2}}, X_i), \quad \tfrac{N}{2} \leq i < N \end{aligned}$$
> Then, either $Z_0, \ldots, Z_{\frac{N}{2}-1}$ is all 0s and $Z_{\frac{N}{2}}, \ldots, Z_{N-1}$ is bitonic, or $Z_0, \ldots, Z_{\frac{N}{2}-1}$ is bitonic and $Z_{\frac{N}{2}}, \ldots, Z_{N-1}$ is all 1s.

I'll generalize the lemma for the case that N, but I'll assume that $X \in 0^*1^*0^*$; in other words, X is "up-down" bitonic. I'll address why it's OK to make this assumption at the end of the problem.

Better Bitonic lemma: Let X be a bitonic sequence with $X \in 0^*1^*0^*$. Let

$$\begin{aligned} Z_i &= \min(X_i, X_{i+\lceil \frac{N}{2} \rceil}), \quad 0 \le i < \lfloor \tfrac{N}{2} \rfloor \\ &= X_i, \qquad\qquad\qquad \lfloor \tfrac{N}{2} \rfloor \le i < \lceil \tfrac{N}{2} \rceil \\ &= \max(X_{i-\lceil \frac{N}{2} \rceil}, X_i), \quad \lceil \tfrac{N}{2} \rceil \le i < N \end{aligned}$$

Then, either $Z_0, \ldots, Z_{\lfloor \frac{N}{2} \rfloor - 1}$ is all 0s and $Z_{\lfloor \frac{N}{2} \rfloor}, \ldots, Z_{N-1}$ is bitonic, or $Z_0, \ldots, Z_{\lfloor \frac{N}{2} \rfloor - 1}$ is bitonic and $Z_{\lfloor \frac{N}{2} \rfloor}, \ldots, Z_{N-1}$ is all 1s.

Hint: OK, I owe you a hint because I had to fix the problem statement (Feb. 9, 2018). Note that any subsequence of a bitonic sequence is bitonic. In particular, the sequence

$$X_0, \ldots, X_{\lfloor \frac{N}{2} \rfloor - 1}, X_{\lceil \frac{N}{2} \rceil}, \ldots, X_{N-1}$$

is bitonic. In English, this is the the same as X if N is even, and all of X except $X_{\lfloor \frac{N}{2} \rfloor}$ if N is odd. This modified sequence has even length whether N is even or odd; so, you can apply the original lemma. Then, figure out what happens with $X_{\lfloor \frac{N}{2} \rfloor}$ when N is odd.

Is it a big restriction to assume that $X \in 0^*1^*0^*$. Not really. In the bitonic sorting algorithm, we always know which kind of bitonic sequence we have at any point in the algorithm. The lemma can be generalized to handle $X \in 1^*0^*1^*$ – i.e. for "down-up" bitonic sequences. In this case, the odd element gets appended to the $Z_0, \ldots, Z_{\lfloor \frac{N}{2} \rfloor - 1}$ sequence instead of prepending it to the $Z_{\lceil \frac{N}{2} \rceil}, \ldots, Z_{N-1}$ sequence.

(a) (**2 points**) Give an example for X and Z where $N = 9$, $Z_0, \ldots, Z_{\lfloor \frac{N}{2} \rfloor - 1}$ is all 0s and $Z_{\lfloor \frac{N}{2} \rfloor}, \ldots, Z_{N-1}$ is bitonic. Your choice should produce $Z_{\lfloor \frac{N}{2} \rfloor}, \ldots, Z_{N-1}$ that **is not** all 0s or all 1s.

(b) (**2 points**) Give an example for X and Z where $N = 9$, $Z_0, \ldots, Z_{\lfloor \frac{N}{2} \rfloor - 1}$ is bitonic and $Z_{\lfloor \frac{N}{2} \rfloor}, \ldots, Z_{N-1}$ is all 1s. Your choice should produce $Z_0, \ldots, Z_{\lfloor \frac{N}{2} \rfloor - 1}$ that **is not** all 0s or all 1s.

(c) (**2 points**) Give an example for X and Z where $N = 9$, $X_0, \ldots, X_{\lfloor \frac{N}{2} \rfloor - 1}$ is not all 0s or all 1s, $X_{\lfloor \frac{N}{2} \rfloor}, \ldots, X_{N-1}$ is not all 0s or all 1s, $Z_0, \ldots, Z_{\lfloor \frac{N}{2} \rfloor - 1}$ is all 0s, and $Z_{\lfloor \frac{N}{2} \rfloor}, \ldots, Z_{N-1}$ is all 1s.

(d) (**10 points**) Prove the better bitonic lemma.
Hints:

- See the proof for the original lemma from the slides.
- You can divide your proof into cases for N even and N odd. You can state that if N is even, the proof for the original lemma applies (and you don't need to give more details).
- For the case where N is odd, you can state that you assume that $X \in 0^*1^*0^*$ and that the argument when $X \in 1^*0^*1^*$ is similar. Of course, you should look over your proof to make sure that the two cases are symmetric, but they should be for any reasonable proof.
- You may find it useful to observe that for $N \ge 3$,

  $$X_0, \ldots, X_{\lceil \frac{N}{2} \rceil - 2}, X_{\lceil \frac{N}{2} \rceil}, X_{N-1}$$

  is bitonic. This is because X is bitonic (by hypothesis), and any subsequence of a bitonic sequence is bitonic. For this observation, this is the subsequence obtained by deleting the element $X_{\lceil \frac{N}{2} \rceil - 1}$.