CpSc 418 **Final Exam** December 5 2013

100 points (101 possible)

Time for the exam: 2 hours, 30 minutes.

Open book: anything printed on paper may be brought to the exam and used during the exam. This includes the textbook, other books, printed copies of the lecture slides, lecture notes, homework and solutions, and any other material that a student chooses to bring.

Calculators are allowed: no restriction on programmability or graphing. There are a few simple calculations needed in the exam, a calculator will be handy, but the fancy features will not make a difference.

No communication devices: That's right. You may not use your cell phone for voice, text, web-surfing, or any other purpose. Likewise, the use of computers, iPods, etc. is not permitted during the exam.

Time for the exam: 2 hours, 30 minutes. Open book: anything printed on paper may be brought to the exam and used during the exam. This includes the textbook, other books, printed copies of the lecture slides, lecture notes, homework and solutions, and any other material that a student chooses to bring. Calculators are allowed: no restriction on programmability or graphing. There are a few simple calculations needed in the exam, a calculator will be handy, but the fancy features will not make a difference. No communication devices: That's right. You may not use your cell phone for voice, text, web-surfing, or any other purpose. Likewise, the use of computers, iPods, etc. is not permitted during the exam.

CpSc 418 **Final Exam** December 5, 2013

Graded out of **100 points** (102 points are possible)

1. Reduce and Scan (**24 points**) One of the problems below can be solved using reduce, the other can be solved using scan. Identify which is which. For the problem that can be solved using reduce, describe the `Leaf`, `Combine`, and `Root` functions – the `Root` function may (or may not) be the identity function. For the problem that can be solved using scan, describe the `Leaf1`, `Leaf2`, and `Combine` functions. Your solutions should be a reasonable approximation of Erlang code, but we won't insist that it compile.

   (a) (**12 points**) Given a list of numbers that is distributed across the worker processes, compute a new list (also distributed across the processes) whose i^{th} element is the maximum element up to and including the current one minus the minimum element up to and including this one. I'll call this function `gap` and here are a few examples:

   ```
   gap([5,2,6,18,-3,4,8,-1,20]) -> [0,3,4,16,21,21,21,21,23];
   gap([]) -> [];
   gap([42]) -> [0];
   ```

   (b) (**12 points**) Given a list of numbers, report the number of consecutive Pythagorean triples. If `x`, `y`, `z` are consecutive elements of the list, then they are a Pythagorean triple iff $\texttt{x}^2 + \texttt{y}^2 = \texttt{z}^2$. Here are few examples:

   ```
   nPythag3([1,2,4,6,8,10,11,3,4,5,12,13,14]) -> 3; % [6,8,10], [3,4,5], [5,12,13]
   nPythag3([1,-2,3,-4,5]) -> 1; nPythag3([0,1,0,1,0,1,0,1]) -> 3;
   ```

2. **Filter Locks (16 points)** I've mentioned several times that Peterson's mutual exclusion algorithm can be extended to apply to any number of clients. The algorithm is called a "filter lock". Here's the code:

   ```
   00: int level[N], victim[N]; // initially all 0

   01: lock(i) {
   02:   for(int j = 1; j < N; j++) {
   03:     level[i] = j;
   04:     victim[j] = i;
   05:     while((victim[j] == i) && ∃ 0 ≤ k < N. (k ≠ i) && level[k] ≥ j);
   06:   }
   07: }

   08: unlock(i) {
   09:   level[i] = 0;
   10: }
   ```

   The key idea in the lock is that each thread must pass through $N-1$ "levels" of locking (the `for(j...)` loop) before entering its critical section. To advance from level `j` to level `j+1`, there must be no other threads at level `j` or higher, **or** there must be some other thread that is the "victim" at level `j`. A consequence is that if there is another thread at this level or higher, then there must be at least one thread "left behind" at each of the lower levels. In particular, if thread x is in the critical section, and thread y is at level $N-1$, there are threads at every level from 1 to $N-1$; thread y is the victim at level $N-1$; and thread y can't advance to the critical section.

   (a) (**8 points**) Show that the filter lock is **not** starvation free.

   (b) (**8 points**) Show that if the statements at lines `03` and `04` are exchanged (i.e. `victim[j] = i;` is performed *before* `level[i] = j;`), then mutual exclusion can be violated.

3. **Sorting (25 points)**

The question finishes showing that the compare-and-swap elements of sorting networks can be replaced by "merge-and-split" operations. Homework 4, question 2, also looked at this. This can be used to obtain practical parallel sorting algorithms for common parallel architectures.

Let M be a positive integer. We will consider merge-and-split operations on lists of length M. In particular,

```
merge_and_split({In0, In1}) -> {Out0, Out1}
    where:
        In0 and In1 are lists of M elements in ascending order.
        Out0 is the first M elements of merge(In0, In1), sorted into ascending order.
        Out1 is the last M elements of merge(In0, In1), sorted into ascending order.
```

For example:

```
merge_and_split({[1, 4, 9, 16, 25, 36], [3, 5, 7, 9, 11, 13]}) ->
   {[1, 3, 4, 5, 7, 9], [9, 11, 13, 16, 25, 36]}.
```

(a) (**5 points**) What is the result of

```
merge_and_split({[0, 0, 0, 0, 1, 1], [0, 0, 0, 1, 1, 1]})
```

?

(b) (**5 points**) Show that if `In0` and `In1` are lists of 0s and 1s in ascending order, and

```
{Out0, Out1} = merge_and_split({In0, In1})
```

then at least one of `Out0` or `Out1` must be clean (i.e. all 0s or all 1s).

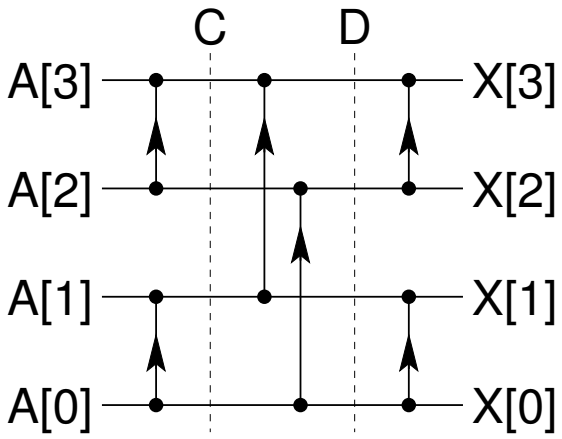
(c) (**10 points**) Let S be a sorting network with N inputs and N outputs. Let S' be the network obtained by replacing every compare-and-swap in S with a merge-and-split operation. Now, let $A_0, A_1, \ldots, A_{N-1}$ be any N sorted lists of 0s and 1s of length M. Apply these lists as the inputs to S', and let $X_0, X_1, \ldots, X_{N-1}$ be the resulting outputs.

Show that there is an input to S', $B_0, B_1, \ldots, B_{N-1}$ such that B is a permutation of X and when B is applied as the input to S', X is the output.

Note: If B is a permutation of X that means there is a one-to-one function, p, such that $B_i = X_{p_i}$.

Hints:

- Use induction on the structure of S'.
- See figure 1 to get an example of the A, X, and B lists, and the function p.
- You may assume the result from question 3b when solving this problem.
- Question 3d is on the next page (below the figure).



Example:

```
A[0] = [0,1,1,1,1], A[1] = [0,0,1,1,1], A[2] = [0,0,0,1,1], A[3] = [0,0,0,0,1]
C[0] = [0,0,0,1,1], C[1] = [1,1,1,1,1], C[2] = [0,0,0,0,0], C[3] = [0,0,1,1,1]
D[0] = [0,0,0,0,0], D[1] = [0,0,1,1,1], D[2] = [0,0,0,1,1], D[3] = [1,1,1,1,1]
X[0] = [0,0,0,0,0], X[1] = [0,0,1,1,1], X[2] = [0,0,0,1,1], X[3] = [1,1,1,1,1]
p(i) = i, B = X
```

Note that the lists for B are the same as those for X. If B is applied to the sorting network shown above (that doesn't sort correctly), the network produces the same result as applying the lists from A. For this example, the permulation, p, is the identity function. For some sorting networks, you may need a permutation, e.g. `B[0] = X[5]`, `B[1] = X[2]`, ...

Figure 1: Sorting Example

(d) (**5 points**) The induction argument from question 3c shows that if the lists $B_1, \ldots, B_{N-1}$ are applied as the inputs to S', then for each `merge_and_split` operation in S', either `{Out0, Out1} = {In0, In1}`, or `{Out0, Out1} = {In1, In0}`. For $0 \leq i < N$, let C_i be the number of 1s in list B_i. Show that if S' sorts the B lists incorrectly, S must sort C incorrectly as well.

Hint: you can do this even if you didn't solve question 3c. In particular, you are allowed to assume what I said about the `merge_and_split` operations.

4. **Matrix Multiplication** (20 points)
Let's say you get a job writing code at a local company that needs to do some moderately intense number crunching. A typical problem is computing products of $N \times N$ matrices, where N ranges from 1000 to 20000.

   (a) (**4 points**) Let's say you've got a laptop with a 2.5GHz, quad-core processor. From mini-assignment 4, we know that multiplying a $N_1 \times N_2$ matrix by a $N_2 \times N_3$ matrix on a machine with a clock frequency of f takes time of roughly $3N_1N_2N_3/f$. What is the time to compute a matrix products for $N = 1000$, $N = 5000$, and $N = 20000$?

   (b) (**4 points**) You try to convince your manager to buy a cluster of linux machines for your computation. Your proposal is for 32, 8-core machines running at 2.5GHz. The total number of processors is 256. For simplicity, we'll pretend that there isn't any multi-threading. Assume that the time to transfer a block m double-precision values between processors (over the network, it's a cluster) is:

$$T_{network} \quad = \quad 0.1\text{ms} + m * 50\text{ns}$$

   where $1\,\text{ms} = 10^{-3}\,\text{seconds}$, and $1\text{ns} = 10^{-9}\,\text{seconds}$.

   Using the algorithm we derived in class (Sept. 24), each processor performs P matrix products where it multiplies a $(N/P) \times (N/P)$ matrix by a $(N/P) \times N$ matrix. Furthermore, each processor sends (and receives) $P - 1$ messages of size N^2/P double precision numbers. For simplicity, assume that the time for sending messages cannot be overlapped with computation time. What is the speed-up when computing matrix products for $N = 1000$, $N = 5000$, and $N = 20000$?

   (c) (**4 points**) Your manager isn't convinced that the speed-up is sufficient to justify the cost, but you really want to her to buy that linux cluster. You read up on algorithms for matrix multiplication and find an algorithm where each processor performs $\sqrt{P}$ matrix products where it multiplies a $(N/\sqrt{P}) \times (N/\sqrt{P})$ matrix by another $(N/\sqrt{P}) \times (N/\sqrt{P})$. Furthermore, each processors sends (and receive) $\sqrt{P}$ messages of size N^2/P double precision numbers. With the new algorithm, what is the speed-up when computing matrix products for $N = 1000$, $N = 5000$, and $N = 20000$?

   (d) (**8 points**) A coworker installs an optimized BLAS (numerical library) which is faster than the simple code I showed in class. You find that the time to multiply a $N_1 \times N_2$ matrix by a $N_2 \times N_3$ matrix on a machine with a clock frequency of f now takes time of roughly $1.2N_1N_2N_3/f$. What is the sequential time (as in question 4a and parallel time (as in question 4c when the BLAS library is used?

      i. What is the sequential time (as in question 4a and parallel time (as in question 4c when the BLAS library is used?
      ii. Write one sentence to summarize the impact of the faster sequential algorithm on the execution time.
      iii. Write one sentence to summarize the impact of the faster sequential algorithm on the speed-up.
      iv. Write one sentence to explain the relationship you observe between execution time and speed-up?

5. **Other Stuff (16 points)**
Answer each question below with 1-3 sentences. Points may be taken off for long answers.

   (a) (**4 points**) What is instruction level parallelism?

   (b) (**4 points**) What is "single-instruction, multiple thread" parallelism?

   (c) (**4 points**) How does map-reduce handle machine and network failures?

   (d) (**4 points**) What is a memory fence?