CpSc 418 **Midterm** February 15, 2023

Graded out of **100 points**

Q	Pts
0.	
1.	
2.	
3.	
4.	
Total	

Time for the exam: 50 minutes.

Open book: anything printed on paper may be brought to the exam and used during the exam. Laptops and calculators are allowed. You are allowed to look up course materials or anything else you want with your computer.

No communication: you are not allowed to send or receive e-mail, use chat, or other communication software. Having such programs open will immediately have you removed from the exam and will be treated as academic misconduct.

Piazza is OK: we will have a piazza thread open. You can make private posts to the instructor if you have a question, are claiming a bug bounty, etc. You are also welcome to raise your hand and the instructor or TA will walk to your seat to hear your question. Any clarifications to the exam will be posted on the piazza thread for all to see.

You can only use Piazza for CPSC 418 – not the Piazza pages for other classes.

No Test books: We have included space with each question for you to write your answers. There are a few blank pages at the end that you can use if you need more space. Note that the space that we provide is intended to be generous.

Bug bounties are in effect and given in midterm points. Only report a suspected error if it affects your ability to complete the exam. To report an error, raise your hand. Due to Mark's hearing difficulties, he may need to step out in the hall to hear you – he doesn't understand whispers. We will post any corrections to piazza, and it will be posted on the screen.

Minor spelling, grammar, and similar errors should be posted to piazza after the exam. If the "error" does not impact your understanding of what question is being asked of how to solve it, then it is a minor error.

Good luck!

CpSc 418 **Midterm** February 15, 2023

Graded out of **100 points**.

0. **Who are you?** (2 points, extra credit)

   (a) Your name: _____

   (b) Your student number: _____

1. **`frog`s and `dog`s** (15 points)
   Consider the following function that returns true if there are more frogs than dogs in a list:

```
0: frog_vs_dog(List) ->
1:   N_frogs = length([frog || X <- List, X == frog]),
2:   N_dogs = length([dog || X <- List, X == dog]),
3:   N_frogs > N_dogs.
```

   As Finn explained in class, Erlang list comprehensions are not implemented with tail-recursion.

   (a) (5 points) Under what condition (i.e. some property of `List`) could this be a problem? Use one sentence to state the condition, and one sentence to say what the problem that causes.

   (b) (5 points) Write a tail recursive function, `fvd_help(List, Acc)`, that returns the value of `N_frogs - N_dogs`. `fvd_help` should use tail-recursion instead of a list comprehension. Do not use functions from the `lists` module or other modules in the Erlang/OTP or class libraries. My solution:
   - uses four patterns, each of which is a single, simple line of Erlang.
   - initializes `Acc` to `0` (see "Question" 1c below.
   - `Acc` is the difference between the number of `frog`s and number of `dog`s seen so far.

```
 0: avg(W, Key) ->
 1:   {Sum, N} = wtree:reduce(W,
 2:     fun(ProcState) -> % Leaf
 3:       MyList = wtree:get(ProcState, Key),
 4:       { lists:sum(MyList),
 5:         % you_need_to_write_this(...)
 6:       }
 7:     end,
 8:     fun({LeftSum, LeftLen}, {RightSum, RightLen}) -> % Combine
 9:       { % you_need_to_write_this(...),
10:         LeftLen + RightLen
11:       }
12:     end
13:   ),
14:   Sum/N.
```

Figure 1: Erlang code for computing average with reduce.

(c) (0 points) I was going to ask you to modify `frog_vs_dog(List)` to use `fvd_help`, but I decided to keep this exam shorter. My revised `frog_vs_dog` is:

```
0: frog_vs_dog(List) -> fvd_help(List, 0) > 0.
```

(d) (5 points) When is it essential to use a tail-recursive function in Erlang? A short (one or two sentence answer) is ideal.

2. **Reduce** (25 points)
Consider the code shown in Figure 1 for computing the average of the elements of a list of numbers. The list is distributed across the workers of `W` and is associated with the key `Key`. If `L` represents the list we could get by concatenating the segments for each worker, then `avg` should return

$$\texttt{avg(W, Key)} = \frac{1}{\texttt{length(L)}} \sum_{\texttt{I=1}}^{\texttt{length(L)}} \texttt{L}_\texttt{I}$$

where $\texttt{L}_\texttt{I}$ denotes the $\texttt{I}^{\text{th}}$ element of list `L`.

(a) (10 points) As you can see, I forgot to write two of the lines of code (lines 5 and 9). Write the code that completes the function:

line 5:

line 9:

(b) (15 points) I want a function, `avg_sq(W, Key)` that computes the average of the squares of the elements of the list associated with `Key`:

$$\texttt{avg\_sq(W, Key)} = \frac{1}{\texttt{length(L)}} \sum_{\texttt{I=1}}^{\texttt{length(L)}} \texttt{L}_\texttt{I}^2$$

Write the changes you need to make to the code for `avg` to implement `avg_sq`. You may assume that your changes from Question 2a have been made. Just list line numbers, and the new code to put at each such line. Hint: my solution changes one line and can be written in one line.

3. **Speed-Up** (30 points)
Consider the computation of the average of the elements in a list using Erlang as shown in Figure 1. Assume that `lists:sum(List)` takes time $5\text{ns} \times$ `length(List)` where 1ns (one nanosecond) is 10^{-9} seconds. Assume that the time to send a message from one process to another is 5μs where 1μs (one microsecond) is 10^{-6} seconds. For simplicity,

- You can ignore the time to evaluate the `Combine` function – it's much smaller than the communication time.
- You can ignore the time to do the final division to compute the average – it's small compared to everything else.
- You don't need to remember the exact details of how `wtree:reduce` is implemented. We will accept any reasonable answer where "reasonable" means to within roughly a factor of two of what `wtree:reduce` actually does. If you're worried, state your assumptions.

(a) (5 points) With the assumptions above, what is the time to compute the average of a list of 1,600,000 elements using one processor (no `wtree` code)?

(b) (5 points) With the assumptions above, what is the time to compute the average of a list of 1,600,000 elements using `avg(W, Key)` when `W` is a worker-tree with 16 workers?

(c) (5 points) What is the speed-up?

(d) (10 points) Now, we convert our code to C. Assume that computing the sum of the elements of an array takes 0.5ns per element, and that the communication time remains 5μs per communication. What is the sequential time, the parallel time, and the speed-up for the C-version of the program?

(e) (5 points) Did the C-version have a larger speed-up or a smaller one? Why? (one or two sentences to answer "Why"?) is sufficient.

4. **The last question** (30 points)
   Answer any three of the four questions below. We will deduct points if you answer all four.

   (a) (10 points) What problem is addressed by MapReduce?

   (b) (10 points) What is the difference between the M and E states in the MESI protocol? Reminder: M=modified, E=exclusive, S=shared, I=invalid.

   (c) (10 points) Name one thing that Lynn Conway is known for.

   (d) (10 points) Which is better, chocolate or szechuan peppercorns? Hint: don't overthink this. Correct answers include but are not limited to "chocolate", "peppercorns", "sushi", or "kayaks". If there is no English word for your answer, write your answer in your preferred language, and write the name of the language.