CpSc 418

Homework 2

Due: Oct. 10, 2013, 11:59pm

Four questions, 105 points. 5% extra credit if solution submitted by 11:59pm on Oct. 6.

Please submit your solution using the handin program. Submit the your solution as
cs418 hw2
Your submission should consist of the following two files:

`hw2.erl` – Erlang source (ASCII text).

`hw2.txt` or `hw2.pdf` – plain, ASCII text or PDF.

The first file, `hw2.erl`, will be your solution to the programming part of the assignment. Unless otherwise stated, you may use any functions that you like from the Erlang library, for example, functions from the `lists` module.
The second file, `hw2.txt` or `hw2.pdf` should give your solutions to the written parts of the assignment.
No other file formats will be accepted.

1. **Parallel Corners (35 points).**

   (a) (**20 points**) Use `wtree:reduce` and your solution to the count corners problem from homework 1 to implement a parallel version of the corner counting problem.
   Your solution should be a function:

   ```
   count_corners(Key, W) -> integer()
   ```

   Where `Key` is the association key for a list that's been distributed across the workers of `W`, and `W` is a worker pool.
   If you don't have a working solution to the sequential count-corners problem, you may use the solution that will be posted by Sept. 28, and state that you are doing so.

   (b) (**5 points**) Measure speed-up versus number of Erlang processes in the worker pool when using 1, 2, 4, 8, 16, 64, and 256 processes running on any of `lin01.ugrad.cs.ubc.ca` through `lin25.ugrad.cs.ubc.ca` with a list of 100,000 elements. Each of these machines has a quad-core, two-way multithreaded Intel Core i7 processor.

   (c) (**5 points**) For the number of threads that achieved the highest speed-up, what is the shortest list that achieves at least 90% of this speed-up. Because timing measurements are inexact, you just need to get a number that is reasonably close to the shortest such length, and a speed up that is reasonably close to 90%. State what you mean by reasonable.

   (d) (**5 points**) Repeat parts b and c running on `gambier.ugrad.cs.ubc.ca`. Gambier has a SPARC T2 processor with eight cores, each of which is eight-way multithreaded. It's an old machine; so it won't be as fast in absolute terms as the linux boxes, but it demonstrates multi-threading nicely.

   Recap of count corners from homework 1. Let `L` be a list of `N` elements. We will say that the $\mathtt{I}^{th}$ element of `L` is "ascending" if it is less than its successor, descending if it is greater than its successor, and "flat" if it is equal to its successor.

   An element is a "corner" if:

   - it is an ascending element, followed by zero or more flat elements followed by a descending element, or
   - it is a descending element, followed by zero or more flat elements followed by an ascending element.

   The count-corners problem is to determine the number of corners in a list. If the length of the list is less than 3, then the number of corners is 0.

2. **Non-parallelizable code (25 points)** For each question below, and answer with two significant digits of accuracy (e.g. 42,000,000) is acceptable. Show your work in enough detail that someone reading your solution can reconstruct your answers.

(a) **(5 points)** Consider a sequential function `f(L)` that when run on a list of length $\mathtt{N}$ has a runtime of $3\mathtt{N} + 5\mathtt{N}\log_2 \mathtt{N}$ nanoseconds. What is the length of the longest list for which f can run in at most 10 seconds?

(b) **(5 points)** Assume that the linear time part is non-parallelizable, but that the $\mathtt{N}\log\mathtt{N}$ part can be perfectly parallelized. Then, the run time on P processors is $3\mathtt{N} + \frac{5N}{P}\log_2 \mathtt{N}$. What is the length of the longest list for which the parallel version of f can run in at most 10 seconds when $P = 100$? What is the speed-up for this choice of $\mathtt{N}$ and P?

(c) **(5 points)** Now, consider a sequential function `g(L)` that when run on an input of size $\mathtt{N}$ has a runtime of $10\mathtt{N} + \mathtt{N}^{2.5}$ nanoseconds. What is the largest input size for which g can run in at most 10 seconds?

(d) **(5 points)** Assume that the linear time part is non-parallelizable, but that the $\mathtt{N}^{2.5}$ part can be perfectly parallelized. What is the largest input size for which the parallel version of g can run in at most 10 seconds when $P = 100$? What is the speed-up for this choice of $\mathtt{N}$ and P?

(e) **(5 points)** Which of f or g achieved the larger speed-up when considering problem sizes that must complete within 10 seconds? For which of f or g could the largest tractable problem size be increased by the greatest factor?

3. **K Largest Elements of a List (35 points)**
We've described the problem of finding the `K` largest elements of a list or array as an example of a `reduce` operation. In this assignment, we'll look at the sequential version; we'll examine parallel implementations in homework 3. Let `maxk(L, K)` return the `K` largest elements of `L` in ascending order. If `length(L) < K`, then `maxk(L, K)` should return all of the elements of `L` in ascending order. The function `hw2:maxx(L, K)` provides one implementation of the `maxk` function.

(a) (**5 points**) Show that the worst-case run time for `hw2:maxx(L, K)` is

$$O\left((\mathtt{K}\log\mathtt{K}) + (\mathtt{N}-\mathtt{K})\mathtt{K}\right)$$

where $\mathtt{N} = \mathtt{length(L)}$ and I'm assuming $\mathtt{K} \le \mathtt{N}$. Note that if $\log\mathtt{K} \ll \mathtt{N}$, the worst-case runtime can be simplified to $O((\mathtt{N}-\mathtt{K})\mathtt{K})$, and if $\mathtt{K} \ll \mathtt{N}$, the worst-case runtime can be further simplified to $O(\mathtt{NK})$.
Hint: The worst-case is achieved if the elements of `L` are in ascending order.

(b) (**5 points**) Measure the run-time for `hw2:maxx(L, K)` for

$$\begin{array}{rcl} \mathtt{length(L)} & \in & \{10000,\ 20000,\ 30000,\ 50000,\ 70000,\ 100000\} \\ \mathtt{K} & \in & \{500,\ 1000,\ 1500,\ 2000\} \end{array}$$

(a total of 24 data points). For all of these cases, $\log\mathtt{K} \ll \mathtt{N}$. Find a constant a so that the measured run-time is approximated by

$$T \quad \approx \quad a(\mathtt{N}-\mathtt{K})\mathtt{K}$$

If you can't find a good approximation, state why. Please state what machine you ran the experiments on: either the name of a CS department machine, or state the type of CPU and clock frequency. For example, it's fine to run the experiments on your own computer.

(c) (**10 points**) For large lists and large values of `K`, we can improve the execution time by building a heap. The function `hw2:list_to_heap` converts a list to a heap. This heap has four kinds of nodes:

`[Max, Left, Right]` where `Max` is the largest value in the sub-heap rooted at this node; `Left` is the left sub-heap, and `Right` is the right sub-heap. By convention, `Left` and `Right` are chosen so that `Left` contains the largest element of the sub-heap rooted at this node.

`[Max, Min]` there are exactly two elements in this sub-heap with $\mathtt{Max} \ge \mathtt{Min}$.

`[V]` there is exactly one element in this sub-heap, and it has the value `V`.

`[]` this heap is empty. We never have `[]` as a sub-heap.

For example,

```
1> hw2:list_to_heap([1,-3,14,207,23,18,17,17,93,5,11]).
[ 207,
  [207, [ 207, [207, 14], [1,  -3]]
        [ 23,  [23,  18], [17, 17]]]
  [93,  [ 93,  5], [11]]]
```

If `H` is a heap, then `hw2:get_max(H)` returns the largest element of `H` – if `H` is empty, then `hw2:get_max(H)` returns the atom `'-infinity'`. The maximum depth of a node in the heap returned by `hw2:list_to_heap(L)` is $\lceil \log_2$ `length(L)` $\rceil$ (assuming `length(L)` > 1).

Write a function, `hw2:del_max(Heap)` that returns `NewHeap`, where `NewHeap` has all of the elements of `Heap` except for the largest. If `Heap` had M identical elements that were the largest, then `NewHeap` will have `M-1` such elements. The maximum depth of a node in `NewHeap` must be less than or equal to the maximum depth of any element of `Heap`. Note that this doesn't require you to keep `NewHeap` nicely balanced after a large number of `del_max` operations.

Your implementation of `del_max(Heap)` function should run in time $O($`height(Heap)`$)$, where `height(Heap)` is the maximum depth of any node in `Heap`.

Hint: My implementation of `del_max(Heap)` is three lines of Erlang. If yours is more than 10, you should ask for help.

(d) (**10 points**) Use your `hw2:del_max` function to implement `hw2:maxy(L, K)`. This function should implement the `maxk` function with a time complexity of

$$O\left(N + K \log N\right)$$

Hint: My solution uses a helper function, and the two functions combined use a total of five lines of code. Again, if your solution seems much more complicated than this, ask for help.

(e) (**5 points**) Measure the run-time for your `hw2:maxy` function for the same test cases as for question 3b. You can use random lists as well as those generated by `lists:seq`. Find constants b_1 and b_2 so that the measured run-time is approximated by

$$T \quad \approx \quad b_1 \mathtt{N} \; + \; b_2 \mathtt{K} \log_2 \mathtt{N}$$

If you can't find a good approximation, state why. Please state what machine you ran the experiments on: either the name of a CS department machine, or state the type of CPU and clock frequency. It's fine to run the experiment on your own computer.

Note: if the `hw2:maxx(L, K)` function is run with a random list for `L`, then the run-time is $O(\mathtt{K}^2 \log(\mathtt{N}/\mathtt{K}))$, where `N` $=$ `length(L)` and assuming `K` $\ll$ `N`. This means that the simple algorithm (`hw2:maxx`) can work quite well for random lists and moderately small values of `K`.

4. **Feedback (10 points)**
For problem on this assignment:

(a) How long did it take you to solve the problem?

(b) Please rate each problem on a scale of 0 to 5 where

0 – Worthless tedium.

1 – Too much work, and I learned little.

2 – A typical homework problem.

3 – Definitely had a favorable learning/effort ratio.

4 – I learned a lot and had fun doing so.

5 – Wow, amazing, life changing, or similar.

Feel free to add other comments as well.