INFOB3CC: Concurrency
Exam 2 (solutions)
30 January 2020, 08:30 – 10:30

Please read the following instructions carefully:

- Write your *name* and *student number* here:

  ....
- Be prepared to identify yourself with your ID card when you submit your exam.
- A maximum of 80 points can be obtained by answering the questions of this exam, to be divided by 8 to obtain the final mark.
- Provide brief and concise answers. Overly verbose responses or nonsense added to otherwise good answers can deduct from your grade.
- If a question does not give you all the details you need, you may make reasonable assumptions. Your assumptions must be clearly stated. If your solution only works under certain conditions, state them.
- Fill in your answers in the space provided. If you run out of space, continue in a separate answer booklet and clearly indicate your name, student number, and the question number.
- You may use diagrams to help explain your answers.
- Read through the paper first and plan your time accordingly.
- Answer questions in English.
- *Good luck!* (:

Please keep in mind that there are often many possible solutions
and that these example solutions may contain mistakes.

Definitions

1. (a) (5 points) According to our definitions in the lecture, what is the difference between task parallelism and data parallelism?

Solution:

- In task parallelism a program is broken into different pieces (tasks) which may be executed in parallel over the available processors. The operation in each task may be different. Explicit threads synchronise with each other using locks/messages/STM which is often difficult to program. Modest parallelism is achievable.
- In data parallelism, the data is split over the available processors and the same task (function) is applied to each data element. Operate simultaneously on bulk data. Implicit synchronisation; single logical thread of control. Massive parallelism is achievable.

(b) (5 points) For efficient code execution, the layout of data in memory must be considered. Explain the difference between the AoS and SoA representations.

Solution:

- AoS: In the array of structures representation, the memory for each object is kept together. This means that the individual fields of successive objects in the array are scattered in memory, which (often) prevents efficient vectorisation.
- SoA: In the structure of array representation, a separate array for each field of the structure is used. This keeps memory access contiguous over the individual fields of the structure, which is often more efficient for parallelisation.

Segmented

2. (a) (5 points) A segmented `scan` can be defined in terms of a regular (non-segmented) `scan`, by the use of an operator lifting function $\oplus^s$. Anneke wishes to implement the segmented plus operator as follows:

$$(f_x, x) \ +^s \ (f_y, y) = \text{if } f_y \text{ then } (true, y) \text{ else } (false, x + y)$$

where x and y are the array values and f_x and f_y are the corresponding flag values. The operator to the `scan` function should satisfy some mathematical property or properties. Which property/properties does/do not hold for this implementation, but should?

Solution: Associativity

(b) (5 points) Explain (or draw) a parallel execution of (segmented) `scan` using Anneke's operator, which demonstrates that the operator does not work correctly.

Solution: In a parallel reduction, Anneke's operator will combine results across flag boundaries. Example: `[T,F,T,F]`.

(c) (5 points) Can you construct a segmented version of the `map` operator, using only a single regular (non-segmented) `map`? Motivate your response.

Solution: Yes. The segmented `map` simply ignores the flag value.

(d) (5 points) Can you construct a segmented version of the `fold` operator, using only a single regular (non-segmented) `fold`? Motivate your response.

Solution: No. A `fold` returns only a single value, whereas a segmented `fold` requires one return value per segment.

(e) (7 points) Run length encoding is a form of lossless data compression in which sequences of the same data value (the runs) are stored as a single value together with the count of how many times that value appeared. For example, the string `MMMOOOOOOOOOWWWWWWOOOOWWW` can be encoded as the array `[(3,'M'), (9,'O'), (6,'W'), (4,'O'), (3,'W')]`.

Using the parallel array functions discussed in the course, provide an implementation of how a run-length encoded string, encoded in an array of (count, value) pairs as shown above, can be decoded into the full uncompressed string. A high-level overview or pseudocode implementation is sufficient. Describe your solution, and include the operator used with the array function(s) of your answer.

Solution:

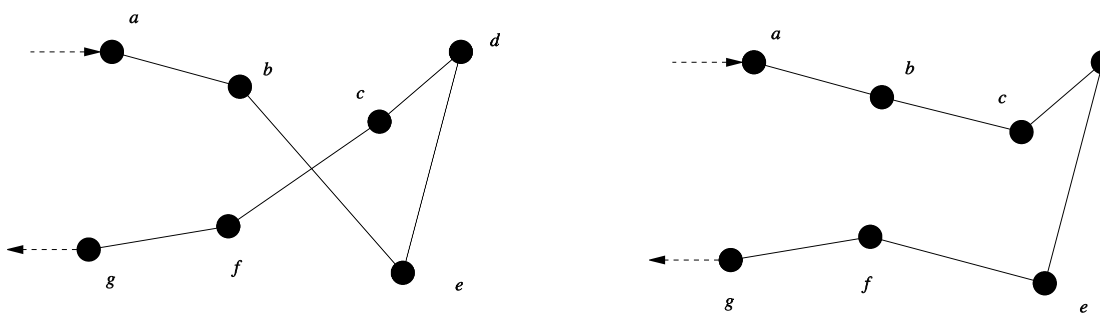
```
rle_decode :: Elt e ⇒ Acc (Vector (Int, e)) → Acc (Vector e)
rle_decode enc =
  let
      (seg, xs)       = A.unzip enc
      T2 offset len   = scanl' (+) 0 seg
      heads           = permute const
                          (fill (I1 (the len)) undef)
                          (λix → I1 (offset!ix))
                          xs
   in
   scanl1Seg const heads seg
```

Parcel Delivery

3. In this question we consider the schedule of a truck which delivers parcels. The route is stored in an array, where the first element of the array is the parcel to deliver first, the second element is the parcel to deliver second, and so on.

In the following questions consider which of the parallel array combinators discussed in the course can be used to implement the described functionality.

(a) (4 points) Some routes can be optimised using the procedure 2-Opt. This method flips the delivery order for part of the route. For example, consider the following delivery schedule:



The route on the left can be optimised using 2-Opt to the one on the right, that is:

$$[a, b, e, d, c, f, g] \xrightarrow{2-\text{Opt}} [a, b, c, d, e, f, g]$$

If the start and end indices of the part of the route to flip are given (in the example above, the indices for c and e), which function can be used to perform 2-Opt? Two answers are possible; select them both!

A. `fold`
B. `gather`
C. `map`
D. `scanl`
E. `scanr`
F. `scatter`
G. `stencil`
H. `zipWith`

(b) (5 points) Is there a reason to prefer one of the approaches in the previous question to the other? Motivate your answer.

Solution: Gather is always safe to execute in parallel. Scatter must deal with the function not being injective and/or surjective. Scattered writes also cause more inter-core communication when threads access the same cache line, even if there is no collision.

(c) (4 points) Using the function `drivingTime :: Parcel → Parcel → Time` we can compute the time to drive between any two parcel delivery addresses.[1] Which of the following operations can be used to compute an array where each element contains the driving time from the previous address? For example, the fourth element of the array will contain the driving time from the third parcel delivery point to the fourth delivery point. Select the correct response.

A. `fold`
B. `gather`
C. `map`
D. `scanl`
E. `scanr`
F. `scatter`
G. `stencil`
H. `zipWith`

(d) (4 points) It is important for the driver to know how long the deliveries will take. Given an array containing the time to drive between each parcel delivery address, what is the most efficient way to compute the total driving time for the route? Select the correct response.

A. `fold`
B. `gather`
C. `map`
D. `scanl`
E. `scanr`
F. `scatter`
G. `stencil`
H. `zipWith`

[1] In the Accelerate library, as used in the third practical, the function `drivingTime` would have an `Exp` type; we ignore this detail for brevity, here and through the remainder of the paper.

(e) (4 points) We want to inform clients of the expected time at which their parcel will be delivered. Given an array containing the time to drive between each parcel delivery address, how do we compute the expected delivery time of each parcel? Select the correct response.

A. `fold`
B. `gather`
C. `map`
D. `scanl`
E. `scanr`
F. `scatter`
G. `stencil`
H. `zipWith`

(f) (7 points) As the parcels can be very heavy, the total mass of the delivery truck should be taken into account when computing the driving time between delivery locations. Given the total mass of all of the parcels in the delivery truck, the function `slowdownFactor :: Mass → Double` returns a multiplicative factor which determines how much slower the delivery truck is compared to an empty truck. For example, a value of 1.0 is the normal driving speed, while a value of 2.0 means the delivery will take twice as long (the truck must drive at half the speed due to the extra mass). After each parcel is delivered, the truck is lighter, and thus for subsequent deliveries can drive faster. Taking this new information into account, how do we compute:

1. the expected delivery time for each parcel; and
2. the total time to deliver all parcels on the route?

Provide an implementation using the array functions discussed in the course (`fold`, `gather`, `map`, `scanl`, `scanr`, `scatter`, `stencil` and `zipWith`). A high-level overview or pseudocode implementation is sufficient. You may assume a function `parcelMass :: Parcel → Mass`, which gives the mass of each parcel. Describe your solution, and include the operator used with the array function(s) of your answer.

Solution:

1. `scanr (+)` to compute the weight of the truck after each delivery
2. `zipWith (λw t → t * slowdown w)` using the array from (1) and array of unadjusted delivery times between each address.
3. `scanl (+) 0` to compute the expected delivery times
4. `fold (+) 0` to compute the total delivery time (or `scanl'` above)

Efficient & Optimal

4. Mergesort is a divide-and-conquer comparison-based sorting algorithm. To sort an array, the algorithm will recursively sort the first and last halves of the input, and then merge the two sorted sub-arrays. Recall that the complexity of the standard sequential Mergesort algorithm is $O(n \log n)$.

(a) (5 points) Analyse the span of parallel `mergesort`, in which the two recursive calls are done in parallel, and merging the sub-arrays is performed sequentially in $O(n)$ time.

Solution: Because the recursive calls are done in parallel they only count once, although the merging is still linear. Then:

$$S(n) = S(n/2) + O(n) = O(n)$$

where using the Master theorem $a = 1$, $b = 2$, and $f(n) = n^1$.

(b) (5 points) Is the algorithm from part (a) efficient and is it optimal? Explain your response.

Solution: The work is:

$$W(n) = 2W(n/2) + \Theta(n) = \Theta(n \log n)$$

using $a = 2$, $b = 2$, and $f(n) = \Theta(n)$ in the Master theorem.
The work is equal to the sequential time ($\Theta(n \log n)$), which is optimal for comparison-based sort, so the overhead is constant. Unfortunately the span is not poly-logarithmic, so the algorithm is not optimal (poly-logarithmic span and constant overhead) nor even efficient (poly-logarithmic span and poly-logarithmic overhead).

(c) (5 points) If the merging of the sub-arrays can be done in parallel in $O(\log n)$ time, what does that mean for the work and span of the `mergesort` algorithm? Using this merging technique is the `mergesort` algorithm efficient and is it optimal? Explain your response.

Solution: The span becomes:

$$S(n) = S(n/2) + O(\log n) = O(\log^2 n)$$

The work is (again) equal to the sequential complexity, so there is no overhead, but the span is now poly-logarithmic, so it is therefore both efficient and even optimal.

THERE ARE NO MORE QUESTIONS

\o/