[20210204] INFOB3CC - Concurrency - 2 - at home

Course: BETA-INFOB3CC Concurrency (INFOB3CC)

Duration:	2 hours
Number of questions:	9
Generated on:	May 24, 2021

Contents: Pages:

[20210204] INFOB3CC - Concurrency - 2 - at home

Course: Concurrency (INFOB3CC)

- The exam is an open book exam. You can consult the material from the book, the webpage or other sources.
- The exam must be made alone. No communication with others is allowed. When we suspect fraud, we may ask you for an oral exam to verify whether you made the exam on your own.
- The first question contains further information regarding the rules and regulations of examination at home. Please read these regulations and acknowledge that you have read and accept these rules.
- Provide brief and concise answers. Overly verbose reseponses or nonsense added to otherwise good answers can deduct from your grade.
- When asked to explain your choice on a multiple choice question, your reasoning should explain why your chosen answer is correct and why the others are not correct.
- 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.
- If you think that the exam contains a mistake, send a private message via Teams to Ivo Gabe de Wolff. We have a separate channel in Teams to make any announcements about the exam, if needed.
- You have two hours to complete the exam. You can go back to previous questions.
- *Good luck! (:*

Number of questions: 9

You can score a total of 40.01 points for this exam, you need 20.01 points to pass the exam.

1 Dear student,

0.01 pt.

This test takes place under special circumstances in which we, even more than usual, rely on your professionalism and integrity.
To ensure that everyone has the same expectations, we ask you to **sign the statement below by typing “agree” in the answer box**.

If you do not agree, the test is in principle invalid. This can only be undone by a decision of the Examination Board, which can be obtained by sending the committee a written statement by email.

- I take this test under my own name / logged in with my own account.
- I take this test by myself, without contact or help from others.
- I will not copy, screen dump or otherwise record or distribute questions and answers during or after the test.
- I will only use the permitted resources that are indicated in the introduction text of the test.

I know that:

- Violation of the aforementioned agreements is regarded as Fraud (see OER art 5.14).
- Answers can be checked for plagiarism.
- The results of this test are preliminary, the examiner can invite me at a later stage for an additional oral examination (for example via Skype or Teams).

Sign by typing the word AGREE in this box

.....

After saving and submitting the answer, the test will be started.

2 You are asked to store a large data set. The data consists of 32 bit numbers, in the range 0 to 4.294.967.295, but we expect that most of the numbers are smaller than 255. Hence we use the following scheme to compress this data:

- If the number is smaller than 255, we store it in a single byte (8 bits).
- If the number is equal to or larger than 255, we store 255 in the first byte and then use 4 bytes to store the 32 bit number.

Values smaller than 255 thus require only a single byte instead of 4, and larger values use 5 bytes instead of 4. If enough values are smaller than 255, we will thus have a more compact representation of the data.

In this question we will consider the conversions between the compressed (1 or 5 bytes per value) and uncompressed format (always 4 bytes per value). The uncompressed data is represented as an array / vector of 32 bit numbers. The compressed data is stored as a vector of 8 bit values.

2 pt. **a.** Given an uncompressed array, we can compute the size of the compressed format with two parallel patterns. Which two patterns are used in the most efficient way to do this? In which order?

		map	stencil	fold	scanl	scanr	permute	backpermute
		A	**B**	**C**	**D**	**E**	**F**	**G**
Step 1	**1**	○	○	○	○	○	○	○
Step 2	**2**	○	○	○	○	○	○	○

2 pt. **b.** Given an uncompressed array, we now want to construct an array with the compressed data. Which two parallel patterns, and in which order, should be used to compute for each element in the uncompressed array, the index in the compressed array?

		map	stencil	fold	scanl	scanr	permute	backpermute
		A	**B**	**C**	**D**	**E**	**F**	**G**
Step 1	**1**	○	○	○	○	○	○	○
Step 2	**2**	○	○	○	○	○	○	○

1 pt. **c.** Given the uncompressed array and the array of indices from the previous question, we can construct the compressed array with a single parallel loop. This operation is similar to a combinator from Accelerate. Which combinator is most similar to this operation?

a. map

b. stencil

c. fold

d. scanl

e. scanr

f. permute

g. backpermute

h. zip

1 pt. **d.** What is the difference with the Accelerate combinator and the operation that you need to perform here?

We will now consider the inverse operation, converting from the compressed format to the uncompressed representation.

First we need to construct an array of booleans, where each boolean of the array denotes whether a new number starts at that index in the compressed array. For instance, in the following compressed array, new numbers start at indices 0, 1 and 6, as the second value is stored at indices 2 to 5.

```
12, 255, 30, 42, 22, 10, 80
T,  T,   F,  F,  F,  F,  T
```

Note that the bytes of values larger than or equal to 255 could also be 255, so a value of 255 in the compressed format does not have to imply the start of a large value.

1 pt. **e.** Can a stencil be used to construct this boolean array?

a. No

b. Yes

3 pt. **f.** This boolean array can be constructed using a scan. We must consider 5 different states, such that each byte has exactly one such state:

0. The start of an element (either a small number or 255 to denote the start of a large number)
1. The first byte of a large number
2. The second byte of a large number
3. The third byte of a large number
4. The fourth byte of a large number

In a sequential implementation, we could scan over the compressed array and assign each value a single state. This is not possible in a parallel implementation, instead we must store a mapping of states.

Explain how this boolean array can be constructed using a parallel scan.

You may use a helper function # for indexing a homogenous tuple, e.g. *(a,b,c,d,e) # 0* equals *a*.

3 Gordon has developed a sequential algorithm to analyse his x-ray crystallography images which runs in $\Theta(n^2)$ time. Cornelius has the idea for a new algorithm which makes use of the periodic structure of the crystal lattice so that the algorithm can be parallelised to run in $\Theta(n \log n)$ steps, at the expense of increasing the work to $\Theta(n^3)$.

1 pt. **a.** Since the labratory has already spent most of its money on the x-ray device, Gordon thinks that they will not be able to afford the processors necessary to make the new algorithm worthwhile. How many processors will be required for Cornelius' algorithm to be worthwhile (asymptotically) over Gordon's algorithm? Motivate your response.

1 pt. **b.** What is the maximum number of processors that the laboratory should purchase for Cornelius' algorithm (asymptotically)? Motivate your response.

4 Consider the following functions:

```
f (I1 index) = I1 $ index * 2
g (I1 index) = I1 $ index + 256
```

We want to perform a backpermute on a GPU using those functions:

```
ys = backpermute (I1 10000) f xs
zs = backpermute (I1 10000) g xs
```

You may assume that all the indices are in bounds.

1 pt. **a.** We want to perform the backpermutes on a GPU. Which of the backpermutes would be faster?

- **a.** The backpermute using *f* is faster than *g*.
- **b.** The backpermute using *g* is faster than *f*.
- **c.** The backpermutes would perform the same.

In an attempt to improve performance, John wants to use *permute* instead of *backpermute*. To use *permute*, he writes the inverse function of *f* and *g*:

```
fInv (I1 index) = if index `mod` 2 == 0 then Just_ (index `div` 2) else
Nothing_
gInv (I1 index) = if index >= 256 then Just_ (index - 256) else Nothing_
```

1 pt. **b.** Which option gives the best performance for functions *f* and *fInv*?

- **a.** permute using fInv
- **b.** backpermute using f

1 pt. **c.** Which option gives the best performance for functions *g* and *gInv*?

- **a.** permute using gInv
- **b.** backpermute using g

5 In this question, we investigate the spreading of a virus. We have a large data set of the number of infected people each day over a large time span, and want to use data parallelism to analyze this data.

We are interested in the spreading factor of the virus. We define the spreading factor at day *i* as the ratio between the number of infections at day *i* and the numbef or infections at day *i+1*:

```
spreading_factor = infections[i+1] / infections[i]
```

The data is stored as floating point numbres. We assume that there are infected persons on all the days of the range of our data, e.g. we are not analyzing the spreading in New Zealand.

2 pt. **a.** Explain how this can be implemented using a *stencil* and a *fold*.

You don't have to consider the edge cases at the boundary of the array.

Helga says that this can be implemented without a stencil, using only a single fold and a map. The fold operates on tuples of a number of infections and an accumulator of the maximum spreading factor. The map converts each element into that form, by coupling it with the factor 0. In the fold, she propagates the previous value and the maximum factor.

```
f (previousInfections, factor1) (currentInfections, factor2) =
(currentInfections, factor1 `max` factor2 `max` currentInfections /
previousInfections)
```

3 pt. **b.** Helga already has a sequential version of this working. Is it also possible to evaluate this in parallel? Explain why this is, or is not, possible.

You don't have to consider the edge cases at the boundary of the array.

Due to data corruption, we accidentally lost the array with the infection counts. Luckily, we still know the number of infected persons at the start of the data range and the array of the spreading factors at all positions.

2 pt. **c.** Explain how the array containing the number of infections each day can be reconstructed efficiently with data parallelism.

You may ignore any issues with rounding errors.

1 pt. **d.** What span does this reconstruction algorithm have?

a. O(1)

b. O(log n)

c. O(n)

d. O(n log n)

e. O(n^2)

6 Given an association array of (key,value) pairs of type *Vector (key, val)*, which is sorted on the keys, we can implement a function:

```
lookup :: Exp key -> Acc (Vector (key, val)) -> Exp (Maybe val)
```

which uses binary search to return the corresponding value of a given key (if it exists) in O(log n) steps.

1 pt. **a.** Amelia has an array of keys of size m for which they would like to look up every corresponding value from the association array of size n. Analyse the work and span of this operation. Motivate your response.

2 pt. **b.** Give a data-parallel algorithm to compute the difference of two sorted association arrays, both of size n. That is, the operation returns only those elements of the first set whose keys do not also appear in the second.

You are not required to write code with perfect syntax, but you must clearly denote the (data-parallel) steps of your approach. You may use any data-parallel patterns from the lectures or functions from Accelerate.

2 pt. **c.** Analyse the aymptotic work and span of your method from the previous question. Motivate your response.

1 pt. **d.** The best sequential algorithm for computing the difference of two sorted arrays, both of size n, requires O(n) work. Is your algorithm efficient and/or optimal? Motivate your response.

7 A scan is more general than a fold. Instead of doing a fold, you can also perform a scan and read the last value of the output to get the result of the fold. This question regards the differences between those two approaches.

1 pt. **a.** Which statement is true about the combination function or operator passed to fold or scan?

- **a.** A scan puts more requirements on the operator than a fold.
- **b.** A fold puts more requirements on the operator than a scan.
- **c.** A scan and fold put the same requirements on the operator.

1 pt. **b.** Can folds and scans be executed efficiently in parallel?

- **a.** A scan cannot be executed in parallel with an efficient algorithm, but a fold can.
- **b.** A fold cannot be executed in parallel with an efficient algorithm, but a scan can.
- **c.** A fold and scan can both be implemented efficiently in parallel.

1 pt. **c.** What is the difference of the span of folds and scans?

- **a.** A fold has asymptotically a higher span than a scan.
- **b.** A scan has asymptotically a higher span than a fold.
- **c.** The span of a fold is only a constant factor larger than the span of a scan.
- **d.** The span of a scan is only a constant factor larger than the span of a fold.

8 Elliot is desiging his own language with data parallelism. He wants to have a multi-backpermute combinator, in which an element of the resulting array does not depend on a single element of the source array, as with a normal backpermute, but can depend on multiple elements of the source array. The multi-backpermute also takes an associative function to combine multiple elements.

The multi-backpermute combinator could for instance be used to, given a triangular matrix, compute per row the sum of all values on that row. For each row we would then take all indices of elements on that row of the triangular matrix.

Elliot wants to execute the multi-backpermute on the GPU. He discovers that the best way to execute this depends on the average number of values from the source array that each element from the output needs, and the variance of that.

2 pt. **a.** Elliot's first idea was to start one thread for each element of the output array and within each thread read all requested values from the input array in a loop.

For which cases would this option work best? Explain why this works efficiently for those cases and why it performs bad for other cases.

1 pt. **b.** What execution strategy could be used to make this perform faster in the cases where Elliot's approach would perform badly?

9 Consider the following algorithm, which performs two recursive calls, a parallel scan over the input and another two recursive calls. The recursive calls operate on arrays of 1/4th the size:

```
procedure undertow(xs)
  n = length(xs)

  if (n == 0) return 0
  if (n == 1) return xs[0]

  a = undertow(xs[0 .. n / 4])
  b = undertow(xs[n * 3 / 4 .. n])

  ys = scanl(\u v -> ... a ... b ... u ... v ..., 0, xs)

  return max(
    undertow(ys[n / 4 .. n / 2]),
    undertow(ys[n / 2 .. n * 3 / 4])
  )
```

The two recursive calls on *xs* can be performed in parallel. The recursive calls on *ys* can also be executed in parallel with each other, but the calls on *ys* can only start when both calls on *xs* have finished.

2 pt. **a.** What is the aymptotic span of this algorithm? Show how you computed the span.

2 pt. **b.** What is the aymptotic work of this algorithm? Show how you computed the work.

Name:			Signature:
Date:	/ /	Birth date:	/ /

Course: BETA-INFOB3CC Concurrency (INFOB3CC) - Questions: [20210204] INFOB3CC - Concurrency - 2 - at home

- The exam is an open book exam. You can consult the material from the book, the webpage or other sources.
- The exam must be made alone. No communication with others is allowed. When we suspect fraud, we may ask you for an oral exam to verify whether you made the exam on your own.
- The first question contains further information regarding the rules and regulations of examination at home. Please read these regulations and acknowledge that you have read and accept these rules.
- Provide brief and concise answers. Overly verbose reseponses or nonsense added to otherwise good answers can deduct from your grade.
- When asked to explain your choice on a multiple choice question, your reasoning should explain why your chosen answer is correct and why the others are not correct.
- 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.
- If you think that the exam contains a mistake, send a private message via Teams to Ivo Gabe de Wolff. We have a separate channel in Teams to make any announcements about the exam, if needed.
- You have two hours to complete the exam. You can go back to previous questions.
- *Good luck! (:*

1 0.01 pt.

Write your answer on the dots:

.....

2 11 pt.

a.

	A	B	C	D	E	F	G	H
1:	○	○	○	○	○	○	○	○
2:	○	○	○	○	○	○	○	○

Choose one option per row

b.

	A	B	C	D	E	F	G	H
1:	○	○	○	○	○	○	○	○
2:	○	○	○	○	○	○	○	○

Choose one option per row

c.

A ○ B ○ C ○ D ○ E ○ F ○ G ○ H ○

d.

Answer:

2

4

6

8

e.

A ○ B ○

f. Answer:

3
2 pt.

a.

Answer:

b.

Answer:

4
3 pt.

a. A ○ B ○ C ○

b. A ○ B ○

c. A ○ B ○

5
8 pt.

a.

Answer:

b.

Answer:

c.

Answer:

	A	B	C	D	E
d.	○	○	○	○	○

6
6 pt.

a.

Answer:

b.

Answer:

c. Answer:

d. Answer:

7
3 pt.

a. A B C

b. A B C

c. A B C D

8
3 pt.

a.

Answer:

b.

Answer:

9
4 pt.

a.

Answer:

b.

Answer:

Correction model

1. 0.01 pt. agree
0.01 pt.

2. 11 pt.

a.
1. 1 pt. A
2. 1 pt. C

b.
1. 1 pt. A
2. 1 pt. D

c. 1 pt. F

d.

Correction criterion	Points
The value should be written to multiple indices (after splitting the value to multiple bytes).	1 point
Total points:	*1 point*

e. 1 pt. A

f.

Correction criterion	Points
Use 5-tuples to denote the state transitions of each element. Each field of the tuple denotes the index of the next state.	1 point
Map the compressed array of bytes to 5 tuples using: \x -> (if x == 255 then 1 else 0, 2, 3, 4, 0)	0.5 point
Use a left-to-right scan (scanl) to compute the resulting state at each position.	0.5 point
Operator of the scan is: \(a,b,c,d,e) x -> (x # a, x # b, x # c, x # d, x # e)	0.5 point
Construct the requested array by mapping over the result of scan with: \(a, _, _, _, _) -> a == 0	0.5 point
Total points:	*3 points*

3. 2 pt.

a.

Correction criterion	Points
The break even point is at P = Wc/Sg = n^3/n^2 = n. Less than this number of processors the algorithms are work bound and the sequential algorithm is faster.	1 point
Total points:	*1 point*

b.

Correction criterion	Points
The transition between the work bound and span bound phase is P = work / span = n^3 / n log n.	1 point
Total points:	*1 point*

4. 3 pt.

a. 1 pt. B

b. 1 pt. B

c. 1 pt. B

5. 8 pt.

a.

Correction criterion	Points
Stencil computes a vector of spreading factors	0.5 point
The stencil computation gets the invections of the current and next day and computes the ratio	0.5 point
The fold takes the output of stencil and computes the maximum value	0.5 point
The fold uses operator 'max' and initial value 0, or is a fold1.	0.5 point
Total points:	*2 points*

b.

Correction criterion	Points
This is not possible	1 point
The operator of a parallel fold should be associative, but this operator isn't associative. OR: The argument 'previousInvections' is not always the number of invections of the previous dat, because of a parallel/tree reduction.	2 points
Total points:	*3 points*

c.

Correction criterion	Points
Scan / prefix sum / postfix sum	1 point
The scan computes for each position the number of infected people, by multiplying the infections at the first day and all spreading factors before that day	1 point
Total points:	*2 points*

d. 1 pt. B

6. 6 pt.

a.

Correction criterion	Points
For each key in the array of size m, we must perform log n work. W(n) = O(m log n)	0.5 point
All of the m tasks can be performed in parallel. S(n) = O(log n)	0.5 point
Total points:	*1 point*

b.

Correction criterion	Points
Determine which elements from the first set exist in the second. These elements need to be discarded, the others we keep.	1 point
Filter/compact the array to only those elements which we need to keep from the first step.	1 point
Total points:	*2 points*

c.

Correction criterion	Points
Work: for each item in the first array, perform logarithmic work: O(n log n) Then filter this result in O(n) (filter uses scan) Total O(n log n)	1 point
Span: all lookups happen in parallel: O(log n) Filter has span O(log n) Total: O(log n)	1 point
Total points:	*2 points*

d.

Correction criterion	Points
Overhead: work of parallel / work of sequential = n log n / n = log n	0.5 point
Efficient: overhead is poly-logarithmic Optimal: overhead is constant So, the algorithm is efficient but not optimal	0.5 point
Total points:	*1 point*

7. 3 pt.

a. 1 pt. C

b. 1 pt. C

c. 1 pt. D

8.
3 pt.

a.

Correction criterion	Points
This would work when one element depends on a small set of variables, or when the difference/variance in those sizes is small.	1 point
Low variance implies low warp divergence	0.5 point
When a large number of source elements has to be read, a long sequential computation is required.	0.5 point
Total points:	*2 points*

b.

Correction criterion	Points
Parallel fold / tree reduction	1 point
Total points:	*1 point*

9.
4 pt.

a.

Correction criterion	Points
f(n) = log(n)	0.5 point
T(n) = 2T(n/4) + log(n)	0.5 point
Case 1 OR: Recursive calls dominate in execution time	0.5 point
Span is T(n) = O(n^0.5) = O(sqrt(n))	0.5 point
Total points:	*2 points*

b.

Correction criterion	Points
f(n) = n	0.5 point
T(n) = 4T(n/4) + n	0.5 point
Case 2 OR: Recursive calls and function f have the same contribution	0.5 point
Span is T(n) = O(n log n)	0.5 point
Total points:	*2 points*

Caesura

Points scored	Grade
40	10
39	9.8
38	9.6
37	9.3
36	9.1
35	8.9
34	8.7
33	8.4
32	8.2
31	8.0
30	7.8
29	7.5
28	7.3
27	7.1
26	6.9
25	6.6
24	6.4
23	6.2
22	6.0
21	5.7
20	5.5
19	5.3
18	5.1
17	4.8
16	4.6
15	4.4
14	4.2
13	3.9
12	3.7
11	3.5
10	3.3
9	3.0
8	2.8
7	2.6
6	2.4
5	2.1
4	1.9
3	1.7
2	1.5
1	1.2
0	1.0