[20210422] INFOB3CC - Concurrency - 3 - at home

Course: BETA-INFOB3CC Concurrency (INFOB3CC)

Duration:	2 hours
Number of questions:	7
Generated on:	May 24, 2021

Contents: Pages:

[20210422] INFOB3CC - Concurrency - 3 - 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: 7

You can score a total of 27 points for this exam, you need 13.5 points to pass the exam.

1 Ayla is creating an app *Doggy Dating* to connect dog owners so that they can meet with other dogs for a play date. Users of the app see the profile photo and description of doggos in their area, and can select dogs which they think will be a good match for their dog (in size, temprament, etc.). If both owners select each other, then there is a match, and the app will put the owners in contact so that they can chat to organise a play date with their dogs.

We have the following data type to record information about each user (doggo):

```
data Doggo = Doggo
  { name          :: String
  , age           :: Int
  , description   :: String
    -- other information...
  , wants_to_meet :: [Doggo]
  , matched_with  :: [Doggo]
  }
```

1 pt. **a.** Because many dog owners can be using the app at once, it is important to make sure that the user database can be safely accessed and modified by many different users (threads) at once. This can be solved by placing a single lock around the entire database so that only one thread can modify the database entries at once, but you think that will not be a good solution. Explain why.

2 pt. **b.** You propose instead to put locks only over the individual fields wants_to_meet and matched_with (so for example, the data type changes such that the field wants_to_meet :: MVar [Doggo]) as these are the only fields which need to be updated as owners decide that they want to meet.

You know that it is possible to create a deadlock when there are more than two locks in the program. Can this problem occur with the proposed change to wants_to_meet and matched_with? If not explain why, otherwise give an example of how this problem can occur and explain how it can be prevented.

2 3 pt. Figaro is writing a multi-threaded web server. Each client of the web server is handled by a separate thread. Since there can be many more client threads than there are CPU cores, in order to prevent oversubscription that would slow the system down, Figaro knows that the client threads should not do the work directly but should instead submit their work to a queue, to be processed in order by a dedicated worker thread.

When the worker thread finishes processing a request, it checks the queue for the next item. If there are no waiting items in the queue, the worker thread goes to sleep. When a client thread has a new request to be processed, it first checks whether the worker thread is active or sleeping. If the worker is sleeping it wakes up the worker thread to have the task processed, otherwise it adds its item to the queue. If the queue is full, the thread should try to submit the work item to a different worker.

He comes up with the following implementation:

```
data Queue = Queue (MVar ()) (IORef [IO ()])

_MAX_SIZE :: Int
_MAX_SIZE = 128

newQueue :: IO Queue
newQueue = do
  waiting <- newEmptyMVar
  queue   <- newIORef []
  return $ Queue waiting queue

worker :: Queue -> IO ()
worker (Queue waiting queue) = loop
  where
    loop = do
      mwork <- atomicModifyIORef queue $ \items ->
                 case items of
                   []   -> ([], Nothing)         -- empty queue
                   q:qs -> (qs, Just q)          -- get the head of the
queue
      case mwork of
        Nothing   -> takeMVar waiting            -- sleep waiting for new
work
        Just work -> work                        -- execute the task
      loop                                       -- look for more work

tryAddWork :: Queue -> IO () -> IO Bool
tryAddWork (Queue waiting queue) work = do
  sleeping <- isEmptyMVar waiting
  if sleeping
     then do writeIORef queue [work]             -- make a queue of the
single element
             putMVar waiting ()                  -- wake up the worker
thread
             return True                         -- successfully added to
```

```
the queue

     else atomicModifyIORef queue $ \items ->
            if length items < _MAX_SIZE
              then (items ++ [work], True)      -- successfully added to
the queue
              else (items,           False)     -- queue is full!
```

Figaro has tested the queue and says that it seems to work, but is not sure. Consider the implementation and explain why the queue works as in the description, or give example scenario(s) of where the queue can go wrong.

3 Newton's method can iteratively approximate a root (a point x such that $f(x) = 0$) of a function f . This can be used to approximate the inverse of a function g , if functions g and its derivative can be easily computed.

For instance, if we want to compute the square root $\sqrt{a}$, then we can use Newton's method to approximate this by searching the root of $f(x) = x^2 - a$. Newton's method says that if x_i is an approximation of the root, then

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{x_i^2 - a}{2x_i}$$

is a better approximation of the root. By iterating this procedure we can end up close enough to the exact solution. The number of iterations depends on the quality of the initial approximation x_0 .

2 pt. **a.** We need to compute the square root of a large list of numbers, on a GPU which cannot compute square roots natively. Explain why this can be executed in parallel.

2 pt. **b.** Explain what performance problem you may get when executing this on a GPU with lockstep execution.

2 pt. **c.** Newton's method should start with an initial approximation. We could for instance start with a $x_0 = 1$ or $x_0 = a$, which would work good when a is close to 1. Large values of a would require more iterations.

We could spend more time in finding a good initial approximation, such that we would require fewer iterations for Newton's method. Explain why this trade off is different on a GPU with lockstep than on a CPU (which also cannot compute square roots natively).

4 Mark has developed a sequential algorithm to analyse satelite images which runs in $\Theta(n)$ time. Peter has the idea for a new algorithm which can be parallelised to run in $\Theta(\log^2 n)$ steps, at the expense of increasing the work to $\Theta(n^2)$.

Note that we write $\log^2 n$ to denote $(\log n)^2$.

1 pt. **a.** Since the company has already spent most of its money on the satelites, Mark 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 Peter's algorithm to be worthwhile (asymptotically) over Mark's algorithm? Motivate your response.

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

5 Consider the following algorithm, which performs four recursive calls of arrays of half the size, a parallel map and a parallel scan:

```
procedure tierra(ps)
  n = length(ps)

  if n == 0 { return 1 }

  a = tierra(ps[0 .. n * 0.5])
  b = tierra(ps[n * 0.5 .. n])

  qs = map(\x -> x + a + b, ps)
  rs = scan((+), qs)

  c = tierra(rs[0 .. n * 0.5])
  d = tierra(rs[n * 0.5 .. n])

  return c*d
```

The two recursive calls before the scan are executed in parallel, then the parallel map is executed, then parallel scan is executed and finally the last two recursive calls are executed in parallel. Note that the first two recursive calls cannot happen in parallel with the last two, as the last two can only start when the scan, map and the first two calls 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.

6 In STM, operations that use or modify TVars are written as transactions in *atomically* blocks. In this question we consider some operation which affects many TVars.

1 pt. **a.** Some operations can either be executed in one large transaction, or multiple short transactions. What option is in general faster?

- **a.** Multiple transactions (multiple calls to *atomically*)
- **b.** One large transaction (one call to *atomically*)

2 pt. **b.** Give two reasons why this is generally faster.

2 pt. **c.** It might not always be possible to split a large operation into multiple smaller transactions. Explain why this may not be possible.

7 Consider we want to perform a gather, known as a backpermute in Accelerate, on a GPU. Given some $k \geq 1$, we perform the index transformation $f(i) = k * i + 7$.

2 pt. **a.** Explain why this operation performs faster if $k = 1$, compared to other values for k .

2 pt. **b.** A backpermute can sometimes be written as a permute, potentially with a different index transformation. Which property or properties <u>must</u> hold on the index transform of the *backpermute*, for this to be possible?

Select all the necessary properties.

- **a.** The index transformation should be injective.
- **b.** The index transformation should be surjective.
- **c.** The index transformation should be its own inverse.
- **d.** The index transformation should be a polynomial.

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

Course: BETA-INFOB3CC Concurrency (INFOB3CC) - Questions: [20210422] INFOB3CC - Concurrency - 3 - 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
3 pt.

a.

Answer:

b.

Answer:

2
3 pt.

Answer:

3
6 pt.

a. Answer:

b. Answer:

c. Answer:

4
2 pt.

a. Answer:

b. Answer:

5
4 pt.

a. Answer:

b. Answer:

6
5 pt.

a. A ○ B ○

b. Answer:

c. Answer:

7
4 pt.

a. Answer:

b. A ☐ B ☐ C ☐ D ☐

There are multiple answers possible

Correction model

1.
3 pt.

a.

Correction criterion	**Points**
The program will be sequentialised	1 point
Total points:	*1 point*

b.

Correction criterion	**Points**
The problem can occur when two users A and B each try to match with the other at the same time.	1 point
Threads must always take locks in a specific order (e.g. by increasing user ID)	1 point
Total points:	*2 points*

2.
3 pt.

Correction criterion	**Points**
The queue does not work	1 point
After checking if the worker is not sleeping, the worker could finish its task, check the queue is empty, and then go to sleep. The worker adds an item to the queue but does not wake the worker.	1 point
Two (or more) threads can at the same time check the worker is asleep. They each write a single item to the queue, so all but one of the items are lost.	1 point
Total points:	*3 points*

3.
6 pt.

a.

Correction criterion	**Points**
Each number can be computed independently / Data parallelism: map pattern / Embarrassingly parallel	2 points
Total points:	*2 points*

b.

Correction criterion	**Points**
Some numbers may need much more iterations, causing warp divergence	2 points
Total points:	*2 points*

c.

Correction criterion	**Points**
A better approximation would reduce the variance in the number of iterations	1 point
On a GPU it may help to spend more time on an initial approximation to reduce warp divergence, whereas we may want to start the iteration directly on the CPU which doesn't have warps.	1 point
Total points:	*2 points*

4.
2 pt.

a.

Correction criterion	**Points**
The break even point is at P = Wp/Sm = n^2/n = 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**
Mark has a sequential algorithm so only one processor can be used.	1 point
Total points:	*1 point*

5.
4 pt.

a.

Correction criterion	**Points**
f(n) = log(n)	0.5 point
T(n) = 2T(n/2) + log(n)	0.5 point
Case 1 OR: recursive calls dominate over f	0.5 point
Span is T(n) = O(n)	0.5 point
Total points:	*2 points*

b.

Correction criterion	**Points**
f(n) = n	0.5 point
T(n) = 4T(n/2) + n	0.5 point
Case 1 OR: recursive calls dominate over f	0.5 point
Span is T(n) = O(n^2)	0.5 point
Total points:	*2 points*

6.
5 pt.

a. 1 pt. A

b.

Correction criterion	**Points**
Smaller transactions are less likely to restart after a changed variable	1 point
When we do have to restart the transaction, we have to throw away less work.	1 point
Total points:	*2 points*

c.

Correction criterion	**Points**
When splitting the operation into smaller transaction, other threads may observe or modify the state between two transactions	2 points
Total points:	*2 points*

7.
4 pt.

a.

Correction criterion	**Points**
Memory coalescing / Consecutive threads read consecutive fields / Instead of multiple small read operations, a warp can perform one large read	2 points
Total points:	*2 points*

b. 1 pt. A

B

C

D

Bonus: 1 pt.

Caesura

Points scored	**Grade**
27	10
26	9.7
25	9.3
24	9.0
23	8.7
22	8.3
21	8.0
20	7.7
19	7.3
18	7.0
17	6.7
16	6.3
15	6.0
14	5.7
13	5.3
12	5.0
11	4.7
10	4.3
9	4.0
8	3.7
7	3.3
6	3.0
5	2.7
4	2.3
3	2.0
2	1.7
1	1.3
0	1.0