NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

CS3211—Parallel and Concurrent Programming

Semester 2: 2021/2022

Time Allowed: 2 hours

Instructions to students:

1. This assessment paper contains FOUR questions and comprises FOURTEEN printed pages. Additionally, the there is an appendix comprising TWO pages.
2. All questions must be answered in the space provided; no extra sheets will be accepted as answers.
3. This is an OPEN BOOK assessment.
4. You are allowed to use pencils. Calculators are not allowed.
5. Using ink or pencil, write down your student number twice: firstly, above the line and secondly, shade the corresponding circle completely in the grid for each digit or letter. DO NOT WRITE YOUR NAME!

STUDENT NO:

STUDENT NUMBER

A								
U ○	0	0	0	0	0	0	0	A N
A ●	1	1	1	1	1	1	1	B R
HT ○	2	2	2	2	2	2	2	E U
NT ○	3	3	3	3	3	3	3	H W
	4	4	4	4	4	4	4	J X
	5	5	5	5	5	5	5	L Y
	6	6	6	6	6	6	6	M
	7	7	7	7	7	7	7	
	8	8	8	8	8	8	8	
	9	9	9	9	9	9	9	

Q1. [26 marks] General Questions

a. [4 marks] Relaxed consistency models at the programming language level address the concern that software complexity is increased by the need for unnecessary synchronization.
Is this statement true or false? Justify your answer.

Answer:

b. [4 marks] The modern C++ memory model exhibits a different behavior when it is running on a weak-consistent architecture (such as ARM). Is this statement true or false? Justify your answer.

Answer:

c. [4 marks] Programs using C++ atomics might suffer of deadlocks. Assume that the program does not use any other synchronization mechanism (such as mutex or semaphores). Is this statement true or false? Justify your answer.

Answer:

d. [4 marks] Explain one advantage and one disadvantage of using channels (message passing) instead of synchronization primitives and shared memory for a program running on one computer. You may refer to Go channels in your explanation.

Answer:

e. [4 marks] In (safe) Rust, data races are not possible. This means that your Rust code will not have race conditions. Is this statement true or false? Justify your answer.

Answer:

f. [6 marks] In this world of multiple cores, cloud computing, web scale, and problems that may or may not be parallelizable, we find the modern developer a bit overwhelmed. In 2005, Herb Sutter authored an article titled “The free lunch is over: A fundamental turn toward concurrency in software”. Toward the end, Sutter states: “We desperately need a higher-level programming model for concurrency than languages offer today.”

Based on the concepts studied in CS3211, answer and justify your answer for the following two questions:

- What is one of benefits that use of concurrency brings nowadays? (2 marks)
- Do you think that the models for concurrency as implemented by various programming languages have progressed enough since 2005 to enable the modern developer to take advantage benefits of concurrency? (4 marks)

Answer:

Q2. [10 marks] C++ Memory Model

a. [5 marks] Consider 2 threads in C++ pseudo-code. There are 2 `std::atomic` variables, `x` and `y` initialized to `0` before the threads have been created.

Thread 1	**Thread 2**

`x.store(1, memory_order_release);`
`y.store(2, memory_order_relaxed);`
	`while (y.load(memory_order_relaxed)!=2);`
	`cout << x.load(memory_order_acquire);`

Are the following statements TRUE or FALSE? Optionally, you may justify your answers.

Answer:

Statement	True/False
1) The code will always print 1.	
2) The code might run into an infinite loop.	
3) The code will never print 0.	
4) The code will print 1 or a value that is assigned to `x` after 1.	
5) The code will print 0, 1, or a value that is assigned to `x` after 1.	

Justification (optional):

a. [5 marks] Consider 3 threads in C++ pseudo-code. There are 3 `std::atomic` variables, `x`, `y`, and `z` initialized to `0` before the threads have been created.

Thread 1	**Thread 2**

`x.store(3, memory_order_relaxed);`
`z.store(4, memory_order_release);`
`x.store(5, memory_order_relaxed);`
	`while (y.load(memory_order_acquire)!=2);`
	`while (z.load(memory_order_acquire)!=4);`
	`cout << x.load(memory_order_relaxed);`

Thread 3

```
while (z.load(memory_order_acquire)!=4);
x.store(1, memory_order_relaxed);
y.store(2, memory_order_release);
```

Are the following statements TRUE or FALSE? Optionally, you may justify your answers.

Statement	True/False
1) The code will print 1, 5, or a value that is assigned to `x` after 1 or 5.	
2) The code will never print 1.	
3) The code will never print 3.	
4) The code will never print 5.	
5) The code might run into an infinite loop.	

Justification (optional):

Q3. [32 marks] Concurrent Programming Paradigms

In answering points a.-e. use the problem description of the **N-body simulation from Appendix.**

a. [6 marks] Identify at least **two** parts of the **N-body simulation** presented in the Appendix (page 15) that could be implemented concurrently.
   Focus on finding parts that would benefit from a concurrent implementation. For each part that you would implement concurrently, briefly explain:
   - What is the benefit, if any, of this new concurrent implementation?
   - How would you quantify the potential benefits of these new implementations?
   - What are the potential disadvantages, if any, of the new implementation?

Answer:

b. [8 marks] You are required to implement concurrently, in **modern C++**, the N-body simulation presented in the Appendix (page 15). Your new implementation should aim to **bring the highest benefits from concurrency**.

Sketch your implementation in modern C++ and explain the following points:

- Which part(s) of the N-body simulation are you implementing concurrently? You may refer to point a. and choose one of the parts presented there.
- What shared data structures are you using (if any)?
- How do you ensure that access to these data structures would not produce race conditions or any other types of synchronization problem?
- What would be the maximum level of concurrency that you expect in your implementation? (i.e. number of threads that could potentially run in parallel)

You can use the line numbers to refer to the lines of pseudo-code in Appendix. Do not copy over the lines that stay the same. Minor mistakes in C++ syntax do not bring marks deductions.

In this answer, you are not allowed to use the quad-tree structure presented in Q4.

Answer:

c. [4 marks] Assume your concurrent implementation of the N-body simulation (Appendix) from point b. does not run as expected (it blocks, crashes, etc.). What the steps that you would take to identify the issues and fix your implementation?
   Make your answer specific to your concurrent N-body simulation. General answers will not receive full credit.

Answer:

d. [8 marks] You are required to implement concurrently, in **Go**, the N-body simulation presented in the Appendix (page 15). Your new implementation should aim to **bring the highest benefits from concurrency by using the message passing paradigm**. As such, your Go implementation should use goroutines, channels, and channel peripherals like `select`. You may use read-only shared data structures, but you are not allowed to use any utilities from `sync` package in Go (mutex, condition variable, pools are not allowed).

Sketch your implementation in Go and explain the following points:

- Which part(s) of the N-body simulation are you implementing concurrently? You may refer to point a. and choose one of the parts presented there.
- Briefly describe the jobs (tasks) done concurrently by the goroutines and their interaction using channels.
- What would be the maximum level of concurrency that you expect in your implementation? (i.e. number of tasks that could potentially run in parallel)

You can use the line numbers to refer to the lines of pseudo-code in Appendix. Do not copy over the lines that stay the same. Minor mistakes in Go syntax do not bring marks deductions.

In this answer, you are not allowed to use the quad-tree structure presented in Q4.

Answer:

e. [6 marks] You are required to design your **Rust** concurrent implementation of the N-body simulation presented in the Appendix (page 15). Your new implementation should aim to **bring the highest benefits from concurrency**.

Choose one of the concurrency programming paradigms discussed for Rust (in CS3211), sketch your implementation in Rust pseudo-code, and explain the following points:

- Which paradigm are you using in this new implementation?
- Which part(s) of the N-body simulation are you implementing concurrently? You may refer to point a. and choose one of the parts presented there.
- Briefly describe the jobs (tasks) done concurrently.
- What would be the maximum level of concurrency that you expect in your implementation? (i.e., number of threads that could potentially run in parallel)

You can use the line numbers to refer to the lines of pseudo-code in Appendix. Do not copy over the lines that stay the same.

In this answer, you are not allowed to use the quad-tree structure presented in Q4.

Answer:

Q4. [12 marks] Concurrency in Action

In answering points a.-e. use the problem description of the **N-body simulation from Appendix.** Suppose you are working on a concurrent implementation of the N-body simulation.

In this question, we improve on the efficiency of finding the number of nearby bodies to a specific body *b* (line 21) for the N-body simulation from Appendix (page 15). We create a tree structure to quickly determine which bodies are potentially in the radius of *b*. We will use a tree structure called a quad-tree.

A quad-tree is a tree data structure in which each node has exactly four children. For this N-body simulation, the 2D space is partitioned into squares, such that each leaf of the quad-tree contains at most *L* bodies. Given a list of bodies and their positions, this quad-tree is built by recursively bisecting the space containing those bodies into nodes.
Once built, a node in the quad-tree has the following information:

- `isLeaf`: shows if this node is a leaf
- `children`: array of 4 node pointers representing the children of this node
- `bodies`: list of bodies at this node

Figure 1 shows an example of quad-tree.

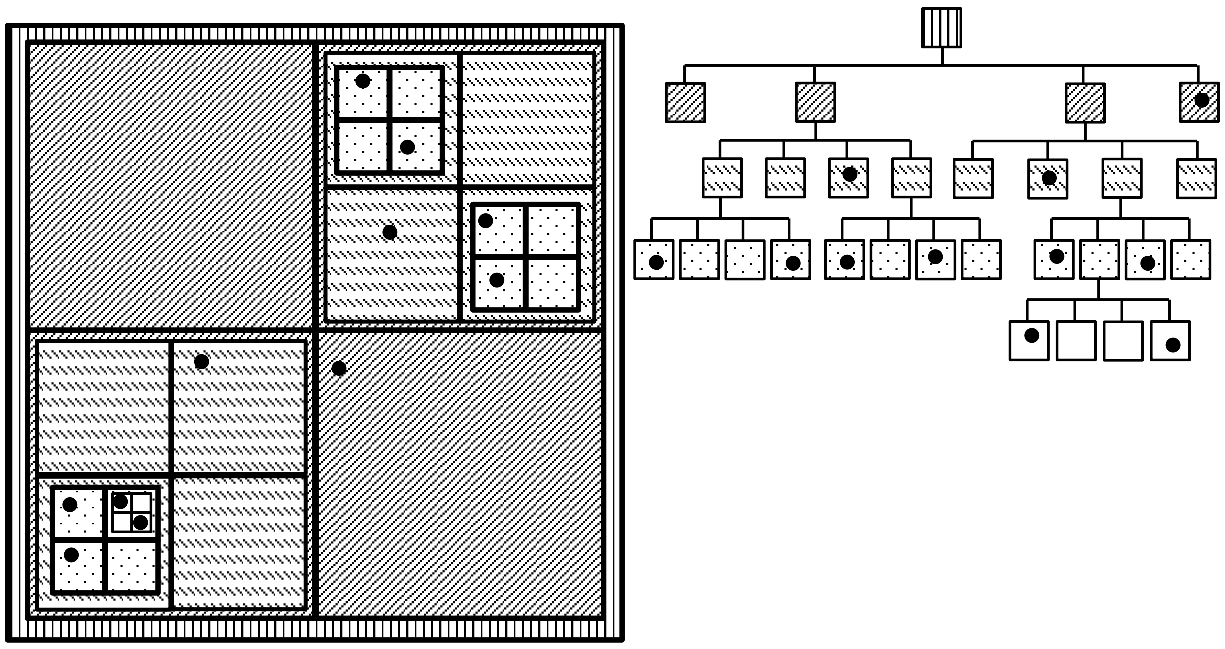

Figure 1: Quad-tree example with at most 1 body in a leaf.

a. [8 marks] Assume that you can build the quad-tree structure efficiently. Using this quad-tree, sketch a concurrent implementation for the N-body simulation from the Appendix and explain the following points:
   - Which part(s) of the N-body simulation are you implementing concurrently?
   - Briefly describe the jobs (tasks) done concurrently.
   - What would be the maximum level of concurrency that you expect in your implementation? (i.e., number of tasks that could potentially run in parallel)

Note the following:

- You may choose to present your answer in any programming language studied in CS3211 (modern C++, Go, or Rust).
- You should not focus on building the tree structure. Instead, you should use this structure for a concurrent implementation of finding nearby bodies.
- In this answer, you should not repeat the implementations presented in Q3 points b., d., or e.

Answer:

b. [4 marks] Suppose that you have a 2D square where the distribution of bodies is not uniform within the space. Explain two advantages of using this quad-tree structure in your concurrent implementation. You might compare with one of the implementations provided in Q3 under points b., d., or e.

Answer:

END of PAPER

Do not write your answers on this sheet. This sheet should not be submitted with your paper.

Appendix: Problem scenario for Questions 3 and 4

N-body simulation

You are tasked to implement a program that simulates the movement of bodies. The bodies have varying masses and initial positions. Each body is defined using an id, and it has a mass, initial velocity, and initial position. Bodies are affected by gravitational forces from nearby bodies. Assume that bodies exist in a 2D square, and they interact with other bodies using gravitational forces in the 2D space*.
The simulation is done in steps. At each step, the resulting force (or acceleration) of each body is computed by summing up the interaction forces with all other bodies.

Assume that the bodies interact only within a specified radius. Consider the following sequential implementation of the **N-body simulation**.

```
Line#  Algorithm sketch
1      find_nearby_bodies(Body b):
2          nearby_bodies = list()
3          for each attractor in all_bodies:
4              if dist(b, attractor) < radius:
5                  nearby_bodies.add(attractor)
6          return nearby_bodies
7
8      computeForce(Body target, Body attractor,
9                   float radius):
10         dist = dist(target, attractor)
11         dir = direction_from_target_to_attractor_body()
12         force = dir * target mass * attractor mass *
13                 (G / (dist * dist));
14         return force
15
16     Main simulation:
17     for each step:
18         for each Body b in all_bodies:
19             // find nearby bodies whose distance to this
20             //  body is less than 'radius'
21             nearby_bodies = find_nearby_bodies(b)
22             // compute the sum of gravitational forces
23             //  that need to be applied to this body
24             force = 0;
25             for each attractor in nearby_bodies:
26                 force += computeForce(b, attractor);
27             // update the position and velocity of
28             //    body b using the computed force
29             update(b)
```

Figure 2: Sequential implementation of N-body simulation

*Note (It is not necessary to use this note in solving the paper):
The gravitational force is computed as follows: $F_{ij} = Gm_i \cdot \frac{m_j(q_j - q_i)}{\|q_j - q_i\|^3}$
The total force F_i on body *i*, due to its interactions with the other N - 1 bodies, is obtained by summing all interactions: $F_i = \sum_{\substack{j=1 \\ i \neq i}}^{j=N} F_{ij} = Gm_i \cdot \sum_{\substack{j=1 \\ i \neq i}}^{j=N} \frac{m_j d_{ij}}{\|d_{ij}\|^3}$

BLANK PAGE

Do not write your answers on this sheet. This sheet should not be submitted with your paper.