Concurrent and Distributed Systems — 2024/25

CS4: Miscellaneous

* Star denotes optional/advanced exercise.

Q1 Parallel Speed-up

No more than ten minutes.

A task consists of 11 work units, where some are dependent on the results of others.

(a) Describe a data dependency pattern where the available parallelism is as high as possible while being less than 11.

(b) Describe a pattern of data dependency where the available parallelism is unity.

(c) Describe a pattern of data dependency where speculation can reduce the expected execution time by approximately one half. How much does the speculation increase the average total energy and average power consumption? Assume each work unit uses the same energy.

Q2 Message Passing and Actors

Using notation like `c!v` to denote a blocking message send of message v on channel c and `v = ?c1 + ?c2` to denote a blocking wait for two integers to arrive on channels c1 and c2, adding their values together and storing them in local variable v,

(a) Give a sequence of message passing operations that could deadlock. Will buffer size and/or program control flow alter whether the deadlock is inevitable for your example?

(b) (*) Is message passing 'better' than shared memory concurrency? [Think of clarity of code, energy of execution and perhaps revist the CS0 question: can you think of an application or algorithm where the shared memory is not just being used for some form of 'message passing'?]

(c) (*)What is load balancing? How can a scheduller help?

(d) (*)The type-A message passing system only supports point-to-point channels with a single writing actor and a single reading actor. The type-W system enables multiple writers to write to the same channel. The type-R system enables multiple readers to read from a given channel. The type-RW system allows both multiple readers and multiple writers for a given channel. [(*)You might also consider the Milner system which enables channel identities to be passed down channels and then be used by any actor that has kept a copy of (a handle for) that channel identity.]

(*)*A computation theory question:* can each of these systems emulate each of the others? In other words, can you automatically adapt a set of actors using one scheme into each of the others?

(*) How might each of these schemes enable the system scheduller or other part of the run-time system to make sensible informed decisions?

Q3 Lightweight Work Stealing (*)

In a work-stealing system, some fixed number of actors/agents/workers each have a local work queue of work items (tasks). Each task is likely to generate further work items that could be freely added to any queue. [*Work stealing is not on the syllabus and not examinable*.]

(a) Why should an actor put fresh work it has generated on its own queue instead of finding a low-occupancy queue and placing the item there?

(b) (*)In a work-stealing system, is a LIFO or FIFO discipline for the task queues likely to cause less lock contention? Or is it best to use a hybrid approach?

Q4 Lock-Free Programming

(a) Your hardware architecture does not supply any atomic arithmetic operators on memory, but it does provided compare-and-swap. How can you increment a variable in memory in a thread-safe way without using locks?

(b) An interrupt routine is triggered at 50 Hz. To form a real-time clock (RTC), its handler increments the time of day stored in four integer variables denoting hour, minute, second and millisecond. Describe a situation where a reader that does not use a lock might get the wrong time. To avoid this situation, the update routine also modifies some additional memory to help readers effect an atomic read of the four fields without using locks. Sketch suitable code. (*) If you are running on an architecture that requires memory fences, where would you insert them?

(c) If transactional memory were available on your machine, how could you code the previous question, regarding the RTC (real-time clock)? Could this still be called lock-free programming?

(d) (*)Perhaps re-using your answer from the C workshop, give the code for insert into a mutable binary search tree where the operations are entirely lock-free. You may assume that compare-and-swap is provided to you as a C function
`int cas<T>(T *address, T old_value, T new_value)` which returns non-zero if the old_value was stored at that address and has now been successfully replaced with the new

Note the mutable binary search tree update function returns no result, but takes the address of the tree root variable as one of its arguments:
`void tree_insert(Tree **root, k_t key, v_t value)`

(e) What is the signature for the insert function for an immutable binary search tree? Can this be implemented with lock-free programming? Or is this a stupid question?

(f) For discussion. If you have used git or a similar revision control system, discuss whether it

uses transactions and/or OCC.