COMP 35112

Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Chip Multiprocessors

Date: Friday 17th May 2019

Time: 14:00 - 16:00

Please answer all THREE Questions
Each question is worth 20 marks

This is a CLOSED book examination

The use of electronic calculators is permitted provided they
are not programmable and do not store text

[PTO]

Note: Where a question asks for instruction-level code, a format similar to ARM assembler is expected. However, marks will not be lost if the format is incorrect, as long as the meaning of each instruction is clear (from an accompanying explanation and/or comments).

Question 1.

a) Explain the key distinctions between parallel programming models that are based on **data-sharing** and those based on **message-passing**.

(2 marks)

b) The recent advent of GP-GPU hardware architectures has led to a **hybrid** style of parallel programming model in which elements of both data-sharing and message-passing are evident. Describe the nature of the underlying architecture of GP-GPU computers and explain how this relates to the data-sharing and message-passing programming models.

(5 marks)

c) Consider the following parallel loop in C-like pseudocode with an OpenMP directive, in which `a` is a shared integer array of size `n` where `n` is very large, `isSquare` is a function that returns a non-zero `int` value if and only if its (integer) argument is a perfect square, otherwise it returns zero, and `procA` is a procedure whose execution time is independent of its argument and is significantly longer than the execution time of `isSquare`. Assume there are no data dependencies between the iterations of the loop.

```
#pragma omp parallel for shared(a) private(j)
   for (j=0; j<n; j++) {
      if isSquare(j+1) { procA(a[j]) }; }
```

Describe how the computational load per iteration varies as the loop index `j` changes from 0 to `n`-1. Explain what **dynamic** loop scheduling is, and why this distribution of the computational load across the iterations requires its use. Explain why it would be undesirable in this case to dynamically schedule only one iteration at-a-time.

(5 marks)

d) Describe the operation of a 'test-and-set' (tas) instruction and explain how it can be used to safely implement a traditional binary semaphore.

(3 marks)

e) Explain why instructions that read-modify-write a value in memory atomically (i.e. guarantee that no other instruction can access the variable while the read-modify-write is in progress) cause implementation problems in a modern RISC multicore processor.

(2 marks)

f) Give a brief explanation of what is meant by a "lock-free" data structure and, in terms of expected performance and code complexity, briefly compare Java classes implemented as lock-free data structures with equivalent classes using Java synchronization techniques (i.e. synchronized methods/blocks or Java locks).

(3 marks)

Question 2.

a) Assume a 'snooping bus' MSI cache coherence protocol in which a cache line can be in one of the three states: M (modified), S (shared), I (invalid). Indicate, with the aid of diagrams, the valid pairs of states in which the same cache lines can exist in a system composed of **two** cores. Explain why these combinations of states are allowed to be valid while all other combinations are forced to be invalid. State any assumptions you make.

(6 marks)

b) In the context of your answer to part a), explain why it is additionally helpful to distinguish the case of a cache line that is in state S in exactly one core (while every other core in the system holds that cache line in state I).

(2 marks)

c) Suppose the state described in part b) is denoted state E (exclusive); the resulting protocol is termed MESI. In a MESI system with just **two** cores, each core is executing the simple program shown below, which reads the shared variable x from memory, adds one to the value, and then writes the new value back to the same shared variable in memory. Assume that the execution is interleaved between the two cores as shown with respect to any snooping bus transactions that may occur. Time is flowing downwards.

core 1	**core 2**
ldr r1, x	
	ldr r1, x
	add r1, r1, #1
add r1, r1, #1	
str r1, x	
	str r1, x

Assuming that the value of x is not cached in either core at the start of the execution, list the cache states in each core for the cache line holding the value of x after each instruction has completed execution. Explain why these states exist.

(4 marks)

d) Supposing that the intended behaviour of the given parallel code is to allow each thread independently to add 1 to the shared variable x, answer the following.

(i) What might cause the given parallel code to add less than 2 to the original value of x?

(2 marks)

(ii) Provide a sequence of code for each thread that will ensure the intended behaviour using the special **load_linked** (ldl) and **store_conditional** (stc) instructions. Explain how ldl and stc work and, hence, how your code achieves the desired outcome.

(6 marks)

Question 3.

a) Explain what is meant by **thread level speculation** (TLS). Describe **two** ways in which threads for TLS might be generated automatically from a sequential program. Use pseudocode examples to illustrate your answer.

(3 marks)

b) Explain the role of Transactional Memory in a chip multiprocessor and describe the key steps that are required when executing a transaction.

(3 marks)

c) Define the terms **readset** and **writeset** and explain how they may be used to implement Transactional Memory.

(2 marks)

d) Explain the difference between **eager versioning** and **lazy versioning** in respect of writes to shared data during execution of a transaction. Under what circumstances would either scheme be preferred?

(3 marks)

e) Explain the difference between **eager validation** and **lazy validation** in respect of detecting conflicts when trying to commit or abort a transaction. Under what circumstances would either scheme be preferred?

(3 marks)

f) Compare and contrast, with examples where appropriate, the following aspects of **Loop-based Thread-level Speculation** (TLS) and **Transactional Memory** (TM):

(i) Differences from the programmer's perspective.

(1 mark)

(ii) Requirements for extra resources and hardware support for their efficient implementation.

(5 marks)

END OF EXAMINATION