COMP35112

Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Chip Multiprocessors

Date: Thursday 2nd June 2016

Time: 09:45 - 11:45

Please answer any THREE Questions from the FOUR Questions provided

---

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) In the context of thread-based data-sharing parallel programming in Java, briefly explain why synchronisation constructs, such as barriers, locks, semaphores and monitors, are needed by a programmer.

(3 marks)

b) Give instruction-level code for implementing a binary semaphore using conventional load (ldr) and store (str) instructions (plus compare and branch instructions) only. Use this code to explain why it is necessary to provide hardware-level support for synchronisation operations in a multicore processor.

(3 marks)

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

(3 marks)

d) 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)

e) Explain how the pair of instructions 'load linked' (ldl) and 'store conditional' (stc) can be used to implement a binary semaphore and discuss how these instructions overcome the problems identified in your answer to part d). Your answer should include an indication of the code required at instruction-level.

(4 marks)

f) The simplest form of barrier has a single shared variable initialised with a value N and has a 'wait' function that is executed by a thread wanting to synchronise at the barrier. A thread calling 'wait' will decrement the variable and if the value is now zero will exit immediately otherwise it will wait at the barrier until the value has been reduced to zero by other threads. Give instruction-level code that implements this barrier using 'load linked' and 'store conditional' instructions and explain how it works.

(5 marks)

Question 2.

a) To what does the term ‘cache coherence’ refer in the context of a shared memory multicore processor? Why is it difficult to maintain a coherent cache, especially in systems with large numbers of cores?

(3 marks)

b) Assume a MESI ‘snooping bus’ protocol in which a cache line can be in one of the four states: M (modified), E (exclusive), 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.

(7 marks)

c) In the context of the MESI cache coherence protocol, explain why it is helpful to know that a cache line is in state E, as opposed to state S.

(2 marks)

d) In a MESI system with **three** 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 amongst the three cores as shown with respect to any snooping bus transactions that may occur. Time is flowing downwards.

core 1	**core 2**	**core 3**
ldr r1, x		
add r1, r1, #1		
str r1, x		
	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 any 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.

(6 marks)

e) Following on from your answer to part d), suppose that core 2’s cache entry for x is now flushed to memory. What value would main memory hold for x after the flush? Is this value in any way ‘correct’? Explain why.

(2 marks)

[PTO]

Question 3.

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

(3 marks)

b) Hardware multicore memory architectures fall into two main classes, namely **shared memory** (in which load and store instructions access a single address space that is shared by every core) or **distributed memory** (in which each core has its own private address space which no other core can access). Both classes are Turing complete (i.e. capable of being programmed to compute any computable function) so either kind of parallel programming model from part a) can be implemented using either class of memory architecture. Explain how you would approach the implementation of each kind of parallel programming model (**data-sharing** and **message-passing**) on each class of architecture (**shared memory** and **distributed memory**), highlighting anything you expect to be difficult, and comment on the performance you would expect to achieve in each case.

(5 marks)

c) Accelerator-based hardware architectures such as GP-GPU (general purpose graphical processing unit) have led to a new kind of parallel programming model in which elements of both data-sharing and message-passing are evident. Briefly explain how the underlying hardware architecture of GP-GPU computers has influenced the programming constructs available in languages such as CUDA or OpenCL.

(4 marks)

d) When a data-sharing style of loop is executed in parallel, **loop scheduling** is used to determine which iterations of the loop are executed by which thread. In the case that every iteration is expected to take the same amount of execution time, **static** loop scheduling is preferred. Explain what this means and describe the difference between **block** scheduling, **cyclic** scheduling and **block-cyclic** scheduling.

(4 marks)

Question 3 continues on the next page

Question 3, continued from the previous page

e) Consider the following parallel loop in C 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.

(4 marks)

[PTO]

Question 4.

a) Explain the role of Transactional Memory in a chip multiprocessor and describe in detail the three key stages that must be passed through when executing a transaction.
(6 marks)

b) In the context of an **implementation** of Transactional Memory, answer the following.

(i) Explain what the **readset** and **writeset** are and what they are used for.
(2 marks)

(ii) Explain the difference between **eager versioning** and **lazy versioning** in respect of writes to shared data during execution of a transaction.
(2 marks)

(iii) Explain the difference between **eager validation** and **lazy validation** in respect of detecting conflicts when trying to commit or abort a transaction.
(2 marks)

c) On a chip multiprocessor with Transactional Memory, each thread contains the transaction below which swaps two elements of a shared integer array `a`, where the elements to be swapped are indexed by variables `i` and `j` which are both private to the threads.

```
atomic {
   int temp = a[i] ;
   a[i] = a[j] ;
   a[j] = temp ;
}
```

(i) Write equivalent code which uses fine-grained locking on the array `a` to achieve the same effect, as far as possible.
(4 marks)

(ii) What problems might cause your code from part c)(i) to deadlock?
(2 marks)

(iii) Explain why the given transactional code cannot possibly deadlock. What other synchronisation problem might the given transactional code suffer from?
(2 marks)

END OF EXAMINATION