Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Chip Multiprocessors

Date: Tuesday 22nd May 2012
Time: 14:00 - 16:00

Please answer any THREE Questions from the FOUR questions provided

For full marks your answers should be concise as well as accurate.
Marks will be awarded for reasoning and method as well as being correct

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.

(4 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.

(8 marks)

c) Consider the Java code fragment below, in which the work done in any call to method do_work(i) is known to be independent of the work done for a call using any other value of i (i.e. there are no loop carried dependences and all iterations of the loop could in principle be executed in parallel).

```java
for (i=0 ; i < N ; i++)   // N is very large
{ do_work(i) }
```

(i) Given that the amount of work done per iteration of the loop is constant, describe the set of appropriate options that exist for scheduling the iterations of the loop to threads in a parallel implementation looking to improve the performance of the code fragment.

(3 marks)

(ii) Now suppose that the amount of work done per iteration of the loop varies substantially, and that the amount of work done for any particular value of i cannot be predicted in advance of execution. Describe the new set of appropriate options that exist for scheduling the iterations of the loop to threads in a parallel implementation looking to improve the performance of the code fragment. Explain why these new options are more expensive to use (in terms of execution time) than those in your answer to part b)(i).

(5 marks)
Question 2.

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

(5 marks)

b) Using the instruction level code for implementing a binary semaphore as an example, explain why it is necessary to provide hardware level support for synchronisation operations in a multicore processor.

(4 marks)

c) Early attempts to provide the hardware support mentioned in part b) entail instructions that can read-modify-write a variable atomically (i.e. guarantee that no other instruction can access the variable while the read-modify-write is in progress). Explain why implementation of such instructions in a modern RISC multicore processor causes undesirable effects and resultant behaviour.

(2 marks)

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

(4 marks)

e) 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 3.

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?  

(4 marks)

b) Assume a ‘snooping bus’ 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.  

(8 marks)

c) In the context of your answer to part b), 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)

d) Suppose the state described in part c) 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.

<table>
<thead>
<tr>
<th>core 1</th>
<th>core 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldr r1, x</td>
<td>ldr r1, x</td>
</tr>
<tr>
<td></td>
<td>add r1, r1, #1</td>
</tr>
<tr>
<td>add r1, r1, #1</td>
<td></td>
</tr>
<tr>
<td>str r1, x</td>
<td>str r1, x</td>
</tr>
</tbody>
</table>

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)

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 correct? Explain why.  

(2 marks)
Question 4.

a) Where multiple hardware units are available, it is possible to use them for **speculation**, thereby speeding up the execution of a sequential program. Explain the general principles of speculation, the constraints under which it should be applied, and the costs of using it. (4 marks)

b) Explain what is meant by **thread level speculation** (TLS). Describe **two** ways in which threads for TLS may be generated automatically from a sequential program. Use pseudocode examples to illustrate your answer. (6 marks)

c) What additional resources (compared to those needed for sequential execution) are required to implement fully general **loop-based TLS**? How are these additional resources utilised at runtime? (7 marks)

d) Briefly describe how multicore hardware might be modified so as to provide support that improves the performance of any activity identified in your answer to part c). (3 marks)

END OF EXAMINATION