Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Chip Multiprocessors

Date: Tuesday 28th May 2013
Time: 14:00 - 16:00

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

(4 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 hardware architecture of GP-GPU computers and explain in detail how this has influenced the programming constructs available in languages such as CUDA or OpenCL.

(6 marks)

c) 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)

d) 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, and procA is a procedure whose execution time is independent of its argument and is significantly longer than the execution time of isSquare.

```
#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.

(6 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 briefly 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.

(5 marks)

e) The following instruction level code is to be executed by N threads after the memory location pointed to by the value in r2 has been initialised to N. Explain what the code is doing and what condition must prevail when all threads reach the line labelled done.

```
loop:  ld1 r1, r2  // load linked into r1 – memory address is in r2
       sub r1, r1, #1  // decrement r1
       stc r1, r2  // store conditional r1 – memory address is in r2
       cmp #1, r1  // compare r1 with 1
       bne loop  // branch if not equal to loop
spin:  ldr r1, r2  // load r1 – memory address is in r2
       cmp r1, #0  // compare r1 with 0
       bne spin  // branch if not equal to spin
done:  …
```

(4 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, 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 a MESI cache coherence protocol, explain why it is helpful to know that a cache line is in state E.  

(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 between the three 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>
<th>core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldr r1, x</td>
<td>ldr r1, x</td>
<td>ldr r1, x</td>
</tr>
<tr>
<td>add r1, r1, #1</td>
<td>add r1, r1, #1</td>
<td>add r1, r1, #1</td>
</tr>
<tr>
<td>str r1, x</td>
<td>str r1, x</td>
<td>str r1, x</td>
</tr>
</tbody>
</table>

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 3’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)
Question 4.

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

(5 marks)

b) 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.

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

Write equivalent code which uses fine-grained locking to achieve the same effect, as far as possible. Explain any difference in behaviour between the locking code and the code using a transaction.

(5 marks)

c) Explain what the readset and writeset are used for in an implementation of 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?

(4 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?

(4 marks)

END OF EXAMINATION