Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Chip Multiprocessors

Date: Monday 18th May 2015
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
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 why cache coherence is a key feature in chip multiprocessor design.  
   (4 marks)

b) 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). Explain briefly what each of these states means in terms of the collective state of the multiple caches.  
   (3 marks)

c) In the context of your answer to part b), explain briefly why it is additionally helpful to distinguish the following states: (i) a cache line that would be in state S in exactly one core while every other core in the system holds that cache line in state I; and (ii) a modified cache line that has been copied by another core (which now holds this cache line in state S) without having been written back to memory.  
   (2 marks)

d) Suppose the states described in part c) are denoted state E (exclusive) and state O (owner); the resulting protocol is termed MOESI. In a MOESI 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 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
  add r1, r1, #1
  str r1, x
```  
```  
  ldr r1, x
  add r1, r1, #1
  str r1, x
```  

Assuming that the value of x is not cached in any of the cores 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.  
   (5 marks)
e) Supposing that the intended behaviour of the given parallel code is to allow each thread independently to add 1 to the shared variable x, what might cause the given parallel code to add less than 3 to the original value of x? (2 marks)

f) The state of a cache line is usually implemented using a small number of tag bits attached to each cache line. Briefly explain how additional tag bits can be used to assist with the implementation of a transactional memory. (4 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. (4 marks)

b) Explain briefly why it is necessary to provide hardware level support for synchronisation operations in a chip multiprocessor. (2 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) Describe the behaviour of the ‘load linked’ and ‘store conditional’ instructions and explain how these can be used to implement a binary semaphore. Discuss how these instructions overcome the problems identified in your answer to part c). Your answer should include the code required at instruction level. (5 marks)

e) The following instruction level code has been written by a novice programmer to try and obtain two semaphores whose addresses are in r2 and r3. Explain in detail why no thread executing this code can ever reach the critical section starting at the line labelled ‘crit:’.

```
frst:  ld1 r1, r2 // load linked first semaphore into r1
       cmp #0, r1 // compare r1 to 0 (busy)
       beq frst // if so, try again

secd:  ld1 r1, r3 // load linked second semaphore into r1
       cmp #0, r1 // compare r1 to 0 (busy)
       beq secd // if so, try again
       mov #0, r1 // have seen both semaphores free, so now try to set them busy
       stc r1, r3 // store conditional 0 (busy) to second semaphore
       cmp #1, r1 // compare r1 (load linked flag) with 1
       bne frst // branch to try again if not equal
       mov #0, r1 // succeed with second semaphore, so now try first
       stc r1, r2 // store conditional 0 (busy) to first semaphore
       cmp #1, r1 // compare r1 (load linked flag) with 1
       bne frst // branch to try again if not equal

crit:  …
```

(4 marks)

f) Is there a way to refactor the code in part e) so that it achieves what it is trying to do? Would there be any residual problems with the refactored code? (3 marks)
Question 3.

a) Explain the key distinctions between parallel programming models that are based on data-sharing and those 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. (6 marks)

c) Why is ‘pure’ functional programming attractive as a means of programming chip multiprocessors? In what ways is it similar to or distinct from data-sharing or message-passing? (3 marks)

d) 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 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 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 d)(i). (5 marks)
Question 4.

a) Describe the main architectural features that are found in a modern general-purpose graphics processing unit (GP-GPU). Your description should include details of the hardware processing units in the GP-GPU, as well as the interface to the ‘host’ platform. (5 marks)

b) CUDA and OpenCL provide programming language extensions that are suitable for writing programs that utilise GP-GPU hardware. Describe the kinds of extension such languages provide, and explain how each kind of extension assists the programmer in utilising the hardware features identified in your answer to part a). Use appropriate pseudocode examples to illustrate your answer. (5 marks)

c) Matrix multiplication requires an inner product to be computed for each element of the output matrix. Assume square matrices of size $n \times n$ in which each element is a double (64-bit floating point) value. The inner product to form the element at the intersection of the $i$th row and the $j$th column is $\sum_{k=1}^{n} a_{ki} \times b_{jk}$. This requires execution of $n$ multiplies and $(n-1)$ adds, or $n$ multiply-adds of an appropriate form, starting the addition at zero. Let the maximum rate of processing in the GP-GPU be $g$ flop/s (64-bit floating point operations per second), the maximum rate of processing in the host CPU be $c$ flop/s, and the maximum DMA transfer rate between the host CPU and the GP-GPU be $b$ bytes/s. Assume that both host CPU and GP-GPU have a floating point multiply-add instruction of the required form, which counts as a single floating point operation. Derive expressions for the total time to multiply two matrices that are initially both stored in the host CPU memory: (i) executed entirely in the host CPU; and (ii) executed in the GP-GPU, returning the result back in the host CPU memory. State any assumptions you make. Hence or otherwise determine the values of $n$ for which it would be faster to perform the matrix multiplication in the GP-GPU. (5 marks)

d) In practice, it may be undesirable to leave either the host CPU or the GP-GPU idle for any period of time. A strategy for avoiding this is the following. Send a fraction $\alpha$ ($0<\alpha<1$) of each input matrix from the host CPU to the GP-GPU (one matrix will send $\alpha n$ columns, the other will send $\alpha n$ rows). The GP-GPU then computes a total of $\alpha n \times \alpha n$ elements of the result (all the data it needs will be in the GP-GPU). Meanwhile the host CPU computes the remaining elements of the result (all the data is still in its memory). The times taken for computing these two parts should be equal. Finally, the result elements in the GP-GPU need to be transferred back to the host CPU. Using the same hardware rates as were given in part c), first determine what the value of $\alpha$ should be, then derive an expression for the time to execute this matrix multiplication program (note: the latter should not include terms involving $\alpha$). (5 marks)