This question paper consists of 5 printed pages, each of which is identified by the Code Number COMP3221.

School of Computing

May/June 2018

COMP3221

Parallel Computation

Answer all 2 questions

Time allowed: 2 hours

TURN OVER FOR QUESTIONS

Question 1

Inspect the segment of code in Fig. 1. This shows a function `addValue` that takes a floating point value `x` and performs some time-consuming calculation on it, returning the value `val`. This is then added to element `index` of a global array `data`. The calculation itself is performed by a separate routine that does not alter the value of `x`.

```
1 void addValue( float x, int index )
2 {
3   float value = performSomeComplexCalculation( x );
4   data[index] = data[index] + value;
5 }
```

Figure 1: Code for Question 1.

(a) Consider first Amdahl's law, which states that the maximum speedup $S^{\max}$ for a parallel program with a fraction f left in serial is

$$S^{\max} = \frac{1}{f + \frac{1-f}{p}} \quad , \tag{1}$$

where p is the number of processing units, *e.g.* threads.

(i) Define the speedup in terms of the serial execution time $t_{\rm s}$ and the parallel execution time $t_{\rm p}$. **[1 mark]**

(ii) Derive Amdahl's law, equation (1). **[4 marks]**

(b) Within a multi-threaded context, the function `addValue()` in Fig. 1 may be called simultaneously by two or more threads. **[2 marks]**

(i) Define a data race, and the conditions under which one may occur.

(ii) Data races are known to sometimes lead to non-deterministic behaviour. Describe what this means. Explain how it can occur in the code in Fig. 1. **[4 marks]**

(iii) At which line number in Fig. 1 does the data race potentially arise? **[1 mark]**

(c) It is suggested that in order to make the function thread-safe, lines 3 and 4 should both be contained within a single critical region, such as that implemented in OpenMP as `#pragma omp critical {...}`.

(i) It is found that making this change does indeed result in a thread-safe `addValue()`. Explain this observation. **[3 marks]**

(ii) However, the performance is significantly reduced, even for a large number of threads p. With reference to Amdahl's law, explain why. **[3 marks]**

(iii) Suggest one way in which moving the start and/or end of the critical region can improve performance while maintaining thread-safety. Give your reasoning. **[3 marks]**

CONTINUED ON FACING PAGE

(d) It is further suggested that using multiple locks to control access to the array `data` should result in a further performance benefit. Outline how this might be implemented, and why the performance might improve. **[4 marks]**

[question 1 total: 25 marks]

Question 2

A reduction operation can be defined as when a collection of elements is reduced to a smaller collection by the repeated application of a combiner function. This combiner function is typically a binary operator $\otimes$ that acts on two elements, returning a single element, *i.e.* $c = a \otimes b$.

(a) For parallel reduction it is important that the operator $\otimes$ is associative, *i.e.*

$$a \otimes (b \otimes c) = (a \otimes b) \otimes c \quad .$$

Why is this? **[2 marks]**

(b) Which of the following operations are commutative, *i.e.* obeys $a \otimes b = b \otimes a$? Which are exactly associative, and which only approximately so? If approximately associative, explain why, and the possible consequence in relation to the equivalent serial reduction. **[4 marks]**

(i) Integer multiplication (ignoring overflow).

(ii) Taking the average of two `float` variables, *i.e.* `0.5f*(a+b)`.

(c) Fig. 2 shows a binary tree as might be employed for reduction. Suppose you were asked to implement this pattern in parallel on a multi-core CPU. How would you ensure the correct computation every time it is run? How would your implementation differ for a distributed memory architecture, and why? You do not need to give any code or pseudo-code as long as your description is clear. **[6 marks]**

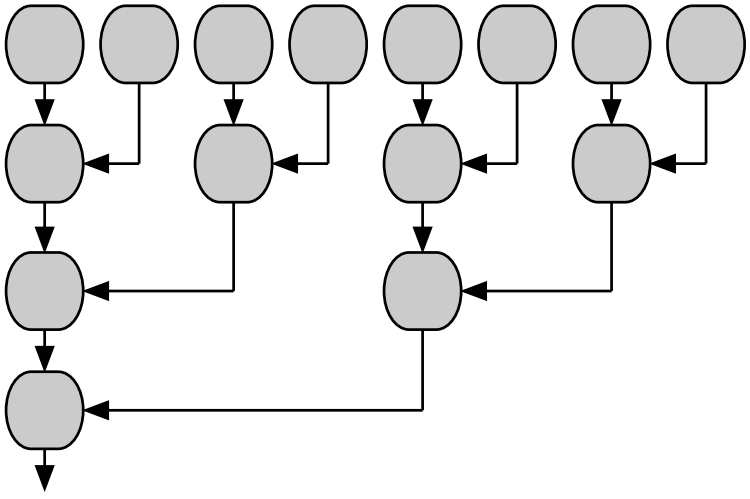

Figure 2: Binary tree for reduction. See question 2(c).

(d) Reduction on a GPU introduces new difficulties, but also potential benefits. Suppose you need to reduce a data set that is very large (but not so large as to exceed memory).

(i) What challenge do large data sets pose when trying to perform an operation such as reduction on a GPU, and what can be done to resolve it? You do not need to provide any implementation details specific to reduction. **[3 marks]**

CONTINUED ON FACING PAGE

(ii) Near the end of the reduction, when the number of remaining calculations to be performed is small, what feature of a typical GPU can be exploited to improve performance? **[2 marks]**

(e) Look again at Fig. 2, and note this can be interpreted as a task graph; that is, a directed acyclic graph describing the dependencies in the calculations.

(i) In which levels of the tree are the reductions between processing units actually being performed? **[1 mark]**

(ii) Regard each node of the tree in which calculations are being performed as a task, and assume each task takes equal time. What is the work and span of this graph? **[2 marks]**

(iii) Consider an arbitrary binary tree that has $p = 2^m$ nodes in the uppermost row, so the version in the figure corresponds to $m = 3$. Again assuming that each task takes equal time, what is the work and span now? **[2 marks]**

(iv) For both of the cases (ii) and (iii), what is the maximum speedup according to the work-span model? **[3 marks]**

[question 2 total: 25 marks]

[grand total: 50 marks]

END