Module Code: COMP322101

Module Title: Parallel Computation

School of Computing

Semester 2 2018/2019

Calculator instructions:

- You **are** allowed to use a non-programmable calculator only from the following list of approved models in this examination: **Casio FX-82**, **Casio FX-83**, **Casio FX-85**.

Dictionary instructions:

- You are **not** allowed to use your own dictionary in this examination. A basic English dictionary is available to use: raise your hand and ask an invigilator, if you need it.

Examination Information

- There are **5** pages to this exam.
- There are **2 hours** to complete the exam.
- Answer **all 2** questions.
- The number in brackets [ ] indicates the marks available for each question or part question.
- You are reminded of the need for clear presentation in your answers.
- The total number of marks for this examination paper is **50**.

Question 1

(a) Let $t_{\rm s}$ be the serial execution time for an algorithm, and suppose that the equivalent parallel algorithm takes a time $t_{\rm p}$ when executed on p processing units.

(i) Define the speedup S in terms of $t_{\rm s}$ and $t_{\rm p}$.

[1 mark]

(ii) Suppose a fraction f of the parallel execution time is actually executed in serial. Derive the Gustafson-Barsis law that the maximum speedup is $S = f + p(1 - f)$.

[2 marks]

(iii) What is the difference between weak and strong scaling? Which is the Gustafson-Barsis law related to?

[2 marks]

(b) Fig. 1 shows part of an MPI program that sends data to the process with the next higher rank, with wrap-around so the process with rank $p - 1$ sends to rank 0. It is found that this cyclic communication pattern causes deadlock for large messages.

```
// Initialise MPI.
int numProcs, rank;
MPI_Comm_size(MPI_COMM_WORLD,&numProcs);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);

// Initialise arrays.
...  // (not shown)

// Send data to process with one higher rank, with wrap-around.
MPI_Send(sendData,n,MPI_FLOAT,(rank==numProcs-1?0:rank+1),
         0,MPI_COMM_WORLD);
MPI_Recv(recvData,n,MPI_FLOAT,(rank==0?numProcs-1:rank-1),
         0,MPI_COMM_WORLD,MPI_IGNORE_STATUS);

// Deallocate arrays and call MPI_Finalize().
...  // (not shown)
```

Figure 1: Code for question 1(b).

(i) What is deadlock, and why does it occur in this situation?

[3 marks]

(ii) One way to resolve this is to change the program logic so that some processes call `MPI_Send()` before `MPI_Recv()`, while others call `MPI_Recv()` before `MPI_Send()`. Give one other solution. You do not need to provide any MPI function names.

[1 mark]

(iii) Someone changes the program in Fig. 1 so that rank 0 calls `MPI_Recv()` before `MPI_Send()`. All other ranks 1 to $p-1$ inclusive call `MPI_Send()` before `MPI_Recv()` as before. This is found to resolve the deadlock. Explain why. You may like to use a diagram to support your answer.

[5 marks]

(iv) However, timing runs show that the total communication time is very long. Explain why. How would you expect the time for the total communication to vary with p?

[3 marks]

(c) Someone is writing an application for a shared memory architecture and identifies a critical region with a potential data race. Rather than use a high level construct (such as OpenMP's `#pragma omp critical`), they decide to implement their own solution using a lock.

(i) Write some pseudo-code to explain how a lock can be employed to ensure only one thread enters the critical region at a time. You may assume that the parallel framework you are using provides a lock object `lock_t` with methods `lock()` and `unlock()` for locking and unlocking respectively.

[3 marks]

(ii) At an even lower level, the lock and unlock routines employ atomic operations to perform their functions. What is an atomic operation, and why are they used by locks?

[2 marks]

(iii) The function `aCAS(int *x, int cmp, int y)` atomically sets `x=y` if `*x==cmp`. It returns the original value of `x` regardless of the result of the comparison. Show how this function can be used to implement the acquiring of a lock at the start of the critical region. Second and subsequent threads should just skip the region. You do not need to show how the lock is released.

[3 marks]

[Question 1 Total: 25 marks]

Question 2

Consider the serial code in Fig. 2. This calculates all `N` values of the floating point array `out[N]` from known values of the array `in[N]` of the same size. The calculations are performed in a loop, using the function `float performComplexCalculation(float)`. The details of this function are not known to us.

```
float in[N], out[N];
...  // Initialise all N elements of in[N].

for(i=0;i<N;i++)
{
    out[i] = performComplexCalculation(in[i]);
}
```

Figure 2: Code for question 2.

(a) You are asked to parallelise this code to run on a multi-core CPU using OpenMP.

(i) A data dependency arises when one calculation depends on the result of another calculation. Are there any data dependencies in the code of Fig. 2? Explain your answer.

[2 marks]

(ii) Would your answer change if `performComplexCalculation` took two values of `in` as input, such as

```
out[i] = performComplexCalculation2(in[i],in[N-i]);
```

If not, why not?

[2 marks]

(iii) How would you parallelise the loop using OpenMP? You do not need to give the precise instruction(s), but your meaning should be clear.

[2 marks]

(b) When profiling it is discovered that the time for `performComplexCalculation()` to return varies greatly, being sometimes very fast and sometimes very slow.

(i) Why might this affect the performance of the parallel implementation? What is this phenomenon known as?

[3 marks]

(ii) How would you modify your answer to question 2(a)iii to improve performance in this situation? As before, the exact OpenMP instruction(s) are not required as long as your meaning is clear.

[2 marks]

(c) It is later decided to parallelise the problem on a distributed memory system. The required implementation is a work pool. One process will be the master that coordinates the whole problem, and all other processes will be slaves that call `performComplexCalculation()`.

(i) In general, give one advantage and one disadvantage of blocking communication compared to non-blocking communication.

[2 marks]

(ii) For this problem blocking communication is to be used. Outline how you would structure an MPI program to implement a work pool. You do not need to provide actual code – pseudo-code, diagrams, and/or clear descriptions will suffice. There is no need to mention the variables the master uses to keep track of progress, but initialisation and termination should be covered.

[7 marks]

(d) For a different problem it is found that all calculations take the same time, but there are now data dependencies. The calculations and their dependencies are shown in Fig. 3.

(i) What is a network such as the one in Fig. 3 known as?

[1 mark]

(ii) Define the critical path for a network such as this. What is this path's length known as?

[2 marks]

(iii) What is the length of the critical path for the network of Fig. 3?

[2 marks]

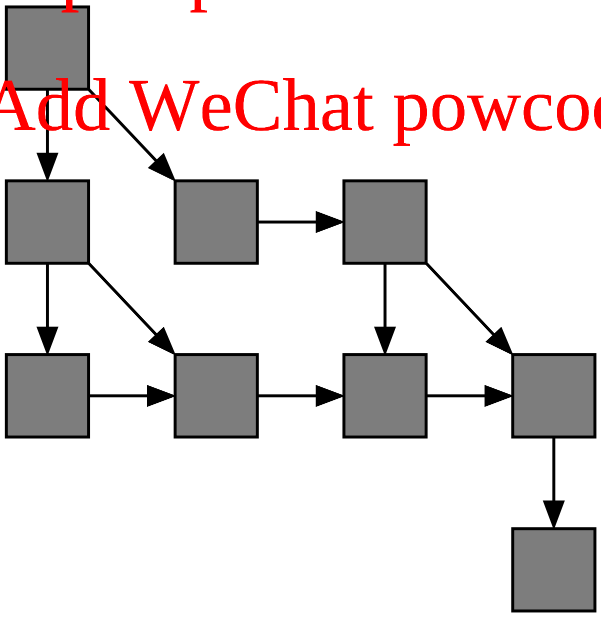

Figure 3: Network for question 2(d). Grey squares correspond to calculations and arrows denote dependencies.

[Question 2 Total: 25 marks]

[Grand Total: 50 marks]