Module Code: COMP322101

Module Title: Parallel Computation

School of Computing

Semester 2 2020/2021

Examination Information

- There are **7** pages to this exam.
- Answer **all 2** questions.
- The total number of marks for this examination paper is **80**.
- The number in brackets **[ ]** indicates the marks available for each question or part question.
- Your submission should be less than **4000 words** in total.
- It is possible to receive full marks using less than **2000 words** in total.
- You may lose marks for including extensive discussions that are not relevant to the question.
- You will not receive any marks, and may lose marks, for simply copying material from your notes verbatim.
- You are reminded of the need for clear presentation in your answers.
- You are required to submit a single **PDF file** *via* the Minerva submission point.

Question 1

(a) The parallel speedup S is defined as the ratio of the serial execution time t_{s} to the parallel execution time t_{p}, $S = t_{\mathrm{s}}/t_{\mathrm{p}}$. Suppose your hardware supports p processing units. What are the maximum and minimum values of S, if such limits exist, in terms of p? Justify your answers by explaining in what type of situation each limit could be approached.

[4 marks]

(b) Suppose you need to solve a problem that is comprised of N independent tasks, each of which takes the same time to complete. You want to execute these in parallel using p processing units. However, N is not a multiple of p.

(i) It is suggested that the N tasks can be distributed over the p processing units as follows: $p-1$ processing units are allocated $\lceil N/p \rceil$ tasks, where $\lceil N/p \rceil$ is N/p rounded up to the nearest integer, and the final processing unit then takes all remaining tasks. What problem or problems can you see with this suggestion?

[4 marks]

(ii) It is now proposed to allocate the tasks to processes in a 'round-robin' manner, cycling through all processing units, allocating one task to each in turn before moving onto the next unit. Provide some pseudo-code for a loop that implements this allocation of tasks.

[4 marks]

(iii) However, the overhead in performing this loop is regarded as too high, and direct expressions are preferred. Give one or two lines of C code that determines the number of tasks `numTasks` that should be allocated to the processing unit with index `unit`, where `0<=unit<p`. Your answer should not include a loop but may include an `if` statement.

[4 marks]

(c) Someone has written some C code that implements the multiplication of a square matrix `M` of size $N \times N$ by a vector `a` of size N, leaving the result in a separate vector `b` also of size N, *i.e.* `b = M a`. The essential serial code is given below.

```
float a[N], b[N], M[N][N];
// ...  (initialise a and M )

// Matrix multiplication:  b = M a
int i, j;
for( i=0; i<N; i++ )
{
    b[i] = 0.0;
    for( j=0; j<N; j++ )
        b[i] += M[i][j] * a[j];
}
```

Since N is very large for the intended application, it is decided to parallelise the matrix multiplication using OpenMP.

(i) For a first attempt, the programmer adds a `#pragma omp parallel for` directive directly before the loop in line 6. However, on testing, this is shown to sometimes give different answers to the serial code. Explain why this happens, and how you would modify the code to fix the problem. Use the line numbers given above wherever possible.

[4 marks]

(ii) It is now decided to leave the outer loop starting on line 6 in serial, and to instead parallelise the inner loop that starts on line 9. A `#pragma omp parallel for` directive is used as before. However, this still does not give the correct answer. Explain why, and explain how you would fix this problem. Again, use the given line numbers in your answer.

[4 marks]

(iii) After fixing the code so that it always works correctly, consider the performance related to memory access. C stores matrices in row-major order, which means that each row is stored in contiguous blocks of memory. That is, row `M[0]` is stored in one block of memory, followed by row `M[1]`, followed by row `M[2]`, and so on. Look carefully at the loop at lines 9 and 10 and recall what you know about memory caches. In this context, would you recommend parallelising the inner loop or the outer loop? Do you think this will be a serious performance issue for this problem? Explain your answers.

[6 marks]

(d) Consider the task graph given in Figure 1. Each node in the graph represents a single task, and the arrows denote dependencies that must be satisfied before the next task can be started. All tasks take an equal amount of time.

(i) How long will it take to complete the task graph using $p = 1$ processing units? You should give your answer in multiples of the time for a single task.

[1 mark]

(ii) How long will it take to complete the task graph using $p = \infty$ processing units, *i.e.* when you are allowed to perform as many tasks in parallel as possible?

[2 marks]

(iii) Finally, how long will it take to complete the task graph using $p = 2$ processing units?

[3 marks]

(iv) What are the quantities calculated in parts (i) and (ii) of this question better known as? In terms of these two quantities, what is the predicted maximum speed-up for this task graph?

[4 marks]

[Question 1 Total: 40 marks]

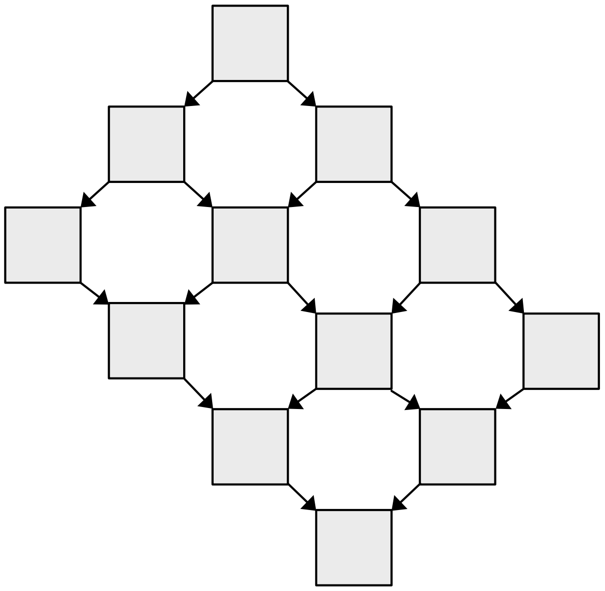
Figure 1: Task graph for question 1(d).

Question 2

Figure 2 gives code for a function `area()` that estimates the area under a given graph of `f(x)` (where `f(x)` returns non-negative values) over the range `x=xmin` to `x=xmax`. It achieves this by summing the areas of `N` rectangles of height `f(x)` and width `h`, where `N>0` is an argument and `h` is calculated from `xmin`, `xmax` and `N`. For the intended application, `f(x)` is time-consuming to evaluate for each `x`. Calculated values are therefore stored in an array `store[N]` so they can be re-used. For all of question 2, you may assume N is a multiple of p.

You are asked to parallelise `area()`. Inspect the code, then answer the following questions.

```
1      // Compute-intensive function of x only.
2      float f( float x )  ...
3
4      // Returns an estimate of the area under f(x) from xmin
5      // to x=xmax.  Calculated values of f() are copied to store[N].
6      float area( float xmin, float xmax, int N, float *store )
7      {
8        float h = (xmax-xmin)/N;
9
10       // Calculate and store f(x) for each x.
11       int i;
12       for( i=0; i<N; i++ ) store[i] = f(xmin+i*h);
13
14       // Estimate the area.
15       float area = 0.0f;
16       for( i=0; i<N; i++ ) area += h * store[i];
17
18       return area;
19     }
```

Figure 2: Code for question 2(a).

(a) Consider first the loop at line 12, which calculates `f(x)` over a range of values of `x`, and stores the results in an array `store[N]`.

(i) Identify any data dependencies in this loop. Explain your answer.

[2 marks]

(ii) On a shared memory CPU architecture and OpenMP, this loop can be parallelised by using a `#pragma omp parallel for` directive just before the loop. Provide pseudo-code for the actions performed by each thread in the case that there are $p = 3$ threads.

[5 marks]

(iii) Now look at the loop on line 16, which calculates the final area from the values stored in the array `store`. How would you parallelise this loop using OpenMP? Do you expect the result of the parallel version of this loop to exactly match the serial calculation? Explain your answer.

[4 marks]

(b) It is now decided that `area()` will be parallelised on a distributed memory CPU architecture. You have a choice of the two cluster configurations shown in Figure 3, both of which have 6 nodes:

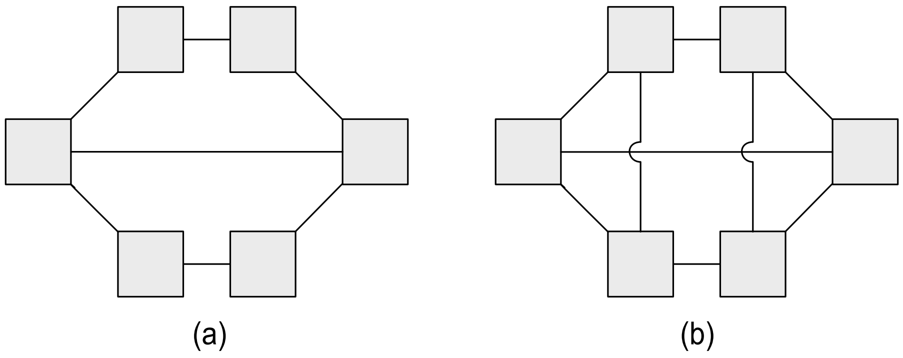

Figure 3: Cluster configurations for question 2(b).

(i) The diameter δ of the network is defined as the maximum shortest distance between pairs of nodes. What is δ for these two networks? Assuming all other aspects (CPUs, interconnect *etc.*) are identical, which configuration would you choose? Explain your answer.

[4 marks]

(ii) Suppose that each process has access to the function `float f(float x)`, but the parameters `xmin`, `xmax` and `N` are only known to rank 0. What MPI function or functions would you use to synchronise these parameter values across all processes? How does the MPI runtime implement each these functions?

[3 marks]

(iii) At the end of the calculation, the MPI process with rank zero should have the full array `store`, and the result of the calculation, `area`. Outline how you would implement the calculation in MPI so that the above requirements are met. You do not need to write working code, but should use the correct MPI function names throughout.

[4 marks]

(c) Now suppose this algorithm is to be executed on a GPU using OpenCL. The chosen approach is to have a single work item for each value of x, and for a single kernel to both calculate `f(x)` and determine the total area, using local memory as far as possible. The final result is stored in the floating point variable `device_area` in global memory, which is then copied back to the CPU. It is no longer a requirement to store evaluations of the function `f()` in the array `store[N]`.

Suppose first that N is within the maximum work group size supported by the device. The first part of the OpenCL kernel is given in Figure 4.

```
1           __kernel
2           void area(float xmin, float xmax, __global float *device_area,
3                       __local float *temp )
4           {
5                int gid = get_global_id(0);
6                int size = get_local_size(0);
7                float h = ( xmax - xmin ) / N;
8                float x = xmin + gid*h;
9                float f = ...; // Definition of f(x) inserted here.
10               ...  (remainder of calculation)
11          }
```

Figure 4: Code for question 2(c).

(i) How would the fourth kernel argument, *i.e.* `__local float *temp`, be set in the corresponding C-code?

[5 marks]

(ii) Fill in the remainder of the kernel in Figure 4, *i.e.* from line 10 inwards, that leaves the final answer in `device_area`. You may assume N is a power of 2. You should use the provided variable names, and the correct OpenCL function names, throughout.

[6 marks]

(iii) Now suppose that the problem size N is larger than the maximum work group size supported by the device. Briefly outline how you would perform the calculation on this larger data set.

[3 marks]

(iv) Given you are told the function `f(x)` is time-consuming, what potential performance issue can you see for a GPU implementation that you would not expect to arise on a CPU?

[4 marks]

[Question 2 Total: 40 marks]

[Grand Total: 80 marks]