Coláiste na Tríonóide, Baile Átha Cliath
Trinity College Dublin
Ollscoil Átha Cliath | The University of Dublin

Faculty of Engineering, Mathematics and Science

School of Computer Science & Statistics

Integrated Computer Science **Trinity Term 2018**
Year 3 Annual Examinations

Concurrent Systems I

Monday 30th April 2018 **Regent House** **14.00-16.00**

Mr Harshvardhan J. Pandit

Instructions to Candidates:

- Answer 2 out of 3 questions
- All questions are marked out of 50
- All code should be commented, indented, and use good programming style
- Exam paper is not to be removed from the venue

Materials Permitted for this examination:

Non-programmable calculators are permitted for this examination – please indicate the make and model of your calculator on each answer book used.

Q1. (a) Compare-and-Swap (CAS) is an atomic instruction used in multithreading to achieve synchronization. It compares the contents of a memory location with a given value, and only if they are the same, modifies the contents of that memory location to a new given value. This is done as a single atomic operation. Most multiprocessor architectures support CAS, and the compare-and-swap operation is the most popular synchronization primitive for implementing both lock-based and non-blocking concurrent data structures. Explain how this can be used to create a lock between threads. Example C code given in the Intel Software Manual demonstrates CAS as follows: [10 marks]

```
bool compare_and_swap(int *accum, int *dest, int newval) {
     if (*accum == *dest) {
          *dest = newval;
          return true;
     } else {
          return false;
     }
}
```

(Question 1 continues on next page)

(Question 1 continued from previous page)

Q1. (b) Examine and state whether each of the following code blocks can be parallelised effectively using SSE (x86). If not, clearly state why. If the code can be vectorised, use SSE intrinsics to vectorise it. Write a short note about the parallelisation strategy you used.

```
// Code Segment #1
void compute(int* array, int SIZE) {
     // array is 16-byte unaligned address
     int i = 1;
     while (i < SIZE) {
         array[i] = array[i] + array[i - 1];
         i = i + 1;
     }
}
```

[10 marks]

```
// Code Segment #2
void compute(float* array, int SIZE, float multiplier) {
     for(int i=SIZE-1; i>=0; i--) {
         array[i] = (array[i] * multiplier) / SIZE;
     }
}
```

[10 marks]

```
// Code Segment #3
int compute(int* array, int SIZE) {
     // array is 16-byte aligned address
     int max = 0;
     for(int i=0; i<SIZE; i++) {
         if (array[i] > max) { max = array[i]; }
     }
     return max;
}
```

[10 marks]

```
// Code Segment #4
bool strcmp(char* string1, char* string2, int strlength) {
    for(int i=0; i<strlength; i++) {
      if (string1[i] != string[2]) {
           return false;
      }
    }
    return true;
};
```

[10 marks]

Q2. A function called `max_matrix` finds the position of the maximum element from a three dimensional matrix of floats. It returns the position of the maximum element in the form of three coordinates designated as `x, y, z`.

(a) Write a C program using OpenMP to parallelise operations for faster execution. Your code is expected to store the position of the maximum in an integer array of size 3.

[30 marks]

(b) Describe your approach towards parallelisation and considerations regarding simultaneous execution of code. Write about underlying assumptions you have made, and how the number of CPU cores or threads will affect its execution. You should provide a reasonable estimate of the time-complexity of your solution.

[20 marks]

Q3. The following code downloads some webpages, then calculates and stores its *hashsum*.

```
01 void download(int SIZE,char* hash) {
02    for(int i=0; i<SIZE; i++) hash[i] = null;
03    char** buffer = get_stream_data(SIZE);
04    for(int i=0; i<SIZE; i++) {
05         char* stream = buffer[i];
06         hash[i] = md5_hashsum(stream);
07    };
08 }

10 char** get_stream_data(int nos) {
11     char** streams = (char**) malloc(nos * sizeof(char*));
12     char buffer[2147483647];
13     for (int stream_no=0; i<nos; i++) {
14          download_stream_from_server(stream_no, buffer);
15          streams[stream_no] = (char*) malloc(
16                  strlen(buffer) * sizeof(char*));
17          strcpy(streams[stream_no], buffer);
18     };
19     return streams;
20 }
```

(a) Identify potential parallelism in the above code [10 marks]

(b) Identify and discuss whether threads or asynchronous co-routines will be better suited for each parallel task identified in Q3(a).

[20 marks]

(c) Discuss the suitability of following architectures in executing the following code assuming the array size is sufficiently large to NOT fit into memory (RAM).

(i) vector processors (ii) symmetric multiprocessors

(iii) out-of-order superscalar (iv) NUMA multiprocessor [5 marks/each]

```
int array[SIZE];     long sum = 0;
for(int i=0; i<SIZE; i++) {
     int sum_i = 0;
     for(int j=i; j<SIZE; j++) {
          int sum_j = 0;
          for(int k=j; k<SIZE; k++)
               sum_j += array[k];
          sum_i += sum_j;
     }
     sum += sum_i;
}
```