UNIVERSITY OF DUBLIN

TRINITY COLLEGE

Faculty of Engineering, Mathematics & Science
School of Computer Science & Statistics

Integrated Computer Science Programme **Trinity Term 2012**
Junior Sophister Examination

Concurrent Systems 1 (CS3014)

14th May 2012 **Luce Lower** **14.00 – 16.00**

Dr David Gregg

Instructions to Candidates:

- Answer 2 out of the 3 questions
- All questions are marked out of 50
- All program code should be commented appropriately

Materials permitted for this examination:

- Non-programmable calculator

1.

a) Modern architectures commonly provide one or more atomic machine instructions that can be used to implement locks. One of these is the atomic compare-and-swap instruction. Explain how this instruction can be used to implement locks on shared-memory parallel computers.

[10 marks]

b) Examine each of the following pieces of code. State if each individual piece of code can be vectorized using SSE. If not, state clearly why. If the code can be vectorized, use SSE intrinsics to vectorize it. Write a short note explaining the parallelization strategy on any code you vectorize.

```
/* Code segment 1 */
void add_scaled(float * a, float * b, float *c, float factor) {
    for ( int i = 0; i < 1024; i++ ) {
        a[i] = b[i] + c[i] * factor;
    }
}

/* code segment 2 */
/* src and dest memory regions do not overlap */
void memcpy(char * dest, char * src, int size) {
    for ( int i = 0; i < size; i++ ) {
        dest[i] = src[i];
    }
}

/* code segment 3 */
float dot_product(float * a, float * b, int s) {
    float result = 0.0;
    for ( int i = 0; i < s * 4, i++ ) {
        result = result + a[i] * b[i];
    }
    return result;
}
```

```
/* code segment 4 */
float stddev(float * mean, float * samples, int size) {
        float sum = 0, result;
        for ( int i = 0; i < size; i++ ) {
                float temp = samples[i] – mean;
                sum = sum + temp * temp;
        }
        result = sum / (float) size;
        return sqrt(result);
}
```

[10 marks for each segment]

2. Computers are always designed with typical applications in mind. The result is that computers to solve different problems have very different features. The following is C code that convolutes an input computer image with a 3X3 convolution kernel to create an output image. A matrix is simply a two-dimensional array. Note that we have set the pointer to the kernel matrix so that kernel[0][0] is the centre element of the matrix. Thus, the elements of the kernel matrix run from kernel[-1][-1] to kernel[1][1].

```
void conv(float ** in, float ** out, float ** kernel, int size)
{
      for ( int i = 1; i < size -1; i++ ) {
            for ( int j = 1; j < size-1; j++ ) {
                  float sum = 0;
                  for ( int k = -1; k <= 1; k++ ) {
                        for ( int m = -1; m <= 1; m++ ) {
                              sum = sum + in[i+k][j+m] *
                                          kernel[k][m];
                        }
                  }
                  out[i][j] = sum;
            }
      }
}
```

a) Identify any potential parallelism in the above code.

[10 marks]

b) Discuss the suitability of various parallel computer architectures for executing this code, assuming a large array. The parallel architectures you should discuss are:

I. out-of-order superscalar

II. VLIW

III. vector processor

IV. multithreaded/simultaneous multithreaded

V. shared memory multi-core processor

VI. multiple chip symmetric multiprocessor

VII. NUMA multiprocessor

VIII. distributed memory multicomputer.

[question continues on next page]

You should explain which aspects of each architecture suit the code well and identify any likely bottlenecks or problems in the running of the code. For each architecture you should also describe any programmer intervention that may be required to parallelize the code for the particular architecture.

[5 marks per architecture]

3. The histogram of an image can be used to measure the distribution of colours in the image. For a greyscale image, that is an image where all colours are different shades of grey, the darkness (or lightness) of a pixel in the image can be represented with a single number. This number is often represented by a single byte value between 0 and 255 inclusive. The histogram is an array of 256 integer values, with one entry for each of the 256 possible shades of grey. If, for example, there are 25 pixels in the image with a greyscale shade of 110, then histogram[110] will have the value 25.

Write a parallel routine to compute the histogram of a grayscale image using C and OpenMP. Your routine should have the following prototype:

```
int * compute_histogram(unsigned char ** image, int height,
                        int width)
```

Where *image* is a two dimensional array of bytes representing the pixels of a greyscale image, *height* and *width* are the dimensions of the image. Your routine should return a new array of 256 integers, containing the histogram of the image.

[30 marks]

Write a short commentary on your function explaining how it works, and why you believe it to be efficient. You should comment on the time and space complexity of each step of your solution, and also explain why you think it is likely to run efficiently on a modern multi-core architecture. If you choose to compute a separate histogram for each part of the image you should comment in particular on the complexity of combining the histograms.

[20 marks]