UNIVERSITY OF DUBLIN

TRINITY COLLEGE

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

Integrated Computer Science Programme **Trinity Term 2014**
Junior Sophister Examination

Concurrent Systems 1 (CS3014)

16th May, 2014 **Exam Hall** **09.30-11.30**

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) A long running discussion is whether it is better for a processor to have a small number of powerful out-of-order cores or a larger number of less powerful in-order cores. Outline the advantages and disadvantages of the two alternatives for different types of applications. Make a case for the configuration of cores that you think makes most sense, whether it be few out-of-order cores, many in-order cores, or something else.

[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 shift(float * array, int size)
{
   for (int i = 1; i < size; i++ ) {
         array[i] = array[i-1];
   }
}
```

```
/* code segment 2 */
/* note that the restrict keyword indicates that arrays
   a and b do not overlap */
void compute(float *restrict a, float *restrict b)
{
   for ( int i = 0; i < 4096; i++ ) {
         b[i] = a[i] * 3.0 + b[i+4];
   }
}
```

```
/* code segment 3 */
float rgb_sum_product(float * pixels) {
    for ( int i = 0; i < 1011; i += 3 ) {
            sum_red = sum_red + pixels[i];
            sum_green = sum_green + pixels[i+1];
            sum_blue = sum_blue + pixels[i+2];
    }
    return sum_red * sum_green * sum_blue;
}
```

```
/* code segment 4 */
void multiply(float ** matrix, float *vec, float * result)
{
    for ( int i = 0; i < 4096, i++ ) {
            float sum = 0.0;
            for ( int j = 0; j < 4096; j++ ) {
                    sum += vec[j] * matrix[i][j];
            }
            result[i] = sum;
    }
}
```

[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 C code implements a simplified version of a key part of the *n*-body problem. The *n*-body problem is the problem of predicting the motion of a group of planets interacting with each other gravitationally.

```
struct planet {
        float x, y, z, vx, vy, vz, mass;
};

void nbody(int nbodies, struct planet * bodies, float dt)
{
        float dx, dy, dz, dist, mag;
        for (int i = 0; i < nbodies; i++) {
                for (int j = i + 1; j < nbodies; j++) {
                        dx = bodies[i].x - bodies[j].x;
                        dy = bodies[i].y - bodies[j].y;
                        dz = bodies[i].z - bodies[j].z;
                        dist = sqrt(dx * dx + dy * dy + dz * dz);
                        mag = dt / (dist * dist * dist);
                        bodies[i].vx += dx * bodies[j].mass * mag;
                        bodies[i].vy += dy * bodies[j].mass * mag;
                        bodies[i].vz += dz * bodies[j].mass * mag;
                }
        }
}
```

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. very long instruction word (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.

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 greyscale 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]

© UNIVERSITY OF DUBLIN 2014