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 and Statistics

Integrated Computer Science
Year 3 Annual Examinations

Trinity Term 2017

Concurrent Systems I

Tuesday 9th May 2017 **RDS Main 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, indented and use good programming style

Materials permitted for this examination:

- Calculator

1.

a. The trade-offs between different types of parallel computer architecture are often strongly influenced by the underlying components that are used to implement them. For example, one goal of complex instruction sets with variable-length instructions was to reduce the size of the program in memory. It was only in the 1980s, when larger memories became much cheaper, that RISC instruction sets became successful. Today, one of the current big changes in parallel computer architecture is the move towards stacked memory, where DRAM is stacked directly on top of the multicore processor. Describe what you think might be the main impacts of stacked memory on parallel computer architecture and software, and present arguments for why you think these impacts are most important and/or likely. [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 compute(float * a, float * b)
{
    for ( int i = 0; i < 1024; i++ ) {
            b[i] = (1.0/(a[i] * a[i])) + 3.14159;
    }
}
```

[10 marks]

```
/* code segment 2 */
float find_max(float * array, int size) {
    float max = array[0];
    for (int i = 1; i < size; i++ ) {
            if ( array[i] > max ) {
                    max = array[i];
            }
    }
    return max;
}
```

[10 marks]

Question 1 continued on next page

Question 1 continued from previous page

```
/* code segment 3 */
float rgb_sum_product(float * pixels) {
    float sum_red = 0.0, sum_green = 0.0, sum_blue = 0.0;
    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;
}
```

[10 marks]

```
/* code segment 4 */
void multiply(float ** restrict matrix, float * restrict vec, float * restrict 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]

2. Detecting collisions between objects in computer games is an important problem. A common technique is to place each object in a bounding sphere, and check whether the spheres have collided. At any point in time, two spheres have collided if the distance between their centres is less than the sum of their radii. An outline of a simple sphere abstract data type is as follows:

```
struct sphere {
        double centre_x;
        double centre_y;
        double centre_z;
        double radius;
};
```

Write a parallel function using OpenMP and C that takes an array of sphere objects as a parameter, and returns the set of pairs of objects that have collided. Your function should be parallel, and should make efficient use of machine resources on a modern multi-core computer. The prototype of your function should look like:

```
int detect_collisions(struct sphere * spheres, int * collisions, int num_spheres);
```

The parameter spheres is an array of spheres that you need to check against one another for collisions. The number of spheres is num_spheres. The parameter collisions is an array of integers. Each adjacent pair of integers in the collisions array represent the fact that the corresponding spheres in the spheres array have collided. For example, if collisions[0] has the value 10, and collisions[1] has the value 14, it means that the spheres at positions 10 and 14 in the spheres array have collided. The order in which the pairs of colliding items appear in the collisions array is not important. You should assume that collisions are common. Finally, the function should return the number of collisions.

The distance between two points, p and q, with x, y and z coordinates is:

$$\sqrt{(p.x-q.y)^2+(p.y-q.y)^2+(p.z-q.z)^2}$$

[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 your solution, and also explain why you think it is likely to run efficiently on a modern multi-core architecture. [20 marks]

3. The phenomenon, where only part of a processor can be powered at any time has been called *dark silicon*. When processor designers are no longer able to power the entire processor at once, it starts to make sense to put different types of individual core and groups of cores on the chip. For example, it might make sense to have one powerful core to execute sequential programs, and to provide a group of GPU cores for graphics.

The following C code convolves an input computer image with a 3x3 convolution kernel to create an output image:

```
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) Consider the case where you are designing a processor to execute the above code and similar types of code, assuming that the arrays are large. You are trying to select which sort of cores might be most suitable for executing this type of code. Make the case for and/or against each of the following type of cores to include in your processor:

I. out-of-order superscalar cores

II. very long instruction word (VLIW) cores

III. vector processor cores

IV. multithreaded/simultaneous multithreaded cores

V. groups of cores organized as symmetric multiprocessors

VI. groups of cores of the sort found in graphics processing units (GPUs)

[5 marks per type of core]

(b) Make the case for what you regard to be the best mix of different types of cores on a single chip. You should make the argument for the above code, but you should also discuss other types of workloads that you consider to be important.

[20 marks]