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 **Trinity Term 2016**
Junior Sophister Annual Examinations

Concurrent Systems

Monday 23rd May 2016 **Goldsmith 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. Moore's law continues to allow increasing numbers of hardware features to be placed on a single chip, but computer designers need to be increasingly careful of power consumption. It has been argued that in the near future it will no longer be possible to power the whole of a processor chip at once. Instead different parts may be powered up at different times, depending on the programs being run at a particular time. This phenomenon, where much of the chip must be powered off at any time has been called *dark silicon*. Briefly describe some of the key problems that might arise with dark silicon, and how it might affect the number and types of cores and other features on future processors.

[10 marks]

b. 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. The following pseudocode shows the high-level behaviour of the atomic compare-and-swap instruction:

```
int compare_and_swap(int * address, int testval, int newval)
    {
    int oldval;

    oldval = * address;
    if (oldval == testval) {
            *address = newval;
    }
    return old_val;
}
```

Explain how this instruction can be used to implement locks on shared-memory parallel computers.

[20 marks]

(Question 1 continues on next page)...

...(Question 1 continued from previous page)

c. 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;
    }
}
```

[10 marks]

```
/* code segment 2 */
float dot_product(float *restrict a, float *restrict b, int size)
{
    float sum = 0.0;
    for ( int i = 0; i < size; i++ ) {
            sum = sum + a[i] * b[i];
    }
    return sum;
}
```

[10 marks]

2. A common problem is to find cases where the same string appears more than once on huge lists of strings. A function is needed that takes an array of strings as a parameter, and writes to the screen all the strings that appear more than once in the array.

   Write a parallel function (or set of functions) that performs this task using C and either OpenMP or pthreads. 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:

   void print_duplicates(char ** strings, int num_strings)

   In the prototype *strings* is the array of C strings containing the strings among which we are searching for duplicates. The number of items in the strings array is stored in *num_strings*. The order in which the duplicates are written out is not important.

   [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. 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 computes the standard deviation of the items in an array.

```
/* compute standard deviation of numbers in array */
double stdev(const double a[ ], const int size, double mean)
{
    double sum = 0.0;
    for ( int i = 0; i < size, i++ ) {
            double diff = a[i] - mean;
            sum = sum + diff*diff;
    }
    return sqrt(sum/((double) size));
}
```

Identify any potential parallelism in the above code. [10 marks]

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]