UNIVERSITY OF DUBLIN

TRINITY COLLEGE

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

Integrated Computer Science **Trinity Term 2013**
Year 3 Examination

Concurrent Systems 1 (CS3014)

Monday 13th May 2013 Luce Lower (205) 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) Performance per Watt is a measure of the energy efficiency of a given computer architecture. It compares a measurement of performance per unit time (such as floating point operations per second) with the amount of power in Watts. Before 2005 the term was hardly known, but today most processor designers aim to build processors with a high performance per watt ratio. Outline the changes in computer architecture that have happened in the last ten years that have led to an increasing focus on power consumption. To what extent is performance per Watt a good measure of the desirability of a given processor? Are there other measures that a hardware designer should take into account when choosing a processor for a computer system?

[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;
   }
}

/* code segment 2 */
float find_max(float * array, int size)
{
   float max = array[0];
```

```
   for (int i = 1; i < size; i++ ) {
         if ( array[1] > max ) {
               max = array[i];
         }
   }
   return max;
}
```

```
/* code segment 3 */
int mem_compare(char * first, char * second, int size)
{
   for ( int i = 0; i < size; i++ ) {
         if ( first[i] != second[i] ) {
               return 0;
         }
   }
   return 1;
}
```

```
/* 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 the inner loops of the Jacobi method for finding the solution of a square system of linear equations.

```
void jacobi(float ** A, float ** B, int size, int nsteps)
{
for ( int t = 0; t < nsteps; t++ ) {
        for ( int i = 0; i < size; i++ ) {
             for ( int j = 0; j < size; j++ ) {
                  B[i][j] = (A[i][j]+A[i][j-1]+A[i][j+1]
+A[i+1][j]+A[i-1][j])/5.0;
             }
        }
        for ( int i = 0; i < size; i++ ) {
             for ( int j = 0; j < size; j++ ) {
                  A[i][j] = B[i][j];
             }
        }
}
```

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.

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. Deep packet inspection (DPI) is used to inspect the contents of network packets at an inspection point, such as at the entry point to a large organization's network. DPI is typically used to identify non-compliant packets, such as network intrusions or viruses, or occasionally for eavesdropping or censorship. DPI typically involves search for byte patterns in the file, which may be regular expressions. Packets may be scored on the basis of the number and type of matches, and those packets that exceed a threshold may be dropped or stored for offline inspection.

   The dpi function below searches the payload bytes of a packet for pattern match. The index in the payload where the pattern starts is recorded in the array occurrences, and the total score of patterns is returned by the dpi function.

```
double dpi(string payload, vector<pattern> patterns,
                                    vector<int>& occurences)
{
double total_score = 0.0;

for ( int i = 0; i < payload.length( ); i++ ) {
      double this_score;
      this_score = pattern_match(payload, i, patterns) ;
      if ( this_score > 0.0) {
                occurences.pushback(i);
            total_score = total_score + this_score;
      }
}
return total_score;
}
```

Note that the function `pattern_match` checks whether the payload starting at a given index matches some pattern. The exact matching algorithm is complex, and the running time may vary considerably depending on the sequence of bytes starting at that index. The pattern match function returns a score for the match, and if the score is greater than zero, a match is recorded.

Create an efficient parallel version of this function using OpenMP, assuming the payload is large. Your function should correctly identify the set of occurrences, but the order in which they are reported is not important. You can allow your function to use more memory if needed.

[30 marks]

Write a short commentary on your function explaining how it works, and why you believe it to be efficient. In particular you should discuss the parallelization strategy, and any overheads of parallelization, such as synchronization, load balancing, etc.

[20 marks]