INFOMOV 2019 FINAL EXAM – EDUC-THEATRON – 17:00 – 19:00

1. The '*External Memory Model*' assumes that the cost of running an algorithm is proportional to the number of memory blocks that need to be read into the cache during the execution of the algorithm. Describe one scenario in which this assumption is valid, and one scenario in which this assumption is invalid, or at least mostly invalid.

   Valid scenarios are plenty. Invalid scenarios: anything that loads some data and operates on it endlessly. The Fibonacci example that several brought up is excellent. Not acceptable: just making things more expensive (sin/cos/sqrt on a stream of data). In that case things are still proportional to the number of accessed blocks.

2. Consider the following data structure:

```
struct Nuke
{
    float x, y, z;          // position
    bool homing;            // flag
    float vx, vy, vz;       // missile velocity
    bool exploded;          // flag
};
```

   Basic idea: you want to minimize memory cost, i.e. the # of touched cache lines. For random access this means aligning the struct to cache line boundaries (make it 2^N in size). For sequential access this means making it as small as possible.

   Layouts for SIMD processing also yielded full points for sequential access.

   Rewrite this structure twice: once for efficient random access of a large array of Nukes, and once for efficient sequential access of a large number of Nukes in a single, continuous array. Motivate your layouts.

   Some pitfalls for Q2: a float is automatically aligned to 4 bytes. A cacheline is not 128bit. A bool is not 1 bit.

3. Some CPU architecture questions, based on "Modern Microprocessors - a 90 minute guide":

   In 30 words or less,

   a) explain what a *pipeline latch* is.
   b) explain what a *bypass* is.
   c) explain what *speculative execution* is.
   d) explain what *predication* is.

   Skipped by most of you, but easily extracted from the document. Quite interesting actually. Still a recommended read. 😉

4. The Core i7-8700K processor uses 2x6MB of 16-way set-associative L3 cache. AMD's K10 uses 6MB of 48-way set-associative cache.

   a) Under what circumstances is 48-way better than 16-way, and when and/or how is 16-way better?
   b) Both processors use much lower set associativety for L1 and L2. Why do you think this is?

   The whole class is cache expert, but for completeness:

   48-way is closer to fully-associative and reduces collisions better than 16-way. It is also more costly in terms of die space, so that goes at the expense of other things. Also, more complex schemes tend to be slower, which applies here: Intel has the faster cache. Also: beyond 16-wide we get diminishing returns, and 48-way (even 64-way was attempted!) is over the top.

5. The following code uses fixed point arithmetic:

```
uint ScaleColor( const uint c, const uint x )
{
   uint redblue = c & 0x00ff00ff, green = c & 0x0000ff00;
   redblue = (redblue * x) & 0xff00ff00;
   green = (green * x) & 0x00ff0000;
   return (redblue + green) >> 8;
}
```

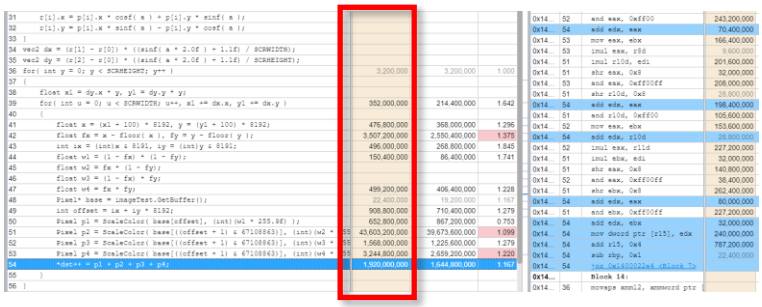
Most of you knew that blue bleeds into green if we don't take measures.

For 'b' the answer is 256, which I didn't expect myself (I thought it was simply not possible and used 255 as the best alternative).

This made the 'trick question' rather easy. ☺

a) Explain why the three color components are not multiplied by x using a single multiplication.
b) Which value of x should we use to get the unscaled color, i.e. 100% of input value c?

6. Consider the following VTune profiling result for the 'rotozooming' example:



Also tricky. I *thought* 45 and 46 took little time and got overlooked by the random sampling of the profiler. This answer is still valid, as this could be the case based on the info in the image.

It is however also possible that the functionality is implemented as part of other asm lines, which point back to lines that are not 45 and 46.

The most plausible reason is that the compiler optimizes the lines away: the multiplication results are probably already in registers and are simply re-used.

Not accepted: the code fills up a latency of an expensive line before it and therefore takes 0 cycles. That may be true, but then the profiler could still sample in the cycles that make up the latency, which is not happening. Also, the lines would not be taking 0 cycles. Instead, the latency would appear smaller.

a) The first column to the right of the source code (in the red box) indicates the number of clock cycles spent on each line, according to the profiler. Apparently, line 45 and 46 did not take any cycles at all, but that seems unlikely. Explain what is going on.
b) Some lines, such as line 50, are represented by a continuous block of assembler instructions. Other lines, such as line 54 (highlighted in blue), are compiled to scattered assembler instructions. Why is this?

7. "A just-in-time (JIT) compiler is a form of self-modifying code." Do you feel this statement is correct? Why or why not? (your answer should demonstrate that you understand the concept of self-modifying code). Not a great question. It assumes that you know what 'JIT' is, and if you do, it requires minimal understanding of the concept of SMC. Free points for most of you.

8. Sometimes a program becomes significantly *slower* when modified from single-threaded to multi-threaded execution, due to *false sharing*. Write down a simple case where this happens. Heavy pseudo code is allowed (within reason), as long as your intention is clear.

(Almost) everyone was able to come up with a simple example, often based on P3 experience.