INFOMOV 2016/2017 EXAM - November 8 – 17.00 - 19.00 – EDUC-ALFA

Answer these questions as elaborate as necessary. Don't be too elaborate; incorrect statements in your answer reduce your score. Negative scores for a question are not possible however.

1. A modern CPU uses a pipeline to process a sequence of instructions, inspired by the 'fetch-decode-execute-write back' sequence. (15 pts)

   a) Why does a typical modern CPU have far more stages than just the original four?

Because not every stage requires the same amount of effort. More stages allow for a more even distribution of work, keeping all stages at work all the time.

   b) Name a disadvantage of having many stages in the pipeline of a CPU.

Several are possible, e.g.: branch mispredictions have greater consequences, dependent instructions must have more instructions between them.

   c) What is, in the context of the CPU instruction pipeline, a 'bubble'?

Inactivity in the pipeline, due to e.g. dependencies.

   d) How is a 'superscalar' pipeline related to instruction level parallelism?

A superscalar pipeline executes instructions that use different execution units in the same clock tick, and thus in parallel.

   e) How does a compiler help a superscalar processor to run at maximum efficiency?

It reorders instructions to maximize the space between dependent instructions. It also reorders to get instructions close together that use different execution units.

2. Your code contains the following snippet:

```
float z = table[20];
a /= z;
b /= z;
c /= z;
d /= z;
e += z;
```

   When inspecting the disassembly, you notice the compiler didn't replace the four division by four multiplications and a reciprocal. Why did it not apply this optimization? (10 pts)

Multiplying by the reciprocal is not as accurate as dividing directly. It can't assume that this is tolerable, and thus does not apply the optimization. You should therefore do this manually.

3. "Going from 4-wide (SSE) to 8-wide (AVX) SIMD and beyond shows diminishing returns." Is this true or false? Explain your answer. (10 pts)

'True' or 'false' are both possible. Using auto-vectorization we typically see diminishing returns. When using SoA data layouts, we benefit from wider hardware automatically, although small problems may still suffer from diminishing returns.

4. The way a GPU runs multiple warps on a single shading multiprocessor is similar to how CPUs perform hyperthreading. Why, do you think, does a CPU only run two threads per hyperthreaded core, while some GPUs can run up to 64 warps per SM? (10 pts)

Hard to say, I'm looking forward to your reasoning. Possible reasons: CPU cores are much more complex, and therefore harder to replicate. CPU cores use caches effectively and the small gains of hyperthreading are likely to show diminishing returns rapidly, therefore the extra die space for more hyperthreading is not worth it.

5. Explain the following concepts in 30 words or less. (15 pts)
   a) False sharing Two cores accessing the same cacheline, forcing L1 synchronization.
   b) Prefetching Manually getting data in the caches in advance of actual use.
   c) Bélády's algorithm Hypothetical optimal caching scheme that keeps data that will be reused soonest (is that English? ☺). Also: 'clairevoyance algorithm'.

6. Most reads and writes from C++ code are 4 or 8 bytes in size. Nevertheless, CPU caches typically use 64-byte cache lines. Why? (10 pts)

This is assuming sequential reads. This way, subsequent data will already be in the cache.

7. Certain AMD processors use a 48-way set associative L3 cache, while Intel uses 16-way. What reasons could each vendor have for this particular configuration? (10 pts)

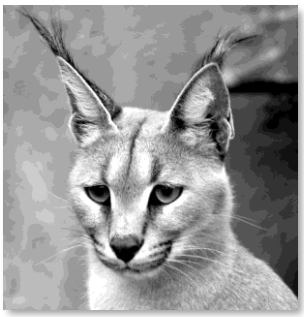
Hard to say, so again I'm looking forward to sensible reasoning. Possible answer: a 48-way cache is closer to the fully associative cache, which is the optimal scheme - as any address can go anywhere in the cache - but expensive to implement. Likely a lot of die space is used to implement this. 16-way is probably a cheaper alternative, where more die space is available for the actual memory.

8. Explain how compaction can help to improve occupancy for GPGPU algorithms that have complex flow control. (10 pts)

Control flow: if, for, while, etc.: leads to inactive streams. Compaction allows us to fill arrays with task id's so a subsequent kernel invocation can start with full occupancy.

Good luck!

It was a pleasure teaching INFOMOV.



This is a Caracal.