~~None~~ Some of this matters

Getting to know the hardware via D, and knowing when to care

Computers are important

- Our jobs & tools
- Do we need to understand them? Depends
- Intuitively a little might be very helpful.
- It's interesting

So how much

- A couple of key takeaways are enough to get out of some tricky spots
- For the experienced programmer this probably means performance
- Could just be semantics for those less experienced with low-level code: Hardwa provides an API.
- My emphasis is mainly on the former.
- This talk is mostly things I didn't mention in DConf online

Processor language

- Compile hello world

```
void main()
{
    import std.stdio;
    writeln("Hello World")
}
```

- The compiler uses something (magic) to turn the code into something we can ru
- We run the something
- The CPU gets pointed some data, the computer prints the text. How?

The processor has a language to tell it what to do

- The statements in this language are called instructions
- The CPU (obviously) sees this as bits and bytes e.g.

```
0F 01 F9
```

- For humans we can represent them textually. Above instruction becomes

```
rdtscp
```

- All of the instructions combined form an instruction *set* - ISA: Instruction set architecture.
- It's an API: Roy's talk, and see also "Is floating-point broken" on stack overflow

Pareto-ish

- Most modern processors speak a broadly similar language.
- Need to be able to branch, do arithmetic, do memory loads and stores.
- That's a handful of instructions, for reasons beyond this slide in reality the ISA m have a 50 to ~200 (apparently 1000 for x86, hard to count)

What's for sale

- Desktop and servers: X86
- Mobile and embedded, now finally the above categories too: Arm
- The new-ish player: RISC-V
- WASM?

Your favourite instruction might be missing

- Cheap(er) processors might not have an instruction you want.
- RISC-V processors may not have an integer divide (especially microcontrollers) example.
- DEC Alpha had no integer divide
- On desktops the missing instruction is probably some SIMD instruciton or other acceleration
- Worth keeping in mind.

The space has got simpler

- "Clever" architectures have mostly been killed off for various reasons
- Intel tried multiple times to be clever (iAPX 432, Itanium)

- iAPX 432 - really interesting (object-oriented, garbage collection in hardware), apparently a flop (before my time)
- Itanium, died last year. Very different in ways beyond this talk.

> We had a good team on paper. Unfortunately, the game was played on grass. (Brian Clough)

- "Worse is better"? Wrong question
- A hidden assumption throughout this talk: Virtual memory with flat address spac Lots of now-dead processors and mainframes did avoid that.

Ripples in the sand - patterns in typical code.

- These patterns are a bound on how much performance we can get for free, estir of what that bound is vary a lot.
- Not much code in between different pieces of control flow (a handful of instructi
- We spend lots of time in loops. Especially "classical" programs like signal proces and things like that
- Memory: The instructions are in memory, we jump to memory, we read memory, write to memory.

Again for emphasis: It's the memory stupid

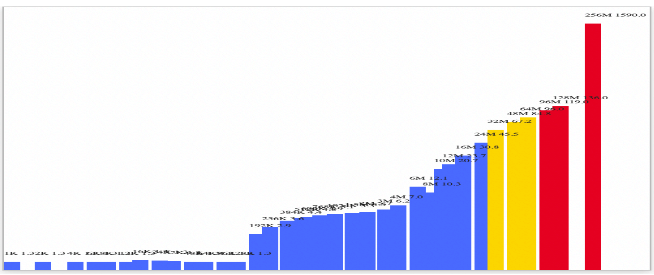
- "It's the memory stupid" is a famous article by Richard Sites from '96

> Across the industry, today's chips are largely able to execute code faster than we can feed them with instructions and data

- Emphasis on "today" - *today* is even worse.
- Takeaway? Memory performance is performance unless you can prove otherwis

Peeking ahead: Memory latency curves

- Graph of memory latency versus size of working set (~memory currently in play
- `https://chipsandcheese.com/memory-latency-test/` run in your browser
- very clear regions where we want to be.
- the worst levels of memory latency are truly terrible.
- hint that there are multiple layers of cache.



Ideal caches

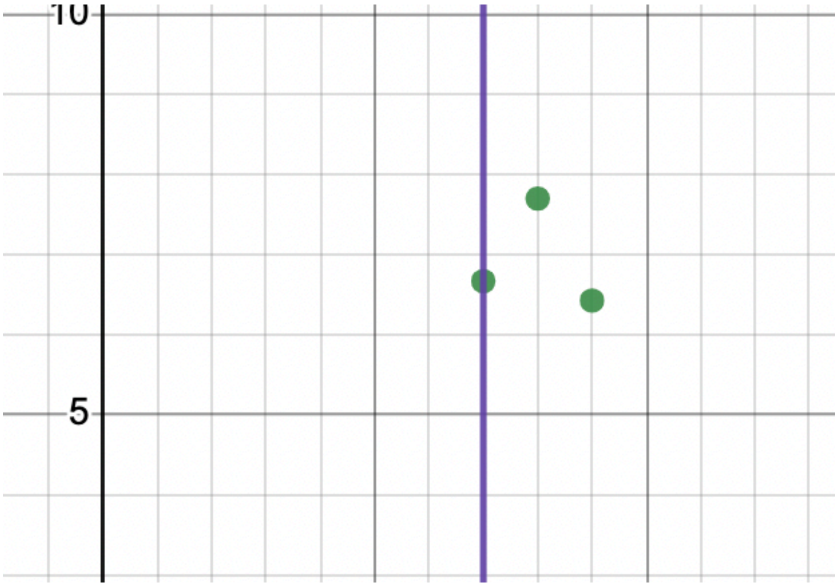
- To keep our ideal CPU fed with instructions and data we'd like
- Infinite capacity
- Infinite bandwidth
- No latency
- Persistence
- Low cost

- Some of these are obviously incompatible (we still have memory *and* disk storag rather one of the two)
- We can get surprisingly close: Approx 3 to 20 cycles up to 12MB on my laptop (s thread)
- Ideal caches are a route to defining certain types of misses

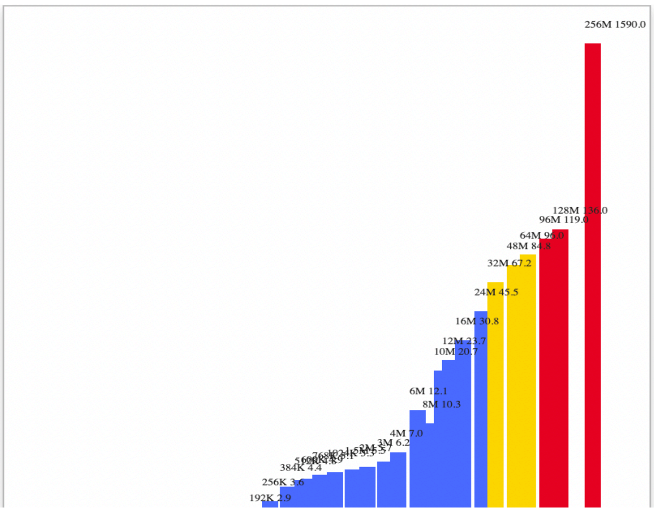
Caching side effects

- Mainly for fun but we can derive good praxis from them
- If you fall asleep now just remember: locality.
- Demonstrate what can be demonstrated although CPU's are explicitly trying to a these having any measurable affect on simple cases.

- If we mutate across a large array in different strides a pattern emerges
- The time per access varies signficantly with the stride
- We hit a stride, time per access suddenly increases *a lot* because we are now (it
out) hitting more than one cache line per access.



- Saw this earlier.
- Very clear jumps in the cost of hitting memory when the amount of memory (handwaving) on the go exceed a given level.
- Some bumps and wiggles in the graph are due to implementation of virtual mem
apparently.
- The levels have their own sub-details e.g. sharing between cores.



Prefetching

- CPU is constantly looking for patterns in the instructions.
- If it thinks it has a pattern, it'll start fetching things into the cache early.
- `core.simd` has `prefetch`.

Virtual Memory

- Every memory access in our code has to be translated into a physical address. T sets out where the address will map to.
- More precisely: Pages, page tables. Details are for a different talk.
- This gets expensive.

Old time rock and roll - Page sizes

- Pages are still 4 KiB by default.
- Covering the address space of a modern phone is millions of pages
- Hugepages are a thing (usually automatic, but worth knowing about since you c for them)

Caching this

- Having the CPU constantly chasing walking page tables is really expensive.
- Cache the mappings in a TLB: Translation lookaside buffer
- First ever modern cache in a computer was motivated by this (Atlas 2, I think)

The textbook cache-aware transformation - AOS -
SOA

```
struct S {
    double x;
    double y;
    Chimpan z;
}
```

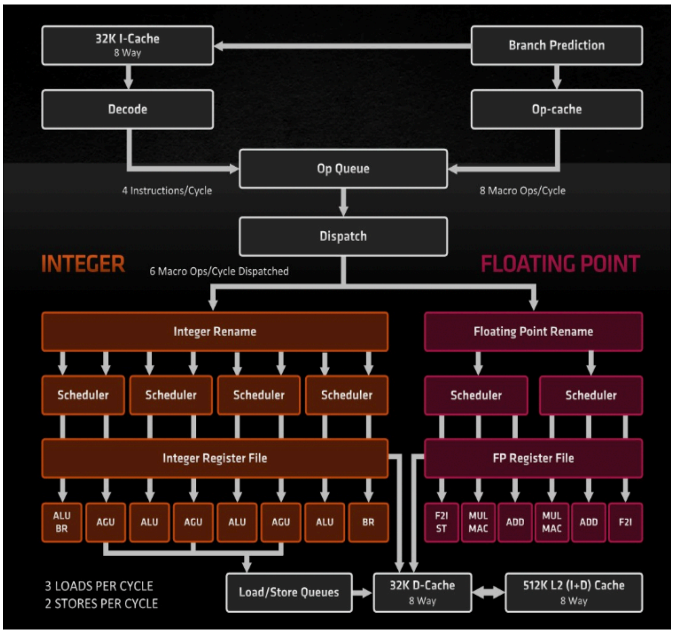
If we are going to access each one in their own burst of accesses, transform to

```
struct A {
    double[] z;
    double[] y;
    Chimpan[] z;
}
```

This can be *extremely* profitable because we can now write to 100% of each cache li
time.

- Concept can be applied much more generally, lay out memory how it's going to b

- As with almost everything else, CPU's have a frontend and a backend.
- Physical register file is much bigger than the architectural registers
- Restricted dataflow. Surprisingly unimportant relative to memory.



The spice must flow - branch prediction

- Average length of a basic block is very small relative to overall code
- Those execution pipes need to be kept busy - the CPU speculatively does work on educated guesswork.
- Actually it speculates everything until it sees things that it can't (not much)
- Conditional branches are fairly easy to predict.
- Static branch prediction gets you ~75% accuracy due to patterns in code.
- Static branch prediction isn't really a thing anymore, don't reorder code based o pipelines that only exist in someones head.
- Top end *dynamic* branch prediction is getting very close to 100% on numerical programs
- AMD use something which is a littttle bit like a neural network.

Predicting indirection

- interfaces, function pointers etc.

```
interface I {
    void foo();
}
```

- CPU has buffers to predict from history where things might jump to (BTBs, more one of them)
- These are much harder to deal with so you can make gains by (again) improving locality.

Oops - spectre, meltdown

- Memory is everything.
- CPU is speculating -> Speculative memory accesses.
- Interaction of this and caches lead to ability to extract information from code tha never "runs".
- Meltdown was a result of Intel forgetting/choosing to check page table. Oops^2

Interesting stuff happening again

- AI workloads are breeding new computer designs, new processors entirely (TPU instructions
- AVX-VNNI, apple extensions: all available from D
- I didn't mention GPUs :(DCompute