TDT4200 Parallel Computing

Bart van Blokland

Welcome to TDT4200 parallel computing!

Since people seem to like to use lecture slides to study for exams, but I personally find those to be unusable for the lectures themselves, I figured I'd use the notes feature to give the best of both worlds instead.

This first lecture is closely following the cookbook on course creation, as it is primarily trying to convey what this course is about.

Mandatory boring bits:

Announcements

Course Overview

Survey:

33 / 88 can’t come on Mondays 12 - 14

14 / 88 can’t come on Wednesdays 12 - 13

30 / 88 can’t come on Thursdays 12 - 14

Weekly lab sessions:

Wednesdays 12:00 – 14:00 +

ITS-015 “Tulipan”

(“Krokus” on ntnu.no/kart)

(might be starting at 13:00 if we use the wednesday lecture period)

Crash Course C++

Today and Thursday (recitation)

Weekly lecture:

Mondays 12:15 – 14:00 in F2

Doesn’t look like this can be changed :(

Final grade:

5 Assignments (25%)

Final Exam (75%)

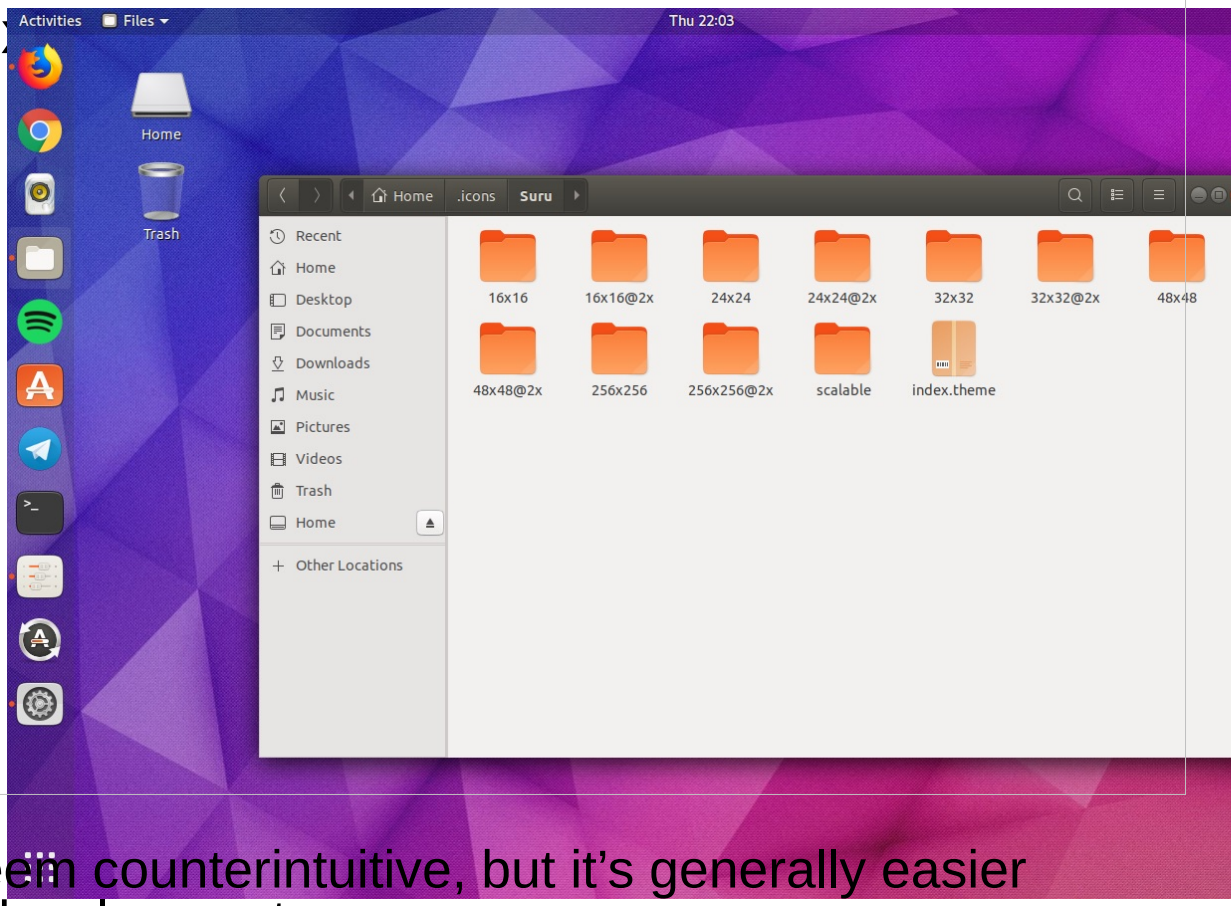
Assignment Schedule (Tentative)

Deadline	Topic
N/A	Assignment 0 (Optional): C Intro
14.09	Assignment 1: Optimisation and Profiling (out on Friday)
28.09	Assignment 2: MPI
12.10	Assignment 3: OpenMP & Pthreads
26.10	Assignment 4: CUDA Basics
09.11	Assignment 5: More on CUDA

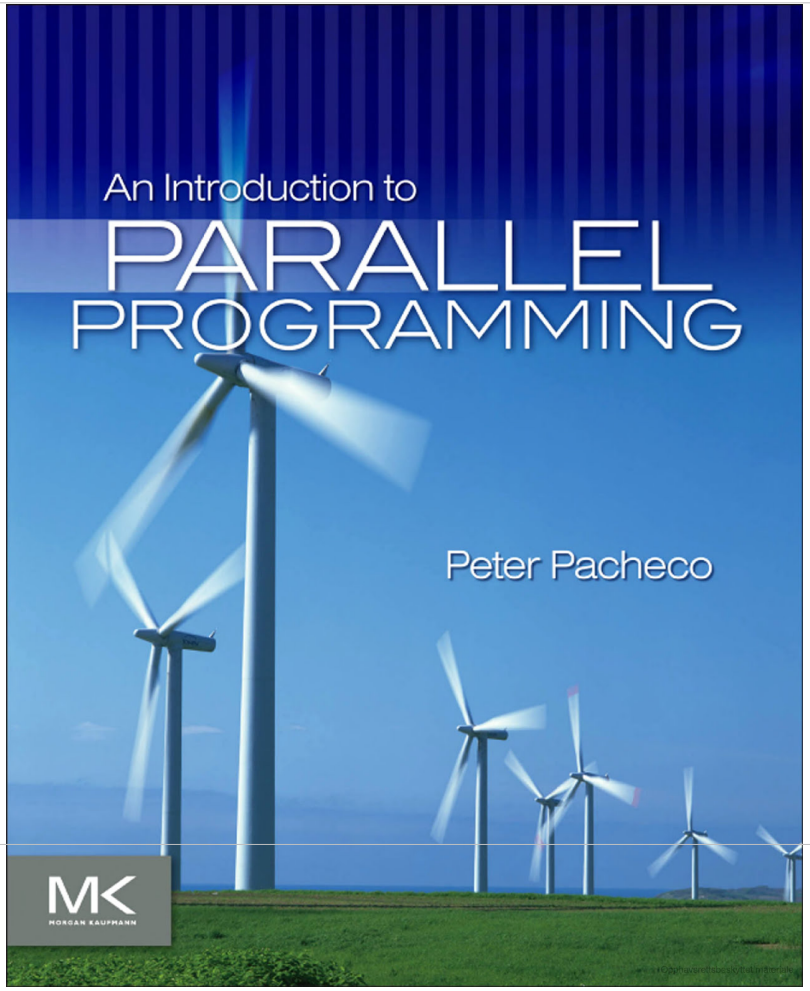
Lecture Schedule (Tentative)

Date	Topic
27.08	Introduction
03.09	Measuring Performance
10.09	Optimising Single Threaded Code
17.09	(Guest Lecture by Jan Christian Meyer) Introduction to MPI
24.09	(Guest Lecture by Anne Elster) More on MPI
01.10	POSIX threads
08.10	OpenMP
15.10	Introduction to CUDA
22.10	CUDA
29.10	CUDA: Warp cooperation
05.11	CUDA: Profiling tools
12.11	OpenCL
19.11	Exam Q&A?

Linu



Might seem counterintuitive, but it's generally easier to do development on.


An Introduction to
PARALLEL
PROGRAMMING
Peter Pacheco
MK
MORGAN KAUFMANN

</mandatory>

Parallel Programming:
Serial Killer?

Now that the mandatory bits of information are out of the way, we can move on to the topic of the day.

This is primarily about what parallel programming is, and why I think it is relevant for you.

What is it (about)?

Why do we care?

What: both in the sense of “what is the course about”, as well as “what is parallel computing”

I’m going to assume you at least have some idea what it is.

15 44 5

There are three numbers on this slide. Simple question: what is their sum?

On the next slide I have put more numbers. I’d like you to work together to compute the sum of these.

Does not make sense to do in parallel: can do this on your own

21	28	12
4	15	28
27	9	2
28	1	4
8	12	19
15	20	8
14		24
21		

You divided the work, communicated, put the answer together. This is a parallel program.

Why did we not divide the work for the previous slide?

Need to communicate to divide work, compute final answer.

In this case the overhead is worth it

More data → More parallelism

Generally, the more data you have, the more parallelism you can use to process it. This is one of the main drivers for parallel computing and HPC today.

Parallel Computing:

+ Faster

But it’s a balance.

More data means you can do more processing in parallel, but doing so also increases the three main counterweights that slow it down.

Parallel Computing:

+ Faster

And it really is big. It usually outweighs the downsides, otherwise there would be no point to it.

Parallel Computing:

+ Faster

- Communication

But it comes at the cost of the need for communication (as we saw previously)

Parallel Computing:

+ Faster

- Communication

- Synchronisation

We also needed to synchronise (wait for each other until you could complete the computation)

Parallel Computing:

+ Faster

- Communication

- Synchronisation

- Load balancing

And you needed a way to divide the work evenly.

Now is this balance what this course is about? Only in part.

MPI

OpenMP

Pthreads

CUDA

OpenCL

Is it perhaps about these libraries I showed you previously? Not really either.

Sure, you need to know all of them for the exam

But what matters here is not the specific library used, but rather the skill of being able to map a problem onto parallel hardware.

But can’t we do that automatically?

Performance?

Is that not the compiler’s job?

No, not really.

Performance?

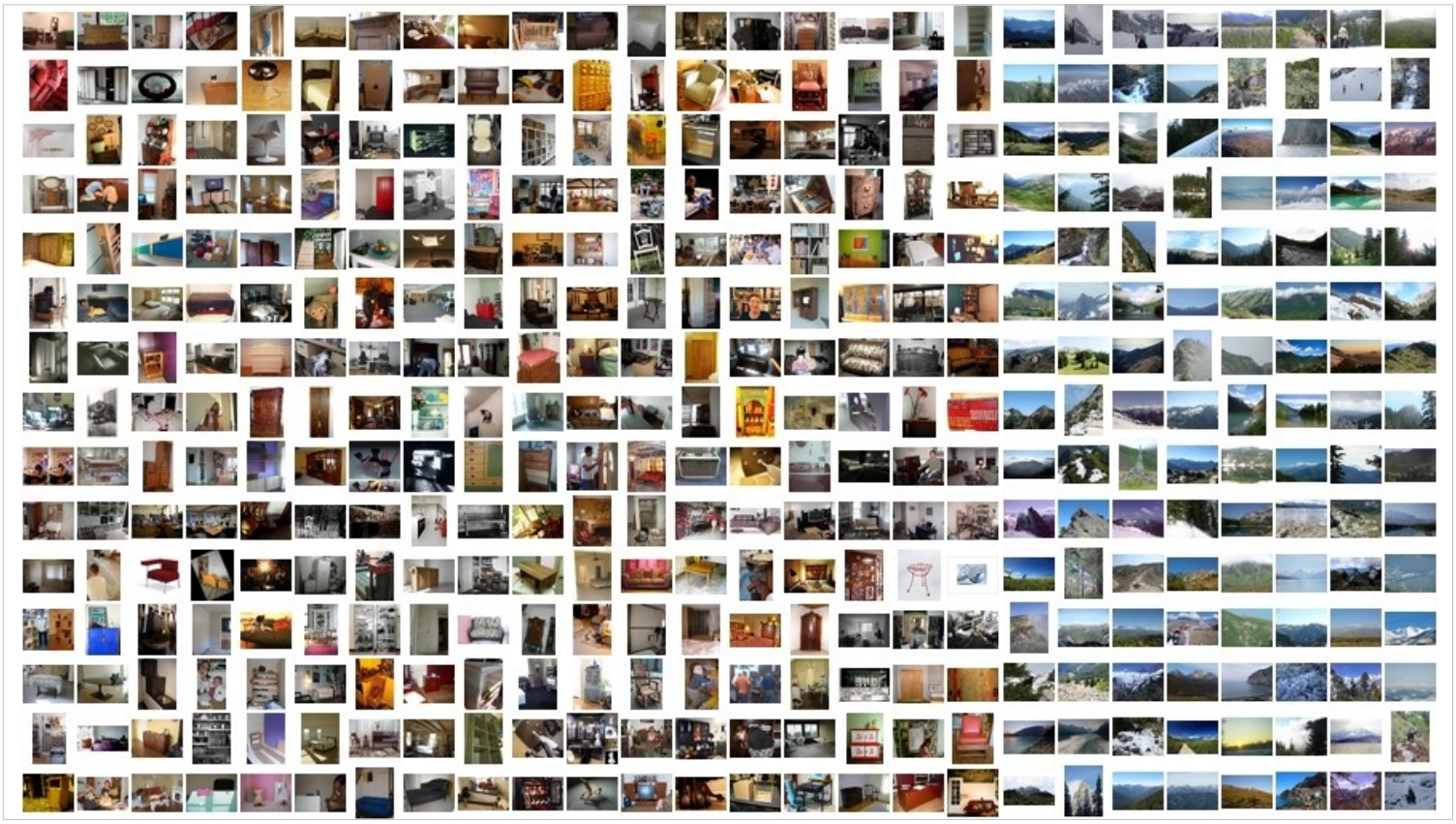
Is that not the compiler’s job?

Course plug: TDT4205

Can’t we just throw a bigger, better processor at the problem?

What is it used for?

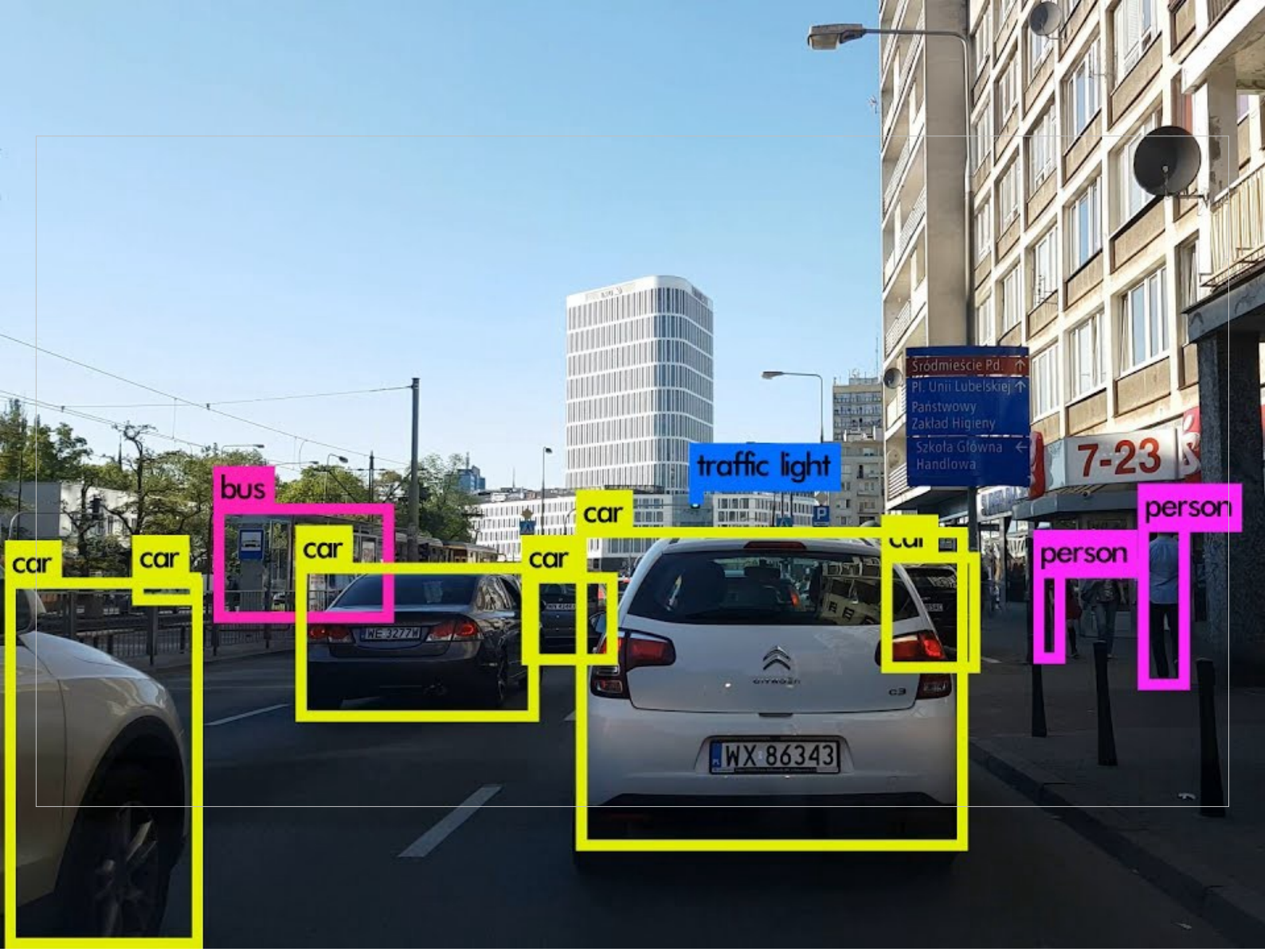
Let’s get back on track. I want to talk a bit about what parallel computing is used for nowadays.

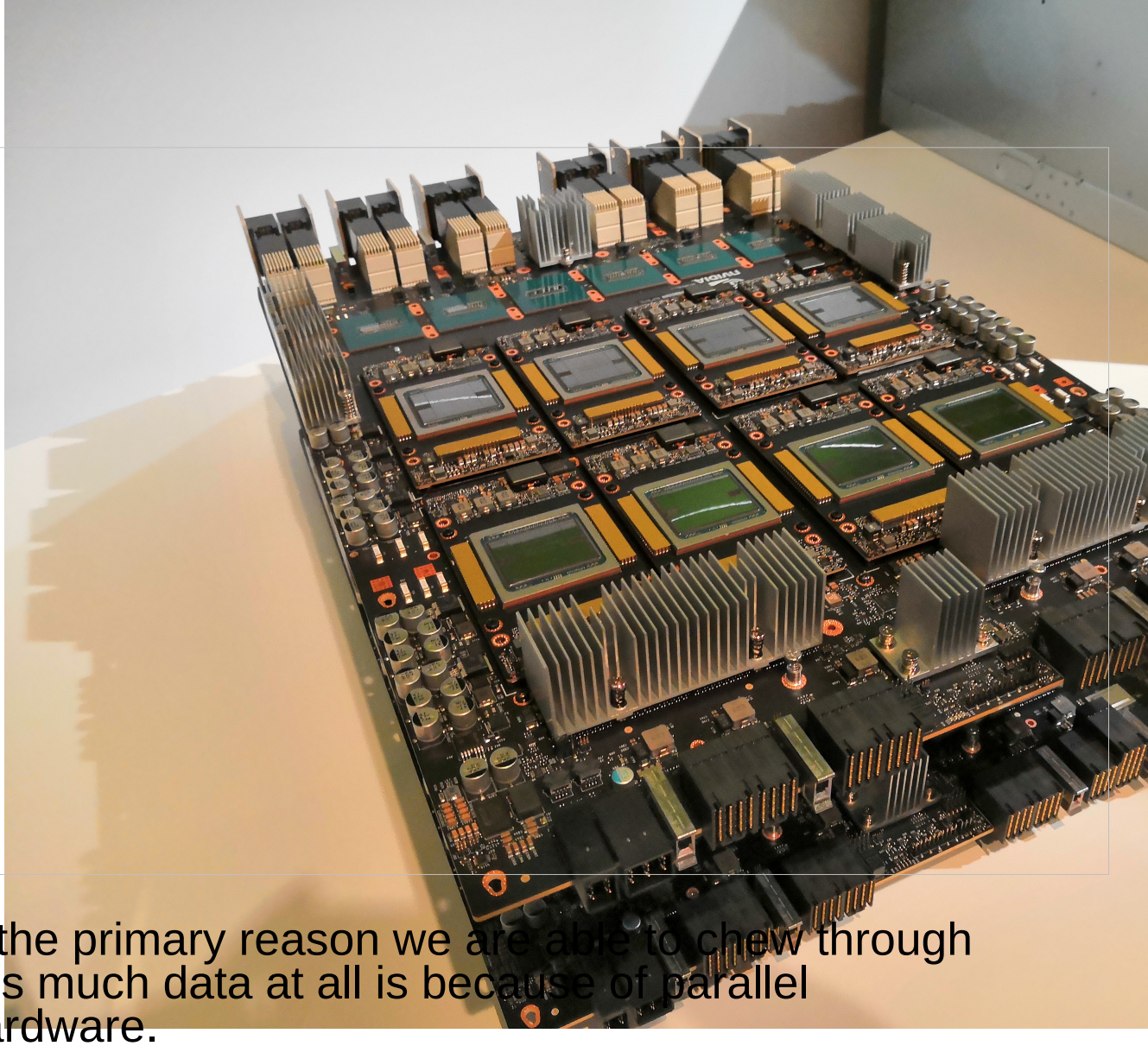


The internet, for one

Imagenet: 14.1M images

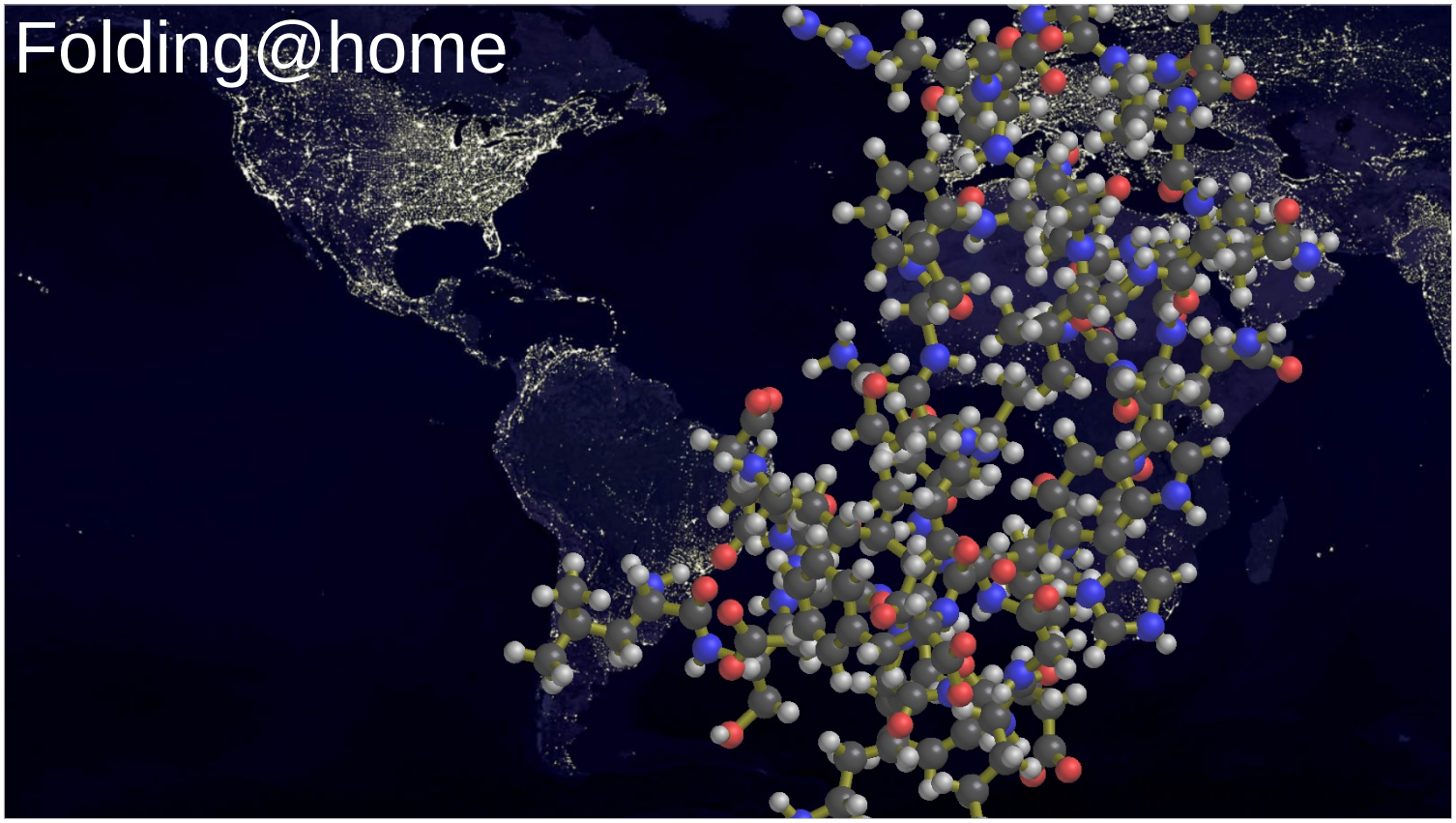
There’s an entire internet of data out there, and the business models from the tech giants are all based on harvesting and interpreting it.


car
car
bus
car
car
traffic light
car
car
person
person
Śródmieście Pd.
Pl. Unii Lubelskiej
Państwowy
Zakład Higieny
Szkoła Główna
Handlowa
7-23
WX 86343

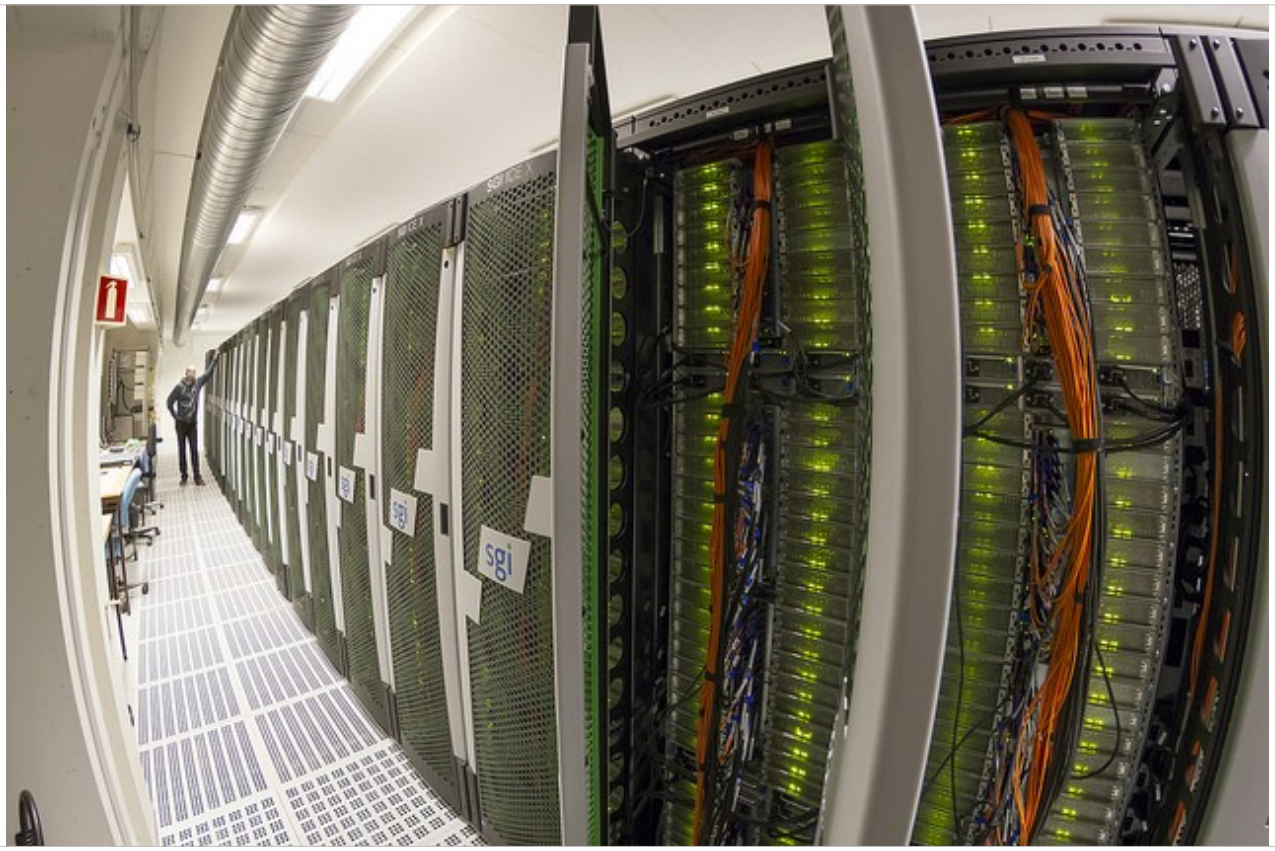


But the primary reason we are able to chew through this much data at all is because of parallel hardware.

One of Google’s datacenters



When it comes to research, a very notable example is the folding@home project.



This is Vilje, one of the cluster machines from the HPC group at NTNU.

Another good example of HPC:

Computer Grahics

I’d like to show you something from my own field:
computer graphics

But at the same time, also show what modern hardware is capable of.

11 300 000 000 000 FLOPS

Here’s a modern graphics card. Its performance has been rated at 11 300 billion floating point operations per second

Much like putting your car in no gear and hitting the throttle

The world’s best supercomputer in 2002

Would have been in the top 500 until 2009

DOOM

But anyway, I want to talk about doom, death, and destruction.

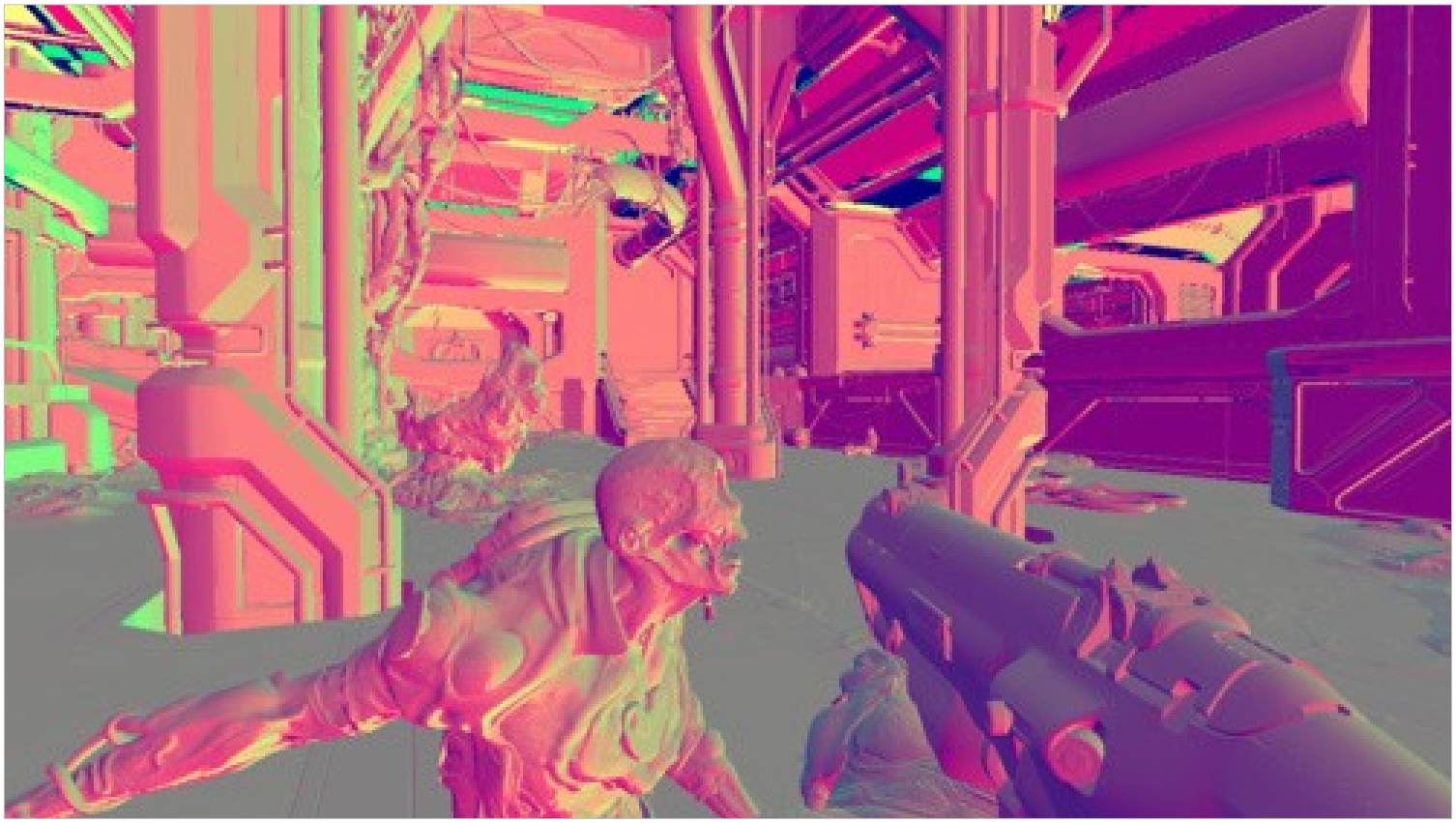
For an excellent writeup of this frame’s graphics breakdown, see here:

http://www.adriancourreges.com/blog/2016/09/09/doom-2016-graphics-study/

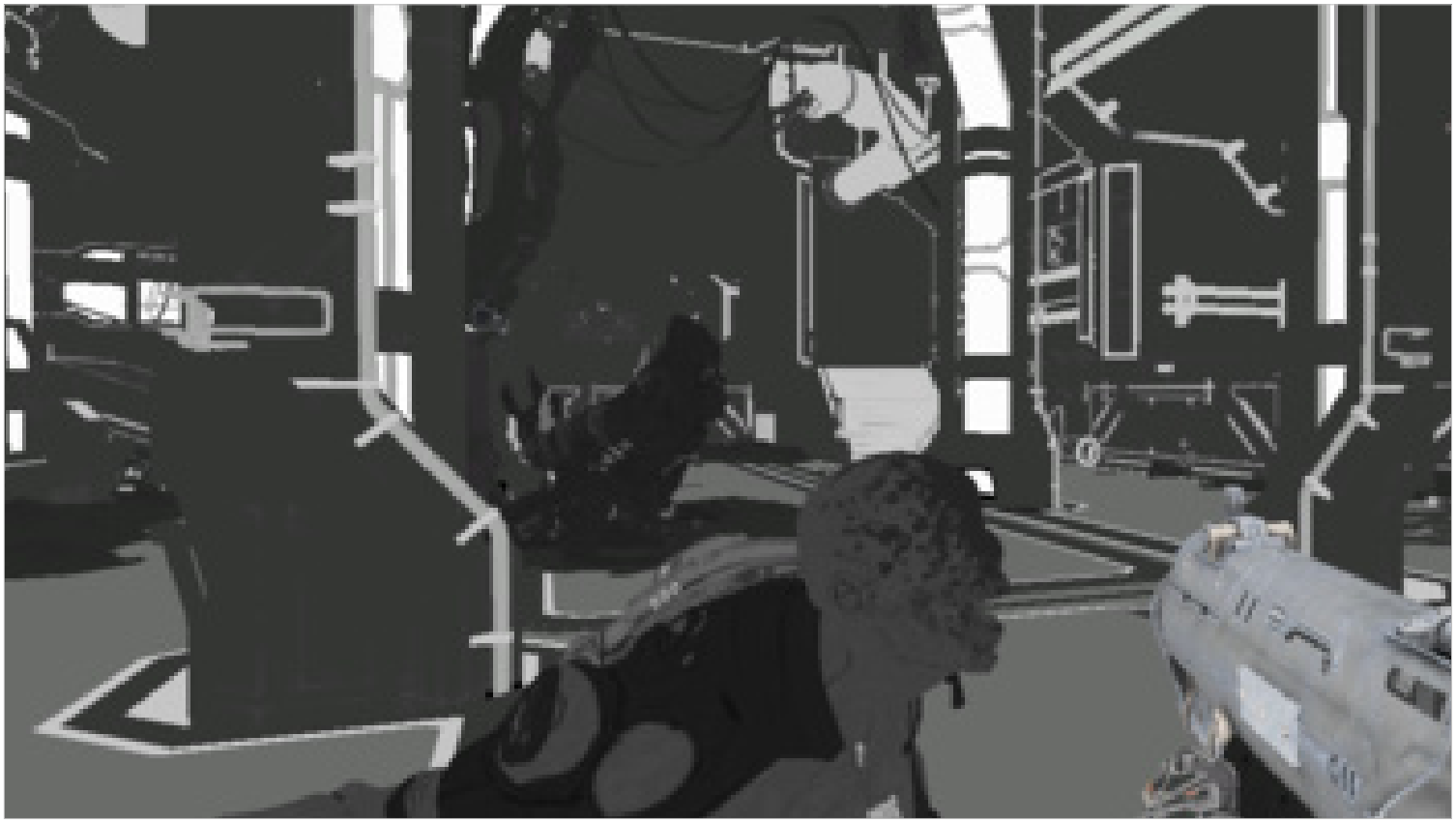
Depth buffer computation

Shadow map rendering

Velocity map

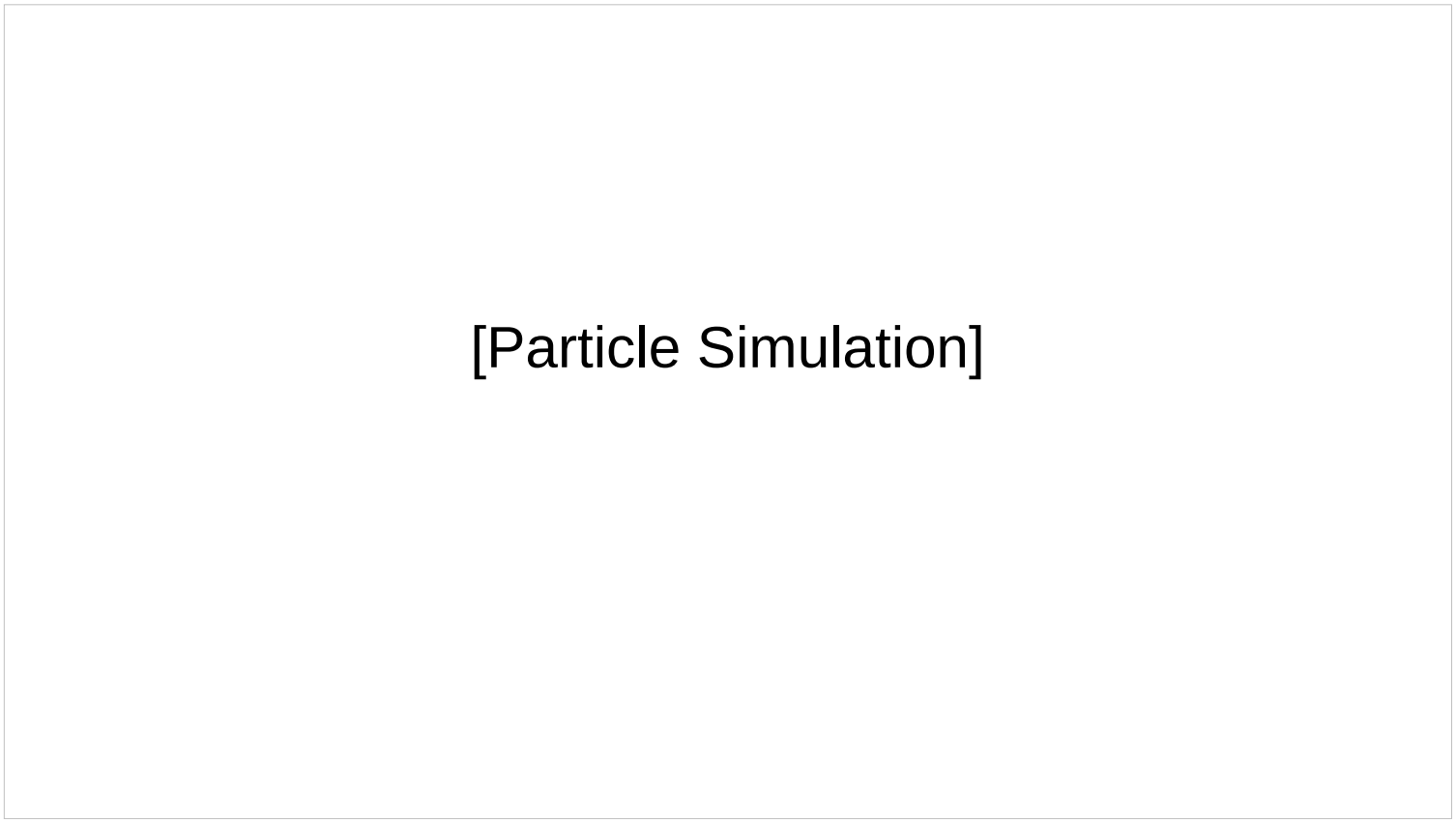


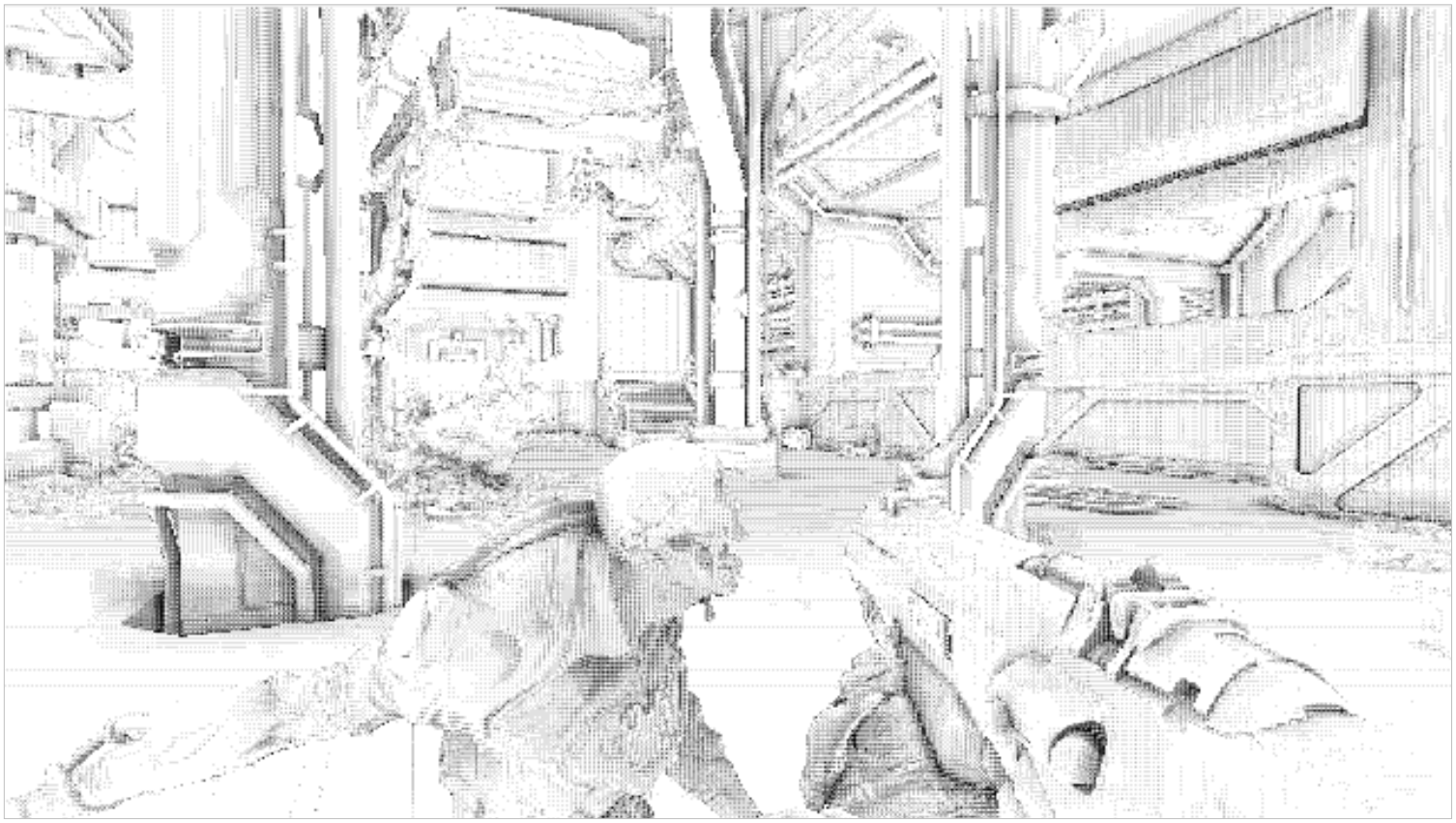
Normal map



Specular map

Lighting


[Particle Simulation]



SSAO

Screenspace reflections

Cubemaps are used to compute reflections on static meshes

Putting together what we have thus far

Fog and blending

Downscaling the rendered image for post-processing effects

Transparent objects and particles

FXAA

Pre-tonemapping

post-tonemapping

HUD

GTX 1080 Ti:

4K Resolution (3840 x 2160)

82 frames per second (average)

8.3 million pixels

A piece of silicon that fits in the palm of your hand

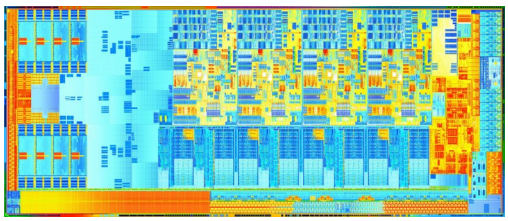
Imagine a datacenter full of these

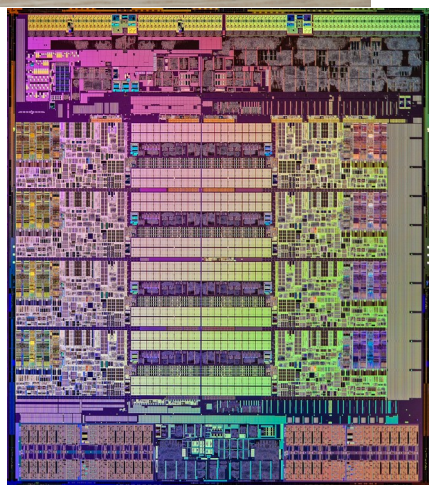
Raises the question: what are the limits?

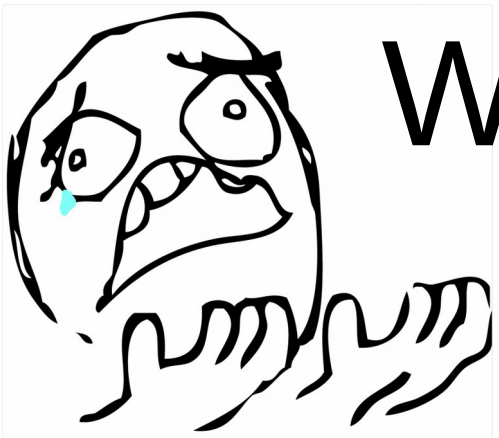
AND TOGETHER WE WILL RULE THE PARALLEL UNIVERSE!

On modern hardware, it is the only way to reach its full potential.

Look around you!







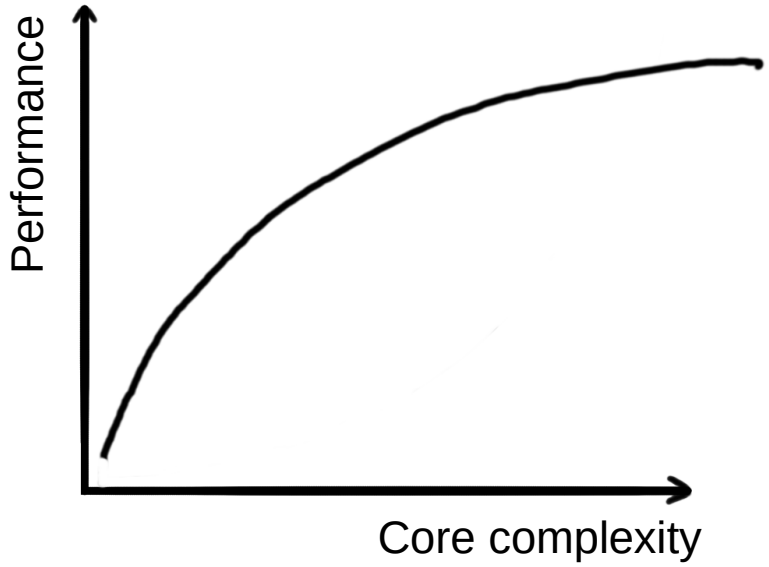
Why?

But after all we've seen, we still don't know why we write parallel code at all?

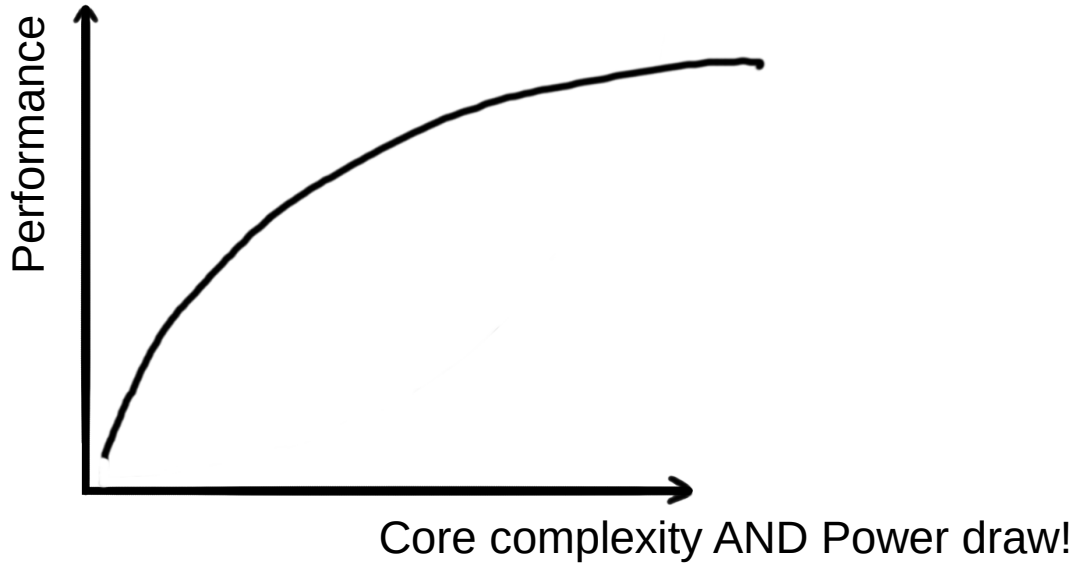
Why don't we just invent a better single-core processor?

Why do we waste so much die space on more of the same?

(I think you can see where this is going..)

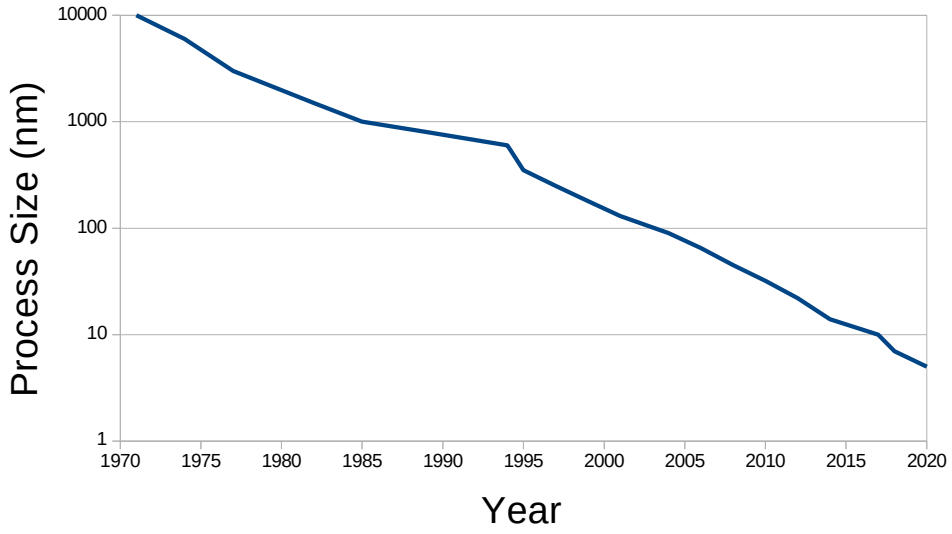


The problem is one we'll also be seeing a lot in single threaded optimisation: the law of diminishing returns.



Which in the case of integrated circuits also means higher power draw.

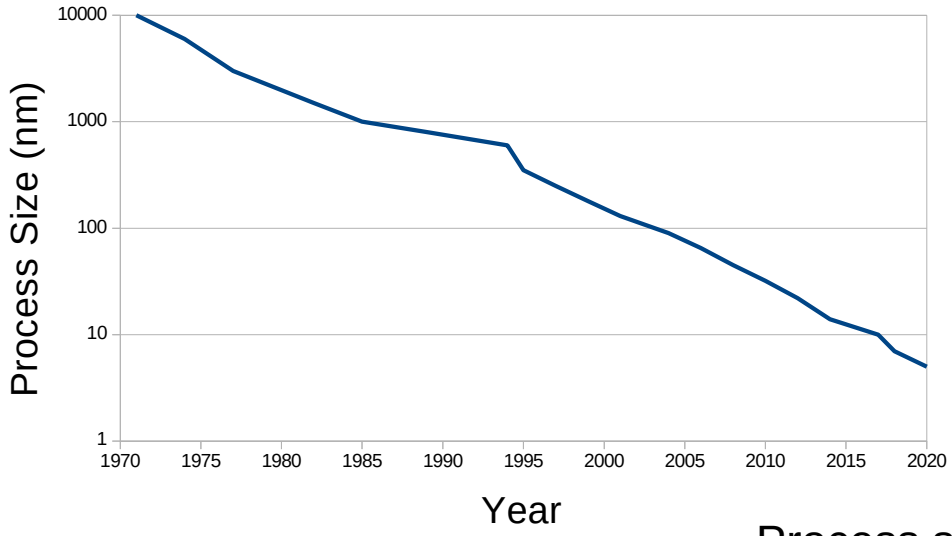
Die Shrink



The primary way out for the industry thus far has been to shrink the process size.

Die shrink is a measure of the size of transistors on an integrated circuit. Each such component requires a number of atoms.

Die Shrink

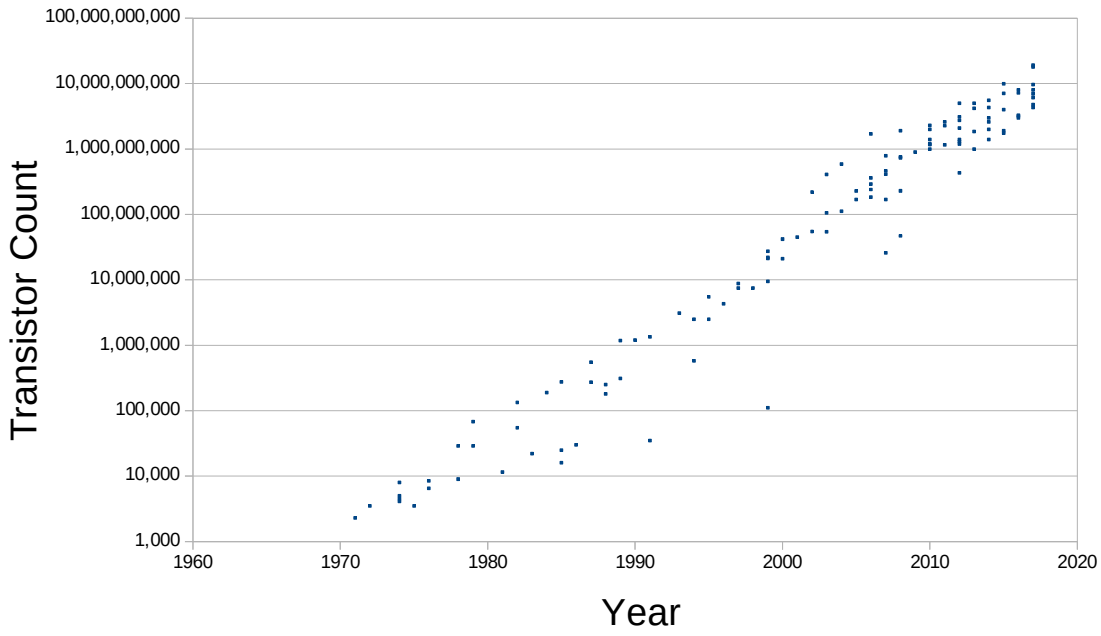


Silicon Atom radius: 0.2 nm

Process size: distance between transistors

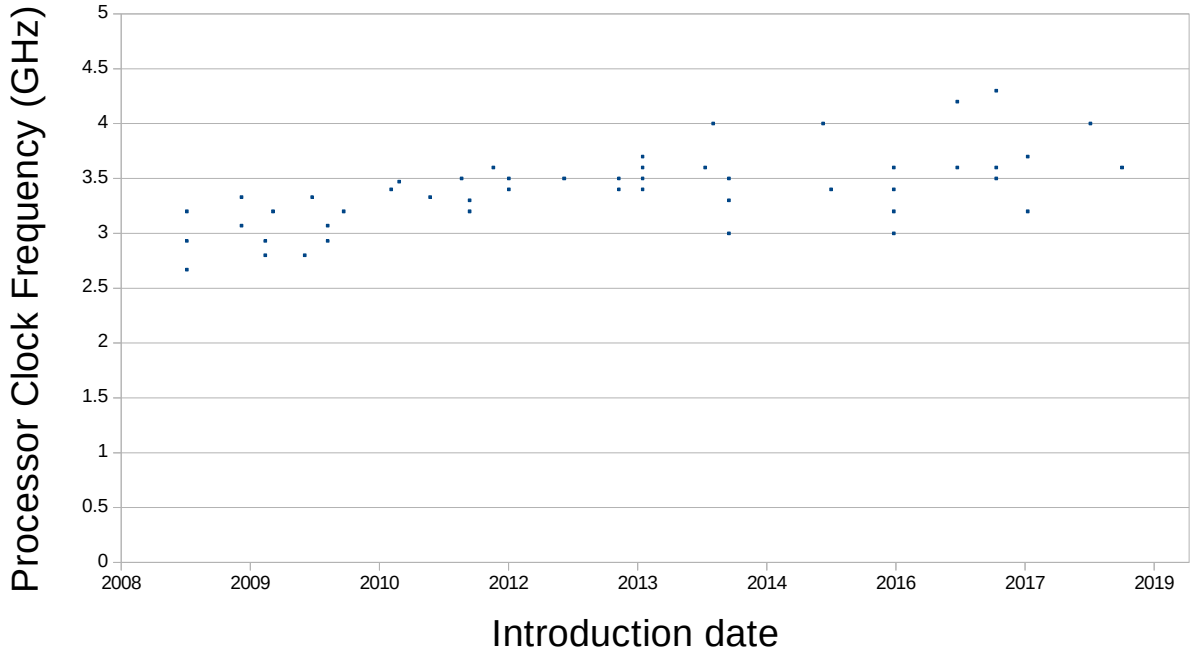
But as you can see we’re approaching the physical limits, because we’d need to use a single atom per transistor.

Moore’s Law



The process shrink has allowed for moore’s law to continue for years.

Reaching the limits of single core performance



But if you look at the clock speed of individual processing cores, it’s more or less at the limit of where we can be.

Possible solutions:

Divide processor into multiple cores

Use multiple processors

So to counteract those problems while still improving performance year after year has been to increase the core count.

Possible solutions:

Divide processor into multiple cores

Use multiple processors

→ Both solutions are parallel computing!

How have chip manufacturers solved this problem?

But wait, there is more..

Unfortunately, there’s another problem.

Level	Time
CPU Cycle (4.3 GHz)	0.23 ns
L1 Cache Access	0.93 ns
L2 Cache Access	3.26 ns
L3 Cache Access	18.4 ns
RAM Access	68.4 ns
Intel Optane Access	14.9 µs
NVMe SSD Access	338 µs
SATA SSD Access	2.0 ms
Mechanical HDD Access	7.6 ms
Ping to uio.no	14.4 ms
Ping to ucla.edu	189.8 ms
Windows installs update	40 m

Memory also holds back single core performance

A single CPU cycle on modern hardware take less than a quarter of a billionth of a second.

These numbers are based on the Intel Skylake X platform, assuming a clock rate of 4.3 GHz.

Data Sources:
- https://www.7-cpu.com/cpu/Skylake_X.html
- https://www.storagereview.com/intel_optane_800p_nvme_ssd_review
  (Optane is based on peak random read)

Level	Time	Time (Human)
CPU Cycle (4.3 GHz)	0.23 ns	1 s
L1 Cache Access	0.93 ns	4 s
L2 Cache Access	3.26 ns	14 s
L3 Cache Access	18.4 ns	79 s
RAM Access	68.4 ns	5 m
Intel Optane Access	14.9 µs	18 h
NVMe SSD Access	338 µs	17 days
SATA SSD Access	2.0 ms	3 months
Mechanical HDD Access	7.6 ms	1 year
Ping to uio.no	14.4 ms	2 years
Ping to ucla.edu	189.8 ms	26 years
Windows installs update	40 m	∞

These numbers are based on the Intel Skylake X platform, assuming a clock rate of 4.3 GHz.

Data Sources:
- https://www.7-cpu.com/cpu/Skylake_X.html
- https://www.storagereview.com/intel_optane_800p_nvme_ssd_review
  (Optane is based on peak random read)

Conclusions:

- Automated parallelisation is complicated

- Processor complexity requires parallel hardware

- Parallel hardware requires parallel software

We already saw before that automated parallelisation is complicated

So how fast *can* we go?

Good news: potentially infinitely fast

Bad news: depending on the problem

```
for(int i = 0; i < 9001; i++) {
}
```

A lot of tasks that can be run in parallel looks like this.

If iterations are independent, you can run them in parallel.

Amdahl's Law

$$S_{latency}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

Gustafson’s Law

$$S_{latency}(s) = (1 - p) + N\ p$$

Summary:

Parallel Computing is good

Overhead is bad

Power limits are a hot topic

Demonstration!

Crash Course C++

```c
int main(int argc, char** argv) {
  int wops = 1/0;
  return 0;
}
```

See you next week!

~~See you next week!~~

We’ll be using C++ in this course

.. It’s definitely not an easy language.

We'll be using C++ in this course

.. It's a powerful language.