CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Introduction to GPUs

CSCS Summer School 2023

Sebastian Keller, Prashanth Kanduri, Jonathan Coles & Ben Cumming

Course Overview

Over these two days we will cover a range of topics:

- Learn about the GPU memory model;
- Implement parallel CUDA kernels for simple linear algebra;
- Learn how to scale our parallel kernels to utilize all resources on the GPU;
- Learn about thread cooperation and synchronization;
- Learn how to profile GPU applications;
- Port the miniapp to the GPU.

Course Overview

We focus on HPC and GPU architectures, specifically:

- HPC development for P100 GPUs on Piz Daint;
- Using CUDA toolkit version 11 and above;
- Some are available on P100 and later GPUs.
  - e.g. double precision atomics.
- Likewise, we won't be covering some features available on the latest "Ampere" or "Hopper" GPUs.

Course Overview

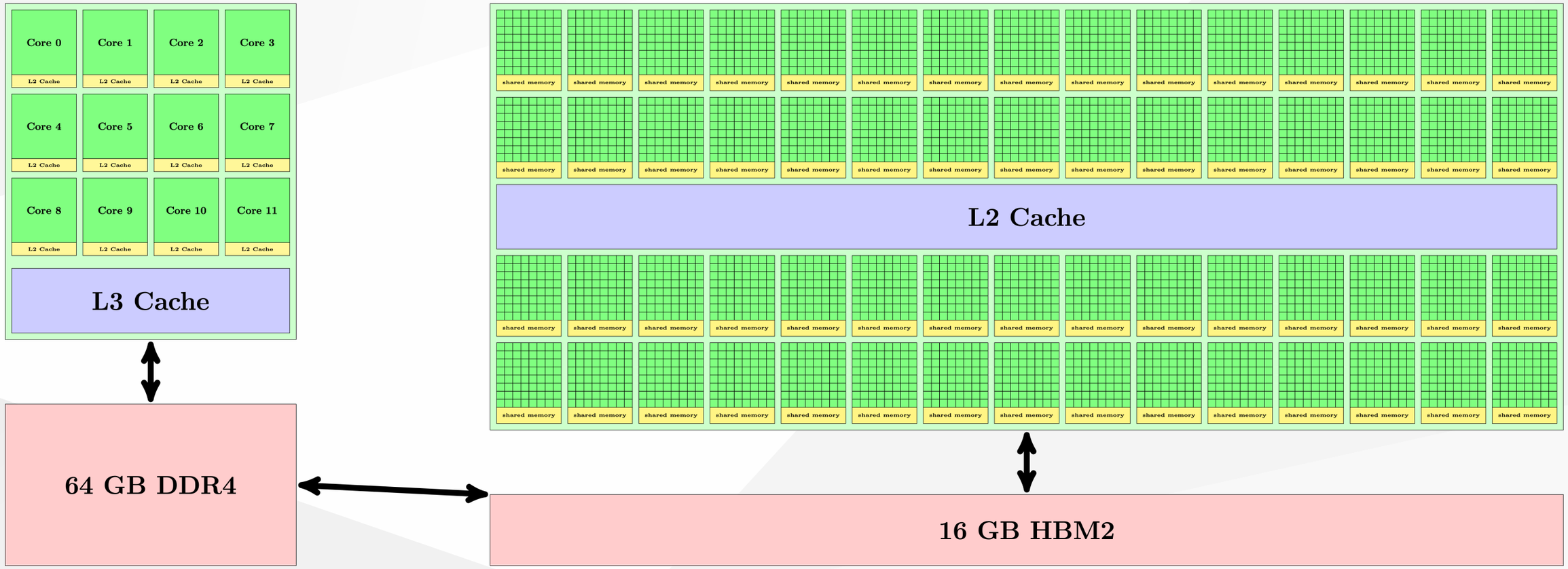
There aren’t many prerequisites for the course:

- No GPU or graphics experience required.
- We assume C++11 knowledge.
- The generic GPU programming concepts from CUDA are useful for when:
  - Developing with OpenACC, OpenCL and GPU-ready libraries.
  - Using ML frameworks that use GPU for compute.

Why GPUs?

There is a trend towards more parallelism “on node”
Multi-Core CPUs get more cores and wider vector lanes:
• 24-core×SMT 4×SIMD 128: IBM Power9 (2017); • 28-core×SMT 2×SIMD 512: Intel Xeon (2020) • 48-core×SMT 1×SIMD 512: Fujitsu ARM A64FX (2020).
Many-Core Accelerators with many highly-specialized cores and high-bandwidth memory:
• NVIDIA P100 GPUs with 3,584 cores (2016); • NVIDIA V100 GPUs with 5,120 cores (2017); • NVIDIA A100 GPUs with 8,192 cores (2020); • NVIDIA H100 GPUs with 18,432 cores (2023)

A Piz Daint Node



... that's a lot of parallelism!

MPI and the Free Lunch

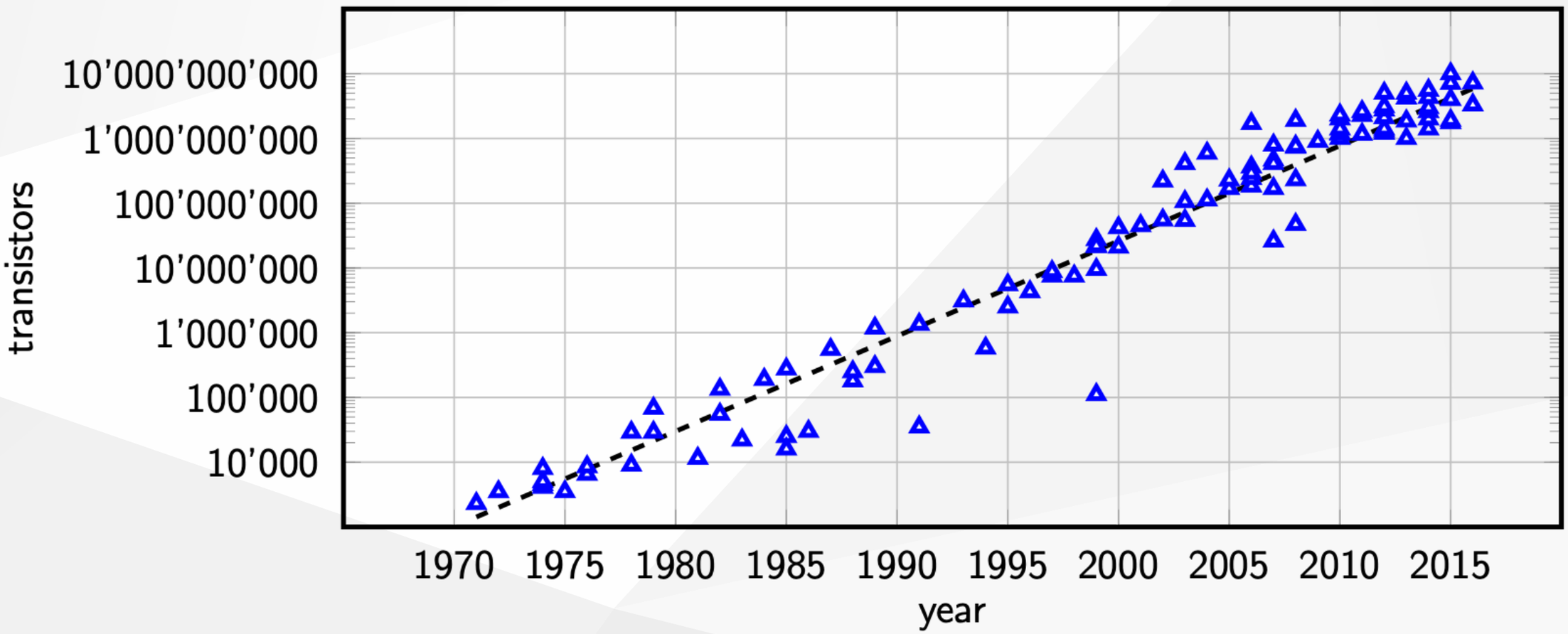
HPC applications were ported to use the message passing library MPI in the late 90s and early 2000s at great cost and effort

- Individual nodes with one or two CPUs
- Break problem into chunks/sub-domains
- Explicit message passing between sub-domains

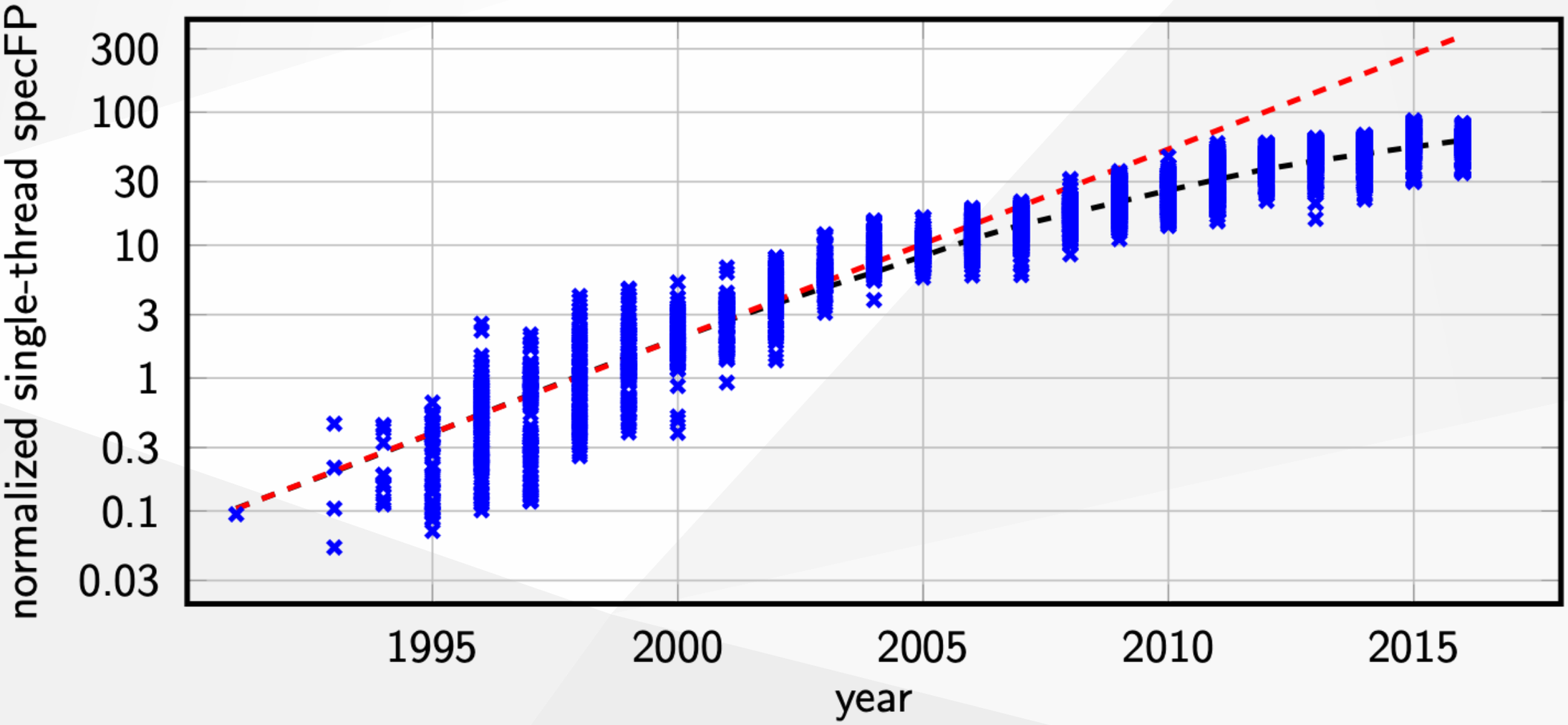
The “free lunch” was the regular speedup in codes as CPU clock frequencies increased and as the number of nodes in systems increased

- With little/no effort, each new generation of processor bought significant speedups.

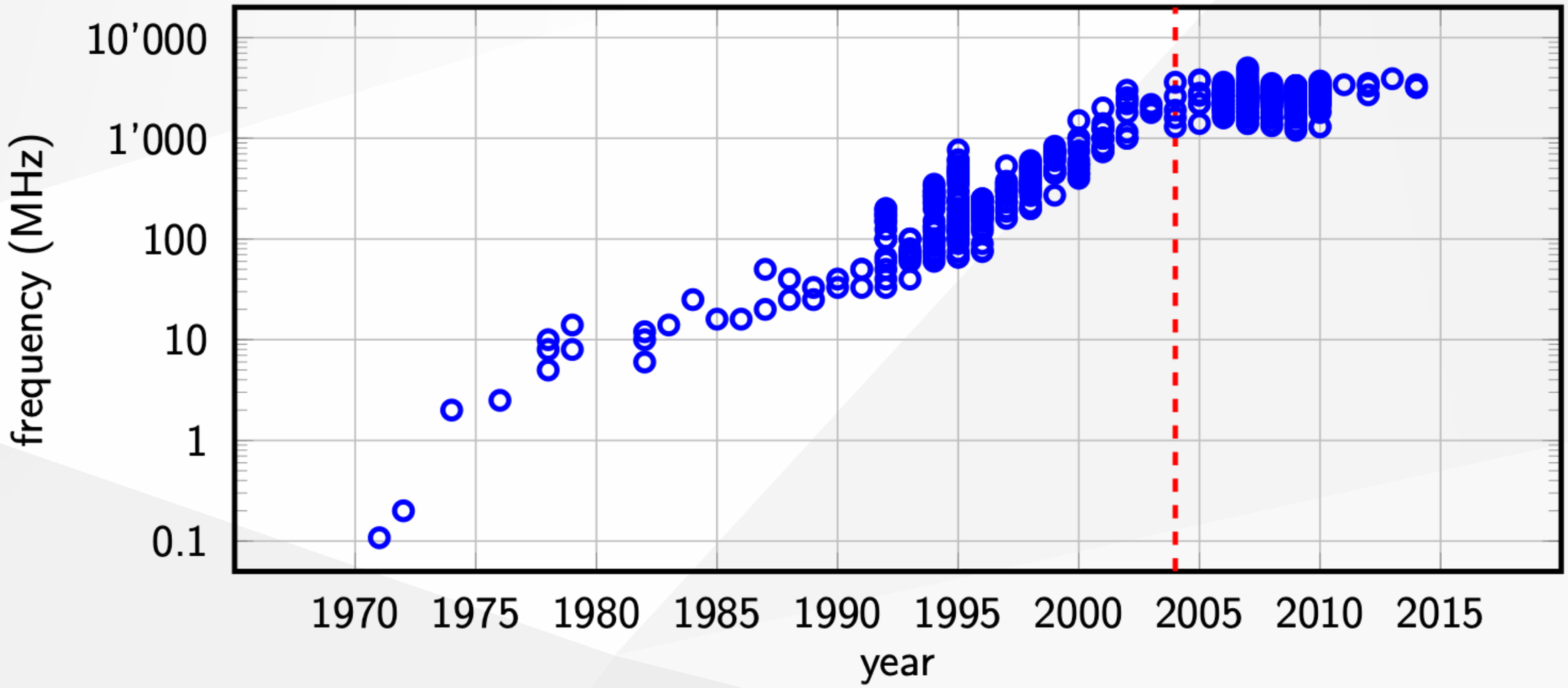
... but there is no such thing as a free lunch.



The number of transistors in processors has increased exponentially for 45 years.



Floating point performance per core is not keeping up...



Clock speeds peaked around 2005.

The Problem? $Power \propto Frequency^3$

How to Speed Up an Application?

There are 3 ways to increase performance:

1. Increase clock speed.
2. Increase the number of operations per clock cycle:
    - vectorization;
    - instruction level parallelelism;
    - more cores.
3. Don’t stall:
    - e.g. cache to avoid waiting on memory requests;
    - e.g. branch prediction to avoid pipeline stalls.

Clock Frequency WILL NOT Increase

In fact, clock frequencies have been going down as the number of cores increases:

- A 4-core Haswell processor at 3.5 GHz (4*3.5=14 Gops/second) has the same power consumption as a 12-core Haswell at 2.6 GHz (12*2.6=31 Gops/second);
- A P100 GPU with 3584 CUDA cores runs at 1.1 GHz.

Caveat

It is not reasonable to compare a CUDA core and an X86 core.

Parallelism WILL Increase

- The number of cores in both CPUs and accelerators will continue to increase
- The width of vector lanes in CPUs will increase
  - 8×SIMD double for AVX512 and SVE (Intel and ARM).
- The number of threads per core will increase
  - Intel SkyLake: 2 threads/core
  - Intel KNL: 4 threads/core
  - IBM Power-8: 8 threads/core

Memory is Slow

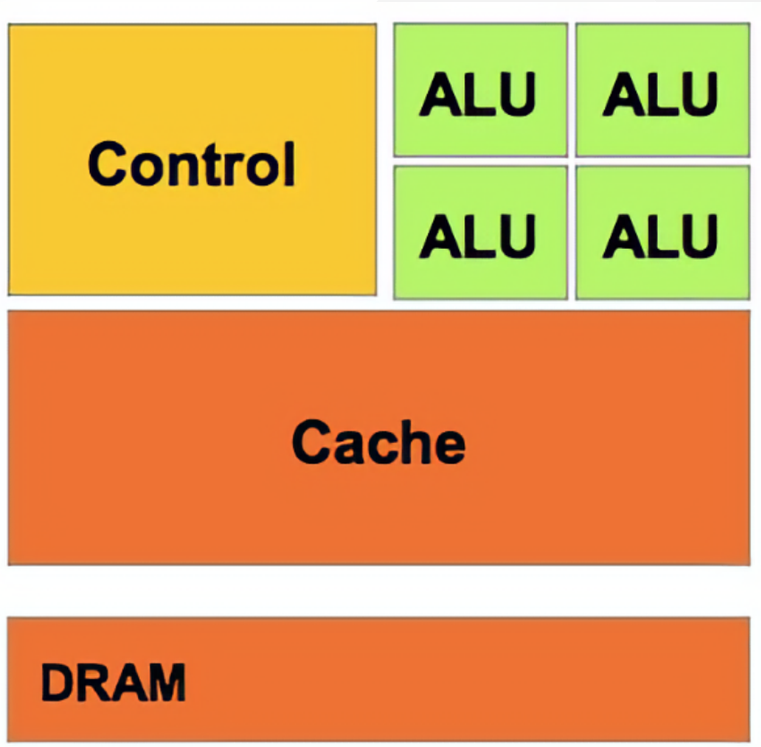
Memory is much slower than processors

- For both CPU and GPU the latency of fetching a cache-line from memory is 100s of cycles...
- ... 100s of cycles that the processor is stalled
- Latency has to be hidden or reduced to minimise stalling

Low-Latency or High-Throughput?

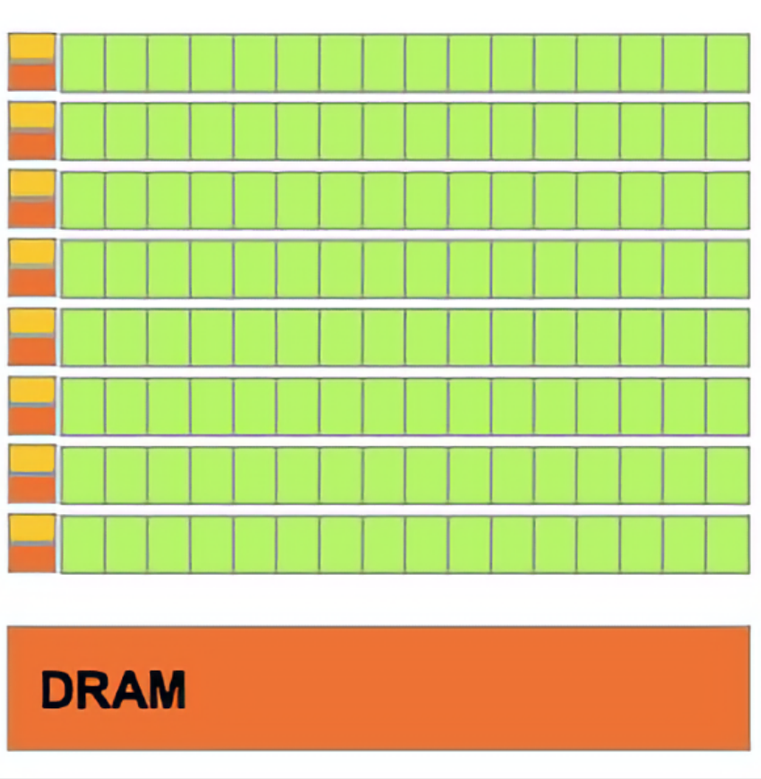
CPU

- Optimized for low-latency access to cached data sets.
- Control logic for out-of-order and speculative execution.



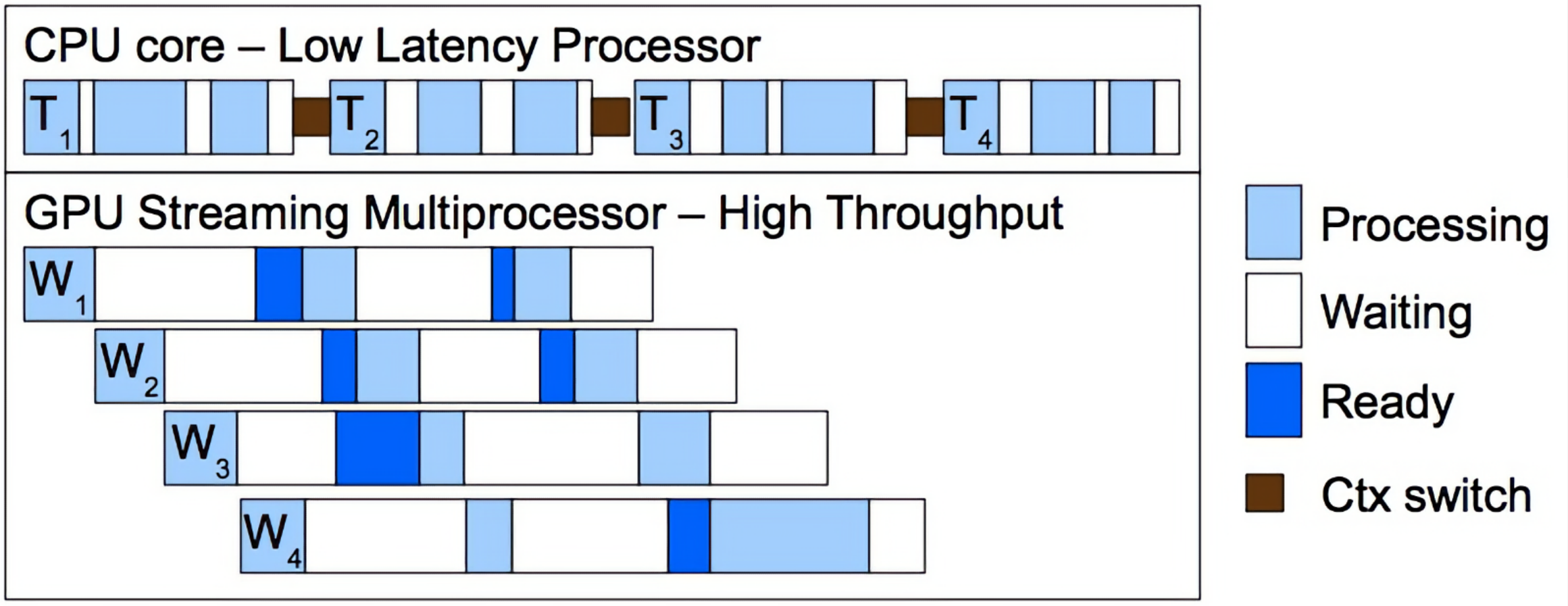
GPU

- Optimized for data-parallel, throughput computation.
- Architecture tolerant of memory latency.
- More transistors dedicated to computation.



GPUs are Throughput Devices

- CPU cores are optimized to minimize latency between operations
- GPUs aim to minimize latency between operations by scheduling multiple warps (thread bundles)



Many Applications aren't Designed for Many-Core

- Exposing sufficient fine-grained parallelism for multi and many core processors is hard.
- New programming models are required.
- New algorithms are required.
- Existing code has to be rewritten or refactored

On-node parallelism will continue to increase:

- Piz Daint @ CSCS (2015): 1 GPU + 1 CPU
- Marconi100 @ CINECA (2020): 4 GPU + 2 CPU
- EUROHPC pre-exascale (2021): 4 GPU + 1 CPU
- US ECP exascale (2021-2023): 4 GPU + 1 CPU

TLDR: Change because power

Writing good concurrent code for many-core is difficult

- But the days of easy speed up each generation of CPU are over
  - Performance gains must not increase power consumption
- This course will be about one type of many-core architecture NVIDIA GPUs
  - CUDA is GPU-specific.
  - Conceptually very close to HIP, OpenCL and SYCL (AMD/NVIDIA/Intel)