



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich



# Introduction to GPUs in HPC

## CSCS Summer School 2017

Ben Cumming, CSCS  
July 13, 2018



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

# Introduction

---

# 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 about concurrent task-based parallelism;
- Learn how to use MPI in CUDA applications;
- Learn how to profile GPU applications.
- Port the miniapp to the GPU.

# Course Overview

We focus on HPC and modern GPU architectures, specifically:

- HPC development for P100 GPUs on Piz Daint;
- Using CUDA toolkit version 8 and above;
- Some features only available on Pascal generation GPUs
  - e.g. double precision atomics.
- Likewise, we won't be covering some features that are available on the latest “Volta” GPUs.

# Course Overview

There aren't many prerequisites for the course:

- No GPU or graphics experience required.
- I assume C++ knowledge...
  - I will be using C++11 (the bits that make C++ easier!)
- The generic GPU programming concepts from CUDA are useful for people interested in OpenACC, OpenCL and GPU-ready libraries.



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

# Why GPUs?

---

There is a trend towards more parallelism “on node”

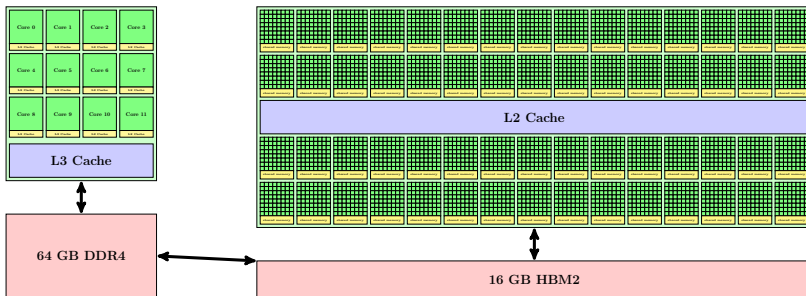
**Multi-core CPUs** get more cores and wider vector lanes:

- 18-core $\times$ 2 thread Broadwell processors from Intel;
- 22-core $\times$ 4 thread Power9 processors from IBM.

**Many-core Accelerators** with many highly-specialized cores and high-bandwidth memory:

- NVIDIA P100 GPUs with 3582 cores;
- NVIDIA V100 GPUs with 5120 cores;
- Intel KNL with 64 cores  $\times$  4 threads.

# A Piz Daint node



...that is a lot of parallelism!



# MPI and the free lunch

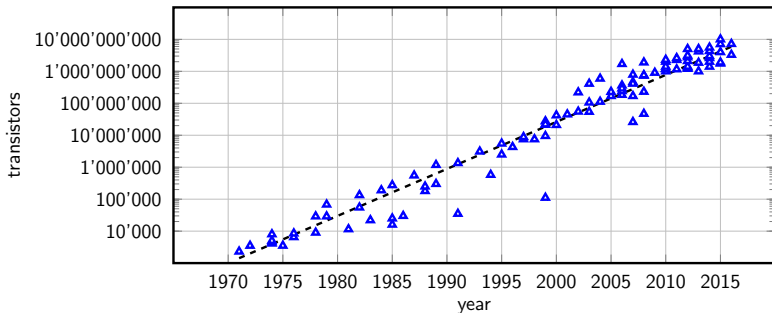
HPC applications were ported to use the message passing library MPI in the late 90s and early 2000s at great cost and effort

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

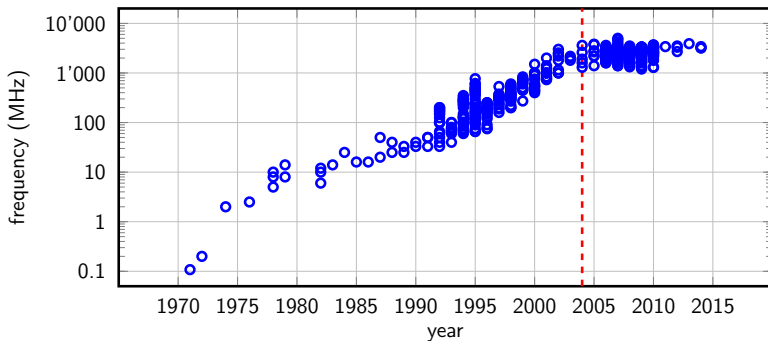
The “free lunch” was the regular speedup in codes as CPU clock frequencies increased and as the number of nodes in systems increased

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

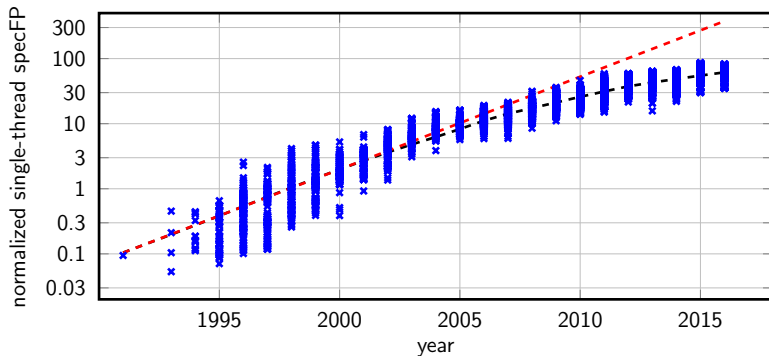
...but there is no such thing as a free lunch



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



The problem:  $\text{power} \propto \text{frequency}^3$



Floating point performance per core is not keeping up

# 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 parallelism;
  - 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 won't increase

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

- A 4-core Haswell processor at 3.5 GHz ( $4 \times 3.5 = 14$  Gops/second) has the same power consumption as a 12-core Haswell at 2.6 GHz ( $12 \times 2.6 = 31$  Gops/second);
- A P100 GPU with 3582 CUDA cores runs at 1.1 GHz.

## Caveat

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

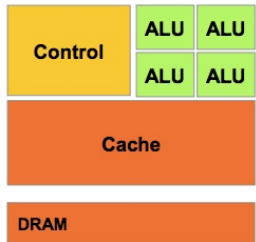
# Parallelism will increase

- The number of cores in both CPUs and accelerators will continue to increase
- The width of vector lanes in CPUs will increase
  - Currently 4 doubles for AVX
  - Increase to 8 double for AVX512 (KNL and Skylake)
- The number of threads per core will increase
  - Intel Haswell: 2 threads/core
  - Intel KNL: 4 threads/core
  - IBM Power-8: 8 threads/core

# 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.

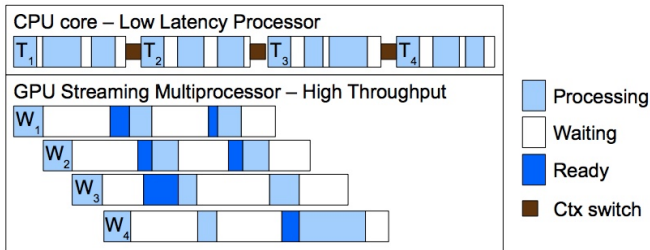


©NVIDIA Corporation 2010



# 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).



©NVIDIA Corporation 2010

# 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.

...and compute nodes are under-utilized

- Users are not getting the most out of allocations.
- The amount of parallelism on-node is only going to increase!

# 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
  - However many concepts are universally applicable to other vector and many-core architectures (e.g. Xeon Phi)