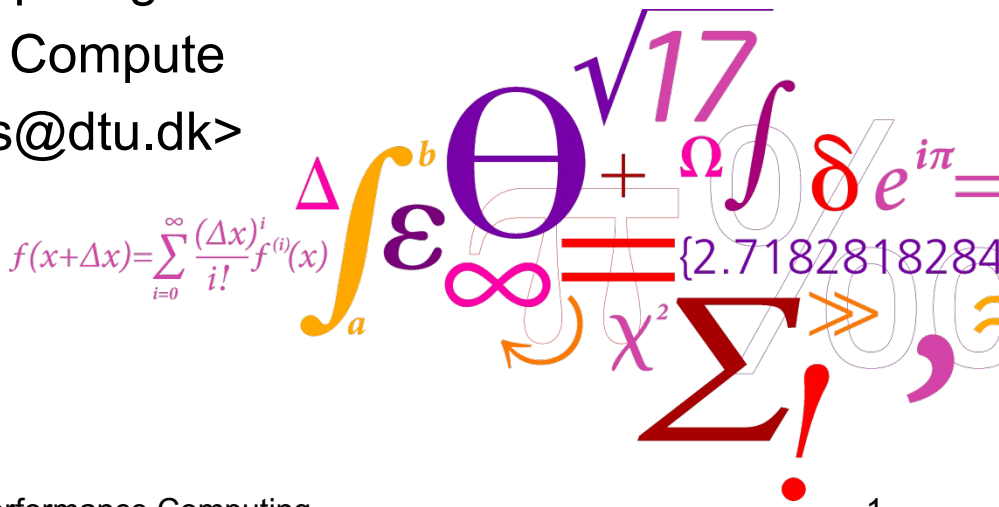
Introduction to GPU Computing


Hans Henrik Brandenborg Sørensen
DTU Computing Center
DTU Compute
<hhbs@dtu.dk>

DTU DCC Computing Center

GPUlab DTU Informatics



Practicalities

- Teacher(s) - Week 3
  - Hans Henrik B. Sørensen
  - Sebastian, Martin, Jonas
- Lectures / GPU Exercises – Mon-Tue
  - Learning by doing!
  - Complementary to Assignment 3 (should not be handed in)
- Assignment #3: GPU Matrix Multiplication and GPU Poisson Problem – Wed-Fri
  - Collaborate on developing the code + report
  - Fill out “responsibilities” in the addendum like week 1+2

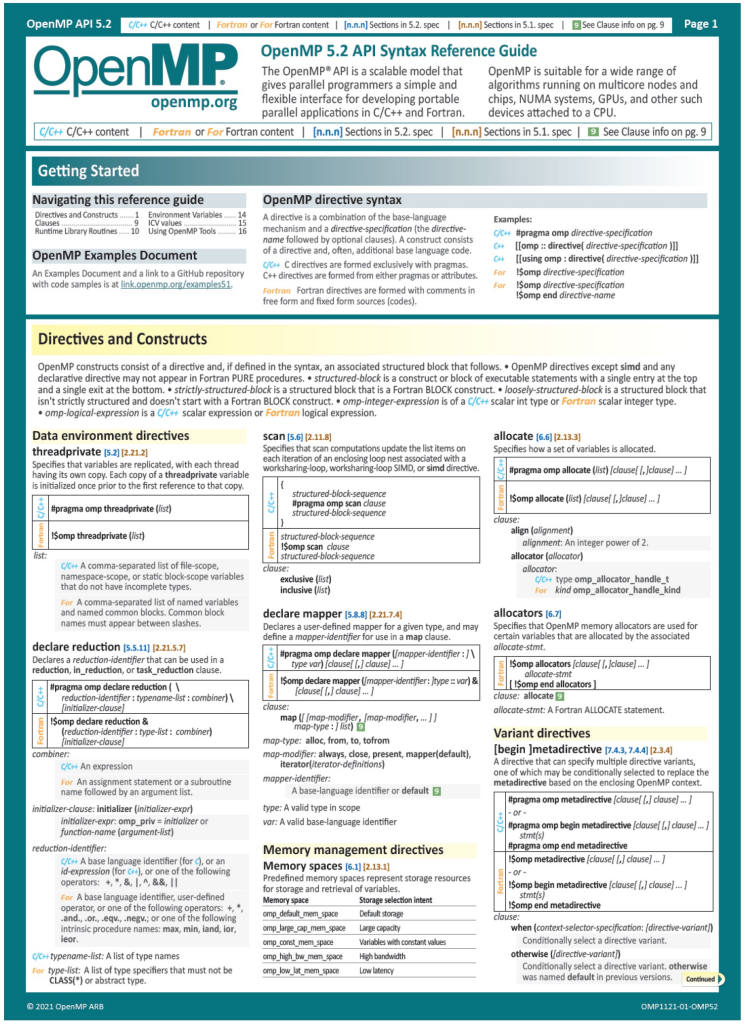
Learning objectives for week 3

- This week we focus on learning
  - Explain high-performance GPUs
  - Interplay of computer components like CPU, GPU, caches, and memory
  - Write parallel programs with OpenMP / Offloading
  - Write efficient programs for multi-core processors and many-core GPUs / Multi-GPUs
- Pre-requisites:
  - Profiency in C/C++
  - Access to a so-called CUDA enabled GPU (e.g. DCC or home-pc with NVIDIA CC.>=8.0).

Overview for week 3

- GPU computing – Mon
  - Introduction to GPU computing
  - OpenMP offload basics
  - OpenMP offload data mapping
- Performance tuning of GPU applications – Tue
  - Introduction to GPU tuning techniques
  - OpenMP offload memory accessing
  - OpenMP offload profiling tools / advanced topics
- Assignment 3 – Wed

OpenMP books for week 3



OpenMP API 5.2 | C/C++ C/C++ content | Fortran or For Fortran content | [n.n.n] Sections in 5.2. spec | [n.n.n] Sections in 5.1. spec | See Clause info on pg. 9 | Page 1

OpenMP
openmp.org

OpenMP 5.2 API Syntax Reference Guide

The OpenMP® API is a scalable model that gives parallel programmers a simple and flexible interface for developing portable parallel applications in C/C++ and Fortran.

OpenMP is suitable for a wide range of algorithms running on multicore nodes and chips, NUMA systems, GPUs, and other such devices attached to a CPU.

C/C++ C/C++ content | Fortran or For Fortran content | [n.n.n] Sections in 5.2. spec | [n.n.n] Sections in 5.1. spec | See Clause info on pg. 9

Getting Started

Navigating this reference guide

OpenMP Examples Document

An Examples Document and a link to a GitHub repository with code samples is at link.openmp.org/examples51.

OpenMP directive syntax

A directive is a combination of the base-language mechanism and a *directive-specification* (the *directive-name* followed by optional clauses). A construct consists of a directive and, often, additional base language code.

C/C++ C directives are formed exclusively with pragmas. C++ directives are formed from either pragmas or attributes.

Fortran Fortran directives are formed with comments in free form and fixed form sources (codes).

Examples:

C/C++ `#pragma omp` *directive-specification*
C++ `[[omp :: directive(` *directive-specification* `)]]`
C++ `[[using omp : directive(` *directive-specification* `)]]`
For `!$omp` *directive-specification*
For `!$omp` *directive-specification* ... `!$omp end` *directive-name*

Directives and Constructs

OpenMP constructs consist of a directive and, if defined in the syntax, an associated structured block that follows. • OpenMP directives except **simd** and any declarative directive may not appear in Fortran PURE procedures. • *structured-block* is a construct or block of executable statements with a single entry at the top and a single exit at the bottom. • *strictly-structured-block* is a structured block that is a Fortran BLOCK construct. • *loosely-structured-block* is a structured block that isn't strictly structured and doesn't start with a Fortran BLOCK construct. • *omp-integer-expression* is of a C/C++ scalar int type or Fortran scalar integer type. • *omp-logical-expression* is a C/C++ scalar expression or Fortran logical expression.

Data environment directives

threadprivate [5.2] [2.21.2]
Specifies that variables are replicated, with each thread having its own copy. Each copy of a **threadprivate** variable is initialized once prior to the first reference to that copy.

C/C++: `#pragma omp threadprivate (`*list*`)`
Fortran: `!$omp threadprivate (`*list*`)`

list:
C/C++ A comma-separated list of file-scope, namespace-scope, or static block-scope variables that do not have incomplete types.
For A comma-separated list of named variables and named common blocks. Common block names must appear between slashes.

declare reduction [5.5.11] [2.21.5.7]
Declares a *reduction-identifier* that can be used in a **reduction**, **in_reduction**, or **task_reduction** clause.

C/C++:
```
#pragma omp declare reduction ( \
  reduction-identifier : typename-list : combiner) \
  [initializer-clause]
```
Fortran:
```
!$omp declare reduction &
  (reduction-identifier : type-list : combiner)
  [initializer-clause]
```

combiner:
C/C++ An expression
For An assignment statement or a subroutine name followed by an argument list.

initializer-clause: **initializer** (*initializer-expr*)
initializer-expr: **omp_priv** = *initializer* or *function-name* (*argument-list*)

reduction-identifier:
C/C++ A base language identifier (for C), or an *id-expression* (for C++), or one of the following operators: +, *, &, |, ^, &&, ||
For A base language identifier, user-defined operator, or one of the following operators: +, *, .and., .or., .eqv., .neqv.; or one of the following intrinsic procedure names: **max**, **min**, **iand**, **ior**, **ieor**.

C/C++ *typename-list:* A list of type names
For *type-list:* A list of type specifiers that must not be **CLASS(*)** or abstract type.

scan [5.6] [2.11.8]
Specifies that scan computations update the list items on each iteration of an enclosing loop nest associated with a worksharing-loop, worksharing-loop SIMD, or **simd** directive.

C/C++:
```
{
  structured-block-sequence
  #pragma omp scan clause
  structured-block-sequence
}
```
Fortran:
```
structured-block-sequence
!$omp scan clause
structured-block-sequence
```

clause:
exclusive (*list*)
inclusive (*list*)

declare mapper [5.8.8] [2.21.7.4]
Declares a user-defined mapper for a given type, and may define a *mapper-identifier* for use in a **map** clause.

C/C++:
```
#pragma omp declare mapper ([mapper-identifier : ] \
  type var) [clause[ [,] clause] ... ]
```
Fortran:
```
!$omp declare mapper ([mapper-identifier :]type :: var) &
  [clause[ [,] clause] ... ]
```

clause:
map ([*[map-modifier*, *[map-modifier*, ...]] *map-type* :] *list*)

map-type: **alloc**, **from**, **to**, **tofrom**

map-modifier: **always**, **close**, **present**, **mapper**(**default**), **iterator**(*iterator-definitions*)

mapper-identifier:
A base-language identifier or **default**

type: A valid type in scope
var: A valid base-language identifier

Memory management directives

Memory spaces [6.1] [2.13.1]
Predefined memory spaces represent storage resources for storage and retrieval of variables.

Memory space	Storage selection intent
omp_default_mem_space	Default storage
omp_large_cap_mem_space	Large capacity
omp_const_mem_space	Variables with constant values
omp_high_bw_mem_space	High bandwidth
omp_low_lat_mem_space	Low latency

allocate [6.6] [2.13.3]
Specifies how a set of variables is allocated.

C/C++: `#pragma omp allocate (`*list*`) [`*clause*`[ [,]`*clause*`] ... ]`
Fortran: `!$omp allocate (`*list*`) [`*clause*`[ [,]`*clause*`] ... ]`

clause:
align (*alignment*)
alignment: An integer power of 2.
allocator (*allocator*)
allocator:
C/C++ type **omp_allocator_handle_t**
For *kind* **omp_allocator_handle_kind**

allocators [6.7]
Specifies that OpenMP memory allocators are used for certain variables that are allocated by the associated *allocate-stmt*.

Fortran:
```
!$omp allocators [clause[ [,]clause] ... ]
  allocate-stmt
[ !$omp end allocators ]
```

clause: **allocate**
allocate-stmt: A Fortran ALLOCATE statement.

Variant directives

[begin]metadirective [7.4.3, 7.4.4] [2.3.4]
A directive that can specify multiple directive variants, one of which may be conditionally selected to replace the **metadirective** based on the enclosing OpenMP context.

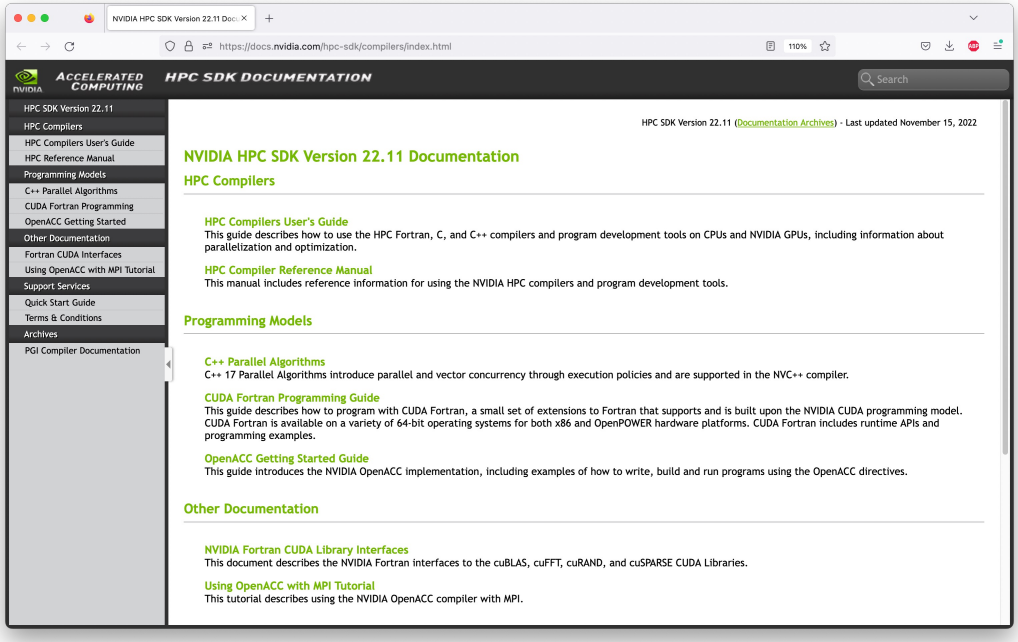
C/C++:
```
#pragma omp metadirective [clause[ [,] clause] ... ]
- or -
#pragma omp begin metadirective [clause[ [,] clause] ... ]
  stmt(s)
#pragma omp end metadirective
```
Fortran:
```
!$omp metadirective [clause[ [,] clause] ... ]
- or -
!$omp begin metadirective [clause[ [,] clause] ... ]
  stmt(s)
!$omp end metadirective
```

clause:
when (*context-selector-specification: [directive-variant]*)
Conditionally select a directive variant.
otherwise (*[directive-variant]*)
Conditionally select a directive variant. **otherwise** was named **default** in previous versions.

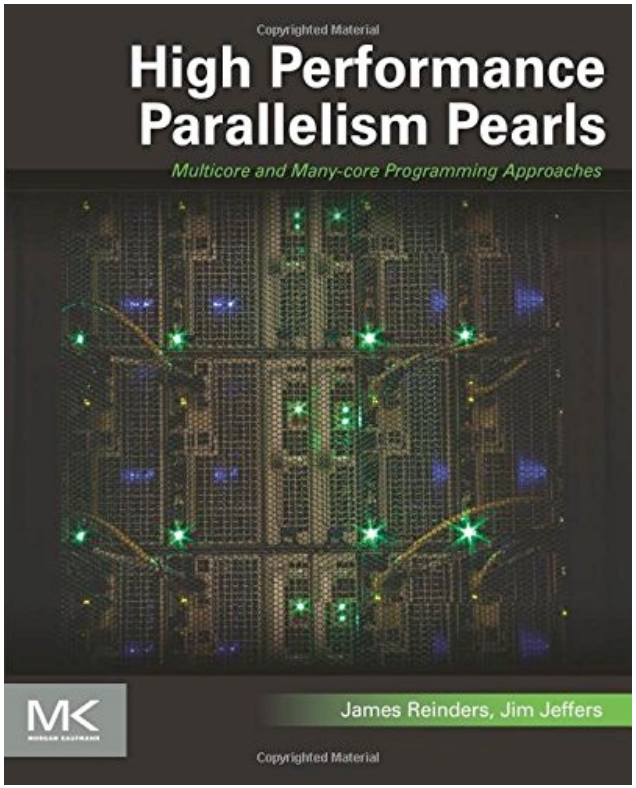
Continued

© 2021 OpenMP ARB | OMP1121-01-OMP52

Our suggestion (if you get hooked)



+



https://docs.nvidia.com/hpc-sdk/compilers/index.html

https://www.openmp.org/spec-html/5.1/openmp.html#openmpch2.html

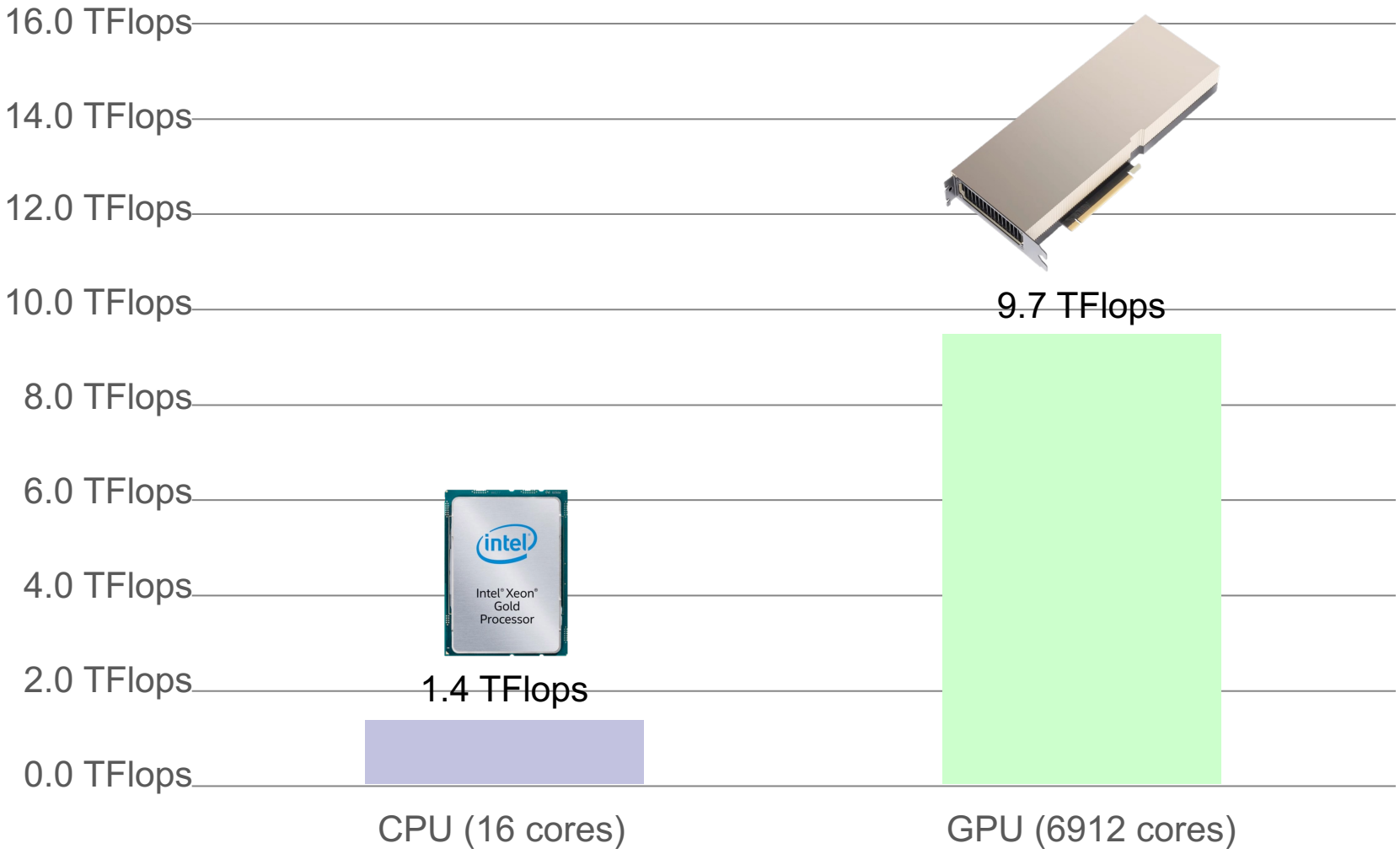
https://stackoverflow.com/questions/tagged/openmp+gpu

Introduction to GPU computing

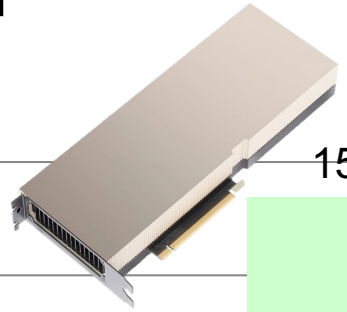
Outline

- Why are GPUs important in HPC?
- Why are GPUs different from CPUs?
- GPUs as accelerators
- GPU hardware for 02614 course
- Motivation slides

Why are GPUs important in HPC?



Why are GPUs important in HPC?



1555 GB/s

141 GB/s

1600 GB/s
1400 GB/s
1200 GB/s
1000 GB/s
800 GB/s
600 GB/s
400 GB/s
200 GB/s
0 GB/s

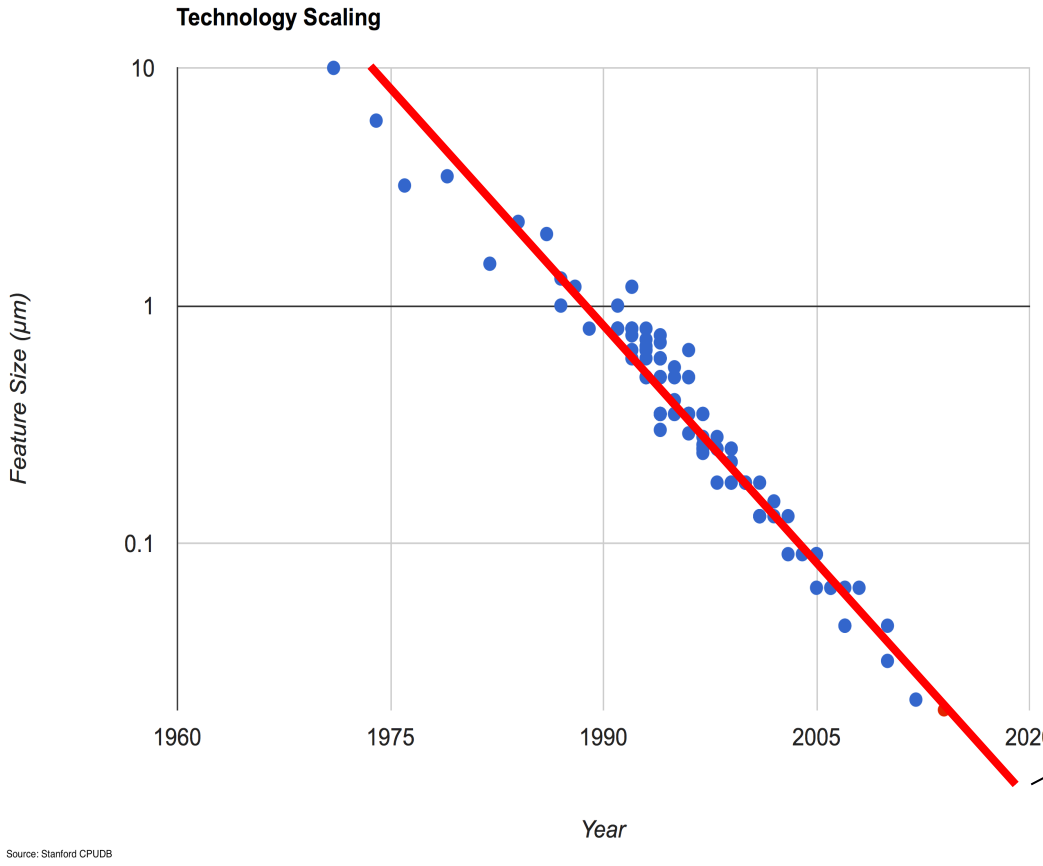
CPU (16 cores)

GPU (6912 cores)

Why are GPUs different from CPUs?

Recap from week 1: Good news

- Transistor size over the years; still decreases

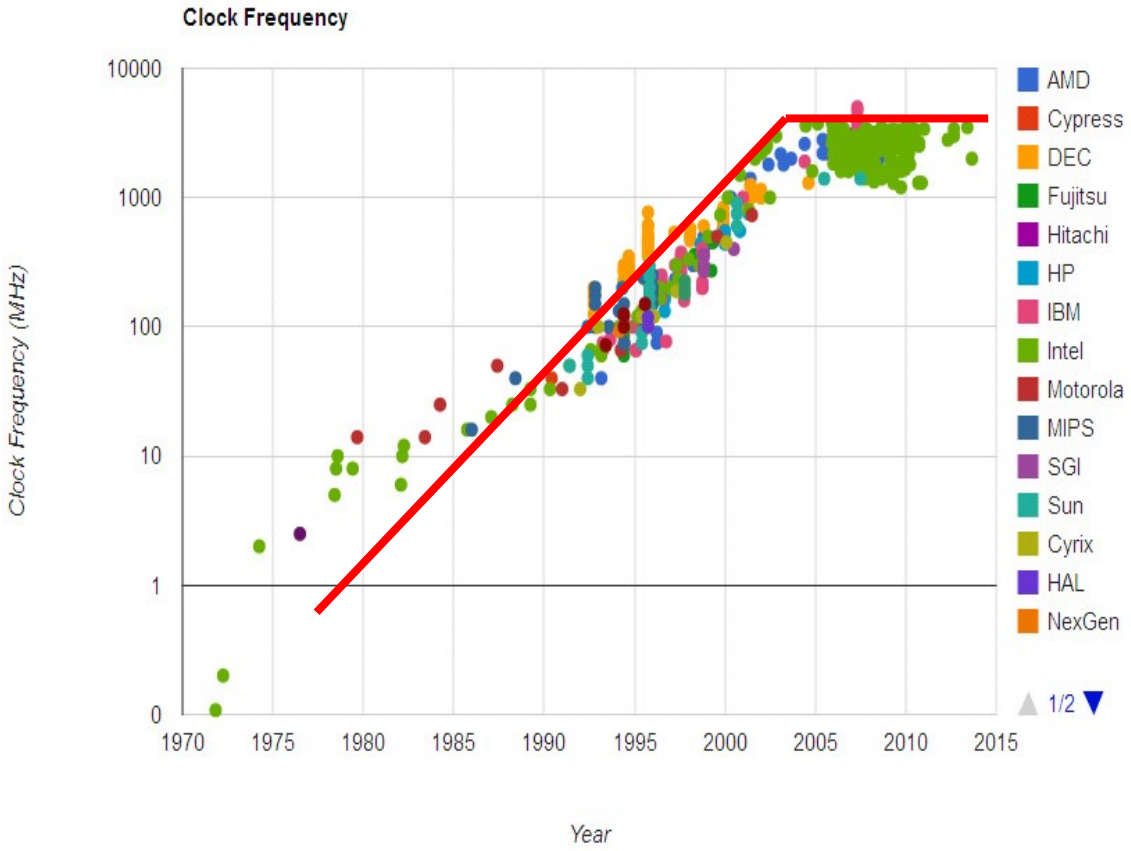


- Smaller
- Faster
- Less power
- More on chip

Currently best: 5 nm

Recap from week 1: Bad news

- Clock speed over the years; stagnated since 2005



- Increasing over many years
- However, over the last decade the clocks speeds have essentially remained constant

Why not increase clock speed?

- Power = Frequency x Voltage^2
- Heat produced depends on power

Why not increase clock speed?

- Power = Frequency x Voltage^2
- Heat produced depends on power
- “Nard scaling”
  - Reduce transistor voltage to counter higher clock speed
  - Broke down in 2005 because of weakened current in the wires (need to distinguish 0 and 1)

Why not increase clock speed?

- Power = Frequency x Voltage^2
- Heat produced depends on power
- “Nard scaling”
  - Reduce transistor voltage to counter higher clock speed
  - Broke down in 2005 because of weakened current in the wires (need to distinguish 0 and 1)
- What matters today: # operations per Watt!
- Trade-off favors slower simpler processors
  - More operations per watt
  - Frequency kept low

Building a modern processor

- What is the goal?

Building a modern processor

- What is the goal? Roughly two choices...

A.	B.
Latency (time to complete a task) [e.g. seconds]	Throughput (tasks completed per unit time) [e.g. jobs/hour]

Building a modern processor

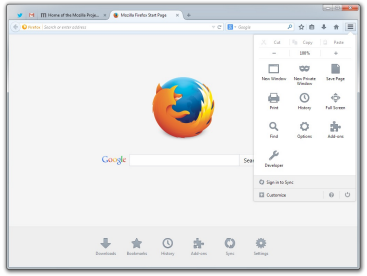
- What is the goal? Roughly two choices...

A.	B.
Latency (time to complete a task) [e.g. seconds]	Throughput (tasks completed per unit time) [e.g. jobs/hour]

- Unfortunately these goals are not always aligned

CPUs target latency (traditionally)

- Usual task of traditional CPUs
  - Desktop applications / OS
    - Lightly threaded
    - Lots of branches
    - Lots of (indirect) memory accesses



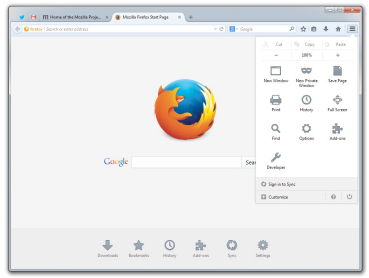
Mac OS

Windows 10

Microsoft Office

- CPUs try to minimize the time to complete a particular task – often to support user interaction!

CPUs target latency (traditionally)



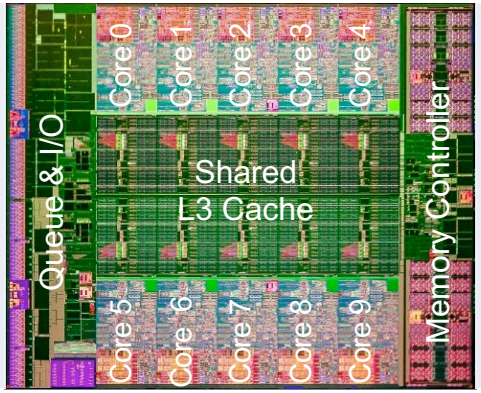
- Usual task of traditional CPUs
  - Desktop applications / OS
    - Lightly threaded
    - Lots of branches
    - Lots of (indirect) memory accesses

Mac OS

Windows 10

Microsoft Office

- CPUs try to minimize the time to complete a particular task – often to support user interaction!
- Complex control hardware
  - \+ Flexibility in performance
  - \+ Lightly parallel
  - − Expensive in terms of power



GPUs target throughput

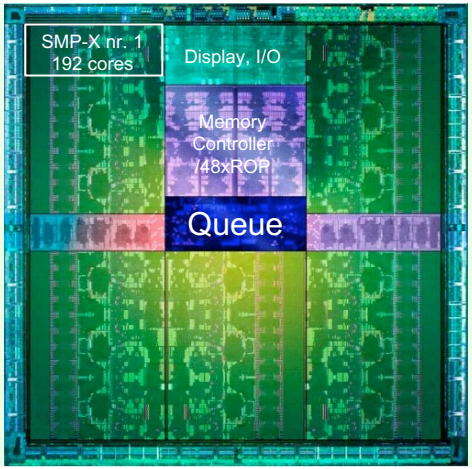
- GPUs are designed to compute pixels – fast!
  - Rendering video games in real-time
  - Play HD movies on smart phones
  - Render visual effects for movies…

- More concerned about the number of pixels per second than the latency of any particular pixel!

GPUs target throughput

- GPUs are designed to compute pixels – fast!
  - Rendering video games in real-time
  - Play HD movies on smart phones
  - Render visual effects for movies…

- More concerned about the number of pixels per second than the latency of any particular pixel!
- Simpler control hardware
  - + More transistors for computation
  - + Power efficient
  - − Highly parallel
  - − More restrictive in performance



Hardware hierarchy

- CPU (typical)
  - Processing Unit (L3 cache)
  - Core (L2 cache / L1 cache / Instr. cache)

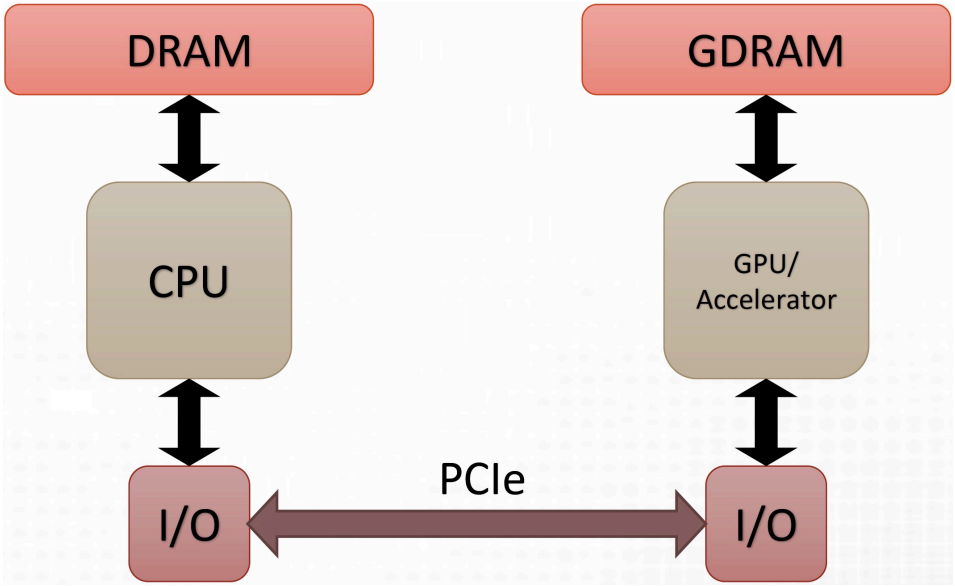
Hardware hierarchy

- CPU (typical)
  - Processing Unit (L3 cache)
  - Core (L2 cache / L1 cache / Instr. cache)
- GPU (Nvidia)
  - Processing Unit (L2 cache)
  - GPC – Graphics Processing Cluster
  - TPC – Texture Processing Cluster
  - SM – Streaming Multiprocessor (64 “cores” / L1 cache)
  - Processing block (Instr. cache)
  - “Core”

GPUs as accelerators

GPUs as accelerators

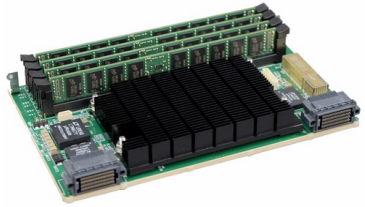
- Problem: Still require OS, I/O, and scheduling
- Solution: “Hybrid system”
  - CPU provides management
  - Accelerators such as GPUs provide compute power



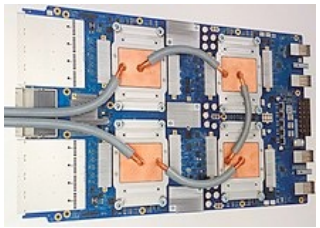
Types of accelerators

- GPUs
  - HPC high-end versions – Tesla branch
  - DP downgraded versions – Titan branch
  - Gamer versions – RTX branch

- Custom many-core processors
  - Japan / China



- TPUs (Tensor Processing Unit)
  - AI accelerator – application-specific integrated circuit (ASIC) developed by Google



Accelerators in Top500

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,463,616	174.70	255.75	5,610
5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096

1st exascale

AMD Instinct (MI250X)

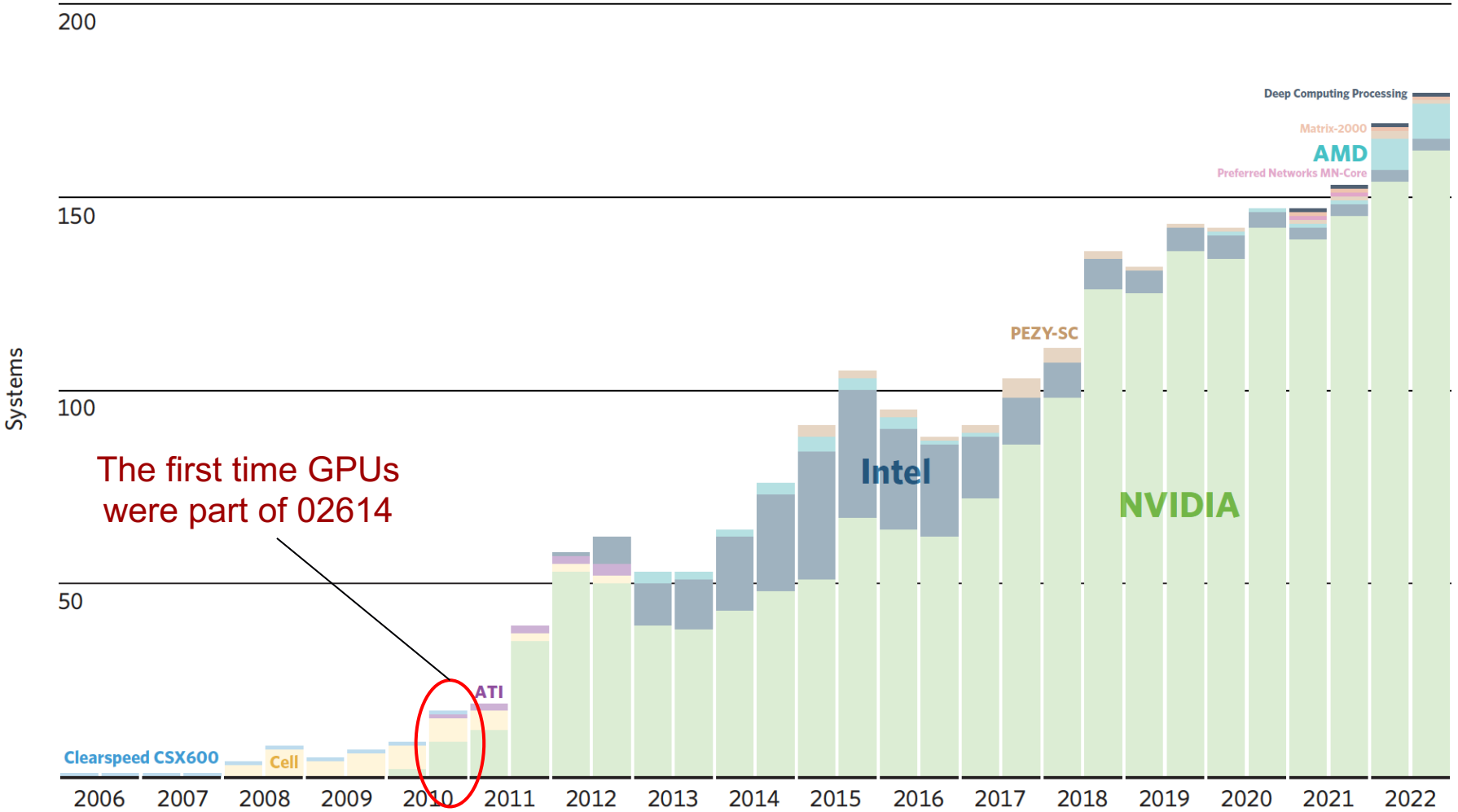
AMD Instinct (MI250X)

Nvidia Tesla (Ampere)

Nvidia Tesla (Volta)

Accelerators in Top500

ACCELERATORS/CO-PROCESSORS



Nvidia GPU architectures

- Five generations of high-performance GPUs:

	# GPUs	Name	Year	Architecture	CUDA cap.	CUDA cores	Clock MHz	Mem GiB	SP peak GFlops	DP peak GFlops	Peak GB/s
Kepler	5	Tesla K40c	2013	GK110B (Kepler)	3.5	2880	745 / 875	11.17	4291 / 5040	1430 / 1680	288
	8	Tesla K80c (dual)	2014	GK210 (Kepler)	3.7	2496	562 / 875	11.17	2796 / 4368	932 / 1456	240
Pascal	8	*TITAN X	2016	GP102 (Pascal)	6.1	3584	1417 / 1531	11.90	10157 / 10974	317.4 / 342.9	480
Volta	22	Tesla V100	2017	GV100 (Volta)	7.0	5120	1380	15.75	14131	7065	898
	12	Tesla V100-SXM2	2018	GV100 (Volta)	7.0	5120	1530	31.72	15667	7833	898
Ampere	6	Tesla A100-PCIE	2020	GA100 (Ampere)	8.0	6912	1410	39.59	19492	9746	1555
Hopper	-	Tesla H100-SXM5	2022	GH100 (Hopper)	9.0	8448	1650	79.18	38984	19492	3110

Source: http://www.hpc.dtu.dk/?page_id=2129

#Cores, Mem, and GB/s keep going up!

Clock freq. level off!

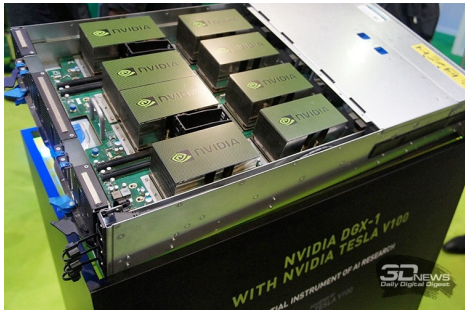
Peak increases 2x every ~3 years!

Trends in HPC due to GPUs

- Improvements at individual computer node level are greatest
  - Less data transfer
  - Heterogeneous computing
  - Avoiding MPI

Trends in HPC due to GPUs

- Improvements at individual computer node level are greatest
  - Less data transfer
  - Heterogeneous computing
  - Avoiding MPI
- "Super-nodes": Nvidia DGX-1
  - Eight tightly linked high-end GPUs
  - 40,960 cores / 960 TFlops in 1 node



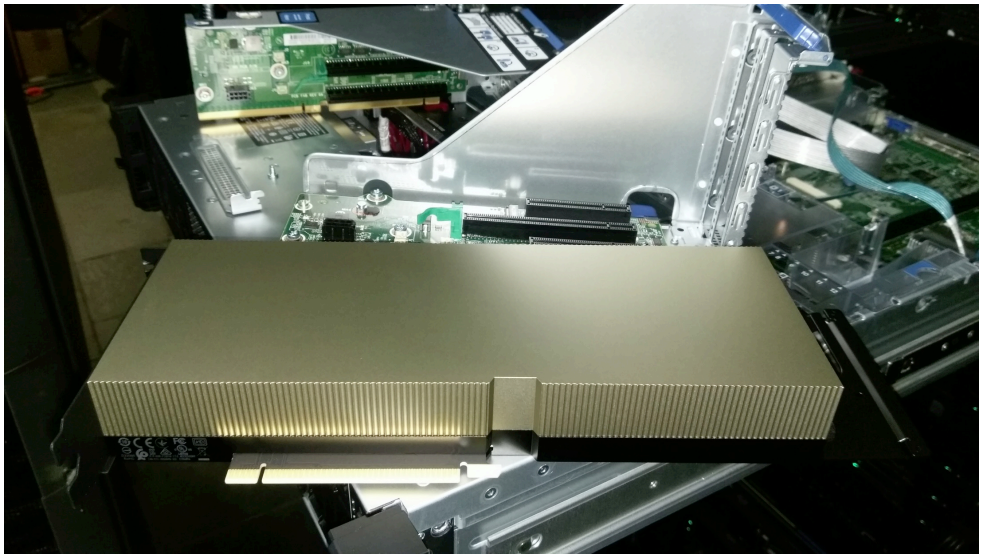
Trends in HPC due to GPUs

- Improvements at individual computer node level are greatest
  - Less data transfer
  - Heterogeneous computing
  - Avoiding MPI
- "Super-nodes": Nvidia DGX-1
  - Eight tightly linked high-end GPUs
  - 40,960 cores / 960 TFlops in 1 node
- Communication costs are increasing
  - Synchronization-reducing algorithms
  - Communication lower-bound algorithms

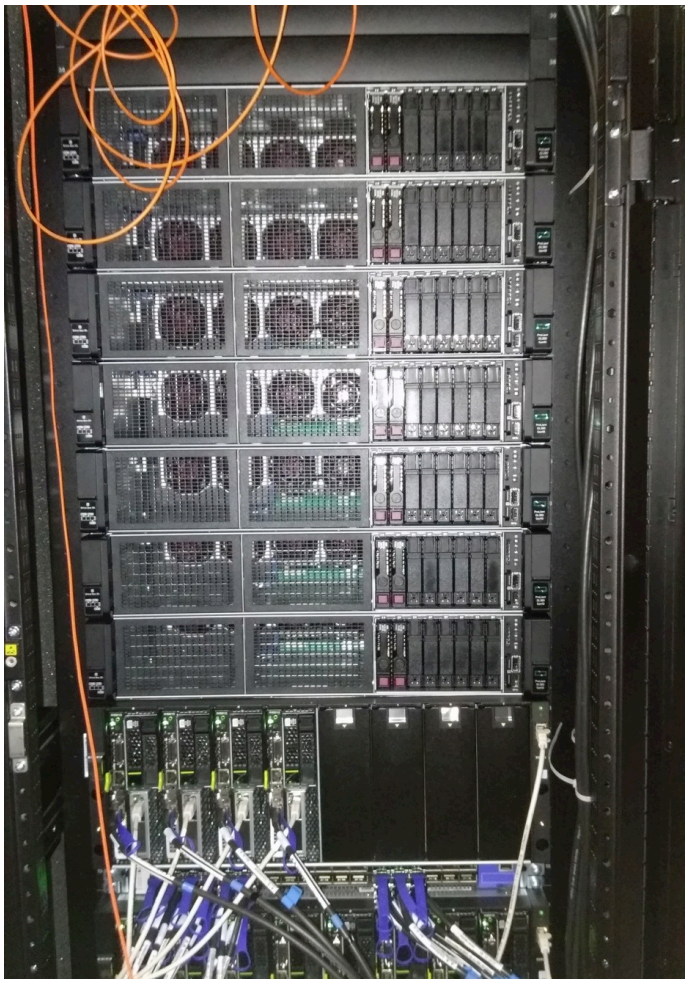
GPU hardware for this course

Ampere cluster for 02614

Delivered: December 23, 2020



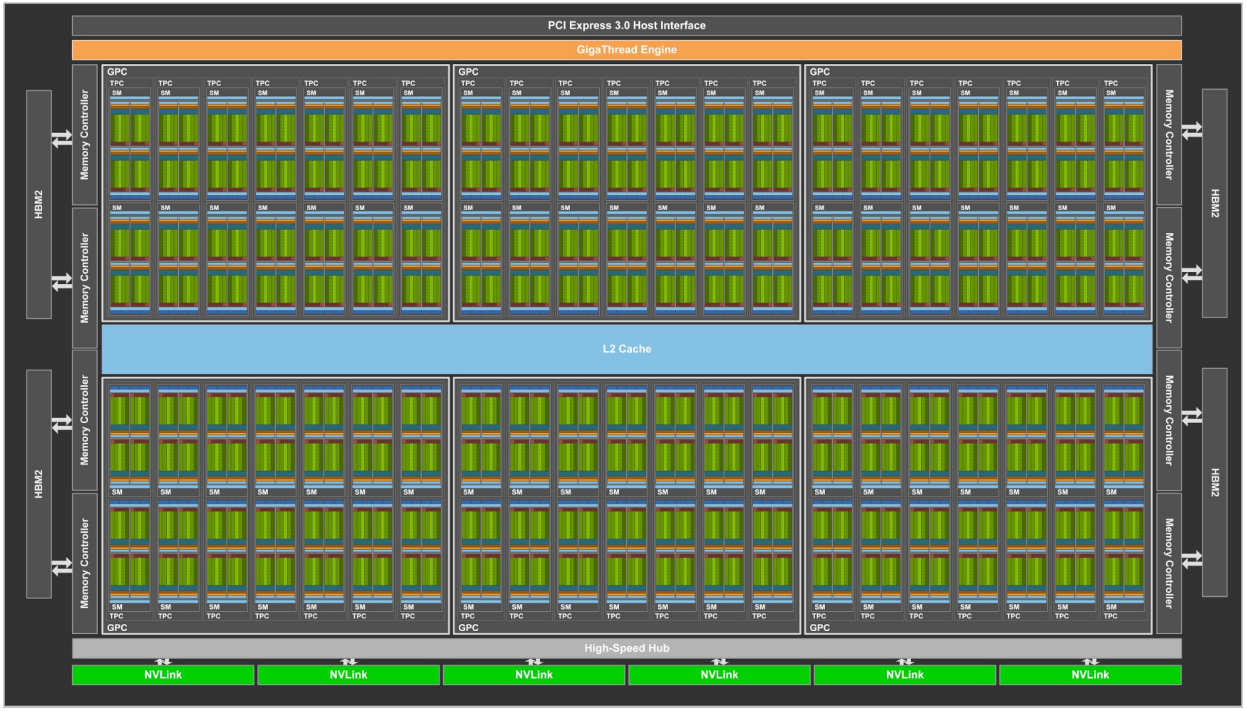
1 HPE node:
2 x Intel Xeon Gold 6126 CPU @ 2.90GHz (16 cores)
2 x Tesla A100-PCIE-40GB (6912 cores)
768 GB DDR4 @ 2666MHz



6 HPE nodes:
Theoretical peak: 133,2 TFlops (FP64)

Ampere cluster for 02614

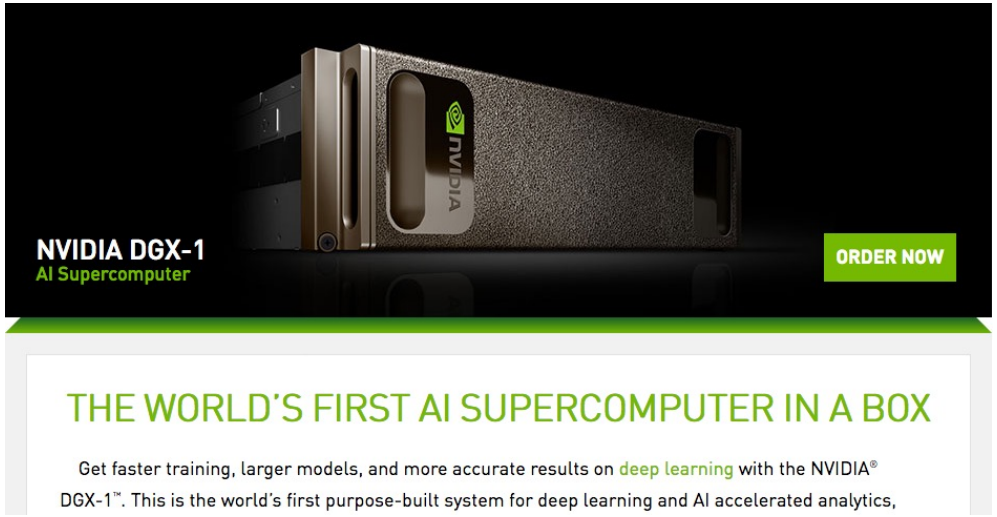
A100-PCIE-40GB



- 108 streaming multiprocessors (SM) – 40 GB memory
- Multi-user system: GPUs are in default mode (shared)
- Batch system: Script sets GPUs to exclusive mode

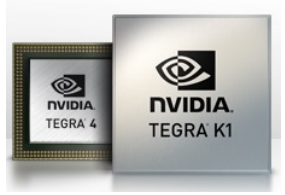
Google trends: GPUs still going up

GPU

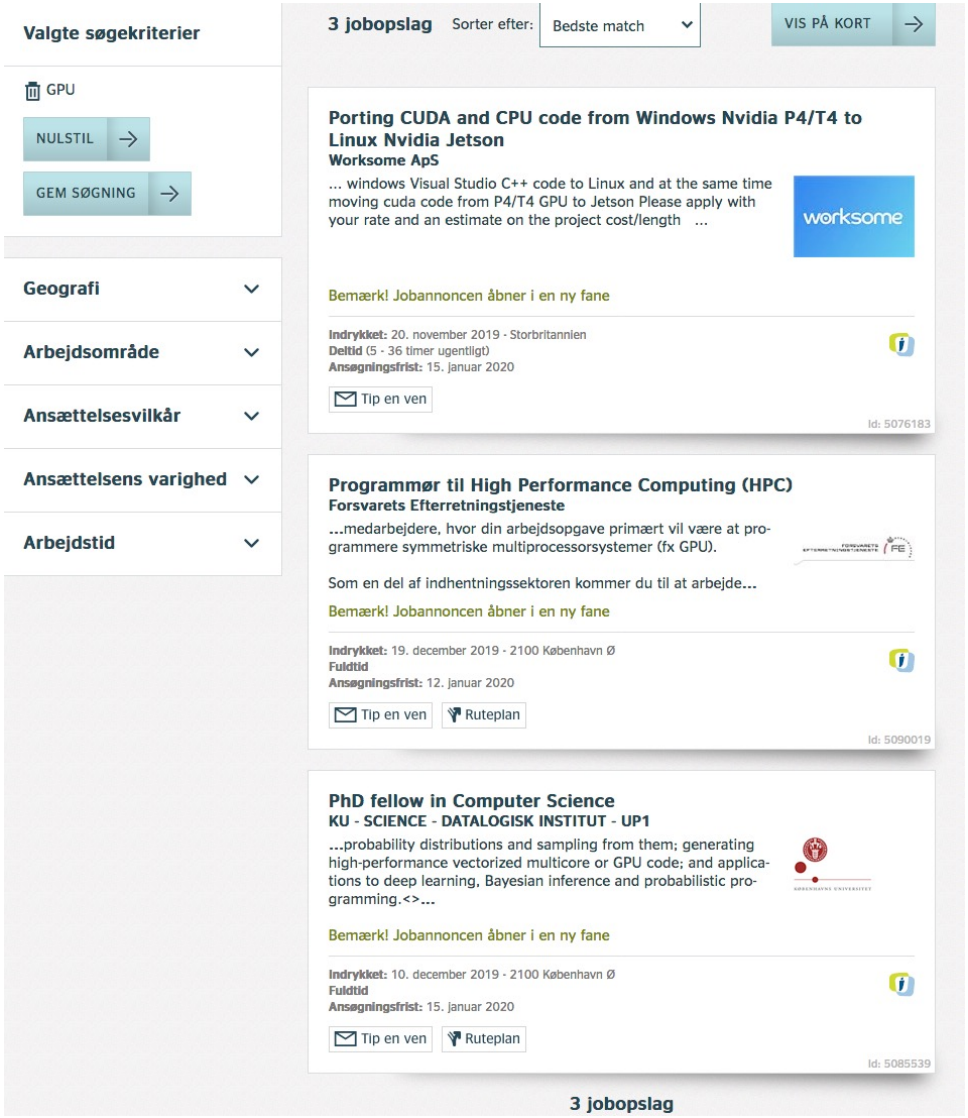


CPU

Machine learning



Motivation Jan 2020



Lukas Christian Høghøj, *Large-scale modelling on GPUs with OpenACC,* 2019

Patrick Møller Jensen and Julian Thomas Reckeweg Olsen, *GPU beamforming*, 2019

Konstantinos Gkanos, *Interactive, real-time room acoustic simulations*, 2019

Gandalf Saxe and Oisin D. Kiær, *Low Energy Transfer Orbits to Mars using Evolution Strategies*, 2019

Mia Sandra Nicole Siemon, *Comparison of GPU programming models*, 2019

Mathias Sorgenfri Lorenz, *Scaling analysis of a multi-GPU Poisson solver*, 2018

Tim Felle Olsen and Mathias Sorgenfri Lorenz, *Large-scale computations on modern GPUs*, 2018

Nick Clausen, *Leveraging the GPU architecture of embedded systems for model predictive control problems*, 2018

…

Motivation Jan 2022 - 2023

PhD scholarship in Cardiac Vector Flow Ultrasound Imaging

Danmarks Tekniske Universitet, Kongens Lyngby

A 3-year PhD scholarship is available at the Center for Fast Ultrasound Imaging (CFU), Department of Health Technology (DTU Health Tech), DTU and at the Department of Cardiology, Rigshosptalet from March 2022. It is conducted in collaboration between the two departments, and the overall purpose of the project is to further develop fast vector flow-based imaging of the heart and make it applicable for clinical use.

This PhD project develops a method for fast vector flow imaging on a Verasonics research scanner and determines its performance and applicability to cardiac diagnostics.

Tjek jobglæden:

391 evalueringer

INDRYKKET: 17-12-2021

"Experience with GPU programming is desirable."

Apply for this job

Apply no later than 21 January 2022
Apply for the job at DTU Health Tech by completing the following form.

Apply online >

International Workshop on OpenMP

↳ IWOMP 2022: **OpenMP in a Modern World: From Multi-device Support to Meta Programming** pp 81–93 | Cite as

Feasibility Studies in Multi-GPU Target Offloading

Anton Rydahl, Mathias Gammelmark & Sven Karlsson

Conference paper | First Online: 20 September 2022

178 Accesses

Part of the Lecture Notes in Computer Science book series (LNCS,volume 13527)

Download book PDF

Download book EPUB

Sections | Figures | References

Abstract

Introduction

Background

Design Patterns

Abstract

Many of the largest supercomputers are based on heterogeneous architectures with multiple general-purpose graphics processing units (GPGPUs) per compute node. While many APIs

https://link.springer.com/chapter/10.1007/978-3-031-15922-0_6#Sec9

End of lecture