COMP 4901Q: High Performance Computing (HPC)

Lecture 1: Introduction

Instructor: Shaohuai SHI (shaohuais@cse.ust.hk)

Teaching assistants: Mingkai TANG (mtangag@connect.ust.hk)

Yazhou XING (yxingag@connect.ust.hk)

Course homepage: https://course.cse.ust.hk/comp4901q/

Intended Learning Outcomes

- Be able to describe the **elements** and **principles** of HPC.
- Be able to master the **system architecture** of high performance computers.
- Be able to learn **parallel programming** languages and frameworks.
- Be able to use one of programming languages or frameworks to **design and develop HPC systems** for real-world science or engineering problems.

Course Schedule

1. Introduction to high performance computing. (1 lecture) (4 Feb, Fri)
    - Course structure, basic concepts of HPC, examples that are using HPC.
2. Introduction to parallel computer architecture. (1 lecture) (9 Feb, Wed)
    - Parallel computer organization, processor (e.g., CPU and GPU), memory, storage, interconnect, etc.
3. Revisit C/C++ programming and common Linux commands. (1 lecture) (11 Feb, Fri)
    - For the preparation of the following classes that will have extensive code examples.
4. Introduction to parallel programming. (1 lecture) (16 Feb, Wed)
    - Overview of parallel programming and related frameworks/libraries/languages.
5. Introduction to OpenMP. (2 lectures) (18 Feb, Fri and 23 Feb, Wed) **(Lab Tutorial 1, 21 Feb, Mon)**
    - Multi-threading programming for CPUs and the framework of OpenMP and its syntax.
6. OpenMP 3.0 and tasking. (2 lectures) (25 Feb, Fri and 2 Mar, Wed)
    - Introduce main features of OpenMP 3.0
7. Introduction to GPU computing. (2 lectures) (4 Mar, Fri and 9 Mar, Wed)
    - Basic concepts of GPU computing, GPU architecture (e.g., stream processors, memory, scheduler, etc.)

Course Schedule

8. CUDA programming on the GPU. (4 lectures) (11, 16, 18, and 23 Mar) **(Lab Tutorial 2, 14 Mar, Mon)**
   - Basic concepts in CUDA programming, how to program CUDA code on the GPU, and some advanced topics in optimizing CUDA code.
9. Introduction to MPI. (2 lectures) (25 Mar, Fri and 30 Mar, Wed) **(Lab Tutorial 3, 28 Mar, Mon)**
   - Basic concepts of MPI and distributed memory programming.
10. Collective communication algorithms in MPI. (4 lectures) (1, 6, 8, and 20 Apr)
    - Some advanced topics in optimized algorithms for collective communication.
11. Combining shared-memory and distributed-memory computing with OpenMP, CUDA, and MPI. (2 lectures) (22 Apr, Fri and 27 Apr, Wed) **(Lab Tutorial 4, 25 Apr, Mon)**
    - Practices and examples to demonstrate how to combine these three technologies to achieve higher performance.
12. A case study: large-scale machine learning in HPC. (2 lectures) (29 Apr, Fri and 4 May, Fri)
    - A detailed example to show how to exploit HPC to support large-scale machine learning
13. Introduction to Hadoop and Spark. (4 May)
    - Introduce some basic concepts of Hadoop and Spark

Lab Sessions

1. Lab Tutorial 1: Programming with OpenMP **(21 Feb, Mon)**
   - Programming environment setup
   - Examples of writing, debugging, compiling, and running code
2. Lab Tutorial 2: Programming with CUDA on GPUs **(14 Mar, Mon)**
   - Programming environment setup
   - Examples of writing, debugging, compiling, and running code
3. Lab Tutorial 3: Programming with MPI on clusters **(28 Mar, Mon)**
   - Programming environment setup
   - Examples of writing, debugging, compiling, and running code
4. Lab Tutorial 4: Examples of combining OpenMP, CUDA, and MPI **(25 Apr, Mon)**
   - Some examples to demonstrate how to combine different parallel programming techniques

Grading (No Final Exam)

- Homework: 2 times, 10% each.
  - Non-programming homework
- Programming assignments: 4 times, 10% each.
  - OpenMP
  - CUDA
  - MPI
  - Combine two or three of the above
- One final project (documentation, code, and presentation): 40%.
  - Proposal
  - Design
  - Coding
  - Presentation

Discussion is encouraged, plagiarism is prohibited!
Academic integrity: http://ugadmin.ust.hk/integrity/student-1.html

Final Project

- Find a challenging problem
  - should be computing-intensive/time-consuming to solve with a single processor
  - can be any science or engineering area you like
- Literature review
  - search related work to understand how they address the problem
  - what are the main issues of the existing methods
- Methodology
  - propose your own method to address the problem
  - compare your method with existing ones
    - theoretically analyze the improvement
    - empirically compare performance on specific hardware (e.g., multi-core CPU, GPU, or clusters)
- Code
  - implement your methods using existing frameworks to run on specific hardware
- Documentation
- Presentation

What is HPC

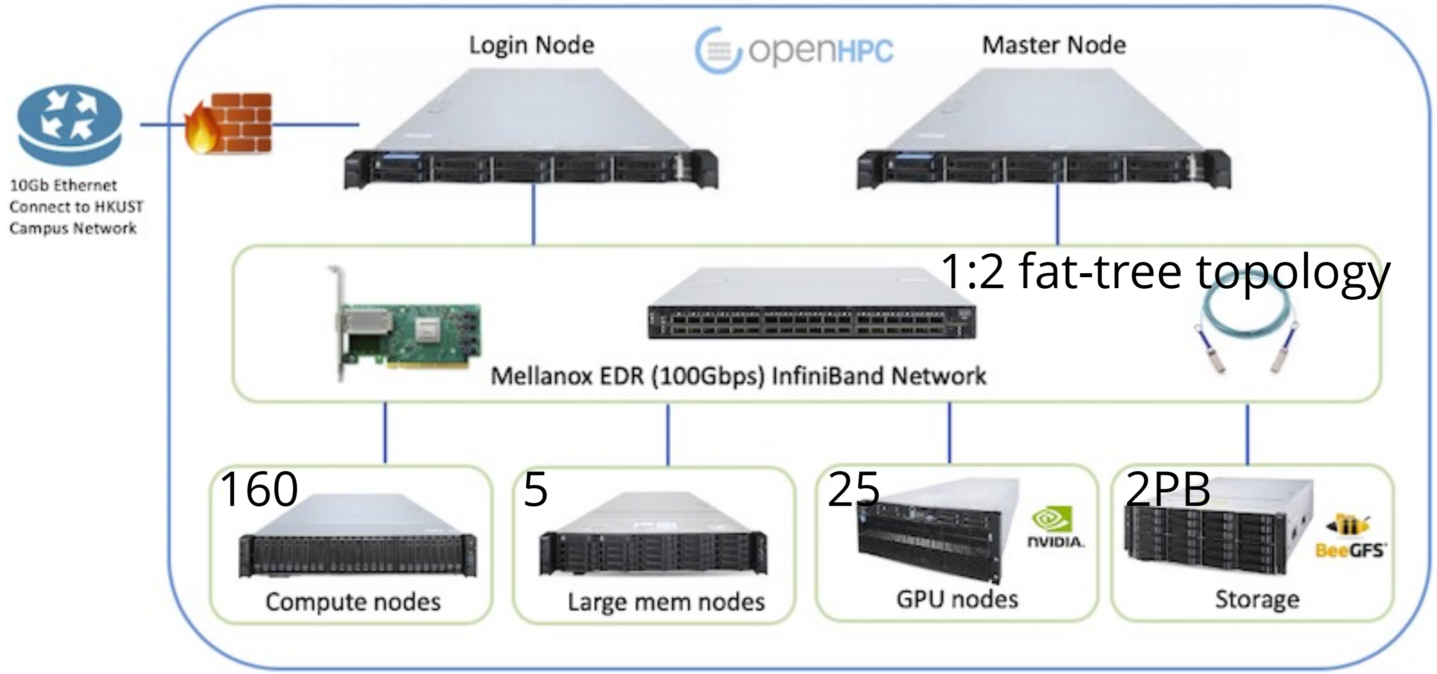
- HPC most generally refers to the practice of aggregating **computing power** in a way that delivers **higher performance** (thus shorter time) to **solve large problems** in science and engineering
  - **Computing power**: many-core processors to supercomputers
  - **Large problems**: can be divided into small problems and solved by parallel computing
  - **Higher performance**: solve the problem as fast as possible

- It is typically talking about two main areas:
  - Hardware
    - From multi-core processors to supercomputers
    - The top500 list: https://www.top500.org/
  - Software stack
    - Frameworks to utilize the computing resources
    - Algorithms to reduce the time/memory complexity

Hardware of Supercomputers

- On a single node
  - Processors: e.g., CPU, GPU, etc.
  - Memory: e.g., DRAM, HBM
  - Storage: HDD, SSD, etc.
  - Interconnects: QPI, UPI, PCIe, NVLink, …
- On a cluster
  - Interconnect between nodes:
    - Ethernet or InfiniBand
  - Network switch
    - Topology of interconnects
    - Fat-tree, BCube, etc.
  - Management nodes
    - Login, Monitor, …

#1 Supercomputer Fugaku: 158,976 nodes, 7,630,848 Cores



HPC cluster at HKUST

Image credit:
https://www.riken.jp/en/news_pubs/news/2020/20201117_2/index.html
https://itsc.ust.hk/services/academic-teaching-support/high-performance-computing/hpc3-cluster/overview

Performance Measure

- FLOP: floating point operation
  - double (default, 8 bytes) or float precision (4 bytes)
  - measure the workload of the problem

- FLOP/s (or FLOPS): floating point operations per second
  - measure the performance of the system at addressing a target problem

- Commonly used units

Kilo:	KFLOP/s = 10^3 FLOPS
Mega:	MFLOP/s = 10^6 FLOPS
Giga:	GFLOP/s = 10^9 FLOPS
Tera:	TFLOP/s = 10^12 FLOPS
Peta:	PFLOP/s = 10^15 FLOPS
Exa:	EFLOP/s=10^18 FLOPS
Zetta:	ZFLOP/s = 10^21 FLOPS

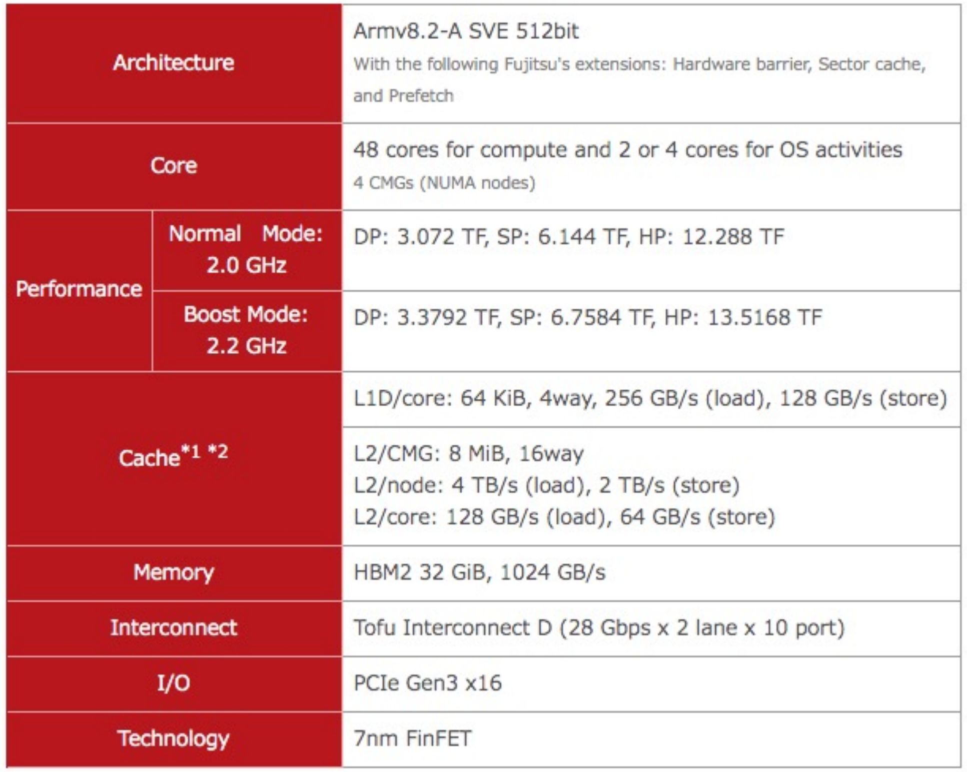
World-wide # 1 (Supercomputer Fugaku): ~537 PFLOPS
World-wide # 2 (Summit): ~200 PFLOPS
The top500 list: https://www.top500.org/

Supercomputer Fugaku (#1)

- Calculate peak performance
  - # of cores: 48
  - SVE 512-bit × 2 vector extensions per core
  - Frequency per core: 2.2 GHz
  - Double precision: 64 bits
  - 512-bit addition or multiply (fma) per cycle
    - 512/64=8 double-precision FLOP per cycle
  - Peak performance = 2.2×10^9 (Frequency) × 8 × 2 × 2 (FLOP per cycle) × 10^{-12} (Tera) × 48 (Cores) = 3.3792 TFLOPS

(Source: https://www.riken.jp/en/news_pubs/news/2020/20201117_2/index.html)

- Supercomputer Fugaku (#1 supercomputer)
  - Peak performance: ~537 PFLOPS
  - 158,976 nodes

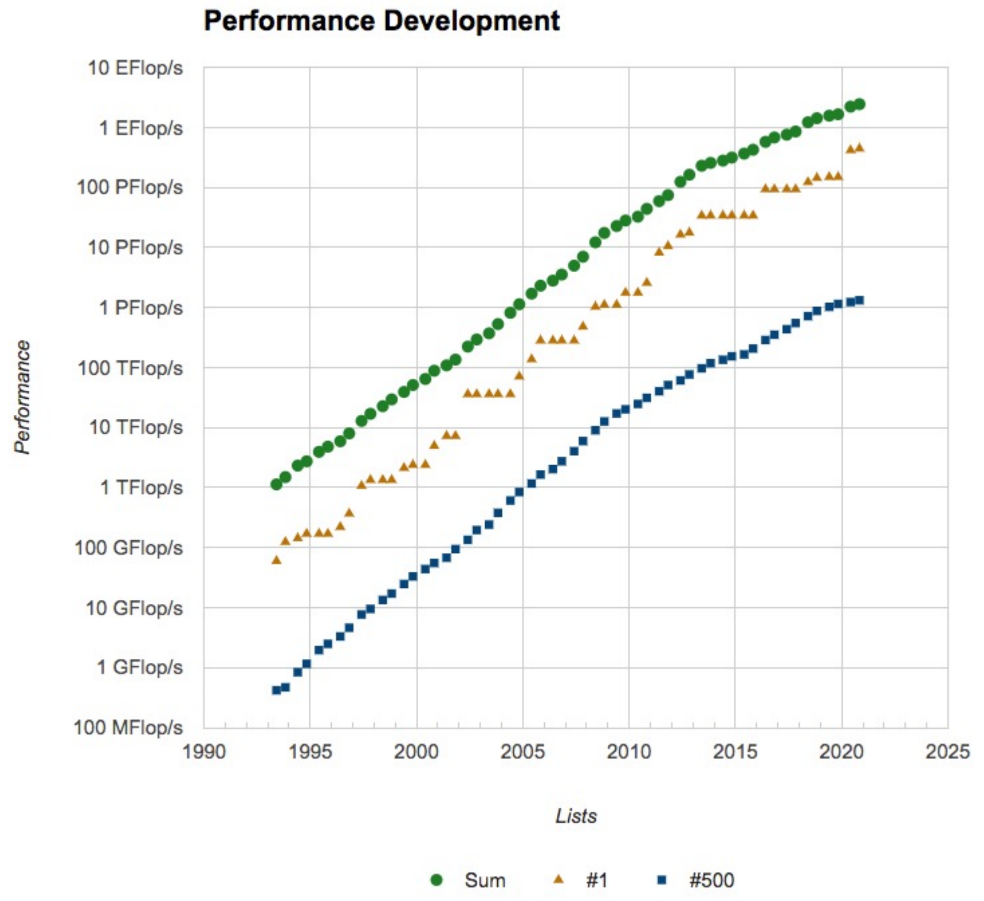


Architecture		Armv8.2-A SVE 512bit With the following Fujitsu's extensions: Hardware barrier, Sector cache, and Prefetch
Core		48 cores for compute and 2 or 4 cores for OS activities 4 CMGs (NUMA nodes)
Performance	Normal Mode: 2.0 GHz	DP: 3.072 TF, SP: 6.144 TF, HP: 12.288 TF
	Boost Mode: 2.2 GHz	DP: 3.3792 TF, SP: 6.7584 TF, HP: 13.5168 TF
Cache[*1 *2]		L1D/core: 64 KiB, 4way, 256 GB/s (load), 128 GB/s (store)
		L2/CMG: 8 MiB, 16way L2/node: 4 TB/s (load), 2 TB/s (store) L2/core: 128 GB/s (load), 64 GB/s (store)
Memory		HBM2 32 GiB, 1024 GB/s
Interconnect		Tofu Interconnect D (28 Gbps x 2 lane x 10 port)
I/O		PCIe Gen3 x16
Technology		7nm FinFET

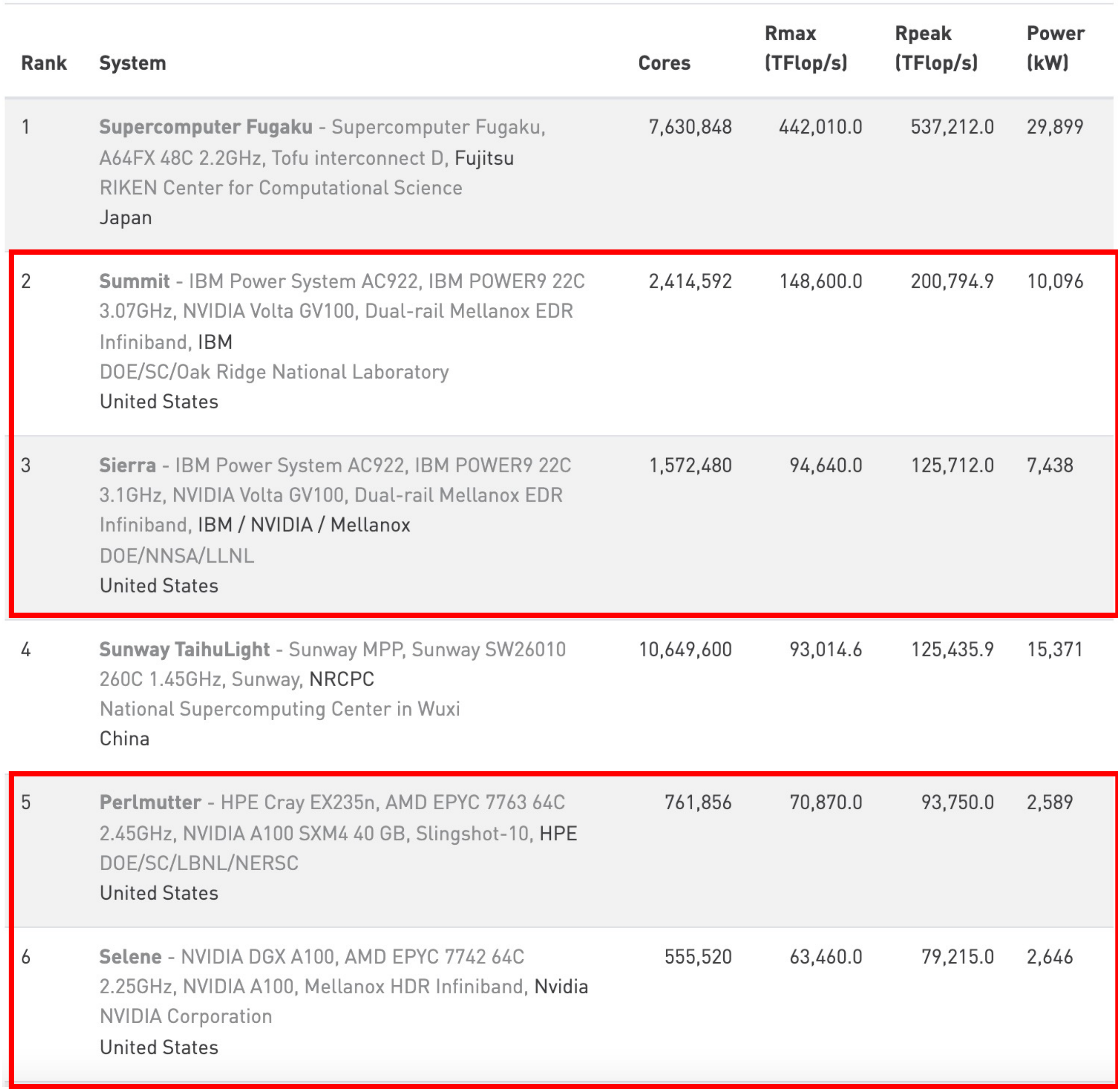
(Source: https://www.r-ccs.riken.jp/en/fugaku/project/outline)

Detailed report at: https://www.icl.utk.edu/files/publications/2020/icl-utk-1379-2020.pdf

Top500



- Graphic Processing Units (GPUs) processors play an important role in supercomputers

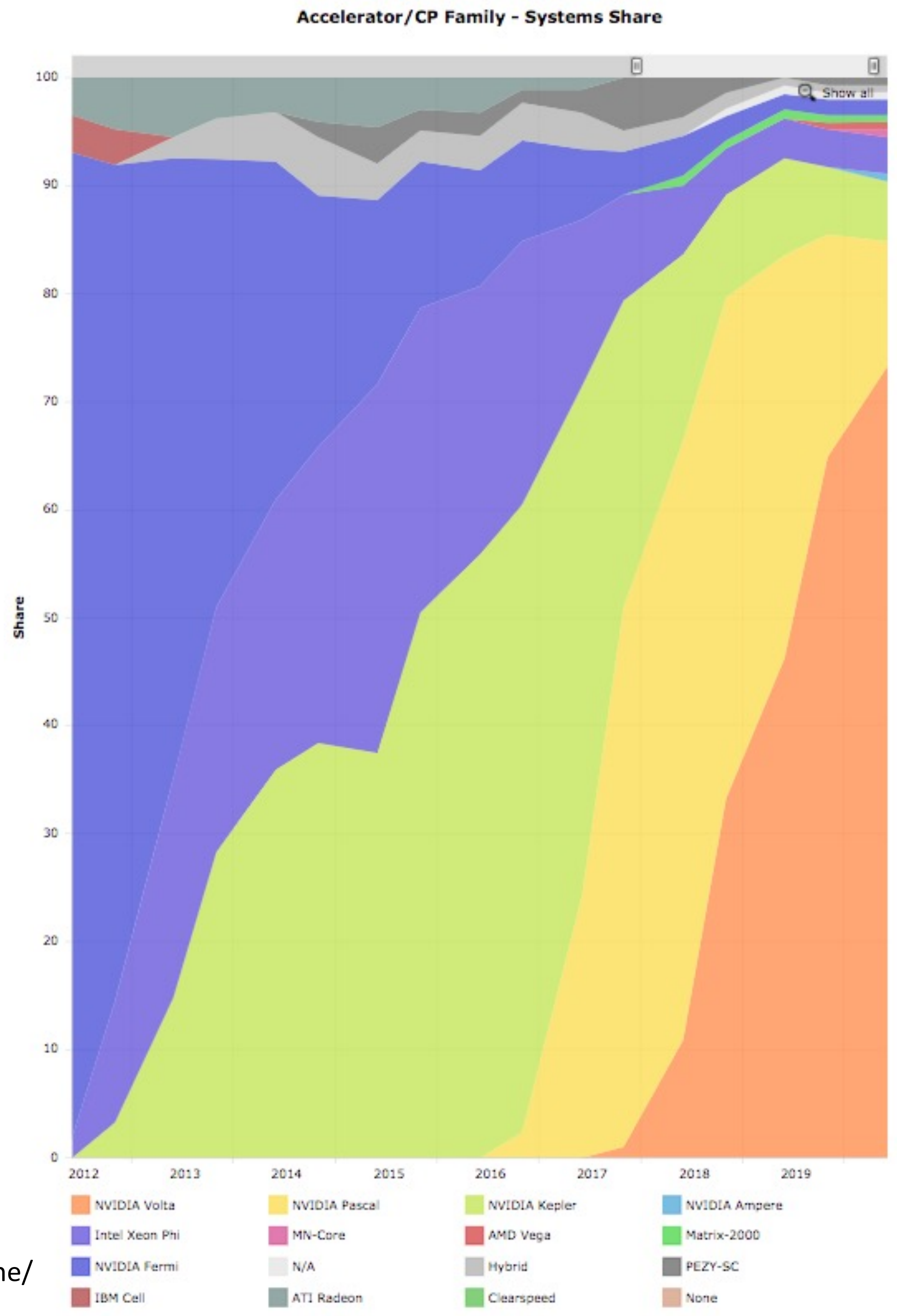


Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70,870.0	93,750.0	2,589
6	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646

Source: https://www.top500.org/statistics/perfdevel/

Top500

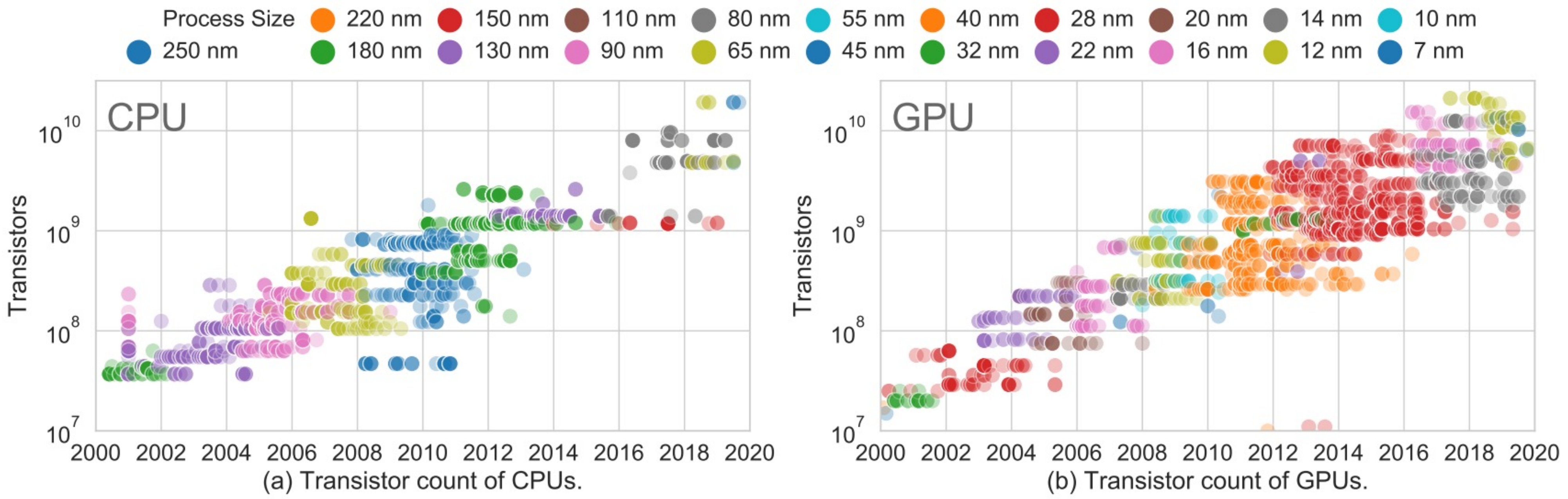
- ~92% systems are with GPUs in the Top500 list (June 2020)



Source: https://www.top500.org/statistics/overtime/

Development of Processors

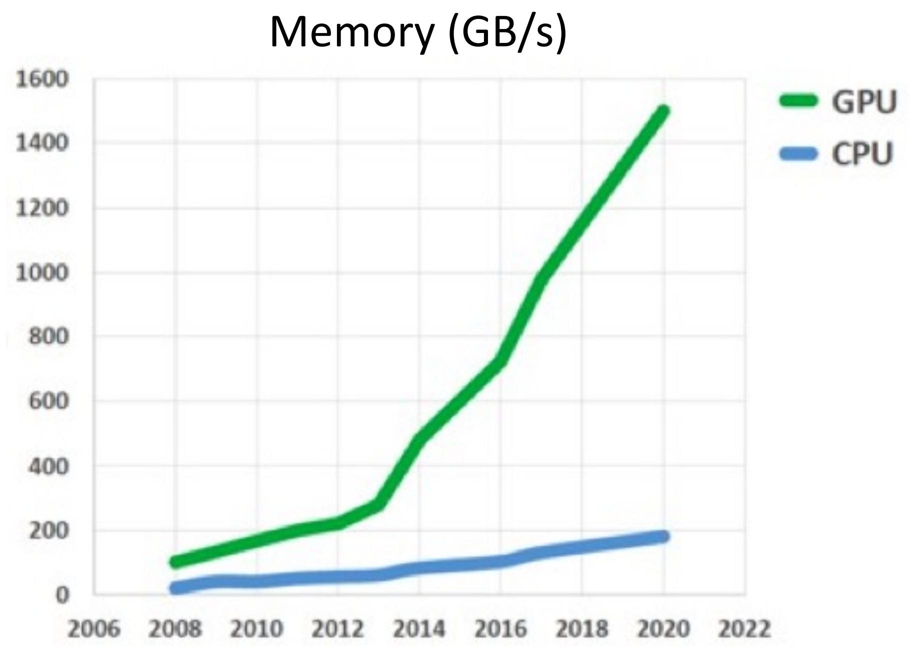
- Moore's law, 1965 by Intel CEO and co-founder Gordon Moore
  - the number of transistors in a dense integrated circuit (IC) doubles about every two years.
- Huang's law, 2018 by Nvidia CEO Jensen Huang
  - the performance of GPUs will more than double every two years

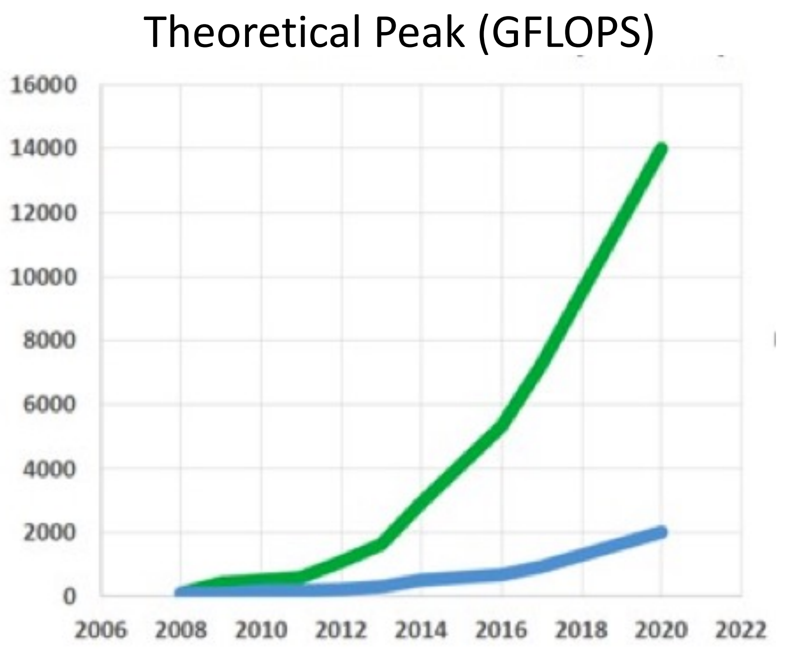


Source: https://arxiv.org/pdf/1911.11313.pdf

Transistor count comparison

Development of Processors





Performance comparison: CPU vs. GPU

Source: https://www.nextplatform.com/2019/07/10/a-decade-of-accelerated-computing-augurs-well-for-gpus/

Why needs HPC

- A large problem: input->process->output
  - Size: too big or Time: too slow
- A large number of HPC applications from science and engineering
  - Engineering: e.g., Computational Fluid Dynamics (CFD) simulation
  - Geoscience
  - Molecular Dynamics
  - Physics
  - Quantum Mechanics
  - Data Analytics
  - …

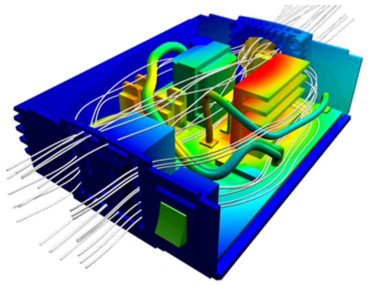
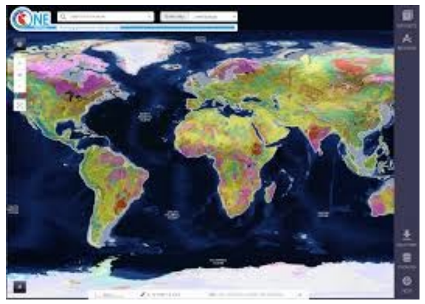
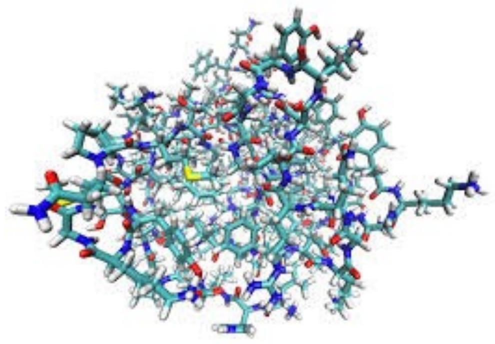
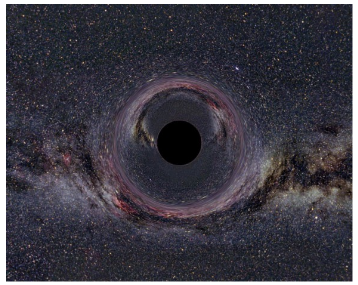
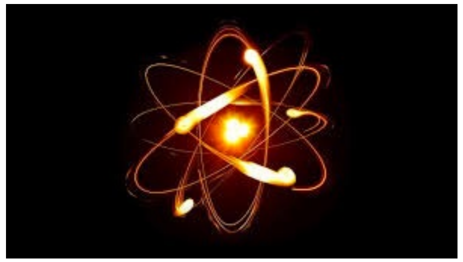
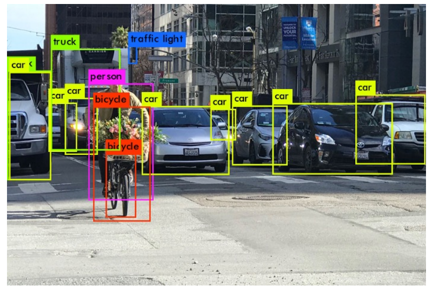
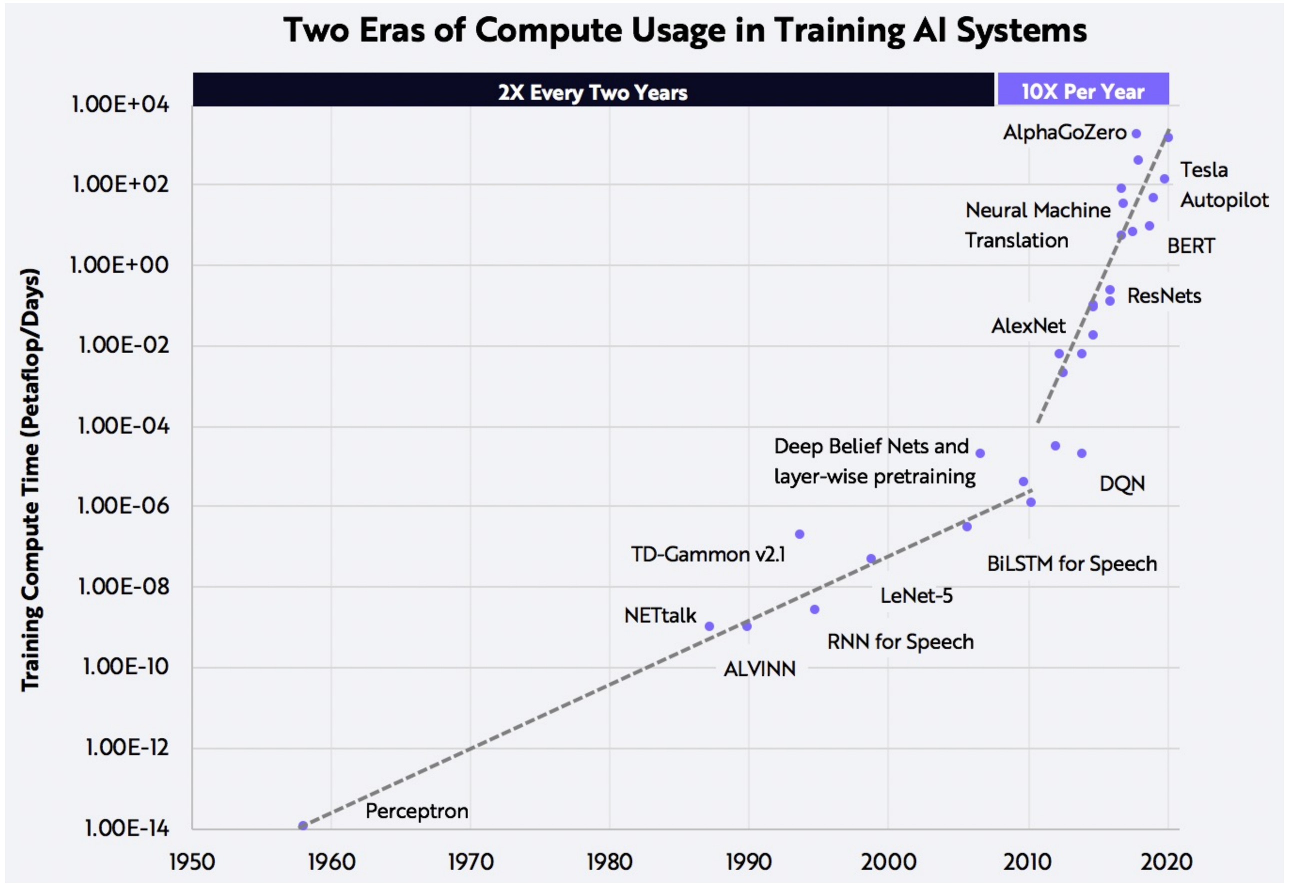







Image credits:
https://www.npd-solutions.com/cfd.html
http://www.onegeology.org/
https://www.epcc.ed.ac.uk/blog/2014/10/06/improving-tinker
http://www.sci-news.com/physics/naked-singularities-saddle-shaped-universe-04886.html
https://online.stanford.edu/courses/ee222-applied-quantum-mechanics-i
https://rishi30-mehta.medium.com/object-detection-with-yolo-giving-eyes-to-ai-7a3076c6977e

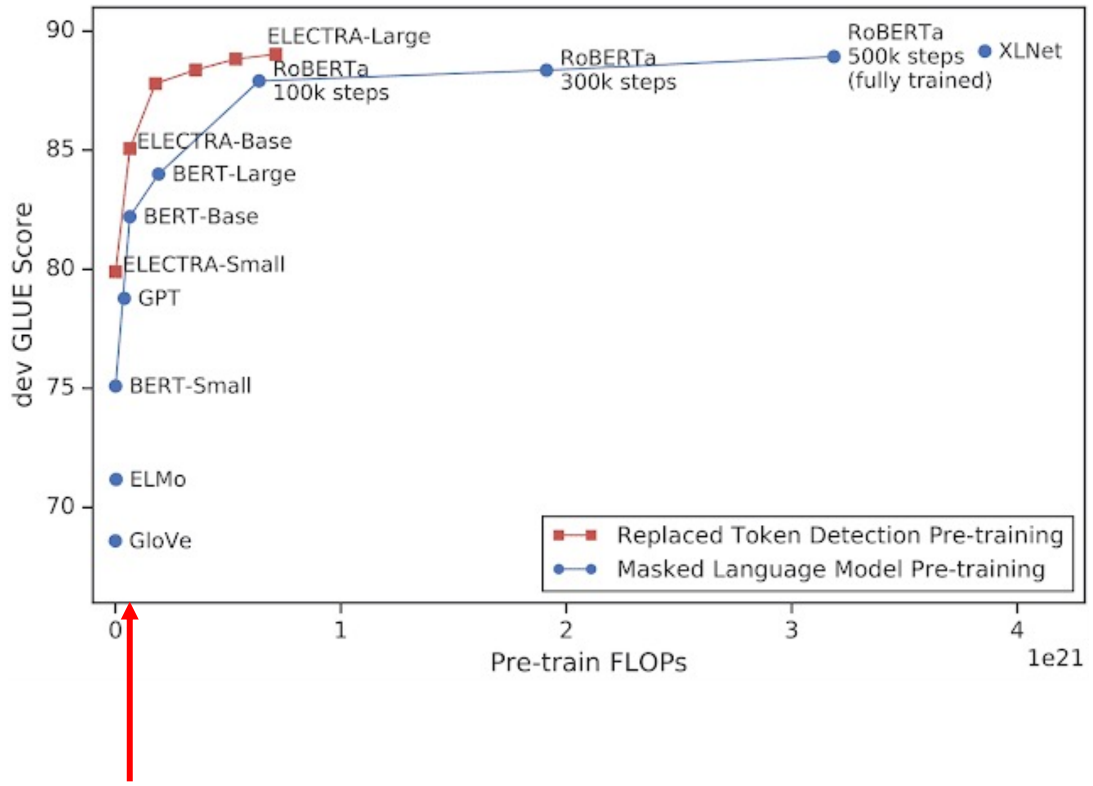
HPC in AI

- AI models are computing hungry



Source: https://ark-invest.com/articles/analyst-research/ai-training/

Source: https://ai.googleblog.com/2020/03/more-efficient-nlp-model-pre-training.html



- #1 supercomputer: 2.146176e18 FLOPS (Half precision for AI)!

HPC in AI

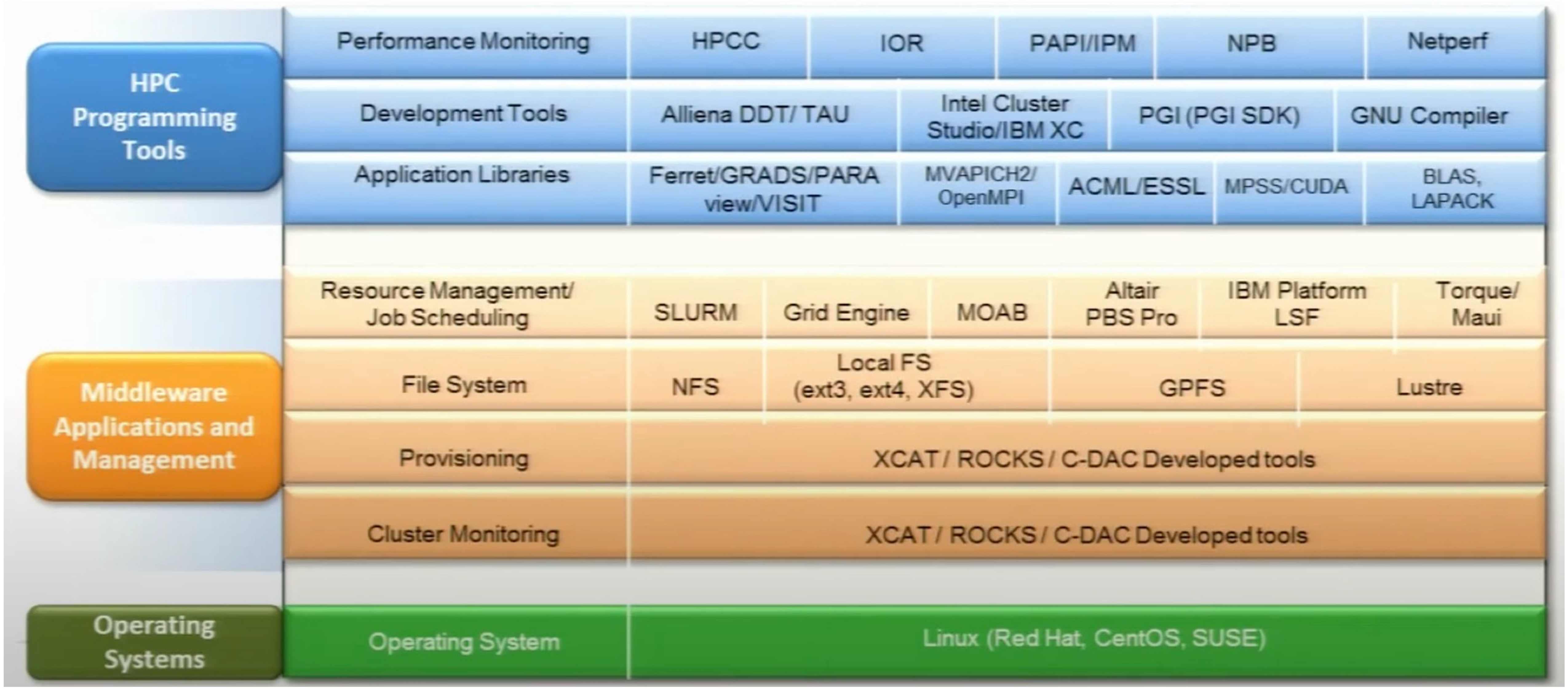
- AI models are computing hungry
- Training large models is still very slow on single GPU/TPU
  - E.g., training a ResNet-50 model with a Tesla P100 GPU takes **11 days**;
  - BERT pre-training with Google TPUv3 takes more than **1.5 months**

Training ResNet-50 to 75+% accuracy (90 epochs)

1 hour	20 minutes	6.6 minutes	1.2 minutes	45.6 seconds	28.2 seconds
June 2017	September 2017	July 2018	April 2019	July 2020	July 2020
Facebook	UC Berkeley	Tencent & HKBU	Fujitsu	Nvidia	Google
256 GPUs	2048 KNLs	2048 P40 GPUs	2048 V100 GPUs	1840 A100 GPUs	4096 TPUv3

December 2021	ResNet-50	BERT
	4320 A100 GPUs, NVIDIA: ~20s	4320 A100 GPUs, NVIDIA: ~14s

How to use HPC: Software Stack



HPC Programming Tools	Performance Monitoring	HPCC	IOR	PAPI/IPM	NPB	Netperf	
	Development Tools	Alliena DDT/ TAU	Intel Cluster Studio/IBM XC	PGI (PGI SDK)	GNU Compiler		
	Application Libraries	Ferret/GRADS/PARA view/VISIT	MVAPICH2/ OpenMPI	ACML/ESSL	MPSS/CUDA	BLAS, LAPACK	
Middleware Applications and Management	Resource Management/ Job Scheduling	SLURM	Grid Engine	MOAB	Altair PBS Pro	IBM Platform LSF	Torque/ Maui
	File System	NFS	Local FS (ext3, ext4, XFS)	GPFS	Lustre		
	Provisioning	XCAT / ROCKS / C-DAC Developed tools					
	Cluster Monitoring	XCAT / ROCKS / C-DAC Developed tools					
Operating Systems	Operating System	Linux (Red Hat, CentOS, SUSE)					

Source: https://nptel.ac.in/courses/128/106/128106014/

How to use HPC: Parallel Programming

- Parallel programming
  - Multi-threading: shared memory
    - **OpenMP** for CPUs, **CUDA** for GPUs
  - Multi-processing: distributed memory
    - **Message passing interface (MPI)**, Spark, etc.
  - Storage
    - Hadoop
  - Interconnect protocol
    - TCP/IP, RDMA, etc.
- Design lower complexity algorithms
  - FFT: Fast Fourier transform with complexity of $O(n\log n)$
    - A fast algorithm for computing discrete Fourier transform (DFT)
    - DFT complexity: $O(n^2)$
    - n is the data size
  - Matrix multiplication: A × B, dimension in: n × n
    - Typically $O(n^3)$
    - It can be reduced to $O(n^{2.37\underline{548}})$ [1] - $O(n^{2.37\underline{28596}})$ [2]
  - …

[1] Don et al., "Matrix multiplication via arithmetic progressions," Proceedings of the nineteenth annual ACM symposium on Theory of computing, 1987.
[2] Josh et al., "A Refined Laser Method and Faster Matrix Multiplication," SODA 2021