THE UNIVERSITY of EDINBURGH

epcc

HPC Architectures

Dr David Henty

d.henty@epcc.ed.ac.uk

Total Slides: 47

Content

- HPC Architecture course aims to cover:
  - **Basic components of HPC systems:** processors, memory, interconnect, storage.
  - **Brief history of HPC systems:** including Moore's Law.
  - **Classification of architectures:** SIMD/MIMD, shared vs distributed memory, clusters
  - **CPU design:** functional units, instructions sets, pipelining, branch prediction, ILP (superscalar, VLIW, SIMD instructions), multithreading.
  - **Caches:** operation and design features
  - **Memory:** operation and design features, including cache coherency and consistency
  - **Multicore CPUs:** including cache and memory hierarchy
  - **System software:** OSs, processes, threads, scheduling, batch systems.
  - **Accelerators:** such as GPGPUs and FPGAs - operation and design features
  - **Interconnects and storage:** operation and design features
  - **Current HPC architectures**
  - **Power Monitoring:** including use cases and efficiency

Practicalities

Timetable:

Monday 11:10-12:00 – practical.
Friday 14:10-16:00 – lectures (x2).

Assessments:

Class test: Multiple choice questions – 10% of course
Exam: 3 compulsory questions, equal marks per question – 90%

Multiple lecturers covering areas of expertise.

Resources:

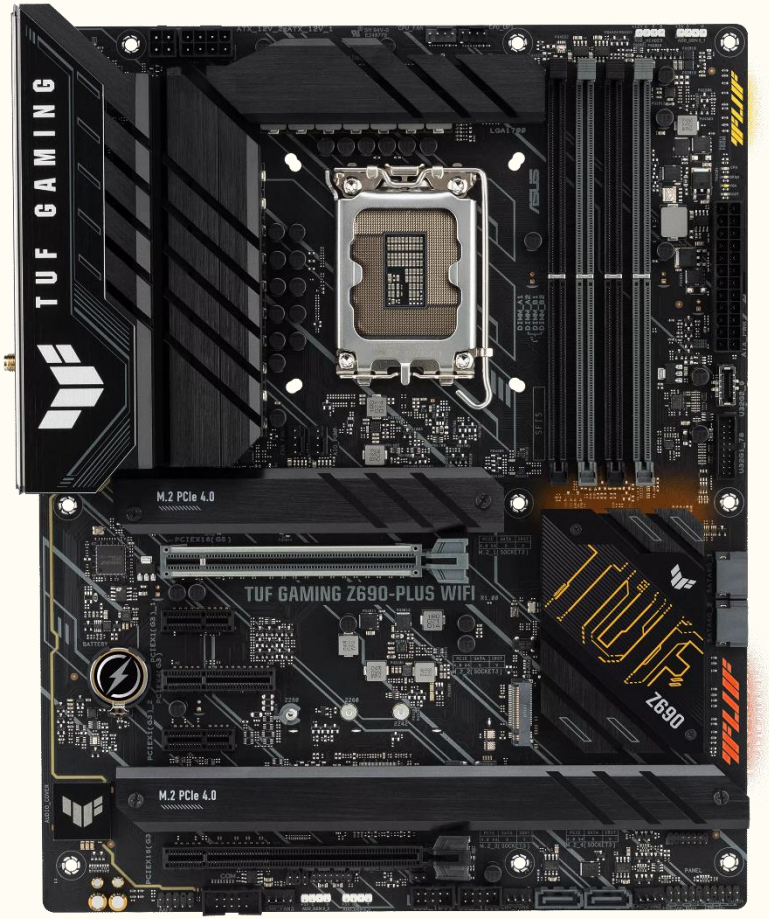
The Science of Computing- Victor Eijkhout,

- https://theartofhpc.com/istc.html

Why HPC?

- Scientific simulation and modelling have driven the need for greater computing power for many decades
  - more recently, Data Science and AI
- Single-core processors cannot be made that have enough resource for the simulations needed
  - making processors with faster clock speeds is difficult due to cost and power/heat limitations
  - expensive to put huge memory on a single processor
- Severe limitations on performance of *serial* computing
- Solution: *parallel* computing
  - divide up the work among numerous connected systems

Building Blocks of (Parallel) Computing



Four principal technologies which make up HPC systems:

- Processors:
  - to calculate
  - includes accelerators like GPUs
- Memory:
  - for temporary storage of data
- Storage:
  - disks/drives for storing input/output data and tapes for long term archiving of data
- ***Interconnect:***
  - ***so processors can talk to each other (and the outside world)***

What do we mean by "performance"?

- For scientific and technical programming use FLOPS
  - Floating Point OPerations per Second
  - 1.324398404 + 3.6287414 = ?
  - 2.365873534 * 2443.3147 = ?
- Modern supercomputers typically measured in PFLOPS (PetaFLOPS)
  - Kilo, Mega, Giga, Tera, Peta, Exa = 10^3, 10^6, 10^9, 10^{12}, 10^{15}
  - a few exascale computers now exist
- Other disciplines have their own performance measures
  - forecast days per day
  - training datasets analysed per second
  - frames per second, database accesses per second, …

Processors

Basic functionality:

- execute instructions to perform arithmetic operations (integer / floating point)
- load data from memory and store data to memory
- decide which instructions to execute next

Arithmetic is performed on values in *registers*:

- local storage in the processor typical size ~100 values.
- moving data between memory and registers must be done explicitly by load and store instructions
- separate integer and floating-point registers

Basic characteristics:

- Clock speed
- Peak floating-point capability (FLOPS)

Processors 2

Clock speed determines rate at which instructions are executed

- modern chips are around 2-3 GHz
- integer and floating-point calculations can be done in parallel
- can also have multiple instruction, e.g. simultaneous add and multiply
- peak flop rate is clock rate x floating-point operations per clock cycle
- e.g. 2 GHz x 8 flops/cycle = 16 GFLOPS

Whole series of hardware innovations

- pipelining
- out-of-order execution, speculative execution

Details become important for extracting high performance

- most features are fairly generic

Moore's Law

"CPU power doubles every 2 years"

- Strictly speaking, applies to transistor density

Held true for ~35 years

- People have predicted its demise many times
- It hasn't happened yet but may be slowing.

Increased performance from increases in parallelism AND clock rate

- Fine grain parallelism (pipelining)
- Medium grain parallelism (superscalar, hardware multithreading)
- Coarse grain parallelism (multiple processors on a chip)
- First two seem to be (almost) exhausted: main trend is now towards multicore
- Clock speed virtually stalled due to power consumption / heat dissipation

Accelerators

Current popular trend is to include additional special purpose processors alongside the main CPU

- Large numbers of relatively simple cores or vector processors
- High memory bandwidth

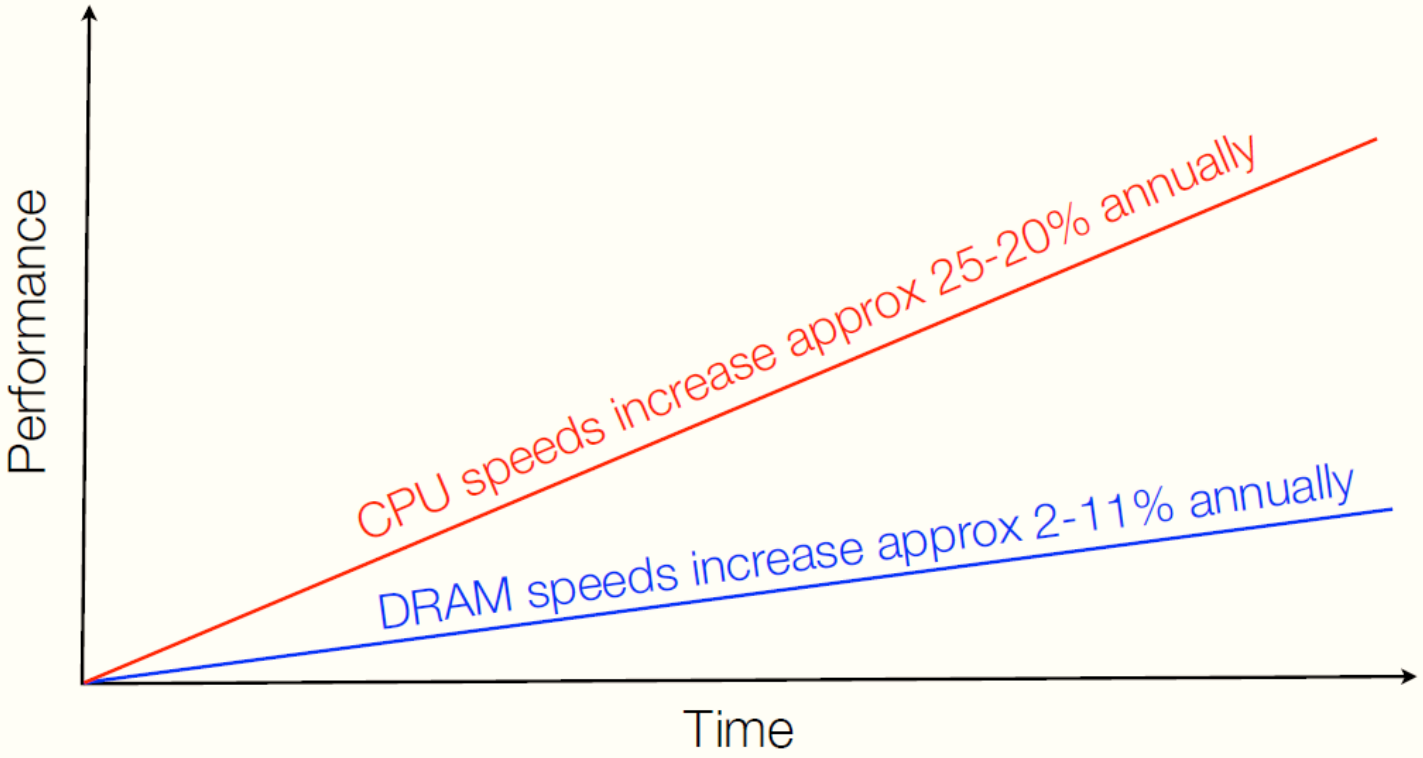
Much of current interest is focussed on GPGPUs (general purpose graphics processing units)

- Low cost due to high mass market volumes
- High bandwidth graphics memory

May not be a long-term solution

- Industry trends suggest tighter integration of simple cores onto a single piece of silicon

Performance Trend – CPU v RAM



Main Memory

Memory speed is often the limiting factor for HPC applications.

- Keeping the CPU fed with data is the key to performance.

Memory is a substantial contributor to the cost of systems.

- Typical HPC systems have a few Gbytes of memory per processor.
- Technically possible to have much more than this, but it is too expensive and power-hungry.

Basic characteristics

- Latency: how long you have to wait for data to arrive.
- Bandwidth: how fast it actually comes in.
- Ballpark figures: 100's of nanoseconds and several Gbytes/s.

Cache memory

Memory latencies are very long

- 100s of processor cycles
- fetching data from main memory over 100x slower than arithmetic

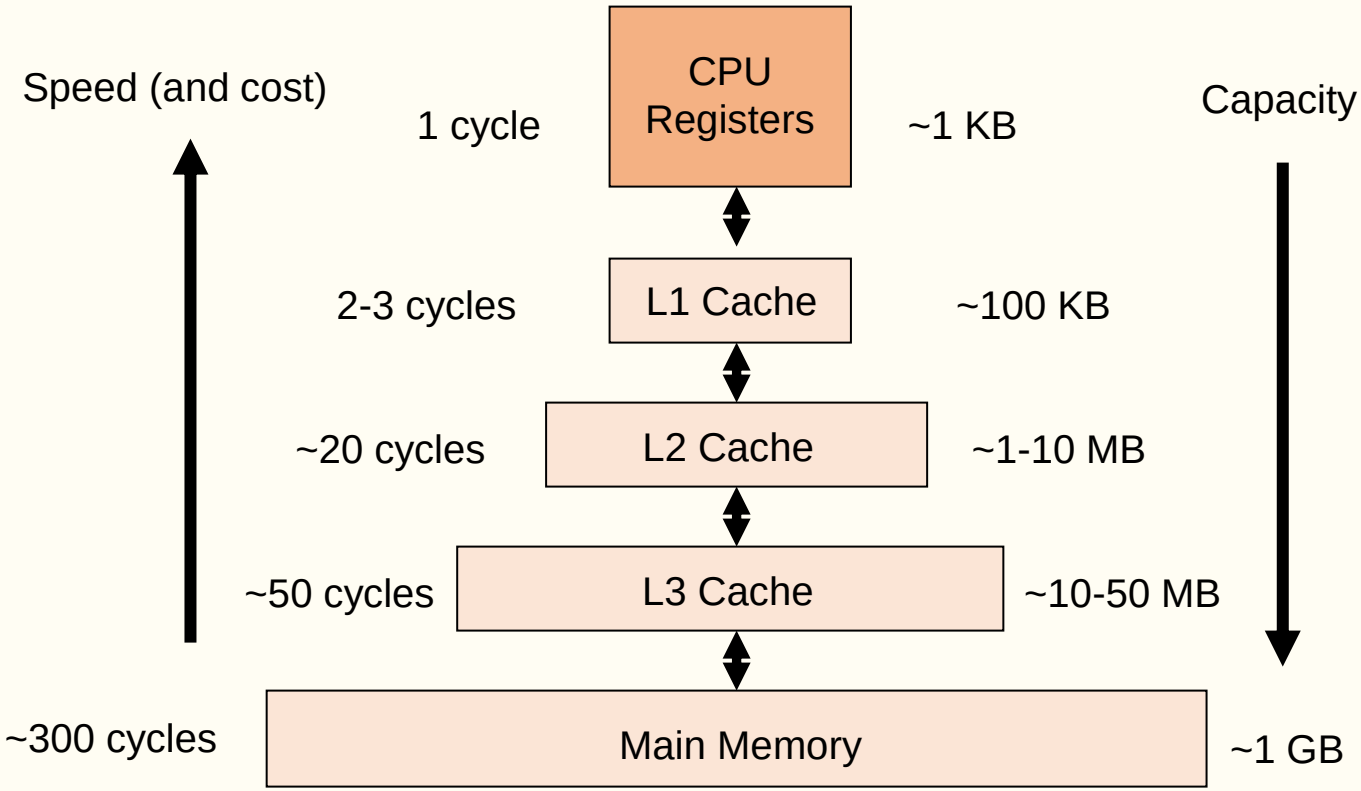
Solution: introduce cache memory

- much faster than main memory
- ...but much smaller than main memory
- …also more expensive and power hungry.
- keeps copies of recently used data

Modern systems use a hierarchy of two or three levels of cache

- top 2 levels typically on-chip

Memory hierarchy



Interconnect

- CPUs need to be able to exchange data.
- For distributed systems we connect via external hardware.
- Characterised by:
  - Latency (around a microsecond).
  - Bandwidth (typically several GB/s)

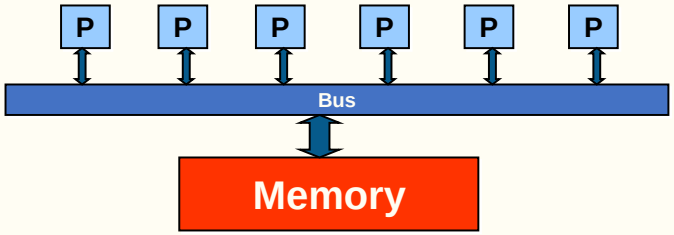
Storage

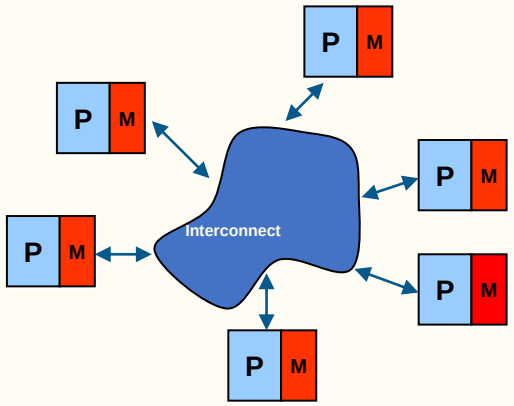
Disks are now very cheap.	pay extra for high performance
	may only need small quantities of very fast disk
Writing to disk can be relatively slow.	often a bottleneck for the largest simulations, especially AI applications
	rates typically limited by the interconnect bandwidth (around tens of Gbytes/sec)
Redundant Array of Inexpensive Devices (RAID).	gang together commodity disks
	looks like a single device
Can be used to increase:	volume - concatenation
	bandwidth - simultaneous writes to many disks
	reliability - parity disk, mirroring

Longer Term Archiving

- Tape store
  - Hierarchical Storage Management
  - stage data from tape robot to disk
  - sometimes done transparently
- Disk farms
  - viable alternative to tapes
  - RAID gives reliability
- Storage Area Networks
  - devices attached to a network
  - accessed as if they were local
  - share devices between machines or relocate at will

Types of HPC systems





Shared-memory: OpenMP

- Multiple processors share a single memory space
- Simple to program for many problems
- Scaling is problematic

Distributed memory: MPI

- Each processing unit has its own memory space
- Excellent scaling properties
- Can be more complex to program due to explicit communications

Accelerators (GPUs)

- Specialist processing units attached to main CPU
- Can be difficult to extract good performance for general scientific computing
- Huge growth in use of AI / Machine Learning

Distributed Shared Memory (clusters)

Dominant architecture is a hybrid of these two approaches:

Distributed Shared Memory.

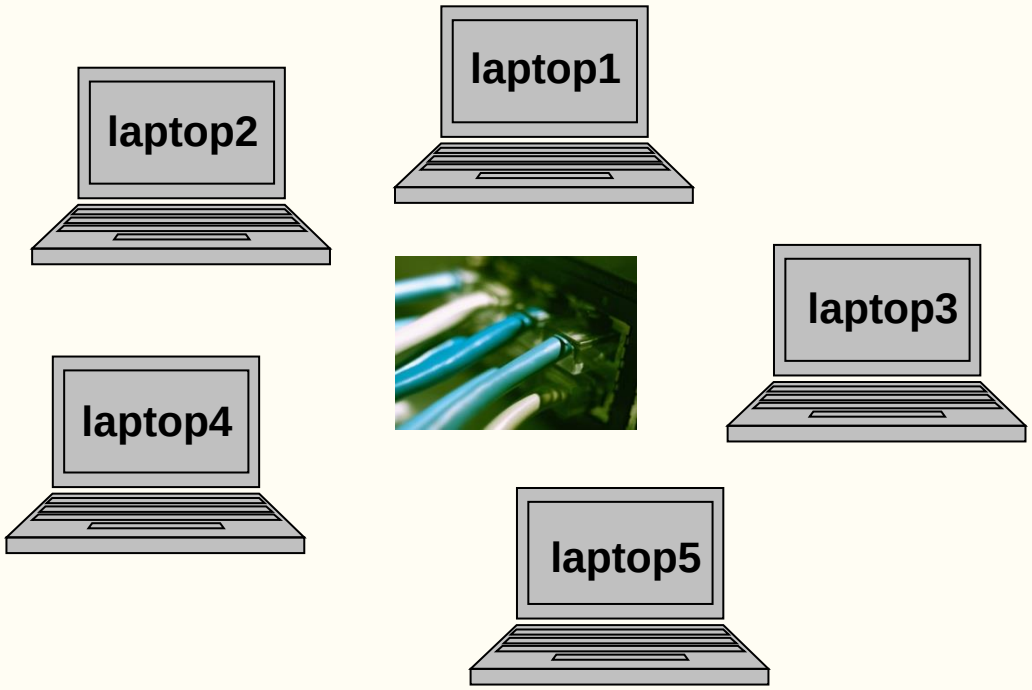
- Due to most HPC systems now being built from commodity hardware – trend to multicore processors.
- Each Shared memory block is known as a *node*.
- Usually 16-128 processors per node.
- Nodes can also contain accelerators.

Majority of users try to exploit in the same way as for a purely (low-multicore) distributed machine

- As the number of cores per node increases this can become increasingly inefficient…
- …and programming for these machines can become increasingly complex

Generic Parallel Machine

- Good conceptual model is collection of multicore laptops
  - come back to what "multicore" actually means later on …
- Connected together by a network



- Each laptop is called a *compute node*
  - each has its own operating system and network connection
- Suppose each node is a quadcore laptop
  - total system has 20 processor-cores

Differences from Cloud computing

Performance

Clouds usually use virtual machines which add an extra layer of software.

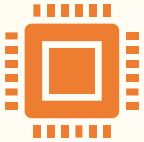
In cloud you often share hardware resource with other users – HPC access is usually exclusive.

Tight-coupling

HPC parallel programming usually assumes that the separate processes are tightly coupled

Requires a low-latency, high-bandwidth communication system between tasks

Cloud usually does not usually have this



Programming models

HPC use high-level compiled languages with extensive optimisation.

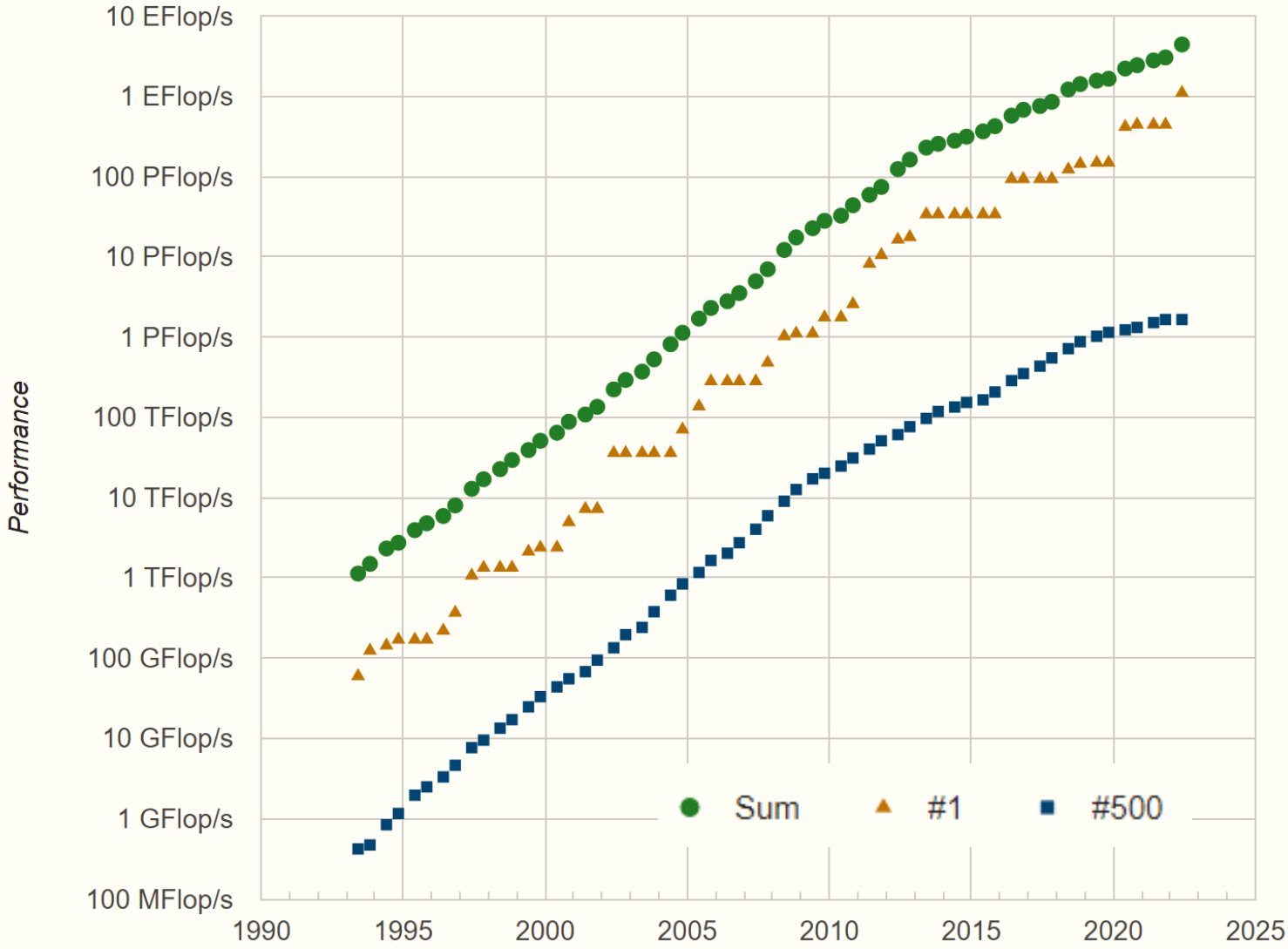
Cloud often based on interpreted/JIT.

Top500 list: www.top500.org

- Announced at International Supercomputing Conference (Europe, June) and Supercomputing Conference (US, November)
  - how fast can you solve an enormous set of linear equations (Linpack benchmark)

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	8,699,904	1,206.00	1,714.81	22,786
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
4	**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
5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107

Performance Development



FLOPS

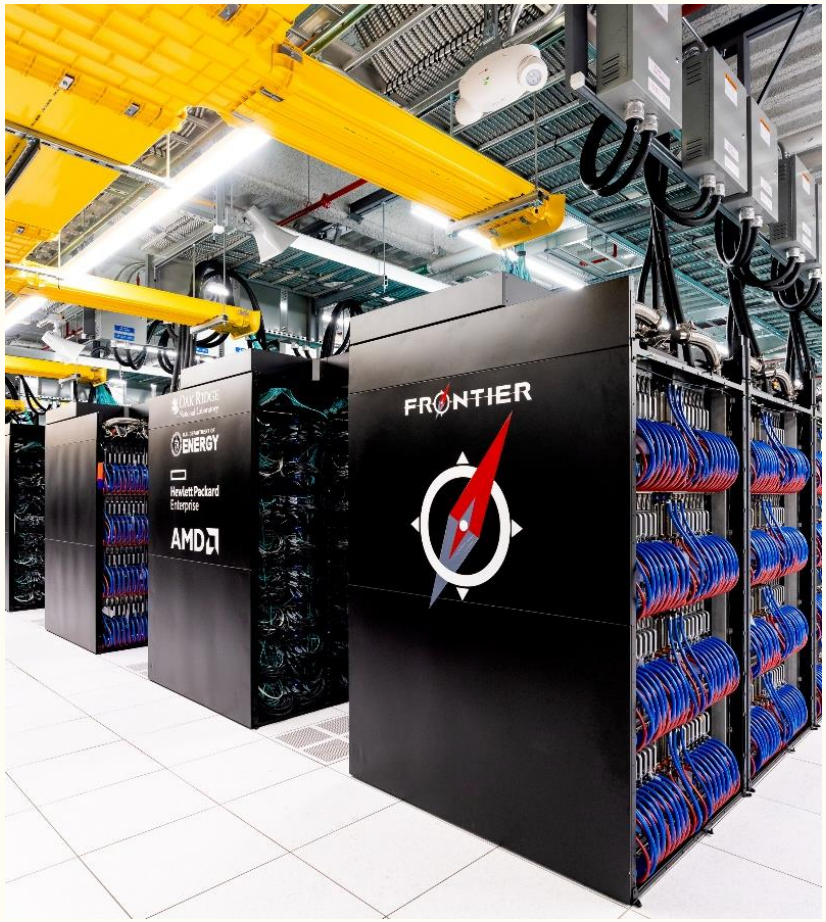
- Yotta: 10^{24}
- Zetta: 10^{21}
- Exa: 10^{18}
- Peta: 10^{15}
- Tera: 10^{12}
- Giga: 10^{9}
- Mega: 10^{6}
- Kilo: 10^{3}

Source: Top500.org

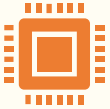
Frontier

Number 1 machine in the world

- 606,208 CPU cores
  - 9,472 AMD EPYC 64 core CPUs
- ~ 8.3 Million GPU cores
  - 37,888 Radeon Instinct MI250X GPUs
- 9472 Nodes:
  - 1 CPU and 4 GPUs per node
  - 4TB flash memory per node, 12GB RAM per GPU
- 1.7 EFLOPS peak performance
  - First Exascale machine at 1.2 EF
- 23MW power – also topped Green500 when unveiled



But why?



Computational Simulation, mostly

Focus on science and the problem to be solved

Simulation of scientific problem or environment

Input of real data

Output of simulated data

Parameter space studies

Wide range of approaches

Explore universe through simulation rather than experimentation

Test theories

Predict or validate experiments

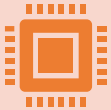
Simulate “untestable” science

Reproduce “real world” in computers

Generally simplified

Dimensions and timescales restricted

HPC - Parallelism



Simulation science drives computing power

- Consistently need more computation power than available
- Runtime of months on a single processor not uncommon
- Parallel programs often start out as serial programs

Why not just make a faster chip?

- Worked in the past, but starting to run into problems

Solution

- Use many CPUs cooperatively on the same problem
- HPC computing synonymous with parallelism

Quantifying Performance

Serial computing concerned with complexity

- how execution time varies with problem size *N*
- adding two arrays (or *vectors*) is *O(N)*
- matrix times vector is $O(N^2)$, matrix-matrix is $O(N^3)$

Look for clever algorithms

- naïve sort is $O(N^2)$
- divide-and-conquer approaches are *O(N log (N))*

Parallel computing *also* concerned with scaling

- how time varies with number of processors *P*
- different algorithms can have different scaling behaviour
- but always remember that we are interested in minimum time!

Performance Measures

T(N,P) is execution time for size *N* on *P* processors

Speedup:	typically, *S(N,P)* < *P*	$S(N,P) = \frac{T(N,1)}{T(N,P)}$
Parallel Efficiency:	typically, *E(N,P)* < 1	$E(N,P) = \frac{S(N,P)}{P} = \frac{T(N,1)}{PT(N,P)}$
Serial Efficiency:	Typically, E(N) <= 1	$E(N) = \frac{T_{best}(N)}{T(N,1)}$

Parallel Scaling

Scaling describes how the runtime of a parallel application changes as the number of processors is increased

Can investigate two types of scaling:

Strong Scaling - increasing P, constant N.

Problem size stays same as number of processors increase.

Decreasing the work per processor.

Weak Scaling - increasing P, increasing N.

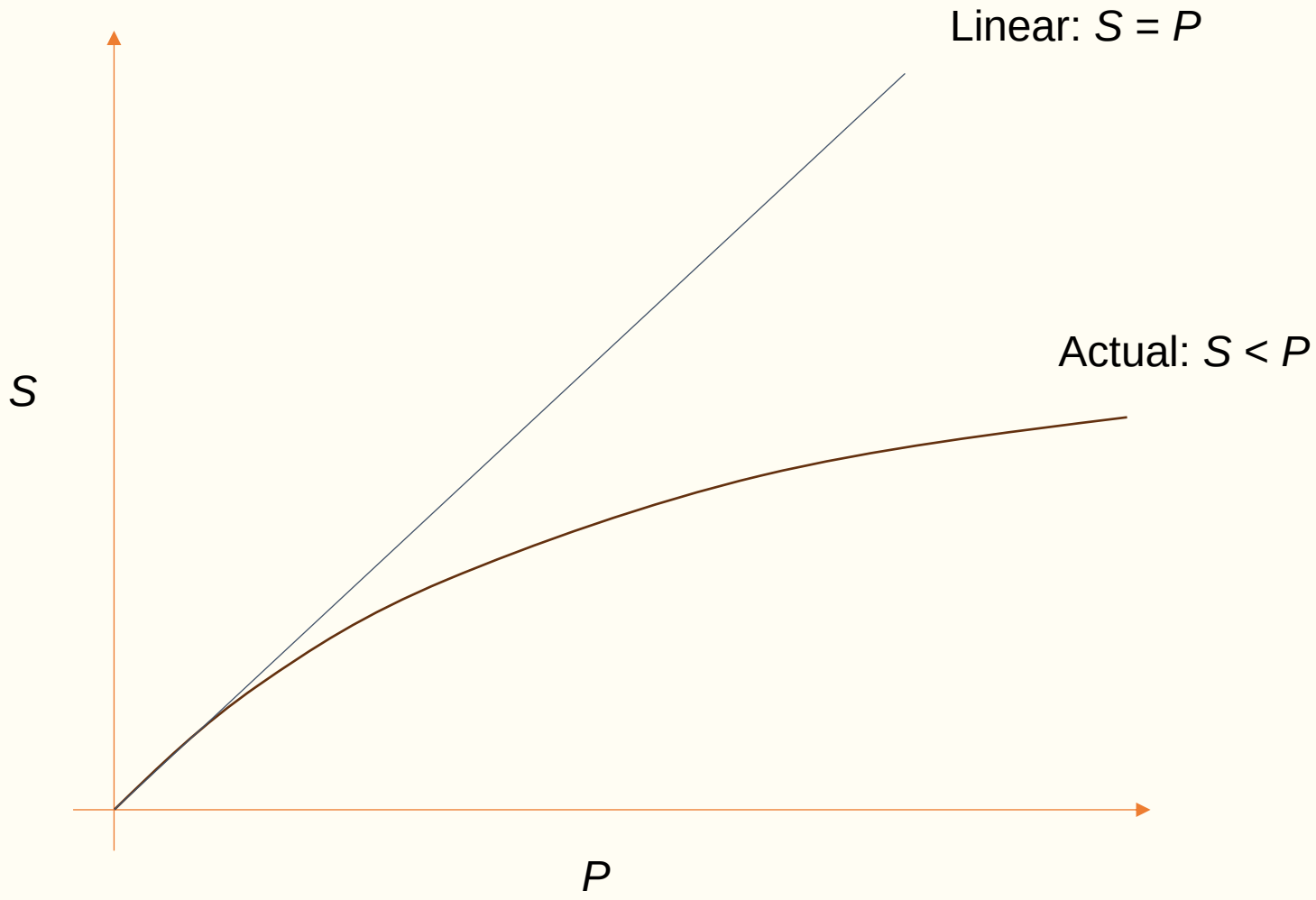
Problem size increases at same rate as the number of processors.

Work per processor stays the same.

Strong Scaling Example

- Have developed a parallel program
  - want to solve a given problem as fast as possible
- Calculation takes 12 hours on 1 processor
  - now run *the same problem* on 10 processors
  - would hope it takes 1.2 hours (speedup of 10)
- We measure runtime of 1.5 hours on 10 processors
  - actual speedup is 120/15 = 8
  - parallelisation is not perfect
  - parallel efficiency = 8/10 = 0.8 (80% of the best we could expect)
- But ... (and we normally ignore this part!)
  - original serial program took only 9 hours
  - serial efficiency is 90/120 = 0.75 (serial code is 25% faster)

Typical strong scaling graph



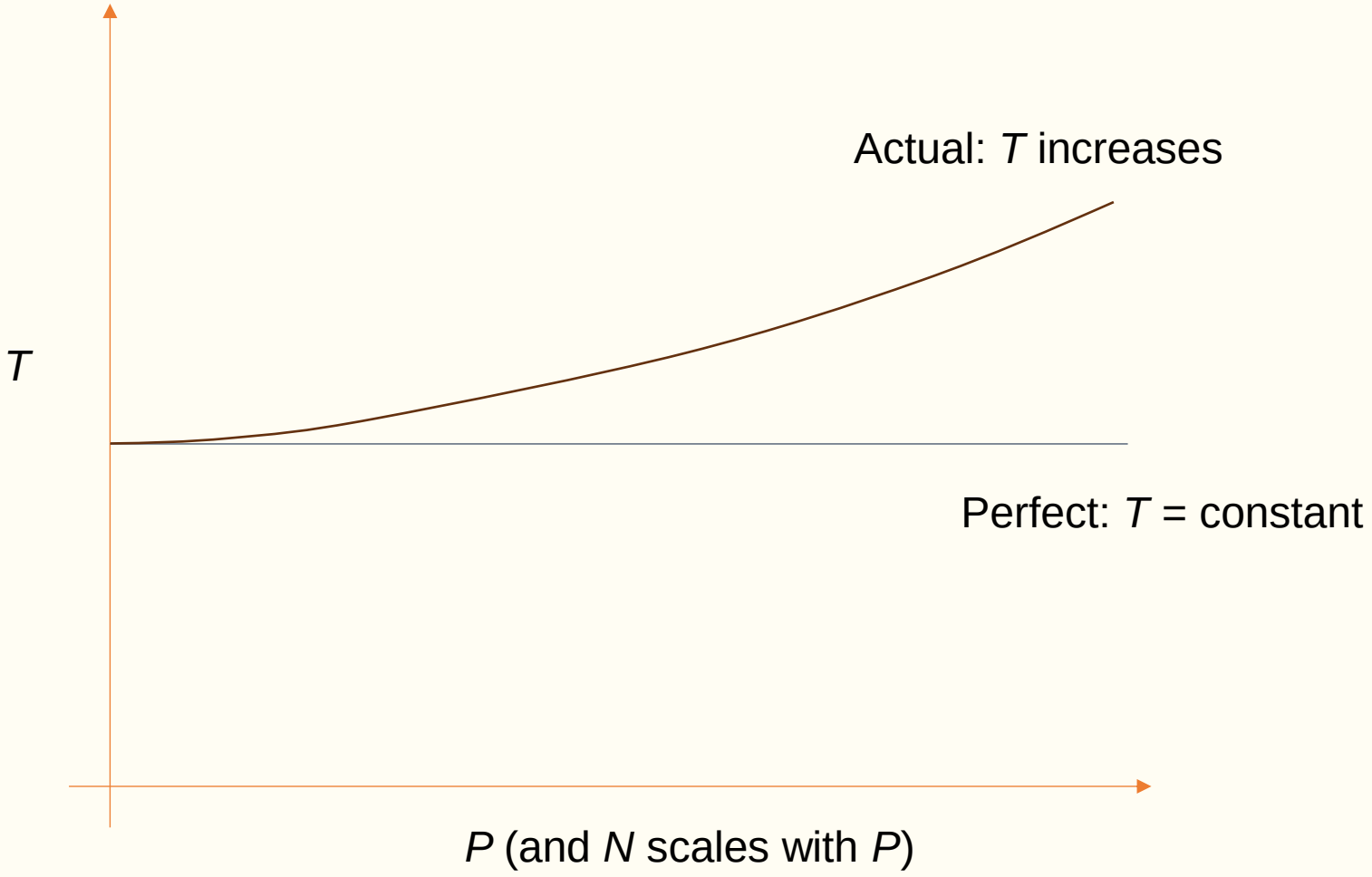
Weak Scaling Example

- Have developed a parallel program
  - want to run bigger problems but in the same time
  - e.g. want to produce the best possible weather forecast for tomorrow

- Calculation takes 12 hours on 1 processor
  - Now multiple both *P* **and** *N* by 10
  - run a problem 10 times larger on 10 times as many processors
    - would hope it still takes 12 hours (constant time)

- We measure runtime of 15 hours on 10 processors
  - parallelisation is not perfect
  - overheads mean an increase in time

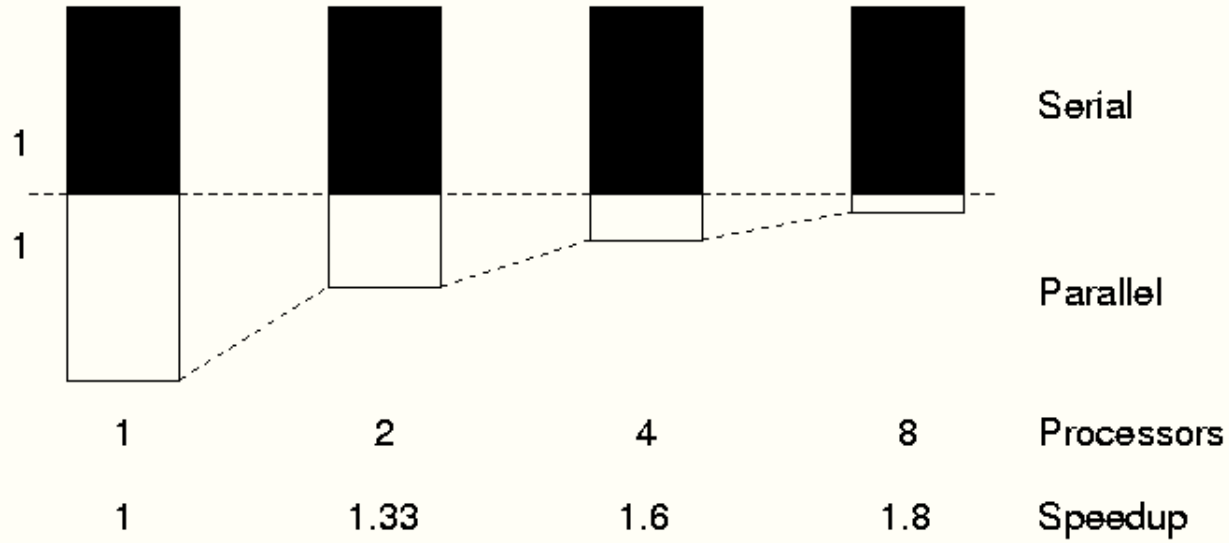
Typical weak scaling graph



The Serial Component

Amdahl's law:

> *"the performance improvement to be gained by parallelisation is limited by the proportion of the code which is serial"*
>
> Gene Amdahl, 1967



Amdahl's law

Assume a fraction α is completely serial
- time is sum of serial and potentially parallel

Parallel time

$$T(N,P) = \alpha T(N,1) + \frac{(1-\alpha)T(N,1)}{P}$$

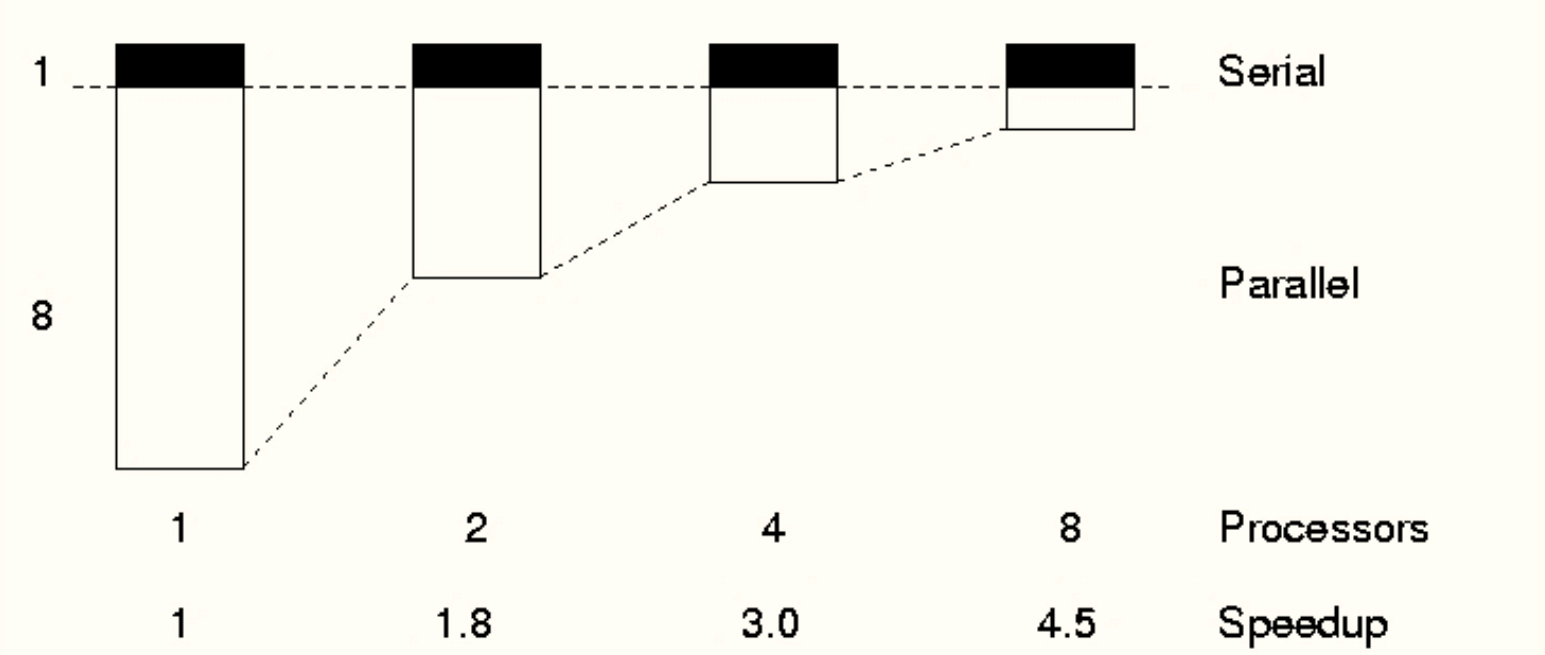
Parallel speedup

$$S(N,P) = \frac{T(N,1)}{T(N,P)} = \frac{P}{\alpha P + (1-\alpha)}$$

- for α = 0, *S* = *P* as expected (i.e. *E* = 100%)
- otherwise, speedup limited by 1/ α for any *P*
- impossible to effectively utilise large parallel machines?

Gustafson's Law

Need larger problems for larger numbers of processors



Utilising Large Parallel Machines

Assume parallel part grows with *N* (i.e is *O*(*N*)); serial part constant with *N* (i.e. *O*(1))

Time

$$T(N,P) = T_{serial}(N,P) + T_{parallel}(N,P)$$
$$= \alpha T(1,1) + \frac{(1-\alpha)NT(1,1)}{P}$$

Speedup

$$S(N,P) = \frac{T(N,1)}{T(N,P)} = \frac{\alpha+(1-\alpha)N}{\alpha+(1-\alpha)\frac{N}{P}}$$

Scale problem size with CPUs, ie set *N* = *P* ***Weak Scaling***

Speedup

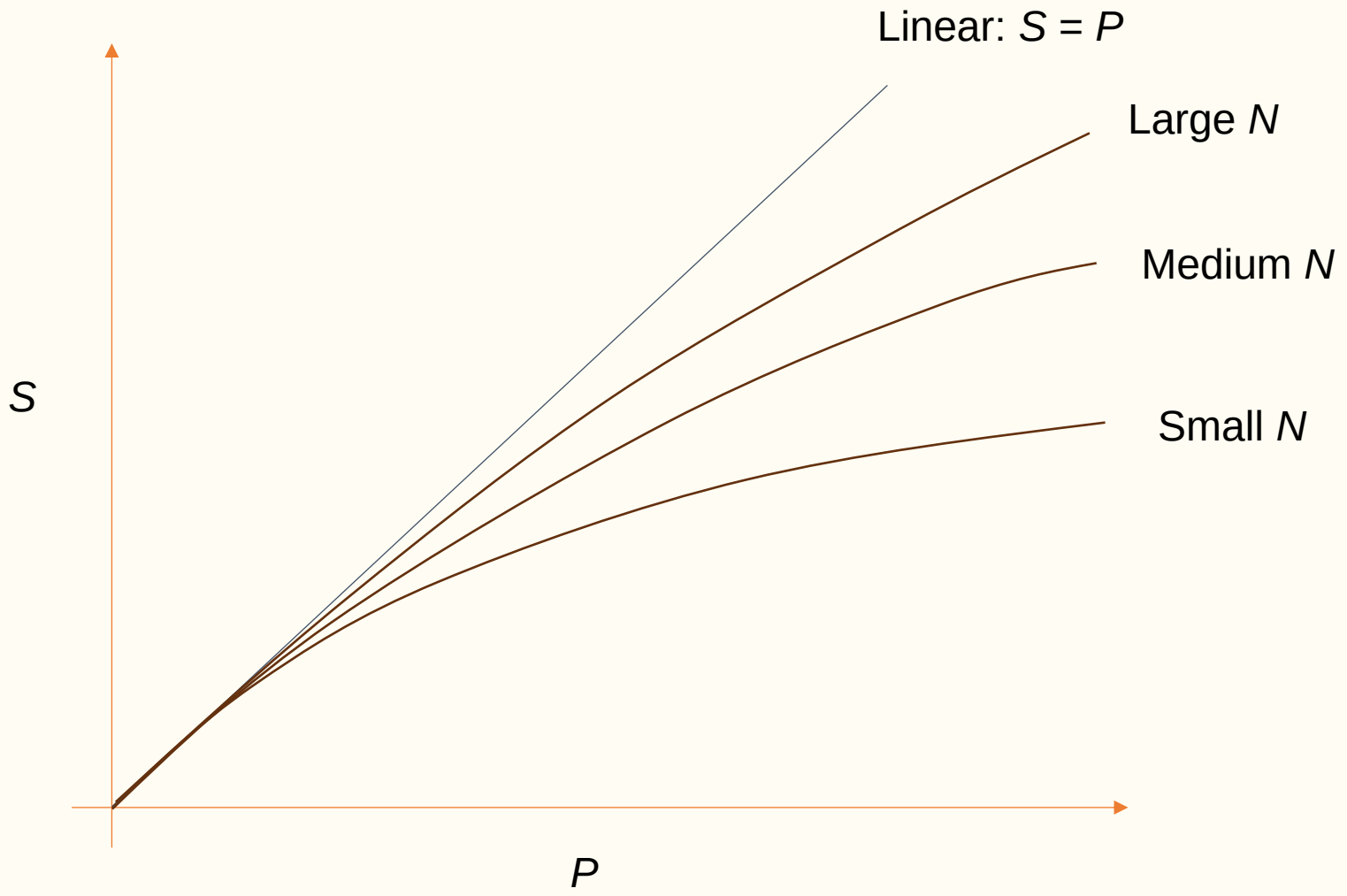
$$S(P,P) = \alpha + (1-\alpha)P$$

Efficiency

$$E(P,P) = \frac{\alpha}{P} + (1-\alpha)$$

Maintain constant efficiency (1-α) for large *P*

Strong scaling showing Gustafson's Law



Performance Summary

Definitions to Remember:

Speed-up equations

Efficiency equations

Amdahl's Law

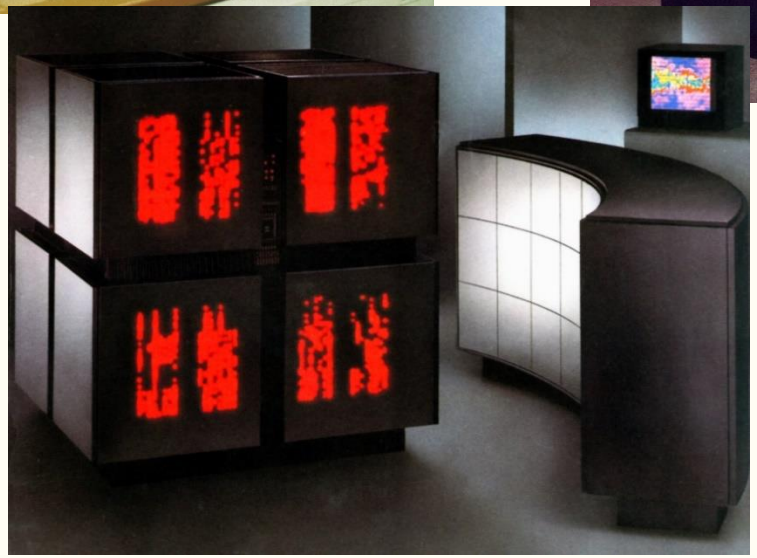
Gustafson's Law

Parallel Computers at Edinburgh

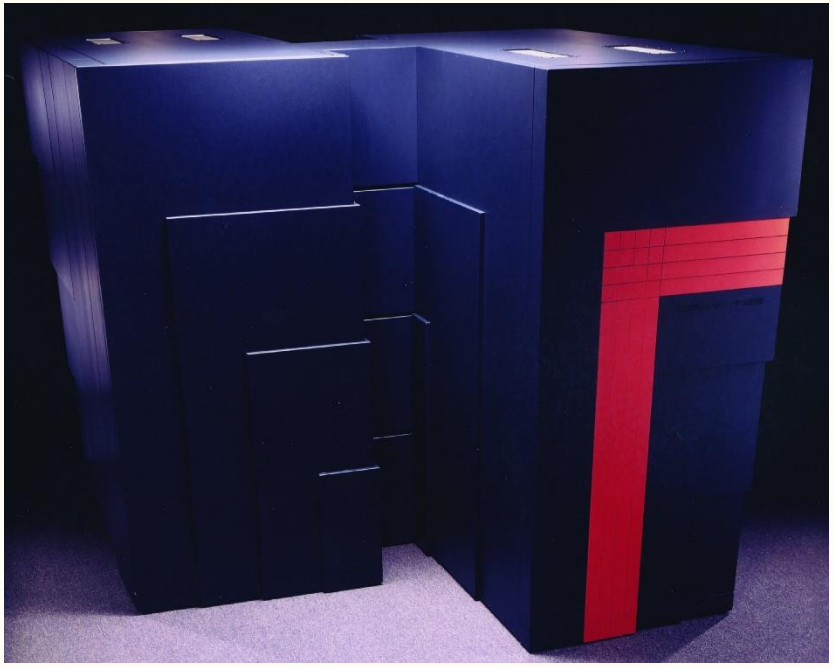
- **Nice summary:**
  https://www.archer2.ac.uk/community/outreach/materials/history_of_machines

- 1981 ICL DAP (SIMD: 4,096 processors) - 0.03 GFlops
- 1991 TMC CM-200 (SIMD: 16,000 processors)
- 1994 Cray T3D (MIMD-NUMA: 512 processors) + Cray Y-MP (Vector) - 76 GFlops
- 1997 Cray T3E (MIMD-NUMA: 344 processors)
- 2005 IBM BlueGene/L (MIMD-DM: 2048 cores) - 5.6 TFlops
- 2007 Cray XE6 (MIMD-DM: 90,112 cores) - >800 TFlops
- 2013 Cray XC30 (MIMD-DM: 118,080 cores) – ARCHER >2.5 PFlops
- 2016 HPE/SGI 8600 (MIMD-DM: 10,080 cores + GPU) – Cirrus
- 2021 HPE Cray Ex/Shasta (MIMD-DM: 748,544 cores) – ARCHER 2 >20 PFLops

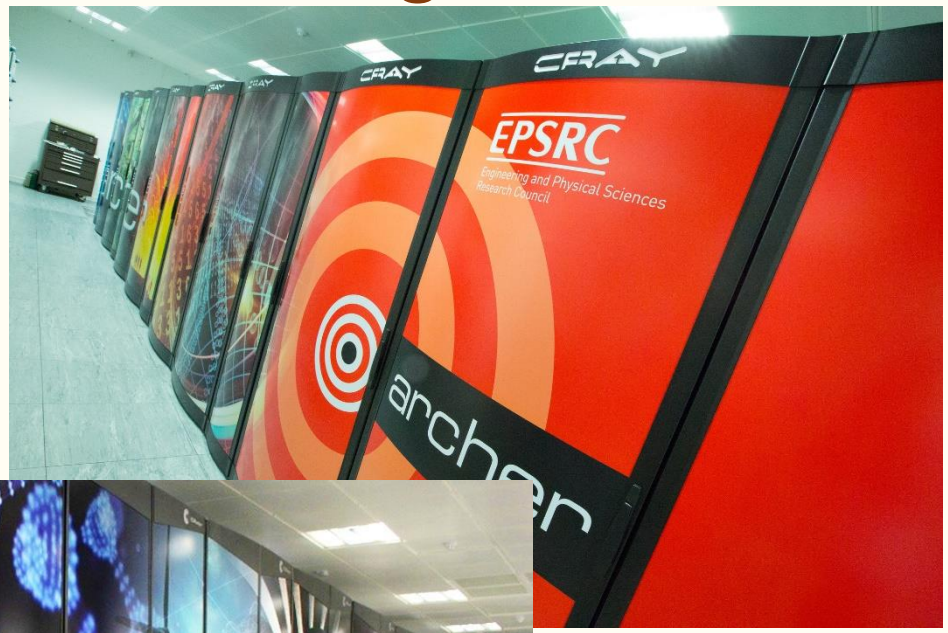
Early Parallel Computers at Edinburgh



More Parallel Computers at Edinburgh



Recent Computers at Edinburgh



HPE Cray EX (Cray Shasta)

Facts and Figures

Lifetime: 2020 - ?
5,848 nodes, each with 2 × AMD Zen2 7742, 2.25 GHz, 64-core CPU
Total 748,544 cores in 23 cabinets
Peak Performance: 28 PFLOP/s (estimated)
Architecture: MIMD-DM
Memory: 1.5 PetaBytes
Programming: MPI

Notes

“Slingshot” interconnect topology
Same base design as Frontier, but no GPUs!

Experience of EPCC

New generation of machines every 3-4 years

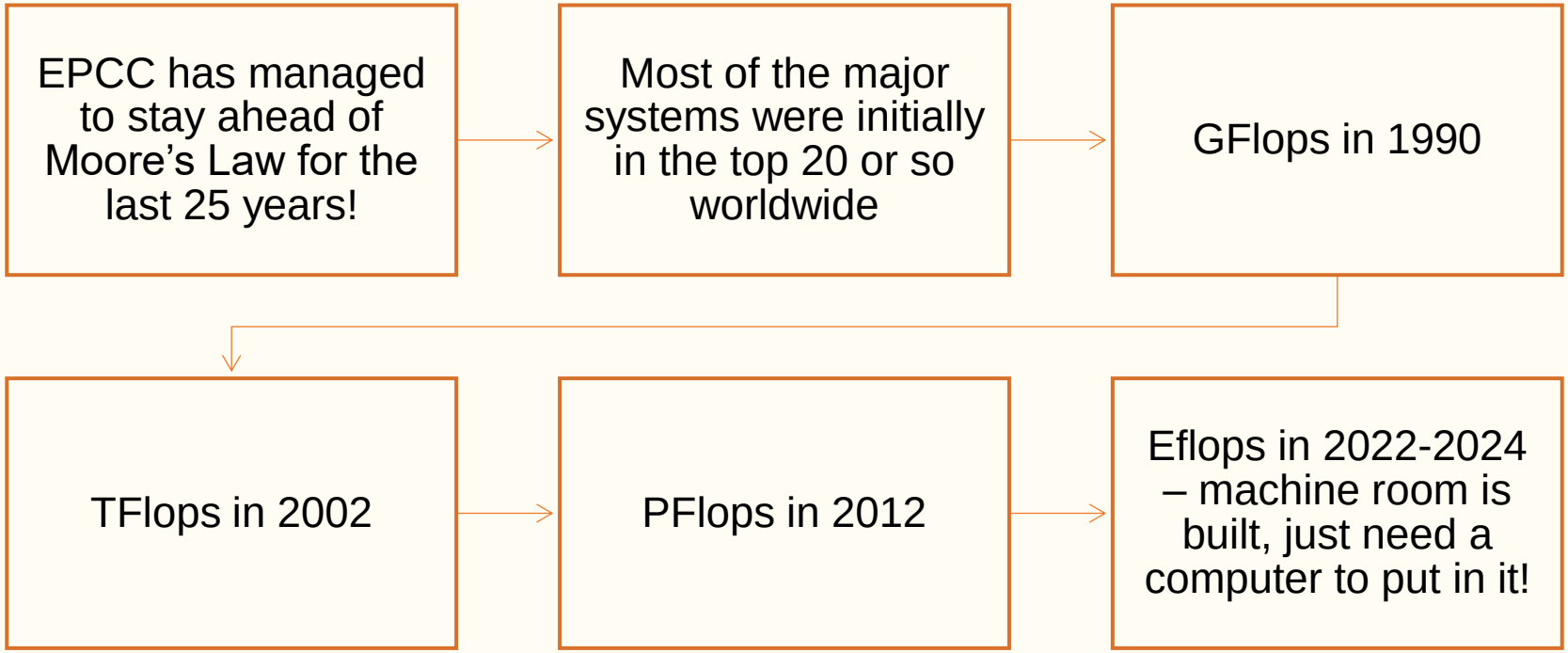
Each generation providing

- significantly greater compute power and larger memories
- better tools and more stable programming environment
- … but number of processors has increased only modestly until recently

Useful lifetime of machines around 5 years

(Nearly) missed out on vector architectures

EPCC Performance



Summary

Parallel computing started as a range of weird and wacky architectures

Each with their own unique languages, OS, tools, etc.

each required substantial investment of time to port to

Standardisation of programming paradigms was vital for parallel computing to mature

MPI, HPF, OpenMP,…

Edinburgh/EPCC has been at the forefront of parallel computing and HPC for 30 years

The future appears extremely exciting for HPC, computational science and EPCC