703650 VO Parallel Systems WS2019/2020
Motivation & A Crash Course in Parallel Hard- and Software

Philipp Gschwandtner

Overview

- what is parallelism, why do we need it?
  - applications and problems
  - “three walls”
- parallelism in hardware
  - HT, multi-/many-core, multi-CPU, clusters, NUMA, …
- parallelism in software
  - task & data parallelism, Flynn taxonomy, shared & distributed memory

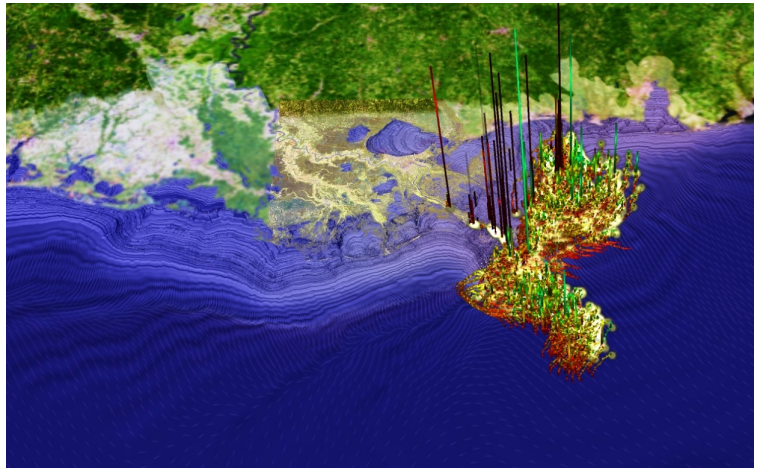
What is Parallel Computing?

- using multiple, **simultaneous** computations in order to **speed up** solving the overall problem
  - note the difference to concurrent computing ("simultaneous" vs. e.g. Pthreads on a single core)
  - generally (but not exclusively) the goal is faster computation of the result

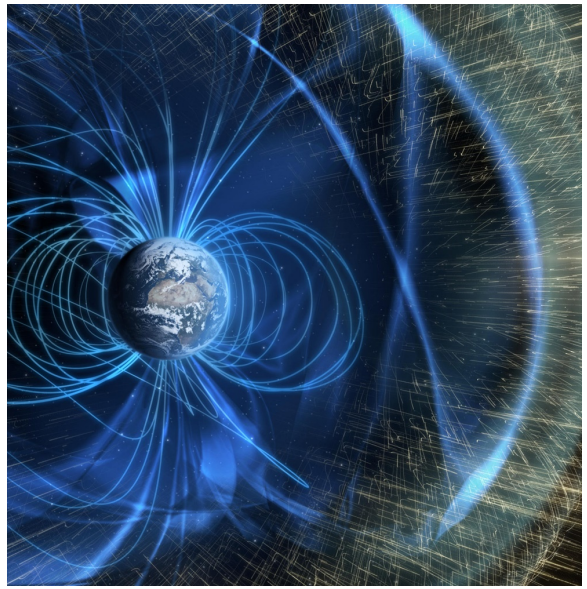
- requires multiple processing elements that can be used simultaneously ("parallel hardware")
  - lots of shapes and forms, more details in a bit

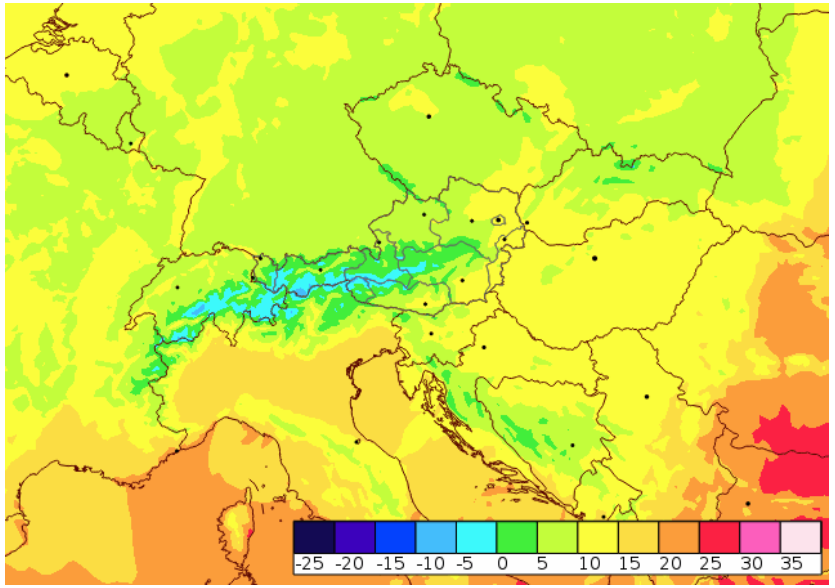
Why do we Need Parallelism?

- Deepwater Horizon oil spill in Gulf of Mexico in 2010
  - research projects to simulate the propagation, carried out by IBM (and UIBK)
  - 3 D smoothed particle hydrodynamics (SPH) simulation using Navier-Stokes equations
- faster-than-realtime simulation required
  - oil takes 20 hours from source to ocean surface
  - computing 1 second requires 3 hours of simulation
  - 30 years of computing time to simulate entire spill



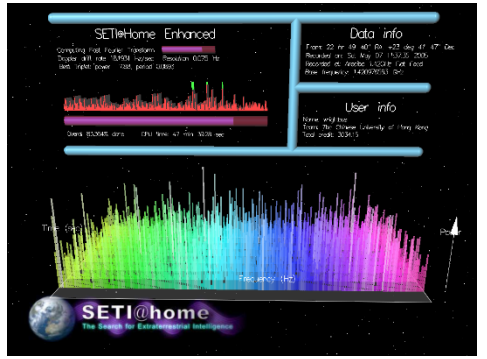
Additional Applications





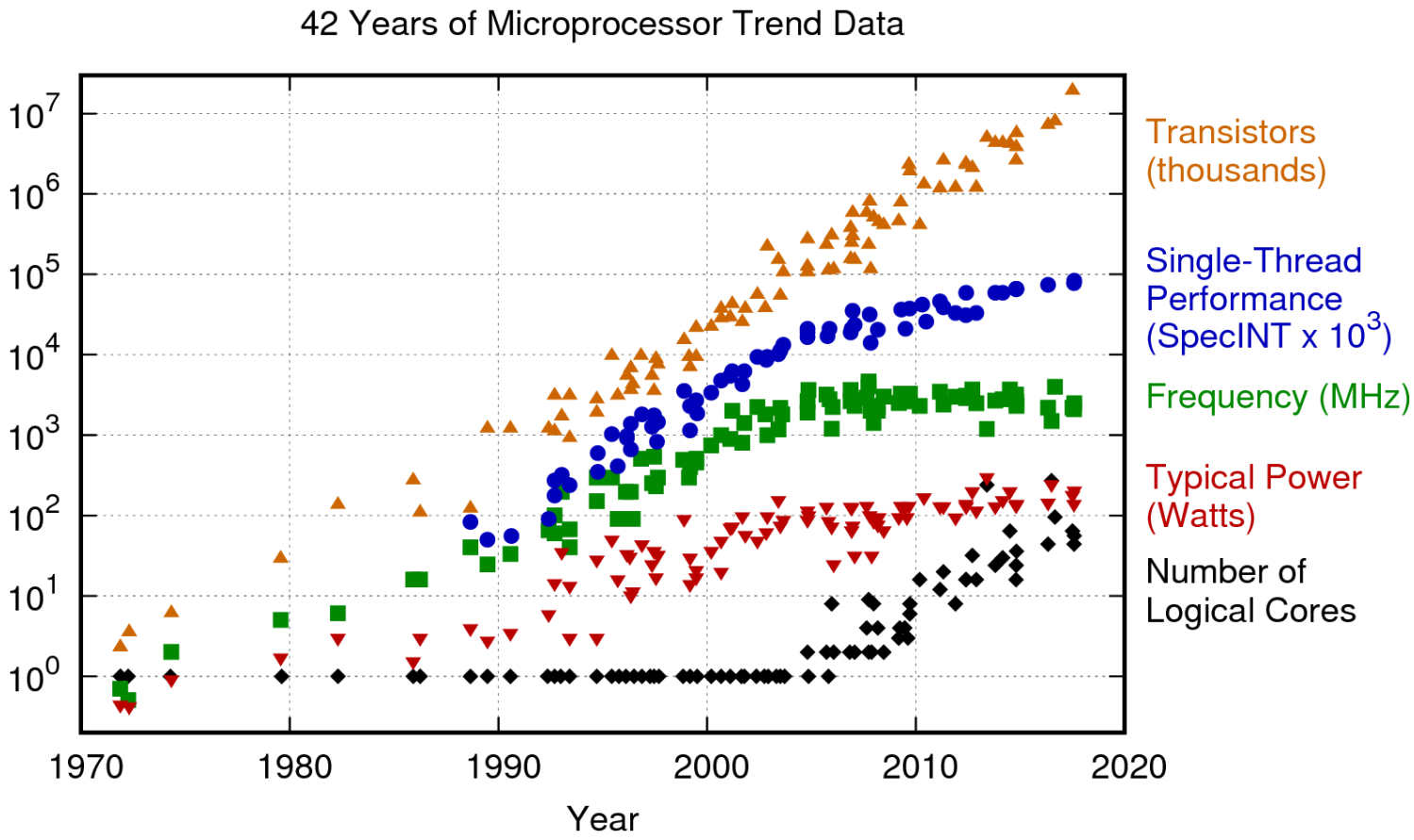
SETI@home BOINC Project

- “Arecibo” radio telescope collected 350 GB of data per day in the 1990s
- too much data to analyze for a single data center at the time
- divided into 350 KB chunks, sent to participating end-users for analysis



Parallelism in Hardware

Why do we Need Parallelism? The Hardware Side

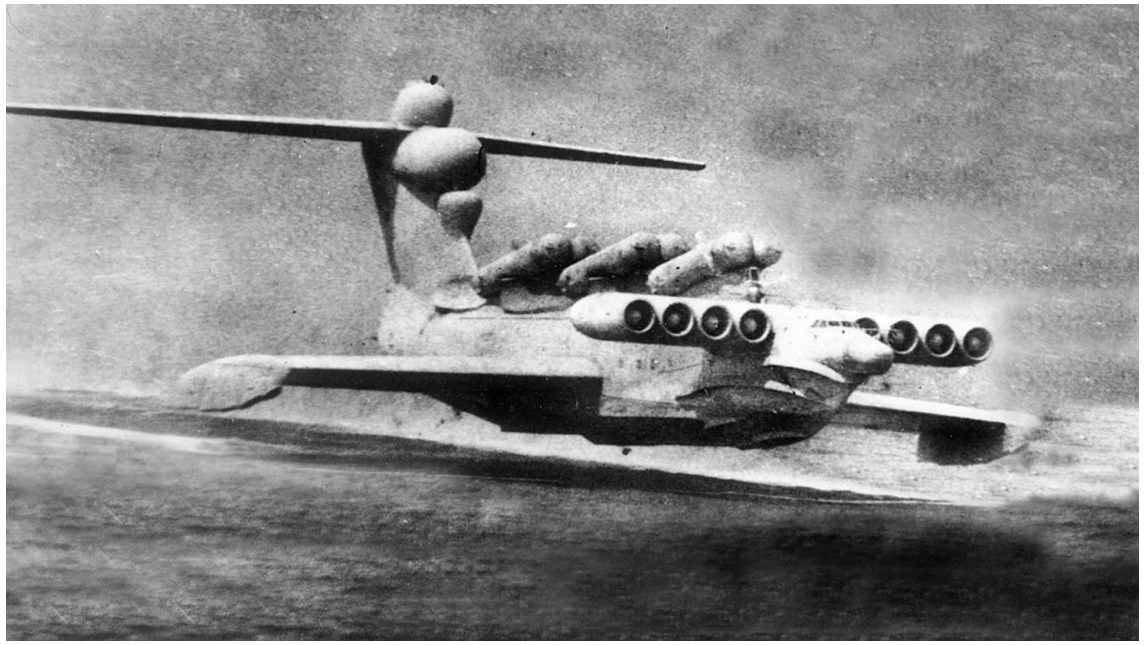


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

“Add Engines Until Airborne” no Longer Viable

- soviet MD-160
  (a “Lun” class ekranoplan)

- just throwing more resources at the problem doesn’t get you very far, and it gets increasingly expensive



Moore's Law

- "the number of transistors in dense integrated circuits doubles approx. every 2 years" – 1965
- also rephrased as "performance doubles every 18 months"
- has been derived from historical data, used as predictive measure for decades
- will likely cease to be true in the 2020s
- compare Intel's architecture updates
  - previously: "Tick-Tock" (2 steps)
  - now: "Process-architecture-optimization" (3 steps)

Parallel Computing is Ubiquitous

- increasing amount of parallel hardware
  - Supercomputers since ~1960s
  - Desktop PCs, laptops since 2005
  - Mobile and embedded devices
    - cellphones since 2011
    - smart watches since 2016
  - …
- sequential computing hardware practically became extinct!

- increasing amount (>120) of parallel or concurrent programming languages and libraries
  - Ada, Akka.NET, Alef, Alice, Apache Beam, Apache Flink, Apache Hadoop, Apache Spark, Ateji PX, Axum, Bloom, BMDFM, C#, C*, C++, C++ AMP, C=, CAL, Chapel, Charm++, Cilk, Cilk Plus, Clojure, CnC, Coarray Fortran, Concurrent Clean, Concurrent Haskell, Concurrent ML, Concurrent Pascal, Constraint Handling Rules, Cpp, Crystal, CUDA, Curry, Cω, D, Dart, E, Ease, ECMAScript, Eiffel, Eiffel SCOOP, Elixir, Elm, Emerald, Erlang, Esterel, FAUST, Fork, FortranM, Fortress, Futhark, Glenda, Go, Golang, Haskell, Hermes, HPF, Hume, Id, Io, Janus, Java, JavaScript, JCSP, JoCaml, Join Java, Joule, Joyce, Julia, LabVIEW, Limbo, Linda, Lustre, Mercury, Millipede, Modula, MPD, **MPI**, MultiLisp, Newsqueak, Occam, Occam-π, OpenCL, OpenHMPP, OpenMP, Orc, Oz, ParaSail, Parlog, Perl, Pict, Pony, Preesm, Prolog, PyCSP, Python, Red, Reia, Ruby, Rust, SALSA, Scala, SequenceL, Sequoia, Signal, SISAL, Smalltalk, SR, StratifiedJS, SuperPascal, SystemC, SystemVerilog, Termite Scheme, Titanium, TNSDL, Unicon, UPC, Verilog, VHDL, X10, XC, ZPL, µC++ (Source: en.wikipedia.org)
- serve as the basis to many more, highly domain-specific languages and libraries (DSLs)

The Three Walls

- power wall
  - increase in transistors means increase in dynamic power consumption
- memory wall
  - growing speed disparity between computational units and memory
- instruction-level parallelism (ILP) wall
  - diminishing returns in overlapping in-core instruction execution

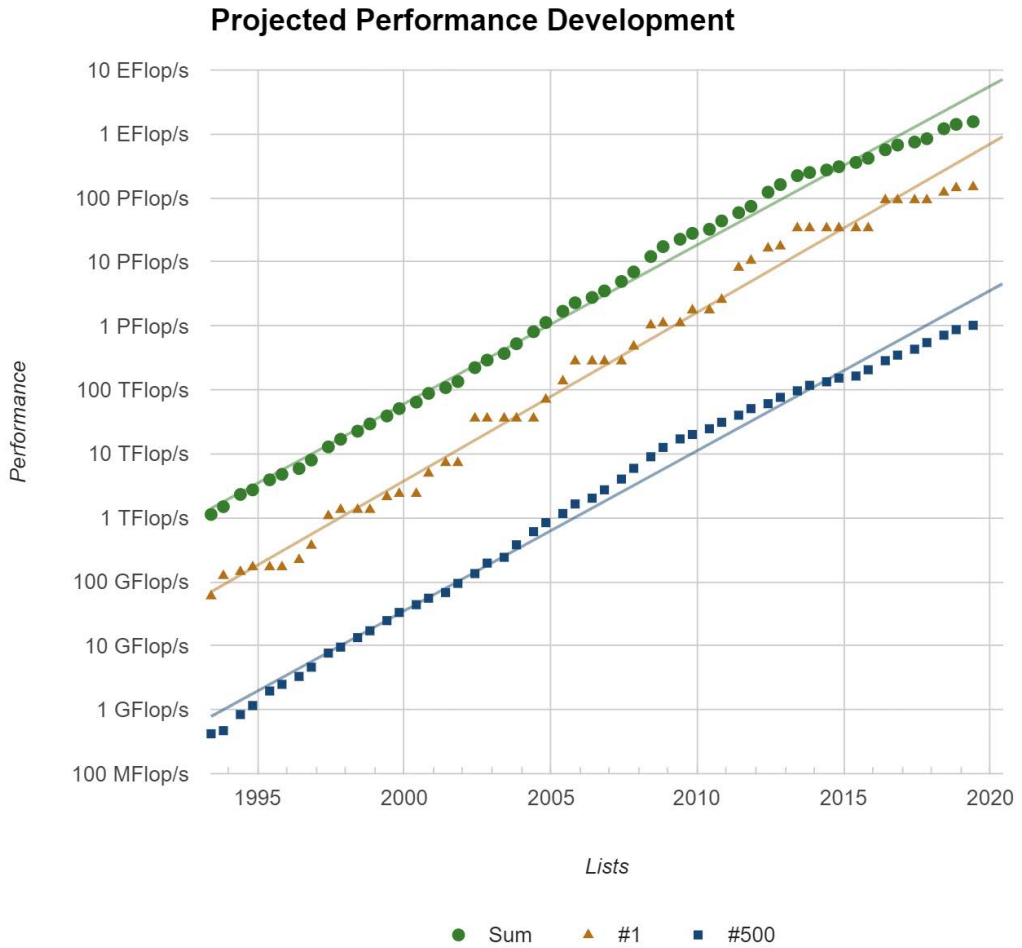
The Power Wall

- frequency increase was not an option anymore
  - $P_{dyn} = C \times \boldsymbol{F} \times \boldsymbol{V^2} \times \alpha \approx \boldsymbol{F^3}$
    (C… capacitance, F… frequency, V… voltage, α… switching factor)

- number of transistors can be increased
  - feature-size reduction allows same area
  - but power consumption and heat dissipation increase
  - not everything run at the same time (“dark silicon”) or only at lower F and V
  - Dennard Scaling Law (“same area – same power consumption”) no longer holds
    - started breaking down around 2005

The Power Wall cont'd

- TOP500 list (https://www.top500.org)
  - fastest supercomputers world-wide
  - released twice every year
  - very interesting analyses and statistics around supercomputing and HPC
  - clearly shows Dennard Scaling impact (delayed effect around ~2015)



The Power Wall cont'd

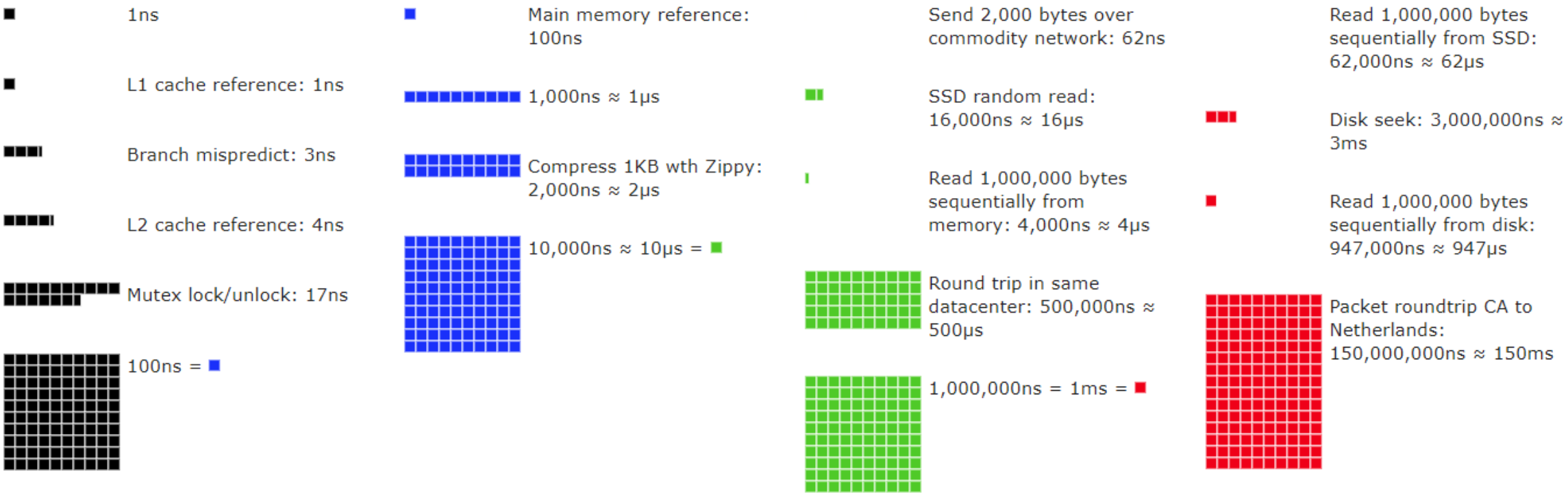
VSC-3 compute node cooling

https://www.youtube.com/watch?v=NxNUK3U73SI

The Memory Wall

- computational speed grows faster than memory speed (latency & bandwidth)
  - executing a single integer ADD: 1 clock cycle (approx. 0.3 ns @ 3 GHz)
  - latency fetching data from RAM: 300 clock cycles (100 ns @ 3 GHz)
  - similar disparity for memory bandwidth and computing speed over the past decades
- lead to the introduction of memory hierarchies (L1, L2, L3, L4 caches) or High Bandwidth Memory (HBM)
- changes the way (parallel) programs are written and optimized
  - focus on latency of load/store instead of computation instructions
  - aim for high data locality

Side Note: Latency Numbers Every Programmer Should Know



The ILP Wall

- instruction-level parallelism is omnipresent (responsibility of compiler and processor designers)
  - pipelining
  - superscalar execution
  - out-of-order execution
  - register renaming
  - branch prediction
  - prefetching
  - …
- however: diminishing returns for ILP optimizations
  - gets increasingly difficult to maintain high utilization of a single, modern processor core

Three-Wall-Summary

- power, memory, and ILP wall represent previous optimization spaces for which the limit has been (almost) reached
  - entirely new approach required to further increase performance
- solution: Introduce high-level parallelism
  - put multiple cores on a CPU or multiple CPUs on a mainboard
  - makes software developers very sad ☹

The New York Times

TECHNOLOGY

TECHNOLOGY; Intel's Big Shift After Hitting Technical Wall

By John Markoff

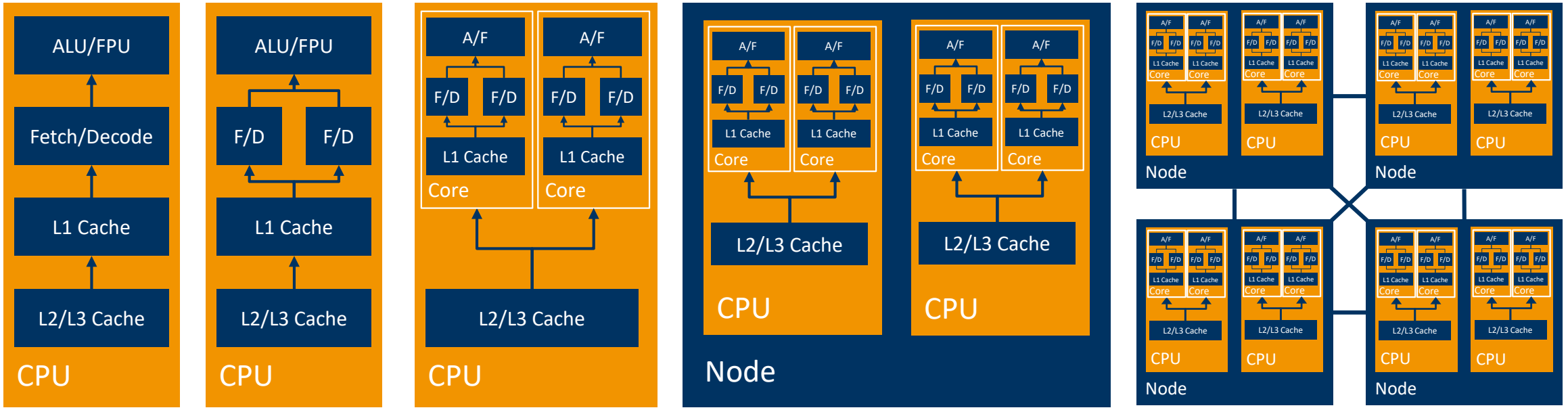
May 17, 2004

Correction Appended

The warning came first from a group of hobbyists that tests the speeds of computer chips. This year, the group discovered that the Intel Corporation's newest microprocessor was running slower and hotter than its predecessor.

What they had stumbled upon was a major threat to Intel's longstanding approach to dominating the semiconductor industry -- relentlessly raising the clock speed of its chips.

Forms of Parallel Hardware (Fictional Architecture)



single-core CPU

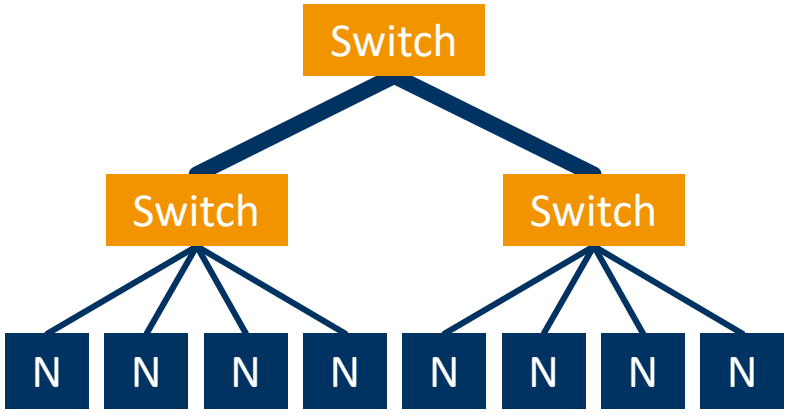
single-core CPU + HT

multi-core CPU + HT

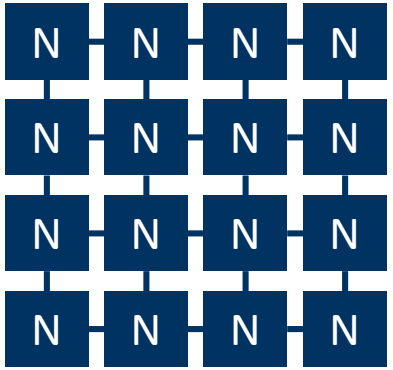
shared memory node of multi-core CPUs + HT

cluster of shared memory nodes of multi-core CPUs + HT

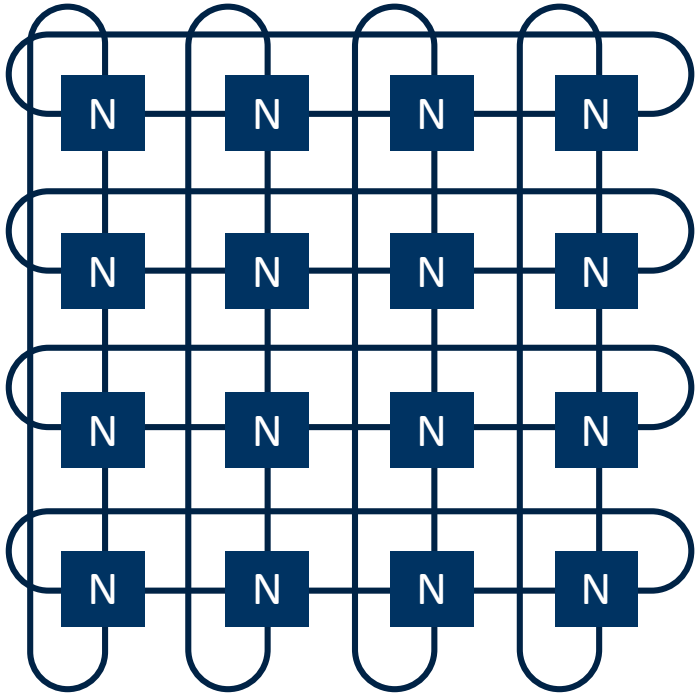
Forms of Parallel Hardware cont'd



(Fat) Tree Topology



2D Mesh



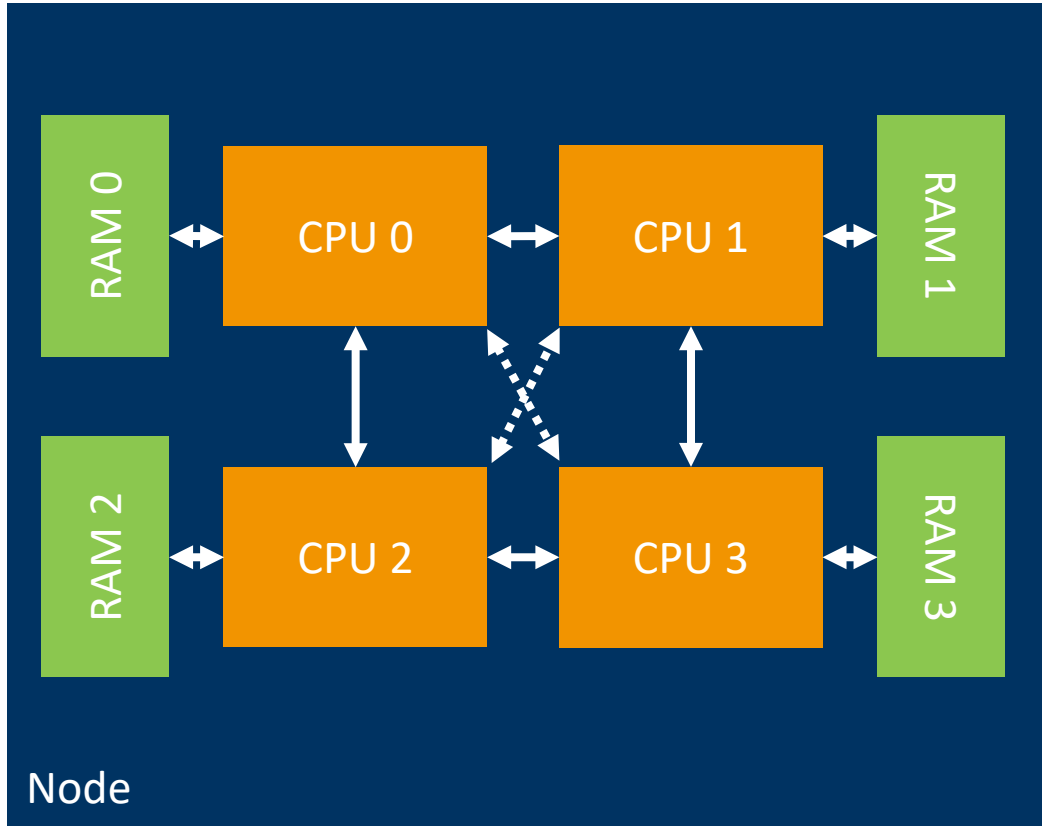
2D Torus

Forms of Parallel Hardware cont'd

- "multi-core" or "many-core" CPU?
  - do you need to seriously think about core interconnects ➔ „many-core"
  - ring bus vs. meshes vs. …
- vectorization
  - in-core parallelization support (e.g. MMX, SSE, AVX)
- NUMA!

Non Uniform Memory Access (NUMA)

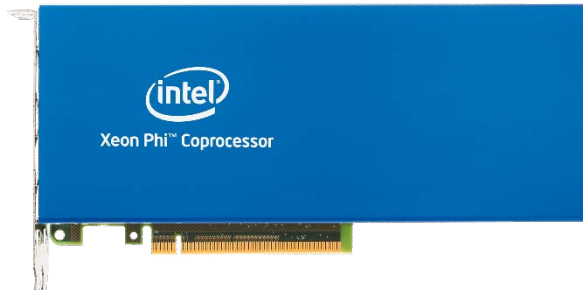
- any RAM accessible from any CPU
  - not at the same cost though!
- usually ccNUMA (cache-coherent)
  - easy to program, but concealed performance bottleneck
- there are tools that can textually/graphically show the NUMA topology (e.g. hwloc's lstopo)



4-socket Intel Skylake configuration
with 2/3 UPI links per CPU

Accelerators are Parallel too!

- GPUs
  - NVidia: Quadro V100, RTX2080, Titan, Tegra/Xavier
  - AMD: RX 5700, Radeon Instict
- FPGAs
  - Xilinx: Virtex
  - Altera/Intel: Stratix
- Misc
  - Intel Xeon Phi
- Several caveats
  - different ISA (and RISC vs. CISC)
  - different address space
  - most require OpenCL, CUDA, VHDL or the like
  - we are not going to work on any of those!



HPC @ UIBK

- LCC2 (2009, 64 cores)
  - 8 nodes
    - 2x Intel Quad-Core L5420 CPUs
    - 32 GB DDR2 RAM
  - Infiniband DDR network
    - 16 Gbit/s bandwidth
    - 2.5 µs latency
  - used for teaching and training only
    - mostly empty
- LEO4 (2018, 1344 cores)
  - 48 nodes
    - 2x Intel Xeon E5-2690 v4
    - 64 GB DDR4 RAM
  - Infiniband EDR network
    - 100 Gbit/s bandwidth
    - 0.5 µs latency
  - used for scientific research
    - along with LEO3 and LEO3e

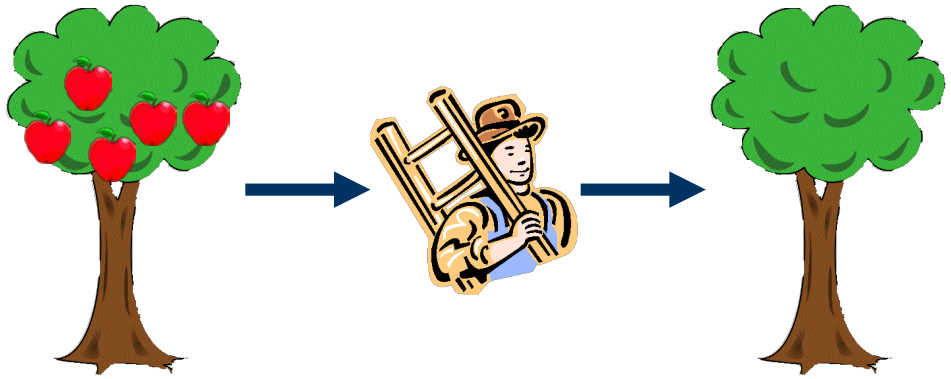
HPC Beyond Innsbruck

- VSC4 in Vienna (18960 cores)
  - 790 nodes
    - 2x 12-Core Intel Xeon 8174
    - 96 GB RAM
  - Intel Omnipath
    - 100 Gbit/s bandwidth
    - 0.1 µs latency
  - ranks 82th world-wide
- Summit in Oak Ridge, TN, USA (>2 million cores)
  - 4608 nodes
    - 2x 22-Core IBM POWER9 + 6x NVidia Volta GV100
    - 512 GB RAM
  - Infiniband EDR network
    - 2x 100 Gbit/s bandwidth
    - 0.5 µs latency
  - ranks 1st world-wide

Parallelism in Software

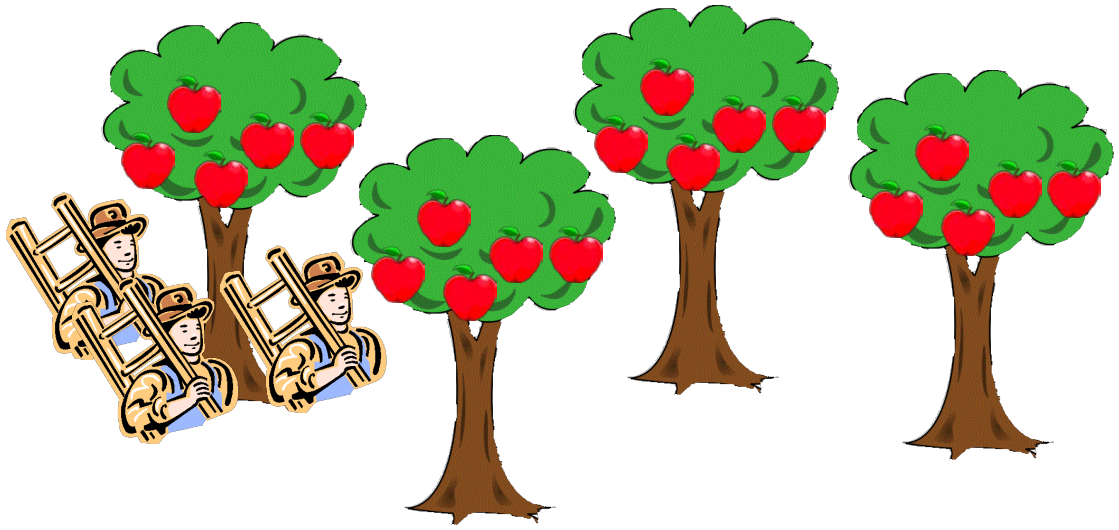
High-Level Types of Parallelism

- data Parallelism
  - execute parts of the same task on different data simultaneously
- task Parallelism
  - execute different tasks within the same problem simultaneously
- consider the work (=task) of having workers (=processing units) pick apples (=data) from a tree



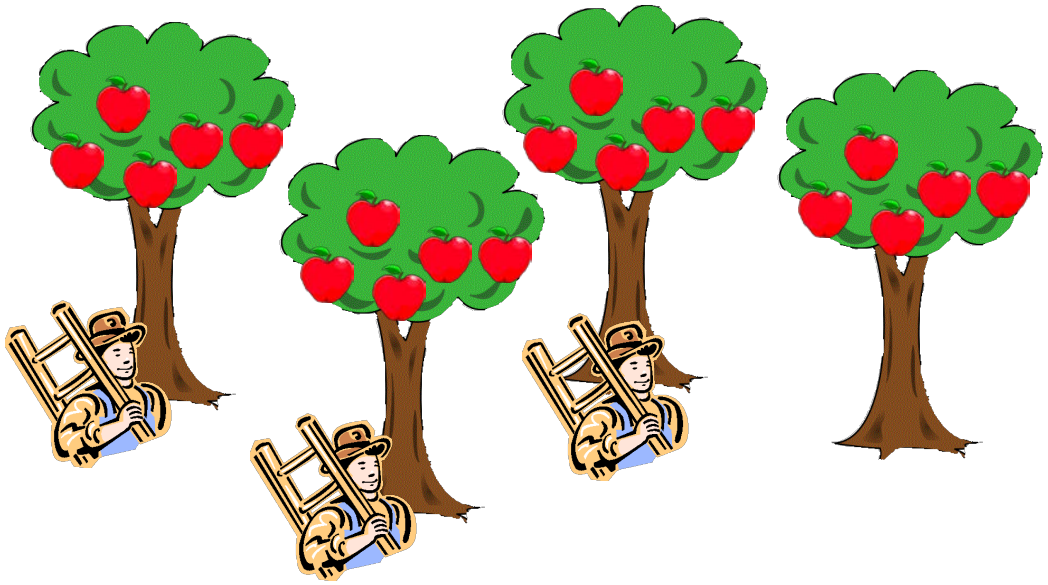
Data Parallelism

- have multiple workers pick multiple apples from the same tree at the same time
  - How many workers per tree? How many apples per worker?
  - What if not all the trees have the same amount of apples?
  - What if not all the workers pick the same amount of apples?



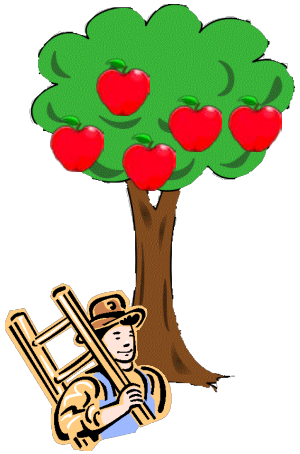
Data Parallelism cont'd

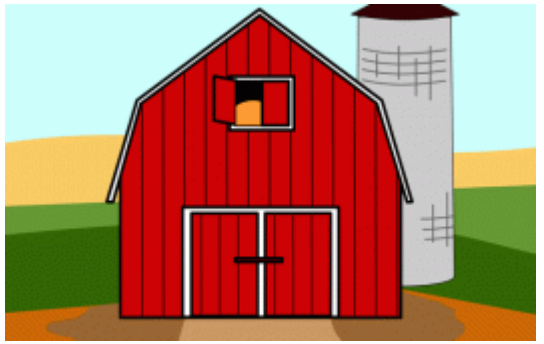
- have multiple workers pick apples from different trees at the same time
  - How many workers?
  - What if not all the trees have the same amount of apples?
  - Nested parallelism?



Task Parallelism

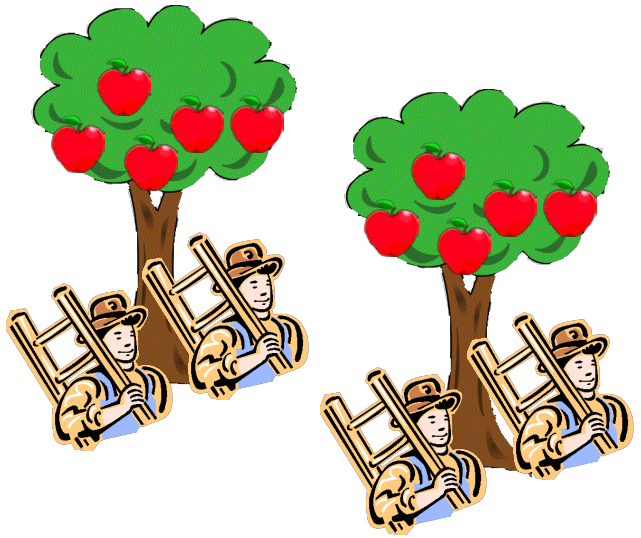
- apples also need to be transported to the farm!
  - How many apples to pick before they are transported to the farm?
  - How fast is each worker at their task?

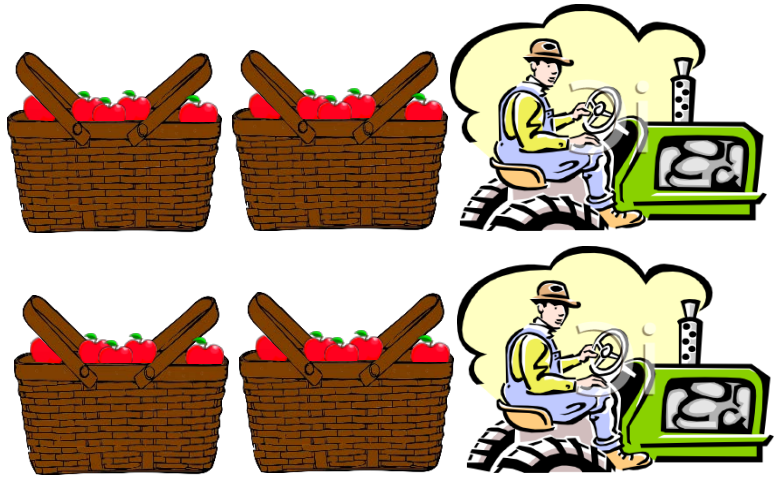


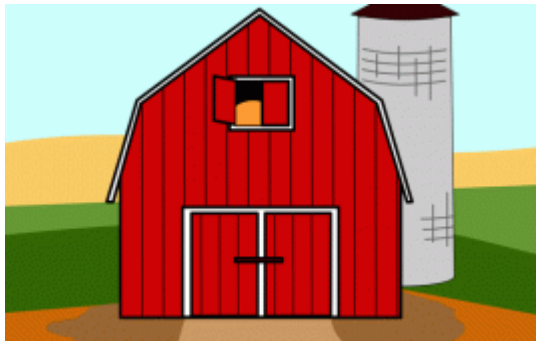


Hybrid Parallelism

- have multiple workers pick apples from multiple trees, to be transported to the farm by multiple workers
  - consider all the complexities yourself







Flynn's Taxonomy

- classification of computer architectures, proposed by Michael Flynn in 1966
- still valid and in use today

Single Instruction Single Data (SISD)	Single Instruction Multiple Data (SIMD)
Multiple Instruction Single Data (MISD)	Multiple Instruction Multiple Data (MIMD)

Flynn's Taxonomy cont'd

- SISD
  - single instruction per time unit
  - single data unit per time unit
  - e.g. basic single-core CPUs
- SIMD
  - single instruction per time unit
  - multiple data units per time unit
    - but all with the same operation at the same time, i.e. in full lockstep
  - e.g. vector units in CPUs (Intel MMX, SSE, AVX; IBM AltiVec; ARM NEON, …), GPUs

Flynn's Taxonomy cont'd

- MISD
  - multiple instructions per time unit
  - single data unit per time unit
  - very rare, mostly used for fault tolerance or redundancy
    - e.g. flight computers for airplanes, rockets or the space shuttle
- MIMD
  - multiple instructions per time unit
  - multiple data units per time unit
  - very large class, includes every multi-core CPU or multi-thread/multi-process software
  - sometimes subdivided into
    - SPMD – single program (like SIMD, but no lockstep)
    - MPMD – multiple programs

MPI Generally Used for Data Parallelism / SPMD

- decompose data into multiple chunks, have multiple processing elements do the same work on their own chunks
- interested in task parallelism?
  - Check out e.g. OpenMP, OpenCL, Cilk, Pthreads, Intel TBB, …
- interested in hybrid parallelism?
  - Can be done with e.g. OpenMP or OpenCL
  - Also: Consider combining the above with MPI (often called MPI+X)

Shared Memory and Distributed Memory Parallelism

- shared memory
  - single memory address space
  - usually based on threads
  - all data can be accessed directly
  - synchronization (e.g. barriers) required to ensure correctness

```
x[0] += 42;
```

- distributed memory
  - multiple memory address spaces
  - usually based on processes
  - data cannot be accessed directly
  - message exchange required to get data and ensure synchronization

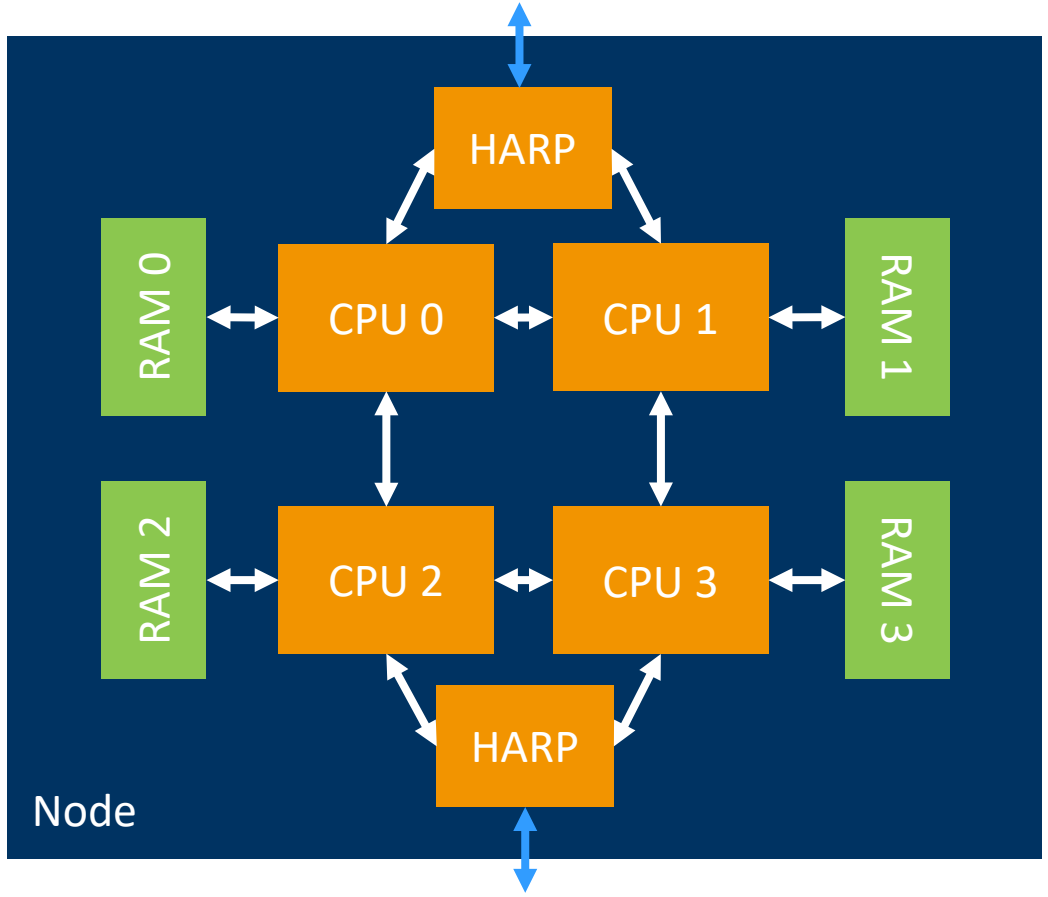
```
x = recv_data(…);
x[0] += 42;
send_data(x, …);
```

Shared/Distributed Memory Implications

- shared memory
  - direct data access hides access cost (interconnect and memory latency)
  - very little explicit communication
    - frequently leads to race conditions
  - can often be done incrementally from sequential code
  - available memory scales with RAM of a single node (limiting factor – except for exotic hardware types such as SGI UV)
- distributed memory
  - data access cost (interconnect and memory) evident in the code
  - a lot of explicit communication
    - frequently leads to deadlocks
  - difficult to do incrementally, often all-or-nothing approach
  - available memory scales with machine size (number of nodes)

Exotic HPC Hardware: MACH-2 @ JKU in Linz

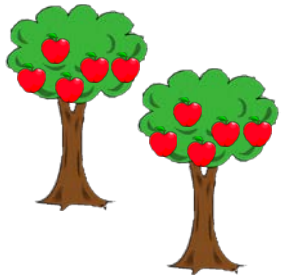
- 1728 cores, 20 TB of RAM
- same components as all other distributed memory systems
- uses SGI NUMAlink to provide global cache coherence
- useful for shared-memory apps that require a lot of RAM



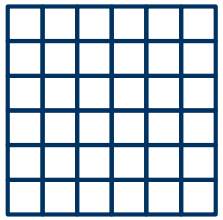
SGI NUMAlink architecture

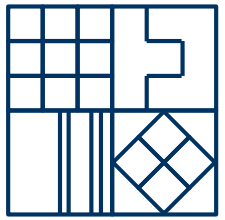
Three Steps to Creating a Parallel Program

- **1:** identify work that can be performed in parallel and/or data that can be worked on in parallel

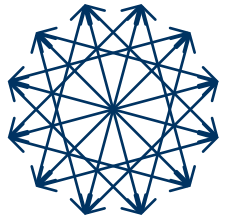


- **2:** partition work and/or data





- **3:** manage data accesses, communication, and synchronization



- most importantly: do all of that BEFORE touching the keyboard!

Summary

- Moore’s law & the three walls
  - parallelism was the only feasible way out
- parallelism in hardware
  - HT, multi-/many-core, multi-CPU, clusters, network topologies
  - NUMA
- parallelism in software
  - data & task parallelism
  - Flynn Taxonomy
  - shared vs. distributed memory

Image Sources

- Parallel Applications: https://www.chemistryworld.com/features/oil-spill-cleanup/3008990.article, Marcel Ritter (UIBK), https://twitter.com/maven2mars/status/984440044659159040, https://www.nasa.gov/ames/image-feature/nasa-highlights-simulations-at-supercomputing-conference-like-aircraft-landing-gear, ZAMG Wettervorhersage 03.10.2019 12:00
- Soviet MD-160: https://gizmodo.com/this-caspian-sea-monster-was-a-giant-soviet-spruce-go-1456423681
- TOP500 Trend: https://www.top500.org/statistics/perfdevel/
- THG Video: https://www.youtube.com/watch?v=NxNUK3U73SI
- Latency Numbers: https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html
- Accelerators: https://www.anandtech.com/show/12579/big-volta-comes-to-quadro-nvidia-announces-quadro-gv100, https://forums.xilinx.com/t5/Xcell-Daily-Blog-Archived/Want-to-get-on-board-the-Xilinx-UltraScale-FPGA-express-now/ba-p/727882, http://www.itmi.co.kr/product/product_list.php?ca_id=1020