Chances and Challenges in Developing Future Parallel Applications

Prof. Dr. Rudolf Berrendorf
rudolf.berrendorf@h-brs.de
http://berrendorf.inf.h-brs.de/

Bonn-Rhein-Sieg University, Germany
Computer Science Department

Bonn-Rhein-Sieg
University

Outline

- Why Parallelism?
- Parallel Systems are Complex
- Programming Parallel Systems
- Chances and Challenges
- Summary

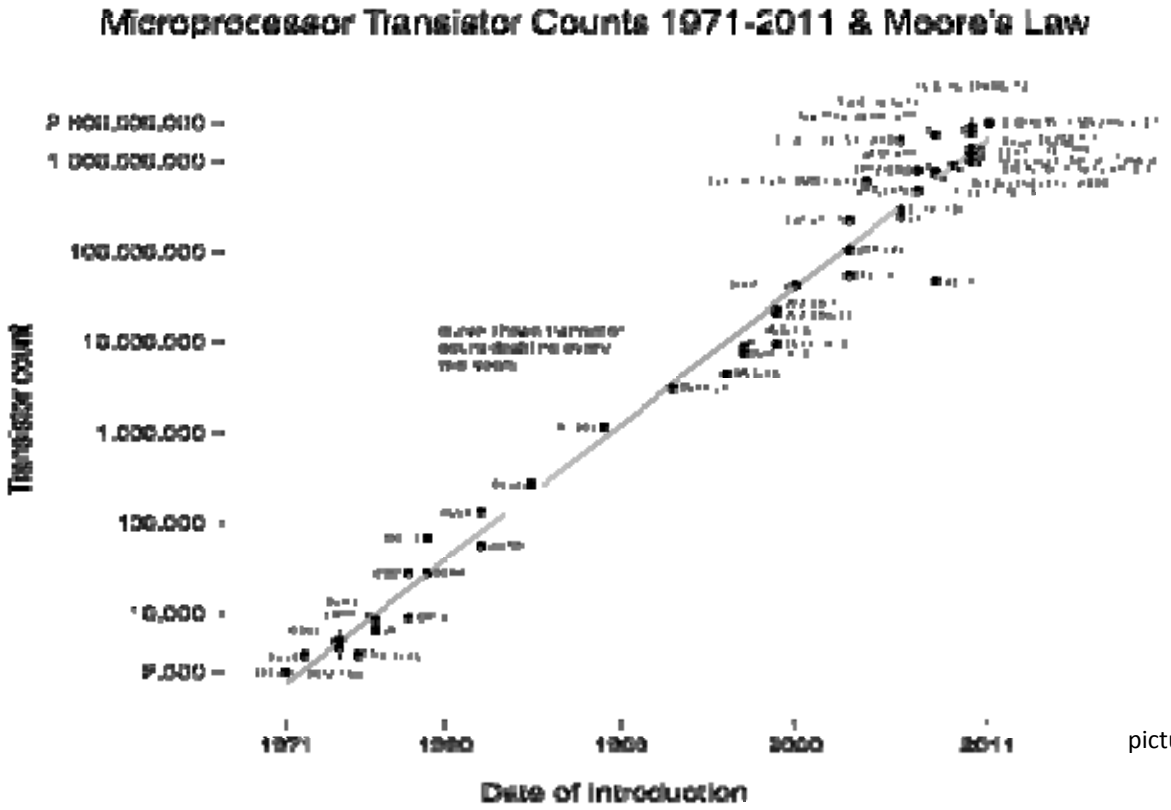
Outline

- Performance Limitations / Why Parallelism?
  - **Fundamental Limitations (relevant to all developers)**
  - Computational Demands (relevant to some developers)

- Parallel Systems are Complex
- Programming Parallel Systems
- Chances and Challenges
- Summary

Limits on Miniaturization

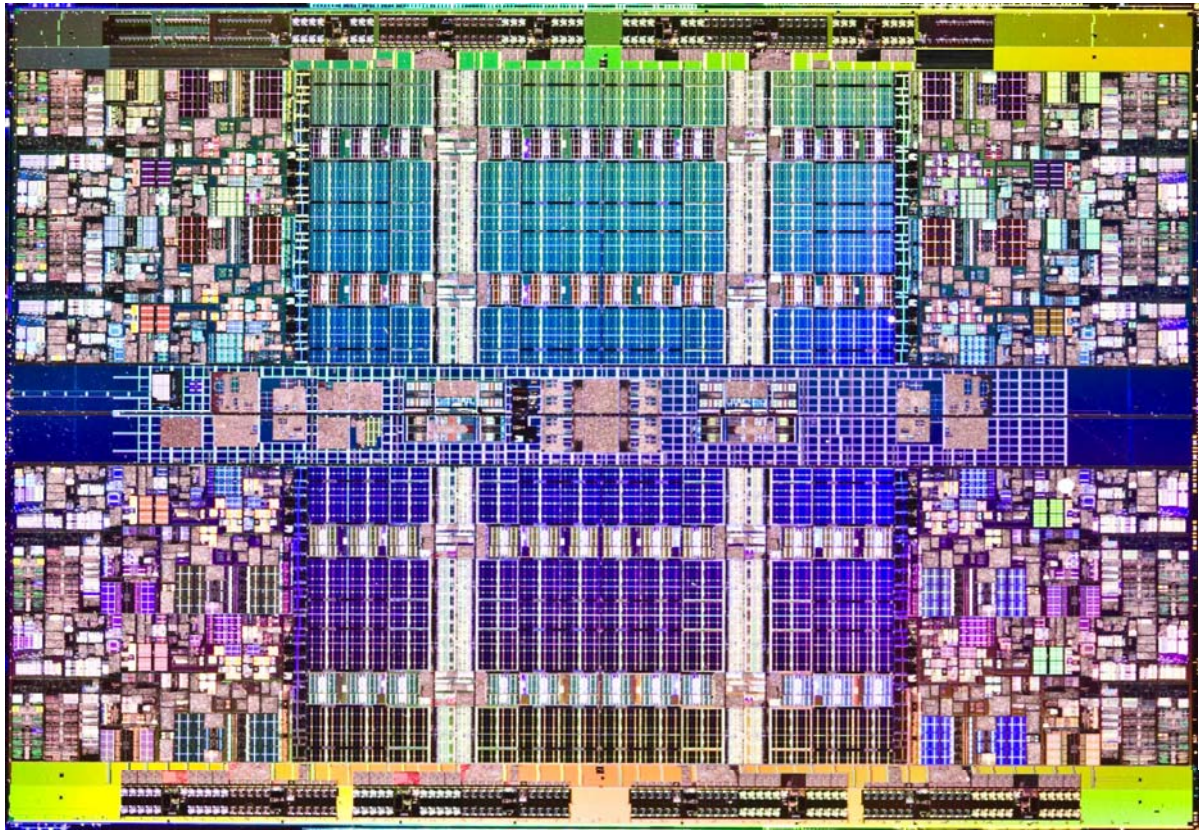
- Moore's Law: shrink structure size / double transistor count
- Currently, Intel uses 22 nm SI-based CMOS technology for processors
- Intel's published plan for the next years: 14 nm, 10 nm, 7 nm, 5 nm
- The size of 1 silicon atom is 0.27 nm



picture source: Wikipedia

Speed

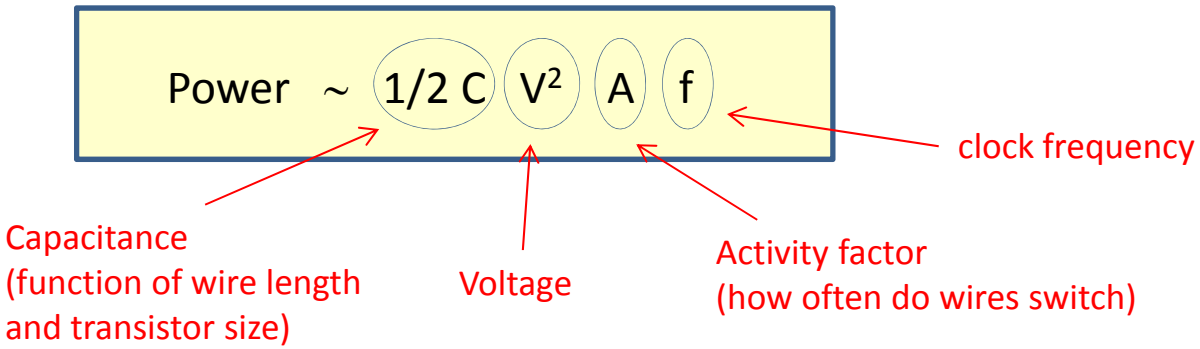
- Data (signals) has to be transfered inside a processor over some distance
- Register, functional unit, processing inside a unit, caches, …
- The speed of light is a fundamental limit for the speed of any such operation inside a processor



Intel Xeon
picture source: Intel

Power Dissipation

- Dynamic CMOS power dissipation:



- Todays mainstream server processors use up to 150 W per processor
- A high end grafic processor uses up to 250 W
- One 42" rack full of servers can use more than 40 kW
- There are large high performance systems that use more than 10 MW

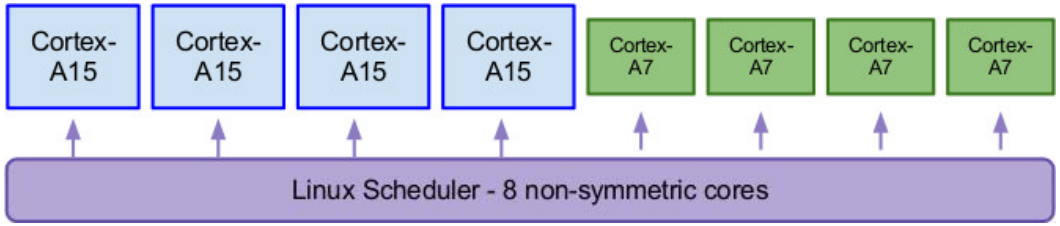
- Power dissipation / cooling is today the main limitating factor
  - for high performance computing
  - for mobile devices

Parallel Processing in Mobile Devices

- How to handle limited power / cooling capacity in mobile devices:
  - Idea: Instead of using one core running with frequency f, use p cores with frequency f/p (ideally)
  - Instead of using fast cores for everything, use additional slower cores for non-critical tasks where these cores are designed for very low power consumption

- big.LITTLE processing of ARM
  - big: high-performance CPU's for short running high-priority tasks
  - LITTLE: power-optimized CPU's for standard low priority tasks

- Implementation for example in Samsung's Exynos 5 Octa processor (used in Galaxy S5)
  - big: 2.1 GHz quad-core ARM Cortex-A15 (optimized for fast execution)
  - LITTLE: 1.5 GHz quad-core ARM Cortex-A7 (power optimized)



picture source: Wikipedia

Outline

- Performance Limitations / Why Parallelism?
  - Fundamental Limitations (relevant to all developers)
  - **Computational Demands (relevant to some developers)**

- Parallel Systems are Complex
- Programming Parallel Systems
- Chances and Challenges
- Summary

High Performance Computing

- There are applications with demands for very high computational capacity

- Own discipline: High Performance Computing (HPC)
  - optimized hardware
  - specialized software infrastructure (libraries / frameworks, language / compiler extensions, system software, batch systems, …)
  - methods for large scale problems
  - own community

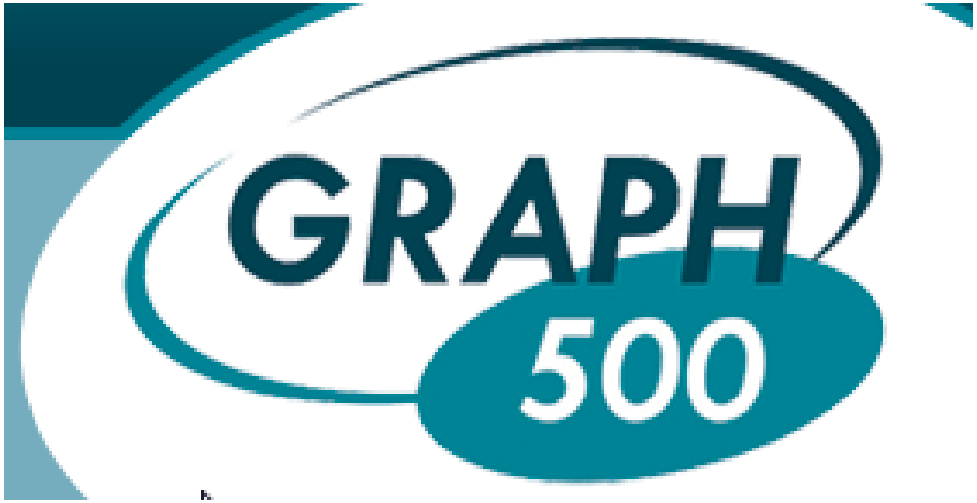
Top500

- List of top 500 most powerful computers in the world
- `http://www.top500.org/`
- List updated twice a year (june, november)
- Rating by **Flop/s** (floating point operations per second) with HPC Linpack benchmark
- **dense linear algebra, in essence limited by floating point operation demands**

picture source: http://www.top500.org/

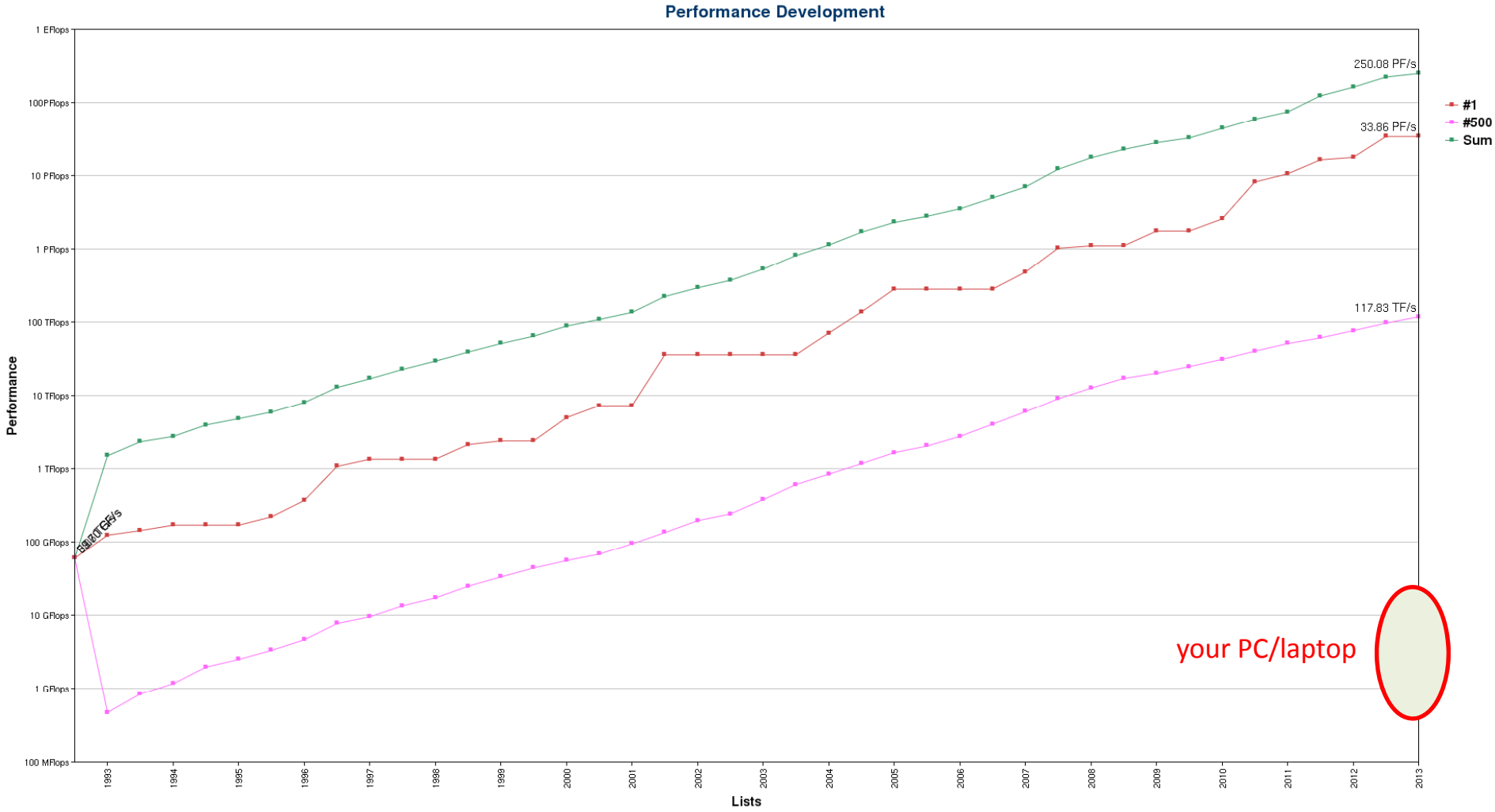
Graph500

- List of top 500 most powerful computers in the world
- `http://www.graph500.org/`
- List published twice a year (june, november)
- Rating by **MTEPS** (million traversed edges per second) with Breadth-First Search on large graphs
- **sparse data; in essence limited by memory bandwidth**



picture source: http://www.graph500.org/

Top500 Actual List



based on picture from http://wwwtop500.org/

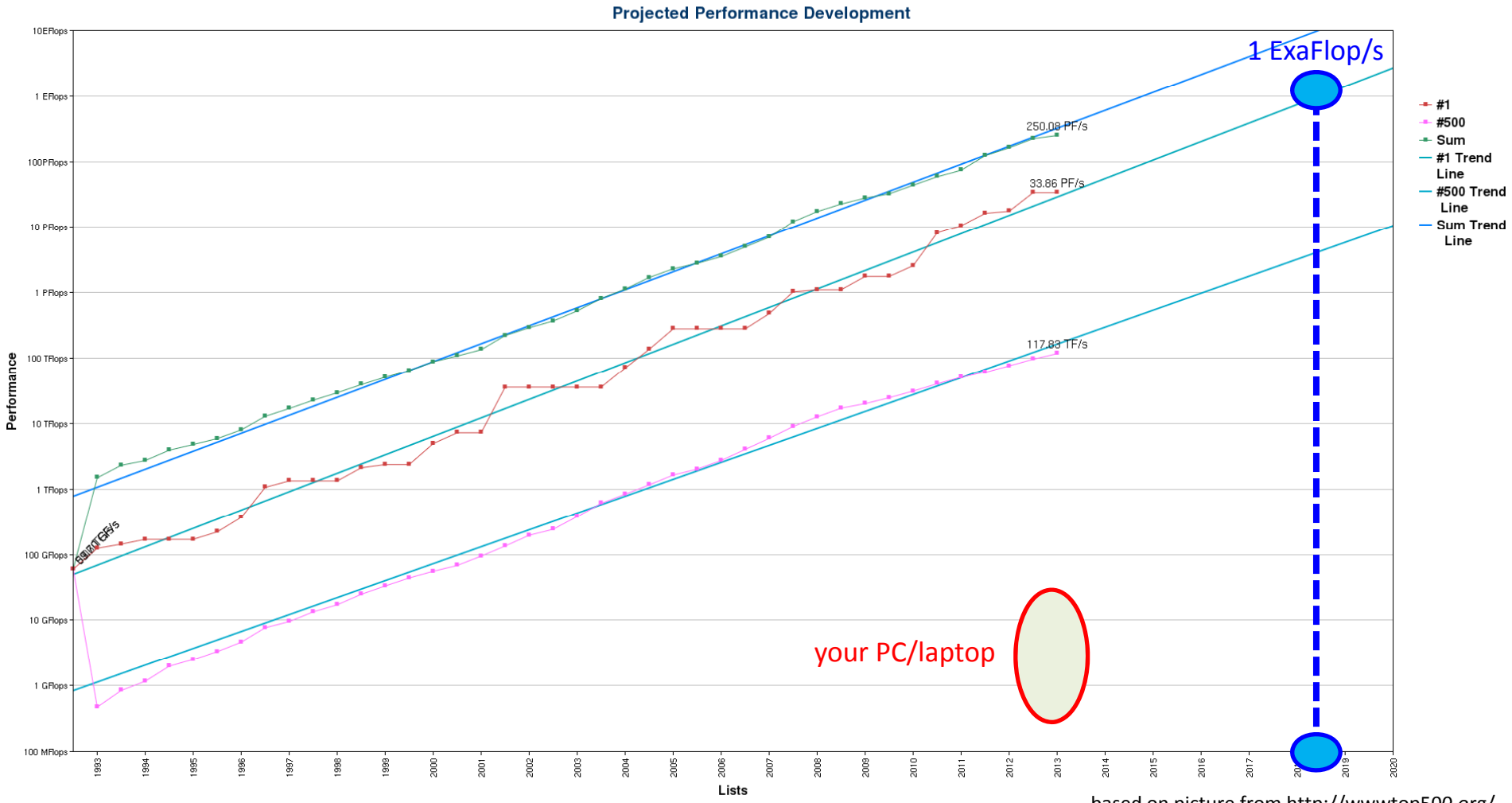
Current No 1

- Tianhe-2 (MilkyWay 2) at National Supercomputer Center in Guangzhou, China
- Cluster of 16.000 nodes, each with
  - 2x 12-core processors Intel Xeon E5-2692
  - 4x Intel Xeon Phi accelerator cards (51 cores each)
- In total 3.2 million cores

picture source http://wwwtop500.org/

Linpack performance	approx. 33.8 PFlop/s
peak performance	approx. 55 PFlop/s
power	approx. 18 MW
main memory	approx. 1.3 PiB
floor space	720 m^2

Top500 Performance Prediction



based on picture from http://wwwtop500.org/

How To Get To 1 ExaFlop/s

system attribute	2010	2015	2018	factor
system peak performance	2 PetaFlops	200 PetaFlops	1 ExaFlops	200x
power consumption	6 MW	15 MW	20 MW	~ 3x
system memory	300 TeraB	5 PetaB	32-64 PetaB	~ 200x
node performance	125 GigaFlops	0.5 or 7 GigaFlops	1 TeraFlops	8x
node memory bandwidth	25 GB/s	100 GB/s	400 GB/s	4x
node concurrency	12	O(100)	O(1000)	~ 100x
system size	18.700	50.000	O(100.000)	~ 10x
MTTI	days	O(1day)	O(1 day)	

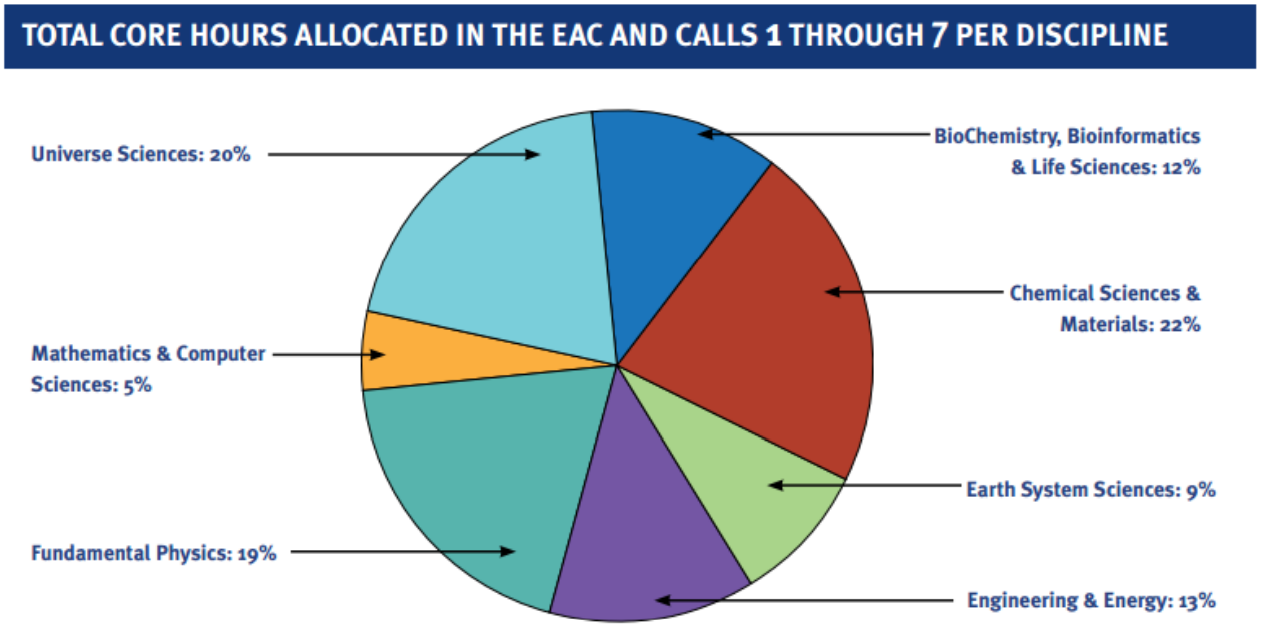
1000x

data source: DOE Exascale Initiative Technical Roadmap

European Initiative PRACE

- PRACE is a large scale european initiative for and around a european HPC infrastructure
- PRACE: Partnership for Advanced Computing in Europe
- `http://www.prace-project.eu/`

- Few Tier-0 centers with high-end systems



picture source: PRACE Annual Report 2013

Why Are These Computers Necessary?

- Do we need such large (and costly) computer systems?
- My Word, Outlook, Firefox, Adress Book,.. does not need such computational ressources…

Basic Research: Atomic-Scale Simulations

- Understanding the details, why DVD-RW, DVD-RAM, Blu-ray really work
- Simulating the behaviour of atoms when changing from chrystal lattice to amorphous material and back

- Simulation of 640 atoms for 300 femtoseconds (that is $3 \cdot 10^{-12}$ s) takes 4 months on 4.000 processors (or approx. 12 million CPU hours) at Juelich Research Centre
- On your laptop: more than 1.300 years simulation time

Scientific Reference: T.Matsunaga et.al.: From local structure to nanosecond recrystallization dynamics in AgInSbTe phase-change materials. Nature Materials 10, 129-134 (2011)

Industry: Geological Exploration

Many compute intensive tasks with geological exploration, for example:

- Detection of reservoirs
- Reservoir modelling

Example BP with their main production system installed in 2013:

- computing needs have increased by a factor of 10.000 within 12 years
- computer: >67.000 CPU's, 536 TB main memory, 23.5 PB disks

source: BP

Industry: Car Development

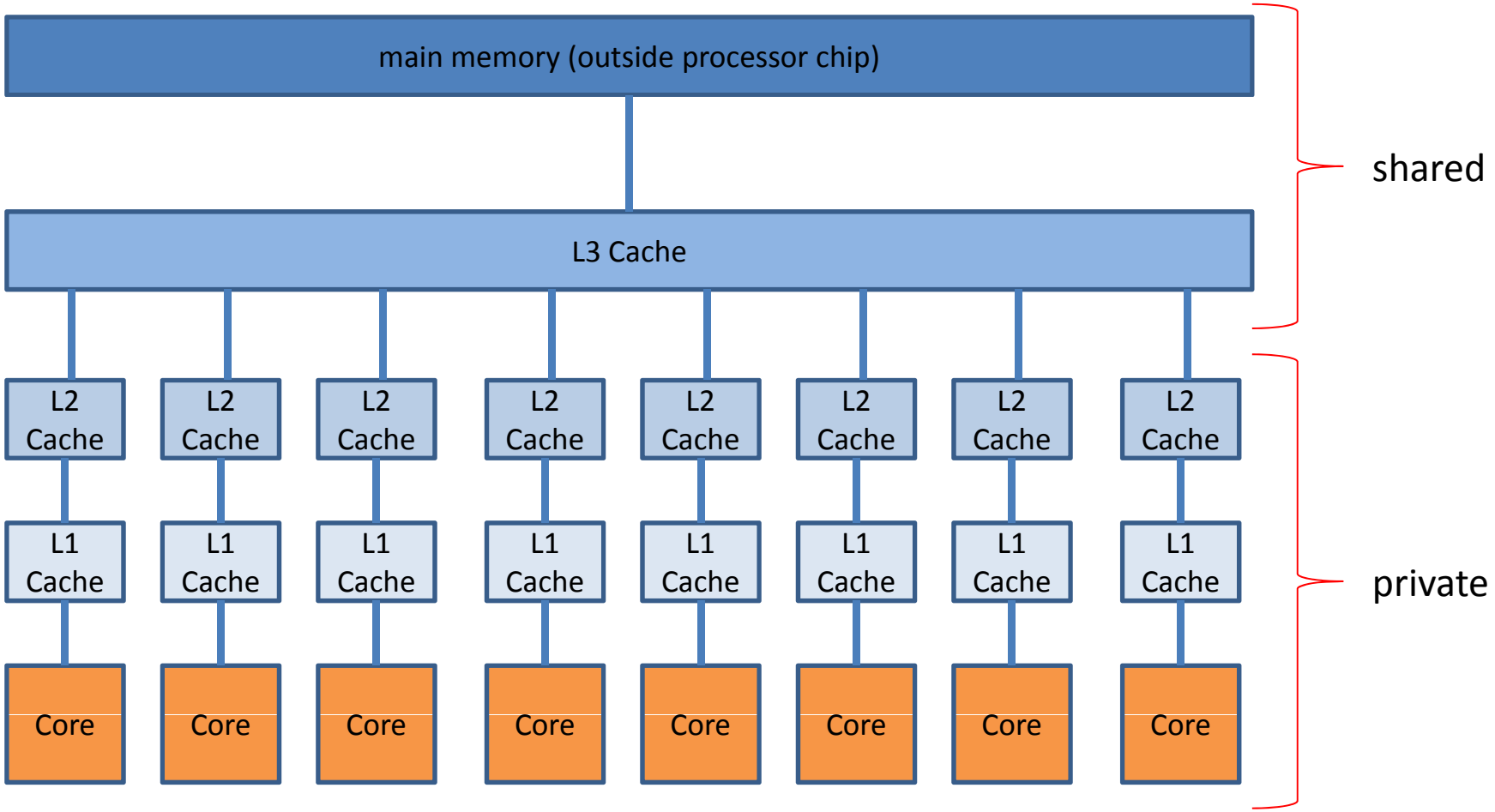
- Development auf Audi TT version 2 (Source: Audi)
- 4 years development time of the car

- Approx. 250 different simulation tools used
- In average 1.000 simulation runs per week
- A maximum of 5.000 simulation runs per week
- One simulation run can take up to 1 week

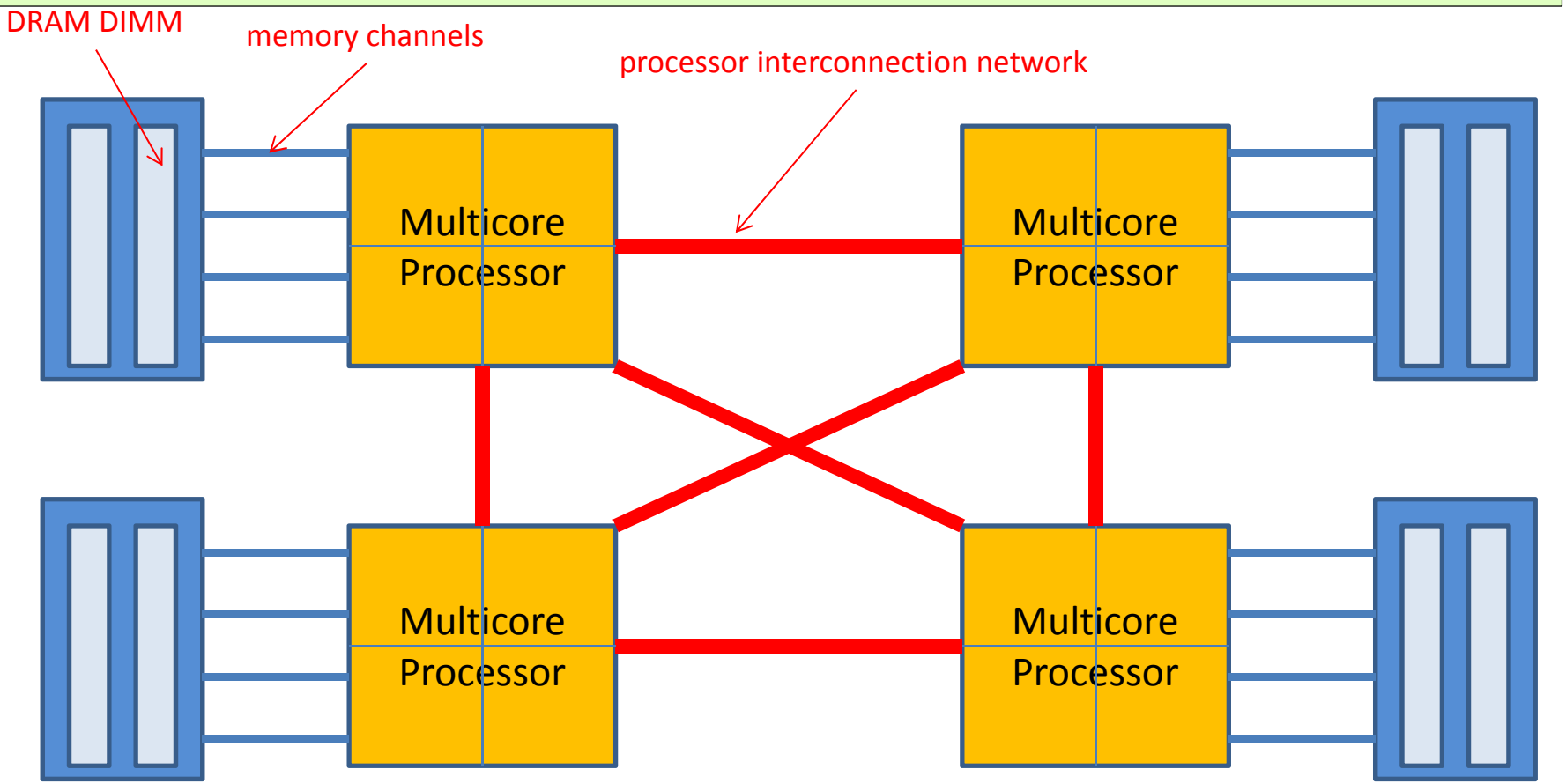
Outline

- Performance Limitations / Why Parallelism?
- **Parallel Systems are Complex**
- Programming Parallel Systems
- Chances and Challenges
- Summary

Generic Multicore Processor



Multiprocessor System



Issues (see later for details):

- limited memory bandwidth if data is allocated on one/few DRAM block
- NUMA: Non-Uniform Memory Access
- Cache coherence protocol with data sharing

Accelerator Technology

- Mainly for compute bound problems, there are accelerator technologies that can boost performance substantially if the problem is suitable and if an accelerator is used appropriately

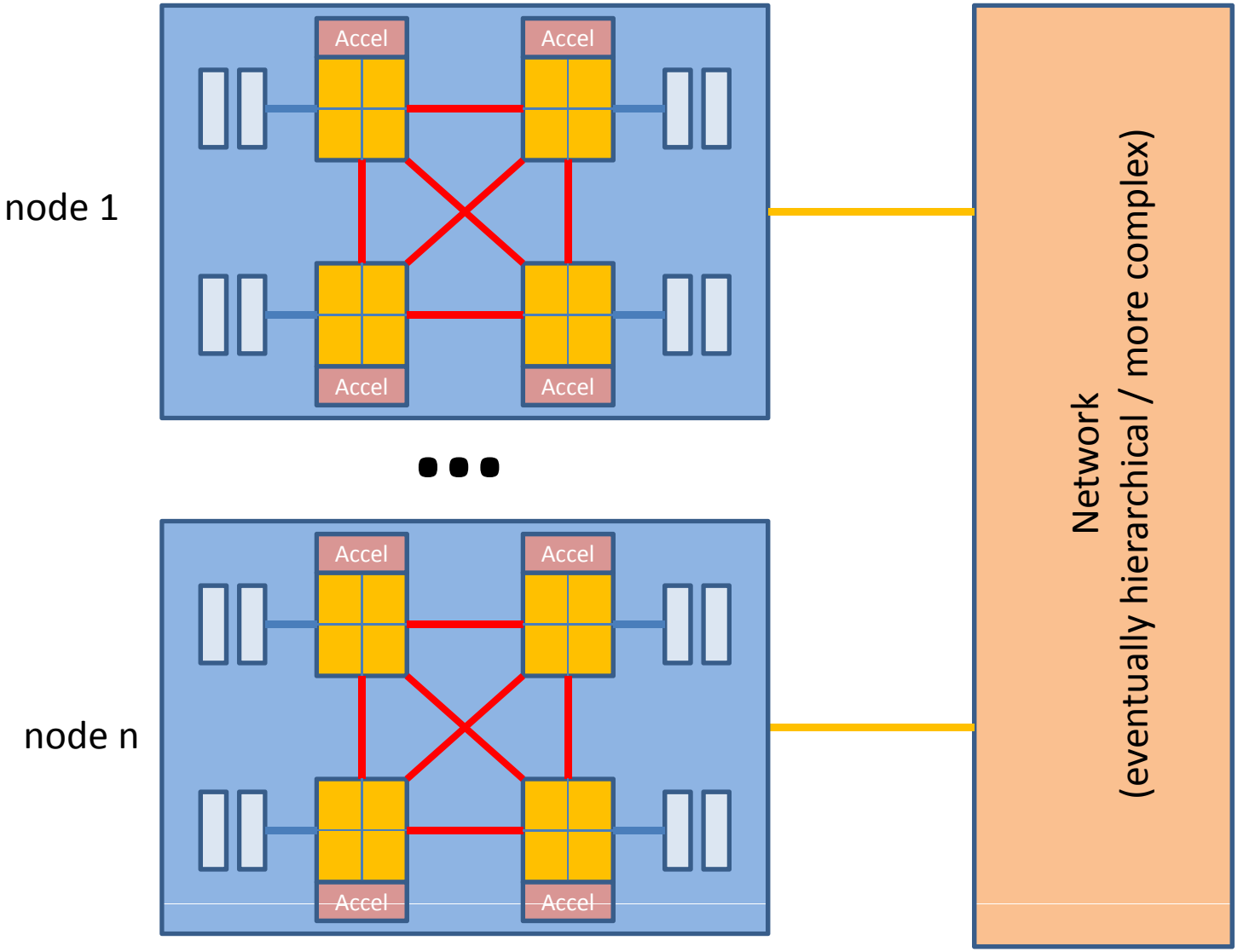
1. SSE / AVX: Extension to Intel/AMD processor architecture to process a vector with one instruction (up to 256 bit at a time)

2. Graphic processors (Nvidia, AMD) on PCIe cards: up to approx. 2.800 tiny processors organized in clusters. Threads teams execute on a cluster in lock step mode

3. Intel Xeon Phi on PCIe card: approx. 60 cores with 4x SMT and wide AVX units (up to 512 bit)

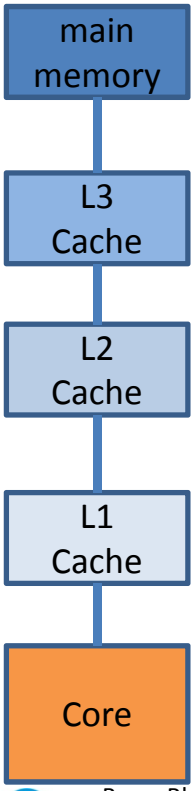
- Problem with 2+3 currently: data transfer, non-coherent separated memory

Large Parallel System



Memory Hierarchy / Topology

- Applications must generate / ensure **data locality** with their access behaviour
- Read latency shown for an Intel Xeon E5-2697v2 based MP system



176 cycles

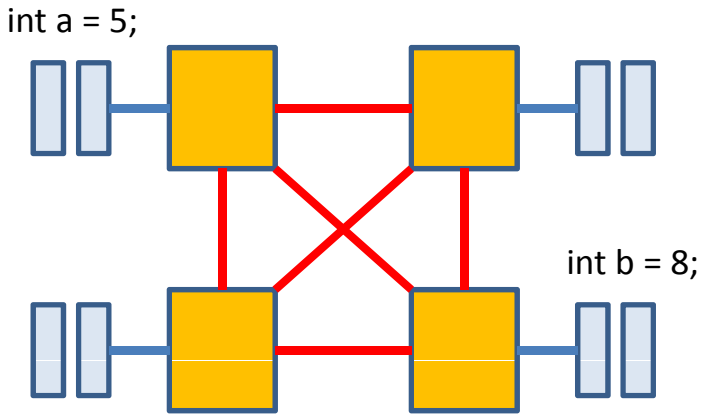
38 cycles

9 cycles

3 cycles

- For multiprocessor systems: **NUMA : Non-Uniform Memory Access**

1. Processors see different access times to different memory locations
2. Data needs to be distributed to get accumulated memory bandwidth
3. Processors should access data near to the processor
4. The coherence protocol between processors may invalidate cache lines



Outline

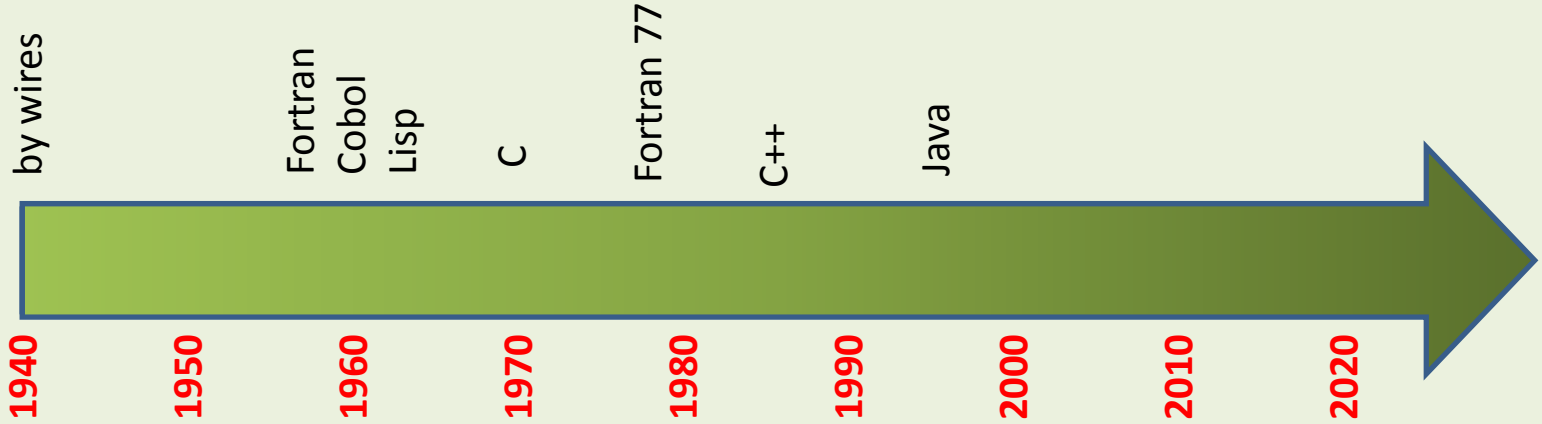
- Performance Limitations / Why Parallelism?
- Parallel Systems are Complex

- **Programming Parallel Systems**

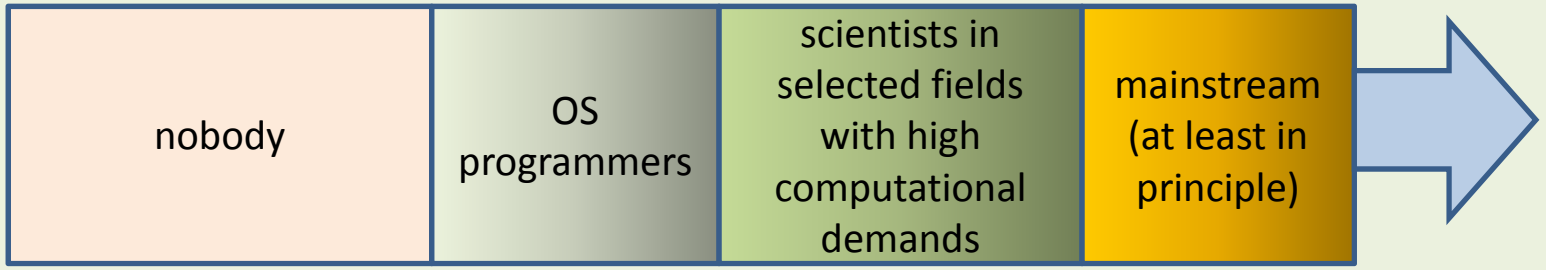
- Chances and Challenges
- Summary

Parallel Programming

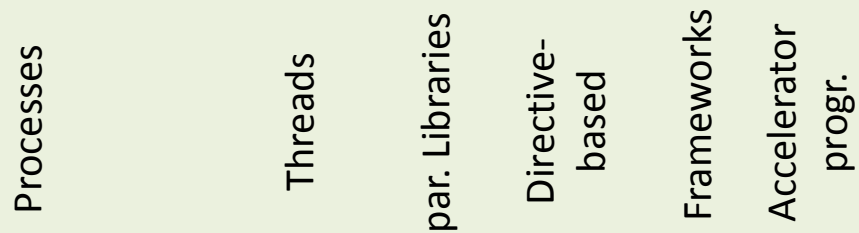
Programming Languages



Parallel Programming Used in Science By



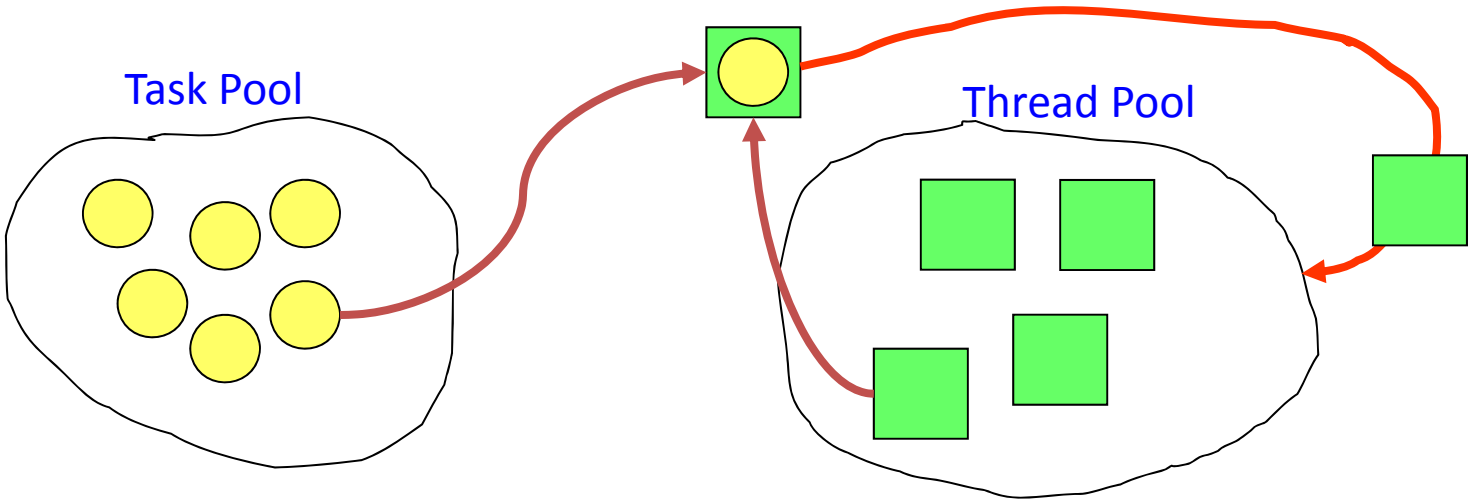
Parallel Programming Concepts



Shared Memory Programming: Threads

- Pure Threads:
  - Posix threads IEEE POSIX 1003.1c-2013, new thread functionality in C11 / C++11
  - Coarse-grained problems, multi-threaded servers

- Thread Pools and task parallelism (e.g., Java Concurrency Framework)
  - Independent tasks are submitted (λ-functions are sometimes quite convenient)
  - A thread in a thread pool picks up a task and works on it
  - A thread pool is configurable and programmable (degree of parallelism, task priorities, FIFO,…)



Shared Memory Programming: OpenMP

- http://www.openmp.org/
- Directive-based parallel programming on top of C, C++, and Fortran
- Fork-Join model
- All relevant compilers (GNU, Intel, PGI, Microsoft,…) understand OpenMP today
- Widespread use from embedded systems up to (single nodes in) HPC systems
- In many application areas the main programming approach today for shared memory parallel programming

- Parallel loops, parallel tasks, accelerator programming with OpenMP 4.0
- But support for data handling / data placement combined with load balance issues is still rather limited

```
double s = 0.0;                          // shared variable
#pragma omp parallel                     // n threads instantiated
 {
  #pragma omp for reduction(+:s)         // loop iterations are
  for(int i=0; i<N; ++i)                 // distributed to threads
      s += a[i] * b[i];
 }
```

Accelerators

- Graphic processors, Intel MIC, FPGA,…
- Accelerators are quite different in architecture and programming compared to "normal" computers
- Low level programming:
  - CUDA: NVIDIA-own programming model for their graphic processors; widely used for Nvidia processors; really good marketing
  - OpenCL: initiated by Apple and others, platform-independent
- High level programming:
  - OpenACC (since 2011): separated initiative by Nvidia/PGI, Cray, CAPS; driving development
  - OpenMP 4.0: all others; lacking 2 years behind

Leaving Shared Memory: Message Passing

- Large parallel systems do not have a single shared memory across all nodes
- Main programming model is message passing
- Standard since 25 years in HPC: MPI (Message Passing Interface)
- `http://www.mpi-forum.org/`

- Rather complex, low level, and error-prone programming
- Distribute data yourself
-
- You do not want to program with MPI unless you are forced to it

Leaving Shared Memory: … Or Not

- Partitioned Global Address Space: PGAS languages

- Couple scalability of distributed memory parallel systems with a shared memory programming model
- Leave part of the created complexity to a compiler / runtime system

- Interesting, but not widely used

- Examples:
  - Coarray Fortran (initial description: R.W.Numrich, J.K.Reid: Co-Array Fortran for Parallel Programming. Fortran Forum, vol. 17, no. 2)
  - Unified Parallel C (UPC; `http://www.upc-lang.org/`)

Outline

- Performance Limitations / Why Parallelism?
- Parallel Systems are Complex
- Programming Parallel Systems

- **Chances and Challenges**

- Summary

Requirements for Future Applications

- In the following, we restrict the discussion to non-trivial applications with certain computational demands (e.g., other than change a single address in an address book)

- Such applications for future computer systems (mobile phones up to HPC) will be restricted by:
  - data movement
  - energy consumption
  - heat generation

- Applications for any computer
  - have to handle parallelism in one or the other way (explicit or implicit)

- Applications for ExaFlop/s-class computers
  - need billion-way parallelism
  - must be able to use multiple / different structures of parallelism simultaneously

Chances: Models with Higher Accuracy

- We will be able to use models with higher accuracy / larger models

- Example: weather forecast

	2002	prediction 2010+
forecast	3-day at 93%	5-day at >90%
7-day forecast	at 62%	at 75%
Horizontal resolution (2 dimensions)	100 km	10 km
Vertical levels	55	100
Time step	30 min	6 min
Computing capability	10 GFLOPS	20-50 TFLOPS
Computing capacity (test, validation, reanalysis, ...)	100 GFLOPS	400 GF - 1 PF
Data volume per day (input / output)	400 MB / 2 TB	1 PB / 10 PB
Networking / Storage per day	4 TB	20 PB
Archival per day	1 TB	10 PB

factor 1000

source: Earth Science Enterprise Computational Technology Requirements Workshop, 2002

Chances: New Functionality

- With more powerful processing capabilities we are able to do new things that were not possible before

- Example: Driverless cars
  - Plans in industry: 2020 first commercial products (starting with limited functionality)
  - It takes a lot of computational power to process all gathered sensor data and even more computational power to make intelligent decisions
  - (You are very power restricted in a car)

Chances: Scaling

- There are already applications that scale to 100.000+ cores

- Example:
  - Barnes-Hut tree code for particle simulations
  - MPI + Threads
  - M.Winkel et.al: Comp. Phys. Commun. 187, 880 (2012)
  - Strong scaling on approx. 250.000 processor cores

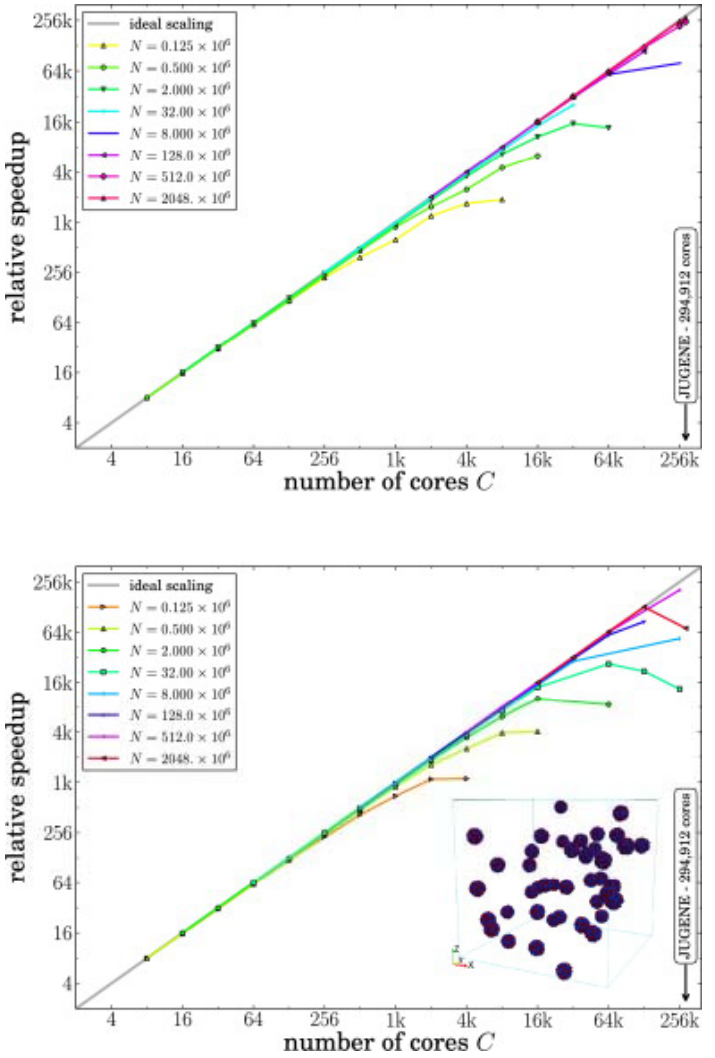


Fig 5: Strong scaling of the tree traversal and force computation part of the algorithm for different total particle numbers *N* and compute cores *C* for the homogeneous (top) and inhomogeneous (bottom) particle distributions. The insets show the test particle setups used for the performance analysis.

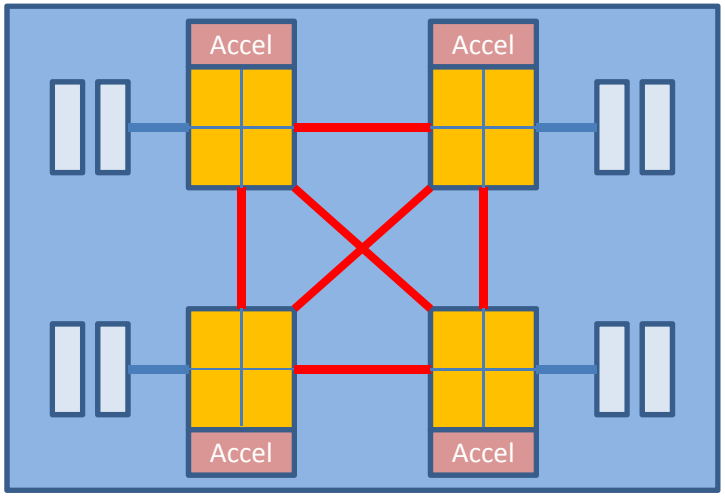
source for picture and description: Comp. Phys. Commun. 187, 880 (2012)

Challenges: Programming Models

- Today, we have no single programming model, that is appropriate for the future in terms of
  - good abstraction of parallelism
  - performance portability
  - scalability
  - convenience
  - team programmability
  - managebility for implementers of such models

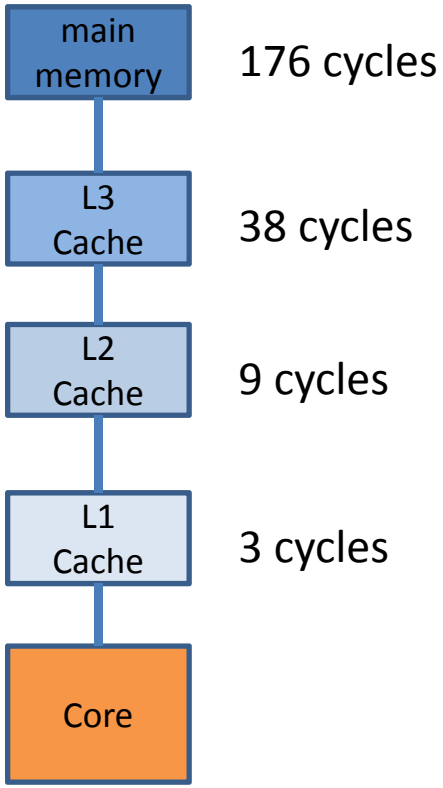
Challenges: System Complexity

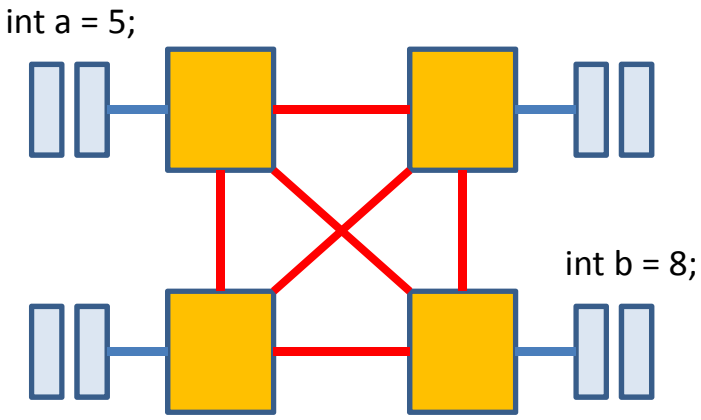
- Parallel systems (small and large) get more and more complex with every generation
- How can we govern system complexity on all levels?
- Can a programmer handle this all? (answer: no)
- Complex compilers, runtime systems and frameworks must do a lot of work for us in the future
- Specialists will program these support systems that many will rely on



Challenges: Memory Hierachy / Topology

- Hierarchies will get deeper and more complex
- Keeping caches coherent takes a lot of time / latency / communication / energy
- Software-controlled caches are an alternative (the idea is not new).





Challenges: Enough Parallelism and Load Balance

- Finding enough parallelism:
  - Finding enough parallel tasks without dependencies might be challenging for applications that do not process TB of data
  - Non-traditional processors / processing may be an alternative, e.g., dataflow computers

- Load Balance:
  - If we have enough parallelism, who handles that?
  - Who is responsible which of 2 billion tasks get scheduled to which cores in a complex system?
  - The problem is even more complex and must be solved as a whole:
    - Load balance itself
    - Communication minimization
    - Data locality

Challenges: Fault Tolerance

- Traditional computers never fail in parts, but as a whole.

- What about systems with millions of important parts (processors, RAM, disc, wires, fans, boards, transceivers,…)?
- Should the failure of one part imply the failure of the system as a whole?
- The discussion is often on hardware. What about the software level?
- We need software, that can handle failures of parts.
- We need applications, that can handle fauilure in parts.
- Software means: OS, middleware, and application level

- Example (in research): fault tolerant MPI (`http://icl.cs.utk.edu/ftmpi/`) as a middleware that supports fault tolerance at an application level

Outline

- Performance Limitations / Why Parallelism?
- Parallel Systems are Complex
- Programming Parallel Systems
- Chances and Challenges

- **Summary**

Summary

- Parallelism is necessary and already available everywhere.
- The power and heat budget is restricted. Getting more computational power with a restricted power budget is challenging now already, and even more in the future.
- Systems get more complex. The number and severity of problems to be solved by a developer will grow with system complexity. Solving many of such things will be delegated to compilers, runtime systems, and frameworks.