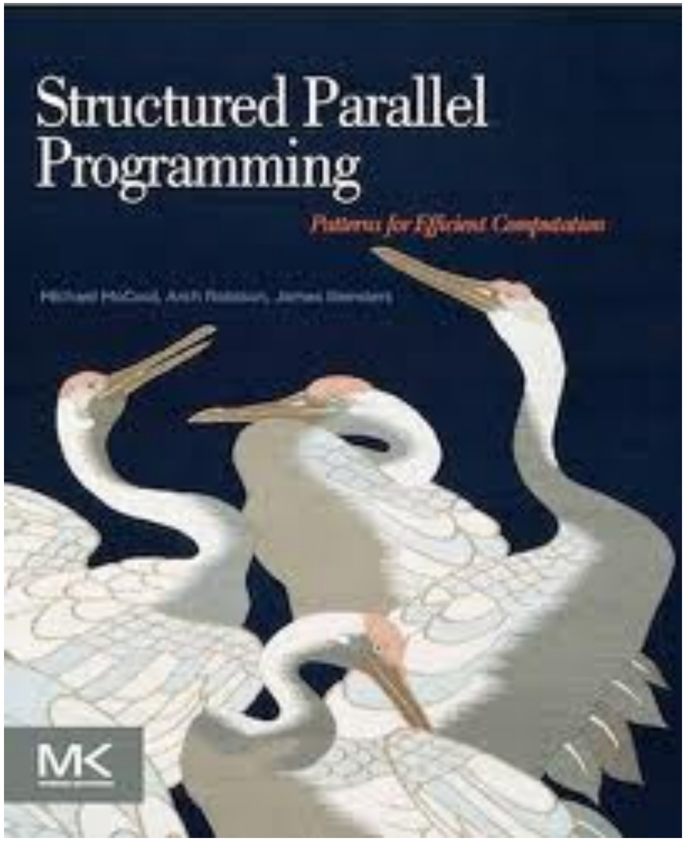
Required Course Book

- "Structured Parallel Programming: Patterns for Efficient Computation," Michael McCool, Arch Robinson, James Reinders, 1st edition, Morgan Kaufmann, ISBN: 978-0-12-415993-8, 2012 http://parallelbook.com/
- Presents parallel programming from a point of view of patterns relevant to parallel computation
  - Map, Collectives, Data reorganization, Stencil and recurrence, Fork-Join, Pipeline
- Focuses on the use of shared memory parallel programming languages and environments
  - Intel Thread Building Blocks (TBB)
  - Intel Cilk Plus



Overview

- Broad/Old field of computer science concerned with:
  - Architecture, HW/SW systems, languages, programming paradigms, algorithms, and theoretical models
  - Computing in parallel
- Performance is the *raison d'être* for parallelism
  - High-performance computing
  - Drives computational science revolution
- Topics of study
  - Parallel architectures
  - Parallel programming
  - Parallel algorithms
  - Parallel performance models and tools
  - Parallel applications

Parallel Processing – What is it?

- A *parallel computer* is a computer system that uses multiple processing elements simultaneously in a cooperative manner to solve a computational problem
- *Parallel processing* includes techniques and technologies that make it possible to compute in parallel
  - Hardware, networks, operating systems, parallel libraries, languages, compilers, algorithms, tools, …
- Parallel computing is an evolution of serial computing
  - Parallelism is natural
  - Computing problems differ in level / type of parallelism
- Parallelism is all about performance! Really?

Concurrency

- Consider multiple tasks to be executed in a computer
- Tasks are concurrent with respect to each if
  - They *can* execute at the same time (*concurrent execution*)
  - Implies that there are no dependencies between the tasks
- Dependencies
  - If a task requires results produced by other tasks in order to execute correctly, the task's execution is *dependent*
  - If two tasks are dependent, they are not concurrent
  - Some form of synchronization must be used to enforce (satisfy) dependencies
- Concurrency is fundamental to computer science
  - Operating systems, databases, networking, …

Concurrency and Parallelism

- Concurrent is not the same as parallel! Why?
- Parallel execution
  - Concurrent tasks *actually* execute at the same time
  - Multiple (processing) resources have to be available
- **Parallelism = concurrency + "parallel" hardware**
  - Both are required
  - Find concurrent execution opportunities
  - Develop application to execute in parallel
  - Run application on parallel hardware
- Is a parallel application a concurrent application?
- Is a parallel application run with one processor parallel? Why or why not?

Parallelism

- There are granularities of parallelism (parallel execution) in programs
  - Processes, threads, routines, statements, instructions, …
  - Think about what are the software elements that execute concurrently
- These must be supported by hardware resources
  - Processors, cores, … (execution of instructions)
  - Memory, DMA, networks, … (other associated operations)
  - All aspects of computer architecture offer opportunities for parallel hardware execution
- Concurrency is a necessary condition for parallelism
  - Where can you find concurrency?
  - How is concurrency expressed to exploit parallel systems?

Why use parallel processing?

- Two primary reasons (both performance related)
  - Faster time to solution (response time)
  - Solve bigger computing problems (in same time)
- Other factors motivate parallel processing
  - Effective use of machine resources
  - Cost efficiencies
  - Overcoming memory constraints
- Serial machines have inherent limitations
  - Processor speed, memory bottlenecks, …
- Parallelism has become the future of computing
- Performance is still the driving concern
- **Parallelism = concurrency + parallel HW + performance**

Perspectives on Parallel Processing

- Parallel computer architecture
  - Hardware needed for parallel execution?
  - Computer system design
- (Parallel) Operating system
  - How to manage systems aspects in a parallel computer
- Parallel programming
  - Libraries (low-level, high-level)
  - Languages
  - Software development environments
- Parallel algorithms
- Parallel performance evaluation
- Parallel tools
  - Performance, analytics, visualization, …

Why study parallel computing today?

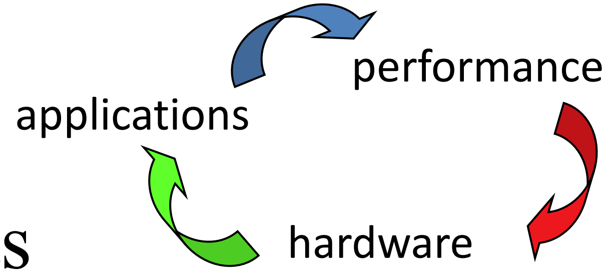
- Computing architecture
  - Innovations often drive to novel programming models
- Technological convergence
  - The "killer micro" is ubiquitous
  - Laptops and supercomputers are fundamentally similar!
  - Trends cause diverse approaches to converge
- Technological trends make parallel computing inevitable
  - Multi-core processors are here to stay!
  - Practically every computing system is operating in parallel
- Understand fundamental principles and design tradeoffs
  - Programming, systems support, communication, memory, …
  - Performance
- Parallelism is the future of computing

Inevitability of Parallel Computing

- Application demands
  - Insatiable need for computing cycles
- Technology trends
  - Processor and memory
- Architecture trends
- Economics
- Current trends:
  - Today's microprocessors have multiprocessor support
  - Servers and workstations available as multiprocessors
  - Tomorrow's microprocessors are multiprocessors
  - Multi-core is here to stay and #cores/processor is growing
  - Accelerators (GPUs, gaming systems)

Application Characteristics

- Application performance demands hardware advances
- Hardware advances generate new applications
- New applications have greater performance demands
  - Exponential increase in microprocessor performance
  - Innovations in parallel architecture and integration



- Range of performance requirements
  - System performance must also improve as a whole
  - Performance requirements require computer engineering
  - Costs addressed through technology advancements

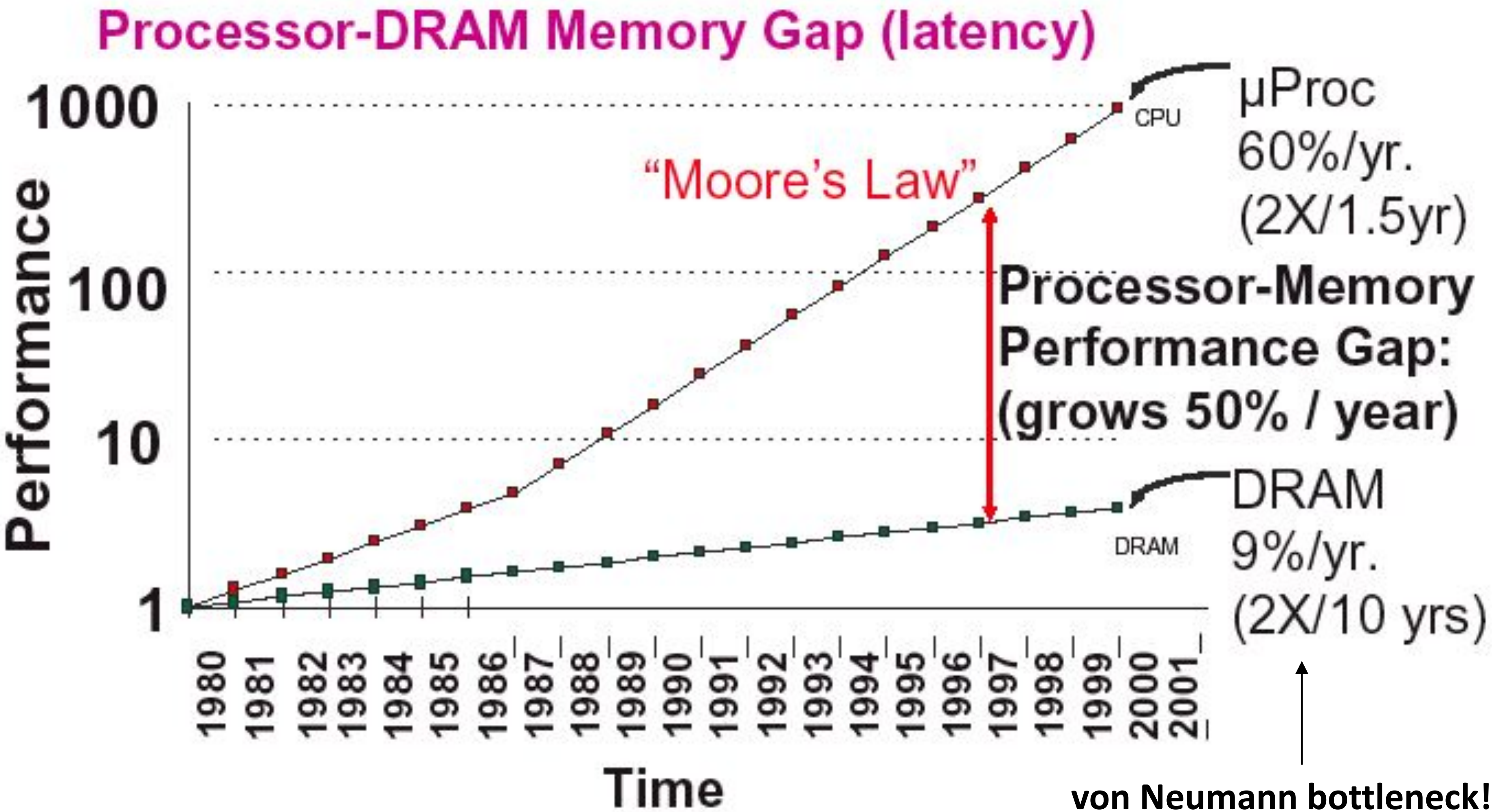
Broad Parallel Architecture Issues

- Resource allocation
  - How many processing elements?
  - How powerful are the elements?
  - How much memory?
- Data access, communication, and synchronization
  - How do the elements cooperate and communicate?
  - How are data transmitted between processors?
  - What are the abstractions and primitives for cooperation?
- Performance and scalability
  - How does it all translate into performance?
  - How does it scale?

Leveraging Moore's Law

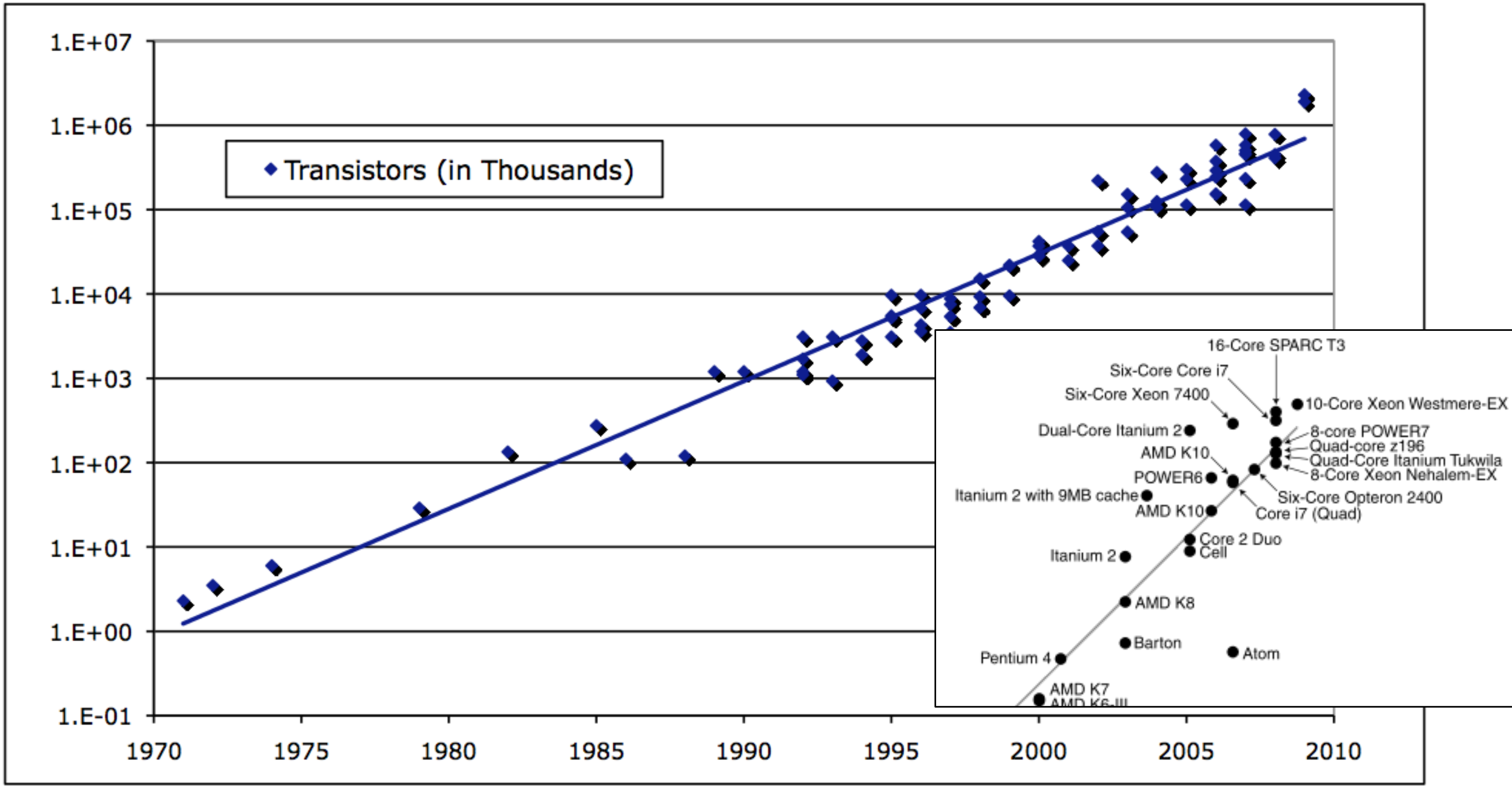
- More transistors = more parallelism opportunities
- Microprocessors
  - Implicit parallelism
    - pipelining
    - multiple functional units
    - superscalar
  - Explicit parallelism
    - SIMD instructions
    - long instruction works

What's Driving Parallel Computing Architecture?



von Neumann bottleneck!!
(memory wall)

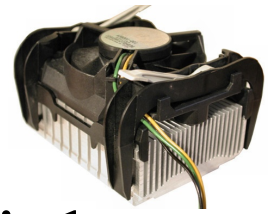
Microprocessor Transitor Counts (1971-2011)



Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç
Slide from Kathy Yelick

What has happened in the last several years?

- Processing chip manufacturers increased processor performance by increasing CPU clock frequency
  - Riding Moore's law
- Until the chips got too hot!
  - Greater clock frequency ⇒ greater electrical power
  - Pentium 4 heat sink
  - Frying an egg on a Pentium 4



- Add multiple cores to add performance
  - Keep clock frequency same or reduced
  - Keep lid on power requirements

Classifying Parallel Systems – Flynn's Taxonomy

- Distinguishes multi-processor computer architectures along the two independent dimensions
  - ***Instruction*** and ***Data***
  - Each dimension can have one state: ***Single*** or ***Multiple***
- SISD: Single Instruction, Single Data
  - Serial (non-parallel) machine
- SIMD: Single Instruction, Multiple Data
  - Processor arrays and vector machines
- MISD: Multiple Instruction, Single Data (weird)
- MIMD: Multiple Instruction, Multiple Data
  - Most common parallel computer systems

Parallel Architecture Types

- Instruction-Level Parallelism
  - Parallelism captured in instruction processing
- Vector processors
  - Operations on multiple data stored in vector registers
- Shared-memory Multiprocessor (SMP)
  - Multiple processors sharing memory
  - Symmetric Multiprocessor (SMP)
- Multicomputer
  - Multiple computer connect via network
  - Distributed-memory cluster
- Massively Parallel Processor (MPP)

Phases of Supercomputing (Parallel) Architecture

- Phase 1 (1950s): sequential instruction execution
- Phase 2 (1960s): sequential instruction issue
  - Pipeline execution, reservations stations
  - Instruction Level Parallelism (ILP)
- Phase 3 (1970s): vector processors
  - Pipelined arithmetic units
  - Registers, multi-bank (parallel) memory systems
- Phase 4 (1980s): SIMD and SMPs
- Phase 5 (1990s): MPPs and clusters
  - Communicating sequential processors
- Phase 6 (>2000): many cores, accelerators, scale, …

Performance Expectations

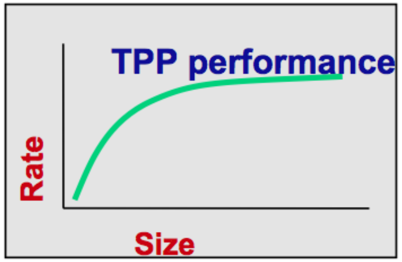
- If each processor is rated at k MFLOPS and there are p processors, we should expect to see k*p MFLOPS performance? Correct?
- If it takes 100 seconds on 1 processor, it should take 10 seconds on 10 processors? Correct?
- Several causes affect performance
  - Each must be understood separately
  - But they interact with each other in complex ways
    - solution to one problem may create another
    - one problem may mask another
- Scaling (system, problem size) can change conditions
- Need to understand performance space

Scalability

- A program can scale up to use many processors
  - What does that mean?
- How do you evaluate scalability?
- How do you evaluate scalability goodness?
- Comparative evaluation
  - If double the number of processors, what to expect?
  - Is scalability linear?
- Use parallel efficiency measure
  - Is efficiency retained as problem size increases?
- Apply performance metrics

Top 500 Benchmarking Methodology

- Listing of the world's 500 most powerful computers
- Yardstick for high-performance computing (HPC)
  - Rmax : maximal performance Linpack benchmark
    - dense linear system of equations (Ax = b)
- Data listed
  - Rpeak : theoretical peak performance
  - Nmax : problem size needed to achieve Rmax
  - N1/2 : problem size needed to achieve 1/2 of Rmax
  - Manufacturer and computer type
  - Installation site, location, and year
- Updated twice a year at SC and ISC conferences



Top 10 (November 2013)

Different architectures

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325
7	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462	5,168.1	8,520.1	4,510
8	Forschungszentrum Juelich (FZJ) Germany	**JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458,752	5,008.9	5,872.0	2,301
9	DOE/NNSA/LLNL United States	**Vulcan** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393,216	4,293.3	5,033.2	1,972
10	Leibniz Rechenzentrum Germany	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM	147,456	2,897.0	3,185.1	3,423

Performance Development in Top 500

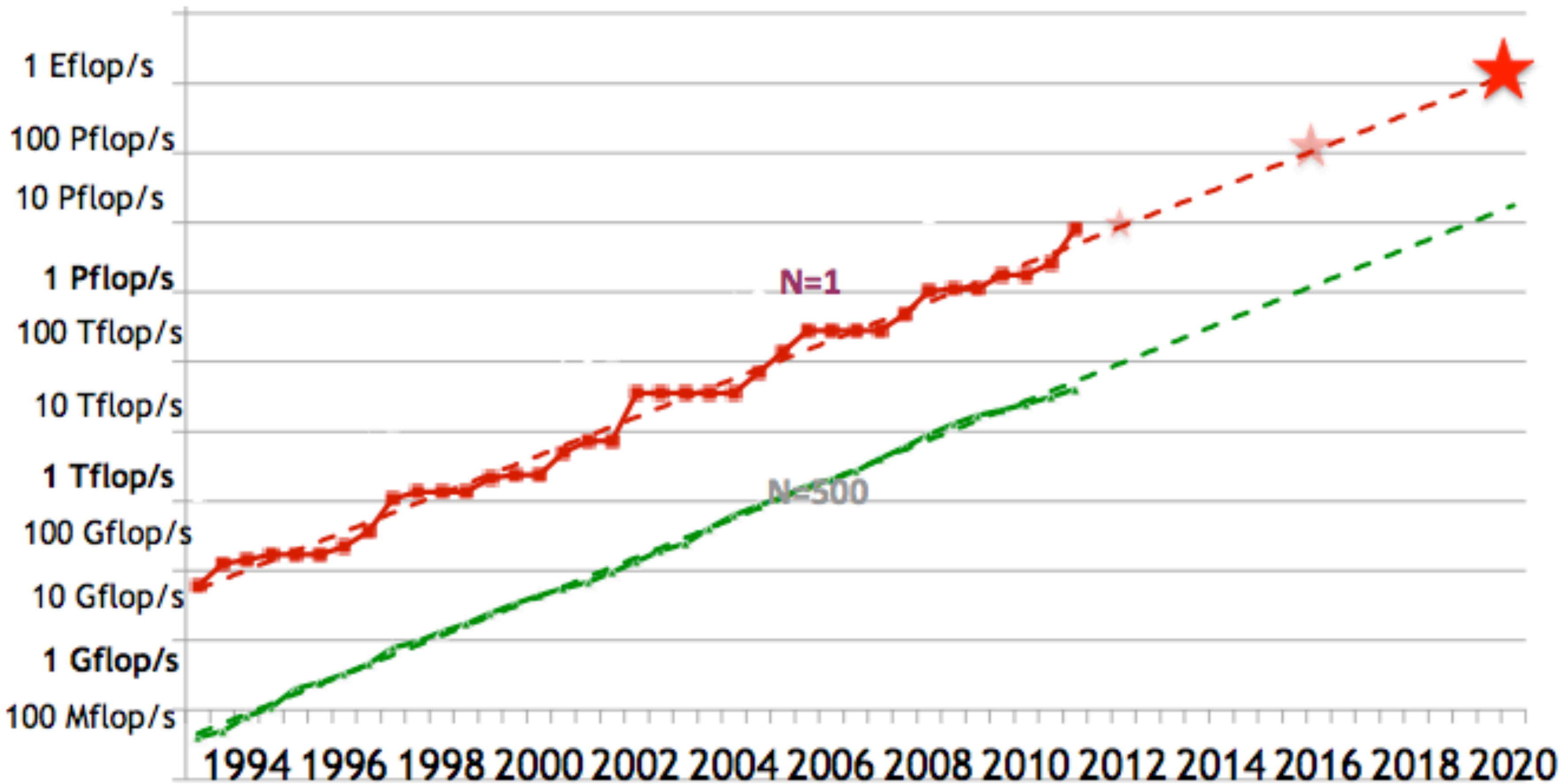


Figure credit: http://www.netlib.org/utk/people/JackDongarra/SLIDES/korea-2011.pdf

Exascale Initiative

- Exascale machines are targeted for 2019
- What are the potential differences and problems?

Systems	2011 K Computer	2019	Difference Today & 2019
System peak	8.7 Pflop/s	1 Eflop/s	O(100)
Power	10 MW	~20 MW	???
System memory	1.6 PB	32 - 64 PB	O(10)
Node performance	128 GF	1,2 or 15TF	O(10) - O(100)
Node memory BW	64 GB/s	2 - 4TB/s	O(100)
Node concurrency	8	O(1k) or 10k	O(100) - O(1000)
Total Node Interconnect BW	20 GB/s	200-400GB/s	O(10)
System size (nodes)	68,544	O(100,000) or O(1M)	O(10) - O(100)
Total concurrency	548,352	O(billion)	O(1,000)
MTTI	days	O(1 day)	- O(10)

Major Changes to Software and Algorithms

- What were we concerned about before and now?
- Must rethink the design for exascale
  - Data movement is expensive (Why?)
  - Flops per second are cheap (Why?)
- Need to reduce communication and sychronization
- Need to develop fault-resilient algorithms
- How do with deal with massive parallelism?
- Software must adapt to the hardware (autotuning)

Supercomputing and Computational Science

- By definition, a supercomputer is of a class of computer systems that are the most powerful computing platforms at that time
- Computational science has always lived at the leading (and bleeding) edge of supercomputing technology
- "Most powerful" depends on performance criteria
  - Performance metrics related to computational algorithms
  - Benchmark "real" application codes
- Where does the performance come from?
  - More powerful processors
  - More processors (cores)
  - Better algorithms

Computational Science

- Traditional scientific methodology
  - Theoretical science
    - Formal systems and theoretical models
    - Insight through abstraction, reasoning through proofs
  - Experimental science
    - Real system and empirical models
    - Insight from observation, reasoning from experiment design
- Computational science
  - Emerging as a principal means of scientific research
  - Use of computational methods to model scientific problems
    - Numerical analysis plus simulation methods
    - Computer science tools
  - Study and application of these solution techniques

Computational Challenges

- Computational science thrives on computer power
  - Faster solutions
  - Finer resolution
  - Bigger problems
  - Improved interaction
  - BETTER SCIENCE!!!
- How to get more computer power?
  - Scalable parallel computing
- Computational science also thrives better integration
  - Couple computational resources
  - Grid computing

Scalable Parallel Computing

- Scalability in parallel architecture
  - Processor numbers
  - Memory architecture
  - Interconnection network
  - Avoid critical architecture bottlenecks
- Scalability in computational problem
  - Problem size
  - Computational algorithms
    - Computation to memory access ratio
    - Computation to communication ration
- Parallel programming models and tools
- Performance scalability