Parallel Computing

Dirk Roose
Office: 200A 03.25

Albert-Jan Yzelman
Office: 200A 03.18

H03F9A

Parallel computing

Dirk Roose & Albert-Jan Yzelman

Semester 1

Aims

Insight in

- parallel computers and available software environments,
- the design and performance analysis of parallel algorithms.

HPC Cluster with 8448 'cores' (VSC)

The student will be able to

- design efficient parallel versions of algorithms with simple data dependencies
- both in the 'shared address space' programming model and in the 'message passing' programming model.

Parallel computing: Content

- Architecture of parallel HPC systems (short)
- Performance analysis on parallel systems
- Design & analysis of parallel algorithms for model problems
  (matrix operations, sorting, ...)
  using the BSP model
- Simple examples in BSPlib, BSPonMPI, MulticoreBSP
  (MPI: Message Passing Interface)
- Dynamic load-balancing
- Guest lecture on Parallel Matching by Rob Bisseling
- ...
-

Parallel computing

Prerequisites

Bachelor-level knowledge of algorithms & programming

Course material

Rob H. Bisseling, *Parallel Scientific Computation.*
A Structured Approach using BSP and MPI.
Oxford University Press, 2004.

+ some papers

Exercises and practical sessions

5 sessions (3 on parallel systems:
multicore processor; HPC-cluster)

no project

Exam

Open book exam

- insight in theory (in particular performance analysis)
- design of an efficient parallel algorithm (high level description)

Course textbook

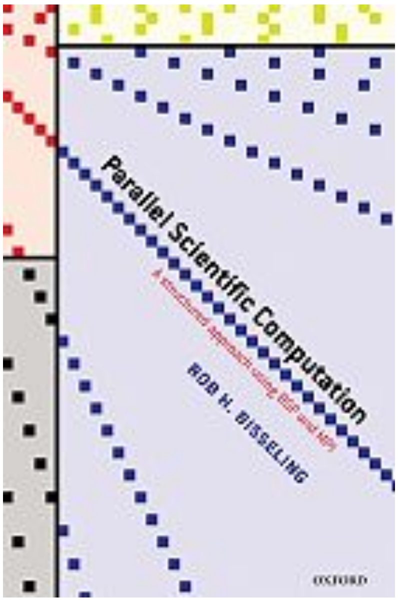
Rob H. Bisseling,
Parallel Scientific Computation.
A Structured Approach using BSP and MPI, Oxford University Press, 2004

(BSP: Bulk Synchronous Parallel;
MPI: Message Passing Interface)

Available in Acco (?)

Table of content of book:

http://ukcatalogue.oup.com/product/9780198529392.do#.UGSxJULv-2U


Parallel Scientific Computation
A structured approach using BSP and MPI
ROB H. BISSELING
OXFORD

Book: Parallel Scientific Computation

- The first text to explain how to use BSP in parallel computing
- Clear exposition of distributed-memory parallel computing with applications to core topics of scientific computation
- Each topic treated follows the complete path from theory to practice
- This is the first text explaining how to use the bulk synchronous parallel (BSP) model […] in parallel algorithm design and parallel programming.

- An appendix on the message-passing interface (MPI) discusses how to program using the MPI communication library. MPI equivalents of all the programs are also presented.

Book: Parallel Scientific Computation (2)

The main topics treated in the book are core in the area of scientific computation: solving dense linear systems by Gaussian elimination, computing fast Fourier transforms, and solving sparse linear systems by iterative methods. Each topic is treated in depth, starting from the problem formulation and a sequential algorithm, through a parallel algorithm and its analysis, to a complete parallel program written in C and BSPlib, and experimental results obtained using this program on a parallel computer.

Additional topics treated in the exercises include: data compression, random number generation, cryptography, eigensystem solving, 3D and Strassen matrix multiplication, wavelets and image compression, fast cosine transform, decimals of pi, simulated annealing, and molecular dynamics.

Extra (separate texts) : sorting, graph coarsening, case studies, scheduling & load balancing

Why parallel computing ?

- **Limits of single computer/processor**
  - available memory size and memory access time
  - performance

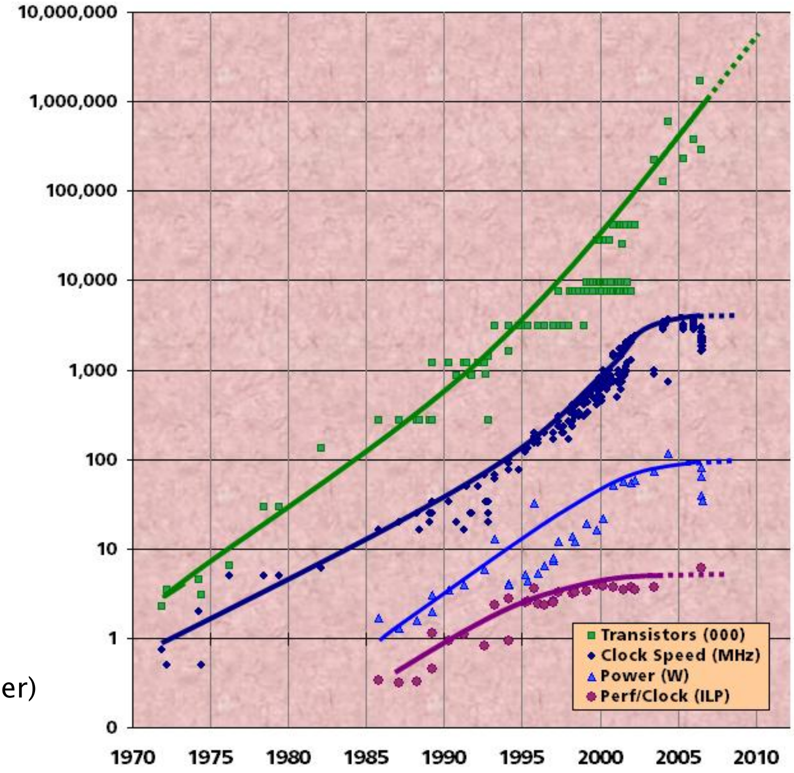
- Until 2007:
  growing **mismatch** between *clock cycle* and *memory access time*
  (clock cycle: + 40% /year; memory access: + 10%/year)
- Since 2007: multicore processors!

- Parallel computing allows
  - to solve problems that don’t fit in the memory of a single processor
  - to solve problems that can’t be solved in a reasonable time on a single core (processor)

Revolution is Happening Now (2005)

- Chip density is continuing to increase ~2x every 2 years
  - Clock speed is not
  - Number of processor cores may double instead
- There is little or no hidden parallelism (ILP) to be found
- Parallelism must be exposed to and managed by software

Source: Intel, Microsoft (Sutter) and Stanford (Olukotun, Hammond)



Manycore processors

Intel Xeon Phi coprocessors beschikken over maximaal 61 cores, 244 threads en 1,2 teraFLOPS aan prestatiekracht en ze zijn beschikbaar in een groot aantal configuraties om aan verschillende wensen op het gebied van hardware, software, werkbelasting, prestatie en efficiëntie te voldoen.

NVIDIA GPU: Tesla K20c:
2496 Cores (Dynamic Parallelism)

Parallel Computing

Introduction & Motivation

adapted version of

slides from

Kathy Yelick and Jim Demmel

EECS & Math Departments

UC Berkeley

www.cs.berkeley.edu/~demmel/cs267_spr11

Outline

- Why ~~powerful~~ all computers must be parallel processors
  - Including your laptops and handhelds
- Large Computational Science and Engineering (CSE) problems require powerful computers
  - Commercial problems too
- Why writing (fast) parallel programs is hard
  - But things are improving
- Principles of parallel computing performance
- ~~Structure of the course~~

Units of Measure

- High Performance Computing (HPC) units are:
  - Flop: floating point operation, usually double precision unless noted
  - Flop/s: floating point operations per second
  - Bytes: size of data (a double precision floating point number is 8)
- Typical sizes are millions, billions, trillions…

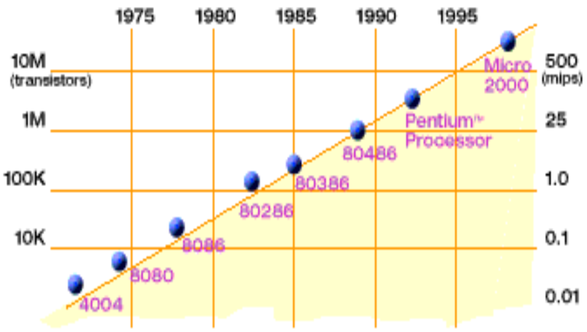
Mega	Mflop/s = 10^6 flop/sec	Mbyte = 2^{20} = 1048576 ~ 10^6 bytes
Giga	Gflop/s = 10^9 flop/sec	Gbyte = 2^{30} ~ 10^9 bytes
Tera	Tflop/s = 10^{12} flop/sec	Tbyte = 2^{40} ~ 10^{12} bytes
Peta	Pflop/s = 10^{15} flop/sec	Pbyte = 2^{50} ~ 10^{15} bytes
Exa	Eflop/s = 10^{18} flop/sec	Ebyte = 2^{60} ~ 10^{18} bytes
Zetta	Zflop/s = 10^{21} flop/sec	Zbyte = 2^{70} ~ 10^{21} bytes
Yotta	Yflop/s = 10^{24} flop/sec	Ybyte = 2^{80} ~ 10^{24} bytes

Why ~~powerful~~ all computers are parallel

(2007)

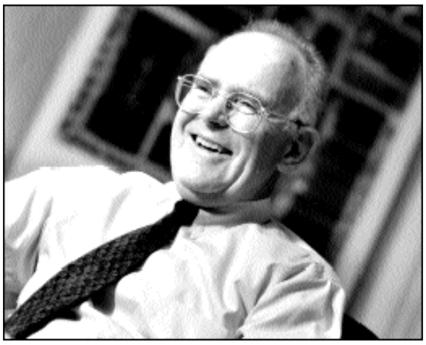
circa 1991–2006

Technology Trends: Microprocessor Capacity



2X transistors/Chip Every 1.5 years
Called "Moore's Law"

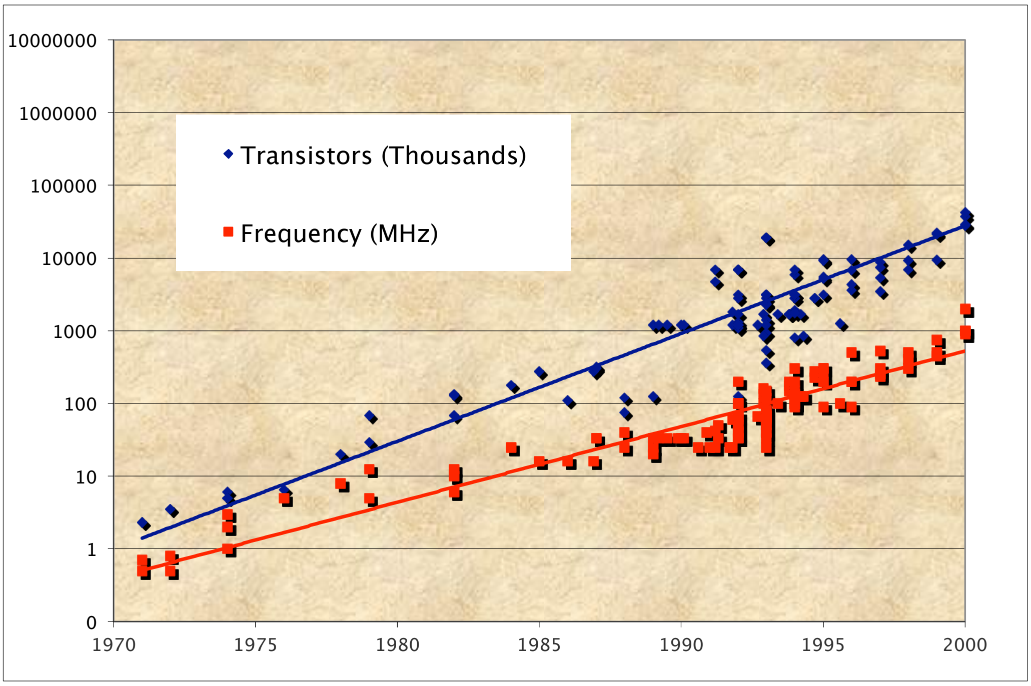
Microprocessors have become smaller, denser, and more powerful.



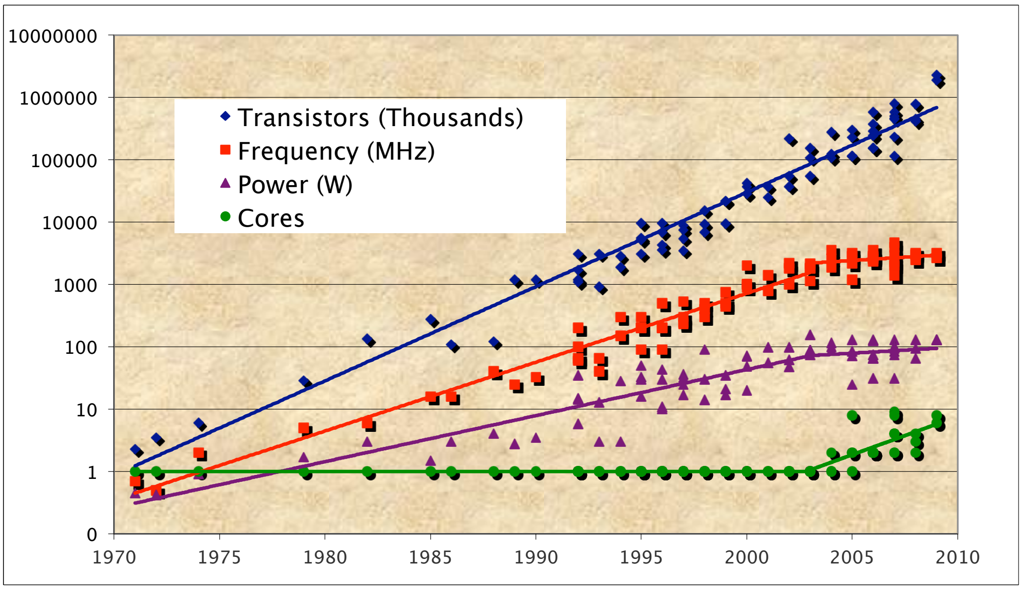
Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Slide source: Jack Dongarra

Microprocessor Transistors / Clock (1970-2000)



Revolution in Processors



- Chip density is continuing to increase ~2x every 2 years
- Clock speed is not
- Number of processor cores may double instead
- Power is under control, no longer growing

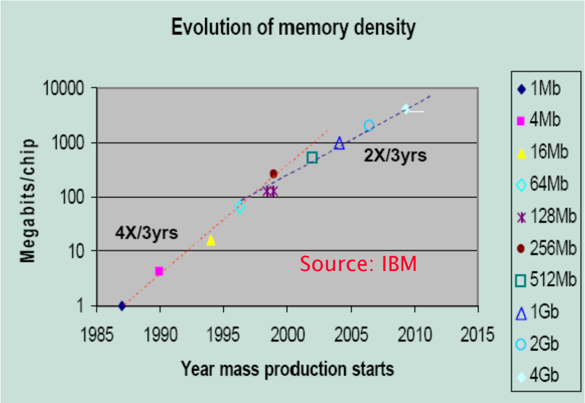
Parallelism in 2011?

- These arguments are no longer theoretical
- All major processor vendors are producing *multicore* chips
  - Every machine will soon be a parallel machine
  - To keep doubling performance, parallelism must double
- Which commercial applications can use this parallelism?
  - Do they have to be rewritten from scratch?
- Will all programmers have to be parallel programmers?
  - New software model needed
  - Try to hide complexity from most programmers – eventually
  - In the meantime, need to understand it
- Computer industry betting on this big change, but does not have all the answers
  - Berkeley ParLab established to work on this

Memory is Not Keeping Pace

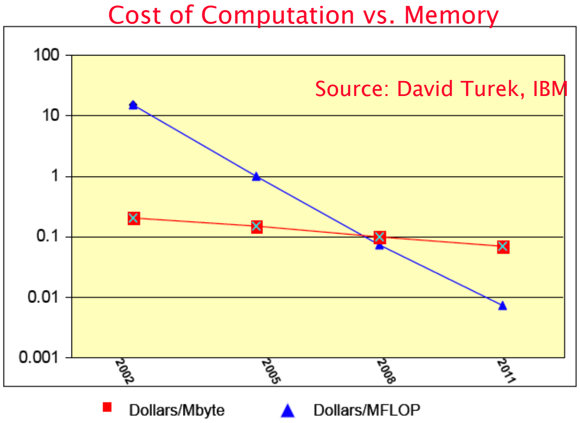
Technology trends against constant or increasing memory per core

- Memory density is doubling every three years; processor logic is every two
- Storage costs (dollars/Mbyte) are dropping gradually compared to logic costs



Source: IBM

Cost of Computation vs. Memory



Source: David Turek, IBM

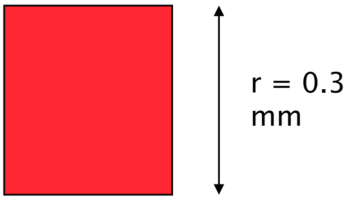
The cost to sense, collect, generate and calculate data is declining much faster than the cost to access, manage and store it

Question: Can you double concurrency without doubling memory?

- **Strong scaling:** fixed problem size, increase number of processors
- **Weak scaling:** grow problem size proportionally to number of processors

More Limits: How fast can a serial computer be?

1 Tflop/s, 1 Tbyte sequential machine



- Consider the 1 Tflop/s sequential machine:
  - Data must travel some distance, r, to get from memory to processor.
  - To get 1 data element per cycle, this means 10^{12} times per second at the speed of light, $c = 3x10^8$ m/s. Thus $r < c/10^{12} = 0.3$ mm.
- Now put 1 Tbyte of storage in a 0.3 mm x 0.3 mm area:
  - Each bit occupies about 1 square Angstrom, or the size of a small atom.
- No choice but parallelism

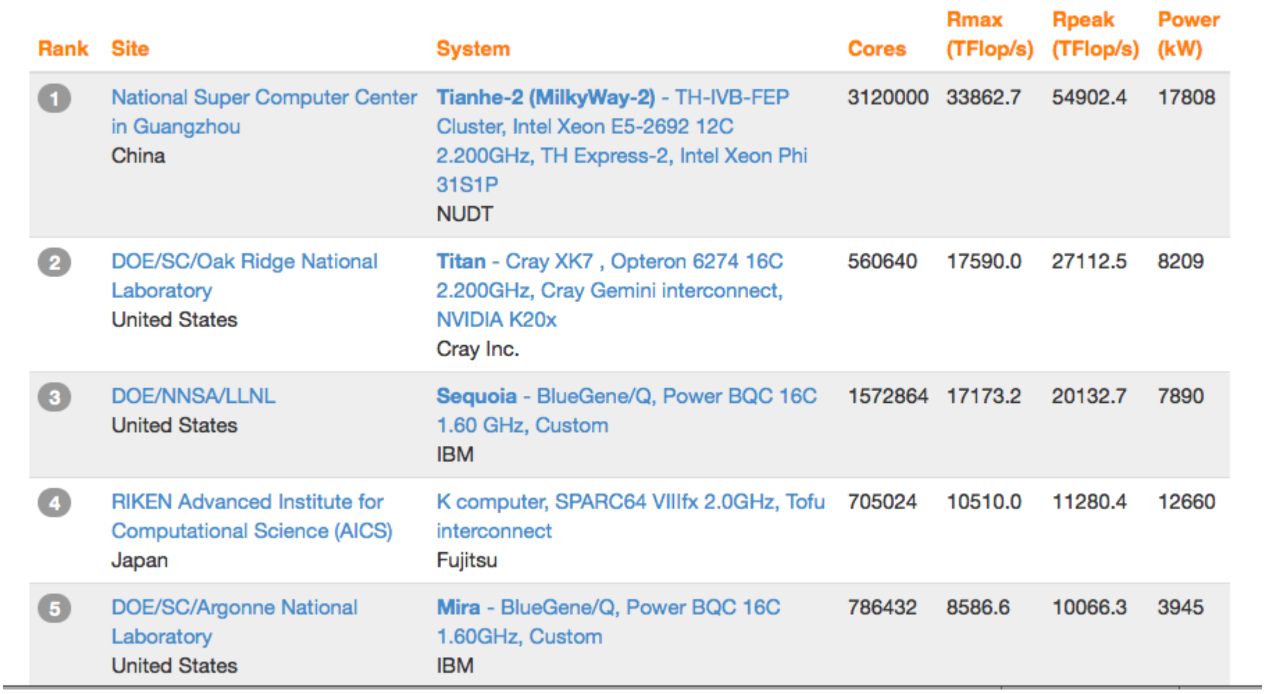
More Exotic Solutions on the Horizon

- Graphics and Game processors
  - Graphics Processing Units (GPUs), e.g., NVIDIA and ATI/AMD
  - Game processors, e.g., Cell for PS3
  - Parallel processor attached to main processor
  - Originally special purpose, getting more general
  - Programming model not yet mature
- FPGAs – Field Programmable Gate Arrays
  - Inefficient use of chip area
  - More efficient than multicore for some domains
  - Programming challenge now includes hardware design, e.g., layout
  - Wire routing heuristics still troublesome;
- Dataflow architectures
  - Have considerable experience with dataflow from 1980's
  - Programming with functional languages?

The TOP500 Project

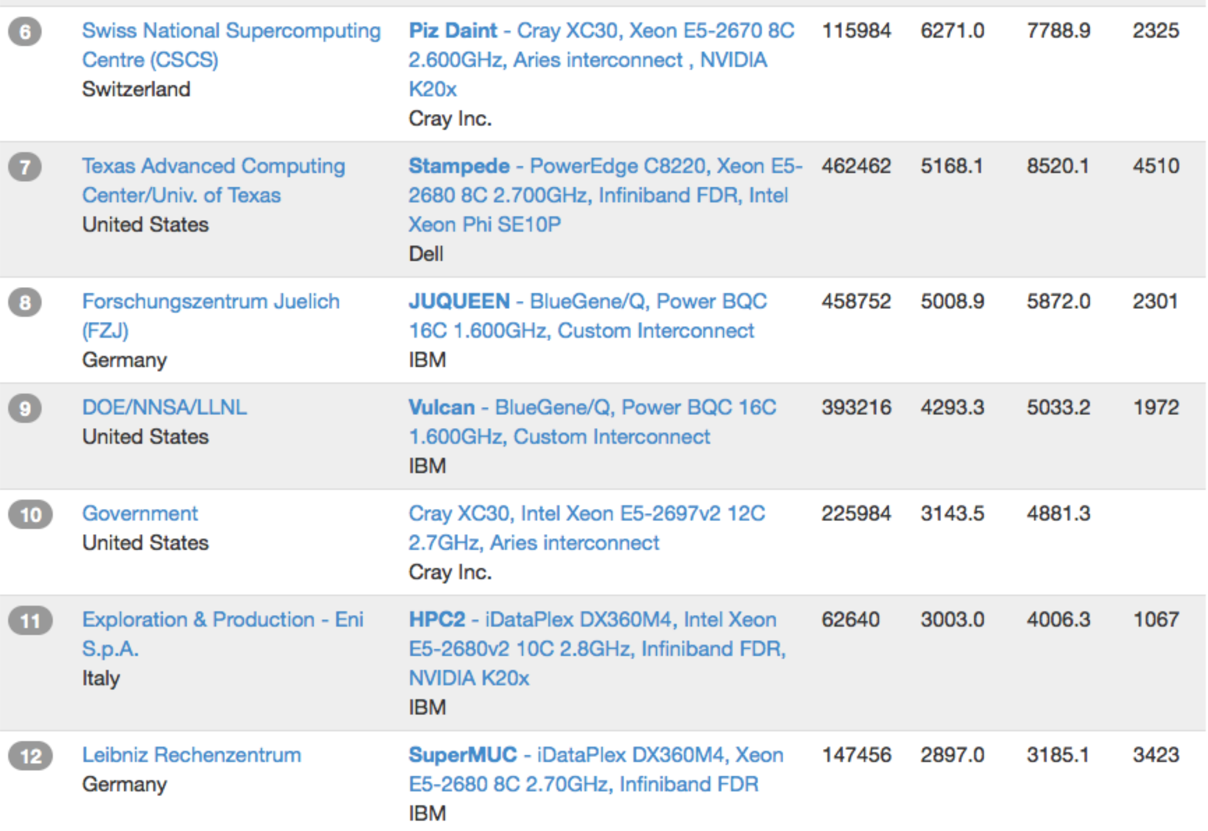
- Listing the 500 most powerful computers in the world
- Yardstick: Rmax of Linpack
  - Solve Ax=b, dense problem, matrix is random
  - Dominated by dense matrix-matrix multiply
- Update twice a year:
  - ISC'xy in June in Germany
  - SCxy in November in the U.S.
- All information available from the TOP500 web site at: www.top500.org

The TOP12 (June 2014)



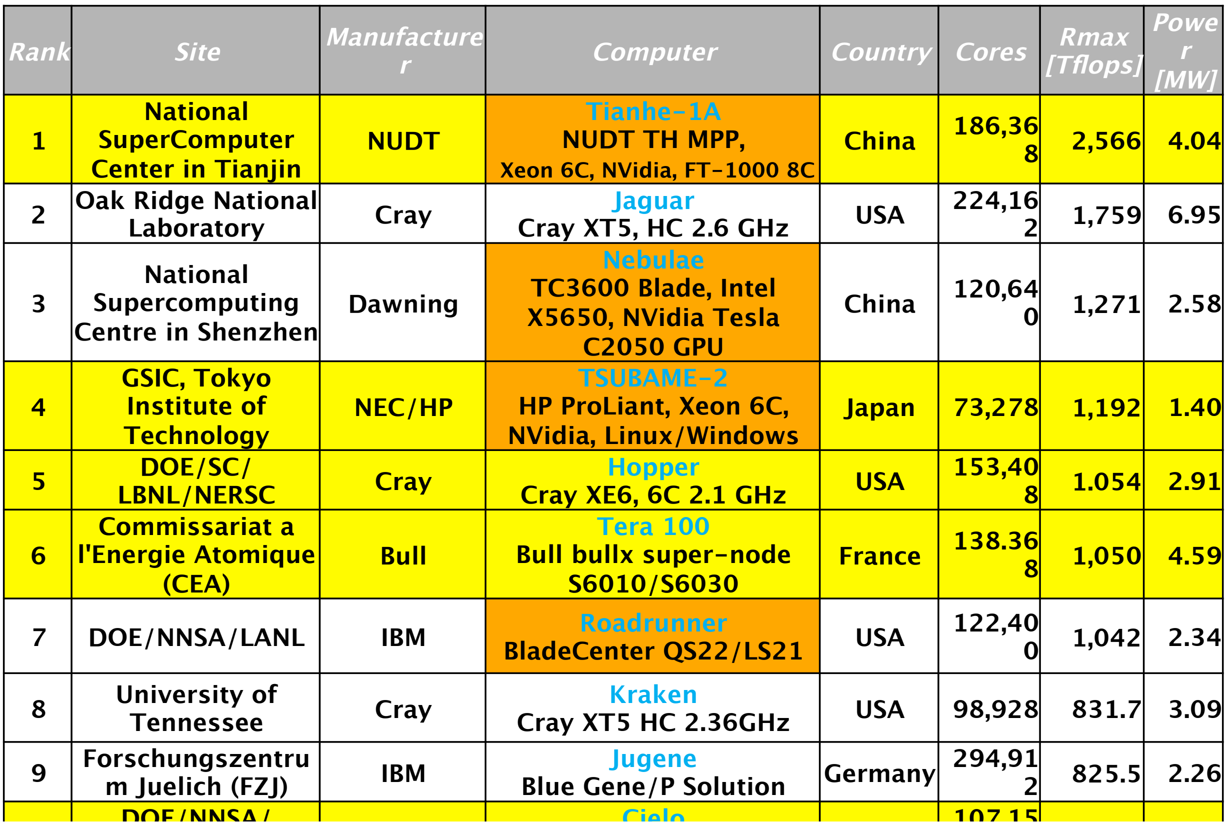
Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3120000	33862.7	54902.4	17808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560640	17590.0	27112.5	8209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1572864	17173.2	20132.7	7890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705024	10510.0	11280.4	12660
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786432	8586.6	10066.3	3945

The TOP12 (June 2014)



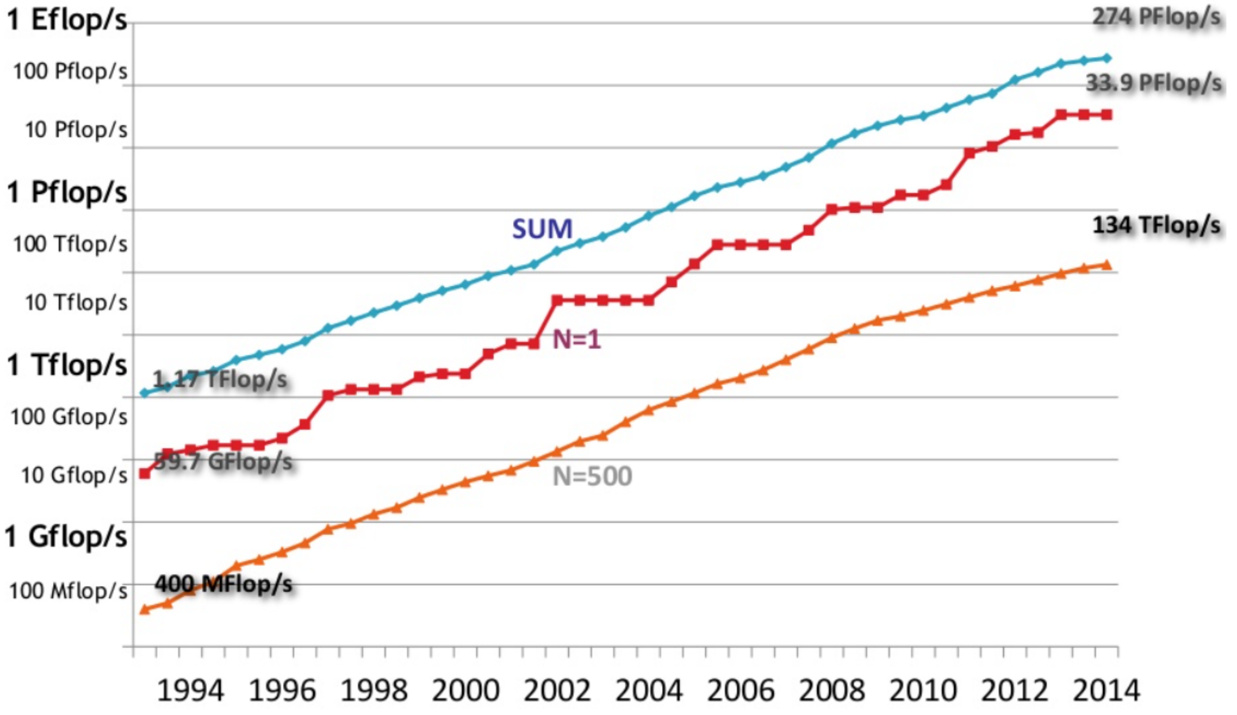
6	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115984	6271.0	7788.9	2325
7	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462462	5168.1	8520.1	4510
8	Forschungszentrum Juelich (FZJ) Germany	**JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458752	5008.9	5872.0	2301
9	DOE/NNSA/LLNL United States	**Vulcan** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393216	4293.3	5033.2	1972
10	Government United States	Cray XC30, Intel Xeon E5-2697v2 12C 2.7GHz, Aries interconnect Cray Inc.	225984	3143.5	4881.3	
11	Exploration & Production - Eni S.p.A. Italy	**HPC2** - iDataPlex DX360M4, Intel Xeon E5-2680v2 10C 2.8GHz, Infiniband FDR, NVIDIA K20x IBM	62640	3003.0	4006.3	1067
12	Leibniz Rechenzentrum Germany	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM	147456	2897.0	3185.1	3423

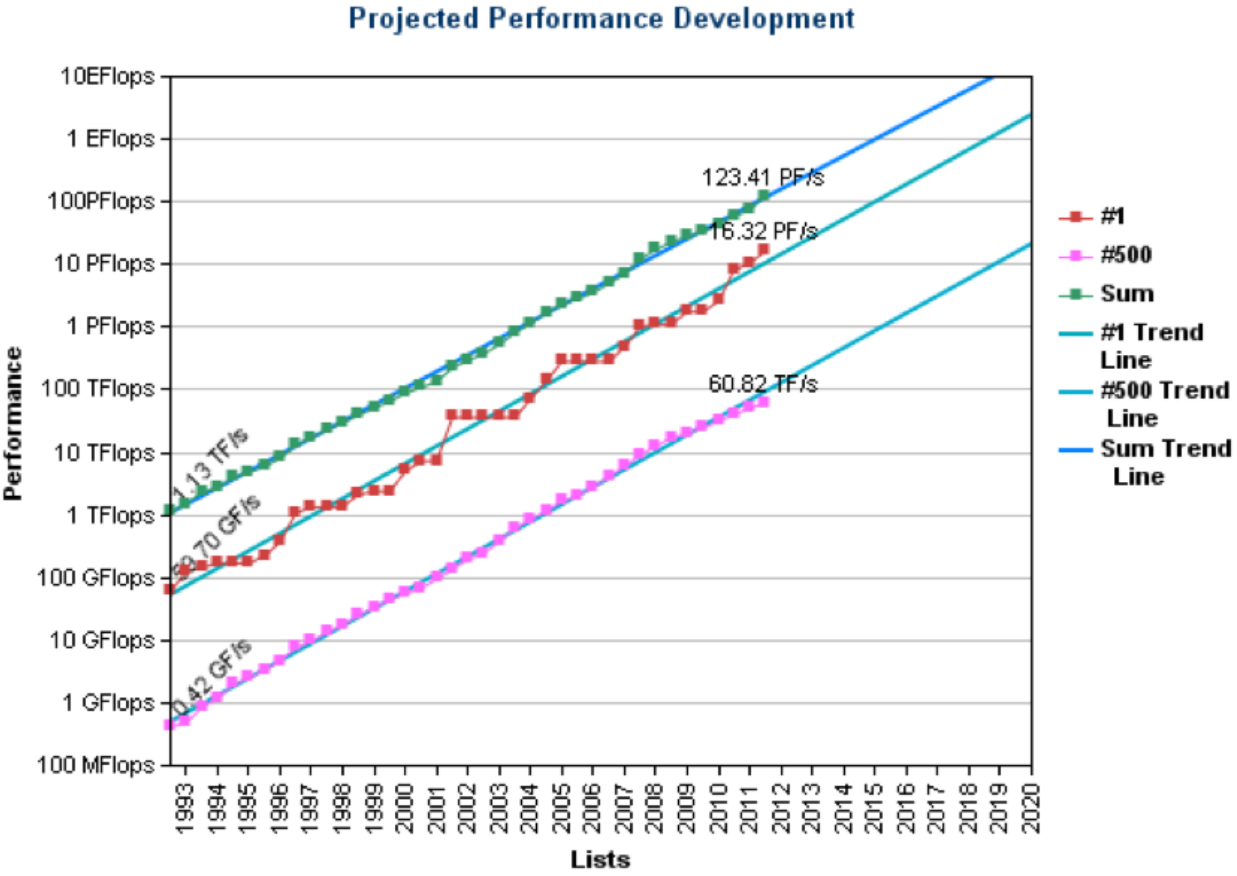
36th List: The TOP10 (Nov. 2010)



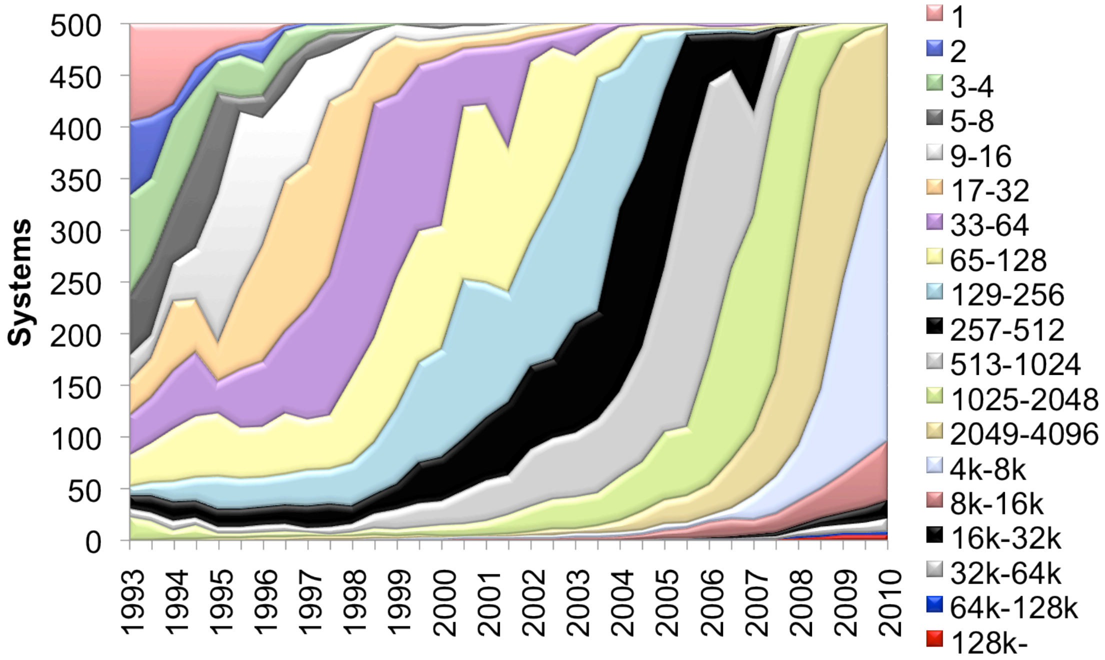
Rank	Site	Manufacturer	Computer	Country	Cores	Rmax [Tflops]	Power [MW]
1	National SuperComputer Center in Tianjin	NUDT	Tianhe-1A NUDT TH MPP, Xeon 6C, NVidia, FT-1000 8C	China	186,368	2,566	4.04
2	Oak Ridge National Laboratory	Cray	Jaguar Cray XT5, HC 2.6 GHz	USA	224,162	1,759	6.95
3	National Supercomputing Centre in Shenzhen	Dawning	Nebulae TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU	China	120,640	1,271	2.58
4	GSIC, Tokyo Institute of Technology	NEC/HP	TSUBAME-2 HP ProLiant, Xeon 6C, NVidia, Linux/Windows	Japan	73,278	1,192	1.40
5	DOE/SC/ LBNL/NERSC	Cray	Hopper Cray XE6, 6C 2.1 GHz	USA	153,408	1.054	2.91
6	Commissariat a l'Energie Atomique (CEA)	Bull	Tera 100 Bull bullx super-node S6010/S6030	France	138.368	1,050	4.59
7	DOE/NNSA/LANL	IBM	Roadrunner BladeCenter QS22/LS21	USA	122,400	1,042	2.34
8	University of Tennessee	Cray	Kraken Cray XT5 HC 2.36GHz	USA	98,928	831.7	3.09
9	Forschungszentrum Juelich (FZJ)	IBM	Jugene Blue Gene/P Solution	Germany	294,912	825.5	2.26
	DOE/NNSA/		Cielo		107,15		

Performance Development




Projected Performance Development
Performance
10EFlops
1 EFlops
100PFlops
10 PFlops
1 PFlops
100 TFlops
10 TFlops
1 TFlops
100 GFlops
10 GFlops
1 GFlops
100 MFlops
123.41 PF/s
16.32 PF/s
60.82 TF/s
1.13 TF/s
59.70 GF/s
0.42 GF/s
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
Lists
#1
#500
Sum
#1 Trend Line
#500 Trend Line
Sum Trend Line

Core Count


Systems
500
450
400
350
300
250
200
150
100
50
0
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
1
2
3-4
5-8
9-16
17-32
33-64
65-128
129-256
257-512
513-1024
1025-2048
2049-4096
4k-8k
8k-16k
16k-32k
32k-64k
64k-128k
128k-

Moore’s Law reinterpreted

- Number of cores per chip will double every two years
- Clock speed will not increase (possibly decrease)
- Need to deal with systems with millions of concurrent threads
- Need to deal with inter-chip parallelism as well as intra-chip parallelism

Outline

- Why ~~powerful~~ all computers must be parallel processors
  - Including your laptops and handhelds
- Large CSE problems require powerful computers
  - Commercial problems too
- Why writing (fast) parallel programs is hard
  - But things are improving
- Principles of parallel computing performance
- Structure of the course

Computational Science- Recent News

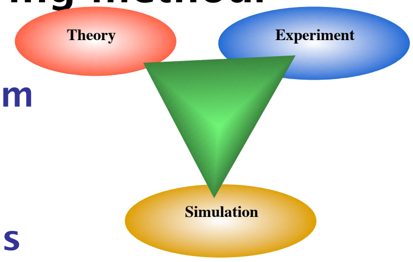
"An important development in sciences is occurring at the intersection of computer science and the sciences that has the potential to have a profound impact on science. It is a leap from the application of computing … to the *integration of computer science concepts, tools, and theorems* into the very fabric of science." - *Science* 2020 Report, March 2006

Nature, March 23, 2006

Drivers for Change

- Continued exponential increase in computational power → simulation is becoming third pillar of science, complementing theory and experiment
- Continued exponential increase in experimental data → techniques and technology in data analysis, visualization, analytics, networking, and collaboration tools are becoming essential in all data rich scientific applications

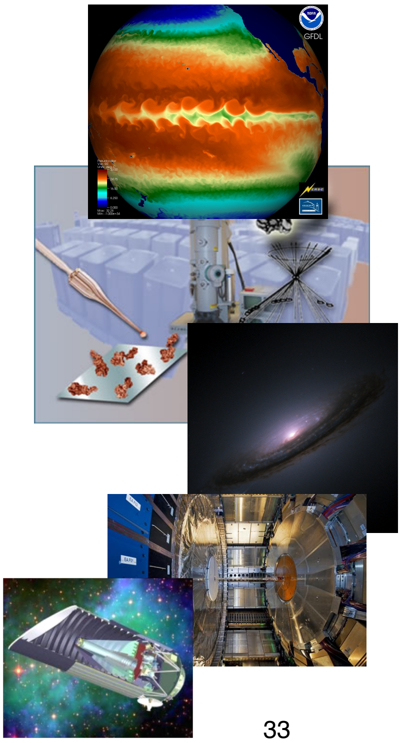
Simulation: The Third Pillar of Science



- Traditional scientific and engineering method:
  - (1) Do theory or paper design
  - (2) Perform experiments or build system
- Limitations:
  - –Too difficult—build large wind tunnels
  - –Too expensive—build a throw-away passenger jet
  - –Too slow—wait for climate or galactic evolution
  - –Too dangerous—weapons, drug design, climate experimentation
- Computational science and engineering paradigm:
  - (3) Use computers to simulate and analyze the phenomenon
  - –Based on known physical laws and efficient numerical methods
  - –Analyze simulation results with computational tools and methods beyond what is possible manually

Data Driven Science

- Scientific data sets are growing exponentially
  - – Ability to generate data is exceeding our ability to store and analyze
  - – Simulation systems and some observational devices grow in capability with Moore’s Law
- Petabyte (PB) data sets will soon be common:
  - – *Climate modeling:* estimates of the next IPCC data is in 10s of petabytes
  - – *Genome:* JGI alone will have .5 petabyte of data this year and double each year
  - – *Particle physics*: LHC is projected to produce 16 petabytes of data per year
  - – *Astrophysics*: LSST and others will produce 5 petabytes/year
- Create scientific communities with “Science Gateways” to data



Some Particularly Challenging Computations

- Science
  - Global climate modeling
  - Biology: genomics; protein folding; drug design
  - Astrophysical modeling
  - Computational Chemistry
  - Computational Material Sciences and Nanosciences
- Engineering
  - Semiconductor design
  - Earthquake and structural modeling
  - Computation fluid dynamics (airplane design)
  - Combustion (engine design)
  - Crash simulation
- Business
  - Financial and economic modeling
  - Transaction processing, web services and search engines
- Defense
  - Nuclear weapons -- test by simulations
  - Cryptography

Economic Impact of HPC

- Airlines:
  - System-wide logistics optimization systems on parallel systems.
  - Savings: approx. $100 million per airline per year.
- Automotive design:
  - Major automotive companies use large systems (500+ CPUs) for:
    - CAD-CAM, crash testing, structural integrity and aerodynamics
    - One company has 500+ CPU parallel system.
  - Savings: approx. $1 billion per company per year.
- Semiconductor industry:
  - Semiconductor firms use large systems (500+ CPUs) for
    - device electronics simulation and logic validation
  - Savings: approx. $1 billion per company per year.
- Energy
  - Computational modeling improved performance of current nuclear power plants, equivalent to building two new power plants.

What Supercomputers Do

Introducing Computational Science and Engineering

Two Examples

- simulation replacing experiment that is too slow
- analyzing massive amounts of data with new tools

Global Climate Modeling Problem

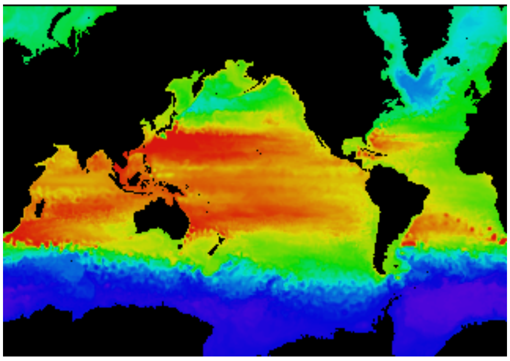
- Problem is to compute:

  f(latitude, longitude, elevation, time) -> "weather" =

  (temperature, pressure, humidity, wind velocity)

- Approach:
  - *Discretize* the domain, e.g., a measurement point every 10 km
  - Devise an algorithm to predict weather at time t+δt given t

- Uses:
  - Predict major events, e.g., El Nino
  - Use in setting air emissions standards
  - Evaluate global warming scenarios

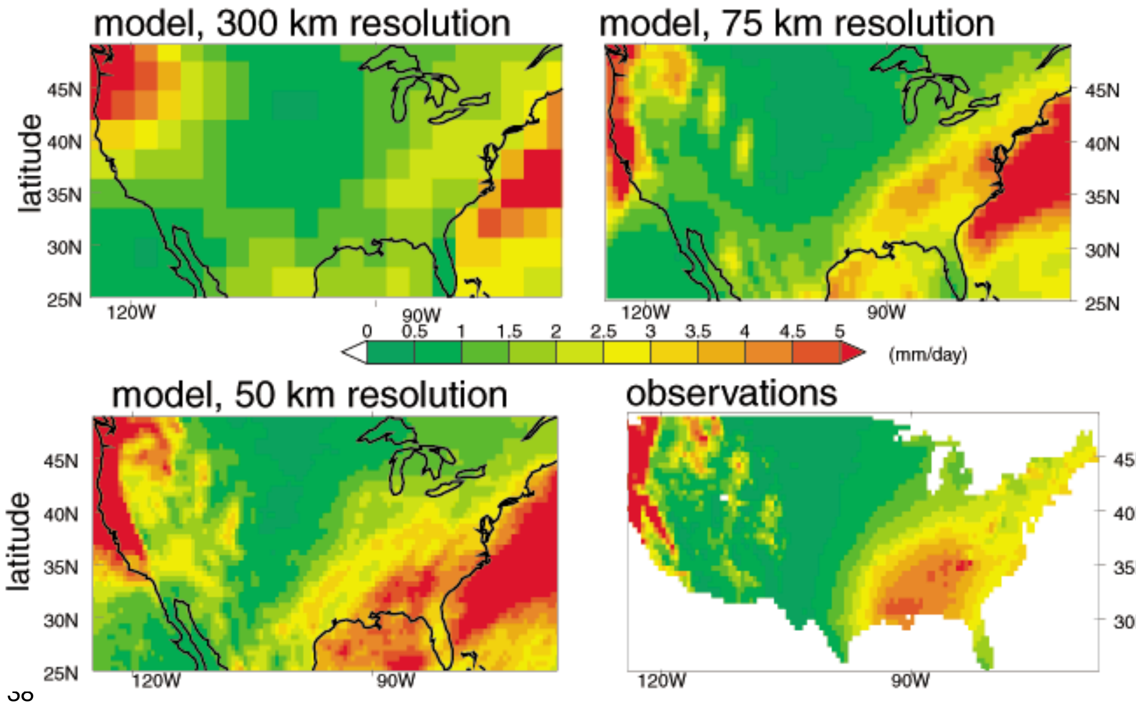


Source: http://www.epm.ornl.gov/chammp/chammp.html

High Resolution Climate Modeling on NERSC-3 – P. Duffy, et al., LLNL

Wintertime Precipitation

As model resolution becomes finer, results converge towards observations



Example : weather prediction (Europe)

- Navier-Stokes equations (PDE) : discretized on a grid
- **Assume** domain : 3000 km x 3000 km x 10 km

  resolution: 1 km x 1km x 0.1 km

  -> grid of size: 3000 x 3000 x 100 = ± 10^9 grid points

  time interval: 48 h. ; time step: 3 min. -> ± 1000 timesteps

  cost per gid point : 1000 flop (flop = floating point operation)

  -> **Total cost:** 10^9 x 1000 x 1000 = 10^{15} flop = 1 Pflop

- PC or workstation (1 Gflops) : 300 hours

  Cluster (e.g. ThinKing) (> 1 Tflops) : < 20 min

- **Required memory :**

  solution only: 10^9 grid points x 5 variables x 8 bytes = 4 10^{10} bytes

  = 40 Gbyte ! --> total required memory: ± 400 Gbyte

Which commercial applications *require* parallelism?

Analyzed in detail in
“Berkeley View” report

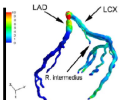
	Embed	SPEC	DB	Games	ML	HPC
1 Finite State Mach.	🟥	🟥	🟥	🟨	🟨	🟦
2 Combinational	🟥	🟦	🟩	🟦	🟩	🟦
3 Graph Traversal	🟥	🟨	🟨	🟨	🟥	🟦
4 Structured Grid	🟥	🟥	🟦	🟨	🟦	🟥
5 Dense Matrix	🟥	🟥	🟨	🟥	🟥	🟥
6 Sparse Matrix	🟨	🟨	🟦	🟥	🟥	🟥
7 Spectral (FFT)	🟨	🟦	🟦	🟨	🟨	🟥
8 Dynamic Prog	🟨	🟦	🟥	🟦	🟥	🟦
9 N-Body	🟦	🟨	🟦	🟨	🟦	🟥
10 MapReduce	🟦	🟩	🟥	🟦	🟥	🟥
11 Backtrack/ B&B	🟦	🟦	🟨	🟦	🟥	🟦
12 Graphical Models	🟦	🟦	🟨	🟦	🟥	🟦
13 Unstructured Grid	🟦	🟦	🟦	🟨	🟨	🟥

Analyzed in detail in
“Berkeley View” report
www.eecs.berkeley.edu/
Pubs/TechRpts/2006/
EECS-2006-183.html

What do commercial and CSE applications have in commc

Motif/Dwarf: Common Computational Methods

(Red Hot → Blue Cool)

	Embed	SPEC	DB	Games	ML	HPC	 Health	Image	Speech	Music	Browser
1 Finite State Mach.	🟥	🟥	🟥	🟨	🟨	🟦	🟦	🟦	🟦	🟦	🟥
2 Combinational	🟥	🟦	🟩	🟦	🟩	🟦	🟦	🟦	🟦	🟦	🟥
3 Graph Traversal	🟥	🟨	🟨	🟨	🟥	🟦	🟥	🟦	🟥	🟩	🟩
4 Structured Grid	🟥	🟥	🟦	🟨	🟦	🟥	🟦	🟥	🟦	🟦	🟦
5 Dense Matrix	🟥	🟥	🟨	🟥	🟥	🟥	🟦	🟥	🟥	🟥	🟦
6 Sparse Matrix	🟨	🟨	🟦	🟥	🟥	🟥	🟥	🟦	🟦	🟥	🟦
7 Spectral (FFT)	🟨	🟦	🟦	🟨	🟨	🟥	🟦	🟩	🟥	🟥	🟥
8 Dynamic Prog	🟨	🟦	🟥	🟦	🟥	🟦	🟦	🟦	🟨	🟦	🟥
9 N-Body	🟦	🟨	🟦	🟨	🟦	🟥	🟩	🟦	🟦	🟦	🟦
10 MapReduce	🟦	🟩	🟥	🟦	🟥	🟥	🟥	🟥	🟨	🟥	🟨
11 Backtrack/ B&B	🟦	🟦	🟨	🟦	🟥	🟦	🟦	🟦	🟦	🟨	🟦
12 Graphical Models	🟦	🟦	🟨	🟦	🟥	🟦	🟦	🟦	🟦	🟥	🟦
13 Unstructured Grid	🟦	🟦	🟦	🟨	🟨	🟥	🟥	🟦	🟦	🟥	🟦

Outline

- Why ~~powerful~~ all computers must be parallel processors
  - Including your laptops and handhelds
- Large CSE problems require powerful computers
  - Commercial problems too
- Why writing (fast) parallel programs is hard
  - But things are improving
- Principles of parallel computing performance
- Structure of the course

Principles of Parallel Computing

- Finding enough parallelism (Amdahl’s Law)
- Granularity
- Locality
- Load balance
- Coordination and synchronization
- Performance modeling

➡ All of these things makes parallel programming even harder than sequential programming.

“Automatic” Parallelism in Modern Machines

- Bit level parallelism
  - within floating point operations, etc.
- Instruction level parallelism (ILP)
  - multiple instructions execute per clock cycle
- Memory system parallelism
  - overlap of memory operations with computation
- OS parallelism
  - multiple jobs run in parallel on commodity SMPs

Limits to all of these -- for very high performance, need user to identify, schedule and coordinate parallel tasks

Finding Enough Parallelism

- Suppose only part of an application seems parallel
- Amdahl’s law
  - let s be the fraction of work done sequentially, so (1-s) is the fraction that is parallelizable
  - P = number of processors

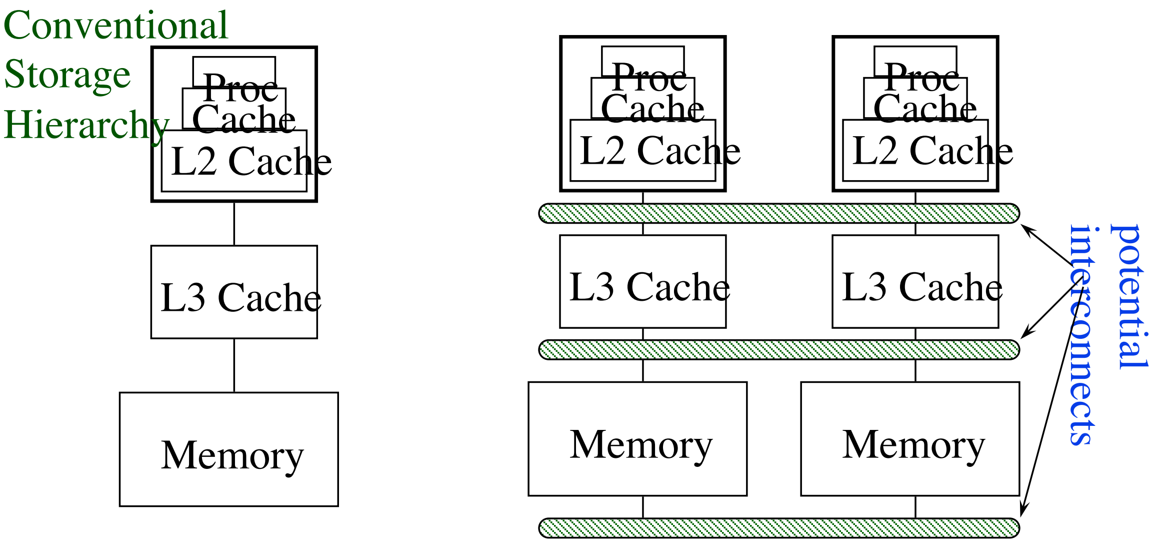
$$\begin{aligned} \text{Speedup}(P) &= \text{Time}(1)/\text{Time}(P) \\ &<= 1/(s + (1-s)/P) \\ &<= 1/s \end{aligned}$$

- Even if the parallel part speeds up perfectly performance is limited by the sequential part
- Top500 list: currently 2nd fastest machine has P~224K; fastest has ~186K+GPUs

Overhead of Parallelism

- Given enough parallel work, this is the biggest barrier to getting desired speedup
- Parallelism overheads include:
  - cost of starting a thread or process
  - cost of communicating shared data
  - cost of synchronizing
  - extra (redundant) computation
- Each of these can be in the range of milliseconds (=millions of flops) on some systems
- Tradeoff: Algorithm needs sufficiently large units of work to run fast in parallel (i.e. large granularity), but not so large that there is not enough parallel work

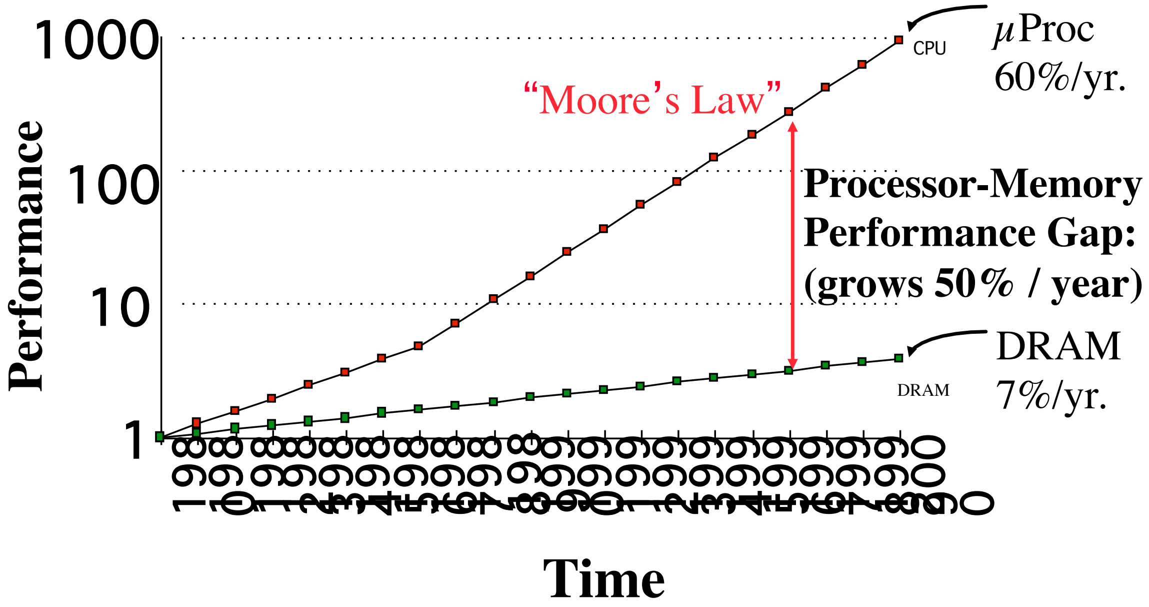
Locality and Parallelism



- Large memories are slow, fast memories are small
- Storage hierarchies are large and fast <u>on average</u>
- Parallel processors, collectively, have large, fast cache
  - the slow accesses to “remote” data we call “communication”
- Algorithm should do most work on local data

Processor-DRAM Gap (latency)

Goal: find algorithms that minimize communication, not necessarily arithmetic



Load Imbalance

- Load imbalance is the time that some processors in the system are idle due to
  - insufficient parallelism (during that phase)
  - unequal size tasks
- Examples of the latter
  - adapting to "interesting parts of a domain"
  - tree-structured computations
  - fundamentally unstructured problems
- Algorithm needs to balance load
  - Sometimes can determine work load, divide up evenly, before starting
    - "Static Load Balancing"
  - Sometimes work load changes dynamically, need to rebalance dynamically
    - "Dynamic Load Balancing"

Parallel Software Eventually – ParLab view

- 2 types of programmers -> 2 layers
- **Efficiency Layer** (10% of today's programmers)
  - Expert programmers build Libraries implementing motifs, "Frameworks", OS, ….
  - Highest fraction of peak performance possible
- **Productivity Layer** (90% of today's programmers)
  - Domain experts / Naïve programmers productively build parallel applications by composing frameworks & libraries
  - Hide as many details of machine, parallelism as possible
  - Willing to sacrifice some performance for productive programming
- Expect students may want to work at either level
  - In the meantime, we all need to understand enough of the efficiency layer to use parallelism effectively

Outline

- Why ~~powerful~~ all computers must be parallel processors
  - Including your laptops and handhelds
- Large CSE problems require powerful computers
  - Commercial problems too
- Why writing (fast) parallel programs is hard
  - But things are improving
- Principles of parallel computing performance
- Structure of the course

Improving Real Performance

Peak Performance grows exponentially, a la Moore's Law

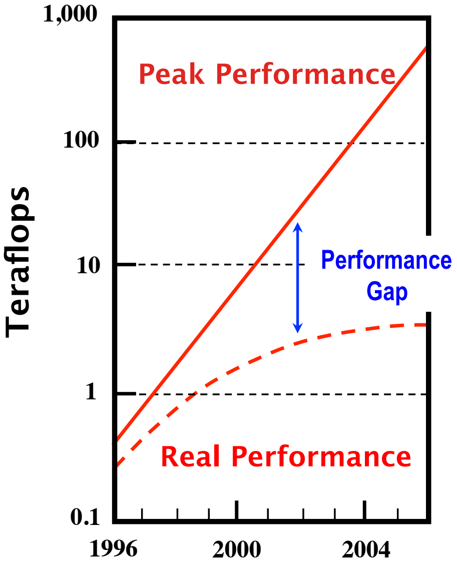
- In 1990's, peak performance increased 100x; in 2000's, it will increase 1000x

But efficiency (the performance relative to the hardware peak) has declined

- was 40-50% on the vector supercomputers of 1990s
- now as little as 5-10% on parallel supercomputers of today

Close the gap through ...

- Mathematical methods and algorithms that achieve high performance on a single processor and scale to thousands of processors
- More efficient programming models and tools for massively parallel supercomputers



Performance Levels

- Peak performance
  - Sum of all speeds of all floating point units in the system
  - You can't possibly compute faster than this speed
- LINPACK
  - The "hello world" program for parallel performance
  - Solve Ax=b using Gaussian Elimination, highly tuned
- Gordon Bell Prize winning applications performance
  - The right application/algorithm/platform combination plus years of work
- Average sustained applications performance
  - What one reasonable can expect for standard applications

When reporting performance results, these levels are often confused, even in reviewed publications

Performance Levels (for example on NERSC-5)

- Peak advertised performance (PAP): 100 Tflop/s
- LINPACK (TPP): 84 Tflop/s
- Best climate application: 14 Tflop/s
  - WRF code benchmarked in December 2007
- Average sustained applications performance: ? Tflop/s
  - Probably less than 10% peak!
- We will study performance
  - Hardware and software tools to measure it
  - Identifying bottlenecks
  - Practical performance tuning (Matlab demo)

Parallel Computing in Data Analysis

- **Finding information amidst large quantities of data**
- **General themes of sifting through large, unstructured data sets:**
  - **Has there been an outbreak of some medical condition in a community?**
  - **bio-informatics**
  - **...**
- **Data collected and stored at enormous speeds (Gbyte/hour)**
  - **telescope scanning the skies**
  - **microarrays generating gene expression data**
  - **...**