CS240A: Applied Parallel Computing

Introduction

CS 240A Course Information

- Web page:

  http://www.cs.ucsb.edu/~tyang/class/240a13w

- Class schedule: Mon/Wed. 11:00AM-12:50pm Phelp 2510
- Instructor: Tao Yang (tyang at cs).
  - Office Hours: MW 10-11(or email me for appointments or just stop by my office). HFH building, Room 5113
- Supercomputing consultant: Kadir Diri and Stefan Boeriu
- TA:
  - Wei Zhang (wei at cs). Office hours: Monday/Wed 3:30PM - 4:30PM
- Class materials:
  - Slides/handouts. Research papers. Online references.
- Slide source (HPC part) and related courses:
  - Demmel/Yelick's CS267 parallel computing at UC Berkeley
  - John Gilbert's CS240A at UCSB

Topics

- High performance computing
  - Basics of computer architecture, memory hierarchies, storage, clusters, cloud systems.
  - High throughput computing
- Parallel Programming Models and Machines. Software/libraries
  - Shared memory vs distributed memory
  - Threads, OpenMP, MPI, MapReduce, GPU if time permits
- Patterns of parallelism. Optimization techniques for parallelization and performance
- Core algorithms in Scientific Computing and Applications
  - Dense & Sparse Linear Algebra
- Parallelism in data-intensive web applications and storage systems

What you should get out of the course

In depth understanding of:

- When is parallel computing useful?
- Understanding of parallel computing hardware options.
- Overview of programming models (software) and tools.
- Some important parallel applications and the algorithms
- Performance analysis and tuning
- Exposure to various open research questions

Introduction: Outline

- Why ~~powerful~~ *all* computers must be parallel computing

  Including your laptops and handhelds

- Why parallel processing?
  - Large Computational Science and Engineering (CSE) problems require powerful computers
  - Commercial data-oriented computing also needs.
- Basic parallel performance models
- Why writing (fast) parallel programs is hard

Metrics in Scientific Computing Worlds

- **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…**

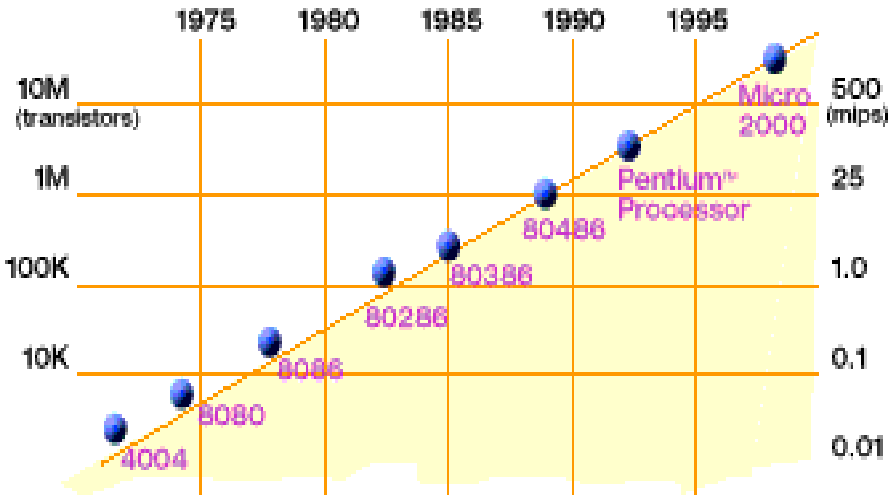
Mega	Mflop/s = 10^6 flop/sec	Mbyte = 2^{20} = 1048576 ~ 10^6 bytes
Giga	Gflop/s = 10^9 flop/sec	Gbyte = 2^{30} ~ 10^9 bytes
Tera	Tflop/s = 10^{12} flop/sec	Tbyte = 2^{40} ~ 10^{12} bytes
Peta	Pflop/s = 10^{15} flop/sec	Pbyte = 2^{50} ~ 10^{15} bytes
Exa	Eflop/s = 10^{18} flop/sec	Ebyte = 2^{60} ~ 10^{18} 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

- **Current fastest (public) machine ~ 27 Pflop/s**
  - Up-to-date list at www.top500.org

From www.top500.org

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	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
2	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1572864	16324.8	20132.7	7890

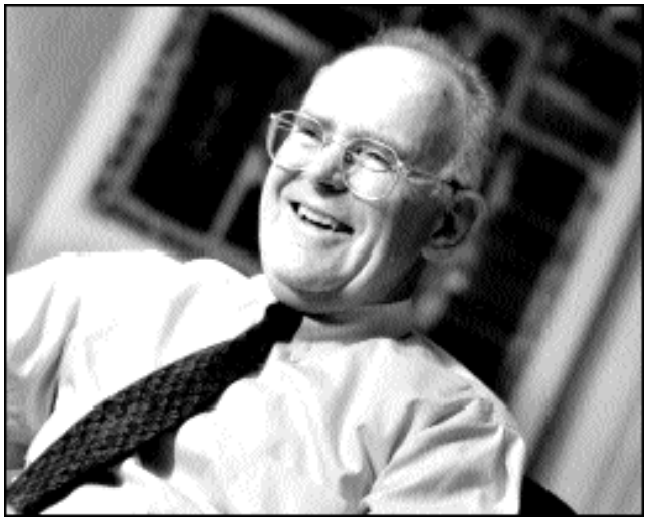
Technology Trends: Microprocessor Capacity



2X transistors/Chip Every 1.5 years

Called "Moore's Law"

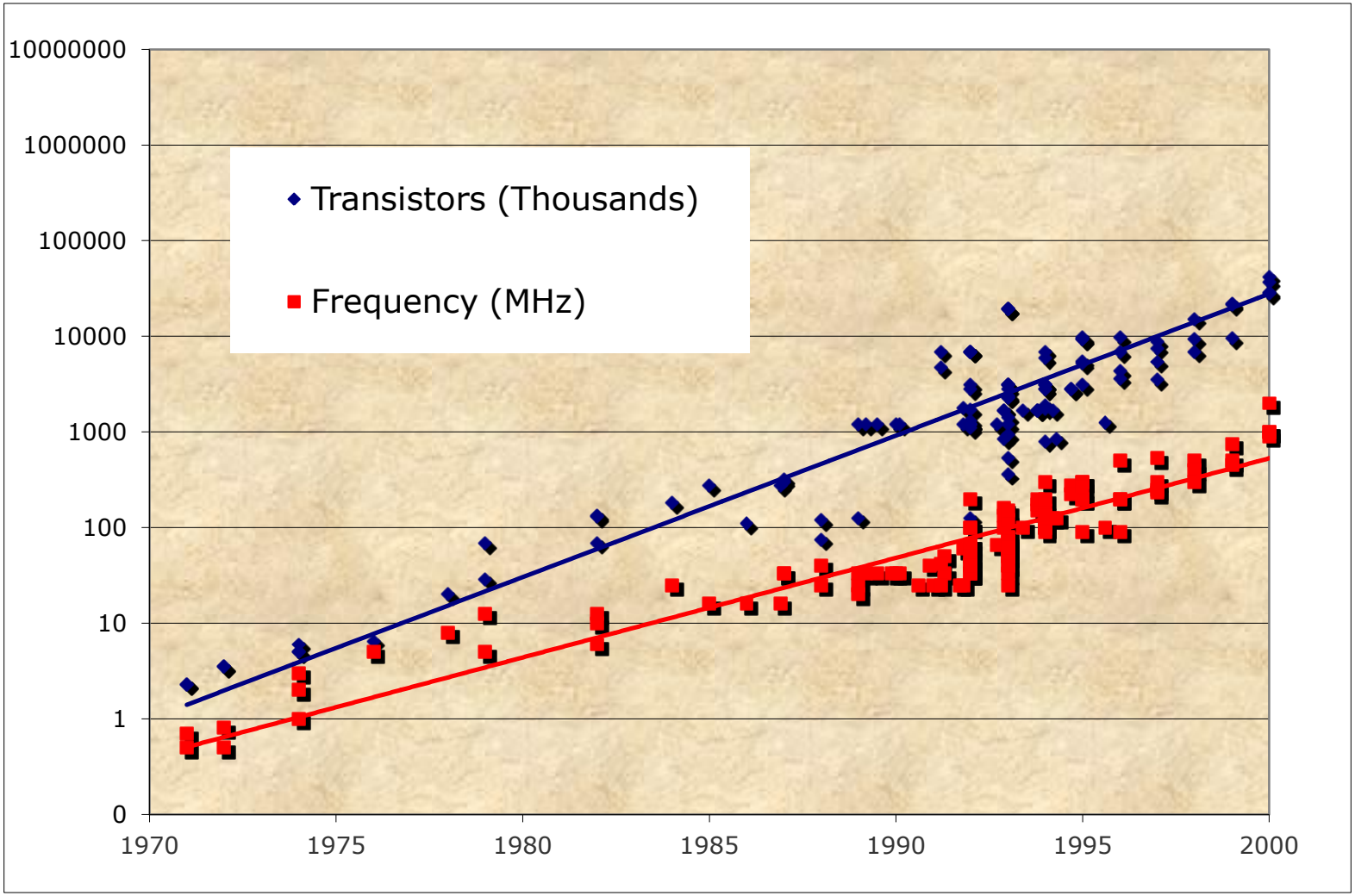
Microprocessors have become smaller, denser, and more powerful.



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)



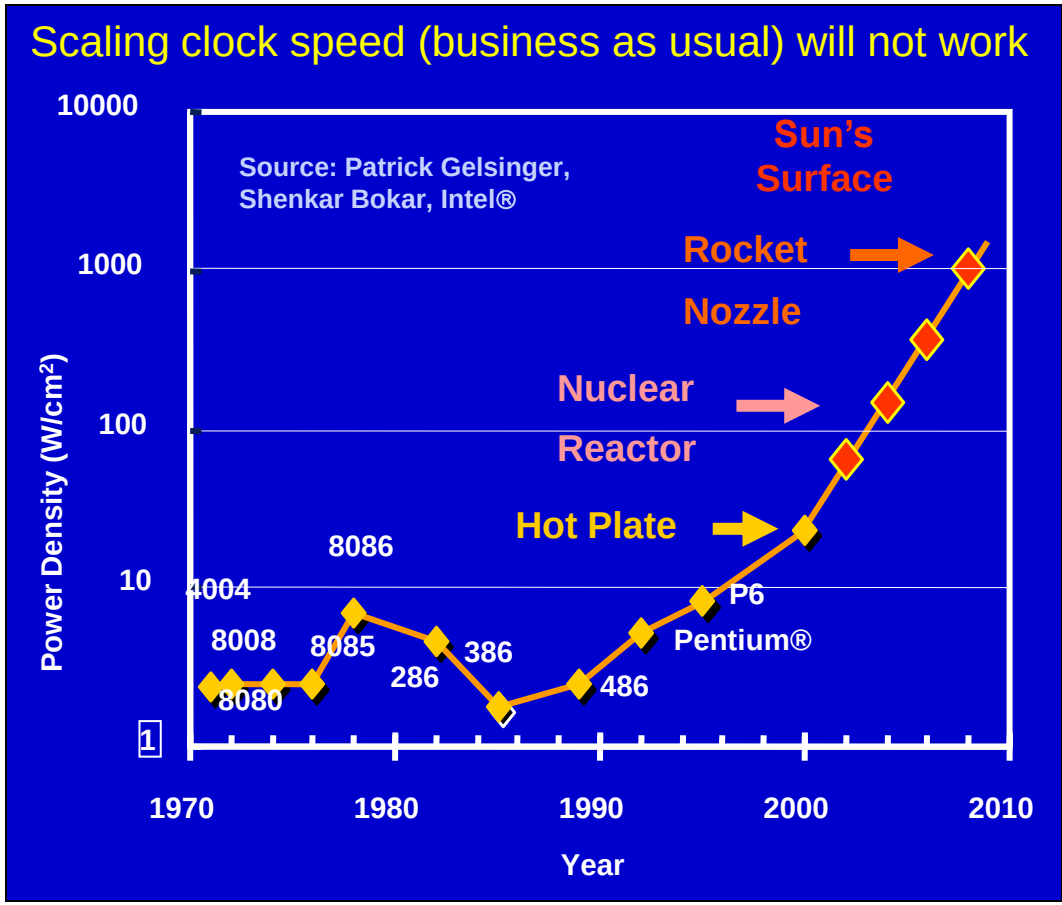
Impact of Device Shrinkage

- What happens when the feature size (transistor size) shrinks by a factor of x ?
  - Clock rate goes up by x or less because wires are shorter
- Transistors per unit area goes up by x^2
  - For on-chip parallelism (ILP) and locality: caches
  - More applications go faster without any change
- But manufacturing costs and yield problems limit use of density
  - What percentage of the chips are usable?

& More power consumption

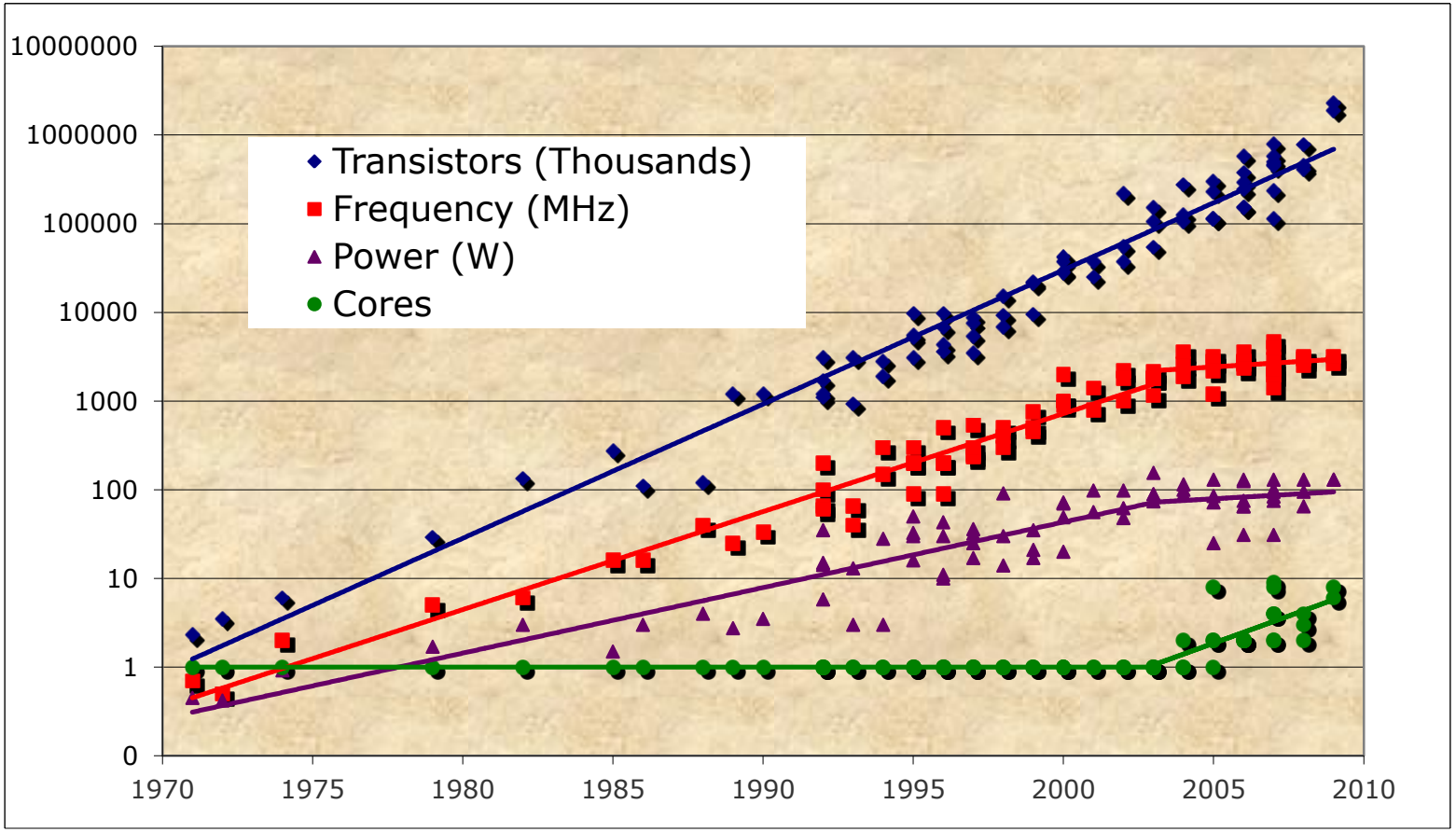
Power Density Limits Serial Performance

- Concurrent systems are more power efficient
  - Dynamic power is proportional to V^2fC
  - Increasing frequency (f) also increases supply voltage (V) → cubic effect
  - Increasing cores increases capacitance (C) but only linearly
  - Save power by lowering clock speed



- High performance serial processors waste power
  - Speculation, dynamic dependence checking, etc. burn power
  - Implicit parallelism discovery
- More transistors, but not faster serial processors

Revolution in Processors



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

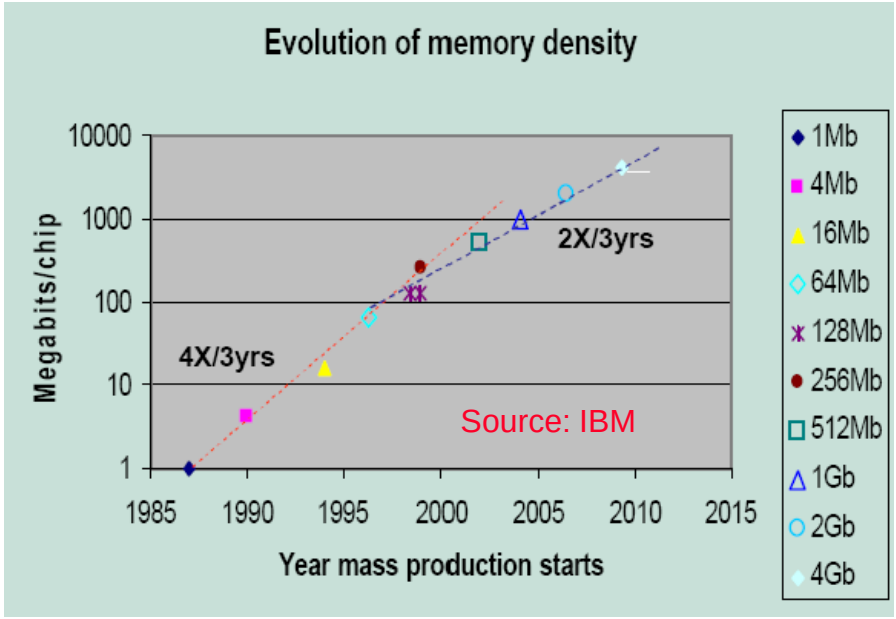
Impact of Parallelism

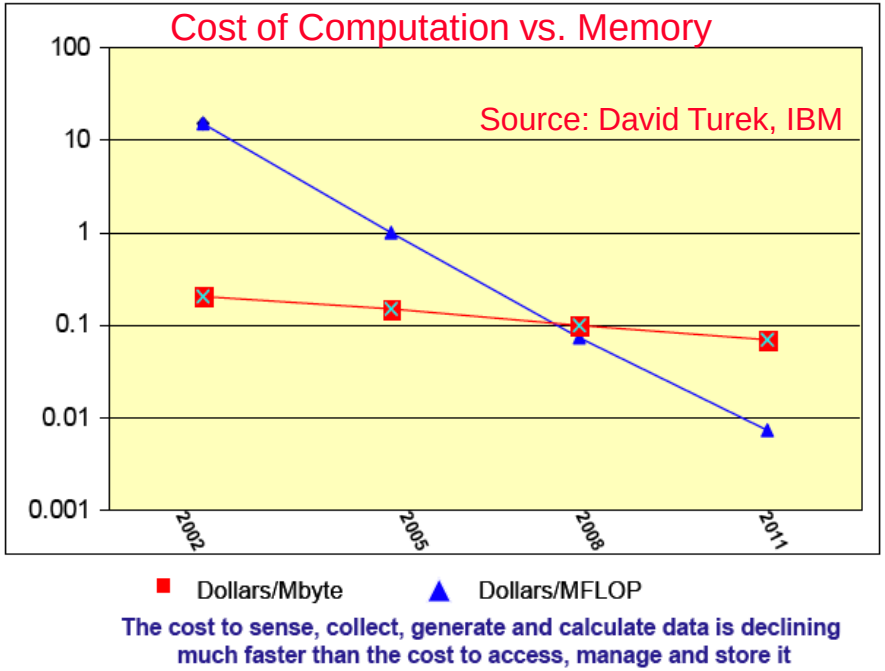
- All major processor vendors are producing *multicore* chips
  - Every machine will soon be a parallel machine
  - To keep doubling performance, parallelism must double
- Which commercial applications can use this parallelism?
  - Do they have to be rewritten from scratch?
- Will all programmers have to be parallel programmers?
  - New software model needed
  - Try to hide complexity from most programmers – eventually
- Computer industry betting on this big change, but does not have all the answers

Memory is Not Keeping Pace

Technology trends against a 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



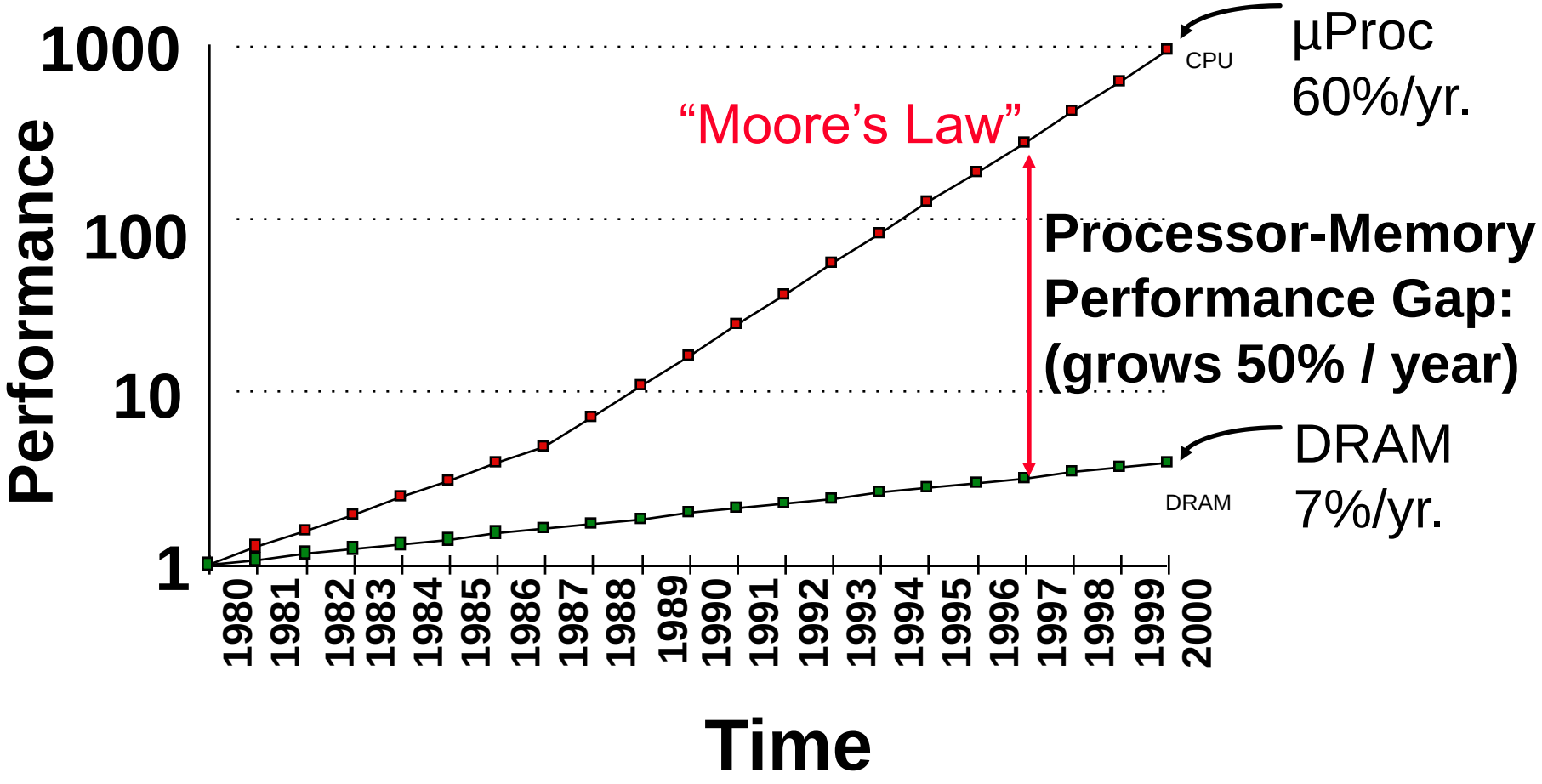


Question: Can you double concurrency without doubling memory?

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

Processor-DRAM Gap (latency)

Goal: find algorithms that minimize communication, not necessarily arithmetic



The TOP500 Project

- Listing the 500 most powerful computers in the world
- Linpack performance
  - 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

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 Computational Science&Engineering and commercial problems require powerful computers

- Basic performance models
- Why writing (fast) parallel programs is hard

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.

Drivers for Changes in Computational Science

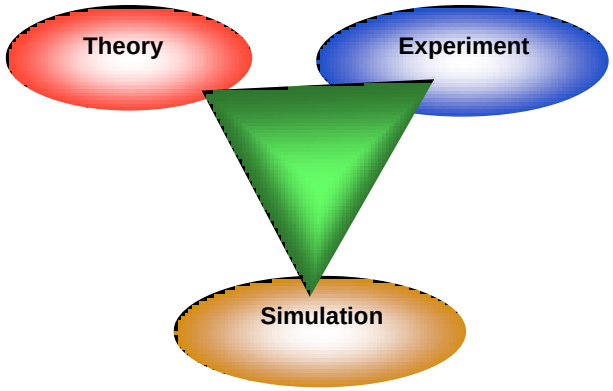
"An important development in sciences is occurring at the intersection of computer science and the sciences that has the potential to have a profound impact on science." - *Science* 2020 Report, March 2006

Nature, March 23, 2006

- **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

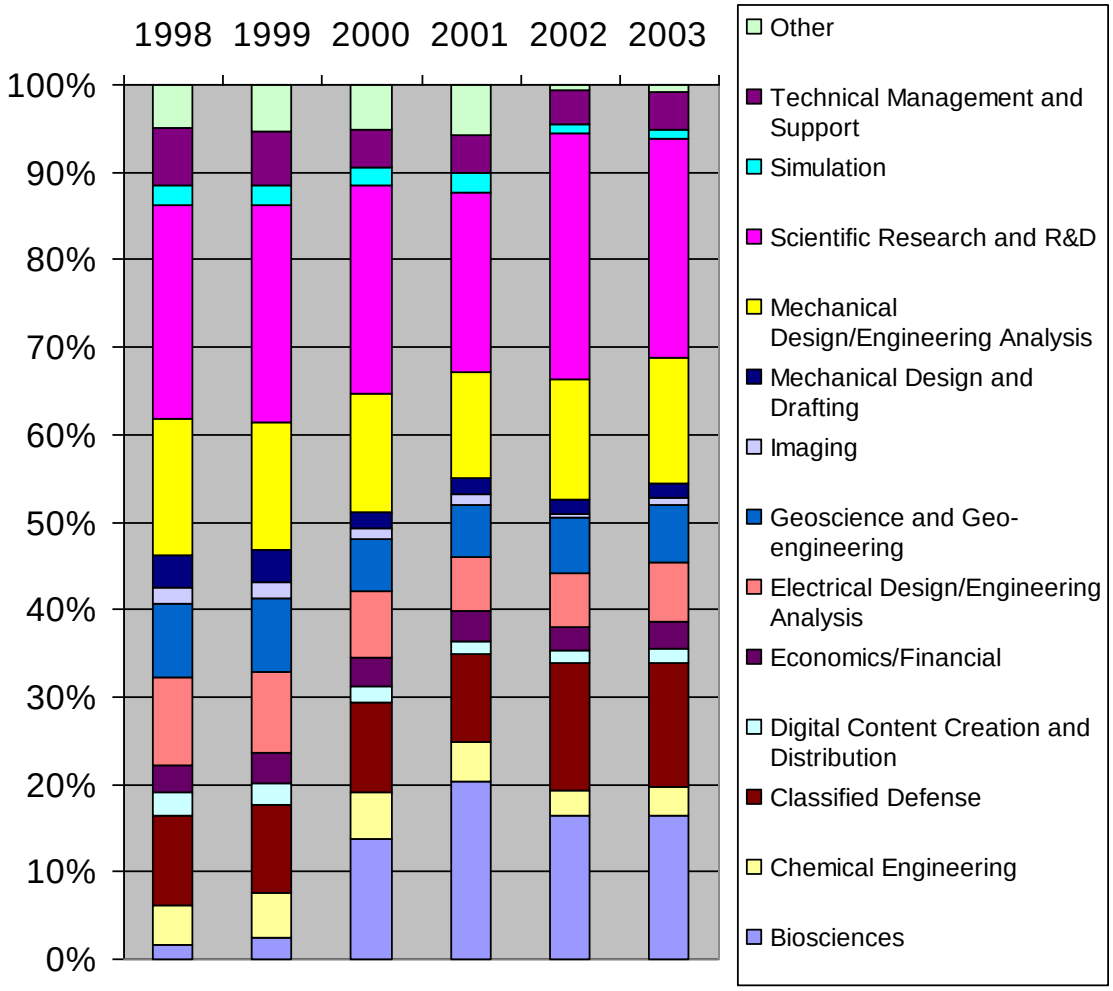
- **Traditional scientific and engineering method:**
  - (1) Do theory or paper design
  - (2) Perform experiments or build system



- **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

$5B World Market in Technical Computing



Source: IDC 2004, from NRC Future of Supercomputing Report

Global Climate Modeling Problem

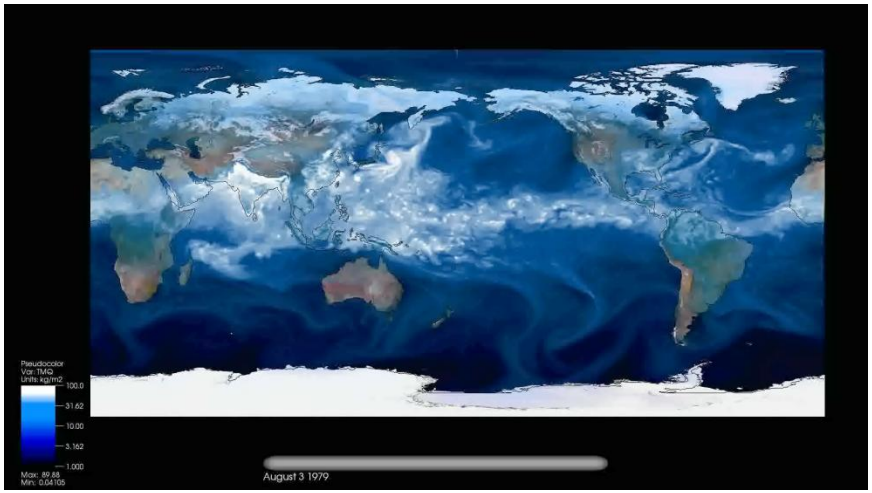
- Problem is to compute:

  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., hurricane, El Nino
  - Use in setting air emissions standards
  - Evaluate global warming scenarios



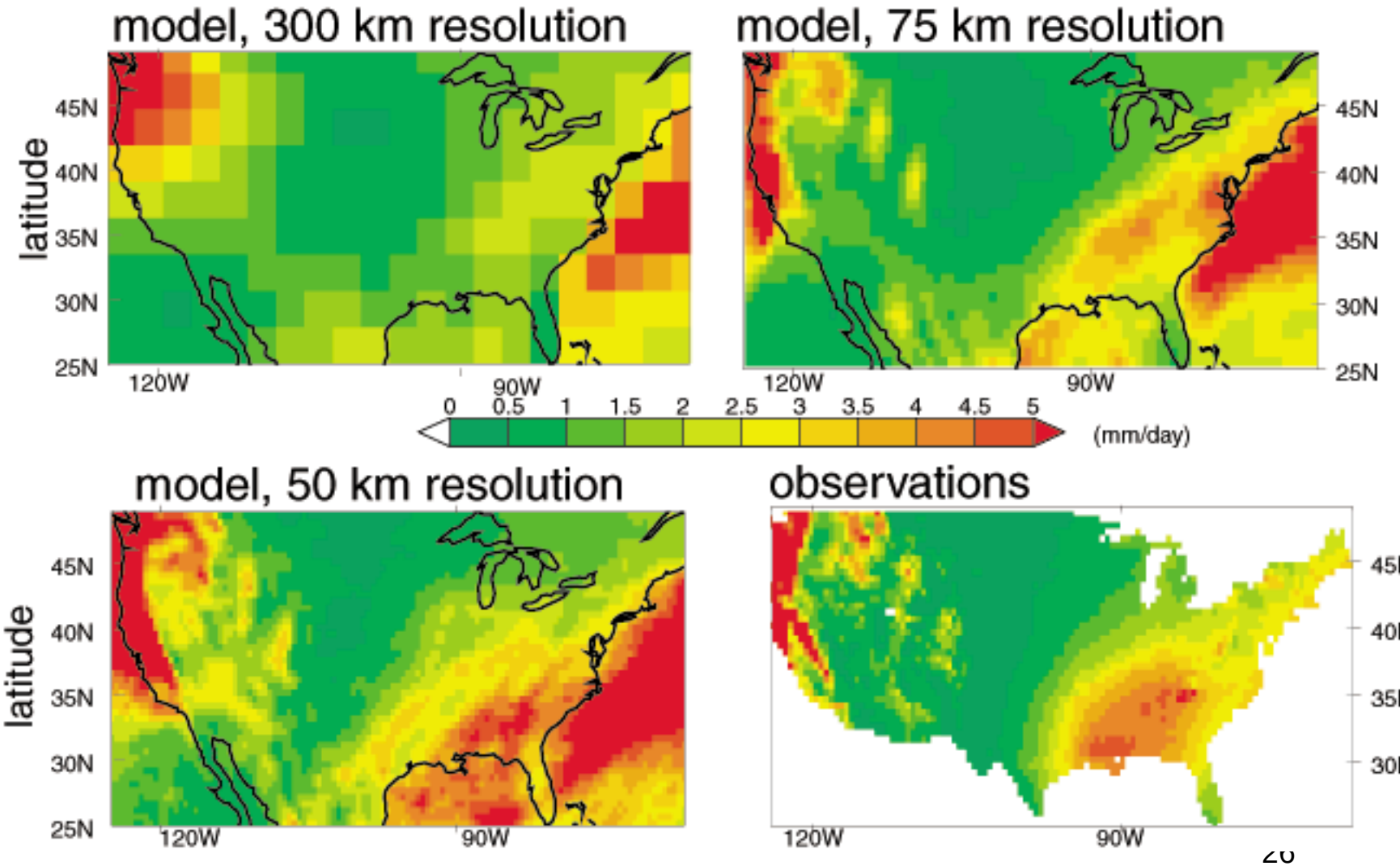
Global Climate Modeling Computation

- One piece is modeling the fluid flow in the atmosphere
  - Solve Navier-Stokes equations
    - Roughly 100 Flops per grid point with 1 minute timestep
- Computational requirements:
  - To match real-time, need 5×10^{11} flops in 60 seconds = 8 Gflop/s
  - Weather prediction (7 days in 24 hours) → 56 Gflop/s
  - Climate prediction (50 years in 30 days) → 4.8 Tflop/s
  - To use in policy negotiations (50 years in 12 hours) → 288 Tflop/s
- To double the grid resolution, computation is 8x to 16x
- State of the art models require integration of atmosphere, clouds, ocean, sea-ice, land models, plus possibly carbon cycle, geochemistry and more
- Current models are coarser than this

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

Wintertime Precipitation

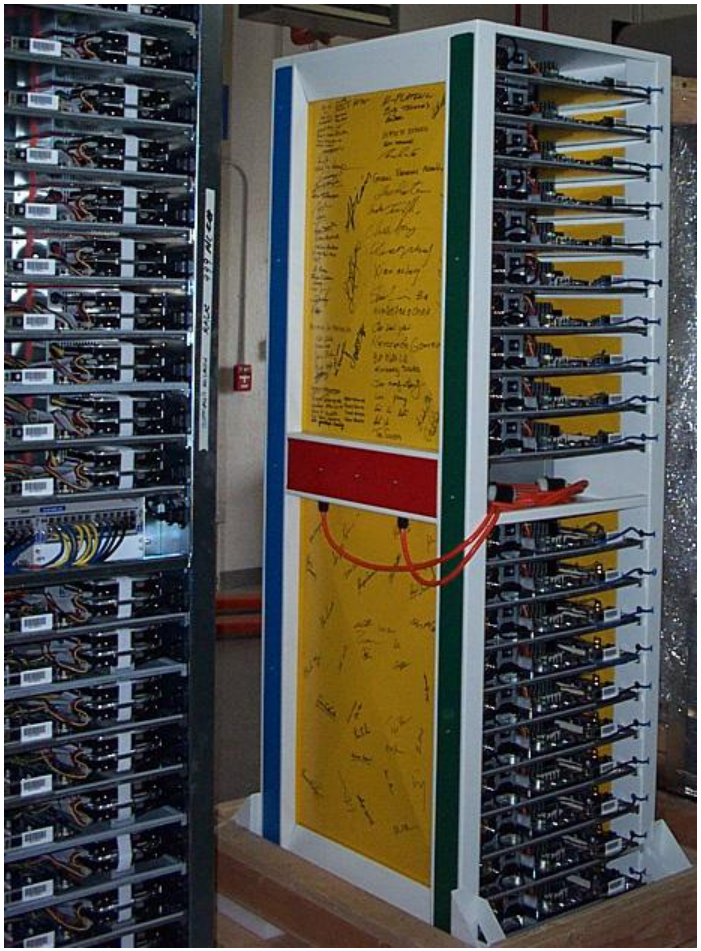
As model resolution becomes finer, results converge towards observations



Scalable Web Service/Processing Infrastructure

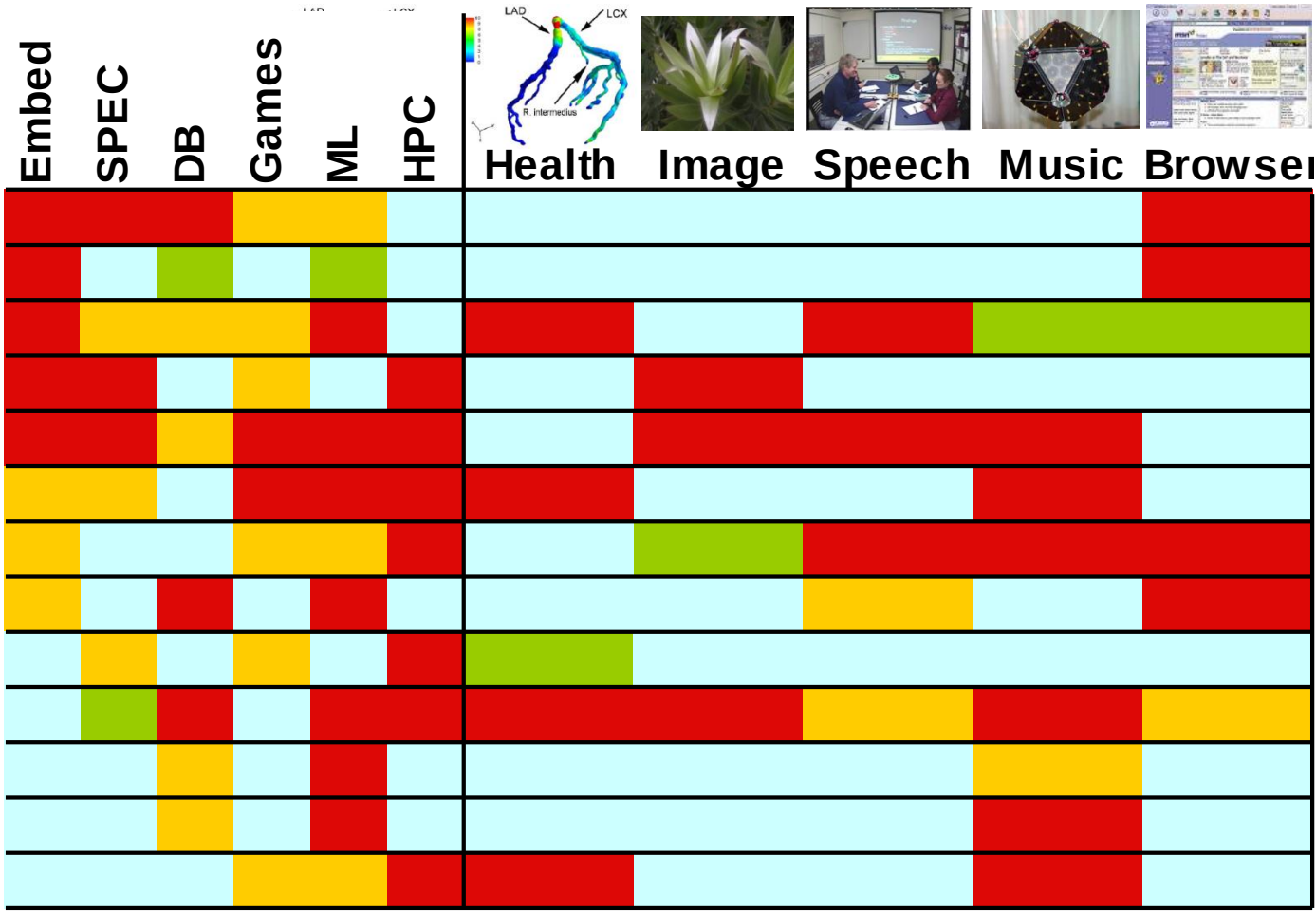
- Infrastructure scalability:
  - Bigdata: Tens of billions of documents in web search
  - Tens/hundreds of thousands of machines.
  - Tens/hundreds of Millions of users
  - Impact on response time, throughput, &availability,
- Platform software
  - Google GFS, MapReduce and Bigtable .
  - UCSB Neptune at Ask
  - fundamental building blocks for fast data update/access and development cycles

…



What do commercial and CSE applications have in common?

Motif/Dwarf: Common Computational Methods
(Red Hot → Blue Cool)



Outline

- Why ~~powerful~~ all computers must be parallel processors

  Including your laptops and handhelds

- Large CSE problems require powerful computers

  Commercial problems too

- Basic parallel performance models
- Why writing (fast) parallel programs is hard

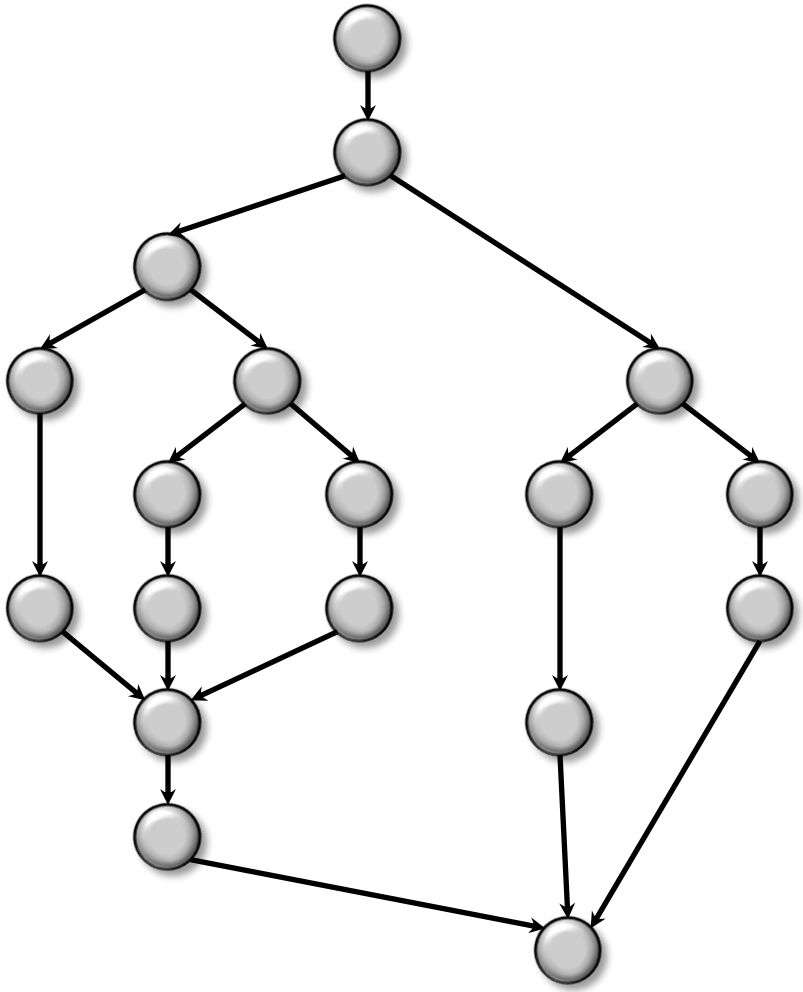
Several possible performance models

- Execution time and parallelism:
  - Work / Span Model with directed acyclic graph
- Detailed models that try to capture time for moving data:
  - Latency / Bandwidth Model for message-passing
  - Disk IO
- Model computation with memory access (for hierarchical memory)
- Other detailed models we won't discuss: LogP, ….

– From John Gibert's 240A course

Work / Span Model

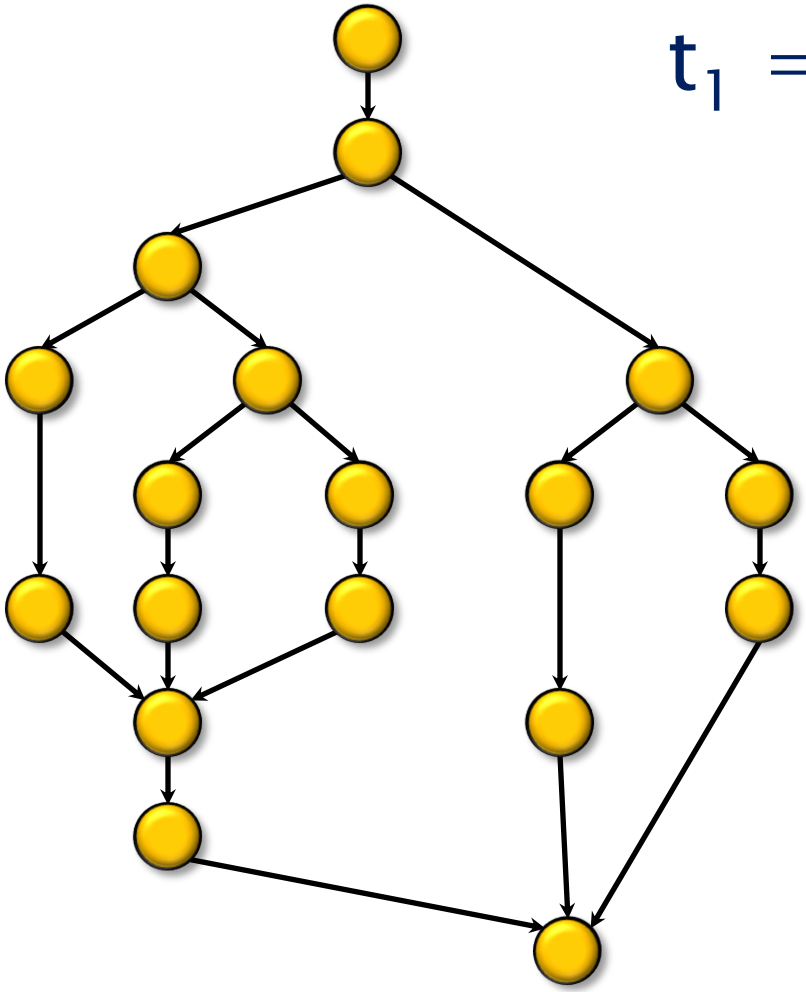
t_p = execution time on p processors



Work / Span Model

t_p = execution time on p processors

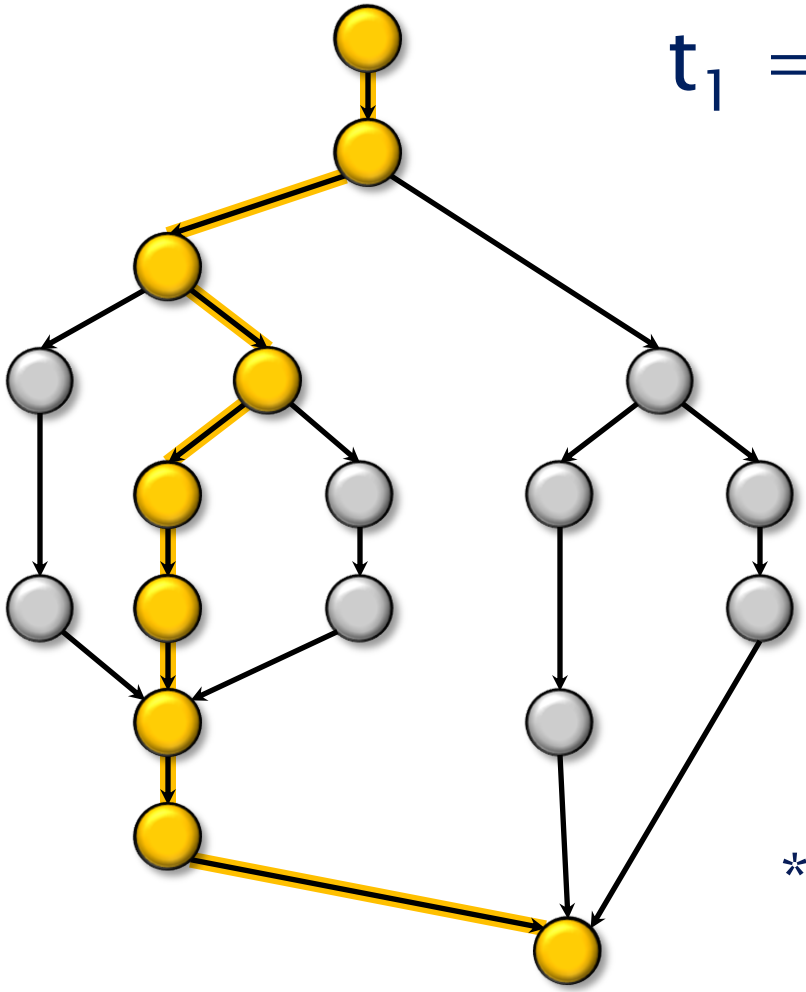
t_1 = ***work***



Work / Span Model

t_p = execution time on p processors

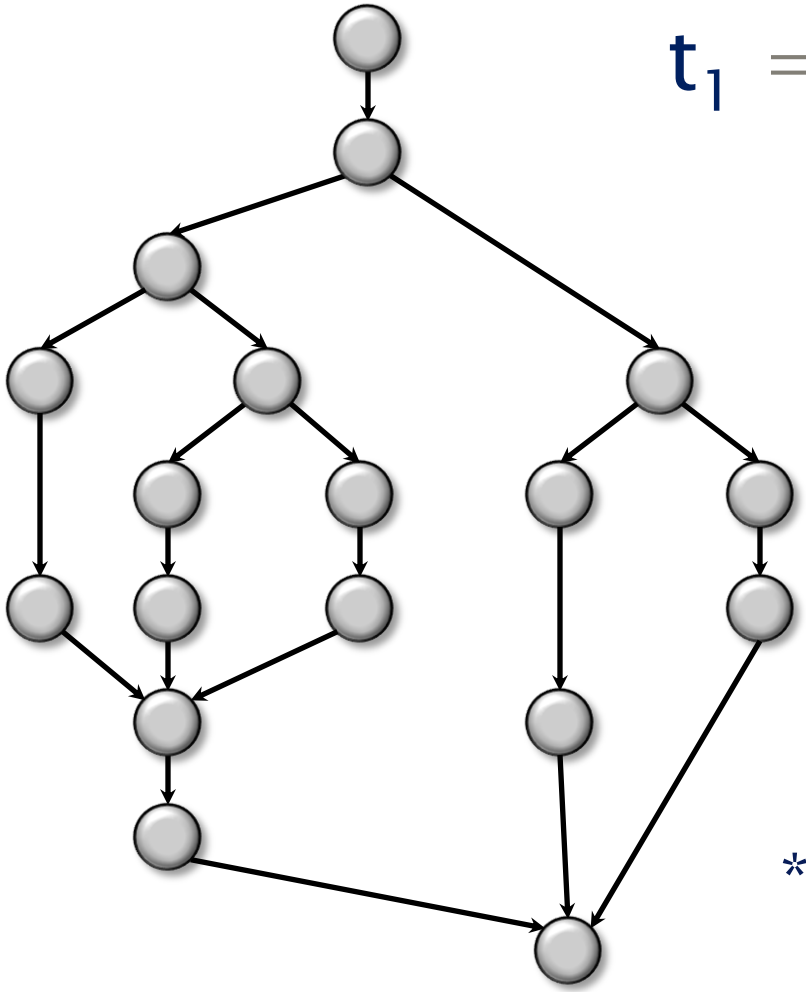
t_1 = *work* t_∞ = *span* *



*Also called *critical-path length* or *computational depth*.

Work / Span Model

t_p = execution time on p processors



t_1 = *work* t_∞ = *span* *

WORK LAW

- $t_p \geq t_1/p$

SPAN LAW

- $t_p \geq t_\infty$

*Also called *critical-path length* or *computational depth*.

Speedup

Def. $t_1/t_P =$ ***speedup*** on p processors.

If $t_1/t_P = \Theta(p)$, we have ***linear speedup***,

$= p$, we have ***perfect linear speedup***,

$> p$, we have ***superlinear speedup***,

(which is not possible in this model, because of the Work Law $t_p \geq t_1/p$)

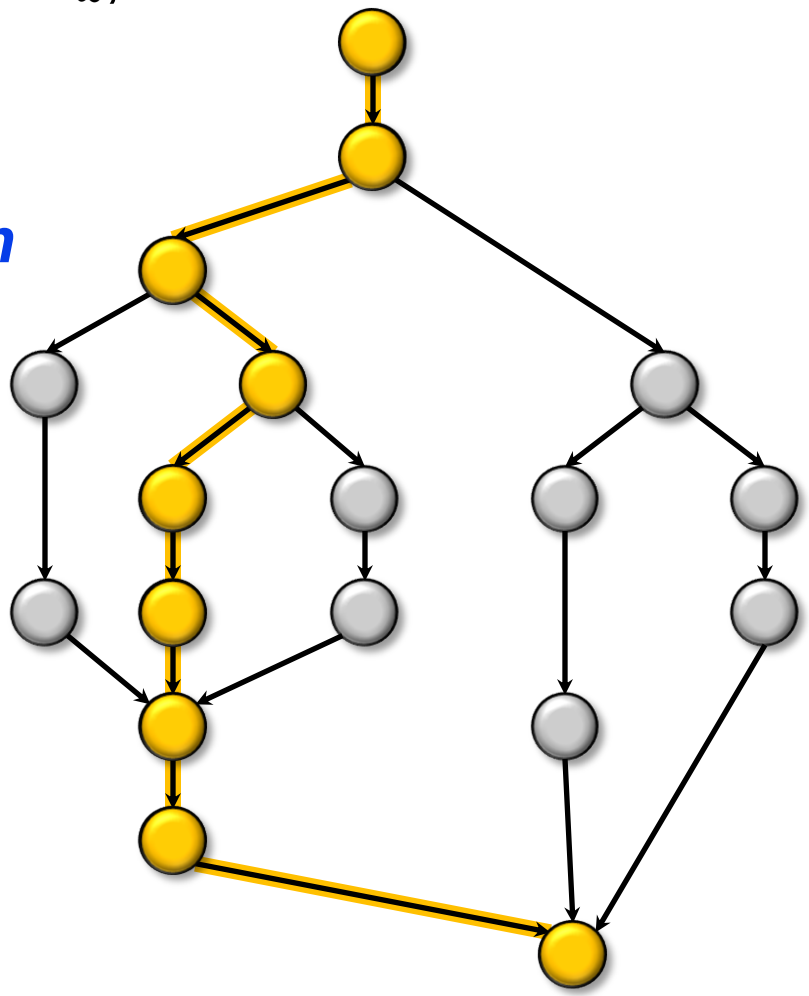
Parallelism

Because the Span Law requires $t_p \geq t_\infty$,

the maximum possible speedup is

t_1/t_∞ = ***(potential) parallelism***

= the average amount of work per step along the span.



Performance Measures for Parallel Computation

Problem parameters:

- n index of problem size
- p number of processors

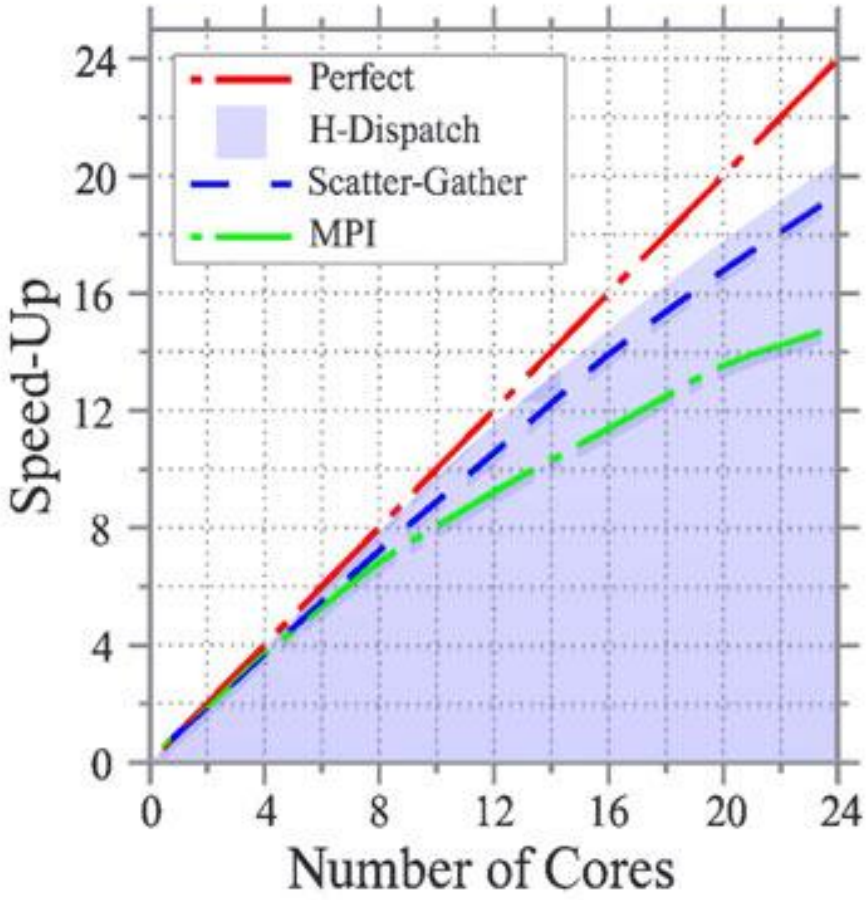
Algorithm parameters:

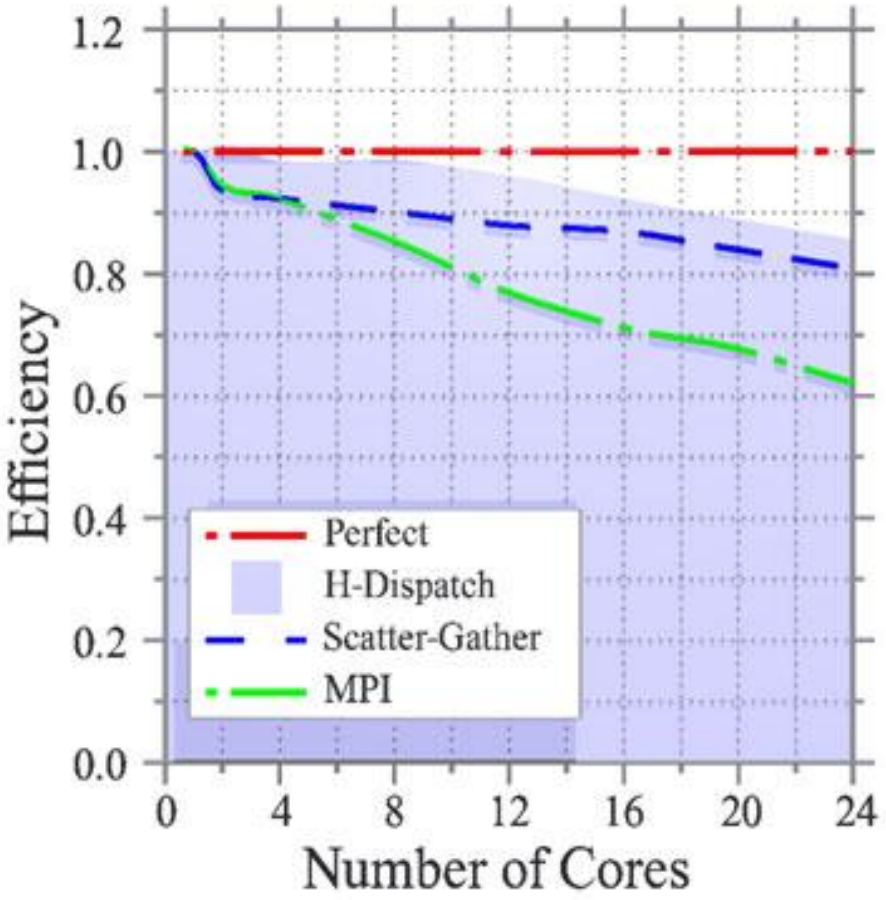
- t_p running time on p processors
- t_1 time on 1 processor = sequential time = "work"
- t_∞ time on unlimited procs = critical path length = "span"
- v total communication volume

Performance measures

- speedup $s = t_1 / t_p$
- efficiency $e = t_1 / (p*t_p) = s / p$
- (potential) parallelism $pp = t_1 / t_\infty$

Typical speedup and efficiency of parallel code





Laws of Parallel Performance

- Work law: $t_p \geq t_1 / p$
- Span law: $t_p \geq t_\infty$
- Amdahl's law:
  - If a fraction s, between 0 and 1, of the work must be done sequentially, then

$$\text{speedup} \leq 1 / s$$

Performance model for data movement: Latency/Bandwith Model

Moving data between processors by message-passing

- Machine parameters:
  - α or $t_{startup}$ latency (message startup time in seconds)
  - β or t_{data} inverse bandwidth (in seconds per word)
  - between nodes of Triton, $\alpha \sim 2.2 \times 10^{-6}$ and $\beta \sim 6.4 \times 10^{-9}$

- Time to send & recv or bcast a message of w words: $\alpha + w*\beta$

- t_{comm} total commmunication time

- t_{comp} total computation time

- Total parallel time: $t_p = t_{comp} + t_{comm}$

Modeling Computation and Memory Access

- Assume just two levels in memory hierarchy, fast and slow
- All data initially in slow memory
  - m = number of memory elements (words) moved between fast and slow memory
  - t_m = time per slow memory operation
  - f = number of arithmetic operations
  - t_f = time per arithmetic operation, $t_f << t_m$
  - $q = f / m$ average number of flops per slow element access
- Minimum possible time = $f * t_f$ when all data in fast memory
- Actual time
  - $f * t_f + m * t_m = f * t_f * (1 + t_m/t_f * 1/q)$
- Larger q means time closer to minimum $f * t_f$

Outline

- Why ~~powerful~~ all computers must be parallel processors

  Including your laptops and handhelds

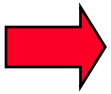
- Large CSE/commerical problems require powerful computers

- Performance models

- Why writing (fast) parallel programs is hard

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
- I/O parallelism in storage level

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 fraction parallelizable
  - P = number of processors

$$\text{Speedup}(P) = \text{Time}(1)/\text{Time}(P)$$

$$\leq 1/(s + (1-s)/P)$$

$$\leq 1/s$$

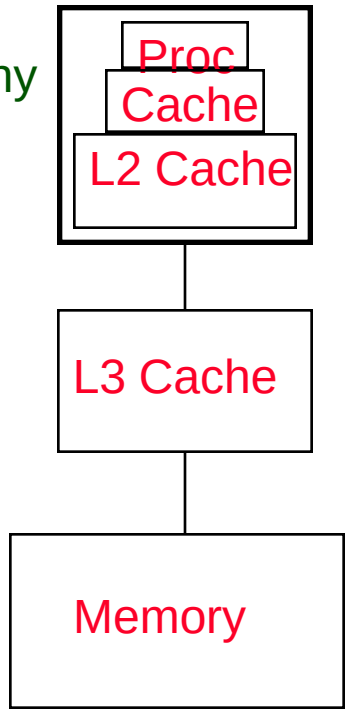
- Even if the parallel part speeds up perfectly performance is limited by the sequential part
- Top500 list: top machine has P~224K; fastest has ~186K+GPUs

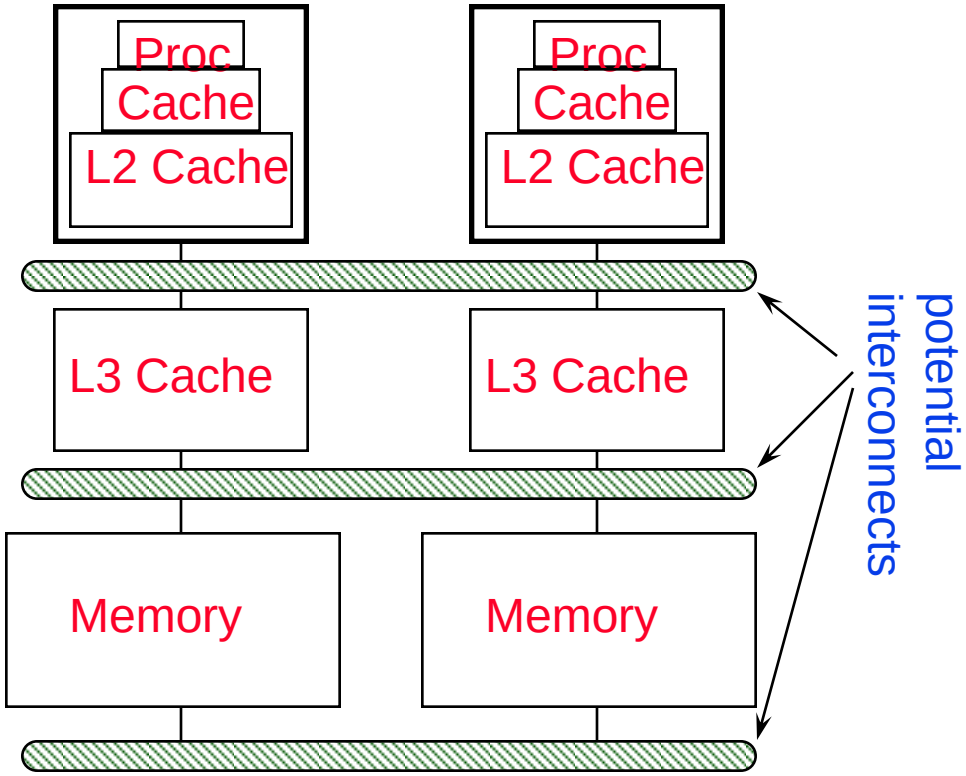
Overhead of Parallelism

- 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 accessing data, 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

Conventional Storage Hierarchy





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

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”

Improving Real Performance

Peak Performance grows exponentially, a la Moore's Law

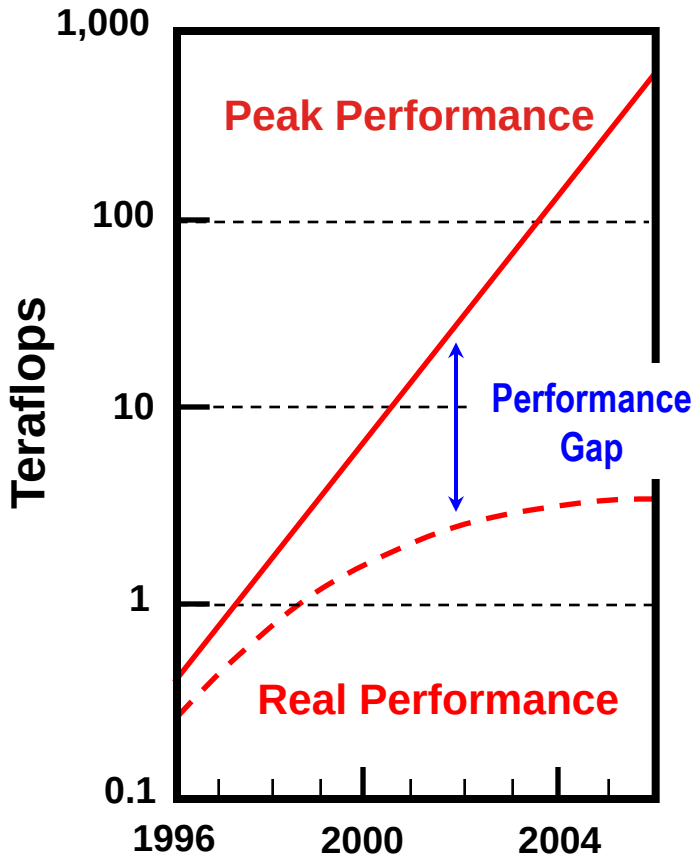
- In 1990's, peak performance increased 100x; in 2000's, it will increase 1000x

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)

What you should get out of the course

In depth understanding of:

- When is parallel computing useful?
- Understanding of parallel computing hardware options.
- Overview of programming models (software) and tools.
- Some important parallel applications and the algorithms
- Performance analysis and tuning
- Exposure to various open research questions

Course Deadlines (Tentative)

- Week 1: join Google discussion group.

Email your name, UCSB email, and ssh key to scc@oit.ucsb.edu for Triton account.

- Jan 27: 1-page project proposal.
  The content includes: Problem description, challenges (what is new?), what to deliver, how to test and what to measure, milestones, and references
- Jan 29 week: Meet with me on the proposal and paper(s) for reviewing
- Feb 6: HW1 due (may be earlier)
- Feb 18 week: Paper review presentation and project progress.
- Feb 27. HW2 due.
- Week 13-17. Take-home exam. Final project presentation. Final 5-page report.