Lecture 1: An Introduction Concurrent and Multicore Programming CSE 436/536, Winter 2017

Department of Computer Science and Engineering
Yonghong Yan
yan@oakland.edu
www.secs.oakland.edu/~yan

Course information

- **Meeting Time:** 3:30 pm – 5:17 pm Monday Wednesday
- **Place: Engineering Center 550**
- **Grade: 45**% for 3 homeworks + 50% project + 5% attendance

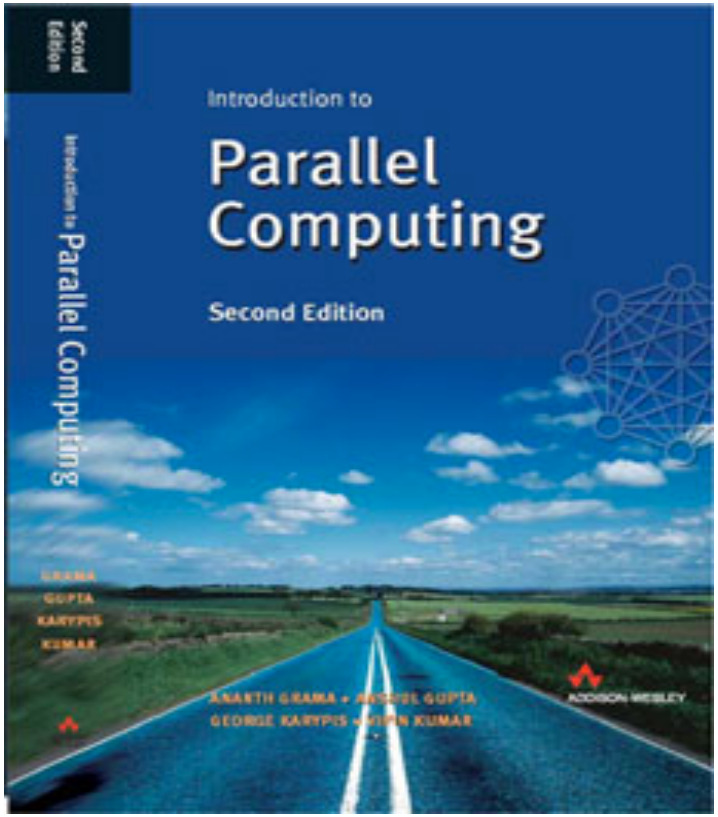
- **Instructor:** Yonghong Yan
  - www.secs.oakland.edu/~yan, yan@oakland.edu
  - **Office**: 534 Engineering Center, **Tel:** (248) 370-4087
  - **Office Hours:** After class or by appointment

- **Public Course website**: http://passlab.github.io/CSE436536/
- **Private and homework submission: moodle** (https://moodle.oakland.edu/course/view.php?id=168842)
- **Syllabus** for more details

Objectives

- Learn fundamentals of concurrent and parallel computing
  - Describe benefits and applications of concurrent and parallel programming.
  - Explain key concepts in parallel computer architectures, e.g. shared memory system, distributed system, NUMA and cache coherence.
  - Understand principles for concurrent program design, e.g. decomposition of works, task and data parallelism, processor mapping, mutual exclusion, locks.
- Develop skills writing and analyzing parallel programs
  - Write parallel program using OpenMP, Cilk/Cilkplus, CUDA, and MPI programming models.
  - Perform analysis of parallel program problem.

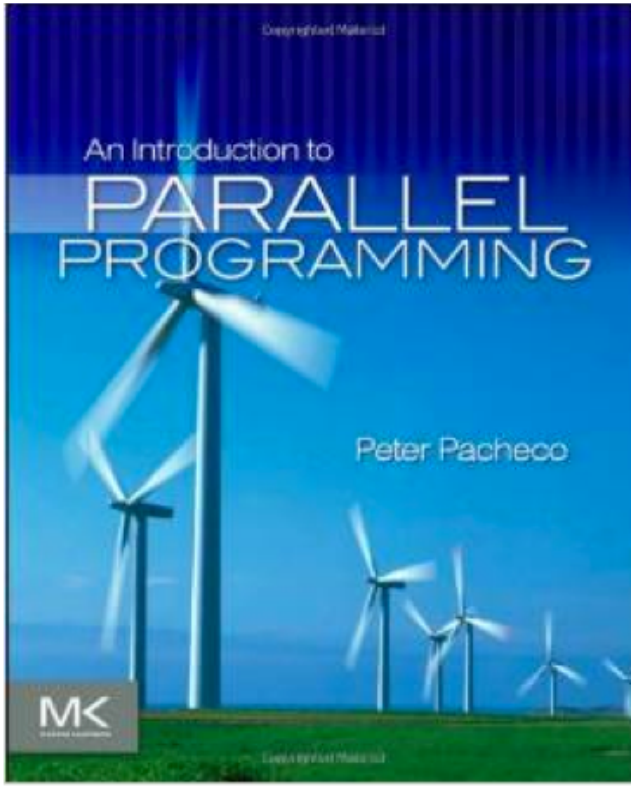
Recommended textbook



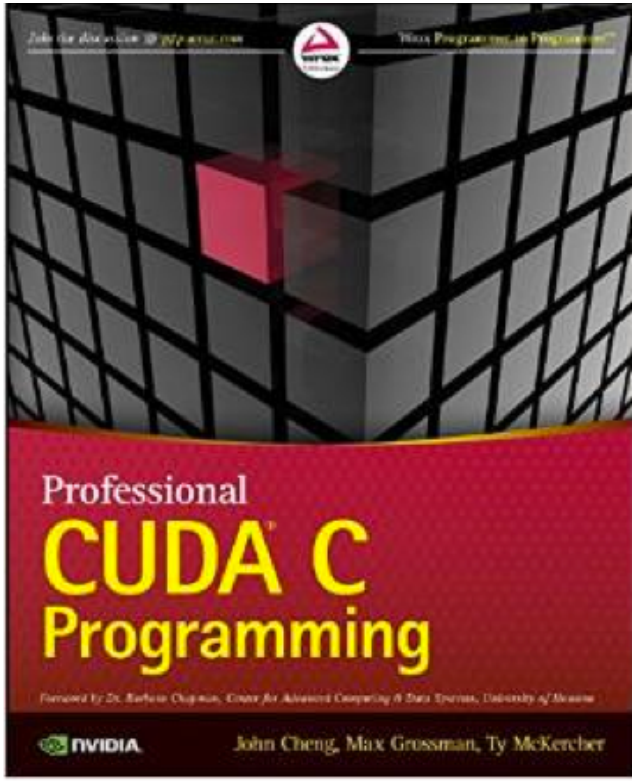
By Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar
Addison-Wesley, 2003

- Lots of materials on Internet.
  - On the website, there is a "Resources" section that provides web page links, documents, and other materials for this course

Other two reference books I used



An Introduction to Parallel Programming, by Peter Pacheco, Morgan Kaufmann Publishers Inc, http://www.cs.usfca.edu/~peter/ipp



Professional CUDA C Programming, by John Cheng, Max Grossman, Ty McKercher, http://www.wiley.com/WileyCDA/WileyTitle/productCdN 1118739329.html

Homework and Project

- Homeworks: Apply theory and practice programming skills
  - Require both good and correct programming
    - Write organized program that is easy to read
  - Report and discuss your findings in report
    - Writing good document
- Project: Study a real challenge and develop solutions
  - Study related work, identify problem, develop solutions, perform experiment and analyze your results
  - Present your findings to the class
  - Report your findings
    - In the form of publishable paper
- Our class will have practice sessions to help the assignments and project.

Topics (Part 1)

- Introduction
- Principles of parallel algorithm design (Chapter 3)
- Programming on shared memory system (Chapter 7)
  - **OpenMP**
  - **Cilk/Cilkplus**
  - **PThread, mutual exclusion, locks, synchronizations**
- Analysis of parallel program executions (Chapter 5)
  - **Performance Metrics for Parallel Systems**
    - **Execution Time, Overhead, Speedup, Efficiency, Cost**
  - **Scalability of Parallel Systems**
  - **Use of performance tools**

Topics (Part 2)

- Parallel architectures and hardware
  - **Parallel computer architectures**
  - **Memory hierarchy and cache coherency**
- Manycore GPU architectures and programming
  - **GPUs architectures**
  - **CUDA programming**
  - Introduction to offloading model in OpenMP
- Programming on large scale systems (Chapter 6)
  - **MPI (point to point and collectives)**
  - Introduction to PGAS languages, UPC and Chapel
- Parallel algorithms (Chapter 8,9 &10)
  - **Sorting and Stencil**

Prerequisites

- Good reasoning and analytical skills
- Skills of C programming
  - macro, pointer, array, struct, union, function pointer, etc.
- Basic knowledge of computer architecture and data structures
  - Memory hierarchy, cache, virtual address
  - Array and link-list
- Familiarity with Linux environment
- Talk with me if you have concern
- The survey

Introduction: What is and why Parallel Computing

Terms:

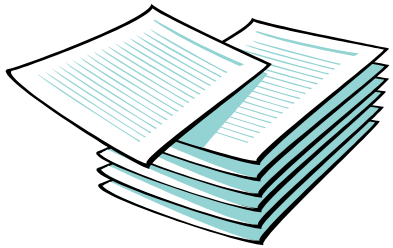
Concurrent and Multicore Programming

Parallel Programming

An example: grading

15 questions

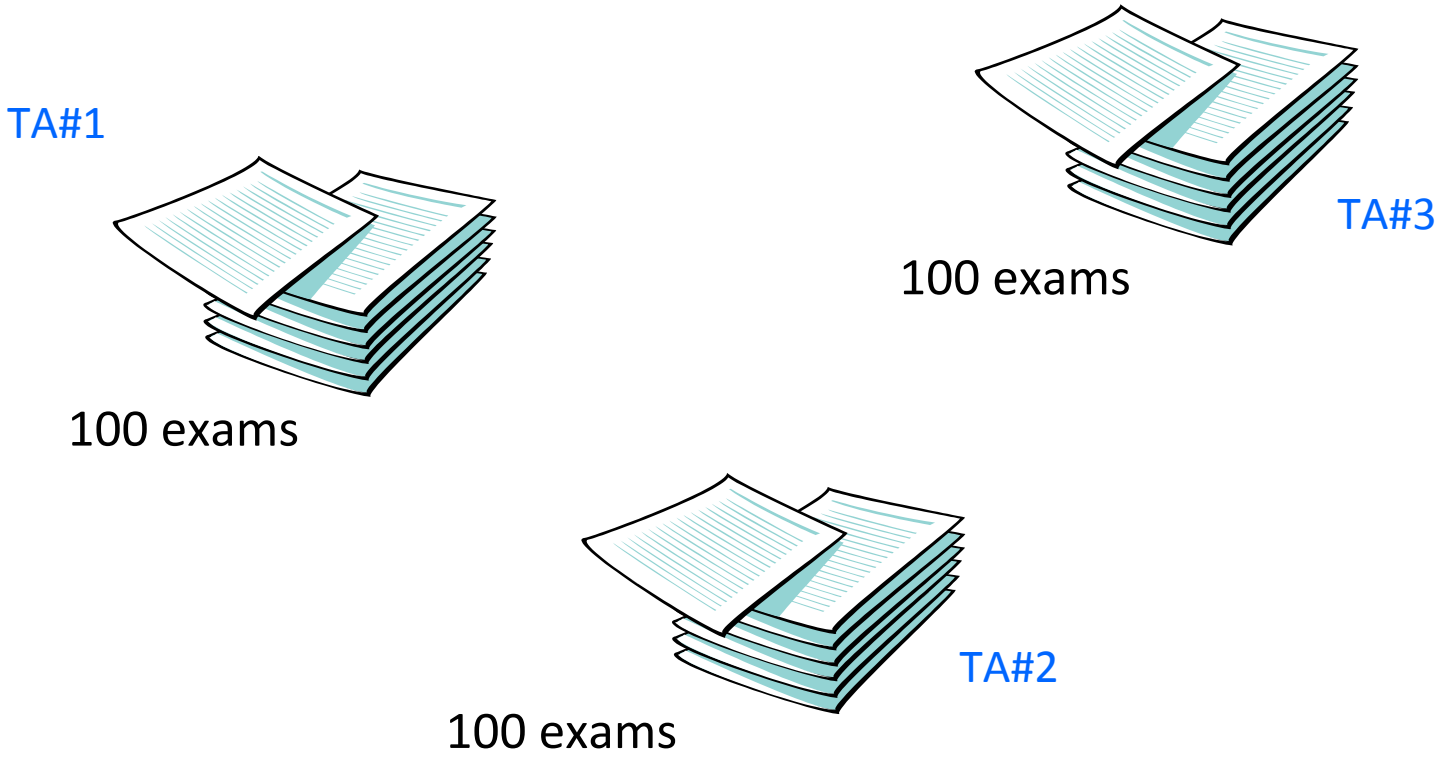
300 exams



Teaching assistants

Division of work – data parallelism

- Each does the same type of work (task), but working on different sheet (data)



Division of work – task parallelism

- Each does different type of work (task), but working on same sheets (data)

TA#1

Questions 1 - 5

TA#3

Questions 11 - 15

TA#2

Questions 6 - 10

Summary

- Data: 300 copies of exam
- Task: grade total 300*15 questions
- Data parallelism
  - **Distributed 300 copies to three TAs**
  - **They work independently**
- Task Parallelism
  - **Distributed 300 copies to three TAs**
  - **Each grades 5 questions of 100 copies**
  - **Exchange copies**
  - **Grade 5 questions again**
  - **Exchange copies**
  - **Grade 5 questions**
- The three TAs can do in parallel, we can achieve 3 time speedup theoretically

Which approach could be faster!

	Data Parallelism	Task Parallelism
Data	Different	Same
Task	Same	Different

Challenges

- Are the three TAs grading in the same performance?
  - One CPU may be slower than the other
  - They may not work on grading the same time
- How the TAs communicate?
  - Are they sit on the same table? Or each take copies and grade from home? How they share intermediate results (task parallelism)
- Where the solutions are stored so they can refer to when grading
  - Remember answers to 5 questions vs to 15 questions
    - Cache and Memory issues

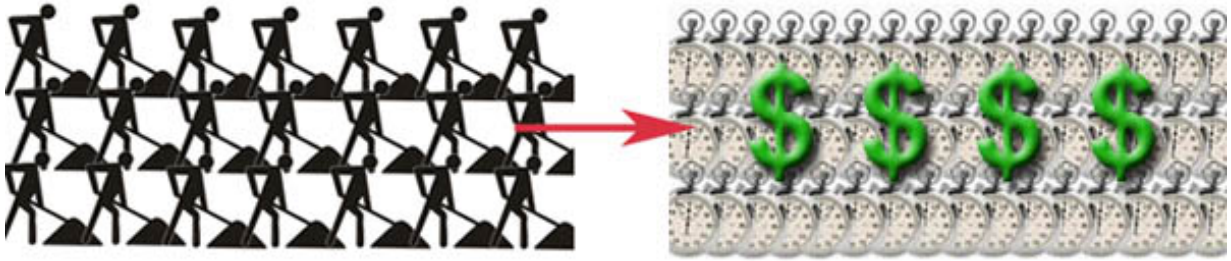
What is parallel computing?

- A form of computation*:
  - Large problems divided into smaller ones
  - Smaller ones are carried out and solved simultaneously
- Uses more than one CPUs or processor cores concurrently for one program
  - Not conventional time-sharing: multiple programs switch between each other on one CPU
    - so fast that we donot notice that they are one after another.
  - Or multiple programs each on a CPU and not interacting
- Serial processing
  - Some programs, or part of a program are inherently serial
  - Most of our programs and desktop applications

*http://en.wikipedia.org/wiki/Parallel_computing

Why Parallel Computing?

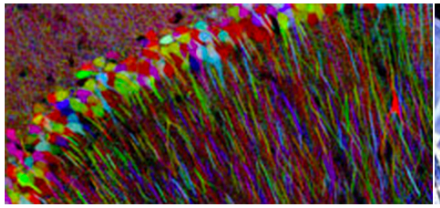
- Save time (execution time) and money!
  - Parallel program can run faster if running concurrently instead of sequentially.



Picture from: Intro to Parallel Computing: https://computing.llnl.gov/tutorials/parallel_comp

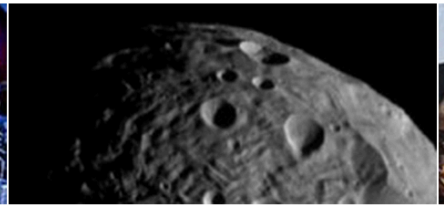
- Solve larger and more complex problems!
  - Utilize more computational resources

Current Grand Challenges



NIH, DARPA, and NSF's BRAIN Initiative, to revolutionize our understanding of the human mind and

DOE's SunShot Grand Challenge, to make solar energy cost competitive with coal by the end of the decade, and EV



NASA's Asteroid Grand Challenge, to find all asteroid threats to human populations and know what to do about

USAID's Grand Challenges for Development, including **Saving Lives at Birth** that catalyzes groundbreaking

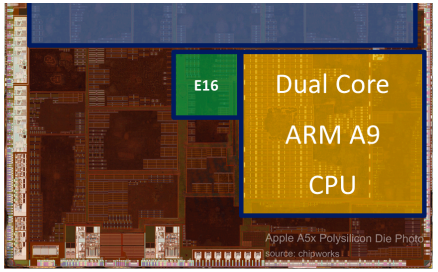
From "21st Century Grand Challenges | The White House", http://www.whitehouse.gov/administration/eop/ostp/grand-challenges
Grand challenges: http://en.wikipedia.org/wiki/Grand_Challenges

High performance computing (HPC) and parallel computing

- HPC is what really needed *
  - Parallel computing is so far the only way to get there!!
- Parallel computing makes sense!

- **Applications that require HPC**
  - Many problem dom
  - Data cannot fit in m
- **Computer systems**
  - Physics limitation: h
  - Parallel systems are widely accessible
    - Smartphone has 2 to 4 cores + GPU now

We will discuss each of the two aspect today!



*What is HPC: http://insidehpc.com/hpc-basic-training/what-is-hpc/
Supercomputer: http://en.wikipedia.org/wiki/Supercomputer
TOP500 (500 most powerful computer systems in the world): http://en.wikipedia.org/wiki/TOP500, http://top500.org/
HPC matter: http://sc14.supercomputing.org/media/social-media

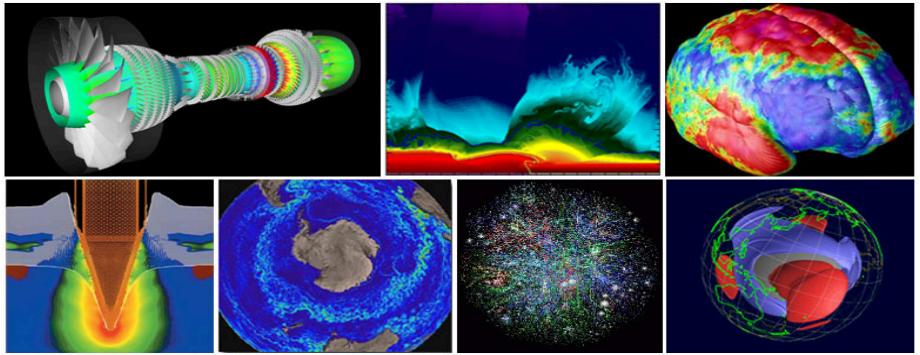
Simulation: The Third Pillar of Science

- Traditional scientific and engineering paradigm:
  1) Do **theory** or paper design.
  2) Perform **experiments** or build system.
- Limitations of experiments:
  - 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 paradigm:
  3) Use high performance computer systems to **simulate** the phenomenon
     - Base on known physical laws and efficient numerical methods.

From slides of Kathy Yelic's 2007 course at Berkeley: http://www.cs.berkeley.edu/~yelick/cs267_sp07/

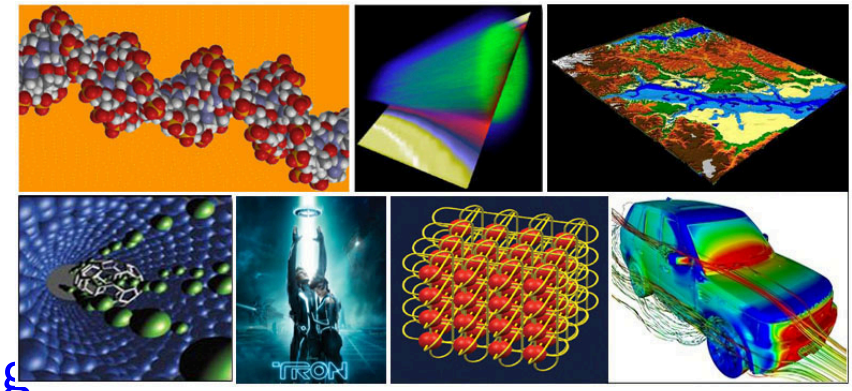
Applications: Science and engineering

- Model many difficult problems by parallel computing
  - Atmosphere, Earth, Environment
  - Physics - applied, nuclear, particle, condensed matter, high pressure, fusion, photonics
  - Bioscience, Biotechnology, Genetics
  - Chemistry, Molecular Sciences
  - Geology, Seismology
  - Mechanical Engineering - from prosthetics to spacecraft
  - Electrical Engineering, Circuit Design, Microelectronics
  - Computer Science, Mathematics
  - Defense, Weapons



Applications: Industrial and Commercial

- Processing large amounts of data in sophisticated ways
  - Databases, data mining
  - Oil exploration
  - Medical imaging and diagnosis
  - Pharmaceutical design
  - Financial and economic modeling
  - Management of national and multi-national corporations
  - Advanced graphics and virtual reality, particularly in the entertainment industry
  - Networked video and multi-media technologies
  - Collaborative work environments
  - Web search engines, web based business services

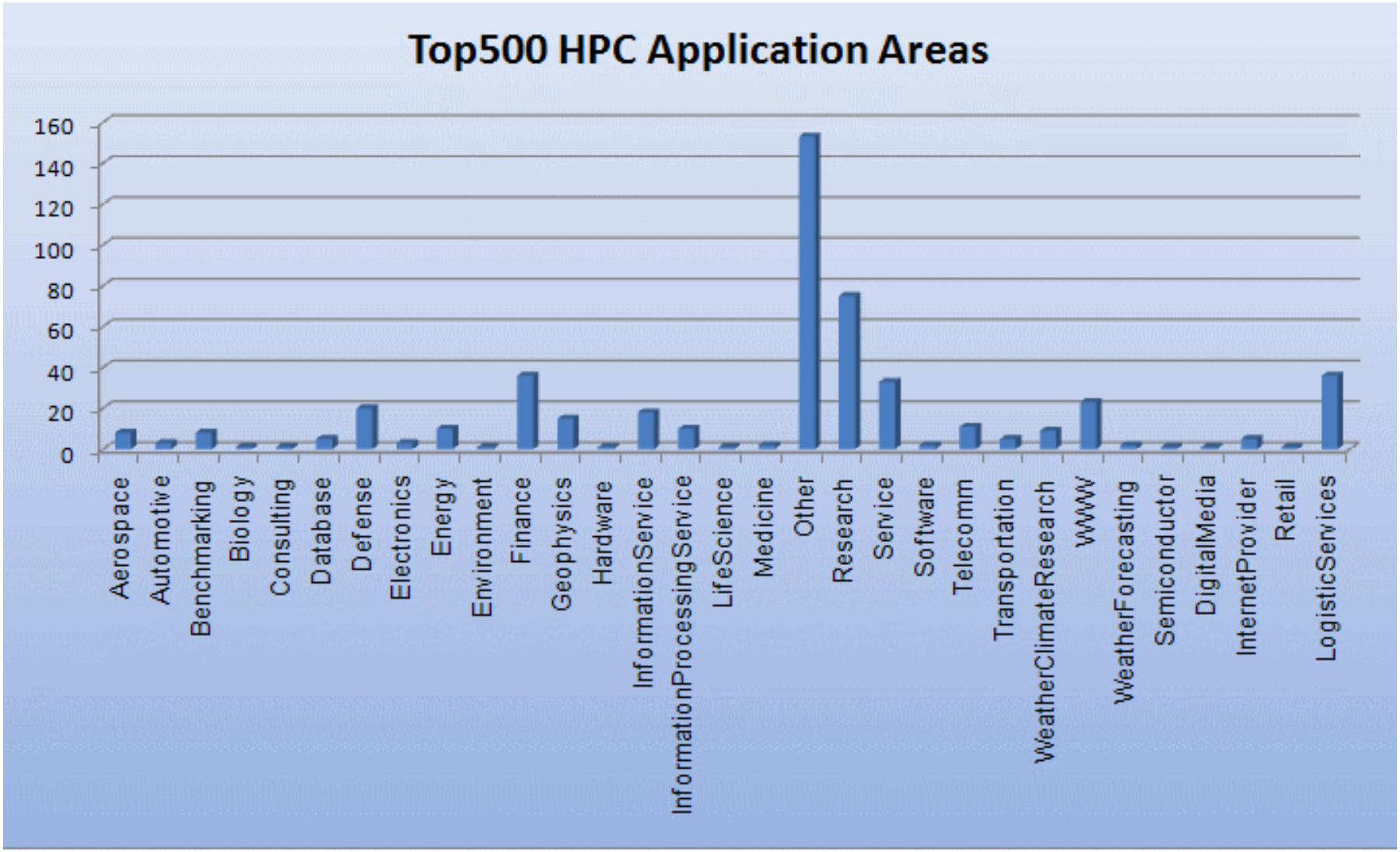


Economic Impact of HPC

- 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.
- Securities industry:
  - Savings: approx. $15 billion per year for U.S. home mortgages.

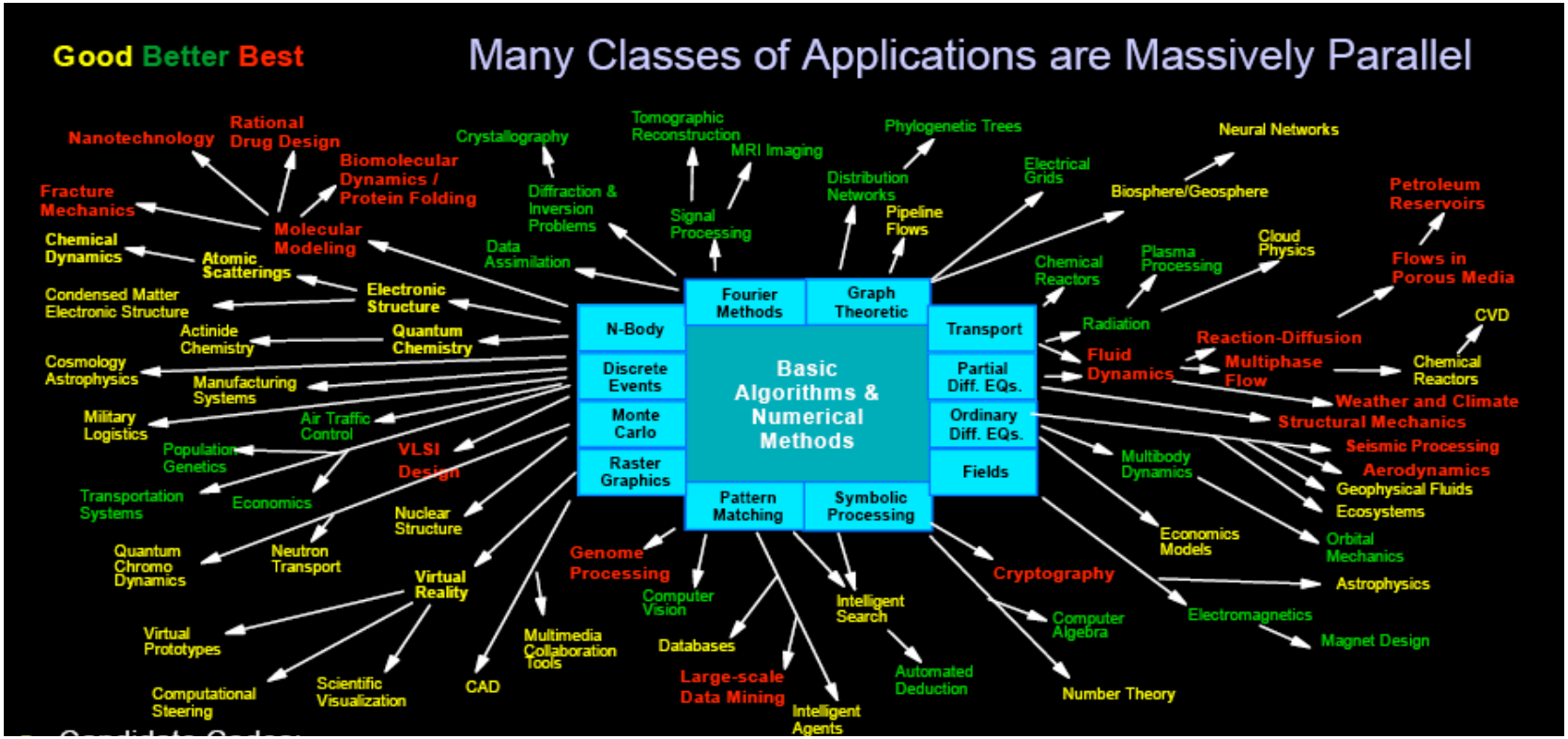
From slides of Kathy Yelic's 2007 course at Berkeley: http://www.cs.berkeley.edu/~yelick/cs267_sp07/

A wide variety of parallel applications globally



Source: top500.org

Parallel applications



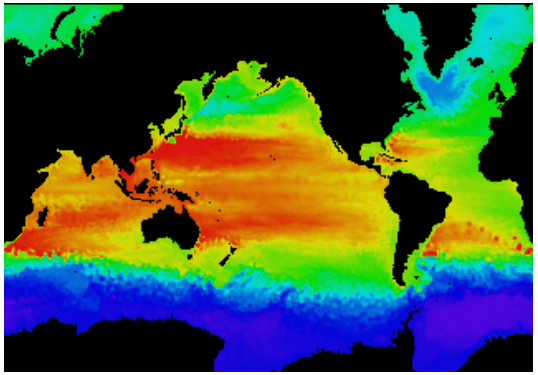
- Example: weather prediction and global climate modeling

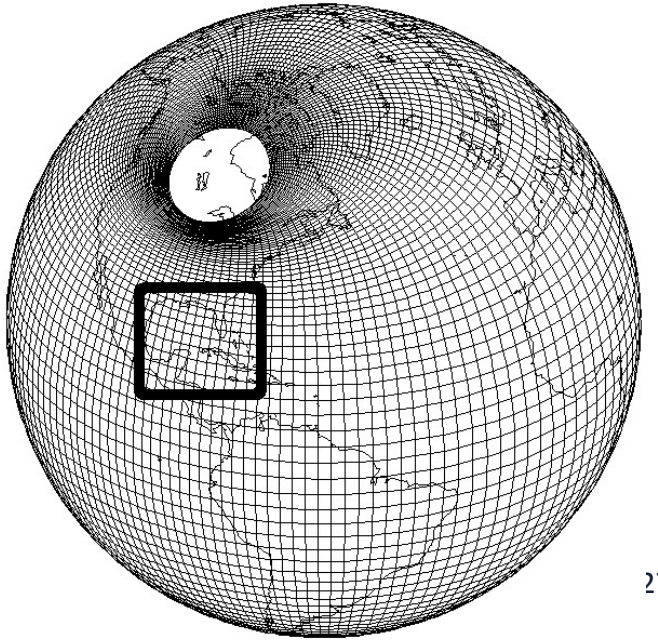
Units of measure in HPC

- **Flop**: floating point operation (*, /, +, -, etc)
- **Flops/s**: floating point operations per second
- **Bytes**: size of data
  - A double precision floating point number is 8 bytes
- Typical sizes are millions, billions, trillions…
  - Mega Mflop/s = 10^6 flop/sec Mzbyte = 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
- See www.top500.org for the units of the fastest machines
  - The fastest: Tianhe-2 or TH-2 (Chinese: 天河-2), 33.86 petaflops
  - The second: DoE ORNL Titan, 17.59 petaflops

Global climate modeling problem

- Problem is to compute:
  - f(latitude, longitude, elevation, time) → 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+dt given t
- Uses:
  - Predict major events, e.g., El Nino
  - Air quality forecasting

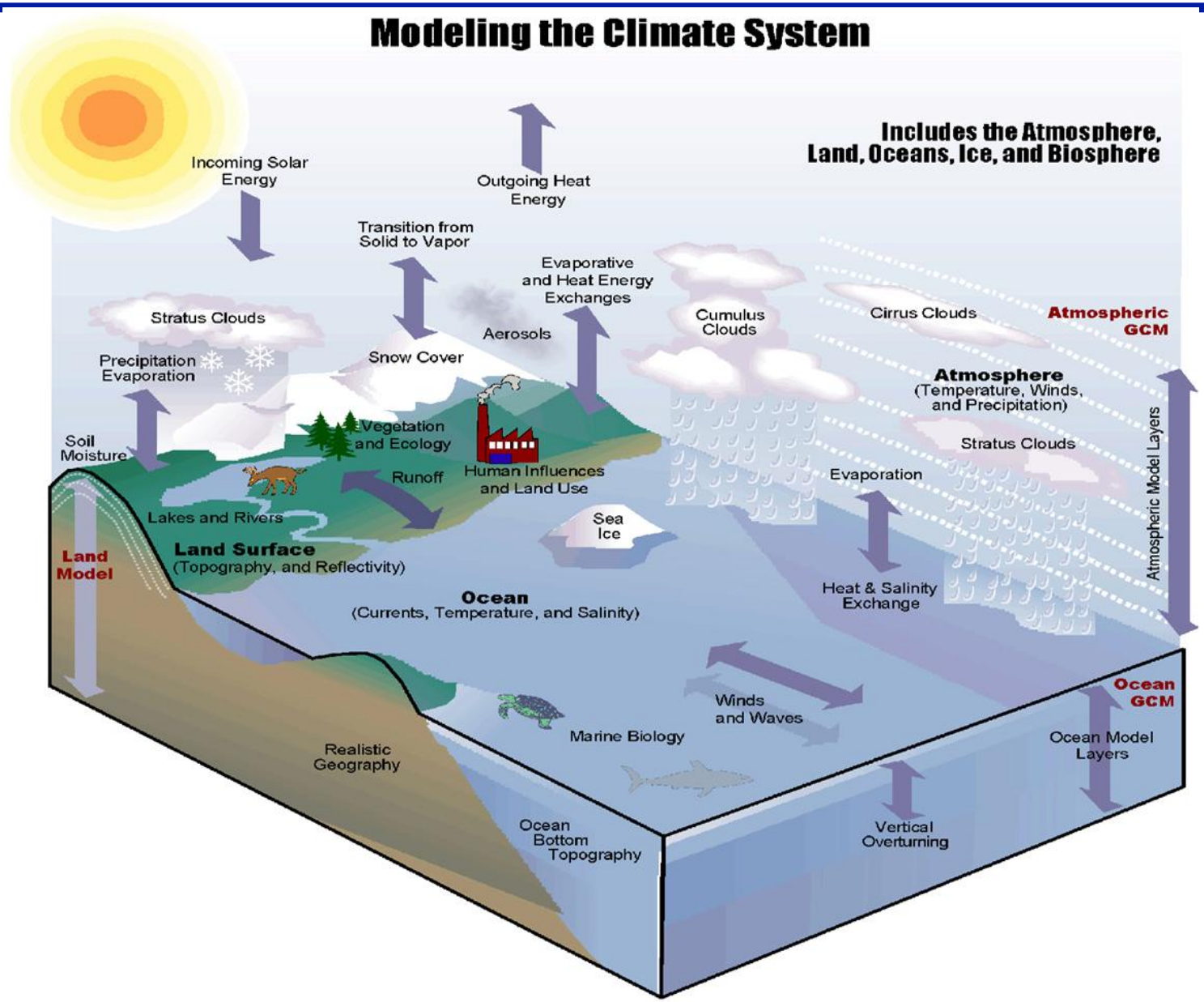




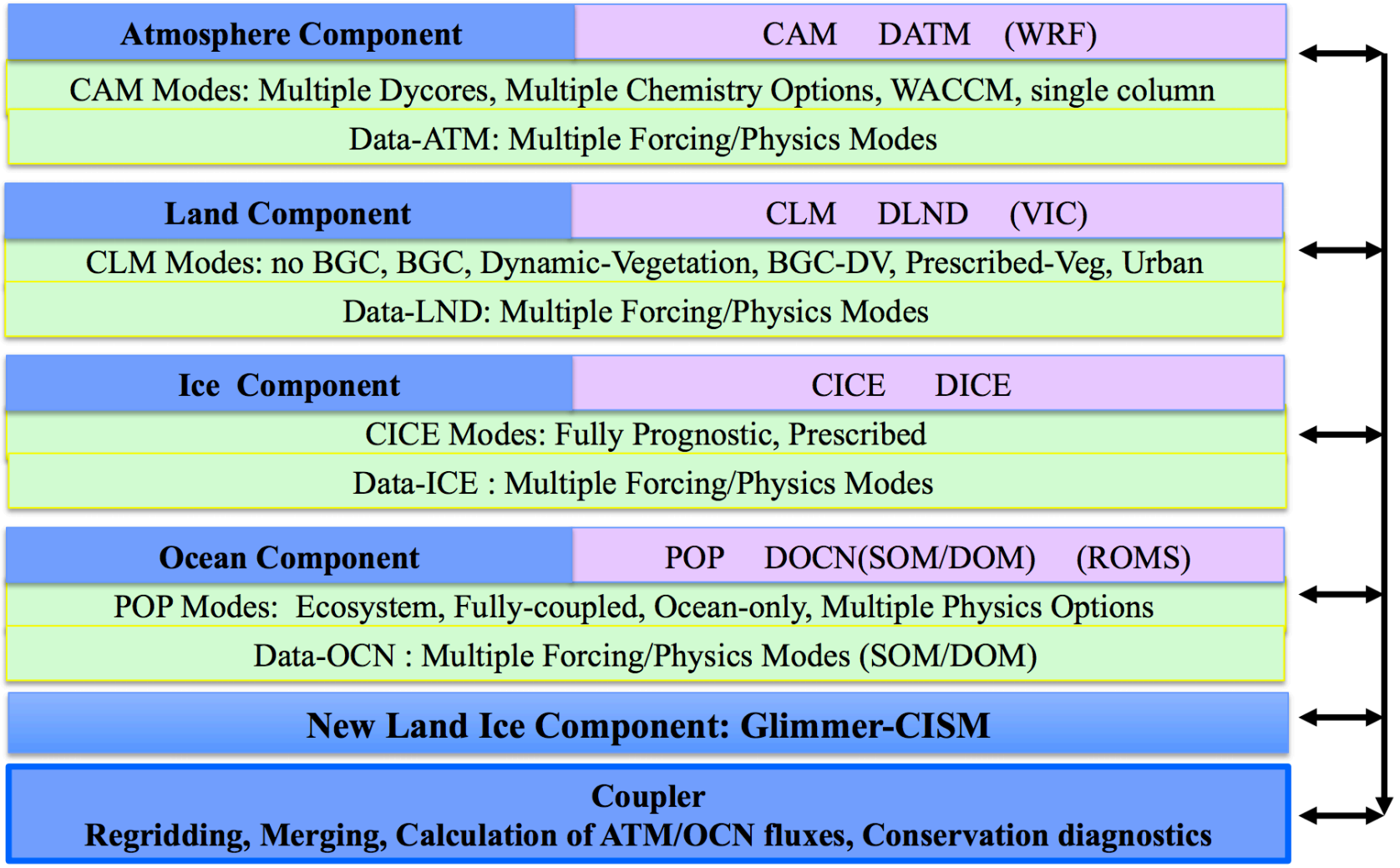
Global climate modeling computation

- 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 x 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, ocean, sea-ice, land models, plus possibly carbon cycle, geochemistry and more

Parameters


Modeling the Climate System
Includes the Atmosphere, Land, Oceans, Ice, and Biosphere
Incoming Solar Energy
Outgoing Heat Energy
Transition from Solid to Vapor
Evaporative and Heat Energy Exchanges
Stratus Clouds
Aerosols
Cumulus Clouds
Cirrus Clouds
Atmospheric GCM
Precipitation
Evaporation
Snow Cover
Atmosphere (Temperature, Winds, and Precipitation)
Vegetation and Ecology
Soil Moisture
Runoff
Human Influences and Land Use
Stratus Clouds
Evaporation
Atmospheric Model Layers
Lakes and Rivers
Sea Ice
Land Model
Land Surface (Topography, and Reflectivity)
Heat & Salinity Exchange
Ocean (Currents, Temperature, and Salinity)
Winds and Waves
Ocean GCM
Marine Biology
Realistic Geography
Ocean Model Layers
Ocean Bottom Topography
Vertical Overturning

Community Earth System Model (CESM)



Picture courtesy of M. Vertenstein (NCAR)

The rise of multicore processors

Semiconductor trend: “Moore’s Law”

Gordon Moore, Founder of Intel

- 1965: since the integrated circuit was invented, the number of transistors/inch^2 in these circuits roughly doubled every year; this trend would continue for the foreseeable future
- 1975: revised - circuit complexity doubles every two years

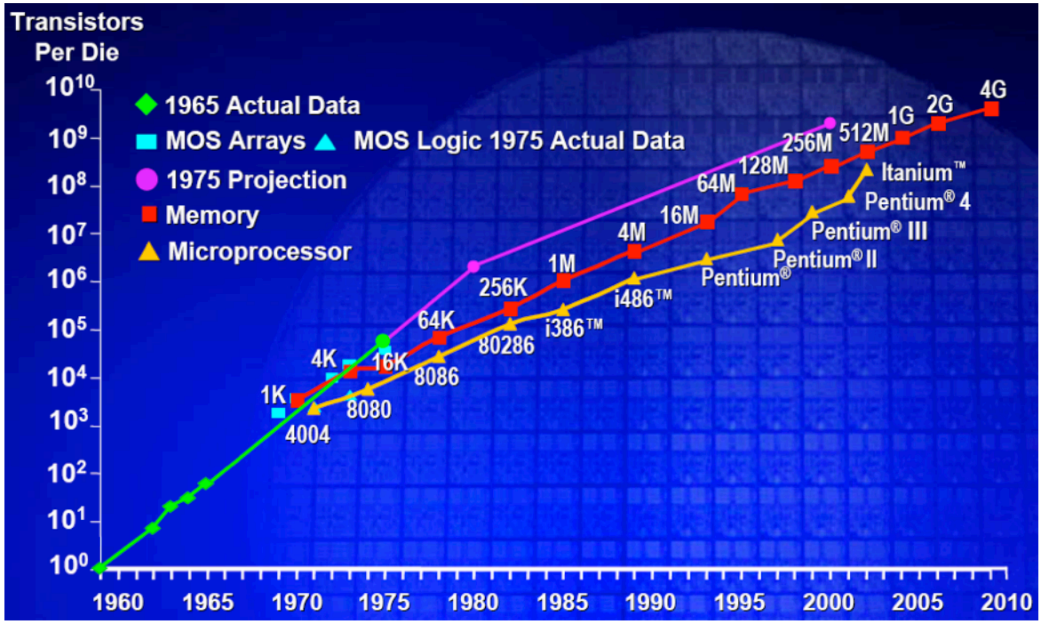
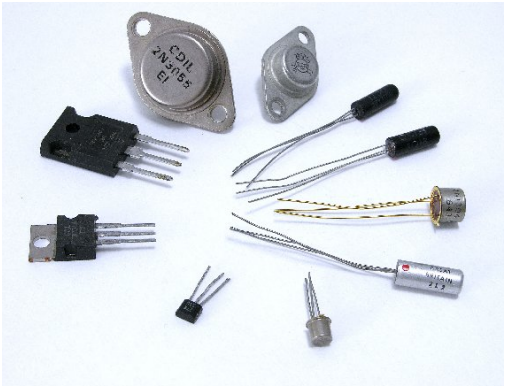
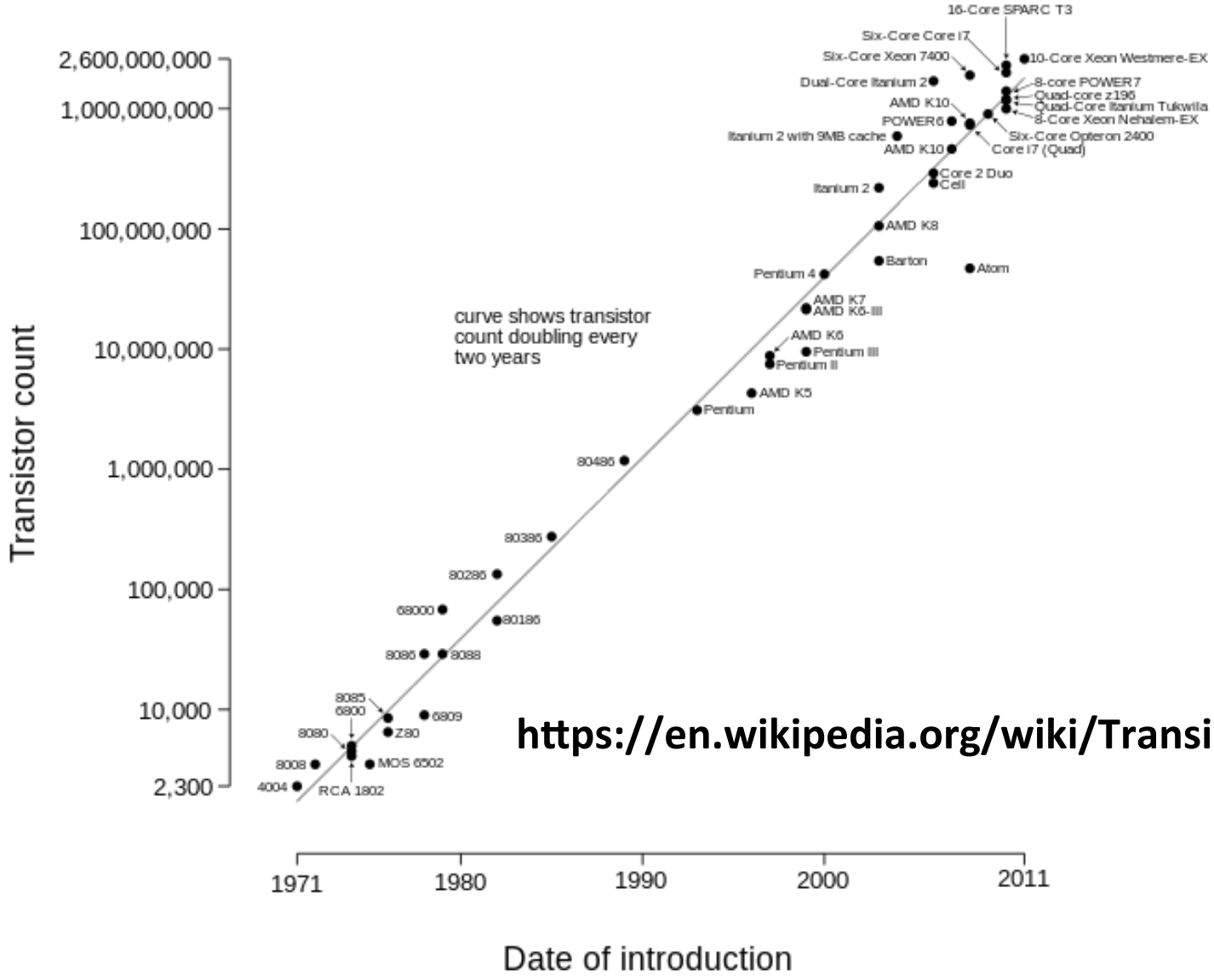


Image credit: Intel



Microprocessor Transistor Counts 1971-2011 & Moore's Law



https://en.wikipedia.org/wiki/Transistor_count

Moore's Law trends

- More transistors = ↑ opportunities for exploiting parallelism in the instruction level (ILP)
  - Pipeline, superscalar, VLIW (Very Long Instruction Word), SIMD (Single Instruction Multiple Data) or vector, speculation, branch prediction
- General path of scaling
  - Wider instruction issue, longer piepline
  - More speculation
  - More and larger registers and cache
- **Increasing circuit density ~= increasing frequency ~= increasing performance**
- Transparent to users
  - An easy job of getting better performance: buying faster processors (higher frequency)

- **We have enjoyed this free lunch for several decades, however …**

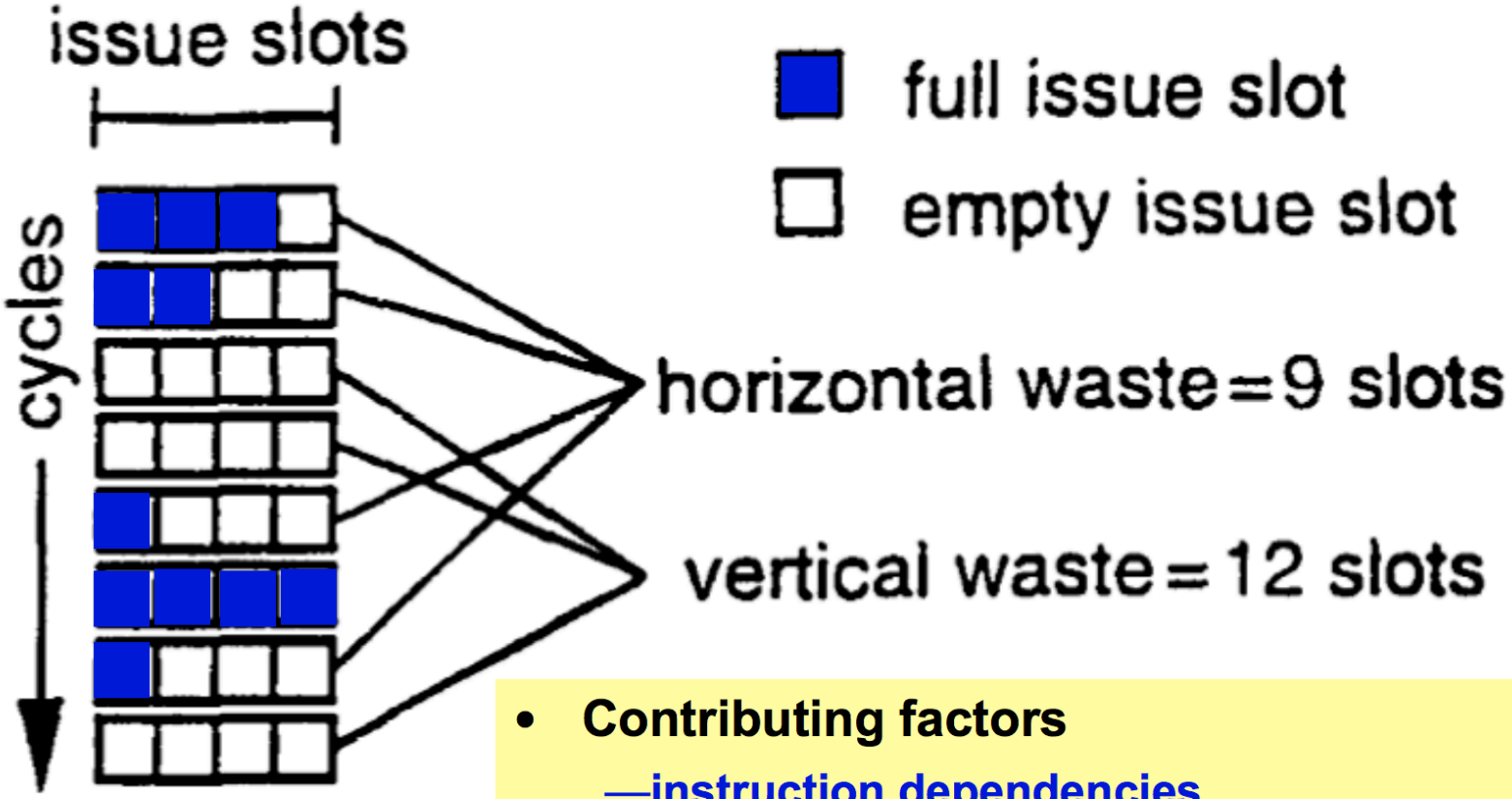
Problems of traditional ILP scaling

- Fundamental circuit limitations[1]
  - delays ⇑ as issue queues ⇑ and multi-port register files ⇑
  - increasing delays limit performance returns from wider issue
- Limited amount of instruction-level parallelism[1]
  - inefficient for codes with difficult-to-predict branches
- Power and heat stall clock frequencies

[1] The case for a single-chip multiprocessor, K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K. Chang, ASPLOS-VII, 1996.

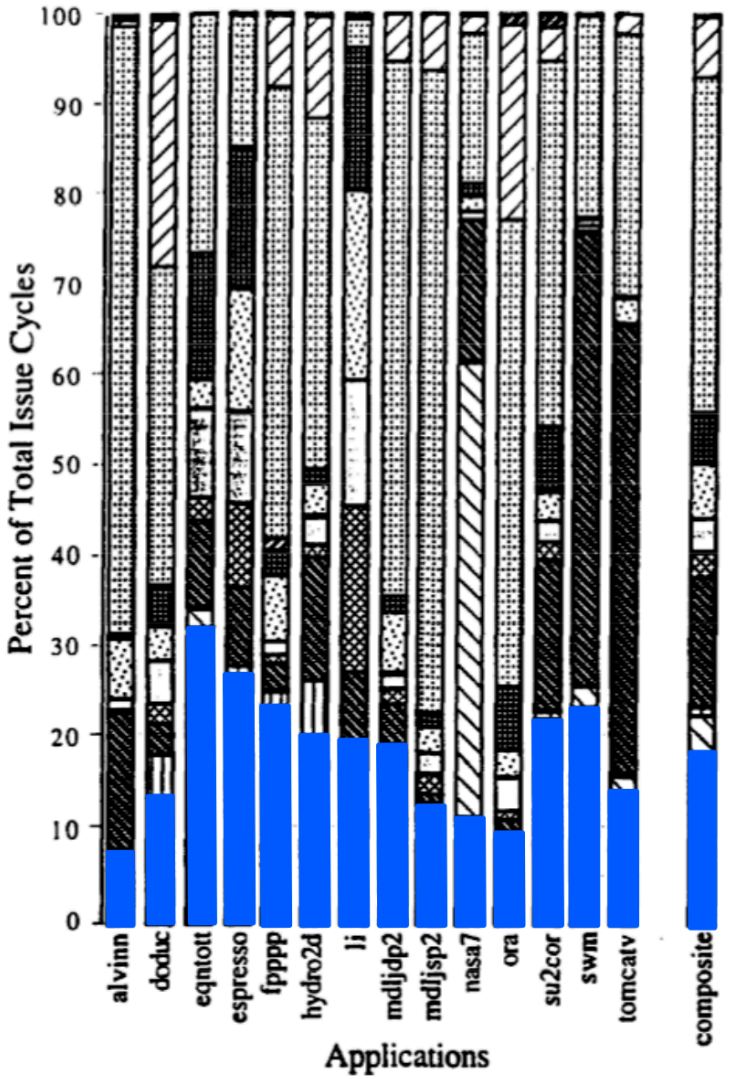
ILP impacts

Issue Waste



- **Contributing factors**
  - **—instruction dependencies**
  - **—long-latency operations within a thread**

Simulations of 8-issue Superscalar



Simultaneous multithreading: maximizing on-chip parallelism, Tullsen et. al. ISCA, 1995.

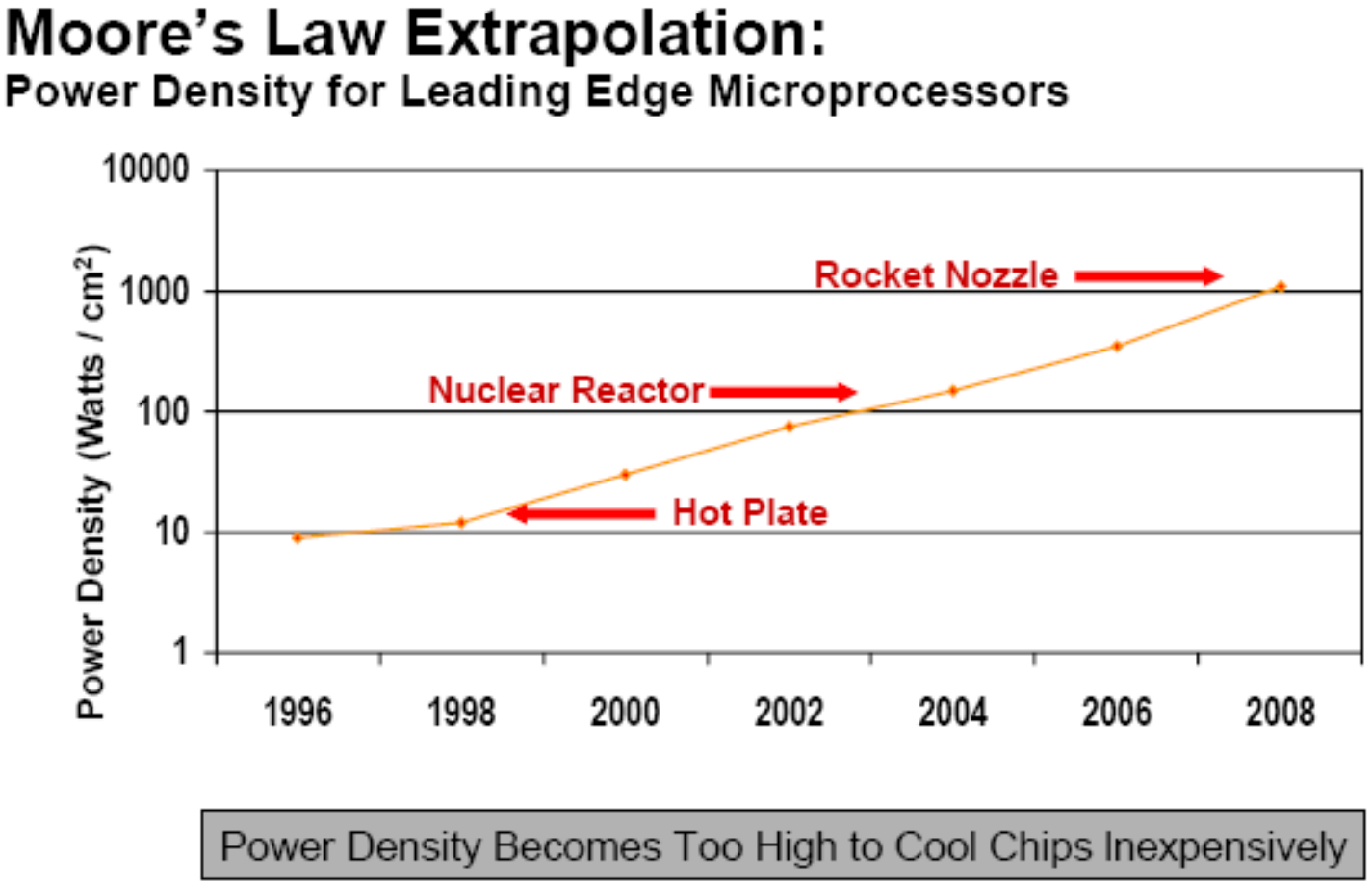
Summary:
Highly underutilized

Applications: most of SPEC92

- On average < 1.5 IPC (19%)
- Dominant waste differs by application
- Short FP dependences: 37%

Power/heat density limits frequency

- Some fundamental physical limits are being reached

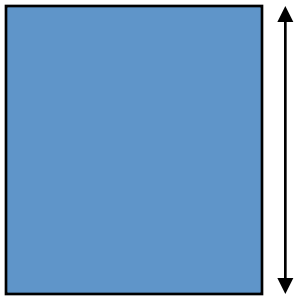


Source: Shekhar Borkar, Intel Corp

We will have this …

More Limits: How fast can a serial computer be?

1 Tflop/s, 1 Tbyte sequential machine

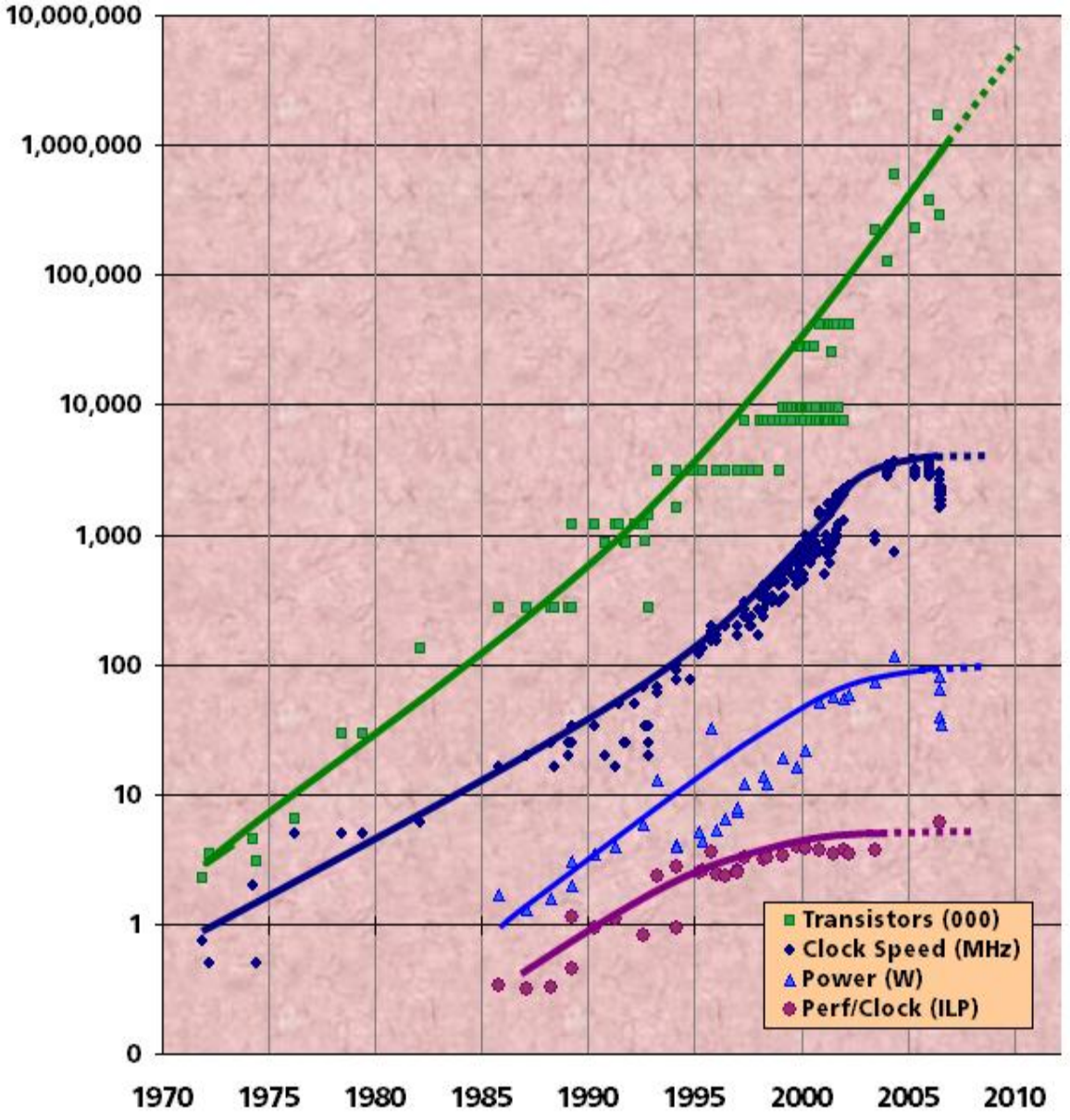


- Consider the 1 Tflop/s sequential machine:
  - Data must travel some distance, r, to go from memory to CPU.
  - To get 1 data element per cycle, this means 10^{12} times per second at the speed of light, $c = 3x10^8$ m/s.
  - Thus $r < c/10^{12} = 0.3$ mm.
- Now put 1 Tbyte of storage in a 0.3 mm x 0.3 mm area:
  - Each bit occupies about 1 square Angstrom, or the size of a small atom
- We are limited by the size of transistors

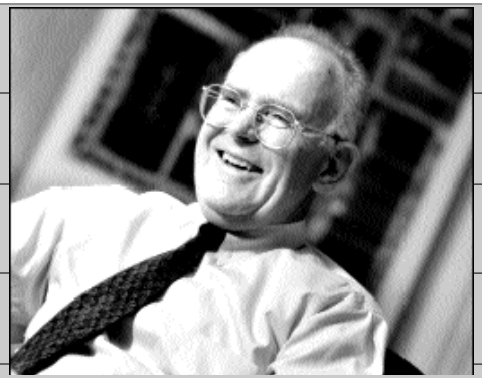
Revolution is happening now

- Chip density is continuing 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
  - No free lunch

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



The trends



2X Transistors/Chip
Every 1.5 Years

Super Scalar/Vector/Parallel

1 PFlop/s (10^{15})
1 TFlop/s (10^{12})
1 GFlop/s (10^{9})
1 MFlop/s (10^{6})
1 KFlop/s (10^{3})

Parallel
IBM BG/L
ASCI White Pacific
ASCI Red
TMC CM-5
Cray T3D

Vector
TMC CM-2
Cray 2
Cray X-MP
Cray 1

Super Scalar
CDC 7600
IBM 360/195
CDC 6600
IBM 7090

Scalar
UNIVAC 1
EDSAC 1

1941 1 (Floating Point operations / second, Flop/s)
1945 100
1949 1,000 (1 KiloFlop/s, KFlop/s)
1951 10,000
1961 100,000
1964 1,000,000 (1 MegaFlop/s, MFlop/s)
1968 10,000,000
1975 100,000,000
1987 1,000,000,000 (1 GigaFlop/s, GFlop/s)
1992 10,000,000,000
1993 100,000,000,000
1997 1,000,000,000,000 (1 TeraFlop/s, TFlop/s)
2000 10,000,000,000,000
2005 131,000,000,000,000 (131 Tflop/s)

1950 1960 1970 1980 1990 2000 2010

Recent multicore processors

- Sept 13: Intel Ivy Bridge-EP Xeon E5-2695 v2
  - 12 cores; 2-way SMT; 30MB cache
- March 13: SPARC T5
  - 16 cores; 8-way fine-grain MT per core
- May 12: AMD Trinity
  - 4 CPU cores; 384 graphics cores
- Nov 12: Intel Xeon Phi coprocessor
  - ~60 cores
- Feb 12: Blue Gene/Q
  - 17 cores; 4-way SMT
- Q4 11: Intel Ivy Bridge
  - 4 cores; 2 way SMT;
- November 11: AMD Interlagos
  - 16 cores
- Jan 10: IBM Power 7
  - 8 cores; 4-way SMT; 32MB shared cache
- Tilera TilePro64

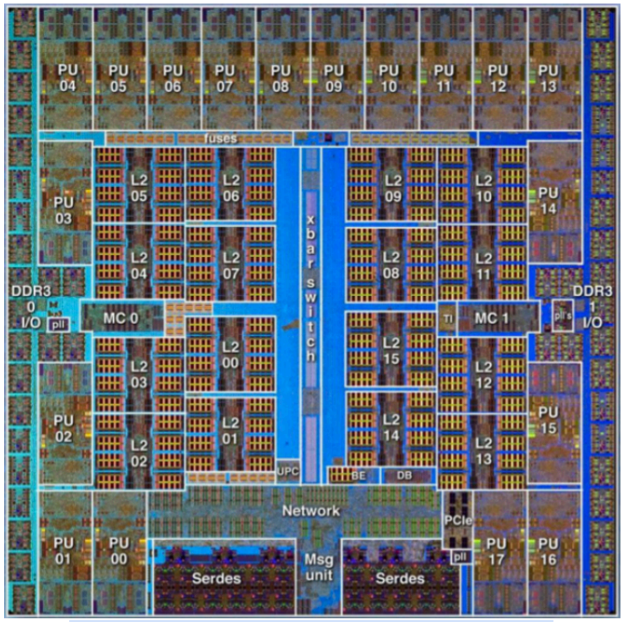
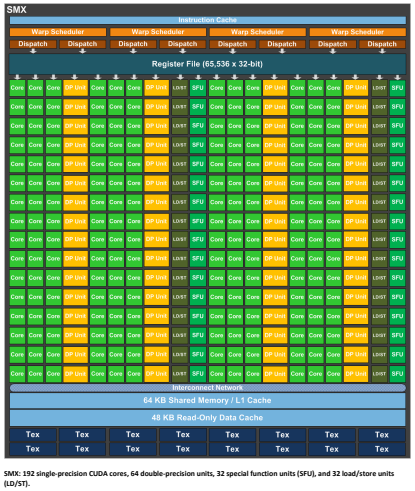


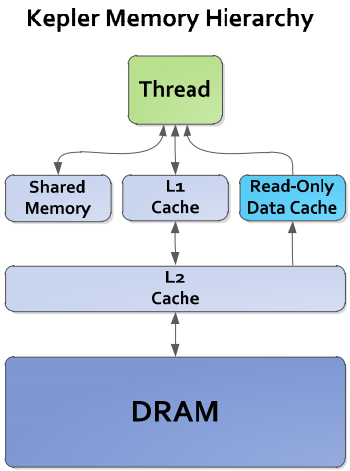
Figure credit: Ruud Haring, Blue Gene/Q compute chip, Hot Chips 23, August, 2011.

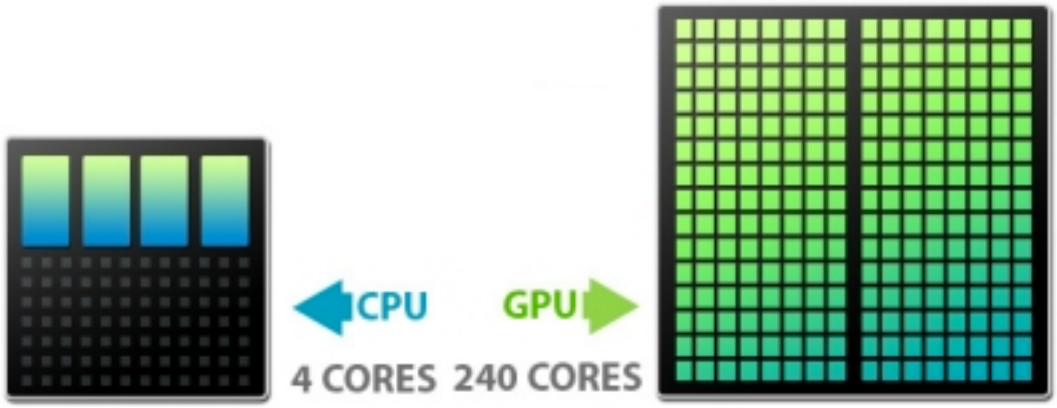
Recent manycore GPU processors

- ~3k cores



SMX: 192 single-precision CUDA cores, 64 double-precision units, 32 special function units (SFU), and 32 load/store units (LD/ST).





Now it's up to the programmers

- Adding more processors doesn't help much if programmers aren't aware of them…
  - … or don't know how to use them.

- Serial programs don't benefit from this approach (in most cases).

We will end up …

Concluding Remarks

- The laws of physics have brought us to the doorstep of multicore technology
  - The worst or the best time to major in computer science
    - IEEE Rebooting Computing (http://rebootingcomputing.ieee.org/)
- Serial programs typically don't benefit from multiple cores.
- Automatic parallelization from serial program isn't the most efficient approach to use multicore computers.
  - Proved not a viable approach
- Learning to write parallel programs involves
  - learning how to coordinate the cores.
- **Parallel programs are usually very complex and therefore, require sound program techniques and development.**

References

- Introduction to Parallel Computing, Blaise Barney, Lawrence Livermore National Laboratory
  - https://computing.llnl.gov/tutorials/parallel_comp
- Some slides are adapted from notes of Rice University John Mellor-Crummey's class and Berkely Kathy Yelic's class.
- Examples are from chapter 01 slides of book "An Introduction to Parallel Programming" by Peter Pacheco
  - Note the copyright notice
- Latest HPC news
  - http://www.hpcwire.com
- World-wide premier conference for supercomputing
  - http://www.supercomputing.org/, the week before thanksgiving week

Vision and Wisdom by Experts

- "I think there is a world market for maybe five computers."
  - Thomas Watson, chairman of IBM, 1943.
- "There is no reason for any individual to have a computer in their home"
  - Ken Olson, president and founder of Digital Equipment Corporation, 1977.
- "640K [of memory] ought to be enough for anybody."
  - Bill Gates, chairman of Microsoft,1981.
- "On several recent occasions, I have been asked whether parallel computing will soon be relegated to the trash heap reserved for promising technologies that never quite make it."
  - Ken Kennedy, CRPC Directory, 1994

Linus: The Whole "Parallel Computing Is The Future" Is A Bunch Of Crock.

WEDNESDAY, DECEMBER 31, 2014 AT 9:09AM

http://highscalability.com/blog/2014/12/31/linus-the-whole-parallel-computing-is-the-future-is-a-bunch.html

Good to Know:

Terminology

- Concurrent computing – a program is one in which multiple tasks can be in progress at any instant.
- Parallel computing – a program is one in which multiple tasks cooperate closely to solve a problem
- Distributed computing – a program may need to cooperate with other programs to solve a problem.

A simple example

- Compute n values and add them together.
- Serial solution:

```
sum = 0;
for (i = 0; i < n; i++) {
   x = Compute_next_value(. . .);
   sum += x;
}
```

Example (cont.)

- We have p cores, p much smaller than n.
- Each core performs a partial sum of approximately n/p values.

```
my_sum = 0;
my_first_i = . . . ;
my_last_i = . . . ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
   my_x = Compute_next_value( . . .);
   my_sum += my_x;
}
```

Each core uses it's own private variables and executes this block of code independently of the other cores.

Example (cont.)

- After each core completes execution of the code, is a private variable my_sum contains the sum of the values computed by its calls to Compute_next_value.

- Ex., 8 cores, n = 24, then the calls to Compute_next_value return:

1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9

Example (cont.)

- Once all the cores are done computing their private my_sum, they form a global sum by sending results to a designated “master” core which adds the final result.

Example (cont.)

```
if (I'm the master core) {
    sum = my_x;
    for each core other than myself {
        receive value from core;
        sum += value;
    }
} else {
    send my_x to the master;
}
```

SPMD: All run the same program, but perform differently depending on who they are.

Example (cont.)

Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

Global sum

8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95

Core	0	1	2	3	4	5	6	7
my_sum	95	19	7	15	7	13	12	14

But wait!

There's a much better way to compute the global sum.

Better parallel algorithm

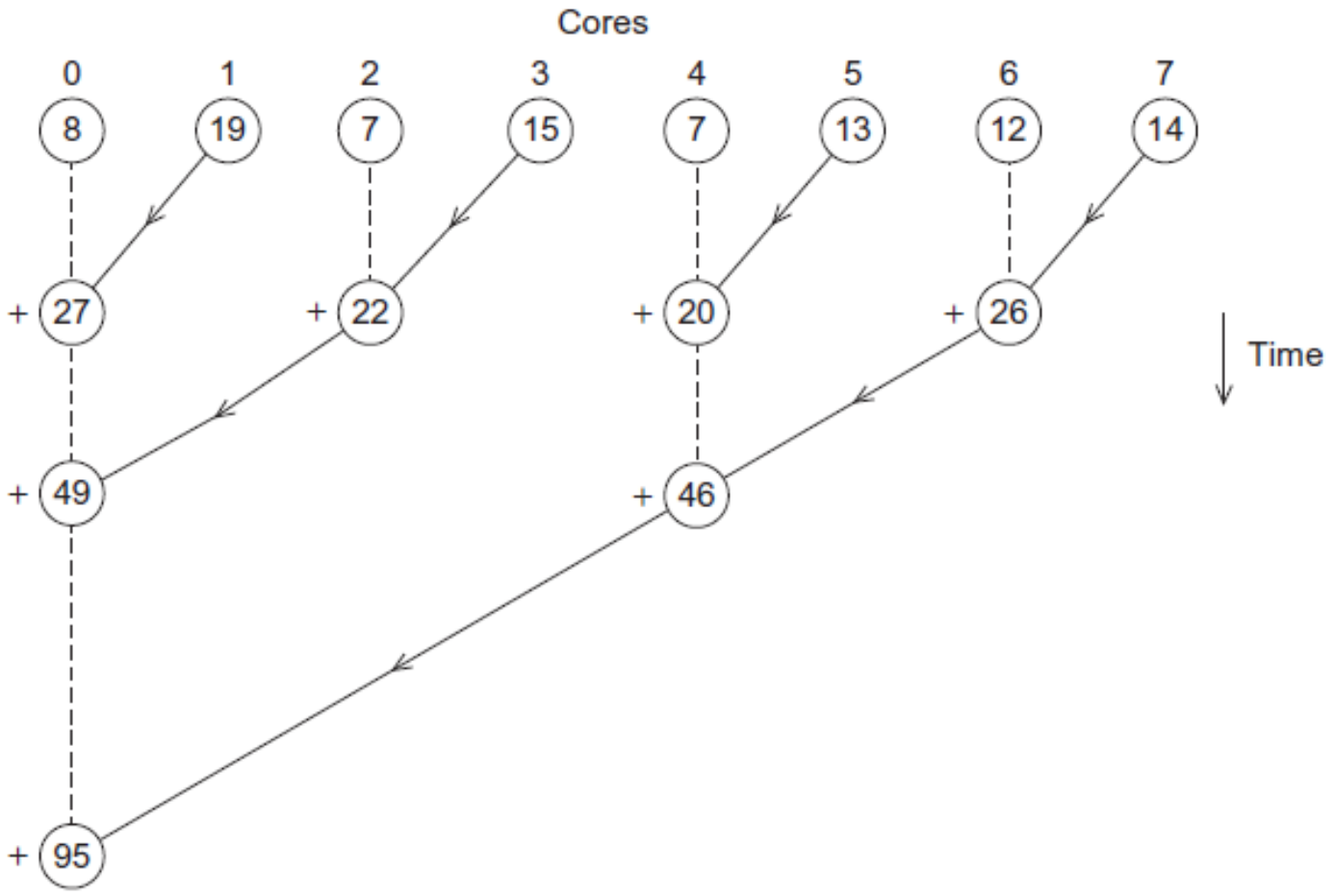
- Don't make the master core do all the work.
- Share it among the other cores.
- Pair the cores so that core 0 adds its result with core 1's result.
- Core 2 adds its result with core 3's result, etc.
- Work with odd and even numbered pairs of cores.

Better parallel algorithm (cont.)

- Repeat the process now with only the evenly ranked cores.
- Core 0 adds result from core 2.
- Core 4 adds the result from core 6, etc.

- Now cores divisible by 4 repeat the process, and so forth, until core 0 has the final result.

Multiple cores forming a global sum



Analysis

- In the first example, the master core performs 7 receives and 7 additions.
- In the second example, the master core performs 3 receives and 3 additions.
- The improvement is more than a factor of 2!

Analysis (cont.)

- The difference is more dramatic with a larger number of cores.
- If we have 1000 cores:
  - The first example would require the master to perform 999 receives and 999 additions.
  - The second example would only require 10 receives and 10 additions.

- That's an improvement of almost a factor of 100!