Multicore Computing
Lecture01 – Introduction to Parallel Computing

SUNG KYUN KWAN UNIVERSITY

남 범 석

bnam@skku.edu

Class Organization

Instructor	Class code	Lecture hours	WebEx	Email
Nam, Beomseok	SWE3021	Mon 12:00 – 13:15 Wed 13:30 – 14:45 #26312	http://skku-ict.webex.com/meet/bnam	bnam@skku.edu

- Q&A
  - http://piazza.com/sungkyunkwan_university/fall2020/swe3021/

- Tentative Grading Policy
  - 2 Exams: 60%
  - 4 programming assignments: 40%

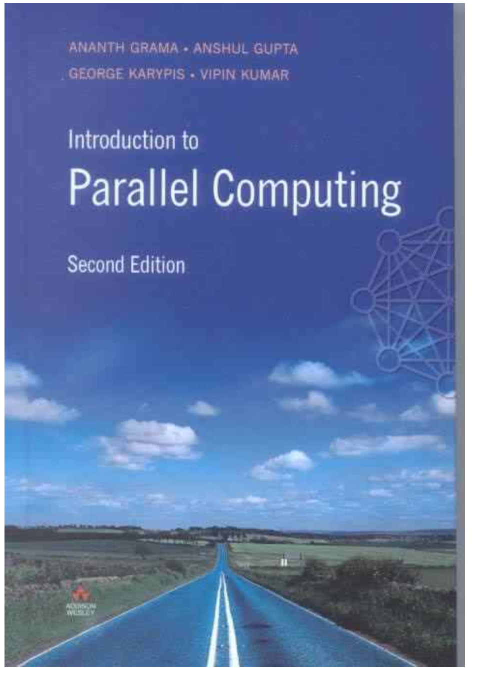
Prerequisites

- Prerequisite Courses
  - Data structures and algorithms (mandatory)
  - System programming (mandatory)
  - Computer Architecture
  - Operating Systems

- Programming Skills
  - Fluent C/C++ programming
  - Programming experience in Linux environment (gcc, gdb, vi/emacs, ...)

Textbooks

- Not Required
  - An Introduction to Parallel Computing
    - A. Grama, A. Gupta, G. Karypis, V. Kumar



Topics

- Parallel computing environment
- Parallel computer architectures
- Parallel programming models and methods
  - Programming with PThreads
  - Programming with OpenMP (shared memory machine)
  - Programming with MPI (distributed memory machine)
  - Programming with CUDA (GPGPU)
- Parallel algorithm design
  - Concurrency, Mutex, Scalability
  - Reasoning about performance
  - Algorithms

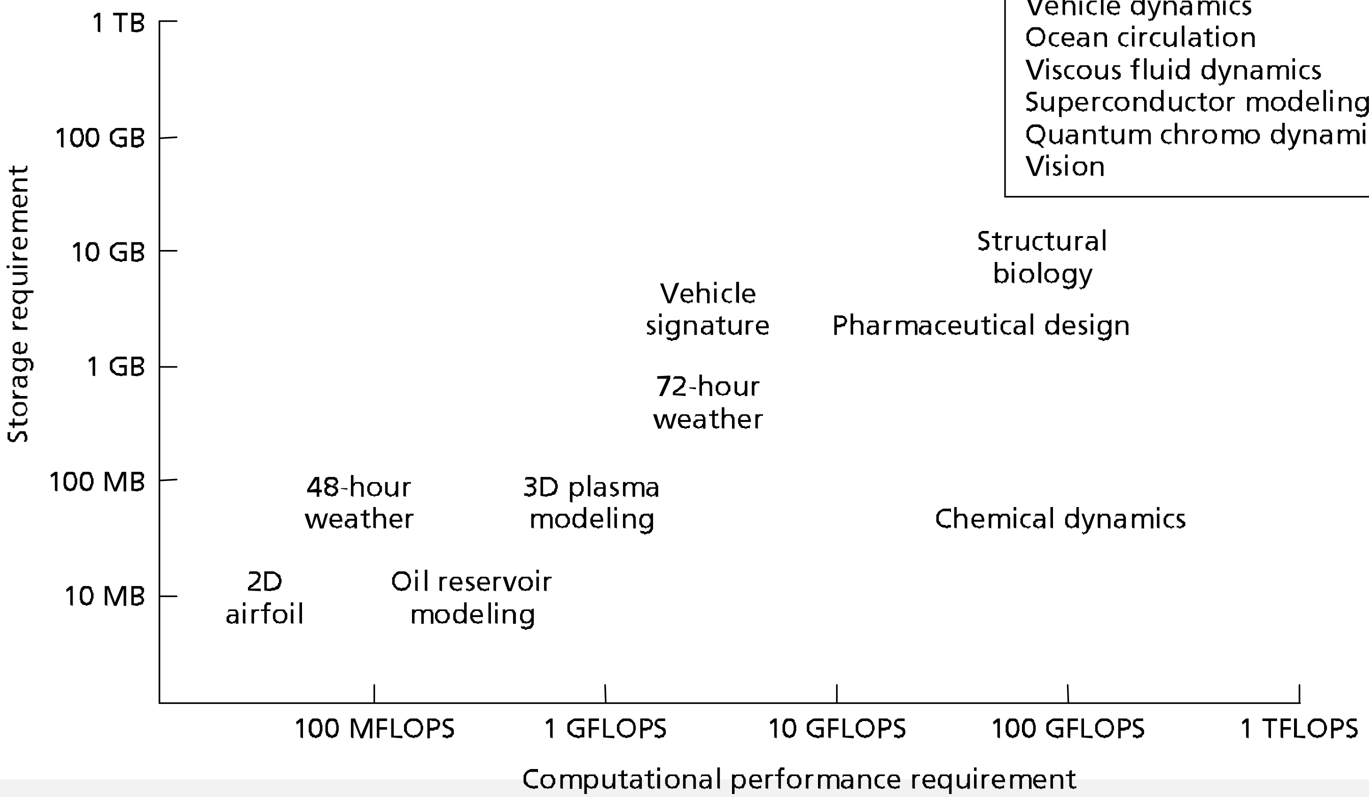
- Parallel computing is a type of computation in which many calculations or the execution of processes are carried out simultaneously
- A parallel computer is a collection of processing elements that cooperate to solve large problems fast

- Different forms of parallelism
  - Call center
    - Calls are generally independent
    - Can be serviced in any order with little interaction among the workers
  - House construction
    - Some tasks can be simultaneously performed (e.g., wiring & plumbing)
    - Some are ordered (e.g., framing must precede wiring)
  - Juggling
    - Event driven: a falling ball ➔ catching & throwing

- Such familiar forms of parallelism also arise in parallel computation

- Q: Why study parallel computing?
- Old days…
  - Scientific Computing Demand
  - → Supercomputing

Grand Challenge problems
Global change
Human genome
Fluid turbulence
Vehicle dynamics
Ocean circulation
Viscous fluid dynamics
Superconductor modeling
Quantum chromo dynamics
Vision



Example: N-body Simulation

- Simulation of a dynamical system of particles
  - N particles (location, mass, velocity)
  - Influence of physical forces (e.g., gravity)
  - Physics, astronomy, …

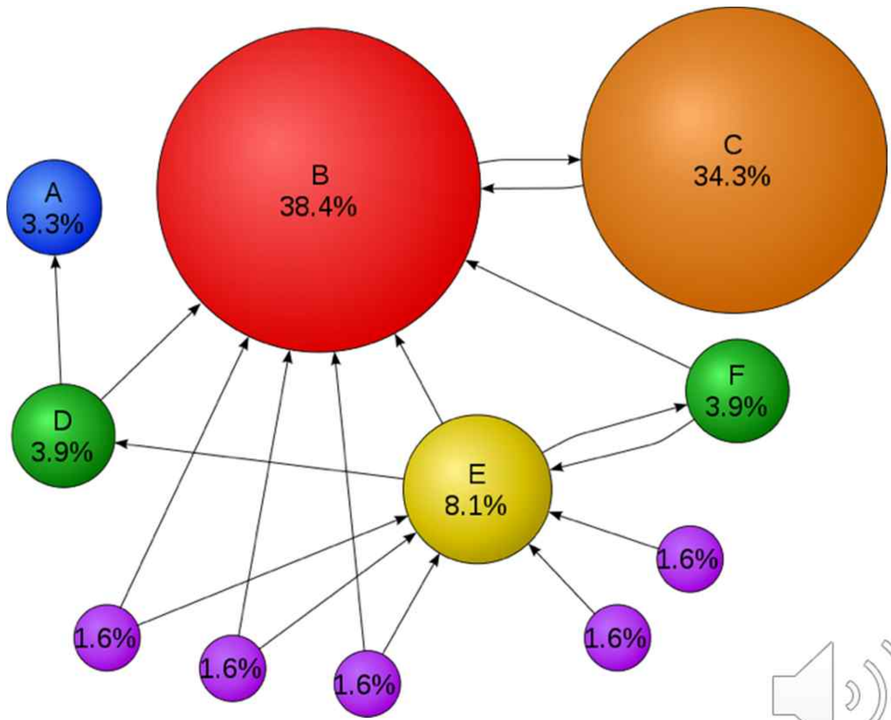
- Q: Why study parallel computing?
- Today
  - Technological convergence
    - Laptops and supercomputers are getting similar
  - Many interesting applications demand high performance
  - CPU clock rates are no longer increasing!
  - Instruction-level parallelism is not increasing either!
  - Parallel computing is the only way to achieve higher performance in the foreseeable future

Example: Graph Computation

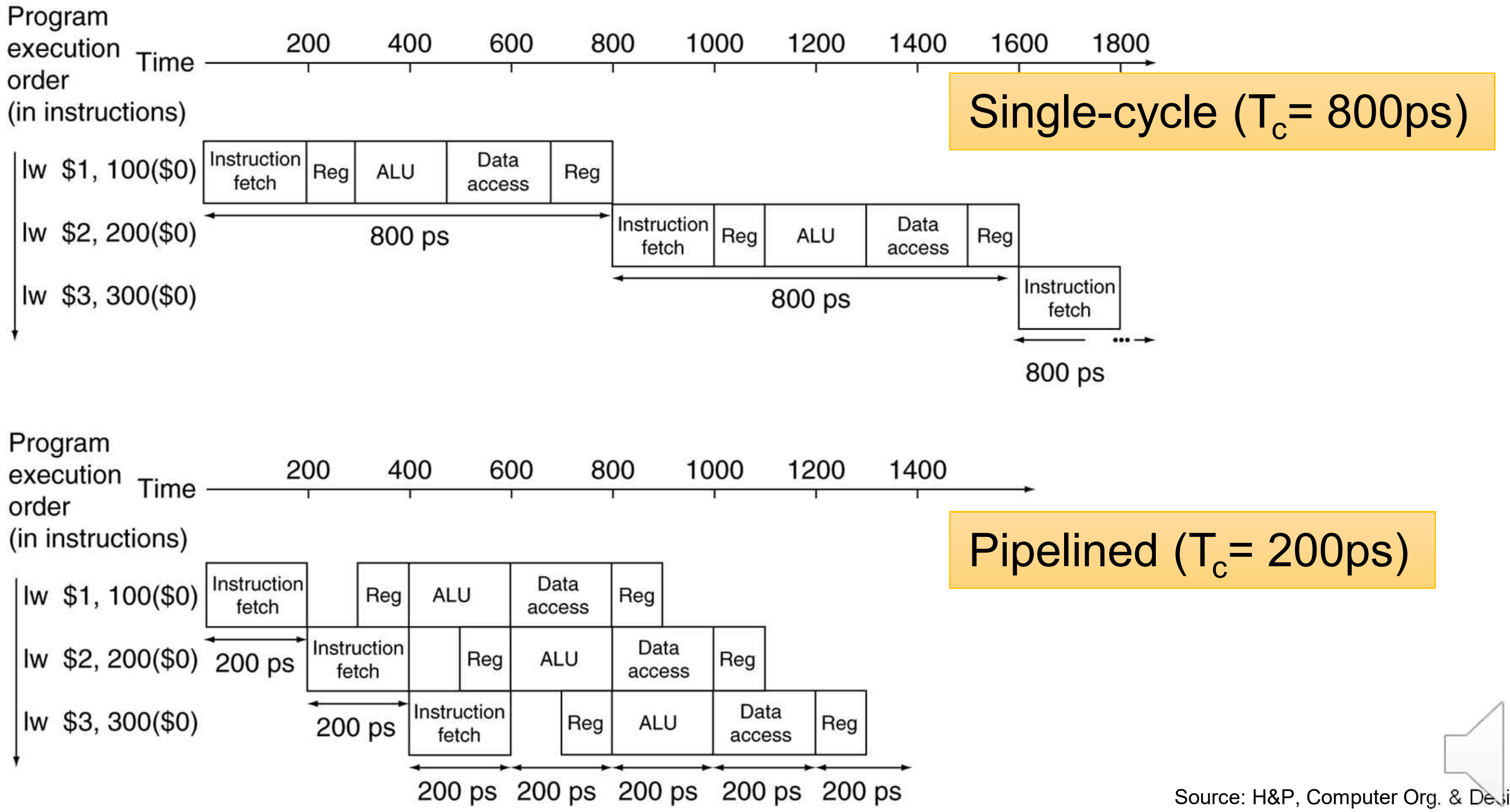
- Google page-rank algorithm
  - Determining the importance of a web page by counting the number and quality of likes to the page
- Relative importance of a web page u
  - Repeat until stabilized

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)},$$

  - Trillion pages in Web



- Parallelism is transparently available to programmers
  - With no effort for parallelization ➔ "Free Lunch!" ☺
  - Instruction Level Parallelism (ILP)
    - Also called as "hidden parallelism"

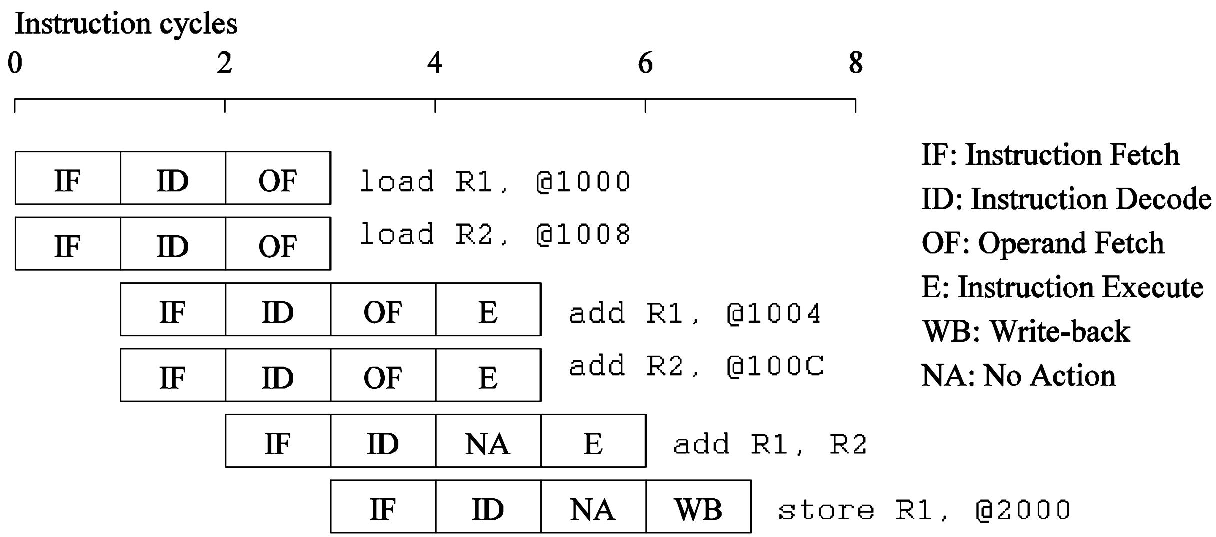


Source: H&P, Computer Org. & Design, 5th ed.

Superscalar Execution

- Exploits instruction-level parallelism
  - With N pipelines, N instructions can be issued (executed) in parallel

1. Load R1, @1000
2. Load R2, @1008
3. Add R1, @1004
4. Add R2, @100C
5. Add R1, R2
6. Store R1, @2000

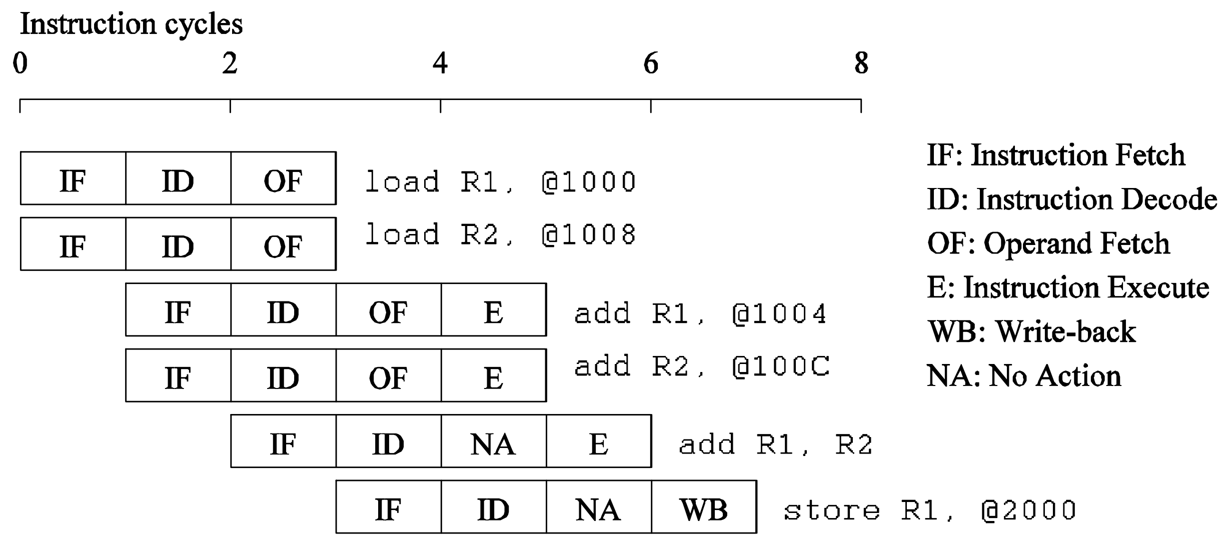


2-way superscalar execution pipeline

Out-of-order Execution

- Exploits instruction-level parallelism
  - With N pipelines, N instructions can be issued (executed) in parallel
  - Reorder independent instructions

1. Load R1, @1000
2. Load R2, @1008
3. Add R1, @1004
4. Add R2, @100C
5. Add R1, R2
6. Store R1, @2000



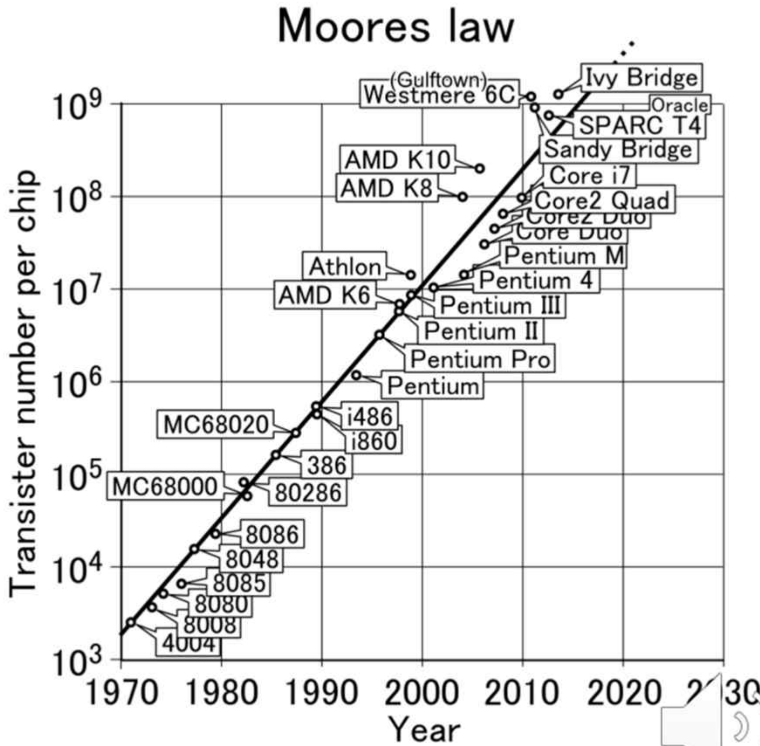
2-way superscalar execution pipeline

Moore's Law

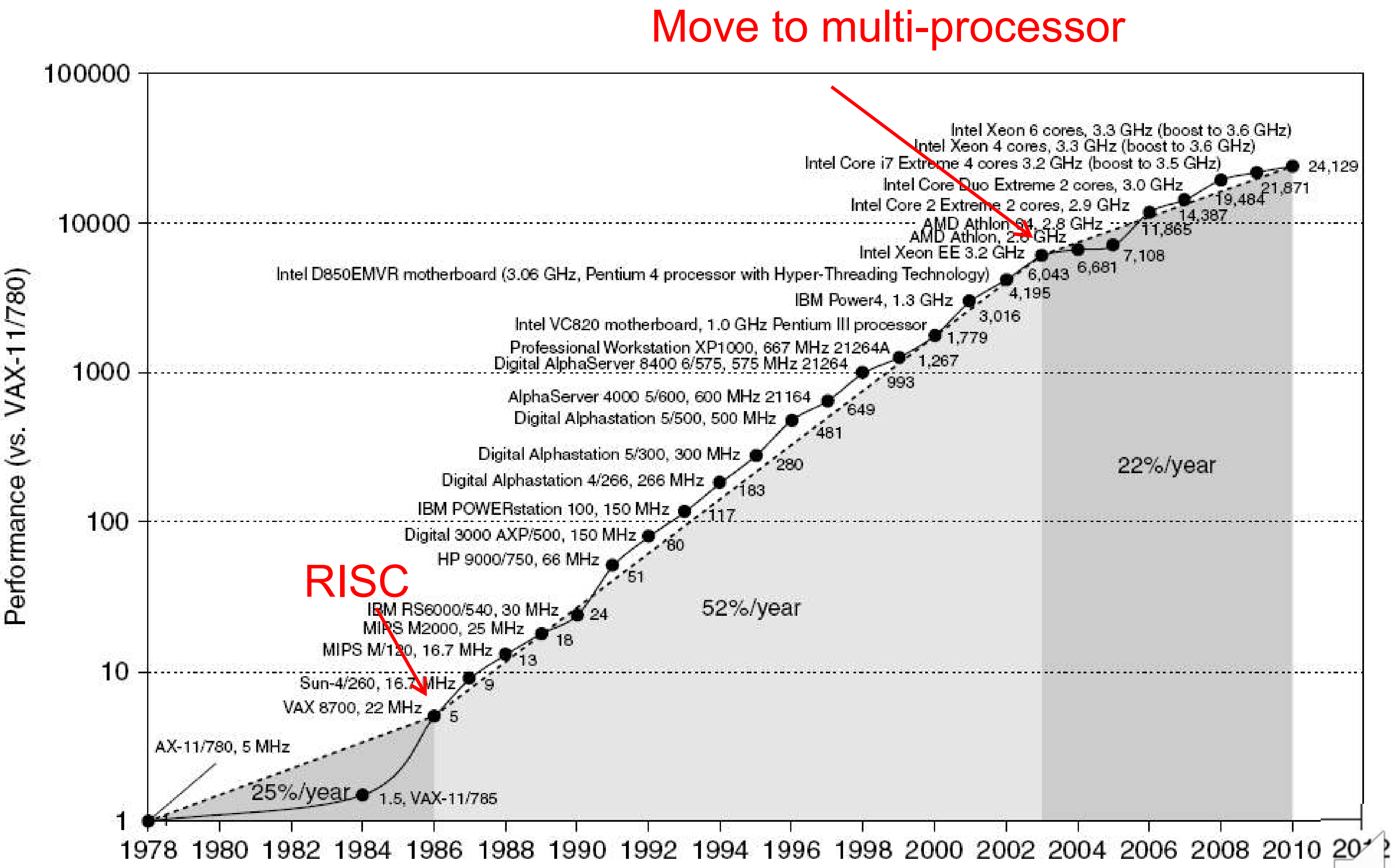
- From 1986 – 2002, microprocessors were speeding like a rocket, increasing in performance an average of 50% per year.
  - the number of transistors/inch2 doubles every 1.5 years

- Since then, it's dropped to about 20% increase per year.

Gordon Moore
Co-founder of Intel co.



Single Thread Performance Curve

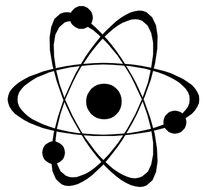


Source: H&P, Computer Architecture, 5th ed.

Evolution of Intel Microprocessors

- Up to now, performance increases have been attributable to increasing density of transistors.

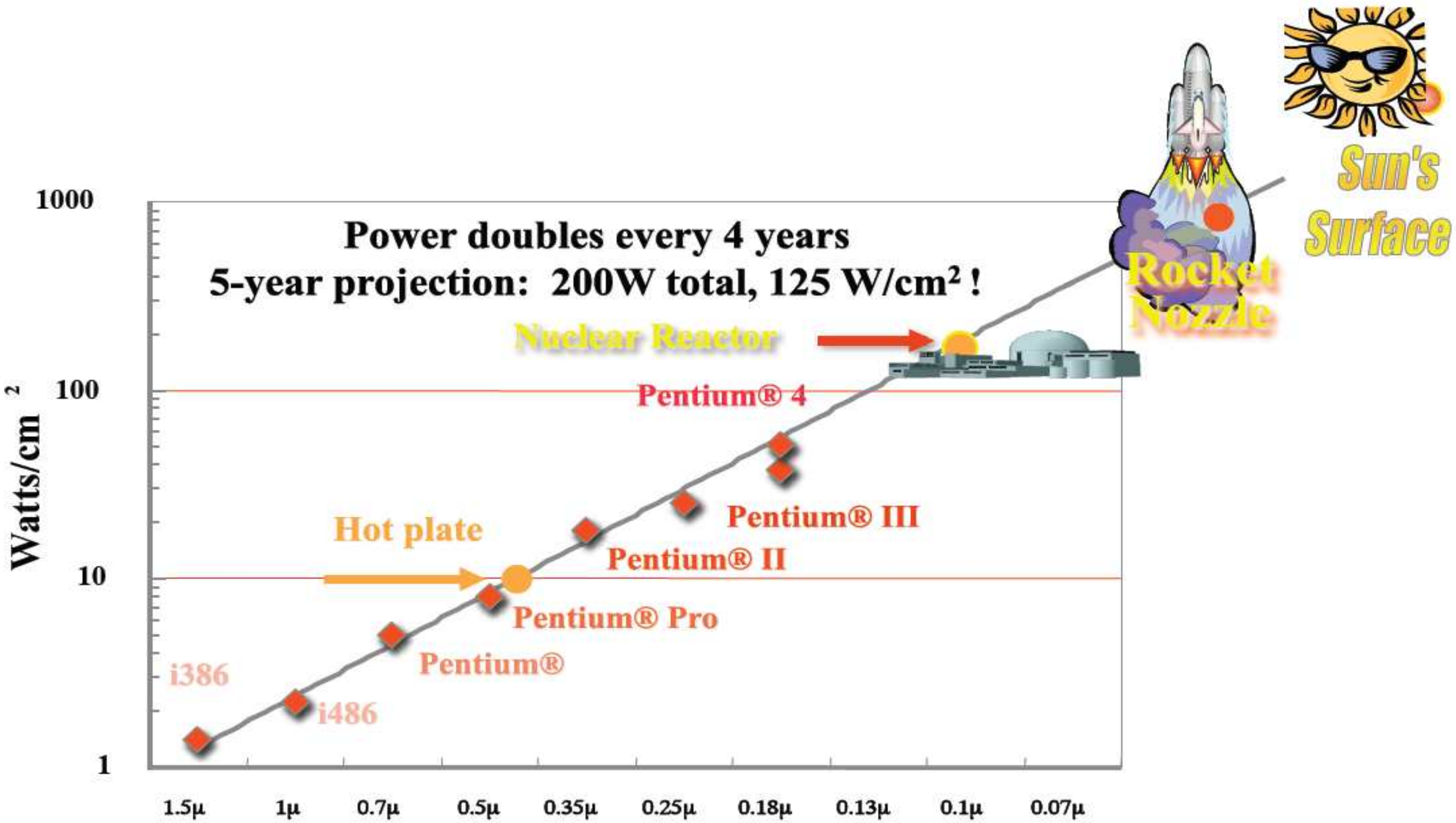
- But there are inherent problems.



- A little physics lesson
  - Smaller transistors = faster processors.
  - Faster processors = increased power consumption.
  - Increased power consumption = increased heat.
  - Increased heat = unreliable processors.

Power/Energy Efficiency

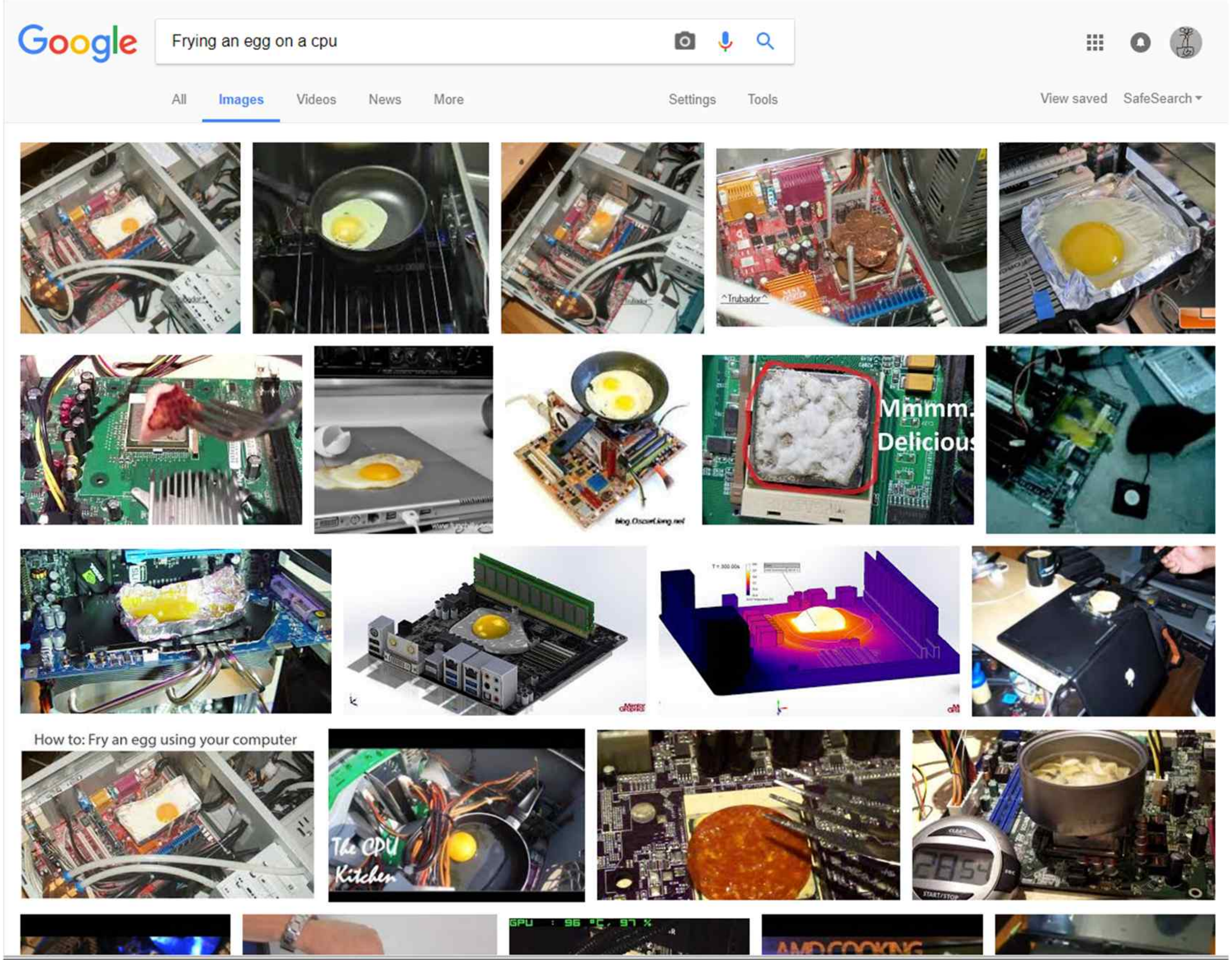
- Heat is becoming an unmanageable problem



From "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies"
– Fred Pollack, Intel Corp. Micro32 conference key note - 1999.

Source: M. Taylor, D '12

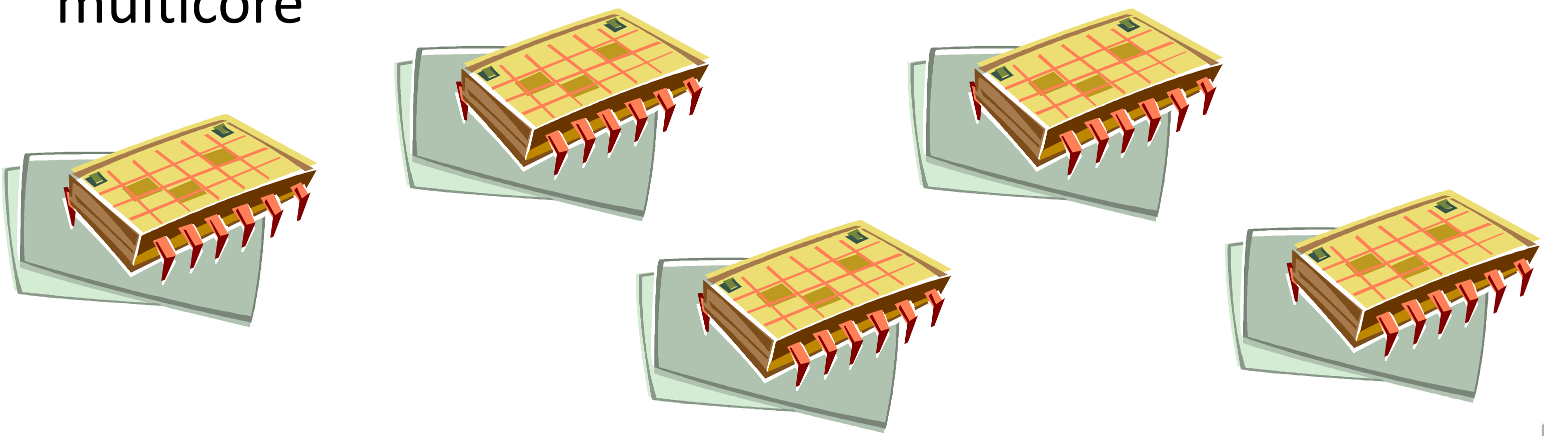
Frying an egg on a CPU



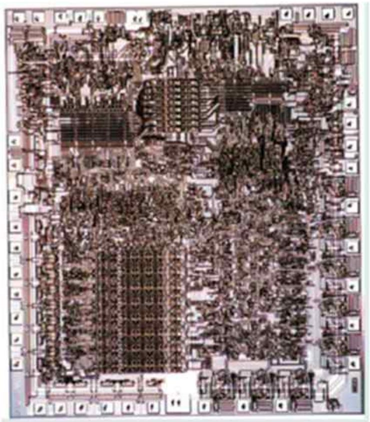
Evolution of Intel Microprocessors

- Move away from single-core systems to multicore processors.

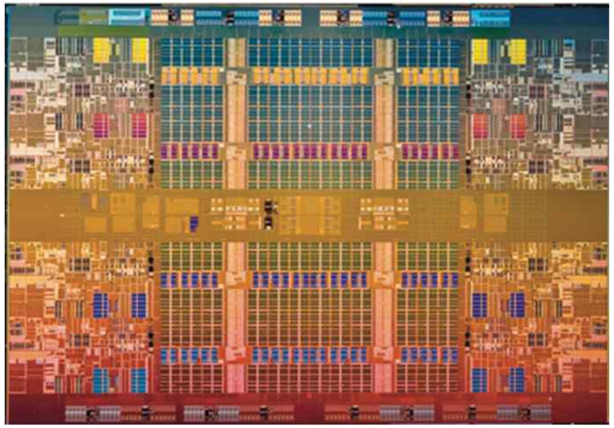
- Instead of designing and building faster microprocessors, put multiple processors on a single integrated circuit.
  - Recent trends: back to simplified microarchitectures + multicore



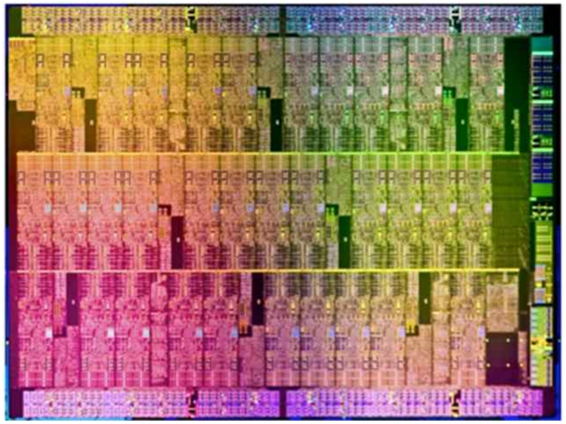
Evolution of Intel Microprocessors



Intel 8088, 1978
1 core, no cache
29K transistors



Intel Nehalem-EX, 2009
8 cores, 24MB cache
2.3B transistors



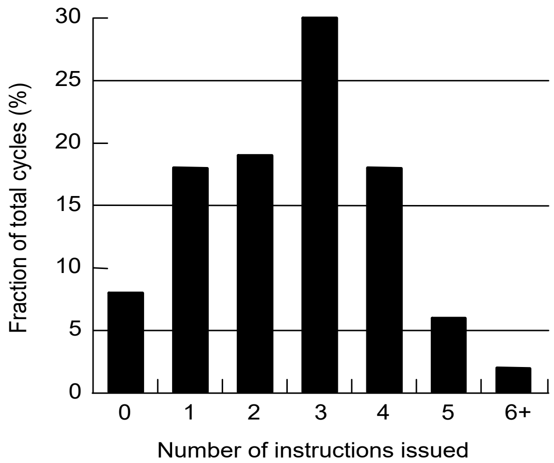
Intel knight landing, 2016
72 cores, 16GB DDR3 LLC
7.1B transistors

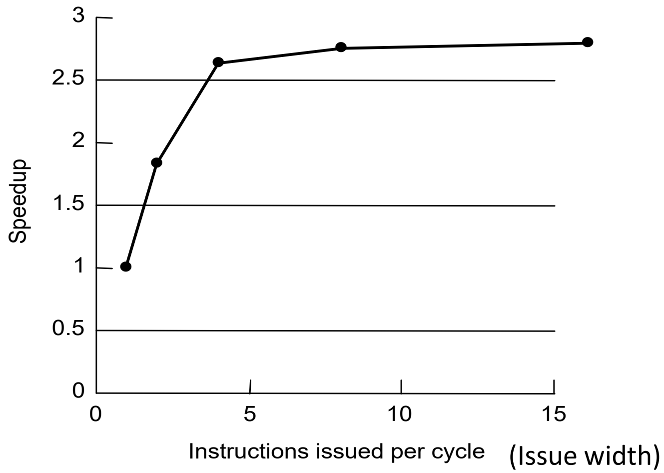
Figure credits: ExtremeTech,
The future of microprocessors, Communications of the ACM, vol. 54, no.5

Limitations of ILP

- Mid 80s to mid 90s
  - Pipelining and simple instruction sets + compiler advances (RISC)

- Limitations of ILP
  -





- Speedup begins to saturate after issue width of 4
  - resources and fetch bandwidth
  - branch prediction
  - Cache miss latencies

Why we need to write parallel programs

- Running multiple instances of a serial program often isn't very useful.
  - Do we need octa-core CPU in our cell phones?
  - Is a parallel application a concurrent application?
  - Can a parallel application run on a single processor?

- ARM big.LITTLE
  - A heterogeneous CPU, coupling battery-saving and slower processor cores (LITTLE) with powerful and power-hungry ones (big).

Cortex A57/A53

Concurrency and Parallelism

- Concurrent is not the same as parallel! Why?
- Concurrent execution
  - Time sharing, i.e., alternate multiple threads on a single processor
  - or execute in parallel on multiple processors – convenient design
- 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

Approaches to the serial problem

- What you really want is to run a program faster.
- Rewrite serial programs so that they're parallel.
- Write translation programs that automatically convert serial programs into parallel programs.
  - This is very difficult to do.
  - Success has been limited.

More problems

- Some coding constructs can be recognized by an automatic program generator, and converted to a parallel construct.
- However, it's likely that the result will be a very inefficient program.
- Sometimes the best parallel solution is to step back and devise an entirely new algorithm.

How do we write parallel programs?

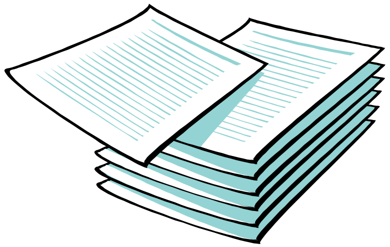
- Task parallelism
  - Partition various tasks carried out solving the problem among the cores.

- Data parallelism
  - Partition the data used in solving the problem among the cores.
  - Each core carries out similar operations on it's part of the data.

Professor P

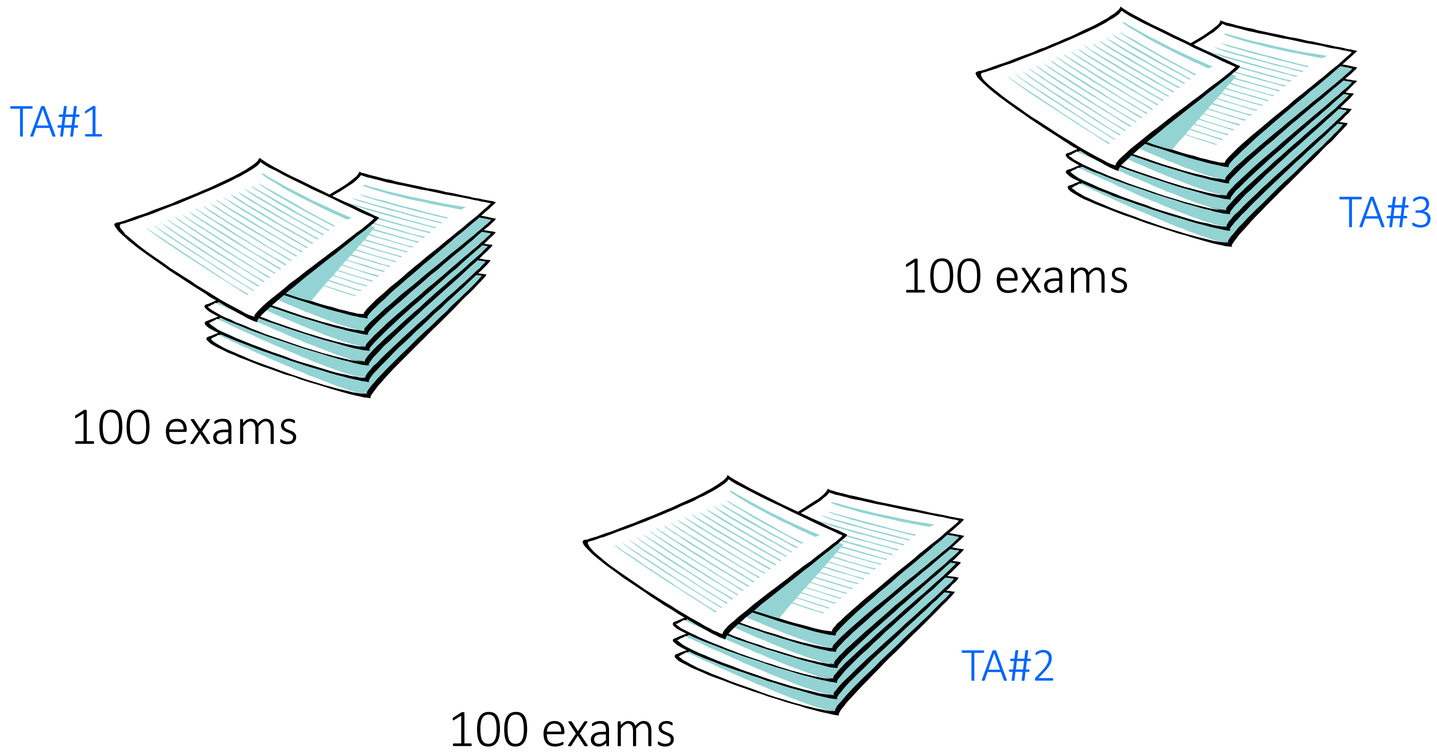
15 questions

300 exams



Professor P’s grading assistants

Division of work – data parallelism



Division of work – task parallelism

TA#1

Questions 1 - 5

TA#3

Questions 11 - 15

Questions 6 - 10

TA#2

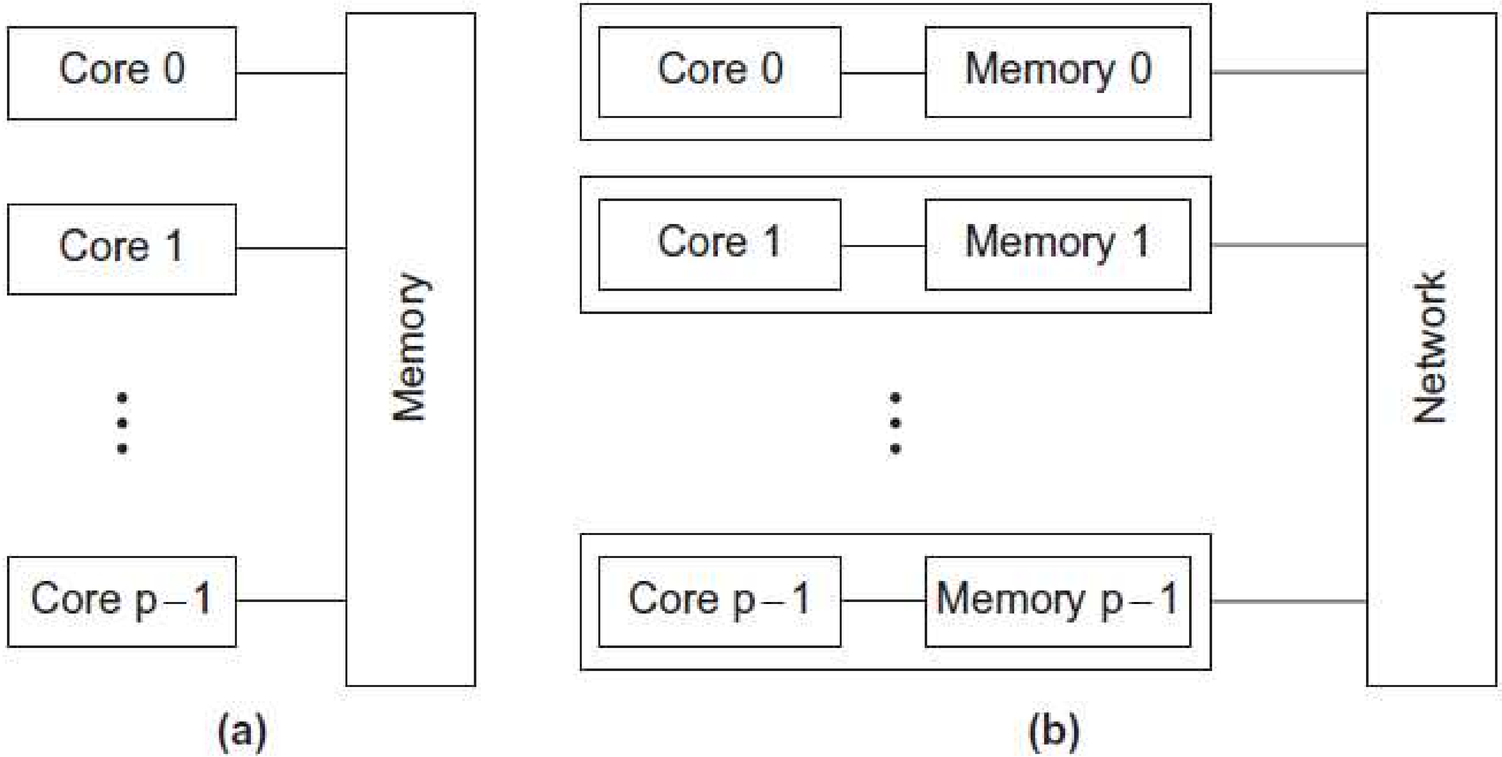
Coordination

- Cores usually need to coordinate their work.
  - Learning to write parallel programs involves learning how to coordinate the cores.
- Communication – one or more cores send their current partial sums to another core.
- Load balancing – share the work evenly among the cores so that one is not heavily loaded.
- Synchronization – because each core works at its own pace, make sure cores do not get too far ahead of the rest.

Type of parallel systems

- Shared-memory
  - The cores can share access to the computer’s memory.
  - Coordinate the cores by having them examine and update shared memory locations.
- Distributed-memory
  - Each core has its own, private memory.
  - The cores must communicate explicitly by sending messages across a network.

Type of parallel systems

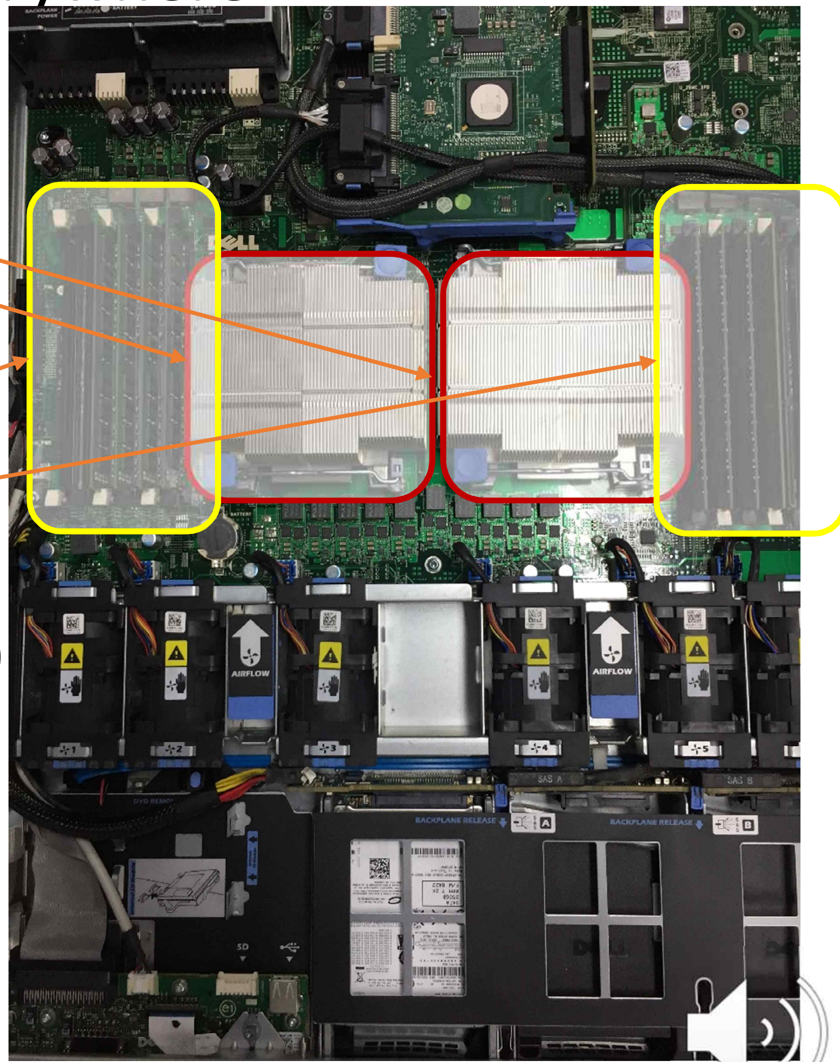


Shared-memory

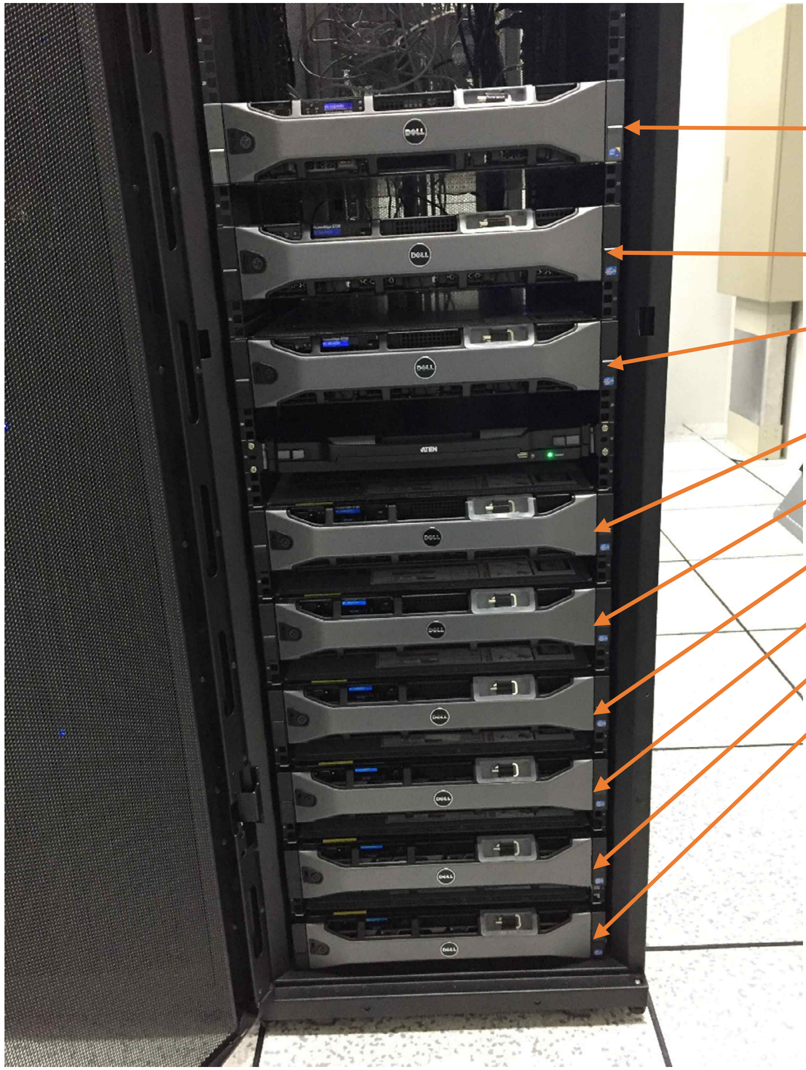
Distributed-memory

Shared Memory Machine

- Multi-core processors are everywhere
  - Dual Intel Xeon CPU
  - NUMA
    (Non-Uniform Memory Access)



Distributed Memory Machine (Cluster)



Master (NFS, NIS)

Slave nodes

Interconnect:
Infiniband 100G

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.

Concluding Remarks

- Parallel computing is a must!
  - To continue improving the performance of applications
- Parallel programming is not easy
  - To achieve all the desired properties such as correctness and performance
- Given the complexity of parallel computing and a wide range of parallel hardware, we need a small set of fundamental principles for parallel programming
  - To achieve high performance, scalability, and performance portability
  - We will learn these principles throughout this course