计算机体系结构

周学海

xhzhou@ustc.edu.cn

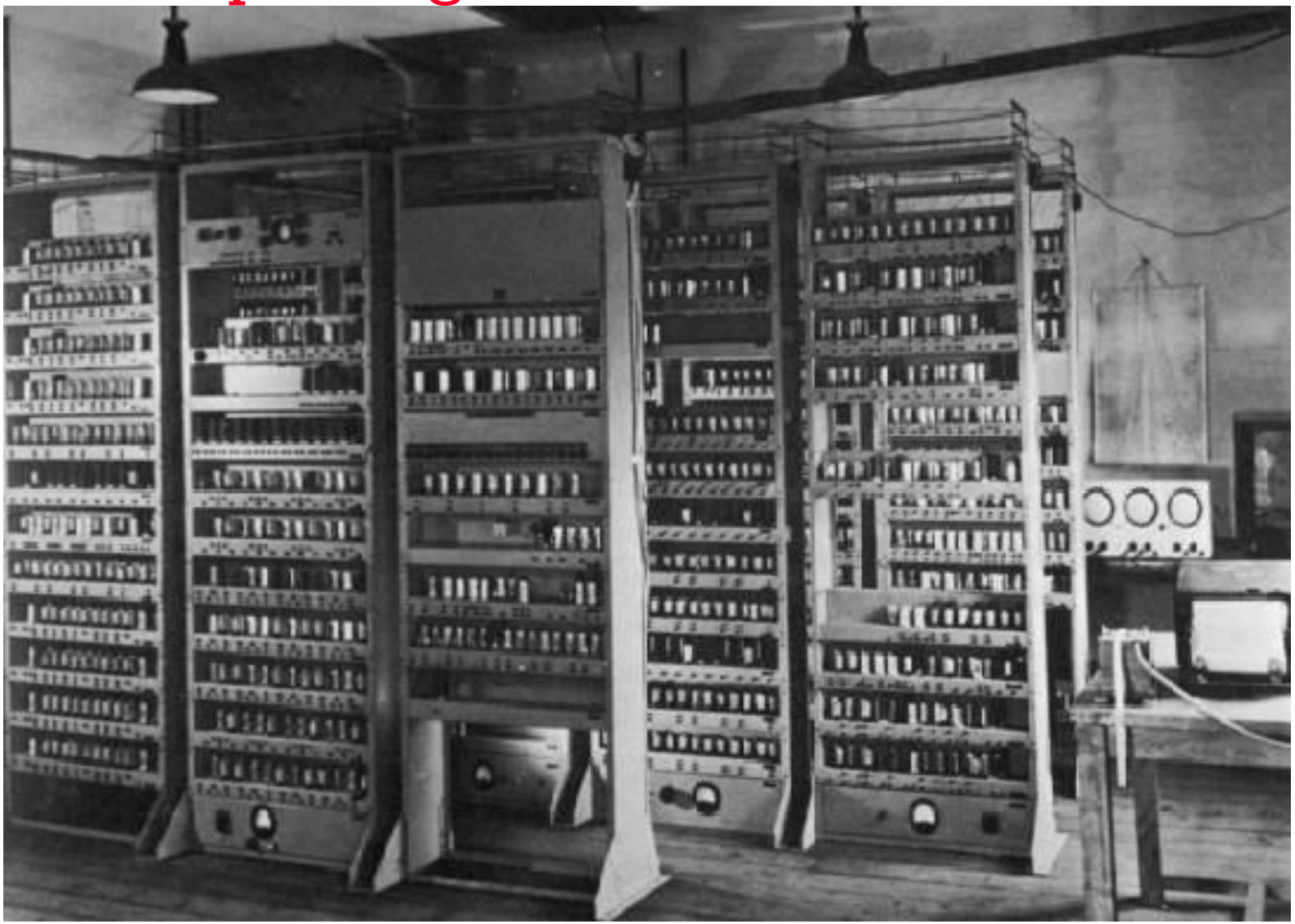
0551-63601556, 63492271

中国科学技术大学

Chapter1 量化设计与分析基础

- **1.1** 引言
  - 计算机的分类
  - 计算机体系结构的定义
  - 现代计算机系统发展趋势
- **1.2** 定量分析基础

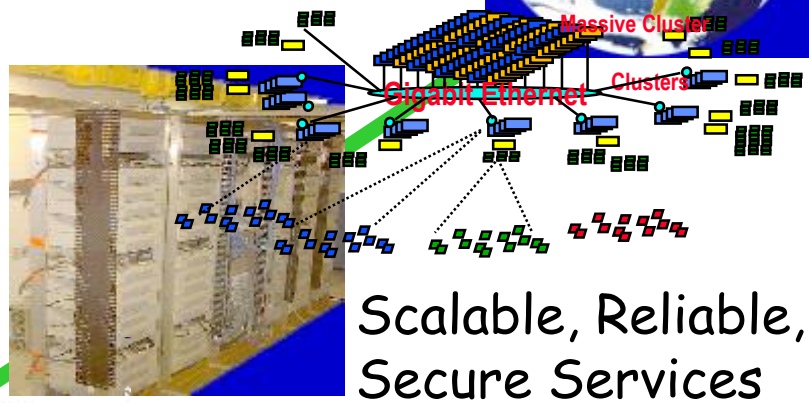
Computing Devices Then•••



EDSAC, University of Cambridge, UK, 1949

Computing Systems Today

- **The world is a large parallel system**
  - **Microprocessors in everything**
  - **Vast infrastructure behind them**



Internet Connectivity

Scalable, Reliable, Secure Services

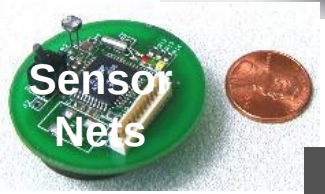
Databases

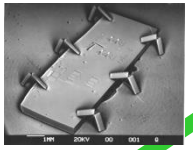
Information Collection

Remote Storage

Online Games

Commerce





MEMS for Sensor Nets

计算机的分类

- 个人移动设备 (PMD)
  - e.g. smart phones, tablet computers
  - >1 billion sold/year
  - Market dominated by ARM-ISA-compatible general-purpose processor in system-on-a-chip (SoC)
  - Plus sea of custom accelerators (radio, image, video, graphics, audio, motion, location, security, etc.)
  - Emphasis on energy efficiency and real-time
- 桌面计算（Desktop Computing）
  - Emphasis on price-performance
- 服务器（Servers）
  - Emphasis on availability, scalability, throughput

计算机的分类（续）

- 集群/仓库级计算机（**Clusters / Warehouse Scale Computers**）
  - **100,000's cores per warehouse**
  - **Market dominated by x86-compatible server chips**
  - **Dedicated apps, plus cloud hosting of virtual machines**
  - **Starting to see some GPU usage, but mostly general-purpose CPU code**
  - **Used for "Software as a Service (SaaS)"**
  - **Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks**
  - **Emphasis on availability and price-performance**
- 嵌入式计算机（**Embedded Computers**）
  - **Wired/wireless network infrastructure, printers**
  - **Consumer TV/Music/Games/Automotive/Camera/MP3**
  - **Emphasis: price**

并行及并行体系结构

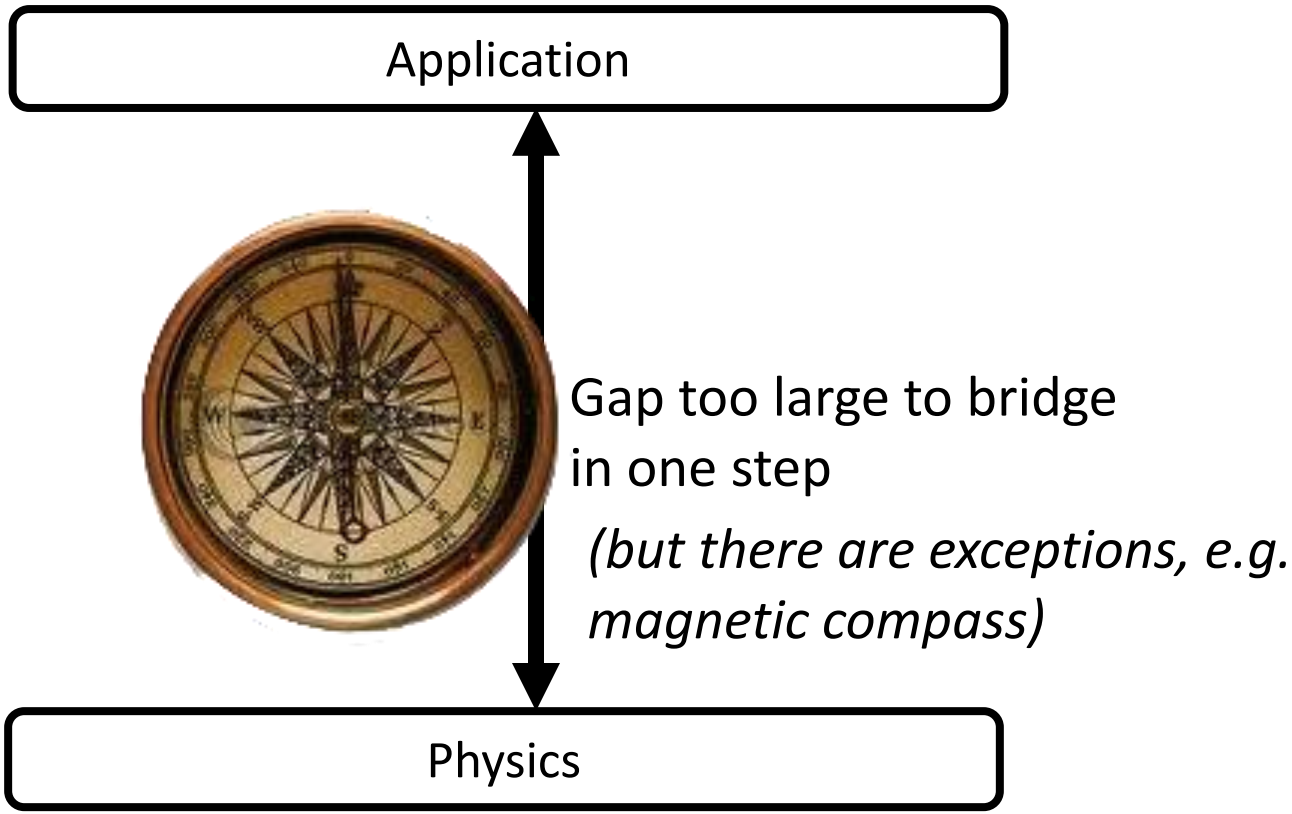
- 应用程序中的并行:
  - **Data-Level Parallelism (DLP)**
  - **Task-Level Parallelism (TLP)**

- 硬件挖掘应用程序的**DLP**或**TLP**的方式)
  - **Instruction-Level Parallelism (ILP)**
  - **Vector architectures/Graphic Processor Units (GPUs)**
  - **Thread-Level Parallelism**
  - **Request-Level Parallelism**

Flynn’s Taxonomy

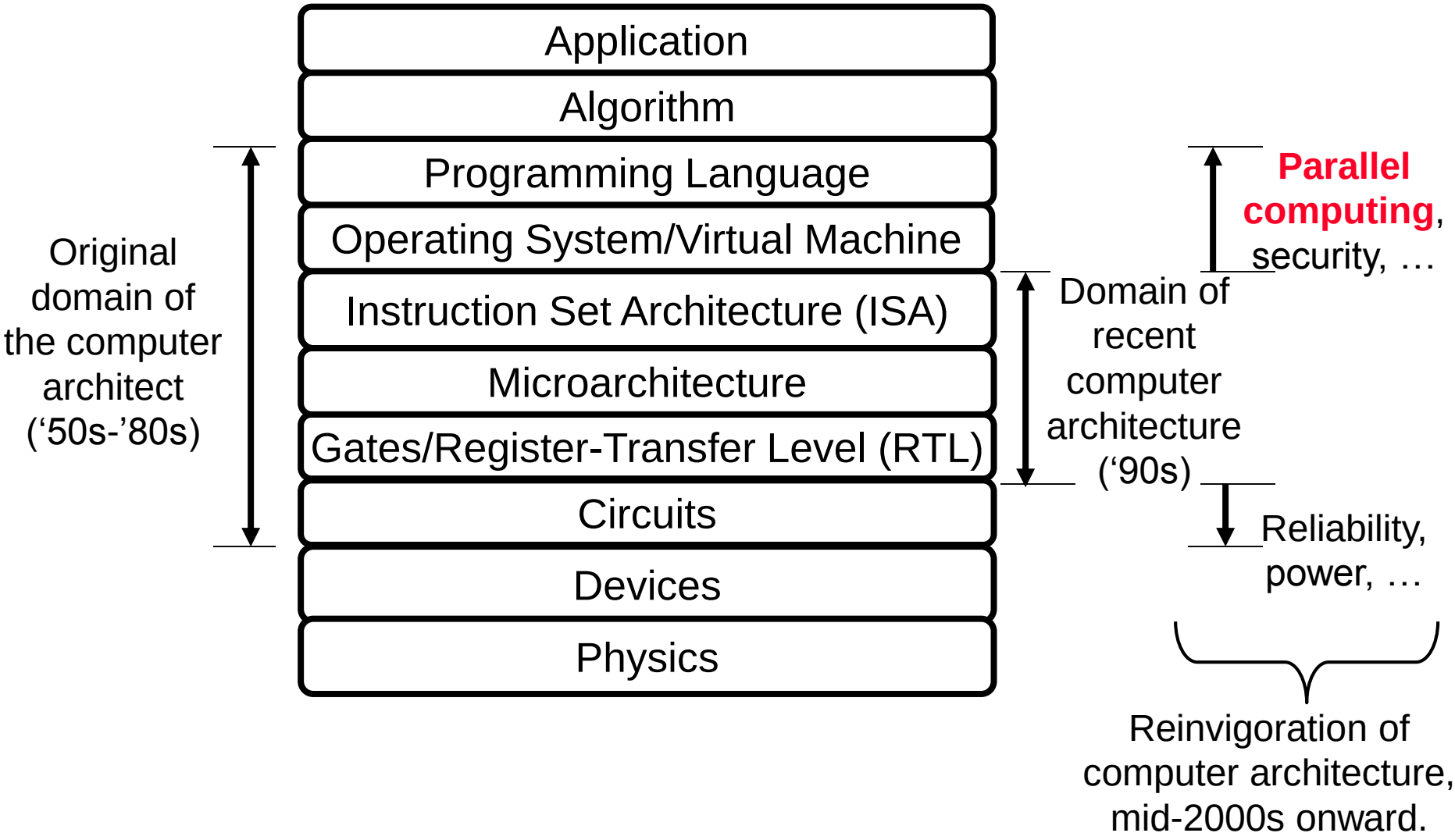
- 单指令流，单数据流(SISD)
- 单指令流，多数据流 (SIMD)
  - Vector architectures
  - Multimedia extensions
  - Graphics processor units
- 多指令流，单数据流 (MISD)
  - No commercial implementation
- 多指令流，多数据流 (MIMD)
  - Tightly-coupled MIMD
  - Loosely-coupled MIMD

计算机体系结构的定义？



In its broadest definition, computer architecture is the *design of the abstraction layers* that allow us to implement information processing applications efficiently using available manufacturing technologies.

现代计算机系统的抽象层次



计算机体系结构的定义

... the attributes of a [computing] system as seen by the programmer, *i.e.* the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls， the logic design, and the physical implementation.

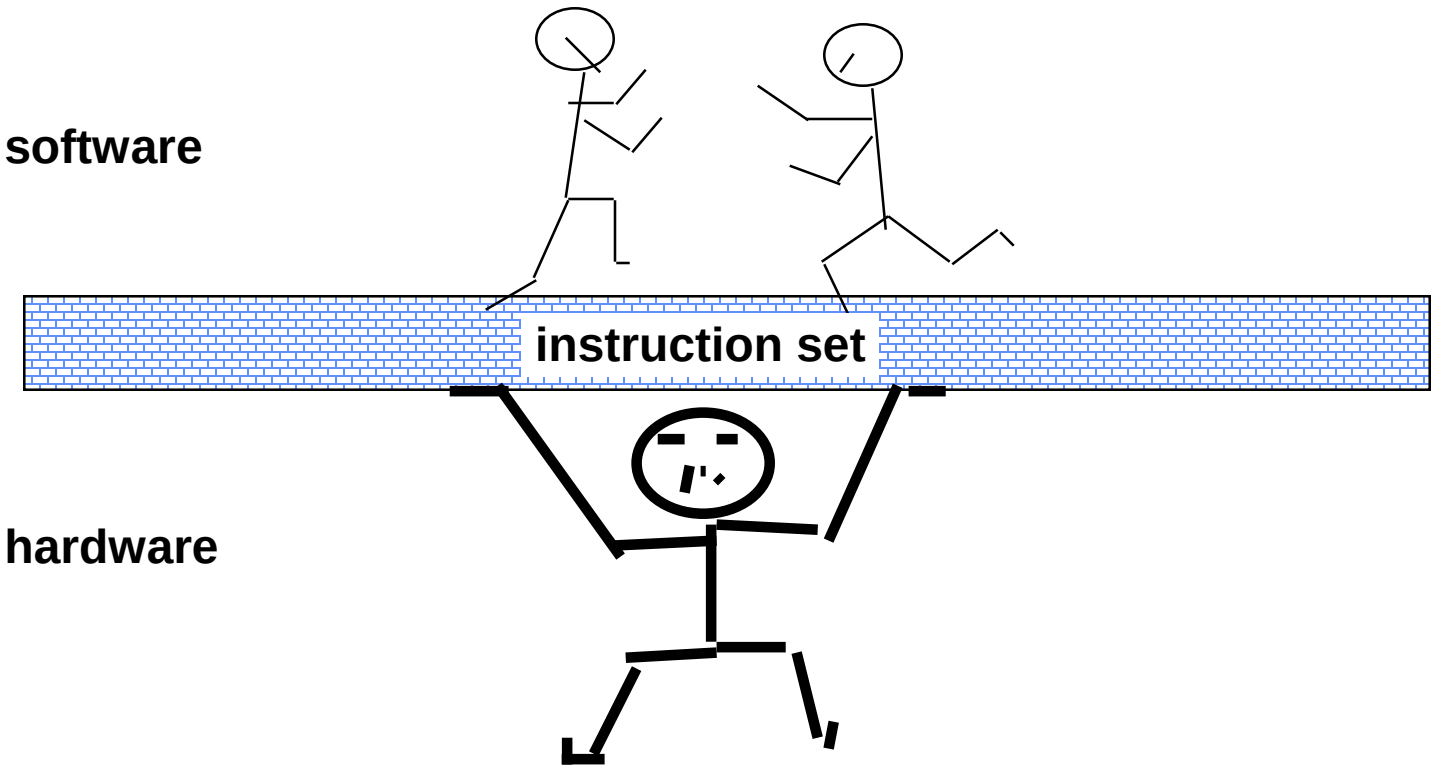
– Amdahl, Blaaw, and Brooks, 1964

计算机体系结构的定义（续）

- **“Old” view of computer architecture:**
  - **Instruction Set Architecture (ISA) design**
  - **i.e. decisions regarding:**
    - » **registers, memory addressing, addressing modes, instruction operands, available operations, control flow instructions, instruction encoding**

- **“Real” computer architecture:**
  - **Specific requirements of the target machine**
  - **Design to maximize performance within constraints: cost, power, and availability**
  - **Includes ISA, microarchitecture, hardware**

ISA: a Critical Interface



ISA需说明的主要内容

- Memory addressing
- Addressing modes
- Types and sizes of operands
- Operations
- Control flow instructions
- Encoding an ISA
- ……
- Properties of a good abstraction
  - Lasts through many generations (portability)
  - Used in many different ways (generality)
  - Provides convenient functionality to higher levels
  - Permits an efficient implementation at lower levels

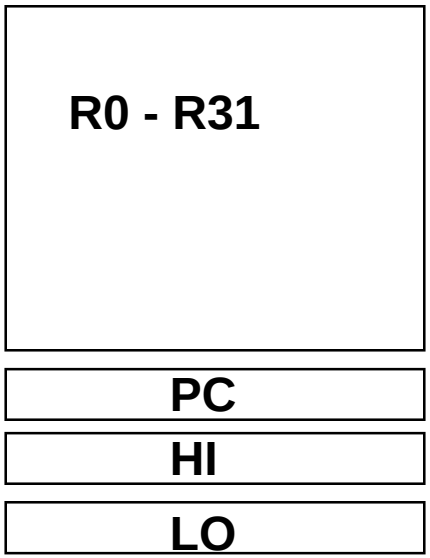
指令集结构举例

- **Digital Alpha (v1, v3) 1992-97**
- **HP PA-RISC (v1.1, v2.0) 1986-96**
- **Sun Sparc (v8, v9) 1987-95**
- **SGI MIPS (MIPS I, II, III, IV, V) 1986-96**
- **Intel (8086,80286,80386, 80486,Pentium, MMX, ...) 1978-96**

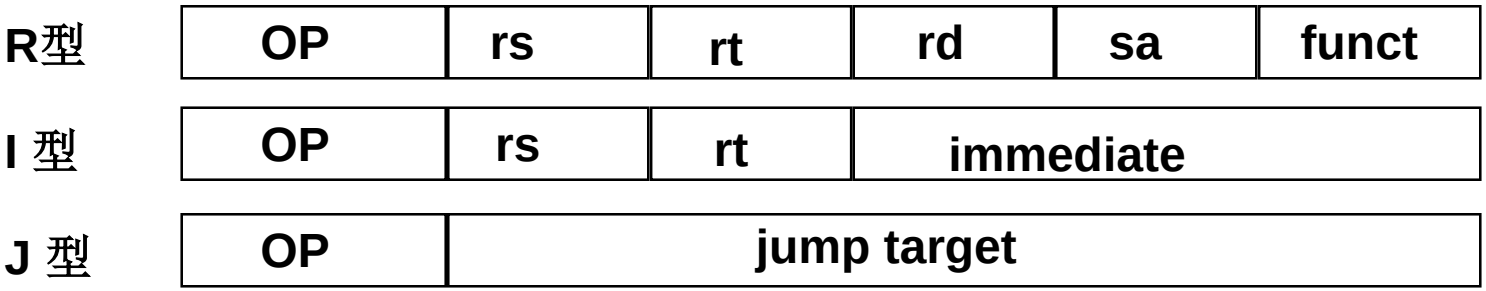
MIPS R3000 Instruction Set Architecture (Summary)

- 指令类型
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - » coprocessor
  - Memory Management
  - Special

Registers



3 种指令格式: all 32 bits wide



计算机组成与实现

- 计算机组成（**Computer Organization or Microarchitecture): ISA的逻辑实现**
  - 物理机器级中的数据流和控制流的组成以及逻辑设计等
- 计算机实现（**Computer Implementation):**计算机组成的物理实现
  - **CPU ， MEMORY**等的物理结构，器件的集成度、速度，模块、插件、底板的划分与连接、信号传输、电源、冷却及整机装配技术等
- 例如
  - 确定指令系统中是否有乘法指令 **(Architecture)**
  - 确定用加法器实现乘法 还是用专门的乘法实现（**Organization)**
  - 器件的选定及所用的微组装技术 （**Implementation)**

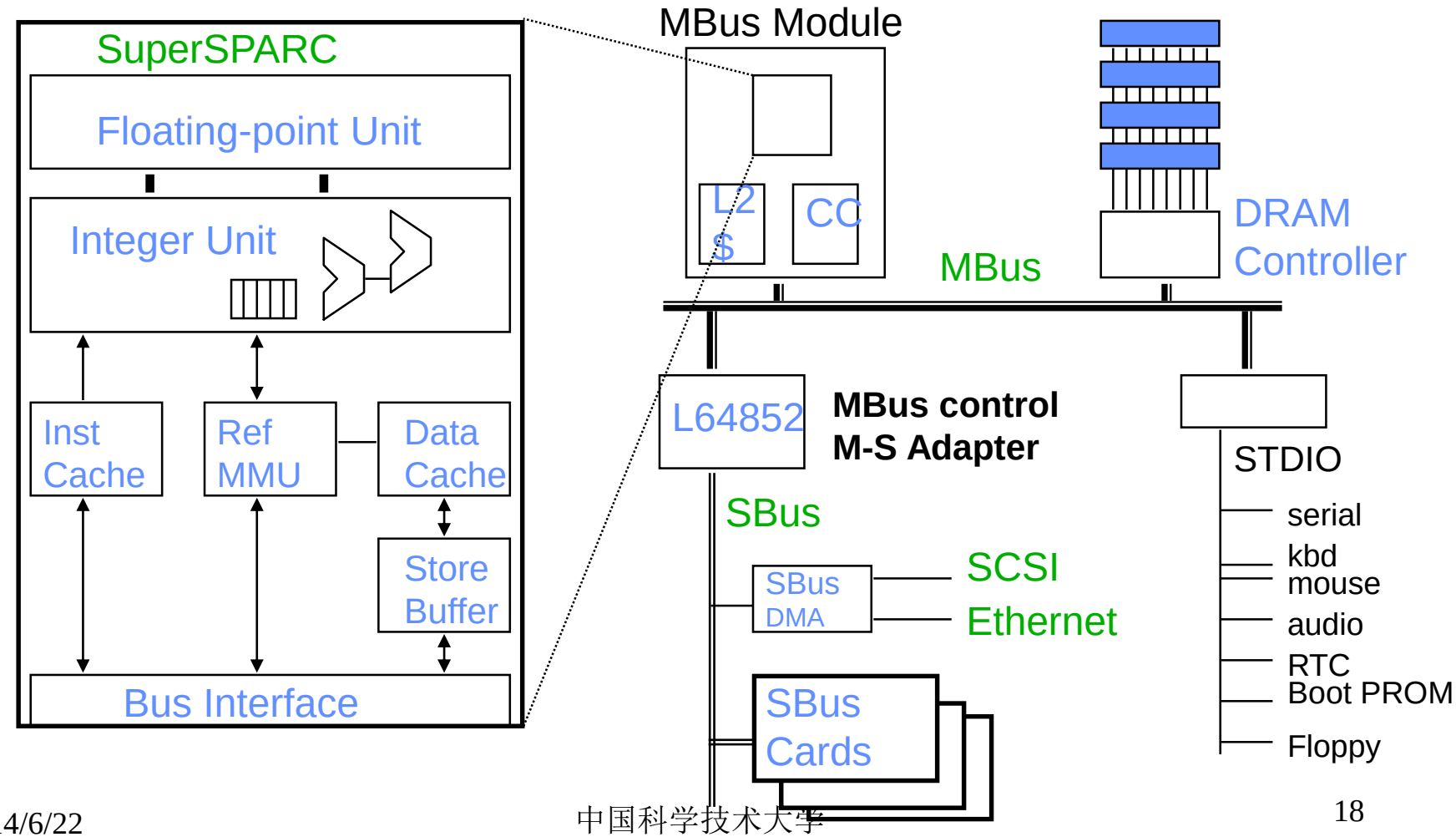
Logic Designer's View

ISA Level

FUs & Interconnect

Example Organization

- TI SuperSPARC^{tm} TMS390Z50 in Sun SPARCstation20



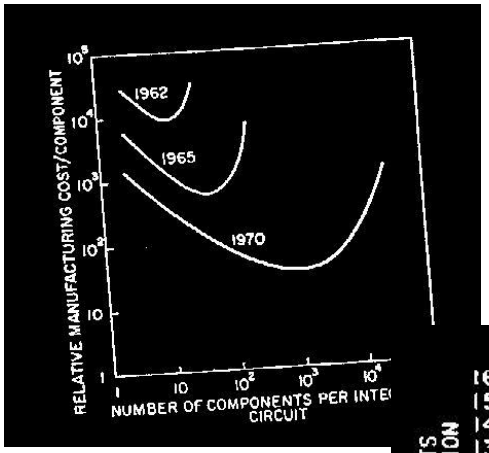
现代计算机系统发展趋势

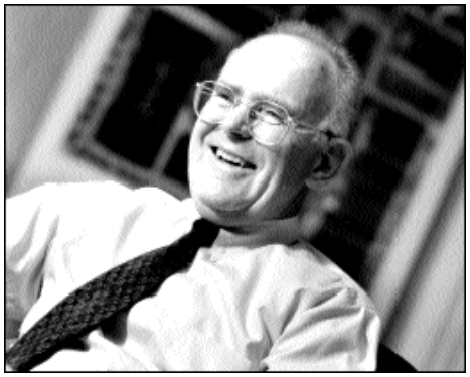
- **Performance**
  - 电路技术的发展
    - » **CMOS VLSI** 取代了原来的**TTL, ECL**技术，提高了器件性能，降低了器件成本。
  - 计算机体系结构技术的发展，提高了低端产品的性能。
    - » **RISC, Superscalar, VLIW, RAID, ….**
- **Price**
  - 开发周期缩短，难度降低
    - » 采用 **CMOS VLSI**，组件减少，系统相对较小。
  - 大规模生产，批量大
  - 系列机的概念，使得服务成本降低。
- **Function**
  - 网络技术，互连网络技术的发展，使得低端产品的功能增强。

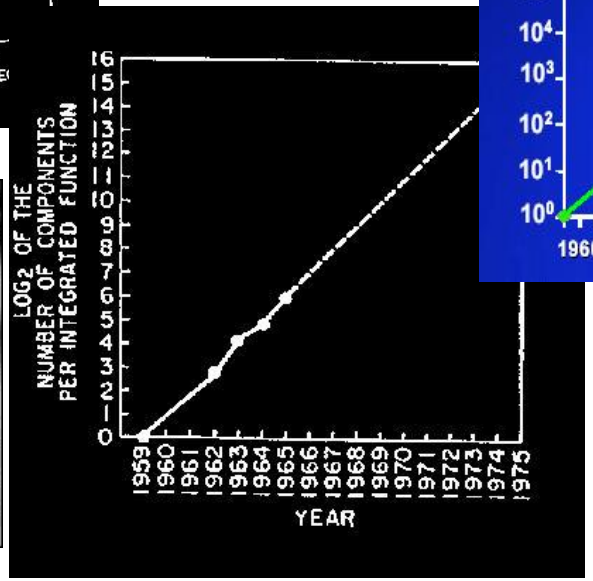
Transistors and Wires

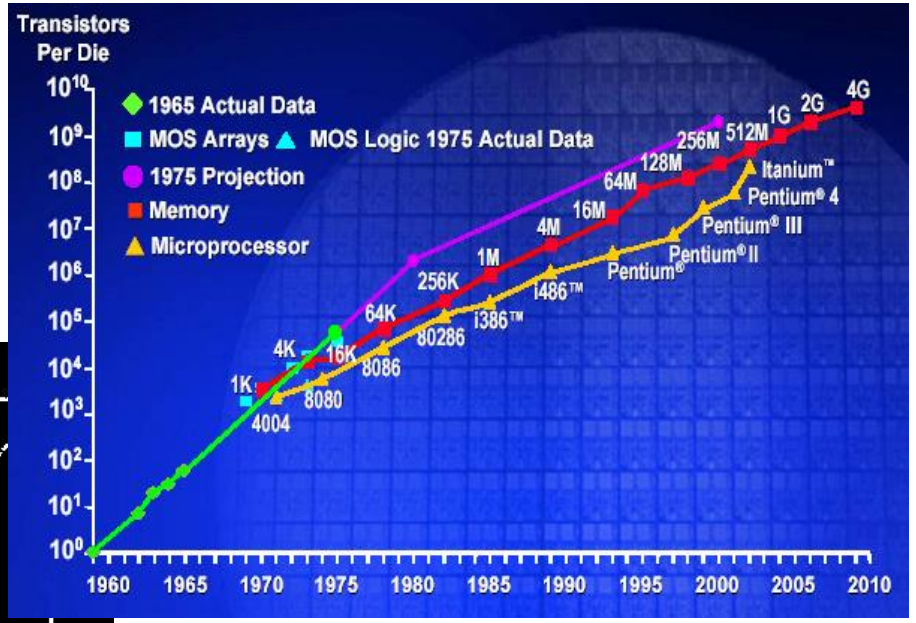
- 特征尺寸（**Feature size**）
  - **Minimum size of transistor or wire in x or y dimension**
  - **10 microns in 1971 to .032 microns in 2011**
  - 晶体管性能线性增长
    - » **Wire delay does not improve with feature size!**
  - 集成度平方增长

Moore's Law

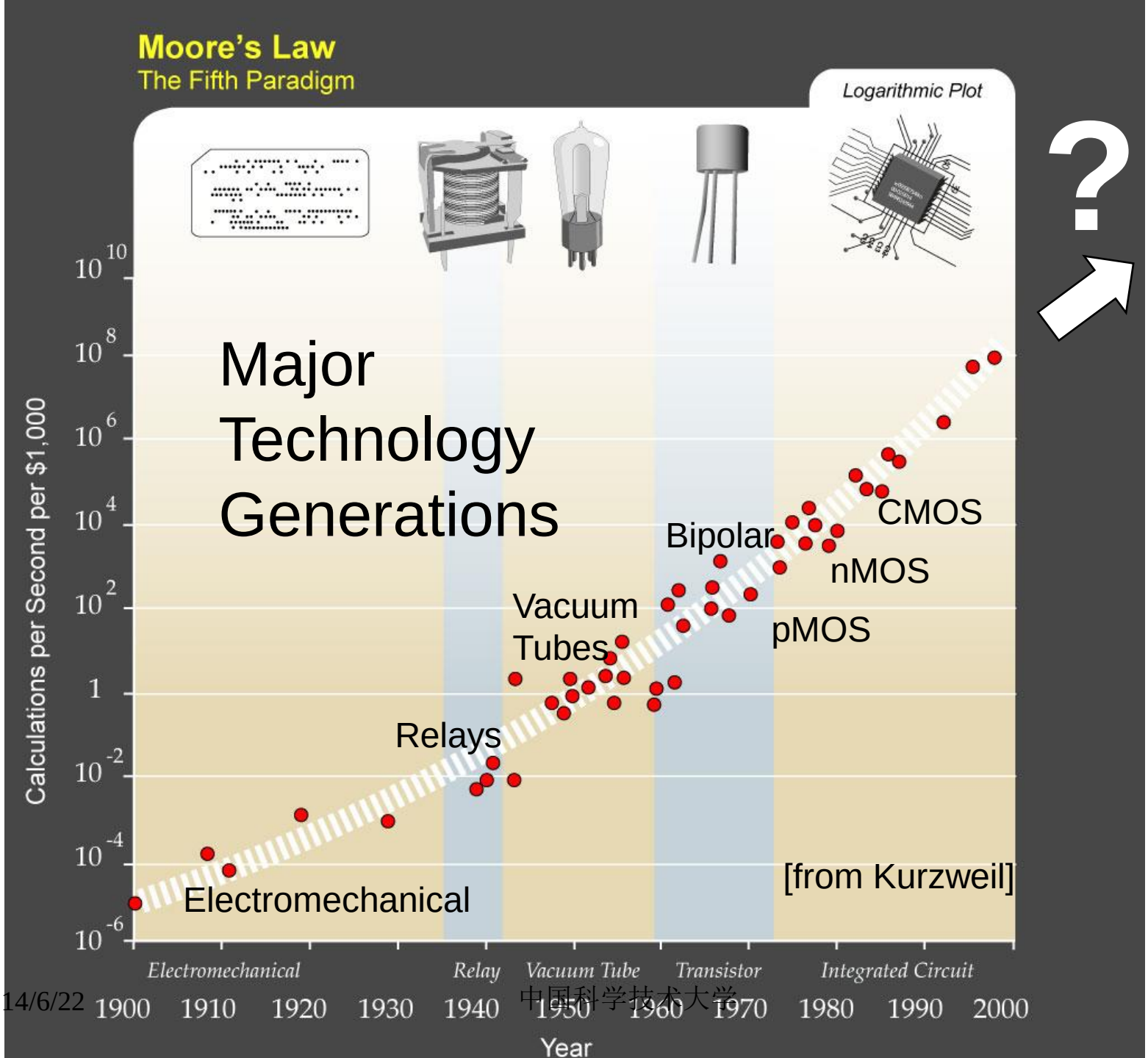








- **"Cramming More Components onto Integrated Circuits"**
  - **Gordon Moore, Electronics, 1965**
- **# on transistors on cost-effective integrated circuit double every 18 months**


Moore's Law
The Fifth Paradigm
Logarithmic Plot
?
Major Technology Generations
Calculations per Second per $1,000
10^{10}
10^{8}
10^{6}
10^{4}
10^{2}
1
10^{-2}
10^{-4}
10^{-6}
Electromechanical
Relays
Vacuum Tubes
Bipolar
CMOS
nMOS
pMOS
[from Kurzweil]
Electromechanical
Relay
Vacuum Tube
Transistor
Integrated Circuit
1900
1910
1920
1930
1940
1950
1960
1970
1980
1990
2000
Year

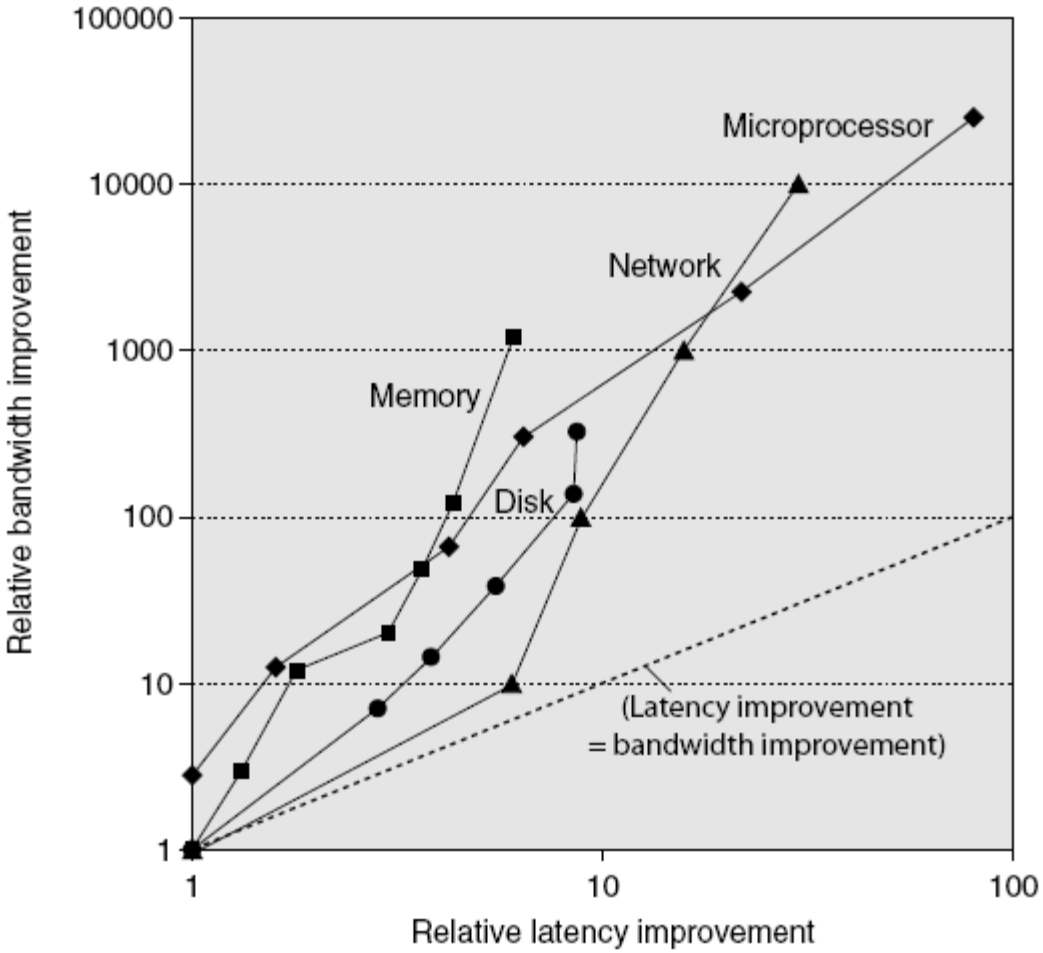
Trends in Technology

- **Integrated circuit technology**
  - **Transistor density: 35%/year**
  - **Die size: 10-20%/year**
  - **Integration overall: 40-55%/year**
- **DRAM capacity: 25-40%/year (slowing)**
- **Flash capacity: 50-60%/year**
  - **15-20X cheaper/bit than DRAM**
- **Magnetic disk technology: 40%/year**
  - **15-25X cheaper/bit then Flash**
  - **300-500X cheaper/bit than DRAM**

Bandwidth and Latency

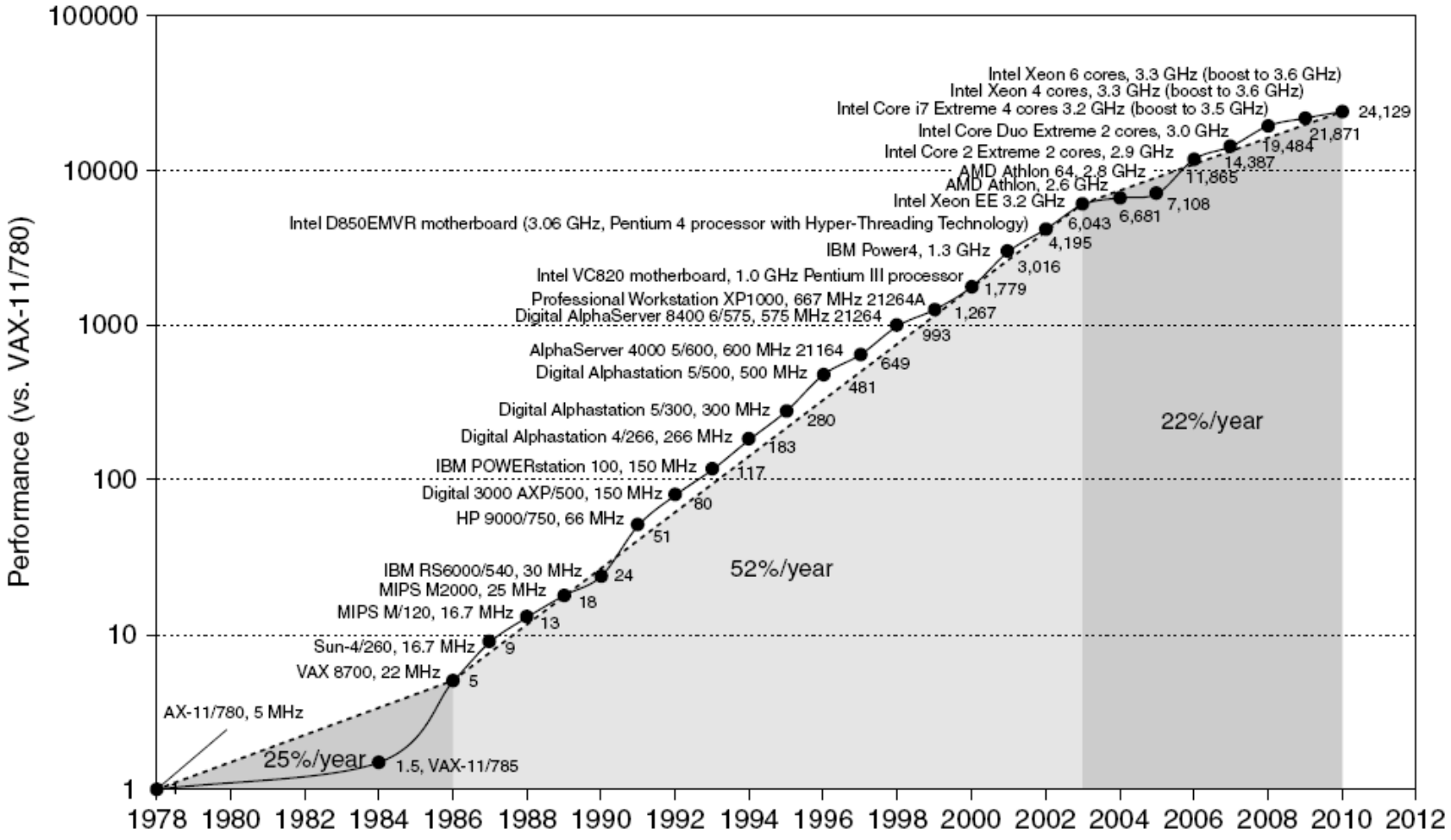
- **Bandwidth or throughput**
  - **Total work done in a given time**
  - **10,000-25,000X improvement for processors and networks**
  - **300-1200X improvement for disks and memory**
- **Latency or response time**
  - **Time between start and completion of an event**
  - **30-80X improvement for processors and networks**
  - **6-8X improvement for memory and disks**

Bandwidth and Latency



Log-log plot of bandwidth and latency milestones

Single Processor Performance



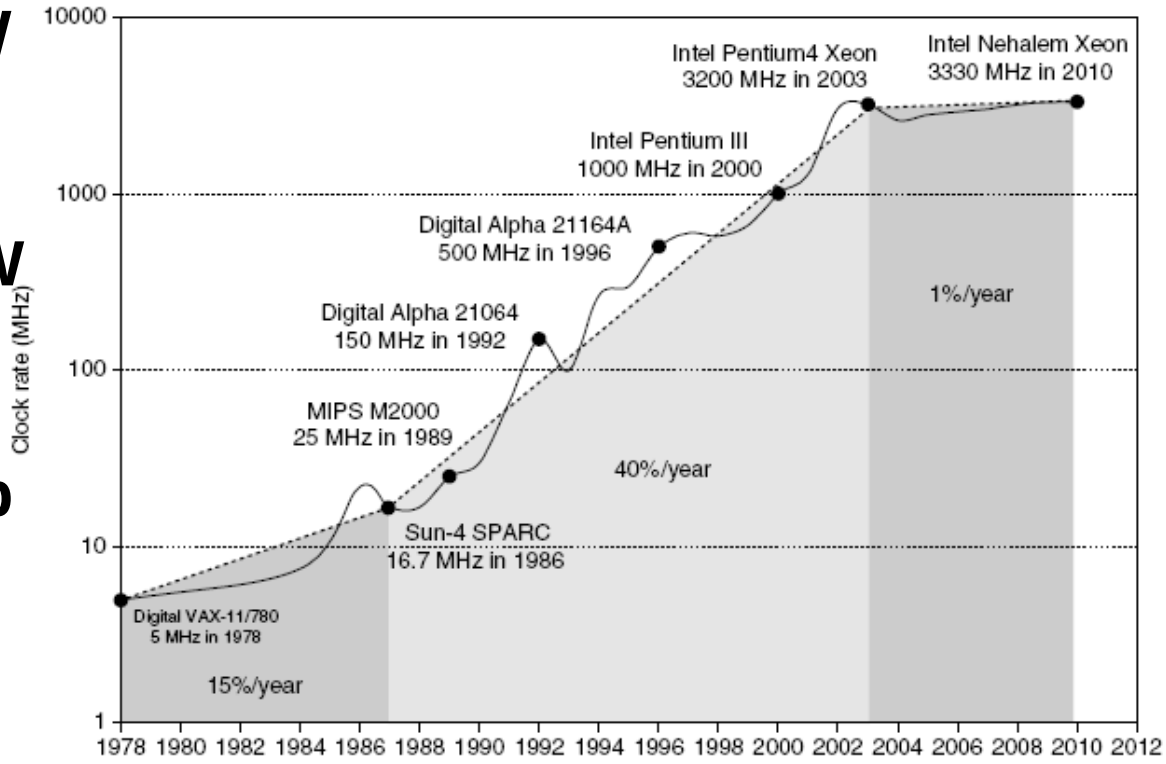
Power & Energy

$$Power_{dynamic} = \frac{1}{2} \times CapacitiveLoad \times Voltage^2 \times FrequencySwitched$$

$$Energy_{dynamic} = CapacitiveLoad \times Voltage^2$$

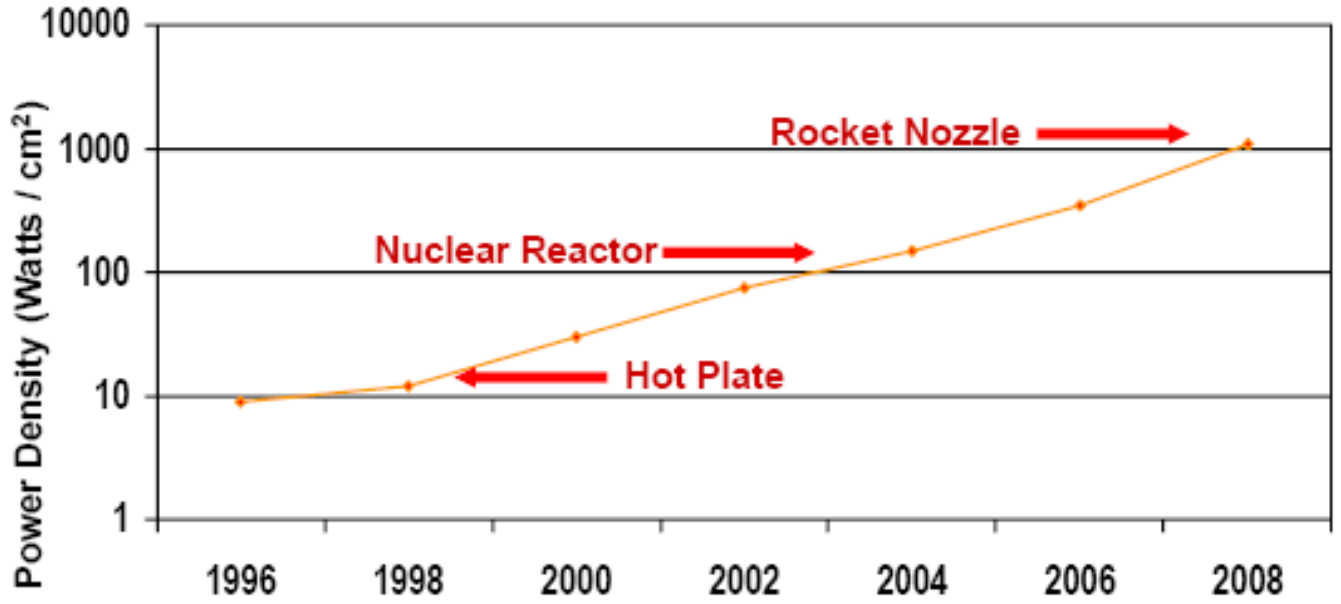
Power

- **Intel 80386 consumed ~ 2 W**
- **3.3 GHz Intel Core i7 consumes 130 W**
- **Heat must be dissipated from 1.5 x 1.5 cm chip**
- **This is the limit of what can be cooled by air**



Limiting Force: Power Density

Moore's Law Extrapolation:
Power Density for Leading Edge Microprocessors



Power Density Becomes Too High to Cool Chips Inexpensively

Source: Shekhar Borkar, Intel Corp

Conventional Wisdom in Computer Architecture

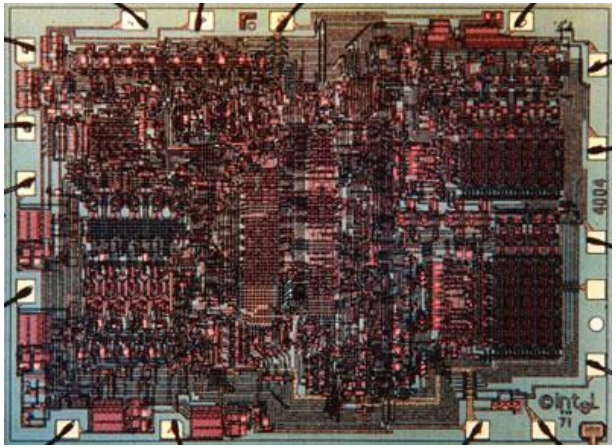
- **Old Conventional Wisdom: Power is free, Transistors expensive**
- **New Conventional Wisdom: "Power wall" Power expensive, Transistors free (Can put more on chip than can afford to turn on)**
- **Old CW: Sufficient increasing Instruction-Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, …)**
- **New CW: "ILP wall" law of diminishing returns on more HW for ILP**
- **Old CW: Multiplies are slow, Memory access is fast**
- **New CW: "Memory wall" Memory slow, multiplies fast (200 clock cycles to DRAM memory, 4 clocks for multiply)**
- **Old CW: Uniprocessor performance 2X / 1.5 yrs**
- **New CW: Power Wall + ILP Wall + Memory Wall = Brick Wall**
  - **Uniprocessor performance now 2X / 5(?) yrs**

⇒ Sea change in chip design: multiple "cores" (2X processors per chip / ~ 2 years)

» **More, simpler processors are more power efficient**

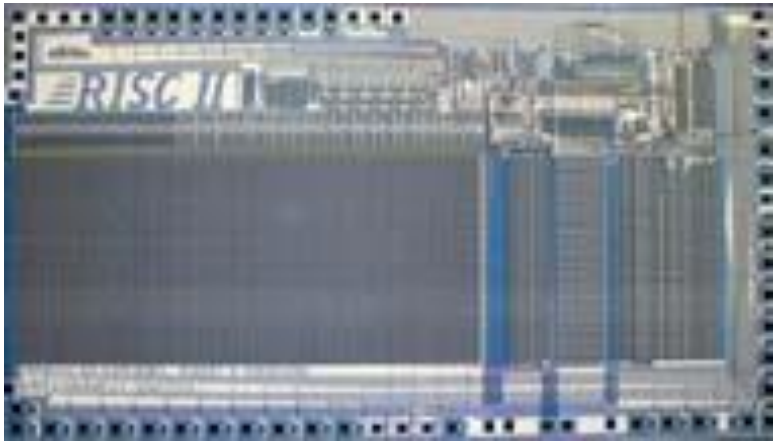
Sea Change in Chip Design

- **Intel 4004 (1971): 4-bit processor, 2312 transistors, 0.4 MHz, 10 micron PMOS, 11 mm^2 chip**



- **RISC II (1983): 32-bit, 5 stage pipeline, 40,760 transistors, 3 MHz, 3 micron NMOS, 60 mm^2 chip**

- **125 mm^2 chip, 0.065 micron CMOS = 2312 RISC II+FPU+Icache+Dcache**
  - **RISC II shrinks to ~ 0.02 mm^2 at 65 nm**
  - **Caches via DRAM or 1 transistor SRAM?**



- **Processor is the new transistor?**

- **"We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"**

  Paul Otellini, President, Intel (2004)

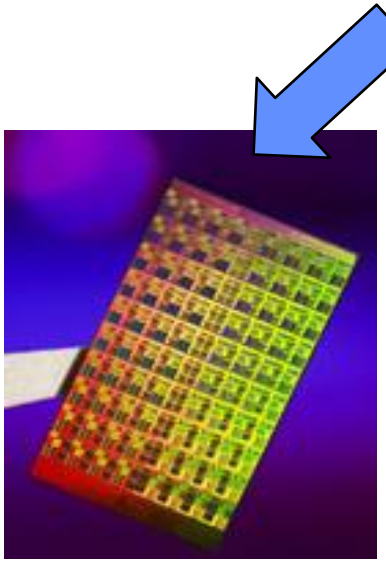
- **Difference is all microprocessor companies have switched to multiprocessors (AMD, Intel, IBM, Sun; all new Apples 2+ CPUs)**

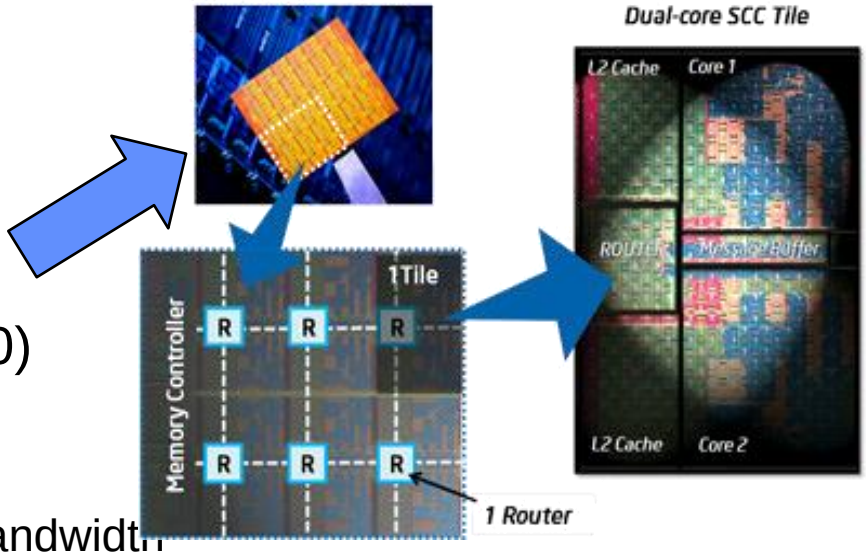
  ⇒ Procrastination penalized: 2X sequential perf. / 5 yrs

  ⇒ Biggest programming challenge: from 1 to 2 CPUs

ManyCore Chips: The future is here

- Intel 80-core multicore chip (Feb 2007)
  - 80 simple cores
  - Two FP-engines / core
  - Mesh-like network
  - 100 million transistors
  - 65nm feature size
- Intel Single-Chip Cloud Computer (August 2010)
  - 24 "tiles" with two IA cores per tile
  - 24-router mesh network with 256 GB/s bisection bandwidth
  - 4 integrated DDR3 memory controllers
  - Hardware support for message-passing





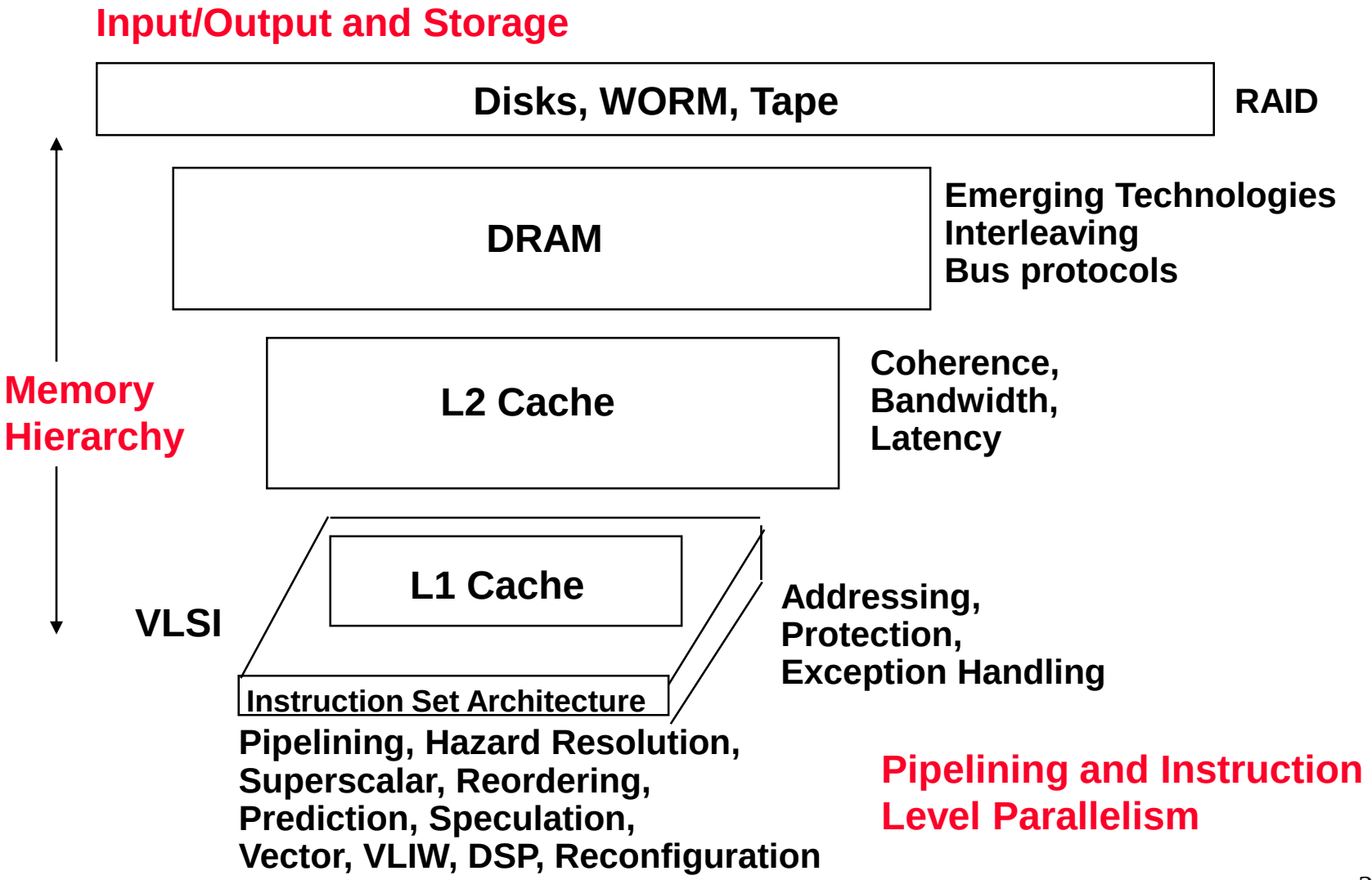
- **"ManyCore" refers to many processors/chip**
  - **64? 128? Hard to say exact boundary**
- **How to program these?**
  - **Use 2 CPUs for video/audio**
  - **Use 1 for word processor, 1 for browser**
  - **76 for virus checking???**
- **Something new is clearly needed here…**

The End of the Uniprocessor Era

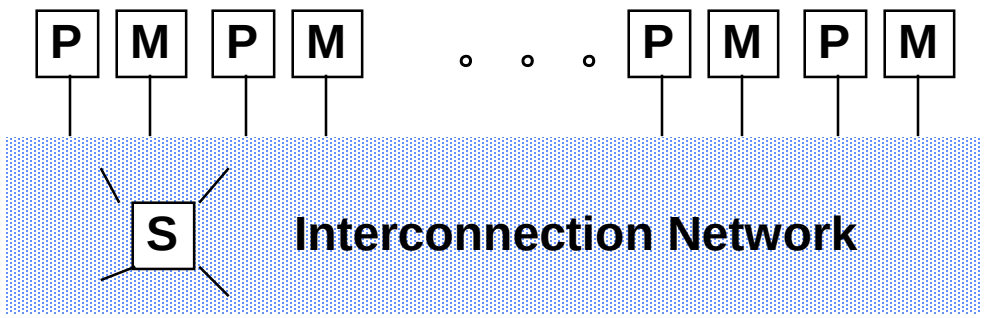
Single biggest change in the history of computing systems

——摘自 Berkeyley CS252

计算机体系结构研究的内容



计算机体系结构研究内容（续）



Processor-Memory-Switch

Multiprocessors
Networks and Interconnections

Shared Memory, Message Passing, Data Parallelism

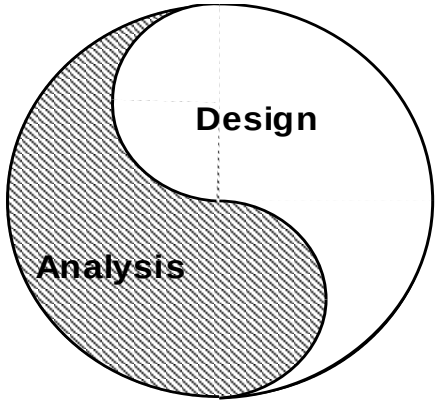
Network Interfaces

Topologies, Routing, Bandwidth, Latency, Reliability

计算机体系结构课程内容的变化

- **1950s to 1960s:** 体系结构课程：运算器
- **1970s to 1980s中:** 体系结构课程:指令集设计
- **1990s:** 体系结构课程：**CPU**设计，存储系统设计，**I/O**系统设计，多处理器，网络
- **2000s:** 体系结构课程：非 **Von-Neumann** 结构, 可配置体系结构等, 多核，片上网络，并行编程模式、低功耗设计等
- **2010s: Self Adapting Systems? Self Organizing Structures? DNA System/ Quantum Computing?**

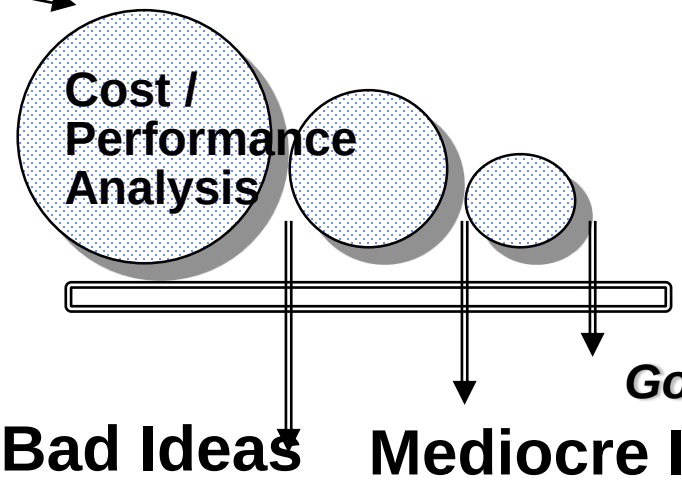
计算机体系结构设计过程

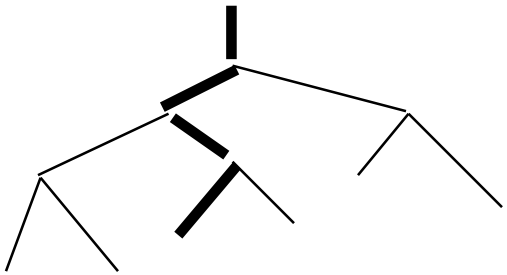


Creativity

体系结构设计是循环渐进的过程:

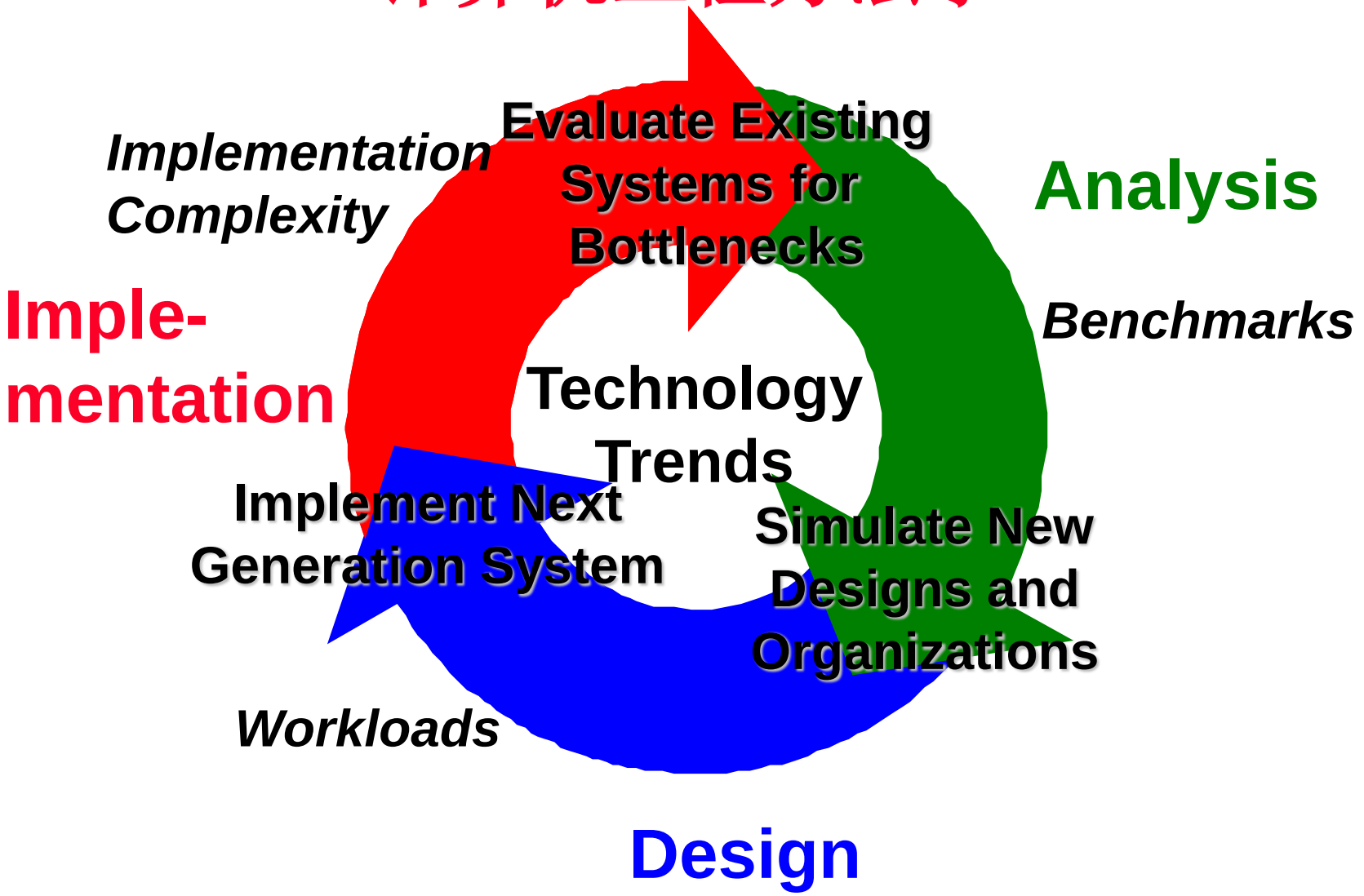
- Search the possible design space
- Make selections
- Evaluate the selections made





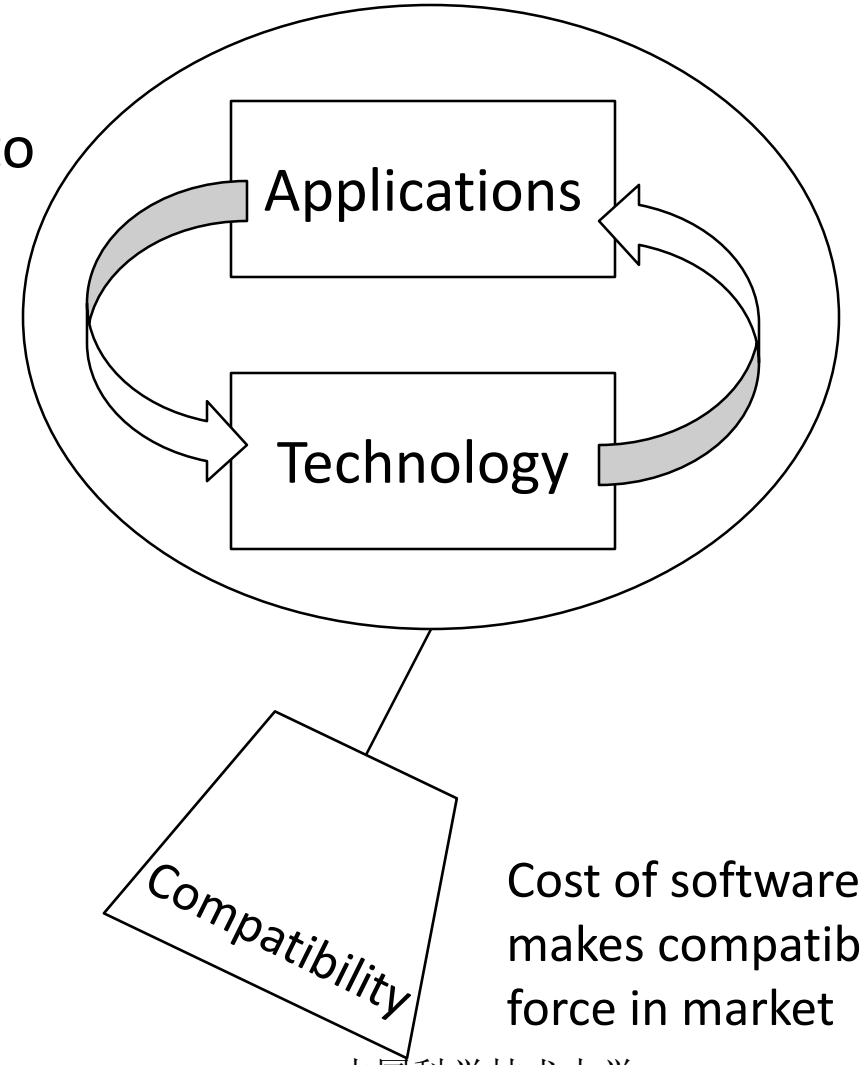
Good measurement tools are required to accurately evaluate the selection.

计算机工程方法学


Evaluate Existing Systems for Bottlenecks
Analysis
Benchmarks
Technology Trends
Simulate New Designs and Organizations
Design
Workloads
Implement Next Generation System
Imple-mentation
Implementation Complexity

体系结构发展的驱动力

Applications suggest how to improve technology, provide revenue to fund development



Improved technologies make new applications possible

Cost of software development makes compatibility a major force in market

本课程的主要内容

5 部分内容

1. **Simple machine design (ISAs, Iron Law, simple pipelines)（Chapter 1, Appendix A, Appendix C)**
2. **Memory hierarchy (DRAM, caches, optimizations) plus virtual memory systems, exceptions, interrupts（Chapter 2，Appendix B)**
3. **Complex pipelining (score-boarding, out-of-order issue) (Chapter 3)**
4. **Explicitly parallel processors (vector machines, VLIW machines, multithreaded machines)（Chapter 4)**
5. **Multiprocessor architectures (memory models, cache coherence, synchronization,) (Chapter 5,Chapter 6)**

课程目标

- 掌握计算机系统定量分析的基本方法和技术
- 深入理解提高CPU性能的基本方法
- 深入理解存储系统的基本原理和基本的优化方法
- 理解数据级并行、线程级并行以及请求级并行的基本原理和方法

课程安排

- 授课
  - 授课总学时60学时，实验30学时
  - 星期三 (7,8) 3C221、五（7，8）3C223
- 评分
  - 平时作业　10％
  - 实验　30％
  - 期中考试　25％
  - 期末考试　35％

教材与主要参考书

- **John L. Hennessy, David A. Patternson, Computer Architecture: A Quantitative Approach. Fifth Edition.** 机械工业出版社，**2012**
- **David A. Patternson, John L. Hennessy, Computer Organization & Design : The Hardware/Software Interface, Third Edition. San Francisco: Morgan Kaufmann Publishers, Inc. 2005**
- 张晨曦等，计算机系统结构教程，清华大学出版社

- **Berkeley CS152, CS252**
- **Elsevier Pte Ltd**
- **Acknowledgements**

关于作弊

- 作业
- 实验
- 考试（测验）

为什么学这门课

深入理解计算机体系结构有助于:

- **Design better computer architectures**
  - **There are still many challenges left**
  - **Example: the CPU-memory gap**
  - **.....**
- **Write better operating systems**
  - **Need to re-evaluate the current assumptions and tradeoffs**
  - **Example: gigabit networks**
- **Write better compilers**
  - **Modern computers need better optimizing compilers and better programming languages**
- **Write better programs**
  - **Understand the performance implications of algorithms, data structures, and programming language choices**

小结－计算机体系结构、组织和实现

- 指令级结构(Instruction Set Architecture)研究软、硬件功能分配以及机器级界面的确定，既由机器语言程序设计者或编译程序设计者所看到的机器物理系统的抽象或定义。但它不包括机器内部的数据流和控制流、逻辑设计和器件设计等。
- 计算机组织(Computer Organization)：ISA的逻辑实现，包括机器级内的数据流和控制流的组成以及逻辑设计等。它着眼于机器级内各事件的排序方式与控制机构、各部件的功能以及各部件间的联系。
- 计算机实现（Computer Implementation）是指计算机组成的物理实现，包括处理机、主存等部件的物理结构，器件的集成度和速度，器件、模块、插件、底板的划分与连接，专用器件的设计，微组装技术，信号传输，电源、冷却及整机装配技术等。它着眼于器件技术和微组装技术，其中，器件技术在实现技术中起着主导作用。
- 计算机体系结构＝**ISA + organizaiton + hardware**
-

Summary

- 计算机体系结构的基本概念
  - **ISA+Organization+Implementation**
- 本课程将涉及的主要内容
  - 简单机器设计（**ISA,** 基本流水线）指令级并行
  - 存储系统 **(Cache, Virtual Memory)**
  - 复杂流水线 （动态指令流调度、动态分支预测）
  - 显式并行处理器（向量处理器、**VLIW**，多线程处理）
  - 多处理器结构
- 体系结构设计面临的新问题

Power Wall + ILP Wall + Memory Wall = Brick Wall

Chapter1 量化设计与分析基础

1.1 引论

- 计算机体系结构的基本概念
- 计算机市场的变化
- 现代计算机系统发展趋势

1.2 定量分析技术基础

- 计算机系统评价
- 计算机性能度量
- 性能设计和评测的基本原则
- 系统结构评价标准

1.2 定量分析技术基础

- 计算机系统评价
- 计算机性能度量
- 性能设计和评测的基本原则
- 系统结构评价标准

客户 vs. 设计者

- 客户：给定一组机器，哪个
  - 性能最好?
  - 价格最低?
  - 性/价比最高（performance / cost）？
- 设计者：面临的设计选择：
  - 最大限度的提高性能
  - 价格最低?
  - 性/价比最高（performance / cost）？
- 我们需要
  - 有基本的评价标准和方法
- 我们的目标是理解性能和成本 与体系结构选择的关系

评价指标

- 执行时间（**CPU Time**、**Wall-clock Time, Elapsed Time**）
- 带宽 （**Bandwidth**）、延迟（**Latency**）
- 峰值速度 （**Peak Performance**）
- 负载 （**load**）、开销 （**Overhead**）
- 利用率 （**Utilization Ratio**），吞吐率 （**Throughput**）
- 加速比 （**Speedup**）效率 （**Efficiency**）
- 基准测试 **Benchmark**
  - 微基准测试 **Micro-benchmark**：测量系统某一方面的分离性能, 如核心程序,合成测试程序等
  - 宏基准测试 **Macro-benchmark**：测量系统总体性能, 如实际应用程序等
- 响应时间（**Response Time**）
- ……

系统评价的基本作用

- 用性能评价软件包，了解系统性能， 对用户选型和配置提出建议
- 针对不同应用，不同软硬件配置进行性能评价和优化，对用户所使用系统提出性能上的建议
- 建立理论模型，对系统的性能进行预测

Benchmarks

- 没有一个标准能反映计算机系统的全部性能，它们代表的只是性能的一个侧面。
- 常用的标准
  - 定点性能
  - 浮点性能
  - Web服务性能
  - 数据处理性能
  - 系统软件性能
  - 科学与工程计算性能

性能的两种含义

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concorde	3 hours	1350 mph	132	178,200

哪个性能高？

- **Time to do the task (Execution Time)**
  - execution time, response time, latency
- **Tasks per day, hour, week, sec, ns. .. (Performance)**
  - throughput, bandwidth

这两者经常会有冲突的。

性能定义

- 之一：性能定义为单位时间完成的任务数
  - **bigger is better**
- 之二：如果我们更关心响应时间（**response time**）

$$performance(x) = \frac{1}{execution\_time(x)}$$

“X 性能是Y的n倍” 是指

$$n = \frac{Performance(x)}{Performance(y)}$$

举例

- Time of Concord vs. Boeing 747?
  - Concord is 1350 mph / 610 mph = 2.2 times faster
    = 6.5 hours / 3 hours
- Throughput of Concorde vs. Boeing 747 ?
  - Concord is 178,200 pmph / 286,700 pmph = 0.62 “times faster”
  - Boeing is 286,700 pmph / 178,200 pmph = 1.60 “times faster”

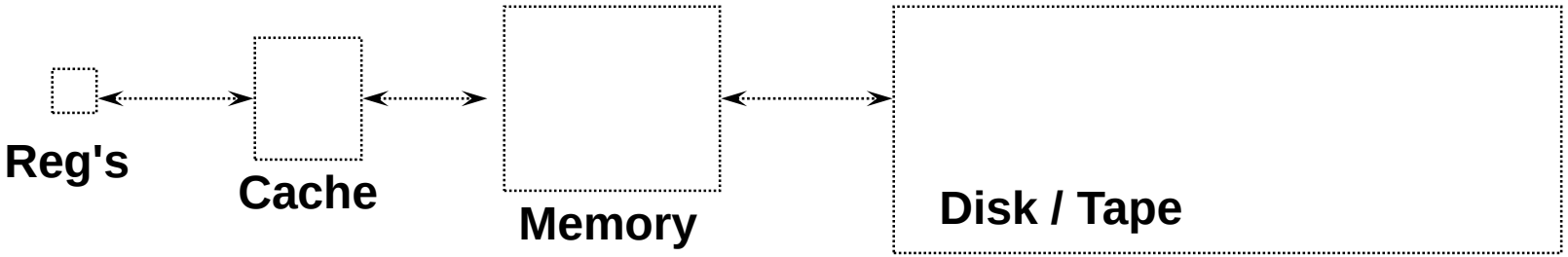
- Boeing is 1.6 times (“60%”) faster in terms of throughput
- Concord is 2.2 times (“120%”) faster in terms of flying time

我们主要关注单个任务的执行时间

程序由一组指令构成，指令的吞吐率（Instruction throughput）非常重要!

性能设计与评测的基本原则

- 并行性
- 大概率事件优先原则
  - 所有指令都需要取指令操作，只有部分指令访问数据
  - 优化指令访问操作比优化数据访问操作优先
- 程序局部性原理
  - 时间局部性
  - 空间局部性

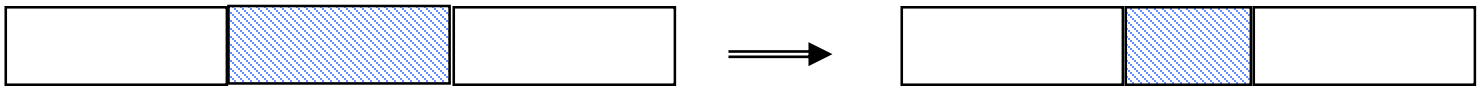


- **Amdahl定律**

Amdahl's Law

- 假设对机器的部件进行了改进（加速比的概念）

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



- 假设可改进部分**E**在原来的计算时间所占的比例为**F**，而部件加速比为**S**，任务的其他部分不受影响，则

$$\text{ExTime(with E)} = ((1-F) + F/S) \times \text{ExTime(without E)}$$

$$\text{Speedup(with E)} = 1/((1-F)+F/S))$$

重要结论(性能提高的递减原则)：如果只针对整个任务的一部分进行优化，那么所获得的加速比不大于**1/(1-F)**

举例

- 假设给定一体系结构硬件不支持乘法运算，乘法需要通过软件来实现。在软件中做一次乘法需要200个周期，而用硬件来实现只要4个时钟周期。如果假设在程序中有10％的乘法操作，问整个程序的加速比？如果有40％的乘法操作，问整个程序的加速比又是多少？
- 假设一计算机在运行给定的一程序时，有90％的时间用于处理某一类特定的计算。现将用于该类计算的部件性能提高到原来的10倍。
  - 如果该程序在原来的机器上运行需100秒，那么该程序在改进后的机器上运行时间是多少？
  - 新的系统相对于原来的系统加速比是多少？
  - 在新的系统中，原来特定的计算占整个计算的比例是多少？

CPU性能度量

- Response time (elapsed time): 包括完成一个任务所需要的所有时间
  - User CPU Time (90.7)
  - System CPU Time (12.9)
  - Elapsed Time (2:39)

  例如：unix 中的time命令

  90.7u 12.9s 2:39 65% (90.7/159)

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

CPU性能度量—CPI

“Average cycles per instruction”

CPI_{ave} = (CPU Time * Clock Rate) / Instruction Count
= Clock Cycles / Instruction Count

$$\text{CPU time} = \text{ClockCycleTime} * \sum_{i=1}^{n} (CPI_i * I_i)$$

$$CPI = \sum_{i=1}^{n} CPI_i * F_i \quad \text{where} \quad F_i = I_i / \text{Instruction Count}$$

"instruction frequency"

CPI计算举例

Base Machine (Reg / Reg)

Op	Freq	CPI_i	CPI_i*F_i	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			1.5	

	Inst Count	CPI	Clock Rate
Program	X	X	
Compiler	X	(X)	
Inst. Set.	X	X	(X)
Organization		X	X
Technology			X

基本评估方法—市场评估方法

- MIPS：每秒百万条指令数

  MIPS = IC/(CPI*IC*T*10^6)=1/(CPI*T*10^6)

  - MIPS依赖于指令集
  - 在同一台机器上，MIPS因程序不同而变化，有时差别较大
  - MIPS可能与性能相反

    举例。在一台load-store型机器上，有一程序优化编译可以使ALU 操作减少到原来的50%,其他操作数量不变。

    F = 500MHZ

    ALU (43% 1) loads (21% 2) stores (12% 2)

    Branches (24% 2)
- MFLOPS 基于操作而非指令，它可以用来比较两种不同的机器。但MFLOPS也并非可靠，因为不同机器上浮点运算集不同。CRAY-2没有除法指令，Motorola 68882有
- SPEC测试

Computer Performance	
Name	FLOPS
yottaFLOPS	**10^{24}**
zettaFLOPS	**10^{21}**
exaFLOPS	**10^{18}**
petaFLOPS	**10^{15}**
teraFLOPS	**10^{12}**
gigaFLOPS	**10^{9}**
megaFLOPS	**10^{6}**
kiloFLOPS	**10^{3}**

基本评估方法—benchmark测试

- 五种类型的测试程序（预测的精度逐级下降）

(1)真实程序：这是最可靠的方法。

(2)修改过的程序：通过修改或改编真实程序来构造基准程序模块。原因：增强移植性或集中测试某种特定的系统性能

(3)核心程序**(Kernels)**：由从真实程序中提取的较短但很关键的代码构成。**Livermore Loops**及**LINPACK**是其中使用比较广泛的例子。

(4) 小测试程序（**toy programs)**：小测试程序代码一般在**100**行以内。

(5)合成测试程序**(Synthetic benchmarks)**：首先对大量的应用程序中的操作进行统计，得到各种操作比例，再按这个比例人造出测试程序。**Whetstone**与**Dhrystone**是最流行的合成测试程序。

基准测试程序套件

- **Embedded Microprocessor Benchmark Consortium (EEMBC)**
- **Desktop Benchmarks**
  - **SPEC2006**
  - **SPEC2000**
  - **SPEC 95**
  - **SPEC 92**
  - **SPEC 89**
- **Server Benchmarks**
  - **Processor Throughput-oriented benchmarks (基于SPEC CPU benchmarks->SPECrate**
  - **SPECSFS, SPECWeb**
  - **Transaction-processing (TP) benchmarks (TPC-A, TPC-C, …)**

性能的综合评价

- 算术平均或加权的算术平均
  - **SUM(Ti)/n 或 SUM（Wi×Ti)/n**
- 规格化执行时间，采用几何平均

$$\sqrt[n]{\prod_{i=1}^{n} Execution\_time\_ratio_i}$$

为什么对规格化数采用几何平均?

	Computer A	Computer B	Computer C
Program P1 (secs)	1	10	20
Program P2 (secs)	1000	100	20
Total time (secs)	1001	110	40

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.0	10.0	**20.0**	0.1	1.0	**2.0**	**0.05**	**0.5**	1.0
Program P2	1.0	0.1	**0.02**	10.0	1.0	**0.2**	**50.0**	**5.0**	1.0
Arithmetic mean	1.0	**5.05**	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Geometric mean	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
Total time	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

定点性能测试

- **SPECint95：** 反映评测系统的单处理器的定点运算性能
  - **SPEC：Standard Performance Evaluation Corporation**
  - **www.spec.org**
  - **8**个真实的应用：仿真技术、人工智能、图像处理、压缩算法、编译器、解释器、数据库
  - 用运行**8**个应用的标准时间**(SUN Ultra5_10, 300MHZ)**，除以实际运行时间得到一个比值，**SPEC_int95**是这**8**个比值乘积的开**8**次方得到的值
- **SPECint_base95:** 采用最保守的优化策略
- **SPECint_rate95：** 反映具有多个处理器系统的性能的可扩展性，允许每个应用同时运行多个实例
  - 比值的计算方法：运行次数*****（应用标准运行时间***1**天中的秒数**/8**个应用中最长的标准运行时间）**/**多次运行的总时间，**SPECint_rate95**是这**8**个比值的乘积开**8**次方。
- **SPECint_base_rate95 —**采用最保守的编译优化策略

定点性能

- **SPECint2000**

  12个应用：压缩算法、编译器、优化组合、棋类游戏、字处理、可视化、PERL语言、群论解释器、面向对象数据库、仿真技术.

  Written in C (11) and C++ (1)

- **Dhrystone**

  发布于1984年，主要包含两类语句，字符串赋值和字符串比较。

浮点性能评测

SPECfp95

评测系统的单处理器的浮点运算性能

10个真实的应用：流体力学、天气预报、量子物理、天文、电子

SPECfp_base95

采用最保守的编译优化策略

SPECfp_rate95

反映具有多个处理器系统的浮点性能的可扩展性

SPECfp_base_rate95

采用最保守的编译优化策略

浮点性能评测

SPECfp2000

14个应用：量子色动、浅水模型、三维电势场、抛物线/椭圆偏微分方程、三维图像库、计算流体力学、图像识别/神经网络、地震波传播仿真、图像处理/人脸识别、计算化学、数论、有限元碰撞仿真、高性能物理加速器设计、污染分布计算

Flops

反映系统单处理器的峰值浮点运算能力
通过指令的不同组合来得到浮点加、减、乘、除的计算能力，尽量使用寄存器，少与内存交互

Web服务性能

- **SPECweb96**
  - 评价Web响应用户Web点击的性能
  - 由客户端向服务器发送HTTP GET请求
  - SPECweb96值是服务器每秒能够支持的连接数量
- **SPECweb99**
  - 评价了Web服务器综合性能
  - 每个客户端运行于400Kb/s的线路上
  - 服务器最多支持320Kb/s以上的客户端连接数
  - 不仅支持HTTP GET操作，还支持POST和Cookie

数据处理性能

Debit Credit

1984年Tandem公司的Jim Gray提出

模拟一个具有多家分支机构银行的出纳操作，采用California银行1970年的数据

只包含银行存款帐户行为一种类型的事务

存款行为记录文件：帐户文件、分支机构文件、出纳文件、操作顺序的历史数据文件

帐户的规模、分支机构数据是系统吞吐量函数，例如：每个TPS应配置10个分支机构，100个出纳员，100000个帐户信息

规定每次出纳操作的时间固定为100秒，合法的结果应有95%的事务在1秒内完成

数据处理性能

- **TPC：Transaction Processing Performance Council，成立于1988年 (www.tpc.org)**
  - 评测计算机系统进行事务处理和数据库操作的性能
- **TPC-A**
  - 使用不同的输入和查询数据
  - 修改密集型事务
  - 评价联机事务处理（**OLTP**）的性能
  - **1995**年后不再使用
- **TPC-B**
  - 集中式数据库处理
  - 不需要终端和网络
  - 数据库操作有大量的磁盘**I/O**
  - 中等量级的系统和应用执行时间
  - 有很多处理之间的集成操作

数据处理性能

TPC-C

1992年开发

用远程终端模拟器模拟大量的终端用户

模拟存在大量地理上分散部门的企业的行为

数据库结构复杂，多种事务处理模型、执行模式，热点现象，全屏终端I/O格式化数据，透明的数据分区和事务处理的回滚

一般表示为tpmC和$/tpmC（Transactions Per Minute Computer）

五种事务：付款(payment)、订单状态查询(order-status)、发货(delivery)、库存级别(stock-level)、新订单(new-order)

每种事务都有响应时间的要求，如new-order设置为5秒

tpmC是系统在满足其它4类事务响应时间要求的前题下，在1分钟内处理new-order事务的数量

数据处理性能

TPC-D

决策支持应用，用于测试系统支持耗时的、只读的数据库操作的性能

每个复杂的查询都要存取数据库的大部分数据，进行多次join, sort, group, scan等操作

17个复杂查询和2个修改操作

极大程度地依赖于查询的优化、数据库表格的划分方法、SQL的效率、和高级索引技术

系统软件性能

- **Lmbench：SGI开发，测试操作系统性能**
  - 操作系统指标：空系统调用时间，进程切换时间，pipe、UDP、TCP、RPC的延迟和带宽，内存、Cache、TLB的读写性能，存储映射的性能
  - 既能反映计算机系统的一些基本性能指标，也能反映操作系统实现的优劣
- **Netperf**
  - 评测计算机系统的网络性能，也可用来评测DLPI（Data Link Provider Interface），Unix Domain Socket的性能
  - TCP、UDP的带宽和请求应答数
  - 按照客户机/服务器模式设计，结果数据是在用户设定的时间段内，两者之间传递的最大数据量

系统软件性能

SPECsfs97

评测系统的NFS性能

采用客户机/服务器模式，客户机向服务器发送特定的NFS请求，得到NFS文件服务器的吞吐量和响应时间

SPECjvm98

使用8个应用来评测JAVA虚拟机的性能

科学与工程计算性能

Linpack

LINear algebra PACKages

解线性方程组和线性最小二乘问题

1000x1000标准

计算饱和峰值

Top500

科学与工程计算性能

SPLASH

- **Stanford大学开发，评测共享存储系统性能**
- **7个完整的应用和5个计算核心程序**
- **科学与工程计算，计算机图形学方面的并行程序**

ParkBench： 评价大型可扩展系统的计算性能

micro-benchmark：获取单处理器的有关体系结构和编译器的基本性能参数；测试内容包括时钟调用、算术运算、内存带宽和延迟、通信延迟和带宽、全局同步操作性能等

kernel-benchmark： 矩阵运算、FFT、偏微分方程、NAS核心，I/O Benchmark

compact application：气候模型、计算流体动力学、财务模型、分子动力学、等离子物理、量子化学、水库模型

compiler：评价HPF编译器

评价服务器性能的基本方法

基本性能参数：CPU、内存、I/O、网络、操作系统、文件系统、编译器、数据库

核心Benchmark：SPECweb, TPC-C, TPC-D, TPC-W, Linpack，MM5，PRIS，FFT，Guass98

实际应用：较真实的硬件配置和软件环境下，用实际应用或简化的、规模缩小的实际应用评价系统的真实性能

评价计算机系统的常见误区

误区一：处理器主频越高的系统性能越好。

误区二：SPEC值越高系统性能越好。

误区三：用户A的应用运行效果很好，所以计算机系统的性能很好。

误区四：系统配置越大，性能越好。

误区五：采用最新先进技术的系统，性能越好。

评测程序缺陷

- 只局限于计算机系统的某一层次，不能得到系统整体的性能特征
- 只关心系统的性能，不关心评测结果的产生原因，无法探知系统的瓶颈，只能为用户选择系统提供帮助，不能对优化提供帮助。

本章小结

- 设计发展趋势

	Capacity	Speed
Logic	2x in 3 years	2x in 3 years
DRAM	4x in 3 years	2x in 10 years
Disk	4x in 3 years	2x in 10 years

- 运行任务的时间
  - Execution time, response time, latency
- 单位时间内完成的任务数
  - Throughput, bandwidth
- “X性能是Y的n倍”：

$$\frac{\text{ExTime(Y)}}{\text{ExTime(X)}} = \frac{\text{Performance(X)}}{\text{Performance(Y)}}$$

本章小结(续)

- **Amdahl's 定律:**

$$\text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

- **CPI Law:**

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

- 执行时间是计算机系统度量的最实际，最可靠的方式

Review

- 性能的度量
  - 评价指标
  - 性能的两种定义
- 性能设计与测试的基本原则
  - 并行性
  - 大概率事件优先原则
  - 局部性原理
  - .....
- **Amdahl's Law**

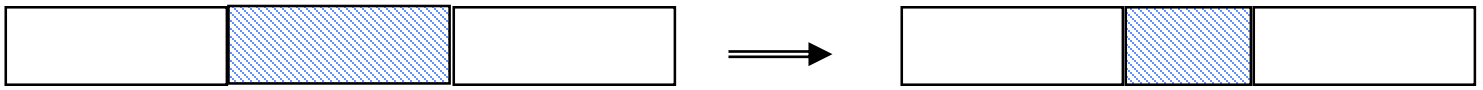
评价指标

- 执行时间（CPU时间、wall-clock time, Elapsed Time）
- 峰值速度 （Peak Performance），负载 （load）、开销（Overhead）
- 利用率 （Utilization Ratio）、饱和性能（Saturate Performance）
- 带宽 （Bandwidth）、延迟（Latency）
- 吞吐率 （Throughput）
- 加速比 （Speedup）、Amdahl定律 (Amdahl Law)
- 效率 （Efficiency）
- 基准测试　Benchmark
  - 微基准测试　Micro-benchmark：测量系统某一方面的分离性能, 如核心程序,合成测试程序等
  - 宏基准测试　Macro-benchmark：测量系统总体性能, 如实际应用程序等
- 响应时间（Response Time）
- ……

Amdahl's Law

- 假设对机器的部件进行了改进（加速比的概念）

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



- 假设可改进部分**E**在原来的计算时间所占的比例为**F**，而部件加速比为**S**，任务的其他部分不受影响，则

$$\text{ExTime(with E)} = ((1-F) + F/S) \times \text{ExTime(without E)}$$

$$\text{Speedup(with E)} = 1/((1-F)+F/S))$$

重要结论(性能提高的递减原则)：如果只针对整个任务的一部分进行优化，那么所获得的加速比不大于**1/(1-F)**

review

- **Amdahl's Law**

  Speedup(with E) = 1/((1-F)+F/S))

- **CPU time = CPI * IC * T**

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

- **基本评估方法－benchmark测试**