计算机体系结构

- 主讲：

张燕咏
邮箱： yanyongz@ustc.edu.cn)
闫宇博：
邮箱： yuboyan@ustc.edu.cn)

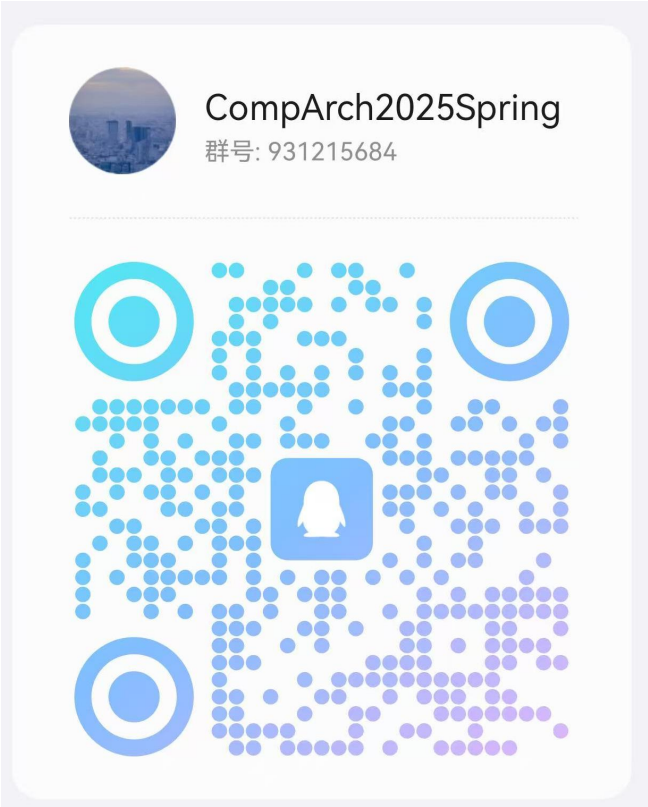
- 课程助教：

韩宇：
QQ：1450556137
邮箱：hanyu2001@mail.ustc.edu.cn
郑涵芮：
QQ：571256826
邮箱：zhr666@mail.ustc.edu.cn
牛志扬：
QQ：2070677947
邮箱：zhiyang_niu@mail.ustc.edu.cn
徐翊然：
QQ：3531448070
邮箱：alexanderxu@mail.ustc.edu.cn

- 课程主页：

http://staff.ustc.edu.cn/~comparch/index.html

- 课程群：931215684

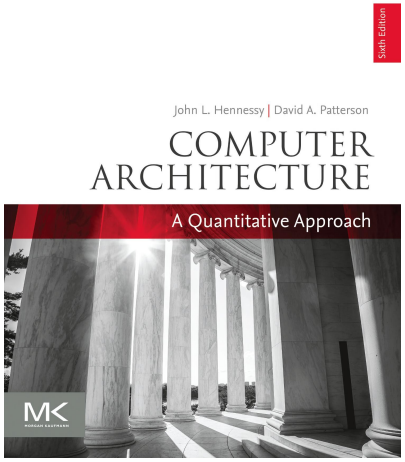


教材与主要参考书

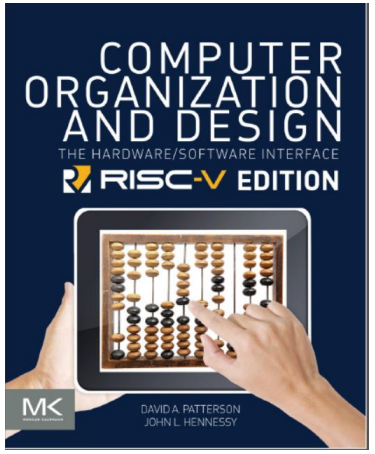
John Leroy Hennessy (born September 22, 1952) is an American computer scientist, academician, businessman, and Chair of Alphabet Inc.[5] Hennessy is one of the founders of MIPS Computer Systems Inc. as well as Atheros and served as the tenth President of Stanford University. Hennessy announced that he would step down in the summer of 2016. He was succeeded as President by Marc Tessier-Lavigne.[6] Marc Andreessen called him "the godfather of Silicon Valley."[7]

Along with David Patterson, Hennessy won the 2017 Turing Award for their work in developing the reduced instruction set computer (RISC) architecture, which is now used in 99% of new computer chips.[8]

David Andrew Patterson (born November 16, 1947) is an American computer pioneer and academic who has held the position of Professor of Computer Science at the University of California, Berkeley since 1976. He announced retirement in 2016 after serving nearly forty years, becoming a distinguished engineer at Google.[3][4] He currently is Vice Chair of the Board of Directors of the RISC-V Foundation,[5] and the Pardee Professor of Computer Science, Emeritus at UC Berkeley.

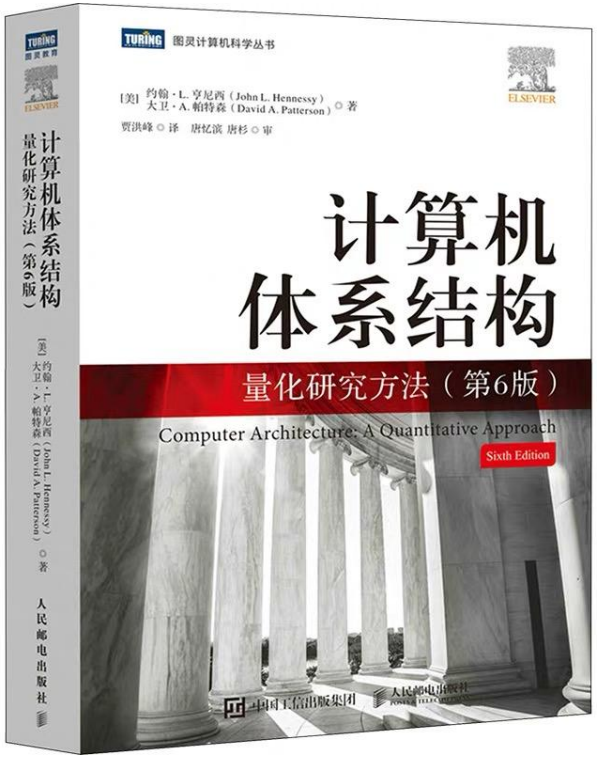


John L. Hennessy, David A. Patternson; Computer Architecture: A Quantitative Approach; sixth Edition.



David A. Patternson, John L. Hennessy ; Computer Organization and Design- The Hardware/Software Interface; RISC-V Edition.

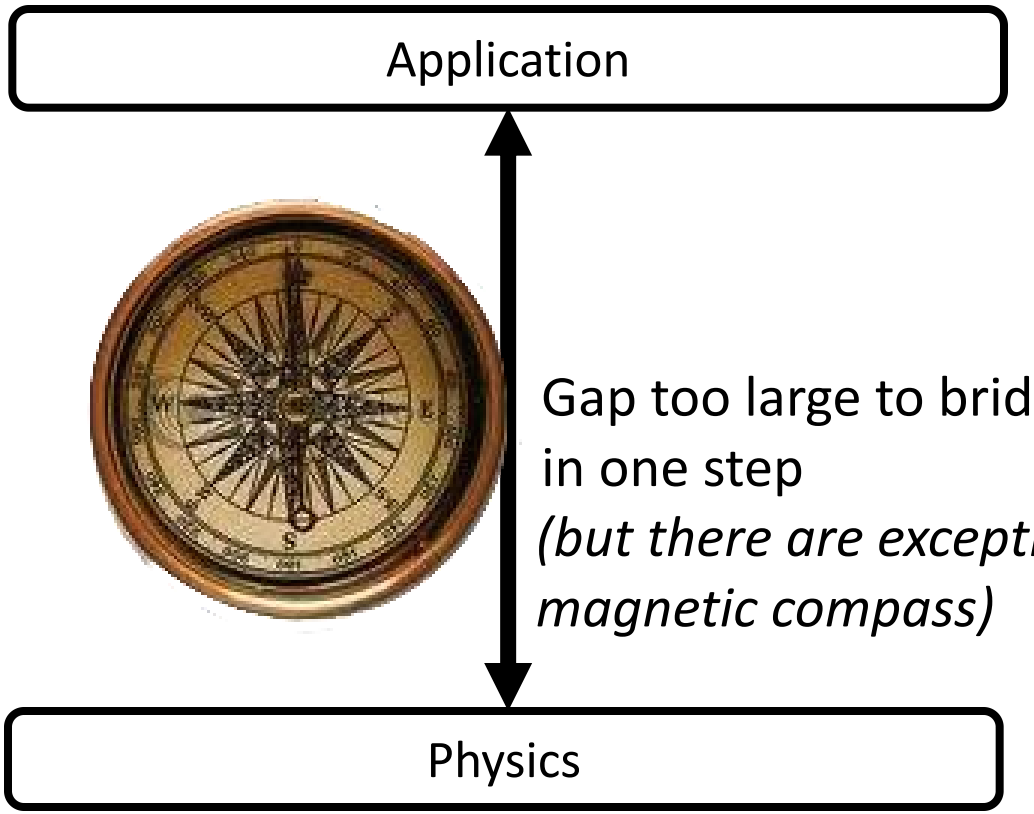
教材与主要参考书



John L. Hennessy, David A. Patternson; Computer Architecture: A Quantitative Approach. Sixth Edition. 人民邮电出版社，2022

David A. Patternson, John L. Hennessy, Computer Organization & Design : The Hardware/Software Interface, RISC-V Edition

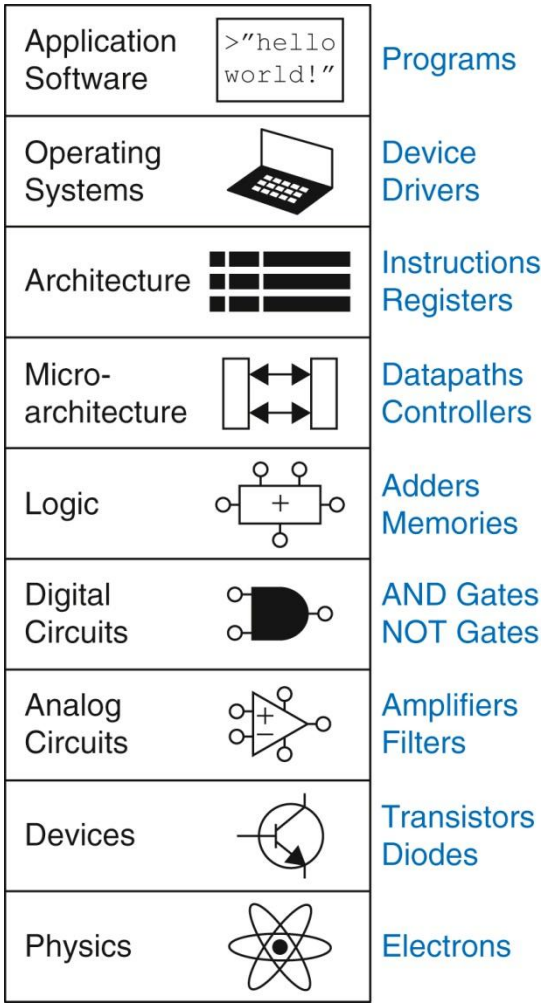
What is Computer Architecture?



广义的定义：计算机体系结构是抽象层的设计，这些抽象层使得我们可以使用可用的制造技术高效地实现信息处理应用系统

现代计算机系统的抽象层次

- Application
- Algorithm
- Programming Language
- Operating System/Virtual Machine
- Instruction Set Architecture (ISA)
- Microarchitecture 微架构
- Gates/Register-Transfer Level (RTL) 寄存器传输层
- Circuits （电路）
- Devices 设备（二极管晶体管）
- Physics 电子



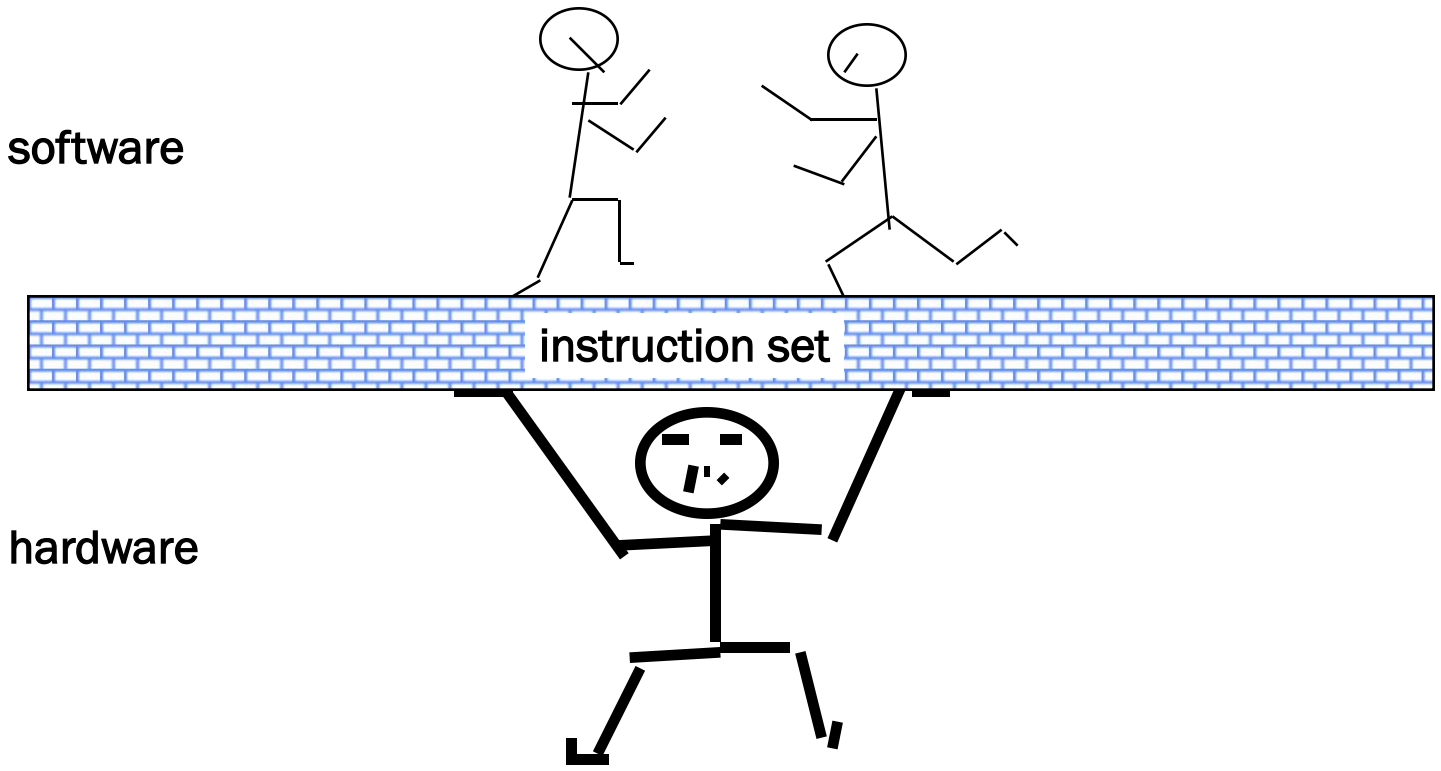
What is Computer Architecture?

- 计算机体系结构是研究如何选择（设计）功能部件和互联方法来满足计算机系统的功能、性能、价格约束的科学
- 描述计算机系统的功能、结构组织和实现的一组规则和方法
- 计算机体系结构是软件设计者与硬件设备设计者（VLSI）之间的中间层，是软件与硬件的接口（Interface）
- 计算机体系结构定义为：一组指令及机器的一系列状态

计算机体系结构的定义（续）

- **过去的观点:**
  - 指令集架构 (ISA) 设计
  - 即体系结构设计需要关注并确定:
    - 寄存器组织、存储模型、寻址方式、指令操作数、硬件支持的操作种类、指令编码方式
    - 如何处理中断和异常?
- **目前的观点:**
  - 根据目标机器的特定需求，在成本、功耗、可用性等约束下最大化机器性能
  - 包括 指令集架构（ISA），计算机组织（微体系结构），硬件实现

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
- ……
- **优秀的ISA所具有的特征**
  - 可持续用于很多代机器上(portability)
  - 可以适用于多个领域 (generality)
  - 对上层提供方便的功能（convenient functionality)
  - 可以由下层有效地实现（efficient implementation)
  - ……

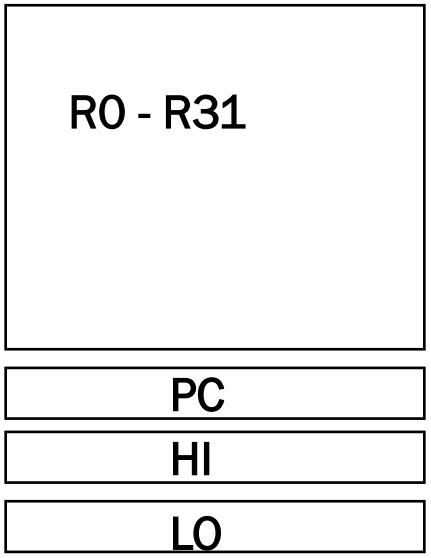
指令集结构举例

- 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
- ARM 1985-now

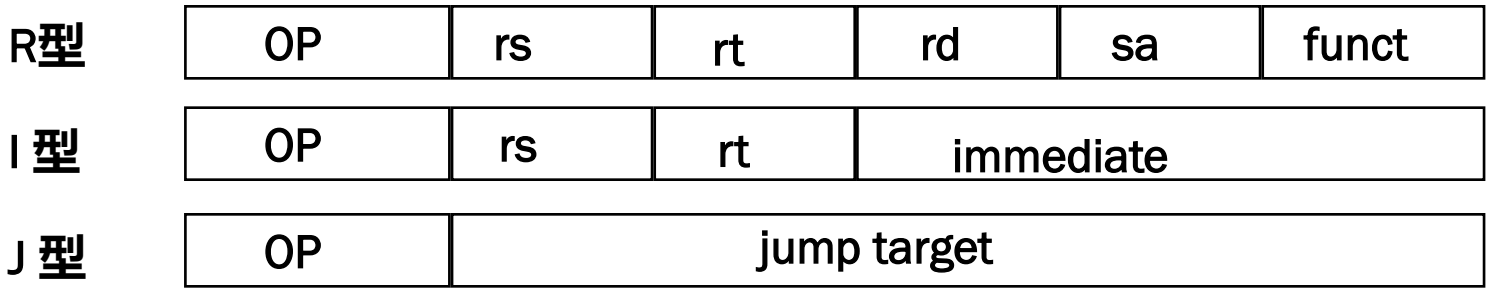
MIPS R3000 Instruction Set Architecture (Summary)

- **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

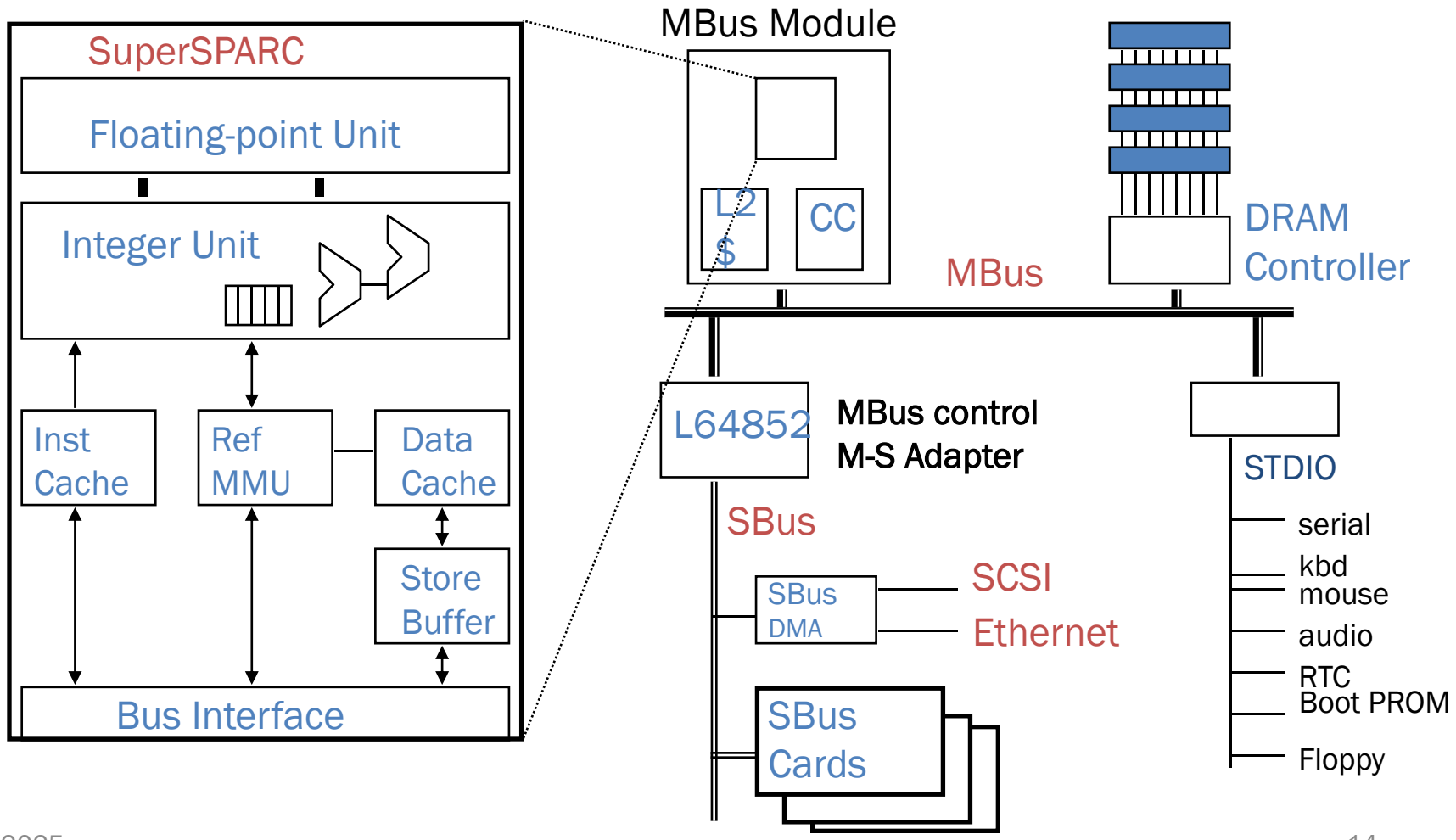


计算机组成与实现

- **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)

Example Organization

- TI SuperSPARC™ TMS390Z50 in Sun SPARCstation20



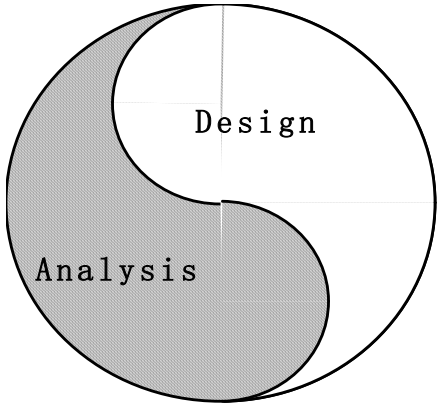
指令集架构 vs.微体系结构

- **Architecture / Instruction Set Architecture (ISA)**
  - Class of ISA: register-memory or register-register architectures
  - Programmer visible state (Register and Memory)
  - Addressing Modes: how memory addresses are computed
  - Data types and sizes for integer and floating-point operands
  - Instructions, encoding, and operation
  - Exception and Interrupt semantics
- **Microarchitecture / Organization**
  - Tradeoffs on how to implement the ISA for speed, energy, cost
  - Pipeline width and depth, cache size, peak power, bus width, execution order, etc

计算机体系结构设计者的任务

- **设计和实现不同档次的计算机系统**
  - Understand software demands
  - Understand technology trends
  - Understand architecture trends
  - Understand economics of computer systems
- **最大化性能、可编程性等指标**
  - 在一定的技术和成本的限制下
- **体系结构现状:**
  - 现代微处理器大多为多核处理器
  - 单芯片中通常集成多个处理器核心
  - 每个处理器核心支持多线程执行

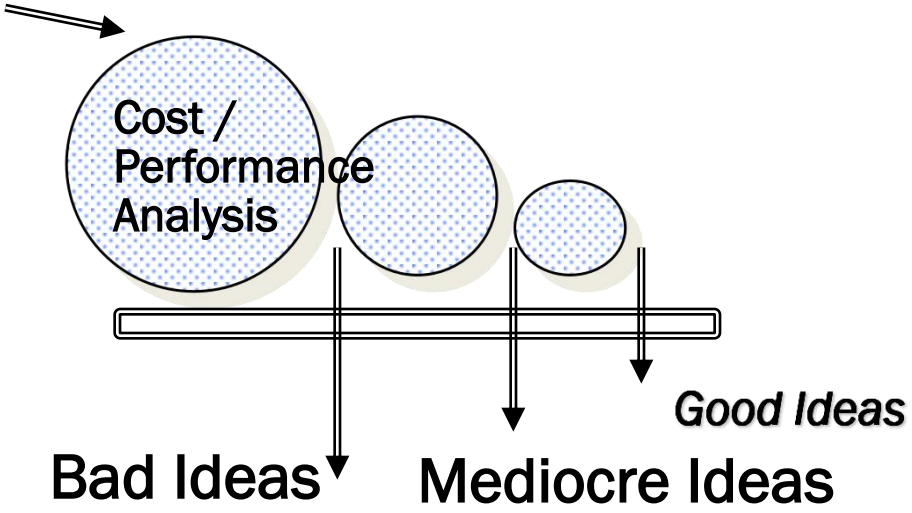
计算机体系结构设计过程

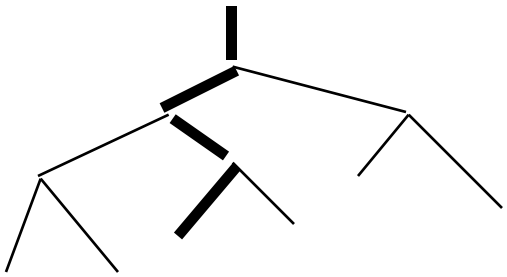


体系结构设计是循环渐进的过程:

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

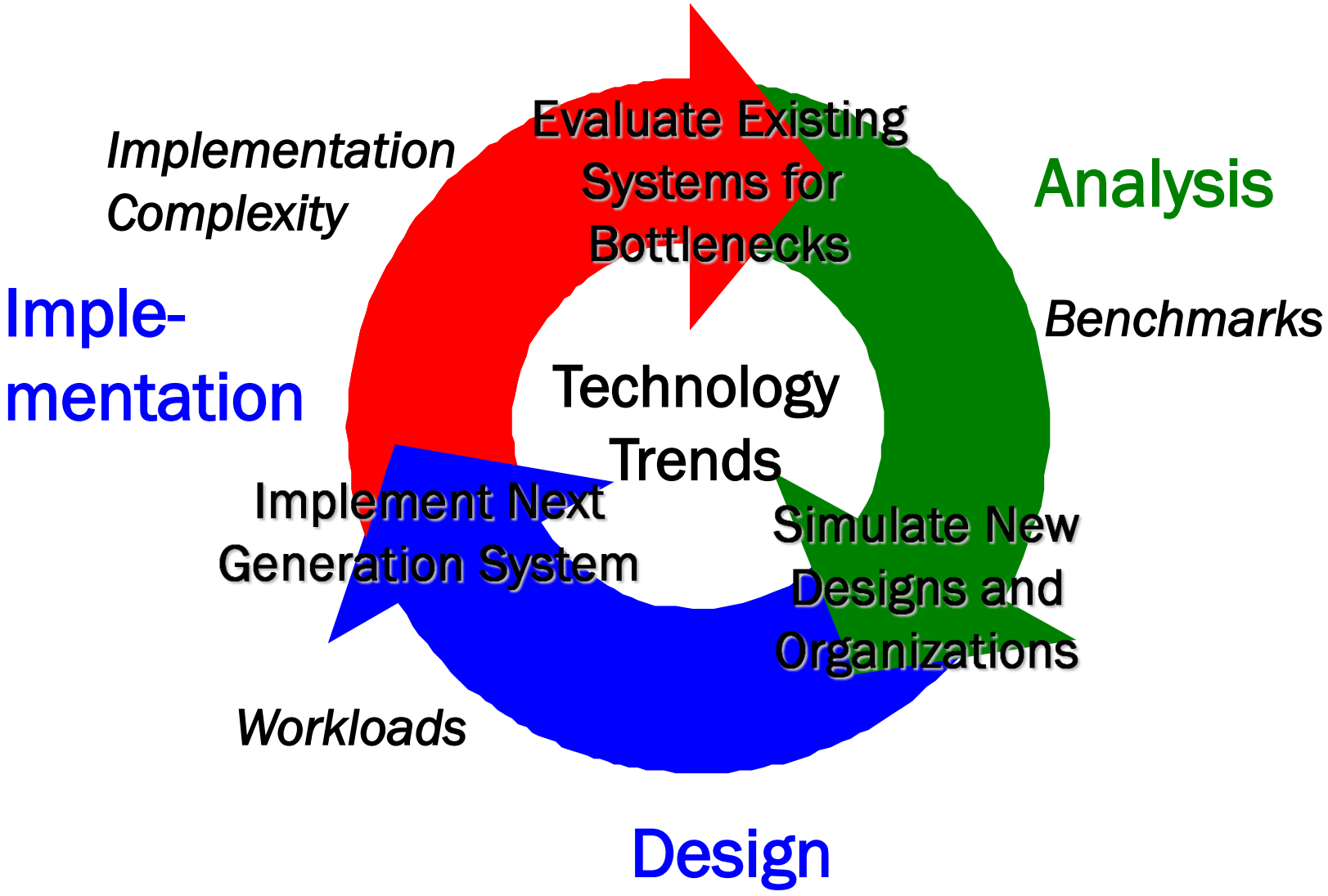
Creativity





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
Simulate New Designs and Organizations
Workloads
Design
Implement Next Generation System
Implementation Complexity
Imple-mentation
Technology Trends

课程目标

- 掌握系统**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**的基本原理和优化方法
- 理解数据级**并行**、指令级并行、线程级并行的基本原理和方法
- 初步理解面向**特定领域**的处理器设计

本课程的主要内容

- **简单机器设计（Chapter 1, Appendix A, Appendix C)**
  - ISAs, Iron Law, simple pipelines
- **存储系统(Chapter 2，Appendix B)**
  - DRAM, caches, virtual memory systems
- **指令级并行(Chapter 3)**
  - score-boarding, out-of-order issue
- **数据级并行(Chapter 4)**
  - vector machines, VLIW machines, multithreaded machines
- **线程级并行(Chapter 5)**
  - memory models, cache coherence, synchronization
- **面向特定领域的处理器体系结构（DSA)**
  - IPU、DSP、GPU

为什么学这门课

深入理解计算机体系结构:

- **开展体系结构研究与设计的基础**
  - 体系结构领域仍然存在许多挑战性问题
    - 例如：CPU与memory之间性能差异
    - 功耗和能耗问题
    - 体系结构与应用特征的适配性问题等
  - .....
- **更好地设计与实现操作系统、编译器**
  - 需要重新评估当前的假设和权衡
    - 例如: 当面临网络性能持续提升、并行系统、异构系统日益普遍
  - 现代计算机需要更好的优化编译器和更好的编程语言
- **更好地设计与实现应用程序**
  - 可更好地理解算法、数据结构和编程语言选择对性能的影响

评分规则

- **评分**
  - 平时作业　　10%
  - 随堂测验　　15%
  - 实验　　40%
  - 期终考试　　35%

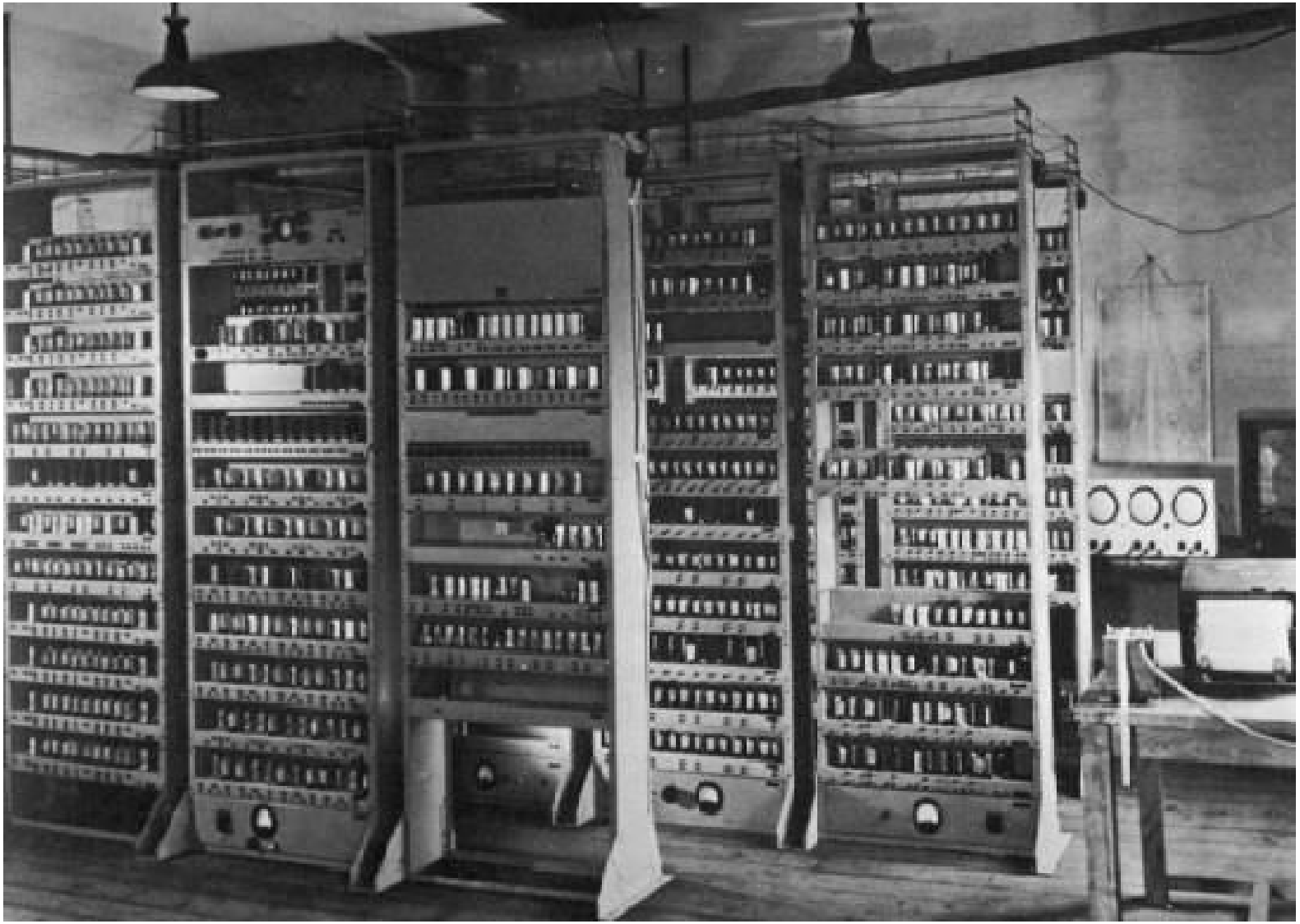
关于作弊

- **作业**
- **实验**
- **考试（测验）**
- **一旦检查出来作弊，这门课不及格**

Chapter1 量化设计与分析基础

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

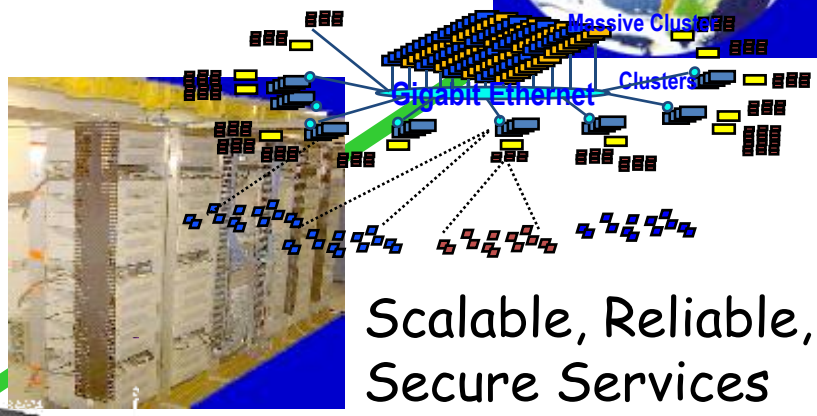
Computing Devices Then…



EDSAC, University of Cambridge, UK, 1949
EDSAC, Electronic Delay Storage Automatic Calculator

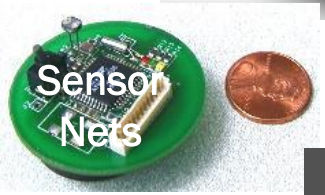
Computing Systems Today

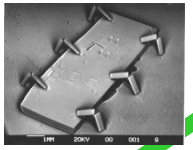
- The world is a large parallel system
  - Microprocessors in everything
  - Vast infrastructure behind them

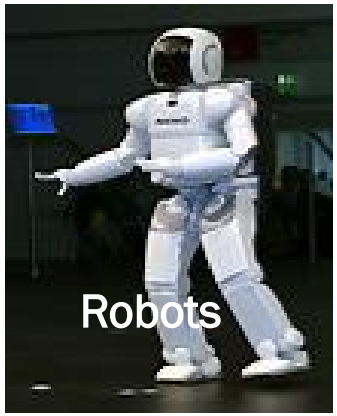


Scalable, Reliable, Secure Services

Internet Connectivity



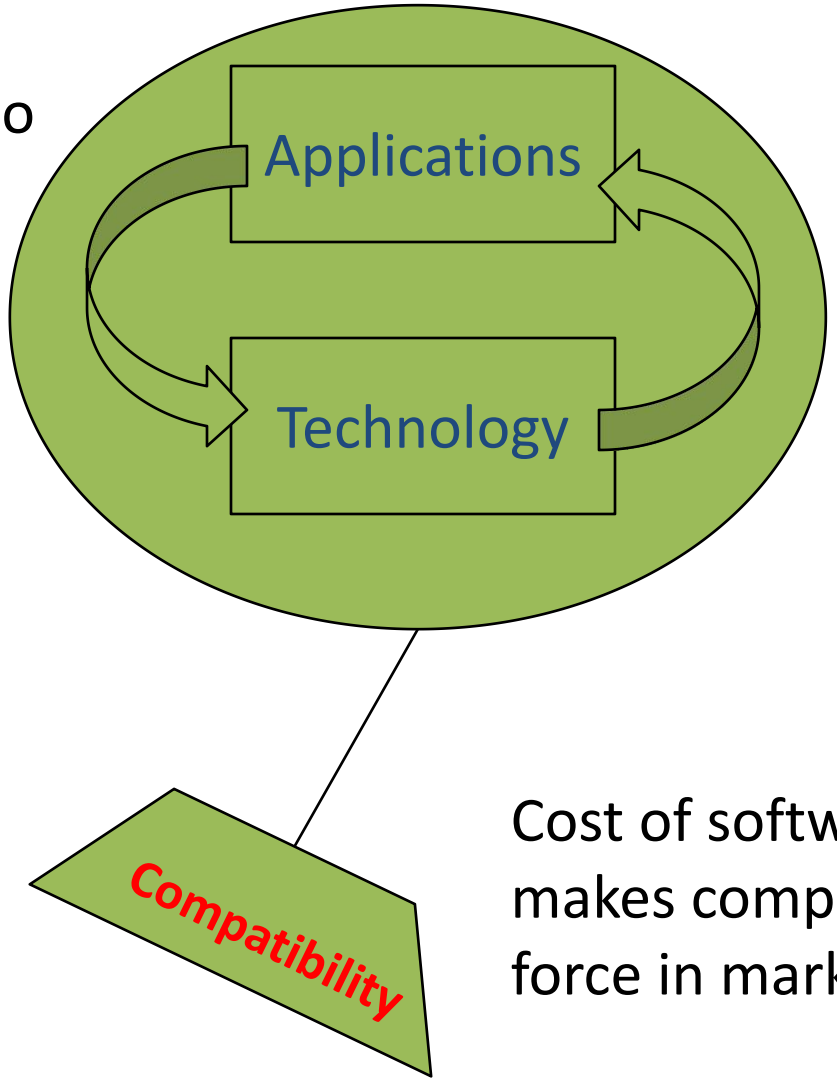




MEMS for Sensor Nets

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

体系结构的发展历史、现状及趋势*

（Dennard scaling）表明，**随着晶体管变得越来越小，它们的功率密度保持不变，因此功率的使用与面积成比例；电压和电流的规模与长度成比例。. 相**

- **体系结构发展历史**

- as transistor density increased, power consumption per transistor would drop
- Voltage and current proportional to the transistor area
- Ignored leakage current and threshold voltage
- Power wall

*A New Golden Age for Computer Architecture: Domain-Specific Hardware/Software Co-Design, Enhanced Security, Open Instruction Sets, and Agile Chip Development

John Hennessy and David Patterson
June 4, 2018
https://www.youtube.com/watch?v=3LVeEjsn8Ts
https://cacm.acm.org/magazines/2019/2/234352-a-new-golden-age-for-computer-architecture/fulltext

  - Denard Scaling 及 Moore's Law 的终结
  - Security

The number of transistors per chip, at constant cost, doubles every 18-24 months

- **体系结构发展机遇**
  - Open Architectures/open ISA (e.g., RISC-V)
  - Domain Specific Languages and Architecture
  - Agile Hardware Development

体系结构发展的机遇*

- **A New Golden Age for Computer Architecture**
  - By Hennessy and Patterson, June 4, 2018, the 45th ISCA
  - https://www.acm.org/hennessy-patterson-turing-lecture
  - Open Architectures
  - Domain Specific Architectures
  - Agile Hardware Development
  - Enhancing Security

体系结构发展的机遇*

- **Open Architectures**
  - 软件技术的进步激发体系结构创新
  - 为什么有开放的编译器、操作系统而没有开放的ISA
  - 现阶段，使用特定的ISA就要给设计方交版权费，设计方很少公开他们做某个选择的原因

RISC V 第五代Berkeley RISC

体系结构发展的机遇*

- **RISC V**
  - A practical ISA that is open-sourced
  - 很多元素来自学校的“处理器设计”相关projects
  - 拟支持嵌入式、PC、Supercomputer、向量机等
  - You can join! https://riscv.org/membership-application/

体系结构发展的机遇*

- **Domain Specific Languages and Architecture**
  - 提高性能的路径：Domain Specific Architectures(DSAs)
  - 根据应用特征调整体系结构来实现更高的效率
    - 对于特定的领域，更有效的挖掘计算并行性
    - 对于特定的领域，更有效的利用内存带宽
    - 消除不必要的精度
  - 并不是仅针对一个专门的应用，而是通过执行软件适应某一领域。
    - 例如机器学习芯片：寒武纪芯片、TPU芯片

体系结构发展的机遇*

- **Agile Hardware Development**
  - Agile development: It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change
  - Agile software development is a norm（敏捷开发）
  - Is agile hardware development possible?

Small programming teams quickly developed working-but-incomplete prototypes and got customer feedback before starting the next iteration

20世纪60年代初IBM机器兼容性问题

- **20世纪60年代初，IBM有4条产品线**
  - 701 ->7094
  - 650 ->7074
  - 702 -> 7080
  - 1401->7010

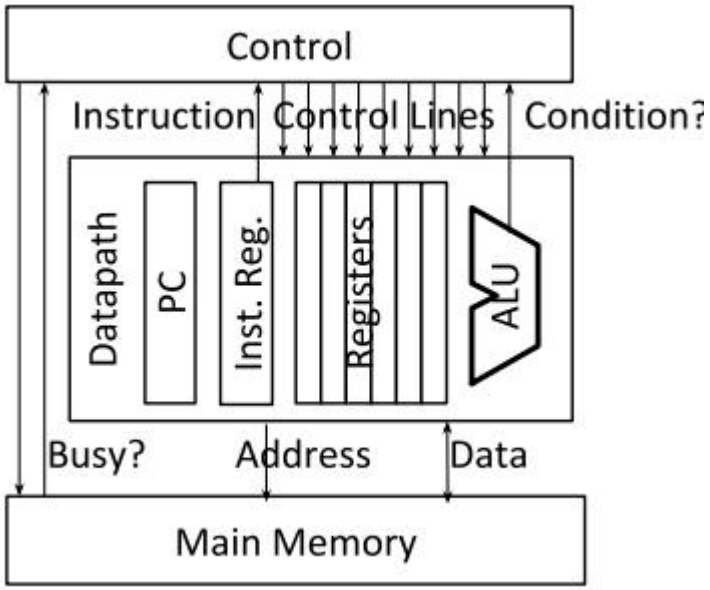
- **每个系统包含各自不同的**
  - ISA
  - I/O 系统及二级存储：磁带、磁鼓和磁盘
  - 汇编器、编译器、库等
  - 市场定位：商业应用、科学计算、实时系统

系列机：IBM System/360 – one ISA to rule them all

The **IBM System/360** (**S/360**) is a family of mainframe computer systems that was announced by IBM on April 7, 1964, and delivered between 1965 and 1978.[1] It was the first family of computers designed to cover the complete range of applications, from small to large, both commercial and scientific. The design made a clear distinction between architecture and implementation, allowing IBM to release a suite of compatible designs at different prices. All but the only partially

数据通路vs控制

- 处理器设计可划分为两部分
  - Datapath： 存储和操作数据
  - Control：产生控制信号作用于数据通路
- 过去对处理器设计师的最大的挑战是产生正确的控制序列



* "Micro-programming and the desig computer,"
M. Wilkes, and J. Stringer. Mathemat 49, 1953.

- ***Maurice Wilkes* 发明了微程序设计方法来设计控制部件* (1953)**
  - Stored program vs circuitry
  - Logic expensive vs. ROM or RAM
  - ROM cheaper than RAM
  - ROM much faster than RAM

Microprogramming, Process of writing microcode for a microprocessor. Microcode is low-level code that defines how a microprocessor should function when it executes machine-language instructions. Typically, one machine-language instruction translates into several microcode instructions. On some computers, the microcode is stored in ROM and cannot be modified; on some larger computers, it is stored in EPROM and therefore can be replaced with newer versions.

IBM 360系列机的控制部件

Model	M30	M40	M50	M65
Datapath width	8 bits	16 bits	32 bits	64 bits
Microcode size	4k x 50	4k x 52	2.75k x 85	2.75k x 87
Clock cycle time (ROM)	750 ns	625 ns	500 ns	200 ns
Main memory cycle time	1500 ns	2500 ns	2000 ns	750 ns
Price (1964 $)	$192,000	$216,000	$460,000	$1,080,000
Price (2018 $)	$1,560,000	$1,760,000	$3,720,000	$8,720,000

Fred Brooks, Jr.

Frederick Phillips "Fred" Brooks Jr. (born April 19, 1931) is an American computer architect, software engineer, and computer scientist, best known for managing the development of IBM's System/360 family of computers and the OS/360

IC技术、微程序技术催生CISC

- 逻辑电路、RAM和ROM使用同样的晶体管实现
- 半导体RAM的速度与ROM的速度基本相同
- 控制存储密度会持续增长（Moores Law)
- 由于采用RAM存储控制信号，容易修复微代码bug

- 综上允许更加复杂的ISAs
- CISC: Complex Instruction Set Computers
- 例如：小型机（TTL server)
  - Digital Equipment Corp. (DEC)
  - VAX ISA in 1977
  - 5K × 96b 微码

Writable Control Store

- **如果控制存储是RAM，那么就可以定制“固件”应用程序：“Writable Control Store”**
- **微程序研究在学术界很流行**
  - Patterson Phd Thesis*
  - 有专门的国际会议SIGMICRO
- **Xerox Alto （Bit Slice TTL） (1973)**
  - 第1台具有GUI和网络的个人计算机
  - BitBlt和网络控制器用微码实现

Chuck Thacker

* Verification of microprograms, David Patterson, UCLA, 1976
** “The design of a system for the synthesis of correct microprograms,” David Patterson, Proc. 8th Annual Workshop of Microprogramming, 1975

微处理器演进

- **上世纪70年代，在CMOS技术进步的推动下，小型计算机和主机ISAs得到了迅速发展**
- **“Microprocessor Wars”：通过添加指令，调整汇编器**
- **Intel iAPX 432：最具雄心的微型计算机始于1975年**
  - 基于32位能力的面向对象体系结构，使用Ada编写的定制操作系统
  - 严重的性能、复杂性(多芯片)和可用性问题导致1981年发布
- **Intel 8086 (1978, 8MHz, 29,000 transistors)**
  - 要求：“Stopgap” 16-bit processor, 52 周开发新的芯片
  - 结果：10人3周开发了汇编级兼容8080的ISA
- **1981年IBM 采用Intel8088 （8位数据总线）研制了IBM PC机**
  - 希望1986年 销售25万台，实际销售1+亿台
  - PC软件的二进制兼容=> 8086前途光明

80年初：微程序控制机器分析

- **用高级语言编程成为主流**
  - 关键问题：编译器会生成什么指令？(ISA vs. Compiler)
- **IBM的John Cocke团队**
  - 为小型计算机801（ECL Server）开发了更简单的 ISA 和编译器
  - 移植到IBM370，仅使用IBM 370的简单的寄存器-寄存器及load/store指令
  - 发现：与原IBM 370相比，性能提高3X
- **80年代初，Emer和Clark （DEC）发现**
  - VAX 11/180 CPI = 10!
  - VAX ISA 的 20%指令 （占用了60%的微码）仅占用了 0.2%的执行时间
- **Patterson：如何修复微处理器中的微程序bug，投稿’ 79DEC 后，引发对ISA合理性的研究**

* "A Characterization of Processor Performance in the VAX-11/780," J. Emer and D.Clark, ISCA, 1984.
** "RISCy History," David Patterson, May 30, 2018, Computer Architecture Today Blog

From CISC to RISC

- **CISC指令系统主要存在以下几方面的问题：**
  - 指令使用频度；CISC指令系统复杂-〉增加研制时间和成本，容易出错；
  - VLSI设计困难，不利于单片集成；
  - 许多复杂指令操作复杂，运行速度慢；
  - 各条指令不规整，不利于先进计算机体系结构技术来提高系统的性能。
- **使用简单ISA**
  - 指令像微指令那样简单
  - 编译生成的指令仅是部分CISC指令
  - 能够采用流水线方式实现
- **技术基础**
  - 使用指令Cache
  - 芯片集成度有很大提高：80年代初，单芯片已经可以集成32位数据通路+小规模cache
  - Chaitin的寄存器分配方法有利于Load-store型ISA

*Chaitin, Gregory J., et al. "Register allocation via coloring." Computer languages 6.1 (1981), 47-57

CISC vs. RISC Today

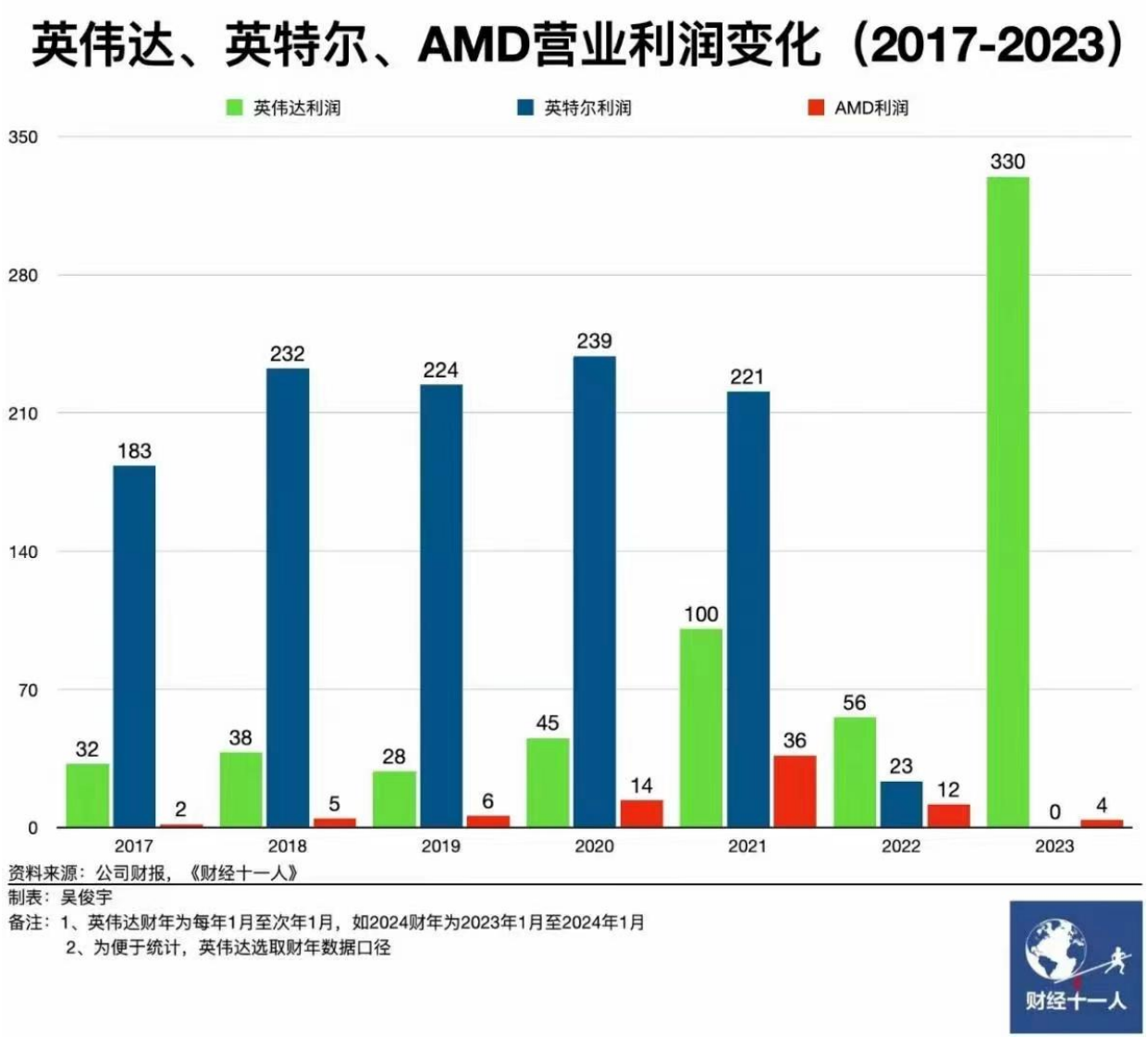
PC 时代

- 硬件将x86指令翻译为内部的RISC 指令
- 在MPU(micro-processing uit)内部使用RISC技术
- x86 ISA在桌面和服务器市场占主导地位
- >350M / year

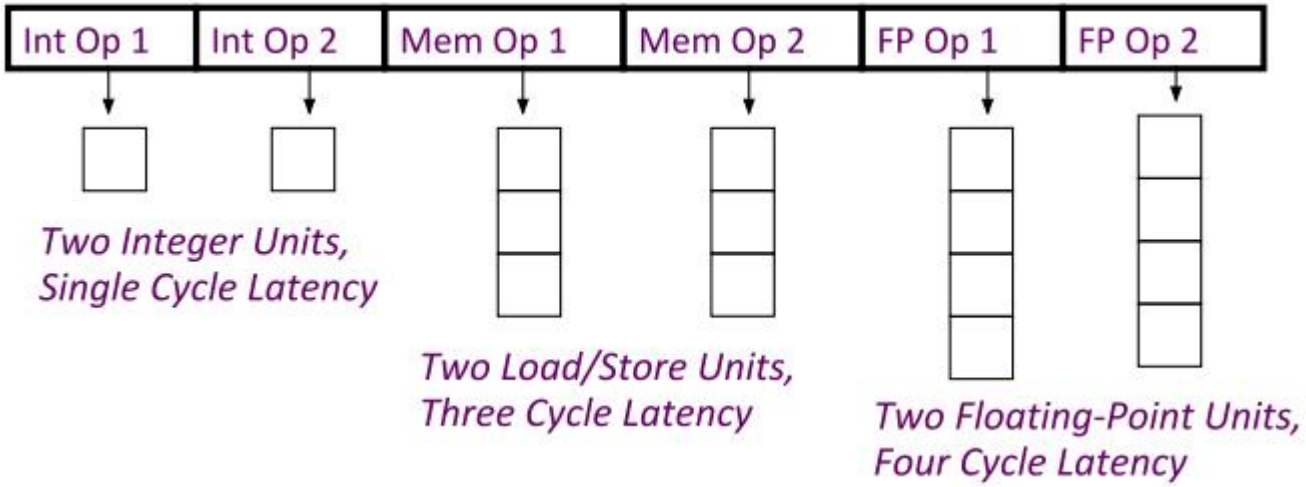
后PC时代: Client/Cloud

- SoC中的处理器核 vs. MPU
- 芯片面积、能耗和性能同样重要
- 99%的处理器是RISC
- >20B total/year in 2017
  - x86在2011年达到顶峰，现在每年下降~8%，
  - x86服务器=>Cloud ~10M servers
  - Total (0.05% of 20B)

Today


英伟达、英特尔、AMD营业利润变化（2017-2023）
英伟达利润
英特尔利润
AMD利润
350
280
210
140
70
0
32
183
2
38
232
5
28
224
6
45
239
14
100
221
36
56
23
12
330
0
4
2017
2018
2019
2020
2021
2022
2023
资料来源：公司财报，《财经十一人》
制表：吴俊宇
备注：1、英伟达财年为每年1月至次年1月，如2024财年为2023年1月至2024年1月
2、为便于统计，英伟达选取财年数据口径
财经十一人

VLIW: Very Long Instruction Word



- 一条指令中包含多个操作
- 每个指令槽表示固定的功能
- 指定了恒定的操作延时
- 体系结构必须保证
  - 指令中的操作是并行的 => no cross-operation RAW check
  - 数据未准备好不使用数据 => no data interlocks

From RISC to Intel/HP Itanium，EPIC IA-64

- **EPIC 是Intel为他们的VLIW结构的命名**
  - Explicitly Parallel Instruction Computing
  - 二进制 目标码兼容的 VLIW
  - 从1994年与HP合作开发
- **EPIC IA-64 是 Intel 32位x86的后继（64位ISA）**
  - IA-64= Intel Architecture 64-bit
  - AMD 有自己的AMD64 技术,2003年推出业界首款64位处理器
  - 第一款Itanium 2002年推出，不兼容IA-32
  - 很多公司放弃RISC转而选择Itanium，因为他们普遍认为这是必然的（Microsoft, SGI, Hitachi,...

VLIW的问题及EPIC的失败

- 编译器无法处理整型类代码(指针)中的复杂依赖项
- 代码量膨胀
- **不可预知的分支**
- **可变的存储访问延迟（不可预知的cache失效）**
  - 乱序执行技术可处理Cache延迟
- **乱序执行覆盖了VLIW的优势**
- *The Itanium approach...was supposed to be so terrific –until it turned out that the wished-for compilers were basically impossible to write."*
  - Donald Knuth, Stanford

小结：体系结构发展历史

- **重要事件**
  - IBM 系列机、微程序设计、RISC、VLIW、EPIC
- **30年来没有出现新的通用CISC ISA**
- **15年来没有出现新的通用VLIW**
  - 在通用计算领域VLIW是失败的
  - 复杂的VLIW结构接近顺序超标量结构，但在大型复杂应用中不存在优势
  - VLIW在嵌入式DSP市场比较成功
    - 简单VLIW，分支简单、没有Cache，小程序
- **RISC！普遍认为RISC原则是通用ISA的最好原则**

Eight Great Ideas in Computer Architecture

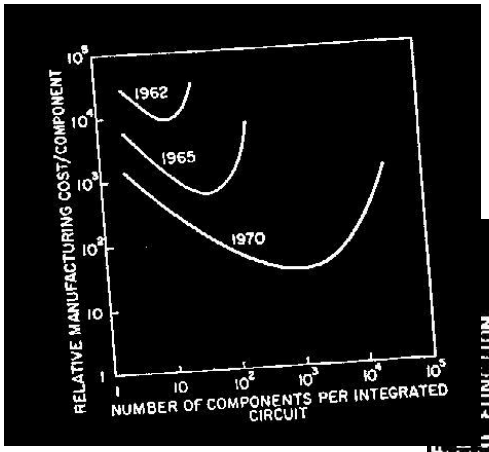
These are eight great ideas that computer architects have invented in the last 60 years of computer design. They are so powerful they have lasted long after the first computer that used them, with newer architects demonstrating their admiration by imitating their predecessors -- Patterson

https://www.elsevier.com/connect/8-great-ideas-in-computer-architecture

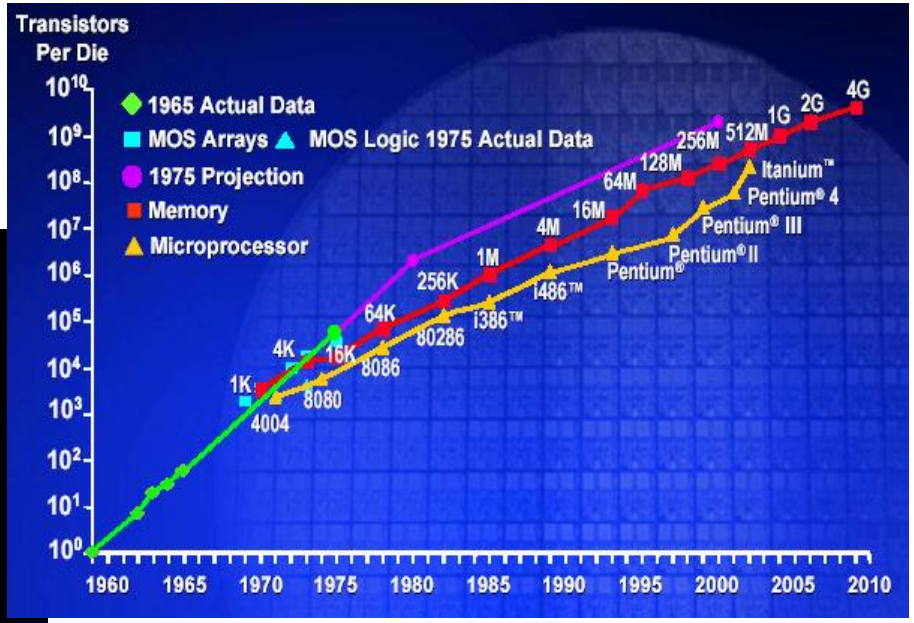
Eight Great ideas in Computer Architecture

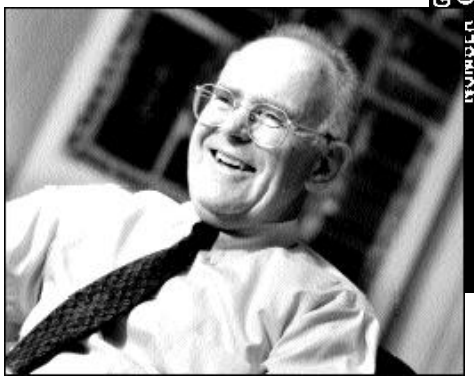
1. Design for Moore’s Law
2. Abstraction to Simplify Design
3. Make the Common Case Fast
4. Dependability via Redundancy
5. Memory Hierarchy
6. Performance via Parallelism
7. Performance via Pipelining
8. Performance via Prediction

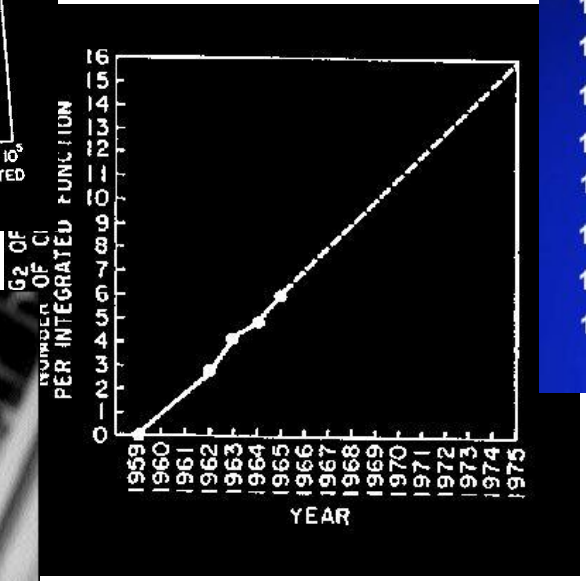
1. Moore's Law（摩尔定律）



Who is Moore?







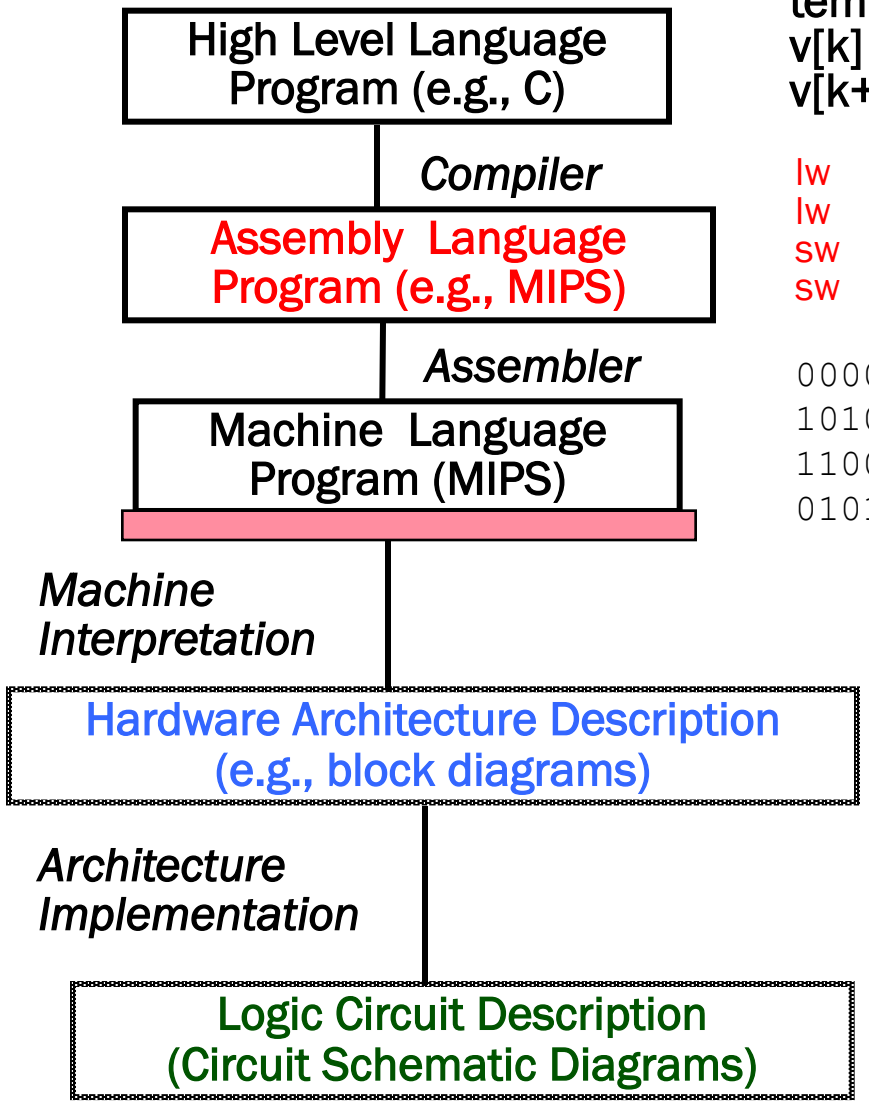
1．集成电路芯片上所集成的电路的数目，每隔18个月就翻一番。
2．微处理器的性能每隔18个月提高一倍，或价格下降一半。
3．用一个美元所能买到的电脑性能，每隔18个月翻两番。

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

1. Design for Moore’s Law

The one constant for computer designers is rapid change, which is driven largely by Moore's Law. It states that integrated circuit resources double every 18–24 months. Moore's Law resulted from a 1965 prediction of such growth in IC capacity made by Gordon Moore, one of the founders of Intel. As computer designs can take years, the resources available per chip can easily double or quadruple between the start and finish of the project. Like a skeet shooter, computer architects must anticipate where the technology will be when the design finishes rather than design for where it starts.

2. Abstraction via Layers of Representation




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw    $t0, 0($2)
lw    $t1, 4($2)
sw    $t1, 0($2)
sw    $t0, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Abstraction uses multiple levels with each level hiding the details of levels below it. For example:

The instruction set of a processor hides the details of the activities involved in executing an instruction.

High-level languages hide the details of the sequence of instructions needed to accomplish a task.

Operating systems hide the details involved in handling input and output devices.

2. Use Abstraction to Simplify Design

Both computer architects and programmers had to invent techniques to make themselves more productive, for otherwise design time would lengthen as dramatically as resources grew by Moore's Law. A major productivity technique for hardware and software is to use abstractions to represent the design at different levels of representation; lower-level details are hidden to offer a simpler model at higher levels. We'll use the abstract painting icon to represent this second great idea.

GIVE ME AN EXAMPLE?

3. Make the Common Case Fast

- 在进行设计选择时，注重优化经常发生的事件
- 优化不经常执行的代码意义不大
- 选择一种（性能）度量方式来确定经常性事件（Common case)

你的任务是把人以最快速度从A点送到B点，绝大多数情况下只有一个人。你选哪辆车？

在设计ISA时该如何利用这个idea?

如何知道哪个指令最常运行？

4. Dependability via Redundancy

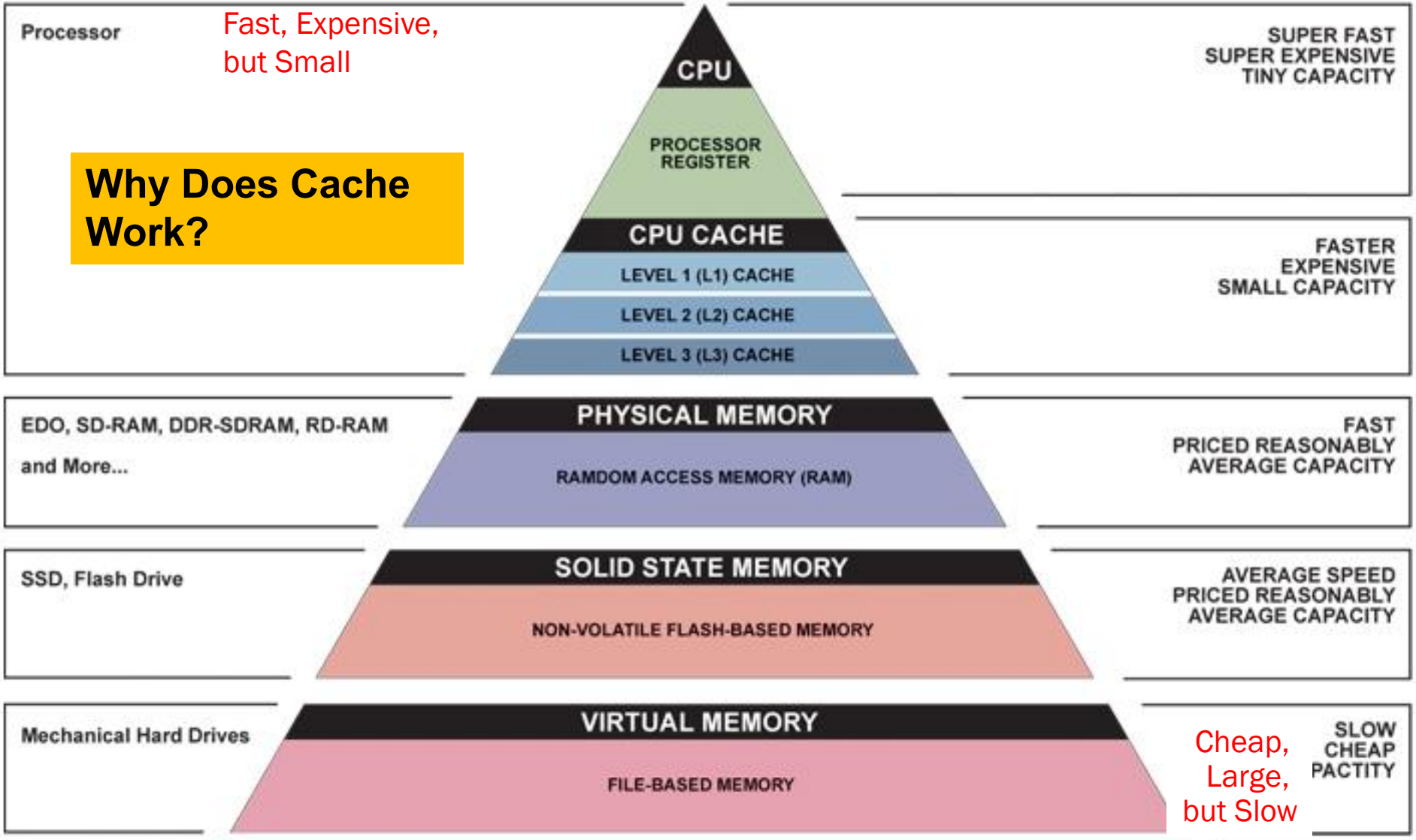
- **通过冗余使得部分部件失效不影响整个系统的运行**

假设你的预算为T，是买一台单价为T的大容量磁盘，还是买100台单价为T/100的廉价磁盘？

Storage servers with 24 hard disk drives and built-in hardware RAID controllers supporting various RAID levels

One of the most important ideas in data storage is the Redundant Array of Inexpensive Disks (RAID) concept. In most versions of RAID, data is stored redundantly on multiple disks. The redundancy insures that if one disk fails the data can be recovered from other disks.

5. Memory Hierarchy



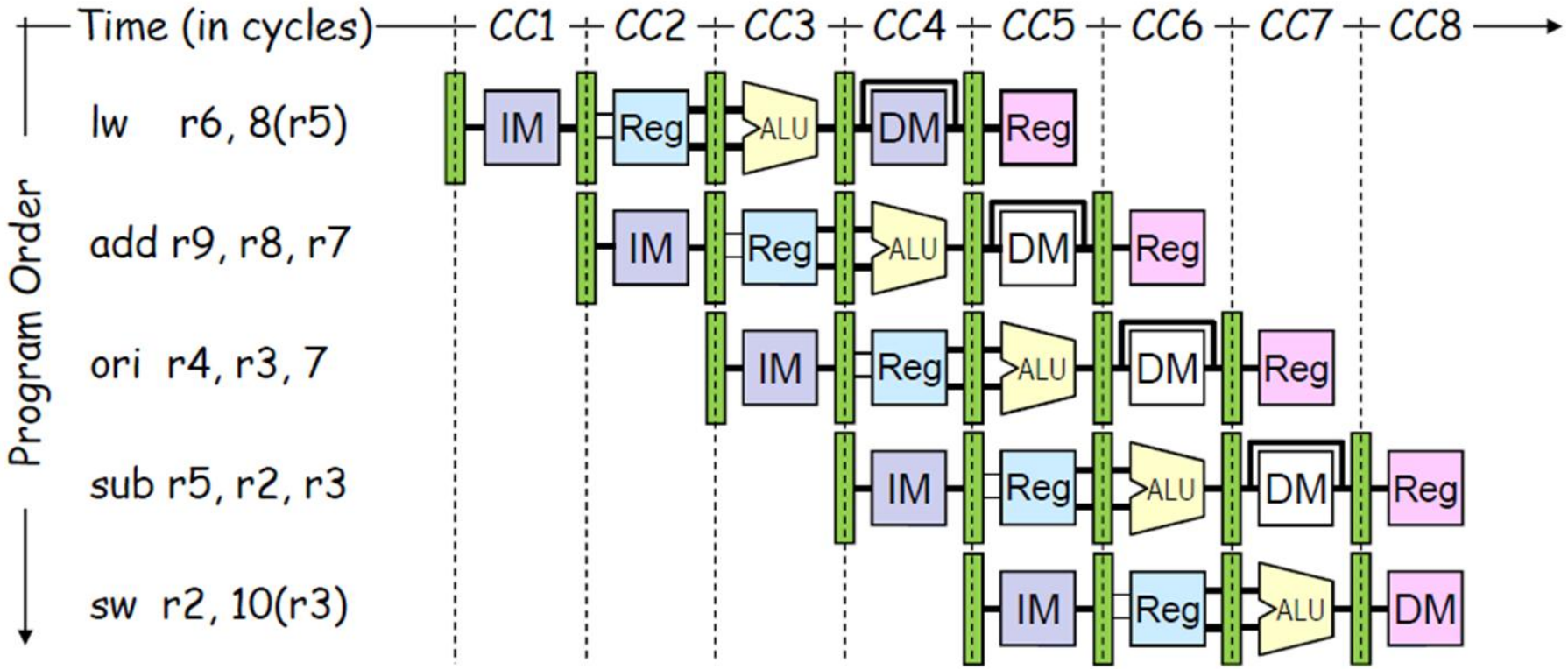
6. Performance Through Parallelism

Doing different parts of a task in parallel accomplishes the task in less time than doing them sequentially. A processor engages in several activities in the execution of an instruction. It runs faster if it can do these activities in parallel.

- Parallel Requests
  Assigned to computer, e.g., Search “Katz”
- Parallel Threads,
  Assigned to core, e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time, GPU is SIMD (Single Instruction Multiple Word)
- Hardware Descriptions
  All gates functioning in parallel at same time

Performance Through Pipelining

This idea is an extension of the idea of parallelism. It is essentially handling the activities involved in instruction execution as an assembly line. As soon as the first activity of an instruction is done you move it to the second activity and start the first activity of a new instruction. This results in executing more instructions per unit time compared to waiting for all activities of the first instruction to complete before starting the second instruction.

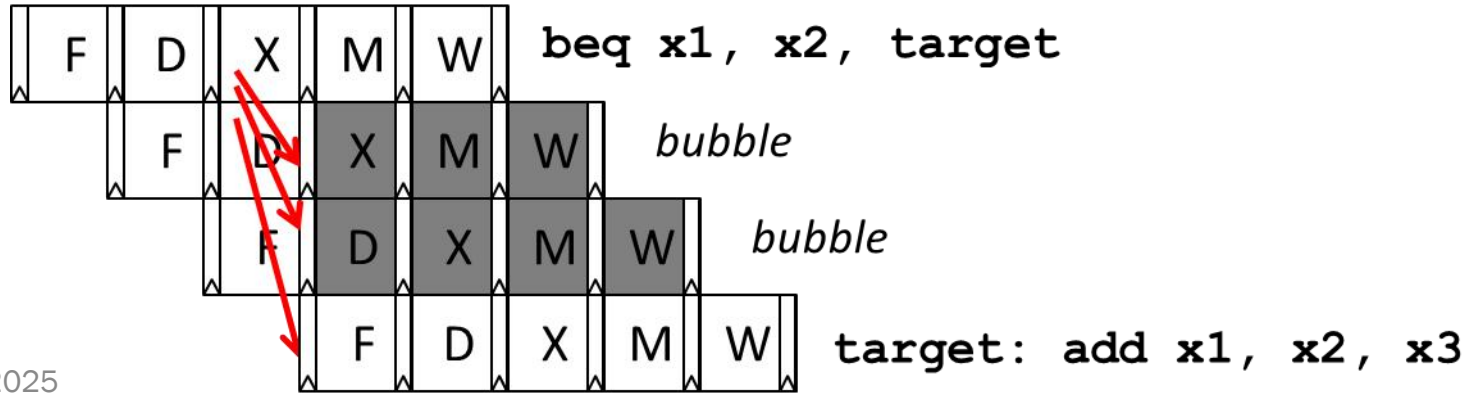


8. Performance Through Prediction

A conditional branch is a type of instruction that determines the next instruction to be executed based on a condition test. Conditional branches are essential for implementing high-level language if statements and loops.

Unfortunately, conditional branches interfere with the smooth operation of a pipeline — the processor does not know where to fetch the next instruction until after the condition has been tested.

Many modern processors reduce the impact of branches with speculative execution: make an informed guess about the outcome of the condition test and start executing the indicated instruction. Performance is improved if the guesses are reasonably accurate and the penalty of wrong guesses is not too severe.



Classes of Computers

计算机的分类

- **个人移动设备 (PMD)**
  - 智能手机、平板电脑
  - ARM-ISA兼容的通用处理器芯片（SoC）在市场上处于统治地位
  - SoC中包含大量的专用加速器 (radio, image, video, graphics, audio, motion, location, security, etc.)
  - 强调能效和实时性(energy efficiency and real-time)
- **桌面计算（Desktop Computing)**
  - 强调性价比(price-performance)
- **服务器（Servers)**
  - 强调可用性、可缩放性、吞吐率(availability, scalability, throughput)

Cost of downtime

Application	Cost of downtime per hour	Annual losses with downtime of 1% (87.6 h/year)	0.5% (43.8 h/year)	0.1% (8.8 h/year)
Brokerage service	$4,000,000	$350,400,000	$175,200,000	$35,000,000
Energy	$1,750,000	$153,300,000	$76,700,000	$15,300,000
Telecom	$1,250,000	$109,500,000	$54,800,000	$11,000,000
Manufacturing	$1,000,000	$87,600,000	$43,800,000	$8,800,000
Retail	$650,000	$56,900,000	$28,500,000	$5,700,000
Health care	$400,000	$35,000,000	$17,500,000	$3,500,000
Media	$50,000	$4,400,000	$2,200,000	$400,000

Figure 1.3 Costs rounded to nearest $100,000 of an unavailable system are shown by analyzing the cost of downtime (in terms of immediately lost revenue), assuming three different levels of availability, and that downtime is distributed uniformly. These data are from Landstrom (2014) and were collected and analyzed by Contingency Planning Research.

availability

"Nines" [edit]

Main article: Nine (purity)

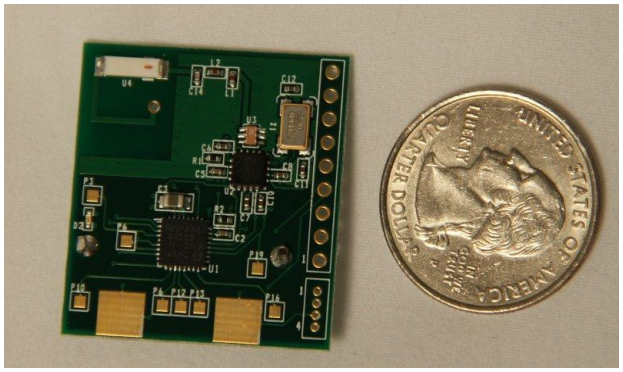
Percentages of a particular order of magnitude are sometimes referred to by the number of nines or "class of nines" in the digits. For example, electricity that is delivered without interruptions (blackouts, brownouts or surges) 99.999% of the time would have 5 nines reliability, or class five.[2] In particular, the term is used in connection with mainframes[3][4] or enterprise computing, often as part of a service-level agreement.

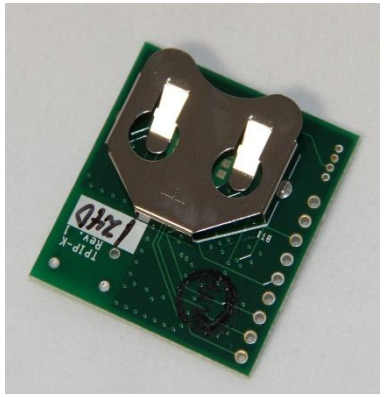
计算机的分类（续）

- **集群/仓储级计算机（Clusters / Warehouse Scale Computers)**
  - 每个数据中心包含10+万个处理器核
  - X86-ISA兼容的服务器芯片在市场上占统治地位
  - 专门的应用程序，加上虚拟机的云托管
  - 现在越来越多地使用gpu、fpga和定制硬件来加速工作负载
  - 其分支：Supercomputers：强调浮点运算性能和高速内部互联
  - 强调可用性和性价比(availability、price-performance、energy)

计算机的分类（续）

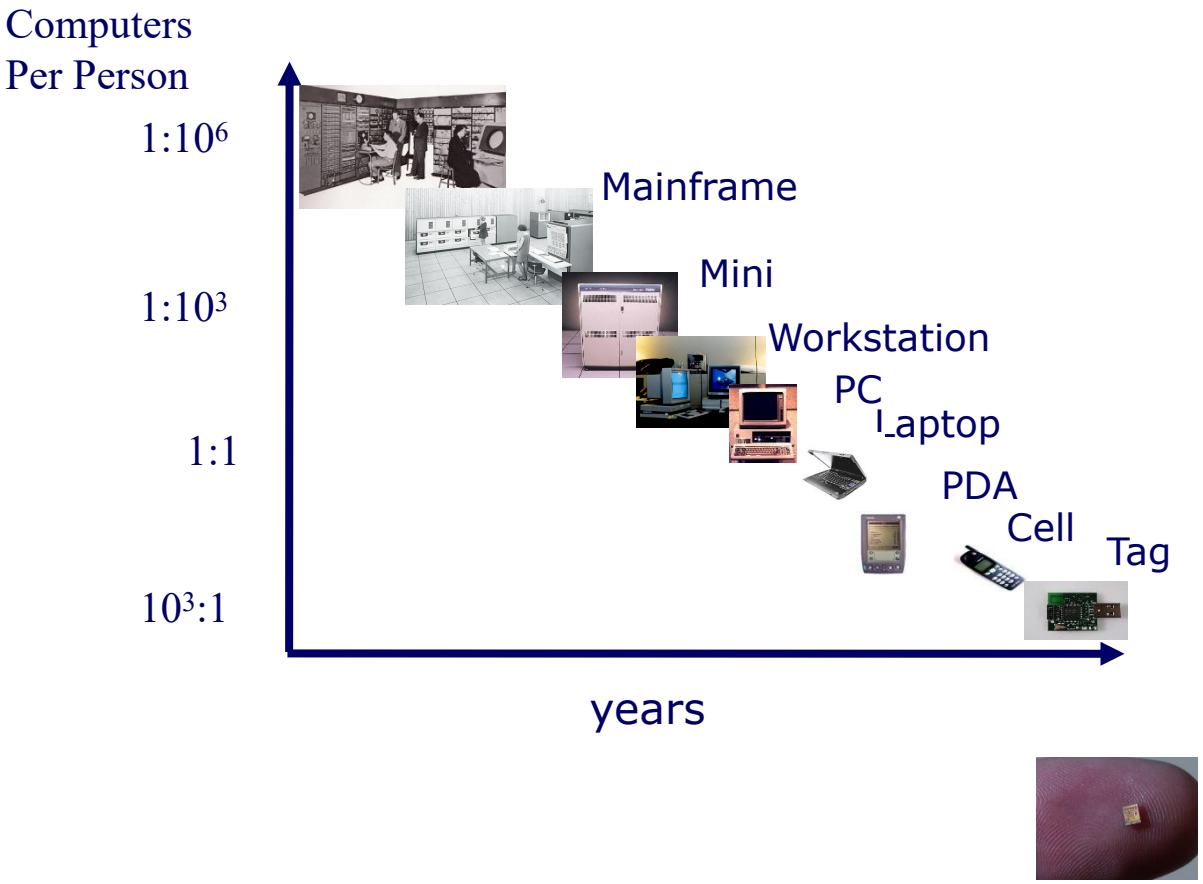
- **嵌入式计算机/物联网（Embedded Computers / Internet of Things）**
  - 有线/无线网络基础设施，打印机
  - 消费类产品
    (TV/Music/Games/Automotive/Camera/MP3)
  - 物联网 (Internet of Things!)
  - 强调价格、能耗及面向特定应用的性能(Emphasis: price、energy、application-specific performance)





The Opportunity

Bell's Law: a new computer class emerges every 10 years



五类主流计算系统特点

Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Internet of things/ embedded
Price of system	$100–$1000	$300–$2500	$5000–$10,000,000	$100,000–$200,000,000	$10–$100,000
Price of microprocessor	$10–$100	$50–$500	$200–$2000	$50–$250	$0.01–$100
Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality	Price, energy, application-specific performance

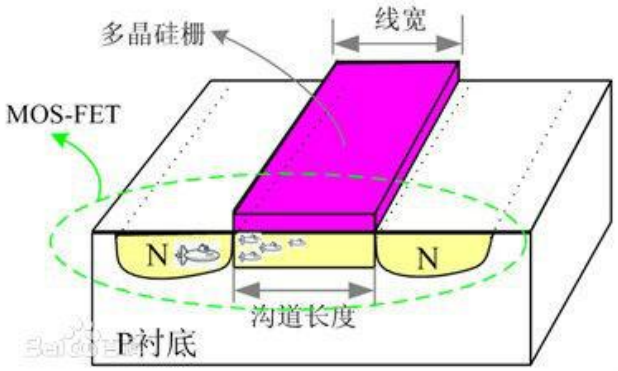
Figure 1.2 A summary of the five mainstream computing classes and their system characteristics. Sales in 2015 included about 1.6 billion PMDs (90% cell phones), 275 million desktop PCs, and 15 million servers. The total number of embedded processors sold was nearly 19 billion. In total, 14.8 billion ARM-technology-based chips were shipped in 2015. Note the wide range in system price for servers and embedded systems, which go from USB keys to network routers. For servers, this range arises from the need for very large-scale multiprocessor systems for high-end transaction processing.

Closer Look at Processor Technology

Closer Look at Processor Technology

- **特征尺寸不断减小（feature size ），芯片集成度不断提高**
  - 电路的速度在不断提高
  - 芯片的面积在不断增加 （10%-20% ）
  - 主频在提高 (功耗成为问题) $C \cdot V^2 \cdot A \cdot f$
  - 单位面积的晶体管数量在增加
- **性能每10年提高100倍**
  - 时钟频率提高了10倍
  - DRAM 的密度每3年提高4倍
- **如何使用这些晶体管?**
  - Parallelism in processing: 有更多的功能部件
    - 每个时钟周期并行多个操作降低了CPI
  - Locality in data access: 更大的Cache
    - 降低了访存的延时从而降低了CPI，提高了处理器的利用率

什么是特征尺寸？(x, y) 维度上最小的尺寸

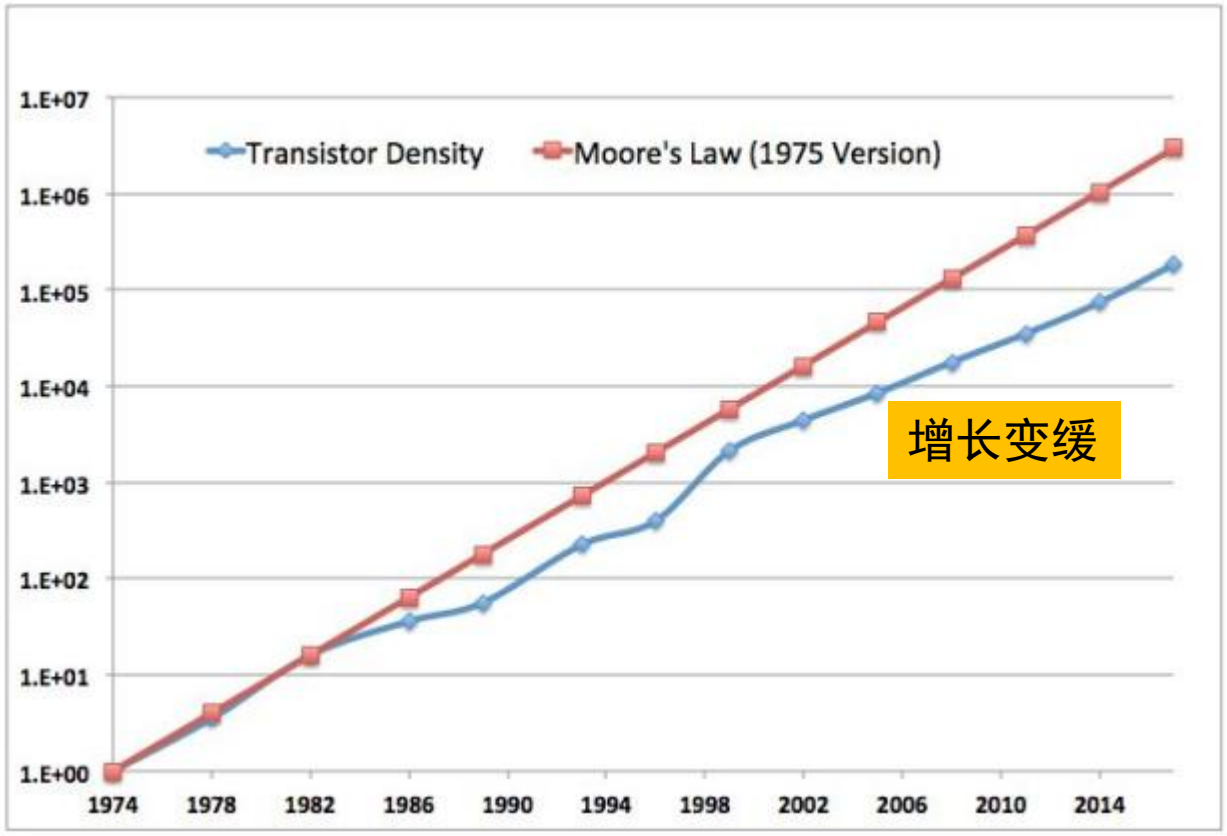


金属氧化物半导体场效应晶体管

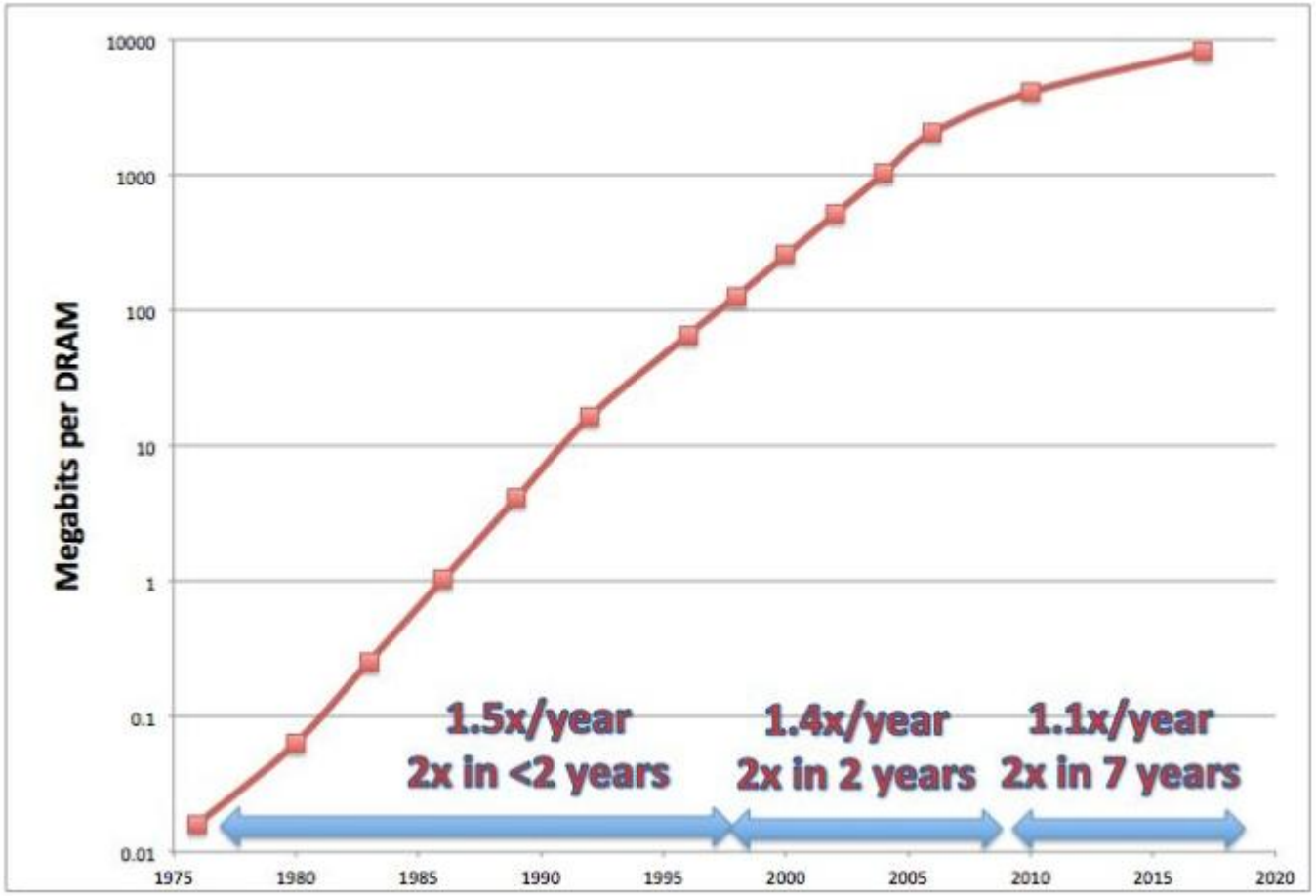
Architecture: Increase in Parallelism

- **1985年以前位级(4-8-16位)并行（Bit level parallelism）**
  - 32-bit 处理器后性能提升变慢
  - 90年代采用64-bit 、128-bit 及更高位 通过向量处理提高性能
  - 重大的拐点：32位处理器和缓存适合(fit)一个芯片
- **80年代中到90年代后期主要发展指令级并行（Instruction Level Parallelism）**
  - 流水线、RISC、编译技术的进步→ 挖掘指令级并行
  - 片上cache及功能部件的增加-> 超标量执行(superscalar)
  - 更精巧的技术: 乱序执行(out of order execution)和硬件投机执行(speculative execution)
- **现状：线程级并行和片上多处理器（thread level parallelism and chip multiprocessors）**
  - 线程级并行超越了指令级并行
  - 在单个芯片中部署多个处理器及互联结构
  - 在处理器芯片内多个线程并行执行 硬件线程+ 软件线程

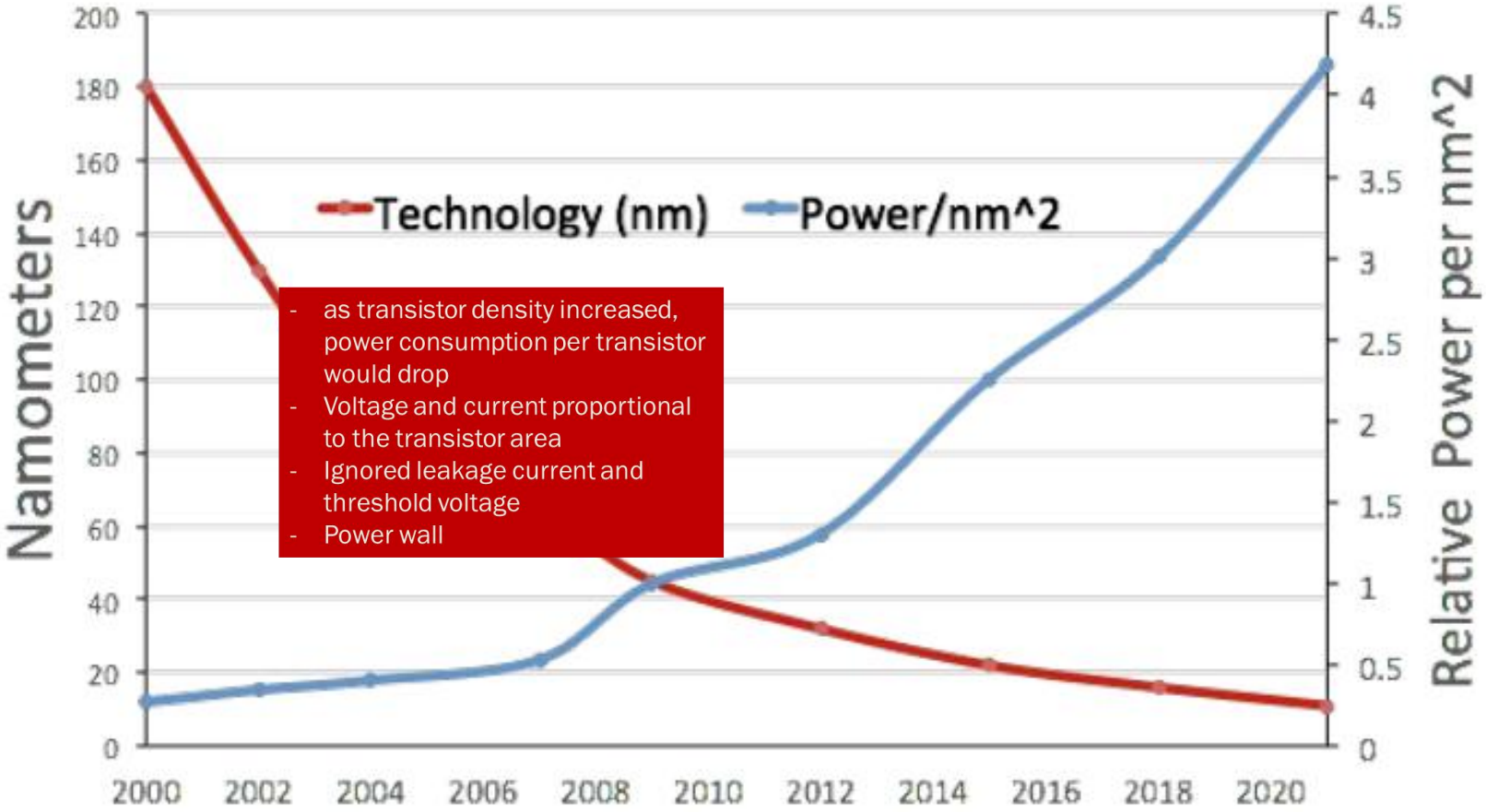
Moore’s Law Slowdown in Intel Processors



Moore’s Law in DRAMs



Technology & Power: Dennard Scaling



Energy scaling for fixed task is better, since more and faster transistors

Why is low-power Important?

- **Power must be brought in and distributed around the chip**
- **Power is dissipated as heat and must be removed**
- **Three things to consider**
  - Peak power （电源能够提供的功率 < 处理器需要的最大功率 => 电压下降 => 无法正常工作)
  - Thermal design power (TDP)热设计功耗， 决定了电路的冷却需求 （often peak power >TDP> 平均功率 ）

Power in Integrated Circuits

- **功耗问题是当今计算机设计面临的最大挑战之一**
  - 芯片内部和外围电路都存在功耗问题
  - 功耗会产生热，须解决散热问题
- **热设计功率（Thermal Design Power (TDP)）**
  - Intel：热设计功率，一种基于最坏情况应用的功率耗散目标
  - 表达了设备的功耗特征
  - 主要用于电源和冷却系统的设计
  - 通常它低于峰值功耗，但高于平均功耗
- **TDP和功耗的关系?**
  - 答案是没有关系。很多测试表明，CPU的最大真实功耗小于TDP值
- **降低频率可导致功耗下降**

The **thermal design power** (**TDP**), sometimes called **thermal design point**, is the maximum amount of heat generated by a computer chip or component (often a CPU, GPU or system on a chip) that the cooling system in a computer is designed to dissipate under any workload.

Some sources state that the peak power rating for a microprocessor is usually 1.5 times the TDP rating.[1]

Thermal Design Power

- **散热设计的依据**
  - 若散热器能力 < TDP → 芯片会因过热降频（throttling），导致性能下降
  - 若散热器能力 >= TDP → 芯片可稳定运行在标称性能

- **系统设计的考虑**
  - 高性能芯片（游戏CPU或服务器GPU）通常TDP 较高（100W 以上），需要更大规模的散热方案（如液冷）
  - 主要用于电源和冷却系统的设计
  - 移动设备（如笔记本电脑）追求低 TDP（15-28W），通过限制功耗来延长续航并减少发热。
  - 芯片厂商通过 TDP 定义产品的“性能档位”。例如，同一架构的 CPU 可能有 35W、65W、95W 等多个 TDP 版本，对应不同的核心数和频率

- **行业趋势**
  - 现代芯片（如 Apple M 系列、AMD Ryzen）通过动态功耗管理（如调节电压/频率）在 TDP 范围内灵活分配功耗，提升能效比。例如，优先冷却高负载核心，而非均匀分配热量

Power versus Energy

- **功耗(Power) 指单位时间的能耗：1 Watt = 1 Joule / Second**
- **一个任务执行的能耗 (energy) = Average Power × Execution Time**
- **Power or Energy? 哪个指标更合适?**
  - 针对给定的任务，能耗是一种更合适的度量指标（joules)
  - 针对电池供电的设备，我们需要关注能效
- **Example: which processor is more energy efficient?**
  - Processor A consumes 20% more power than B on a given task
  - However, A requires only 70% of the execution time needed by B
- **Answer: Energy consumption of A = 1.2 × 0.7 = 0.84 of B**
  - Processor A consumes less energy than B (more energy-efficient)

Power & Energy

- **Dynamic Energy动态能耗 ∝**

  Capacitive Load（电容负载） × Voltage²

  – 晶体管在开关时，从0-1-0 或 1-0-1逻辑跃迁的脉冲能量

  – Capacitive Load =输出晶体管和导线的电容负载

    - 确实呈现出逐渐降低的趋势，但这一过程并非线性，且受多重因素制约

  – 20年来晶体管供电电压已经从5V降到1V

- **Dynamic Power ∝**

  时钟周期

  Capacitive Load × Voltage² × Frequency Switched

动态能耗和功耗

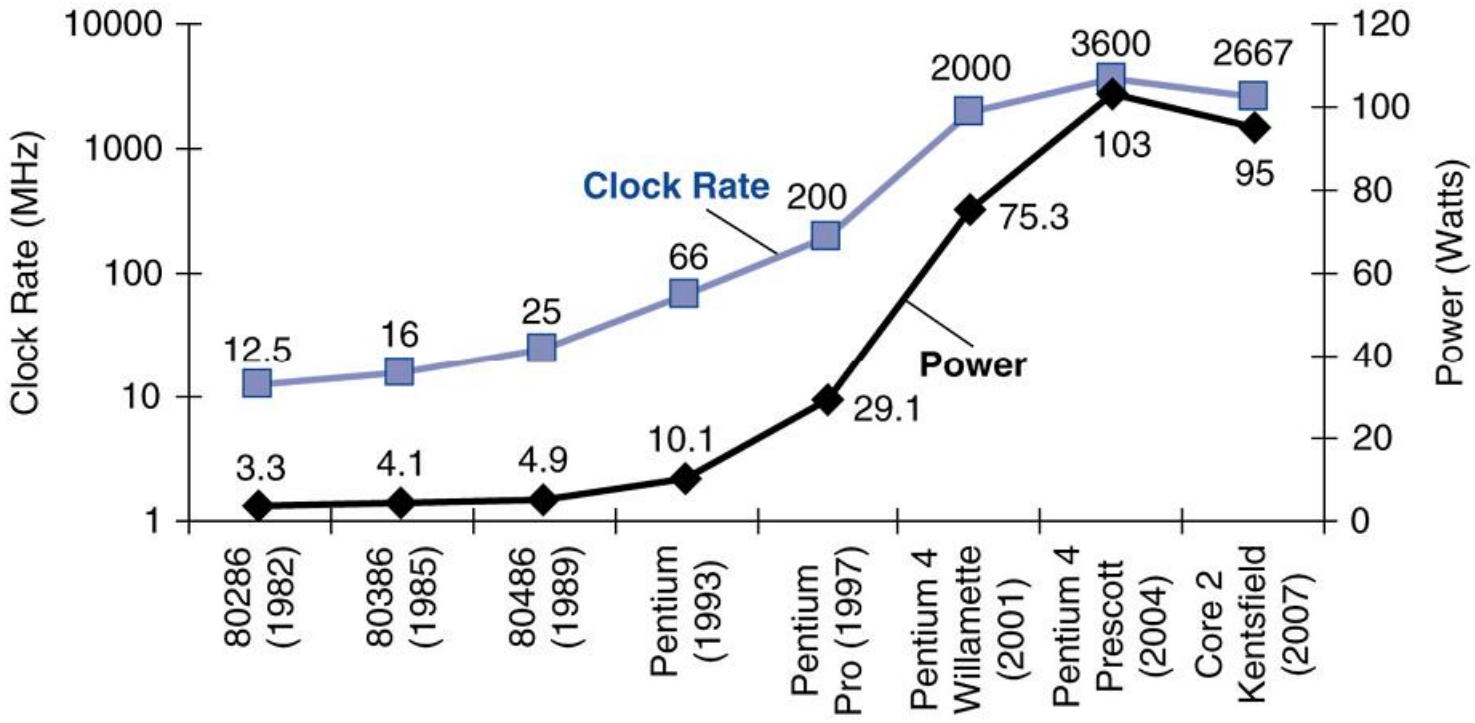
- **针对CMOS技术，动态的能量消耗是由于晶体管的on和off状态的切换引起的**

- **Dynamic Energy ∝ Capacitive Load × Voltage²**
  - the energy of pulse of the logic transition of 0-1-0 or 1-0-1
  - Capacitive Load = Capacitance of output transistors & wires
  - Voltage has dropped from 5V to below 1V in 20 years

- **Dynamic Power ∝ Capacitive Load × Voltage²× Frequency Switched**

- **降低频率可以降低功耗**
- **降低频率导致执行时间增加 ->不能降低能耗**
- **降低电压可有效降低功耗和能耗**

Trends in Power & Clock Frequency



$$\text{Power} \propto \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

30X

5V→1V

1000X

减少动态功耗的技术

- **关闭不活动模块或处理器核的时钟(Do nothing well)**
  - Such as a floating-point unit when there are no FP instructions
- **Dynamic Voltage-Frequency Scaling (DVFS)**
  - 在有些场景不需要CPU全力运行
  - 降低电压和频率可降低功耗
- **针对典型场景特殊设计 (Design for the typical case)**
  - 电池供电的设备(手机、笔记本等)常处于idle状态，DRAM和外部存储采用低功耗模式工作以降低能耗
- **Overclocking (Turbo Mode)**
  - 当在较高频率运行时，先以较高频率在少量核上运行一段时间，直到温度开始上升至不安全区域。一个core以较高频率运行，同时关闭其他核

特性	超频（Overclocking）	涡轮模式（Turbo Mode）
控制主体	用户手动调整（BIOS/软件）	芯片内置算法自动控制
频率提升幅度	通常更高（极限超频可达20-50%）	保守（通常5-20%）
持续时间	可持续（依赖散热能力）	短暂（受TDP/温度限制）
电压管理	需手动优化（过高导致发热，过低导致不稳定）	芯片动态调节（基于预定义曲线）
适用场景	发烧友、极限性能需求	普通用户、能效敏感场景

Dynamic Voltage Scaling (DVS)

- **DVS scales the operating voltage of the processor along with the frequency.**
- **Since energy is proportional to v^2, ($v \propto f$) DVS can potentially provide significant energy savings through frequency and voltage scaling => DVFS**
- **can reduce voltage and frequency by (say) 10%; can slow a program by (say) 8%, but reduce dynamic power by 27% (1-.9*.9*.9), reduce total power by (say) 23%, reduce total energy by 17% (1-.77*1.08)**

有一次注意到当服务器的工作数量增加时，完成同一个工作需要的时间却变短了。能解释下这可能是为什么吗？

低功耗技术-DVFS

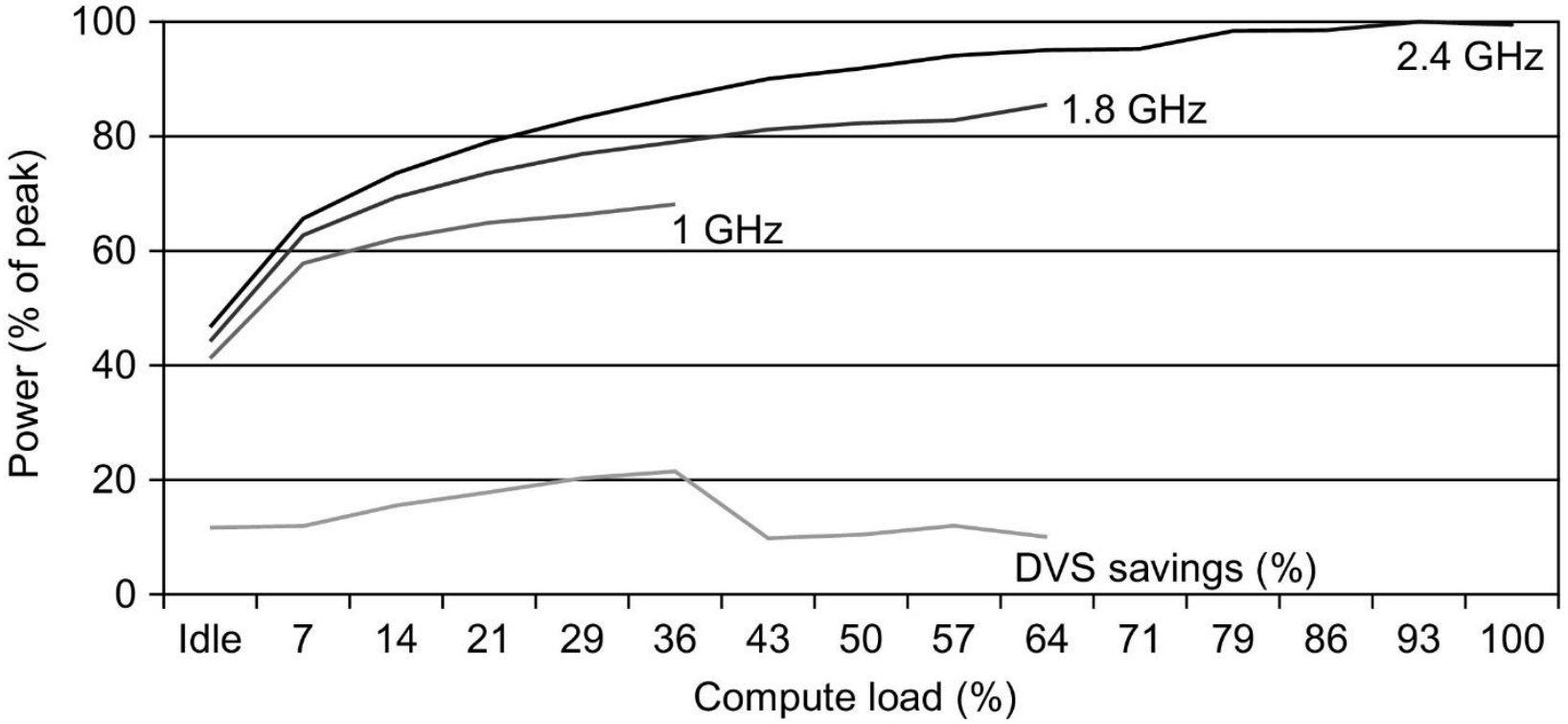


Figure 1.12 Energy savings for a server using an AMD Opteron microprocessor, 8 GB of DRAM, and one ATA disk. At 1.8 GHz, the server can handle at most up to two-thirds of the workload without causing service-level violations, and at 1 GHz, it can safely handle only one-third of the workload (Figure 5.11 in Barroso and Hölzle, 2009).

静态功耗（Static Power）

- **当晶体管处于off状态时,漏电流产生的功耗称为静态功耗**
- **晶体管越小，漏电越严重**
  - 65nm工艺，静态功耗占总功耗约20%
  - 7nm工艺：静态功耗占比可达40-60%（高频运算时动态功耗仍主导，但待机时静态功耗成主要问题）
- **Static Power = Static Current × Voltage**
  - Static power increases with the number of transistors
- **静态功耗有时会占到全部功耗的50%**
  - Large SRAM caches need static power to maintain their values
  - 手机在睡眠模式下，静态功耗可能消耗总电量的30%以上
  - 全球数据中心约25%的能耗来自空闲状态（静态功耗主导），每年浪费数百亿美元。
- **Power Gating: 通过切断供电减少漏电流**
  - To inactivate modules to control the loss of leakage current
  - 挑战：欢迎延迟（微秒级）和状态保存开销

有关功耗和能耗小结

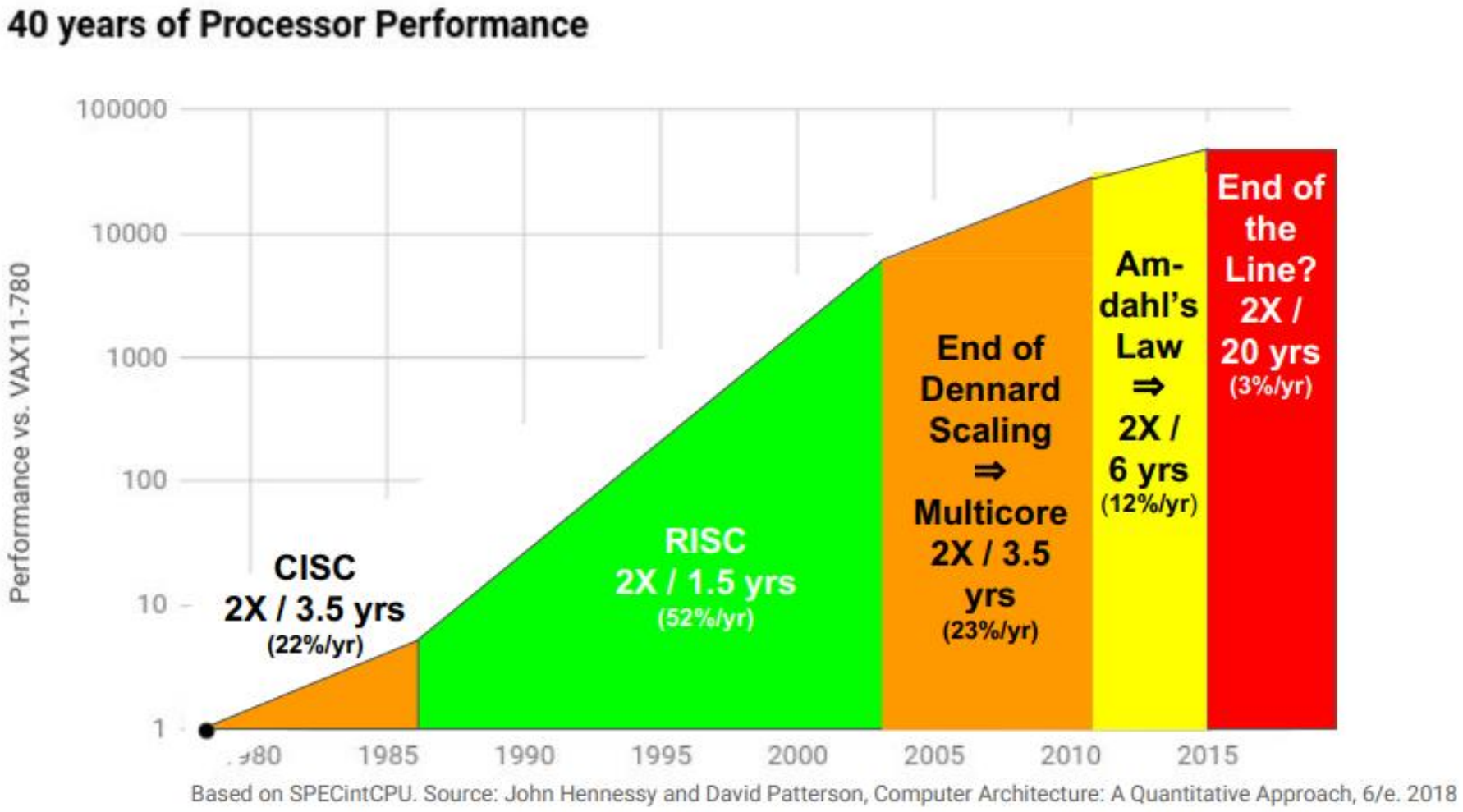
- **给定负载情况下能耗越少，能效越高，特别是对电池供电的移动设备。**
- **功耗应该被看作一个约束条件**
  - A chip might be limited to 120 watts (cooling + power supply)
- **Power Consumed = Dynamic Power + Static Power**
  - 晶体管开和关的切换导致的功耗为动态功耗
  - 由于晶体管静态漏电流导致的功耗称为静态功耗
- **通过降低频率可节省功耗**
- **降低电压可降低功耗和能耗**

与能效相关的其他指标

- **EDP (Energy Delay Product)**
  - EDP = Energy * Delay = Power * Delay^2
- **Performance per Power**
  - FLOPS per watt (Scientific computing)
- **SWaP (space, wattage and performance)**
  - Sun Microsystems metric for data centers, incorporating energy and space.
  - SWaP = Performance / (Space * Power)

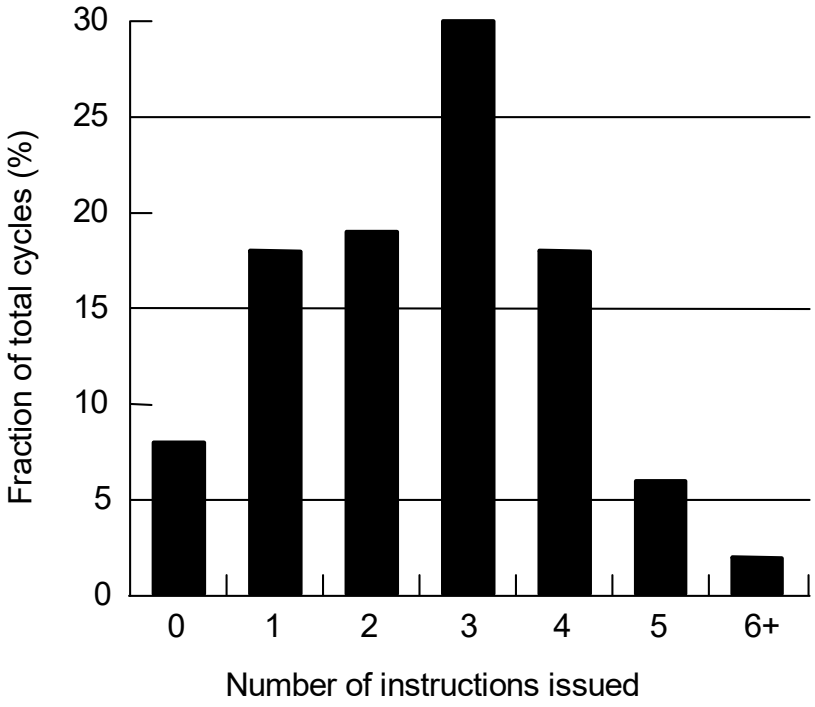
Current and Future Trends

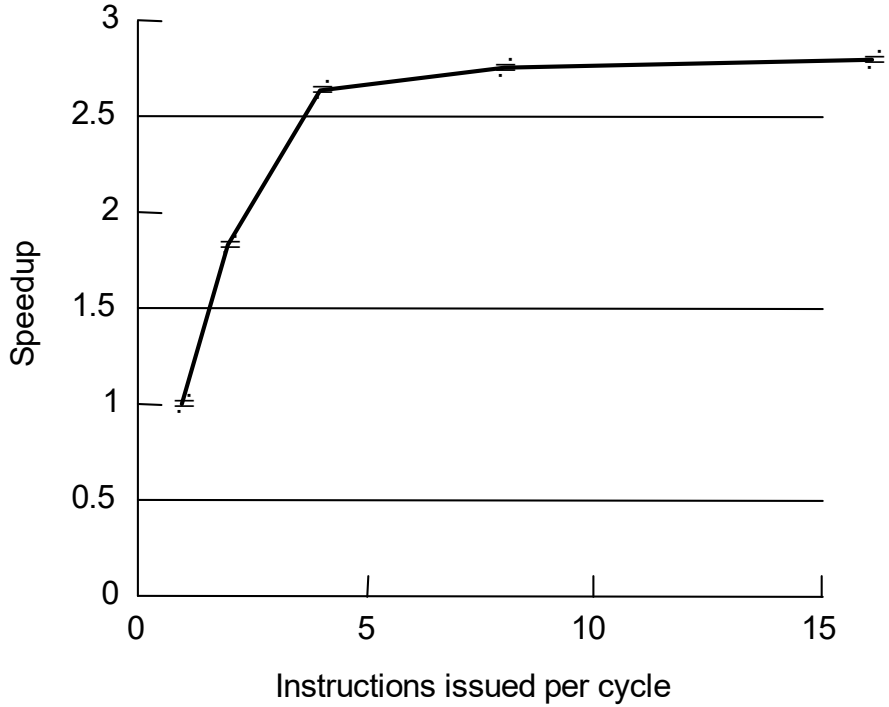
End of Growth of Single Program Speed?



So, What is next?

How far will Instruction Level Parallelism (ILP) go?





理想超标量模型执行下有限的ILP：无限的资源和存取带宽、完善的预取和重命名机制，但是使用现实的Cache。结果：4条指令并行取得了最大的ILP benefit； 5以后则收获甚微

平均来讲，每五条指令有一条control transfer instruction

Why do we have very limited ILP?

有关体系结构的新旧观念

- **Old Conventional Wisdom: Power is free, Transistors expensive**
- **New CW: “Power wall” Power expensive, Transistors free**
- **Old CW: 通过编译、体系结构创新来增加指令级并行 (Out-of-order, speculation, VLIW, …)**
- **New CW: “ILP wall” 挖掘指令级并行的收益越来越小**
- **Old CW: 乘法器速度较慢，访存速度比较快**
- **New CW: “Memory wall” 乘法器速度提升了，访存成为瓶颈 (200 clock cycles to DRAM memory, 4 clocks for multiply)**
- **Old CW: 单处理器性能2X / 1.5 yrs**
- **New CW: Power Wall + ILP Wall + Memory Wall = Brick Wall**
  - 单处理器性能 2X / 5(?) yrs

  ⇒ 芯片设计的巨大变化: multiple “cores” (2X processors per chip / ~ 2 years)
    - 越简单的处理器越节能

计算机安全现状及挑战

- **虽然目前有一些保护措施，但没有很好使用**
  - 域、环等机制
  - 似乎作用不大，并且开销较高
- **过去希望：软件能够消除攻击路径**
  - 程序验证
  - 内核态、用户态
- **Spectre & meltdown漏洞: 利用推断执行机制=>时序攻击**
- **许多微体系架构存在这类漏洞，这些漏洞是体系结构定义的bug**
- **需要CA2.0来避免利用这类漏洞的攻击**
- **硬件必须有助于安全系统的建立**

小结：计算机系统设计方面的巨大变化

- **在过去的50年，Moore’s law和Dennard scaling(登纳德缩放比例定律)主宰着芯片产业的发展**
  - Moore 1965年预测：晶体管数量随着尺寸缩小按接近平方关系增长（每18个月2X）
  - Dennard 1974年预测：晶体管尺寸变小，功耗会同比变小（相同面积下功耗不变）
  - 工艺技术的进步可在不改变软件模型的情况下，持续地提高系统性能/能耗比
- **最近10年间，工艺技术的发展受到了很大制约**
  - Dennard scaling over (supply voltage ~fixed)
  - Moore’s Law (cost/transistor) over?
  - Energy efficiency constrains everything
  - ……
- **功耗问题成为系统结构设计必须考虑的问题**
- **软件设计者必须考虑:**
  - Parallel systems
  - Heterogeneous systems

小结：体系结构挑战性问题

- **性能提升处于停滞状态**
  - Moore定律正在减速
  - 邓纳德缩放定律正在失效
    - 微体系架构技术：指令集并行、多核等技术的低效会消耗能量

现代体系结构发展趋势

- **性能提升的基本手段：并行**
- **应用需求**
  - 计算的需求不断增长，如Scientific computing, video, graphics, databases, deep learning…
- **工艺发展的趋势**
  - 芯片的集成度不断提高，但提升的速度在放缓
  - 时钟频率的提高在放缓，并有降低的趋势
- **体系结构的发展及机遇**
  - 指令集并行受到制约
  - 线程级并行和数据级并行是发展的方向
  - 提高单处理器性能花费的代价呈现上升趋势
  - 面向特定领域的体系结构正蓬勃发展

1.2 定量分析技术基础

- 性能的含义
- CPU性能度量
- 计算机系统性能度量

Defining CPU Performance

- **X比Y性能高的含义是什么?**
- **Ferrari vs. School Bus?**
- **2013 Ferrari 599 GTB**
  - 2 passengers, 11.1 secs for quarter mile
- **2013 Type D school bus**
  - 54 passengers, 36 secs quarter mile time
- **响应时间: e.g., time to travel ¼ mile**
- **吞吐率/带宽: e.g., passenger-mi in 1 hour**

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。

举例

- Time of Concord （协和） vs. Boeing 747?
  - Concord is 1350 mph / 610 mph = 2.2 times faster than Boeing
    = 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!

以时间（Time）度量性能

- **Response Time**
  - 从任务开始到任务完成所经历的时间
  - 通常由最终用户观察和测量
  - 也称Wall-Clock Time or Elapsed Time
  - Response Time = CPU Time + Waiting Time (I/O, scheduling, etc.)
- **CPU Execution Time**
  - 指执行程序（指令序列）所花费的时间
  - 不包括等待I/O或系统调度的开销
  - 可以用秒(msec, μsec, ...) , 或
  - 可以用相对值（CPU的时钟周期数 (clock cycles)）

以吞吐率度量性能

- **Throughput = 单位时间完成的任务数**
  - 任务数/小时、事务数/分钟、100Mbits/s
- **缩短任务执行时间可提高吞吐率（throughput)**
  - Example: 使用更快的处理器
  - 执行单个任务时间少 ⇒ 单位时间所完成的任务数增加
- **硬件并行可提高吞吐率和响应时间(response time)**
  - Example: 采用多处理器结构
  - 多个任务可并行执行, 单个任务的执行时间没有减少
  - 减少等待时间可缩短响应时间

相对性能

- **某程序运行在X系统上**

$$performanc\ e(x) = \frac{1}{execution\ \_\ time(x)}$$

- **X 性能是Y的n倍” 是指**

- $$n = \frac{Performanc\ e(x)}{Performanc\ e(y)}$$

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

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

CPU 性能公式-CPI

$$\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} = \frac{\text{Seconds}}{\text{Program}} = \text{CPU time}$$

$$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time}$$

不同类型的指令具有不同的CPI

Let $\mathrm{CPI_i}$ = clocks per instruction for class i of instructions
Let $\mathrm{IC_i}$ = instruction count for class i of instructions

$$\text{CPU cycles} = \sum_{i=1}^{n}(CPI_i \times IC_i)$$

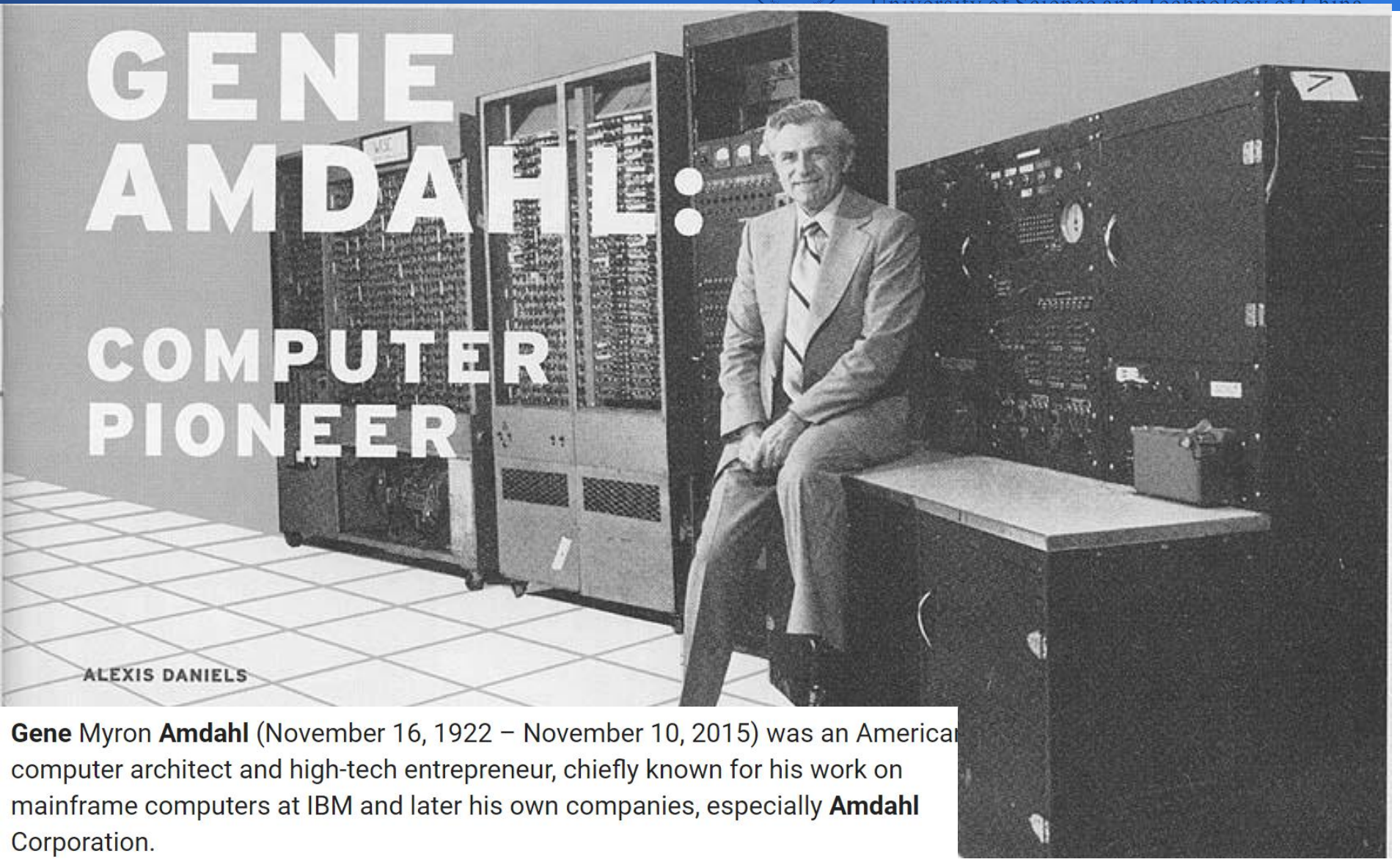
$$CPI = \frac{\sum_{i=1}^{n}(CPI_i \times IC_i)}{\sum_{i=1}^{n} IC_i} \qquad Feq_i = \frac{IC_i}{\sum_{i=1}^{n} IC_i}$$

$$\text{CPI} = \sum_{i=1}^{n}(CPI_i \times Feq_i)$$

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	

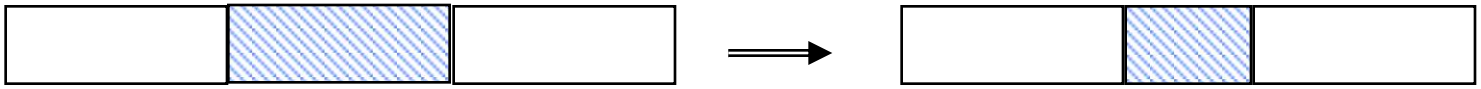


Gene Myron **Amdahl** (November 16, 1922 – November 10, 2015) was an American computer architect and high-tech entrepreneur, chiefly known for his work on mainframe computers at IBM and later his own companies, especially **Amdahl** Corporation.

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

- **ExTime(with E) = ((1-F) + F/S) X ExTime(without E)**
- **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)

Current processor has 1 core; next generation has n cores. Can we achieve a speedup of n?

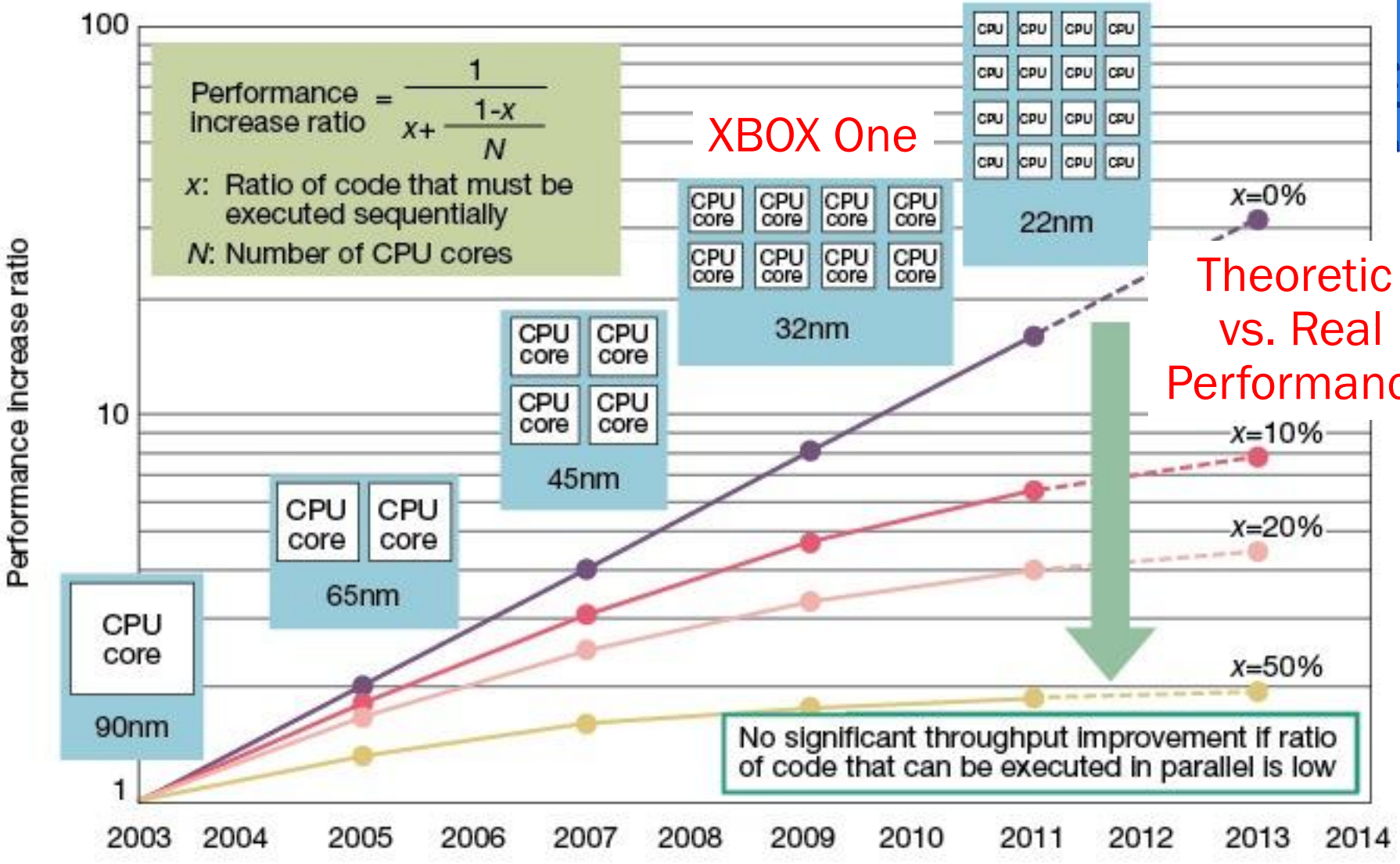


Fig 3 Amdahl's Law an Obstacle to Improved Performance Performance will not rise in the same proportion as the increase in CPU cores. Performance gains are limited by the ratio of software processing that must be executed sequentially. Amdahl's Law is a major obstacle in boosting multicore microprocessor performance. Diagram assumes no overhead in parallel processing. Years shown for design rules based on Intel planned and actual technology. Core count assumed to double for each rule generation.

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60%的乘法操作，问整个程序的加速比又是多少?

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

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？（1.11）1/(0.9+0.1/50)
- 如果有60%的乘法操作，问整个程序的加速比又是多少？（2.43）1/(0.4+0.6/50)

举例

- **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？[s=1/(.1+.9/10)]

    10+9=19
  - 新的系统相对于原来的系统加速比是多少？5.26
  - 在新的系统中，原来特定的计算占整个计算的比例是多少？9/19

Amdahl定律的假设

- **Amdahl 定律的假设**
  - 问题规模是常量
  - 研究随着处理器数目的增加性能的变化

- **有别于Amdahl定律的另一视角**
  - 我们通常用更快的计算机解决更大规模的问题
  - 将处理时间视为常量，研究随着处理器数目的增加问题规模的增加情况（可扩放性）
  - Gustafson–Barsis's Law （古斯塔夫森定律）

基本评估方法 - 市场评估方法

- **MIPS：每秒百万条指令数 【millions of instructions per second】**
  - MIPS = IC/(execution time x 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（Standard Performance Evaluation Corporation）**

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，但是很多情况下不可行。

(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。一般是小几千行。

(4) 小测试程序（toy programs）：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**
  - SPEC2017
  - 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, ...)
- **.....**

Standard Performance Evaluation Corporation (www.spec.org)

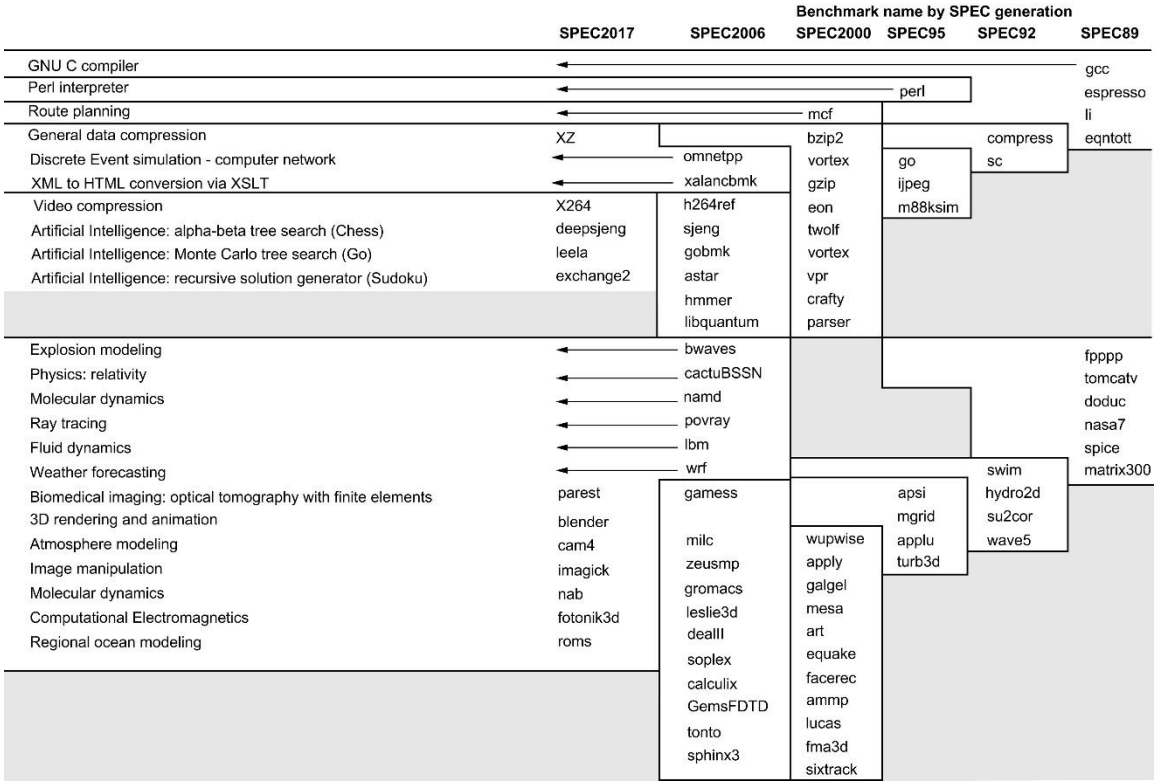


Figure 1.17 SPEC2017 programs and the evolution of the SPEC benchmarks over time, with integer programs above the line and floating-point programs below the line. Of the 10 SPEC2017 integer programs, 5 are written in C, 4 in C++., and 1 in Fortran. For the floating-point programs, the split is 3 in Fortran, 2 in C++, 2 in C, and 6 in mixed C, C++, and Fortran. The figure shows all 82 of the programs in the 1989, 1992, 1995, 2000, 2006, and 2017 releases. Gcc is the senior citizen of the group. Only 3 integer programs and 3 floating-point programs survived three or more generations. Although a few are carried over from generation to generation, the version of the program changes and either the input or the size of the benchmark is often expanded to increase its running time and to avoid perturbation in measurement or domination of the execution time by some factor other than CPU time. The benchmark descriptions on the left are for SPEC2017 only and do not apply to earlier versions. Programs in the same row from different generations of SPEC are generally not related; for example, fpppp is not a CFD code like bwaves.

Category	Name	Measures performance of
Cloud	Cloud_IaaS 2016	Cloud using NoSQL database transaction and K-Means clustering using map/reduce
CPU	CPU2017	Compute-intensive integer and floating-point workloads
Graphics and workstation performance	SPECviewperf® 12	3D graphics in systems running OpenGL and Direct X
	SPECwpc V2.0	Workstations running professional apps under the Windows OS
	SPECapcSM for 3ds Max 2015™	3D graphics running the proprietary Autodesk 3ds Max 2015 app
	SPECapcSM for Maya® 2012	3D graphics running the proprietary Autodesk 3ds Max 2012 app
	SPECapcSM for PTC Creo 3.0	3D graphics running the proprietary PTC Creo 3.0 app
	SPECapcSM for Siemens NX 9.0 and 10.0	3D graphics running the proprietary Siemens NX 9.0 or 10.0 app
	SPECapcSM for SolidWorks 2015	3D graphics of systems running the proprietary SolidWorks 2015 CAD/CAM app
High performance computing	ACCEL	Accelerator and host CPU running parallel applications using OpenCL and OpenACC
	MPI2007	MPI-parallel, floating-point, compute-intensive programs running on clusters and SMPs
	OMP2012	Parallel apps running OpenMP
Java client/server	SPECjbb2015	Java servers
Power	SPECpower_ssj2008	Power of volume server class computers running SPECjbb2015
Solution File Server (SFS)	SFS2014	File server throughput and response time
	SPECsfs2008	File servers utilizing the NFSv3 and CIFS protocols
Virtualization	SPECvirt_sc2013	Datacenter servers used in virtualized server consolidation

Figure 1.18 Active benchmarks from SPEC as of 2017.

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**

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

  - SPEC采用这种方法(SPECRatio)

SPEC 性能综合

$$\text{SPEC Ratio} = \frac{\text{Time on Reference Computer}}{\text{Time on Computer Being Rated}}$$

$$\frac{\text{SPEC Ratio}_A}{\text{SPEC Ratio}_B} = \frac{\dfrac{\text{ExecutionTime}_{Ref}}{\text{ExecutionTime}_A}}{\dfrac{\text{ExecutionTime}_{Ref}}{\text{ExecutionTime}_B}} = \frac{\text{ExecutionTime}_B}{\text{ExecutionTime}_A} = \frac{\text{Performance}_A}{\text{Performance}_B}$$

$$\text{Geometric Mean of SPEC Ratios} = \sqrt[n]{\prod_{i=1}^{n} \text{SPEC Ratio}_i}$$

SPECfp2000 Execution Times & SPEC Ratios

Benchmark	Ultra 5 Time (sec)	Opteron Time (sec)	SpecRatio Opteron	Itanium2 Time (sec)	SpecRatio Itanium2	Opteron/ Itanium2 Times	Itanium2/ Opteron SpecRatios
wupwise	1600	51.5	31.06	56.1	28.53	0.92	0.92
swim	3100	125.0	24.73	70.7	43.85	1.77	1.77
mgrid	1800	98.0	18.37	65.8	27.36	1.49	1.49
applu	2100	94.0	22.34	50.9	41.25	1.85	1.85
mesa	1400	64.6	21.69	108.0	12.99	0.60	0.60
galgel	2900	86.4	33.57	40.0	72.47	2.16	2.16
art	2600	92.4	28.13	21.0	123.67	4.40	4.40
equake	1300	72.6	17.92	36.3	35.78	2.00	2.00
facerec	1900	73.6	25.80	86.9	21.86	0.85	0.85
ammp	2200	136.0	16.14	132.0	16.63	1.03	1.03
lucas	2000	88.8	22.52	107.0	18.76	0.83	0.83
fma3d	2100	120.0	17.48	131.0	16.09	0.92	0.92
sixtrack	1100	123.0	8.95	68.8	15.99	1.79	1.79
apsi	2600	150.0	17.36	231.0	11.27	0.65	0.65
Geometric Mean			20.86		27.12	1.30	1.30

Geometric mean of ratios = 1.30 = Ratio of Geometric means = 27.12 / 20.86

几何平均的两个重要特性

$$\frac{\text{Geometric mean}_A}{\text{Geometric mean}_B} = \frac{\sqrt[n]{\prod_{i=1}^{n} \text{SPECRatio A}_i}}{\sqrt[n]{\prod_{i=1}^{n} \text{SPECRatio B}_i}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{SPECRatio A}_i}{\text{SPECRatio B}_i}}$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{\dfrac{\text{Execution time}_{\text{reference}_i}}{\text{Execution time}_{A_i}}}{\dfrac{\text{Execution time}_{\text{reference}_i}}{\text{Execution time}_{B_i}}}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Execution time}_{B_i}}{\text{Execution time}_{A_i}}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Performance}_{A_i}}{\text{Performance}_{B_i}}}$$

- 几何平均的比率等于比率的几何平均
- 几何平均的比率 等于 性能比率的几何平均
  - 与参考机器的选择无关

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

小结：定量分析基础

- **性能度量**
  - 响应时间 (response time)
  - 吞吐率 (Throughput)
- **CPU 执行时间 = IC × CPI × T**
  - CPI（Cycles per Instruction)
- **MIPS = Millions of Instructions Per Second**
- **Latency versus Bandwidth**
  - Latency指单个任务的执行时间，Bandwidth 指单位时间完成的任务量（rate）
  - Latency 的提升滞后于带宽的提升 (在过去的30年)
- **Amdahl’s Law 用来度量加速比（speedup)**
  - 性能提升受限于任务中可加速部分所占的比例
- **Benchmarks：指一组用于测试的程序**
  - 比较计算机系统的性能
  - SPEC benchmark：针对一组应用综合性能值采用SPEC ratios 的几何平均

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**

Acknowledgements

- **These slides contain material developed and copyright by:**
  - John Kubiatowicz (UCB)
  - Krste Asanovic (UCB)
  - John Hennessy (Standford)and David Patterson (UCB)
  - Chenxi Zhang (Tongji)
  - Muhamed Mudawar (KFUPM)
- **UCB material derived from course CS152、CS252、CS61C**
- **KFUPM material derived from course COE501、COE502**