University of Science and Technology of China

计算机体系结构

周学海
xhzhou@ustc.edu.cn
0551-63492149
中国科学技术大学

review-历史

- **体系结构发展中的重要事件**
  - IBM360系列机、微程序控制器、微处理器、RISC、VLIW、EPIC
- **软件兼容性问题 →ISA**
- **微程序技术→ISA的实现和扩展**
- **集成电路技术→**
  - CPU的功能和性能不断提高
  - 微处理器大战→CISC
- **复杂指令集ISA的合理性问题分析→RISC**
- **指令级并行技术→VLIW**
- **VLIW的二进制兼容问题→EPIC**

- **Great ideas in Computer Architecture**

review-现状

- **计算机系统-产品形态**
  - **个人移动设备 (PMD)**
    - 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
  - **集群/仓储级计算机**（Clusters / Warehouse Scale Computers）
    - Emphasis on availability and price-performance
  - 嵌入式计算机（Embedded Computers）
    - Emphasis: price
- **体系结构设计面临的新问题**
  - Power Wall + ILP Wall + Memory Wall = Brick Wall

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：晶体管数量随着特征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

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

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, …
- **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

体系结构的发展机遇--DSA

- **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
  - RISC Five 第五代Berkeley RISC
- **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
    - ……
  - 并不仅针对一个专门的应用(ASIC), 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
- **Agile Hardware Development**

Chapter1 量化设计与分析基础

- **1.1 引言**
  - 计算机体系结构的定义
- **1.2 体系结构发展历史、现状及趋势**
  - 现代计算机系统发展趋势
- **1.3 定量分析基础**

1.3 定量分析基础

性能的含义

CPU

CPU
性能度量

计算机系统
性能度量

1、性能的含义是什么？
2、如何度量CPU的性能？
3、如何度量和比较计算机系统的性能？

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 in quarter mile
- **2013 Type D school bus**
  - 54 passengers, quarter mile time?

- **响应时间(Speed): e.g., time to travel ¼ mile**
- **吞吐率/带宽(Throughput): e.g., passenger-mi in 1 hour**
- **其他指标（System）：**
  - Availability, scalability, performance per Watt

性能的含义

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, response time, latency
- **Tasks per day, hour, week, sec, ns. ..**
  - throughput, bandwidth

这两者经常会有冲突的。

举例：Concorde vs. Boeing 747

- **Time of Concorde 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”

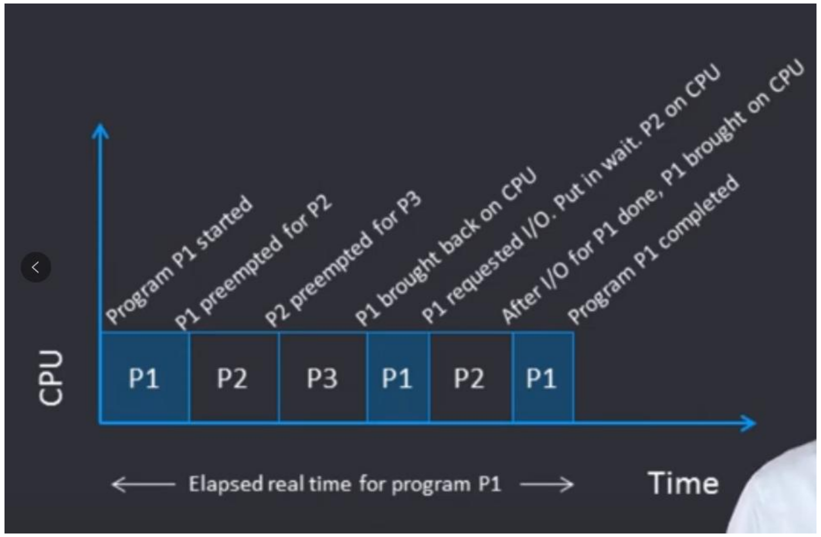
- **Boeing is 1.6 times (“60%”) faster in terms of throughput**
- **Concorde is 2.2 times (“120%”) faster in terms of flying time**

- **评估程序的性能时：任务的粒度不同，使用的术语会所差异**
  - **以整个程序为任务**→程序的执行时间 （execution time）
  - 程序由一组指令构成，**以指令为任务**→指令的吞吐率

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)）



Multiprogrammed CPU works on other process when current rpcess stalled for I/O

quiz

- **下列描述哪个有错误**
  - wall-clock time 指任务总的执行时间，包括IO、操作系统开销等
  - 多线程处理可提高任务的吞吐率
  - CPU time 不包括IO时间
  - 多线程处理可提高单个线程的执行时间

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

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

1.3 定量分析基础

性能的含义

CPU

CPU性能度量

计算机系统性能度量

CPU性能度量

- **Response time (elapsed time): 完成一个任务所需要的所有时间**
  - 例如：unix 中的time命令
  - 90.7s 12.9s 2:39 65% (90.7/159)

  - User CPU Time (90.7)
  - System CPU Time (12.9)
  - Elapsed Time (2:39)

CPU 性能公式-CPI

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

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

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

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

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

- **注意：Instruction count 指的是（动态）执行的指令条数**

不同类型的指令具有不同的CPI

假设： ***CPI_i*** = 第i类指令执行所需的时钟周期数

IC_i = 所执行的第i类指令条数

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

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

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

Improving Performance

- CPU Time = $IC \times CPI \times T = IC \times CPI / f$
- CPU性能公式体现了如何提高性能
- 不幸的事情：三个参数之间存在相关性-trade-off

Reduce	How	Side effect
IC	增强指令功能/复杂性	CPI，T(1/f)会增加
CPI	使指令功能简单，完成指令所需的cycles减少	IC增加
T	每个cycle完成的动作减少	CPI 增加

影响程序执行性能的主要因素

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

CPI计算举例

Base Machine (Reg / Reg)

Op	Freq	CPI_i	CPI_i*F_i	(% Time)
ALU	50%	1	0.5	(33%)
Load	20%	2	0.4	(27%)
Store	10%	2	0.2	(13%)
Branch	20%	2	0.4	(27%)
			1.5	

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

举例

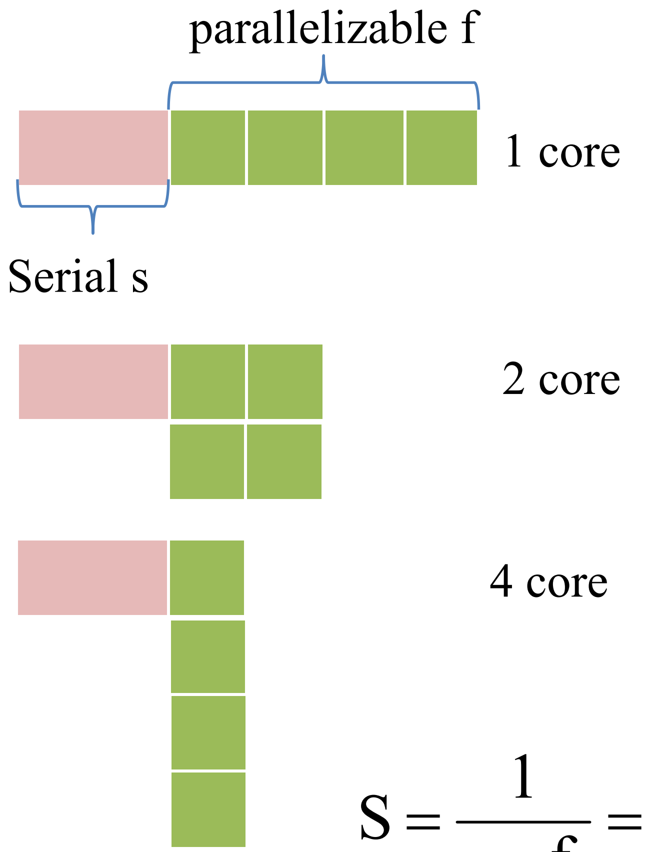
- **假设给定一体系结构硬件不支持乘法运算，乘法需要通过软件来实现。在软件中做一次乘法需要200个周期，而用硬件来实现只要4个时钟周期。如果假设在程序中有10%的乘法操作，问整个程序的加速比？如果有40%的乘法操作，问整个程序的加速比又是多少？**

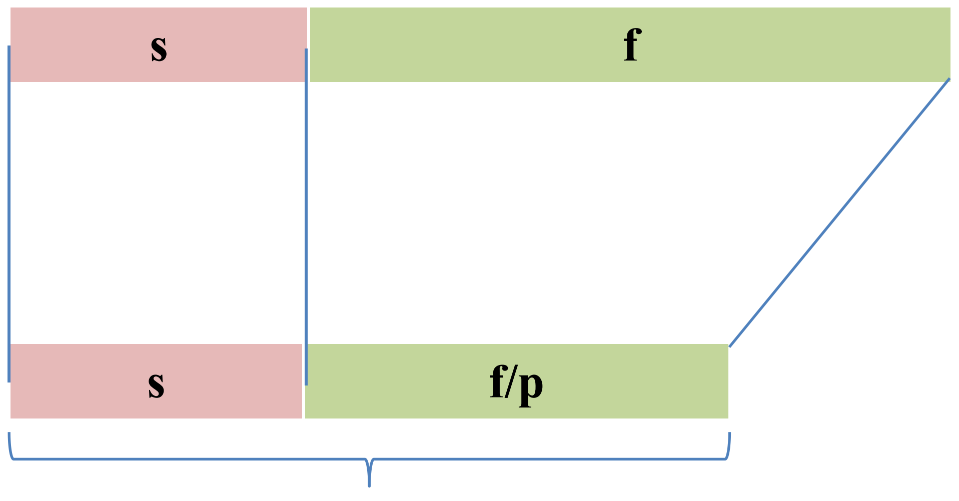
- **假设一计算机在运行给定的一程序时，有90%的时间用于处理某一类特定的计算。现将用于该类计算的部件性能提高到原来的10倍。**
  - 如果该程序在原来的机器上运行需100秒，那么该程序在改进后的机器上运行时间是多少？
  - 新的系统相对于原来的系统加速比是多少？
  - 在新的系统中，原来特定的计算占整个计算的比例是多少？

Test Yourself

- **Measurements:**
  - Frequency of FP ops: 25%
  - Average CPI of FP ops: 4.0
  - Average CPI of other instrs: 1.33
  - Frequency of FPSQR: 2%
  - CPI of FPSQR: 20
- **2 design alternatives:**
  - Decrease CPI of FPSQR to 2.0
  - Decrease average CPI of all FP ops to 2.5
- **Which one is better?**

Amdahl's Law



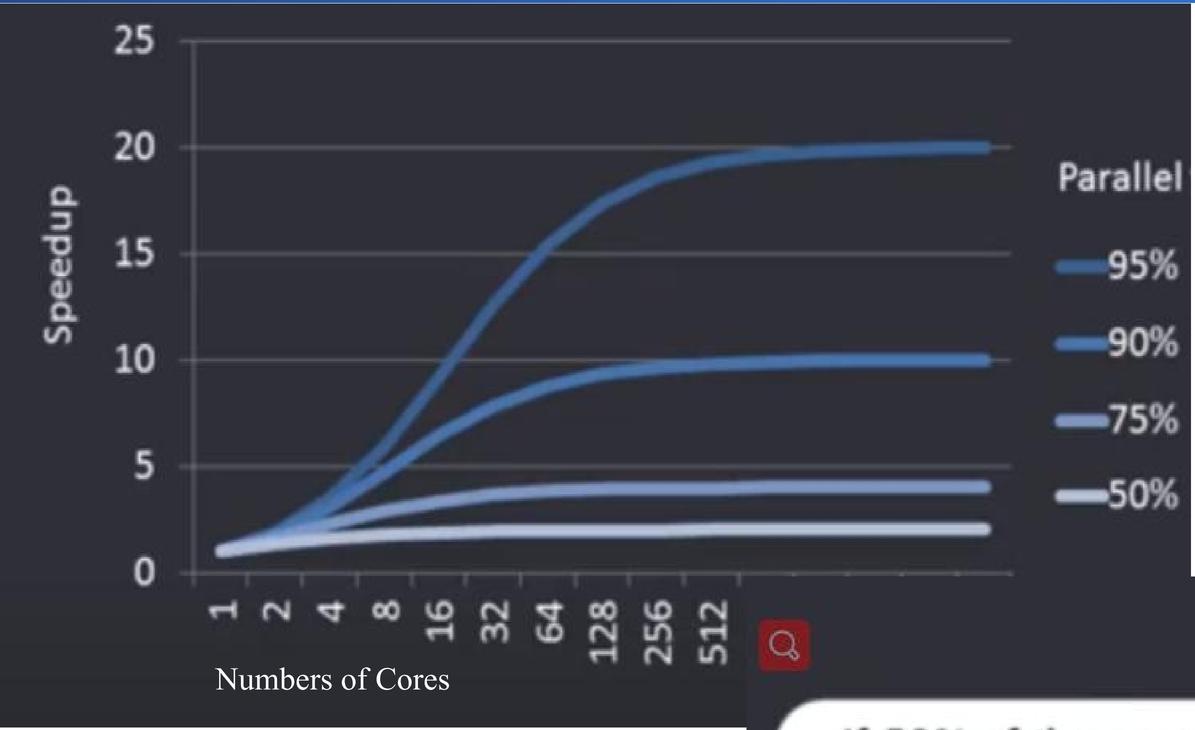


Time = s+f/p

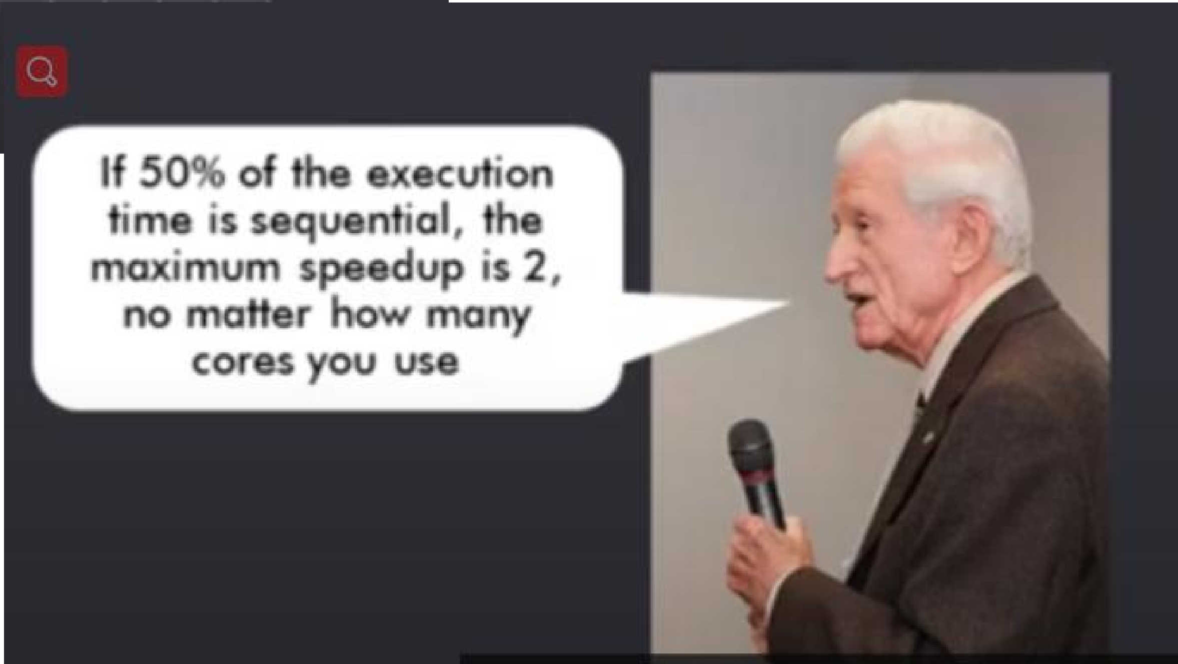
$$S = \frac{1}{s + \frac{f}{p}} = \frac{1}{1 - f + \frac{f}{p}}$$

S = speedup
s = 1-f = serial_fraction
f = 1=s = parallel-fraction
p = # processor_cores

Speedup for different Parallel Fractions

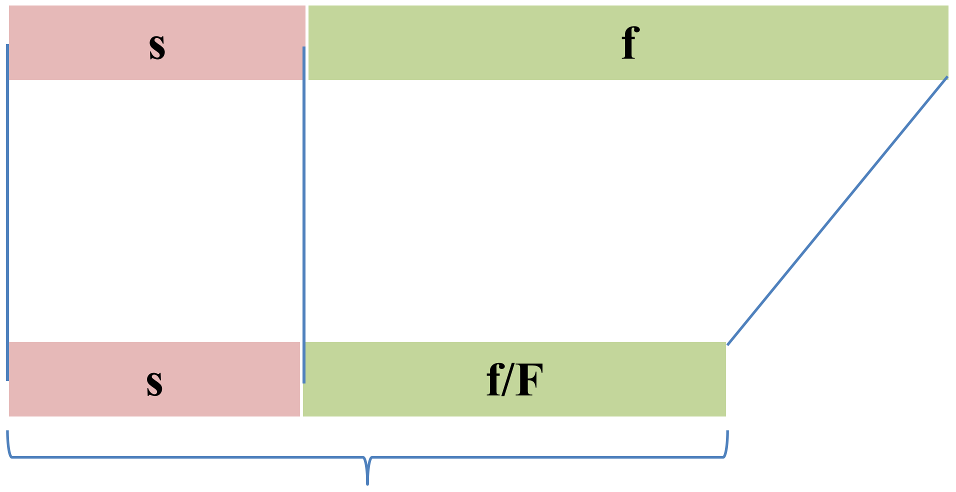


Numbers of Cores


If 50% of the execution time is sequential, the maximum speedup is 2, no matter how many cores you use

Amdahl's Law Generalization

- 假设我们可以改进程序执行的某一部分的比例为f, 这部分改进的加速比为F
- 整个程序的加速比是多少?



Time = s+f/F

$$S = \frac{1}{s + \frac{f}{F}} = \frac{1}{1 - f + \frac{f}{F}}$$

S = speedup
s = 1-f = fraction we cannot improve
f = 1=s = fraction we can improve
F = improvement factor

Amdahl's Law -Examples

- **FP instructions improved to run 2X as fast, but only 10% of all executed instructions are FP**

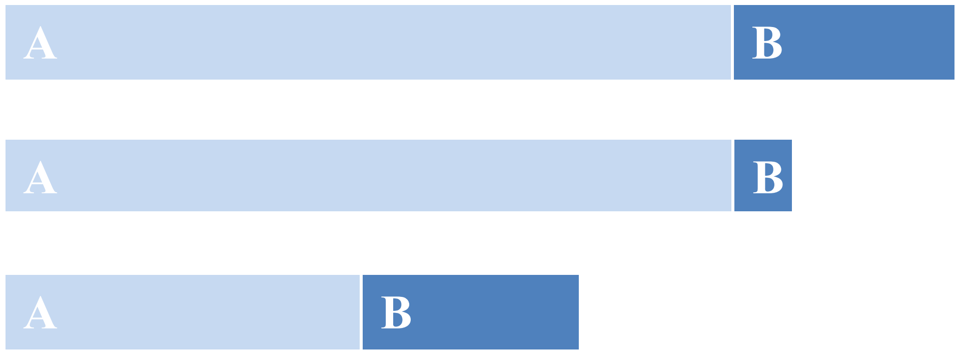
$$S = \frac{1}{1-f+{}^{f}\!/_{F}} = \frac{1}{0.9+{}^{0.1}\!/_{2}} = {}^{1}\!/_{0.95} = 1.053$$

- **We want overall speedup of 2 and can accelerate FP instructons by 4X**
  - What should be the fraction of FP instructions?

$$S = \frac{1}{1-f+{}^{f}\!/_{F}} \rightarrow 2 = \frac{1}{1-f+{}^{f}\!/_{4}} \rightarrow f = {}^{0.5}\!/_{0.75} = 66.7\%$$

Amdahl’s Implications

- **2个相互独立的部分A和B**
  - Make B 5x faster
  - Make A 2x Faster



- **Implications:**
  - Make the common case fast
  - 收益递减效应：优化的收益将越来越小

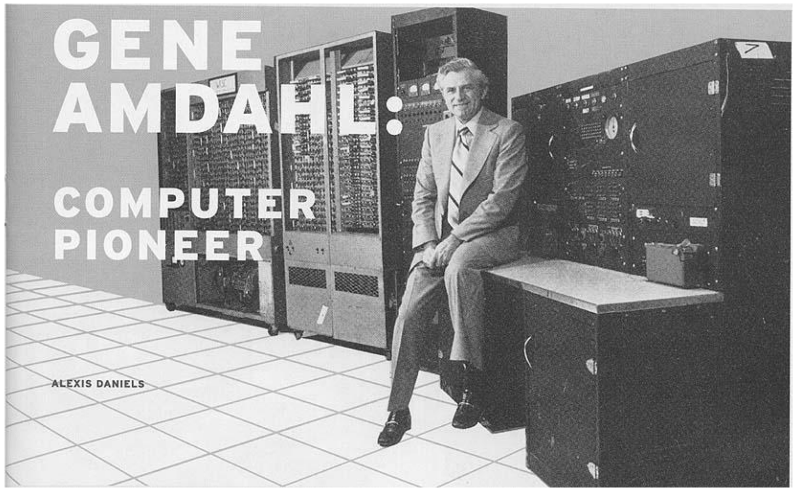
Design Principles to Make Processors Faster

- **充分挖掘并行性**
  - Pipelining，ILP, DLP, TLP
- **充分利用程序的局部性**
  - 程序执行的90%时间是在执行程序中的10%代码
  - 准确预测将要执行的代码和数据有利于性能提升
- **Make the common case fast**
  - Amdahl’s law

并行程序性能评估三大定律

- **假设：**
  - Ws： 串行部分执行时间； W：串行执行总时间； p处理器数目，f：串行执行的时间占比

- **Amdahl定律**
  - 工作负载不变
- **Gustafson–Barsis定律**
  - 执行时间不变
- **Sun and Ni's定律**
  - 考虑并行划分时，存储器访问性能



用Amdahl定律评估多核系统的性能

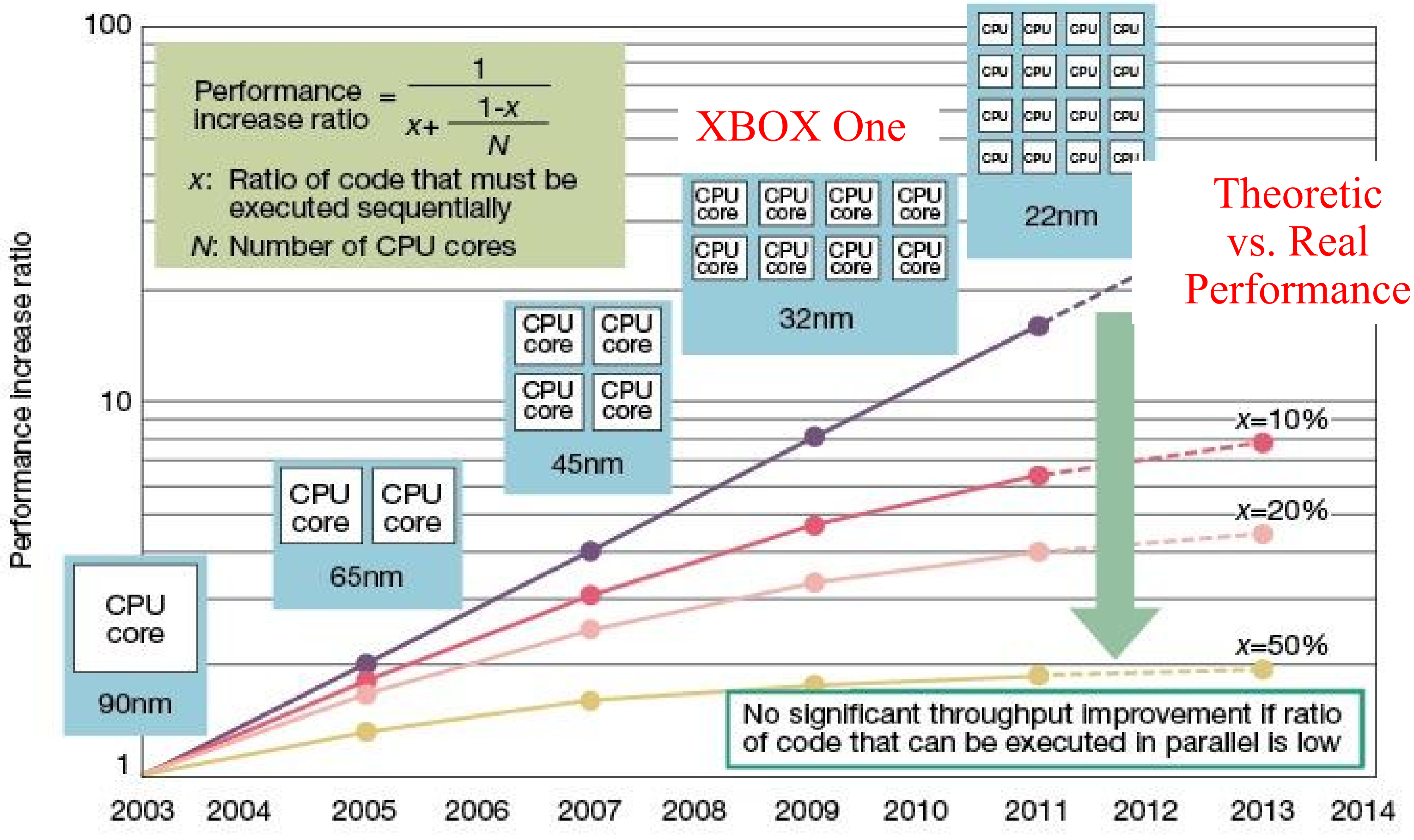
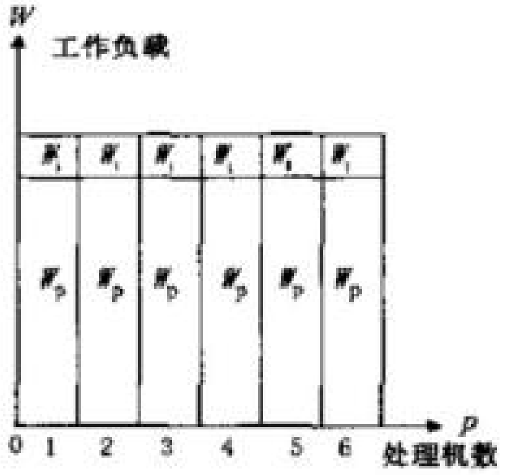


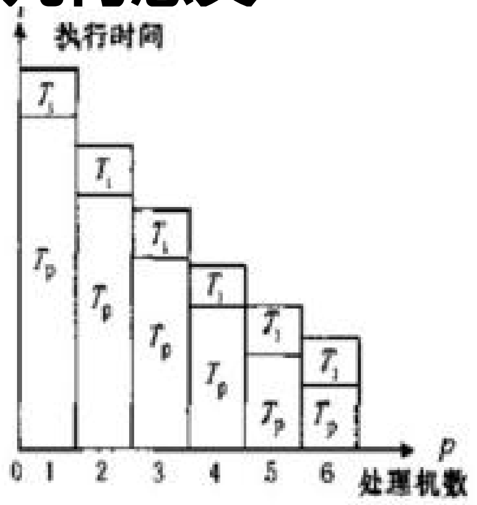
Fig 3 Amdahl's Law an Obstacle to Improved Performance Performance will not rise in the same proportion as the increase in CPU cores. Performance gains are limited by the ratio of software processing that must be executed sequentially. Amdahl's Law is a major obstacle in boosting multicore microprocessor 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.

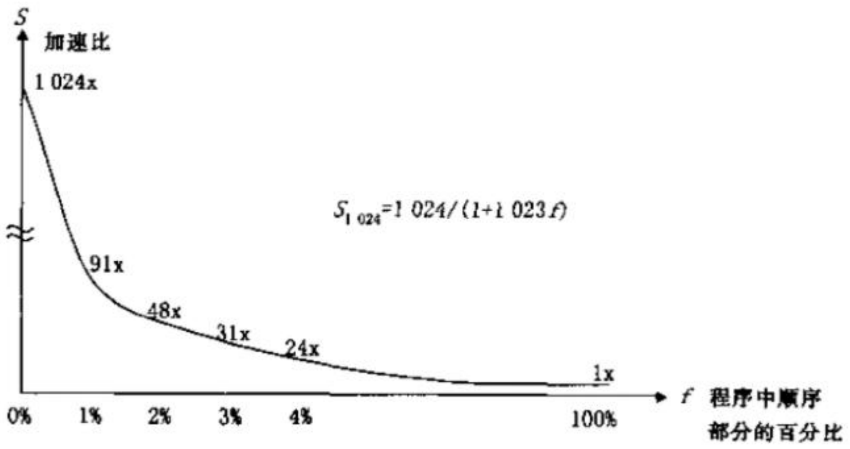
Amdahl定律

- **Amdahl 定律的假设**
  - 串行算法是给定问题的最优解决方案
    - 一些问题本质上是并行的，采用并行实现可大量减少所需计算步骤
  - 处理器核数增加，问题规模保持不变
- **Speedup = (Ws + (1-f)W)/(Ws+ (1-f)W/p)**

- **Amdahl 定律的几何意义**







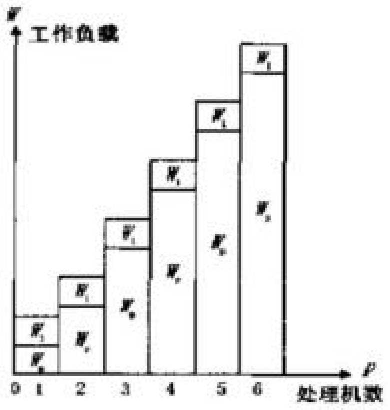
计算负载固定不变

处理器数目增加，执行时间缩短

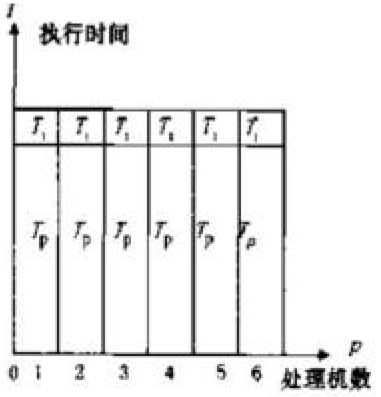
随着程序中串行执行占比提高,加速比下降

古斯塔夫森定律

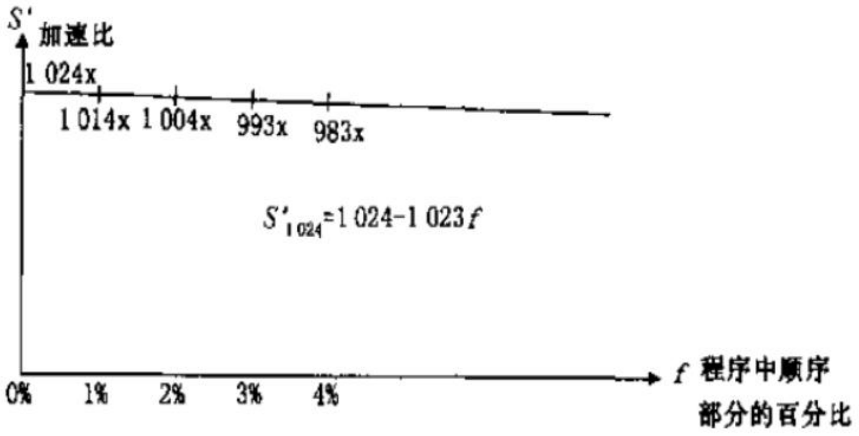
- **有别于Amdahl定律的另一视角：扩展加速比（古斯塔夫森定律）**
  - 我们通常用更快的计算机解决更大规模的问题
  - 将处理时间视为常量，研究处理器数目的增加时，问题规模的增加情况（可扩放性）
    - 当问题规模增大时，程序的串行部分保持不变
    - 当增加处理器的数量时，每个处理器执行相同负载
  - Gustafson–Barsis’ s Law
- **Scaled Speedup = (Ws+(1-f)W×p) / (Ws + (1-f)W) = p-f(p-1)= (1-f)p+f**
  - **在串行比例保持不变的情况下，加速比与处理器数目p 基本成线性关系**
    - p：处理器核数 f：串行执行部分的时间占比



计算负载增加，为保证执行时间不变，需要增加处理器数



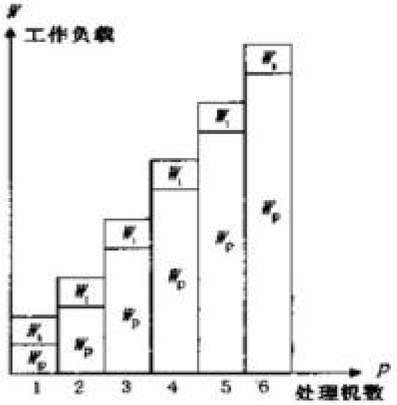
增加处理器数目，保证执行时间不变



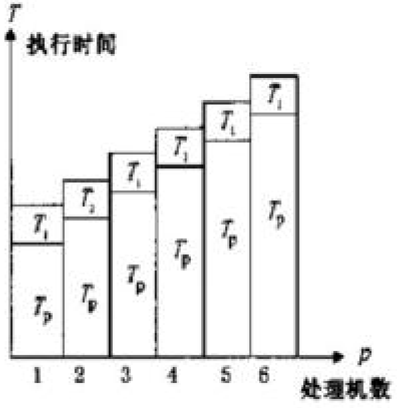
随着程序中串行执行占比提高,加速比下降幅度减弱

Sun and Ni's定理

- **假设:**
  - Ws：串行部分执行时间；W：串行执行总时间；p处理器数目，f：串行执行的时间占比
- **Amdahl定律**
  - Speedup = (Ws + (1-f)W)/(Ws+ (1-f)W/p)
- **Gustafson定律的几何意义**
  - Speedup = (Ws+(1-f)W×p) / (Ws + (1-f)W)
- **Sun and Ni's定律**
  - Amdahl和Gustafson定律的推广：考虑增大问题规模时，存储器访问开销会发生变化。
  - 引入了G(p)表示存储容量增加到p倍时，并行工作负载增加G(p)倍。加速比公式可表示为:
- **Speedup = (Ws+(1-f)G(p)W)/(Ws + (1-f)G(p)W/p)**
  - G(p) = 1：Amdahl 定律，工作负载无增加
  - G(p) = p：Gustafson定律



处理能力随处理器数目的增加C而增加



处理器增加，执行时间也随着增加

1.3 定量分析基础

性能的含义

CPU CPU性能度量

计算机系统性能度量

基本评估方法 - 用什么评估指标?

- **MIPS：每秒执行的百万条指令数**
  - MIPS = IC/(CPI×IC×T×10^6)=1/(CPI×T×10^6)
  - MIPS依赖于指令集，比较两种不同的机器参考价值有限
  - 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，可以比较两种不同的机器。但MFLOPS也有局限性.**
  - MFLOPS = 浮点操作次数/(程序执行时间×10^6)
  - 与机器的浮点运算集有关。CRAY-2没有除法指令，Motorola 68882有
  - 与测试程序有关。测试程序中整数、浮点数操作的比例影响性能
- **SPEC：Standard Performance Evaluation Corporation**

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 - 用什么测试集?

五种类型的测试程序（预测的精度逐级下降）

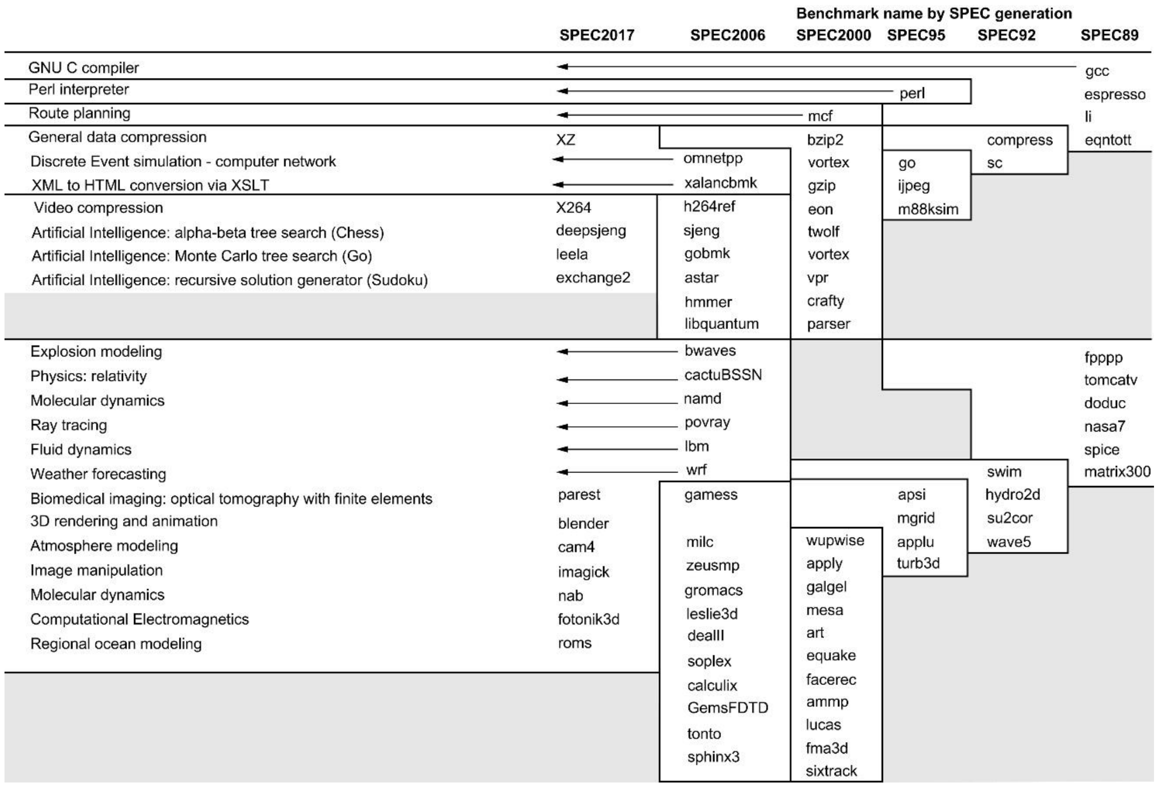
- **真实程序：这是最可靠的方法。**
  - portability problems
  - important activity has to be eliminated
- **修改过的程序：**
  - 增强 portablility
  - 专注于系统某一方面的性能：例如：删除IO操作创建面向CPU性能测试的bennchmarks
  - 使用脚本来产生与程序的交互行为

- **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。**
  - 真实程序中小的，核心的代码片段
  - Livermore Loops及LINPACK是其中使用比较广泛的例子。
  - 主要测量CPU的某一方面的性能
- **小测试程序（toy programs)： 一般在100行以内。**
  - Quicksort， Sieve of Eratosthenes(素数筛选)....
  - 测试程序简单，规模比较小，但用于评估性能的作用有限
- **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。

Benchmark Suites

- **Benchmark suite = collection of benchmarks**
- **最成功最常用的测试集: SPEC （Standard Performance Evaluation Corporation)**

- **起步于CPU性能评估**
- **Desktop Benchmarks**
  - SPEC CPU2006
    - 12 int benchmarks: CINT 2006
    - 17 FP benchmarks: CFP2006
  - SPEC CPU 2017
  - SPEC2000



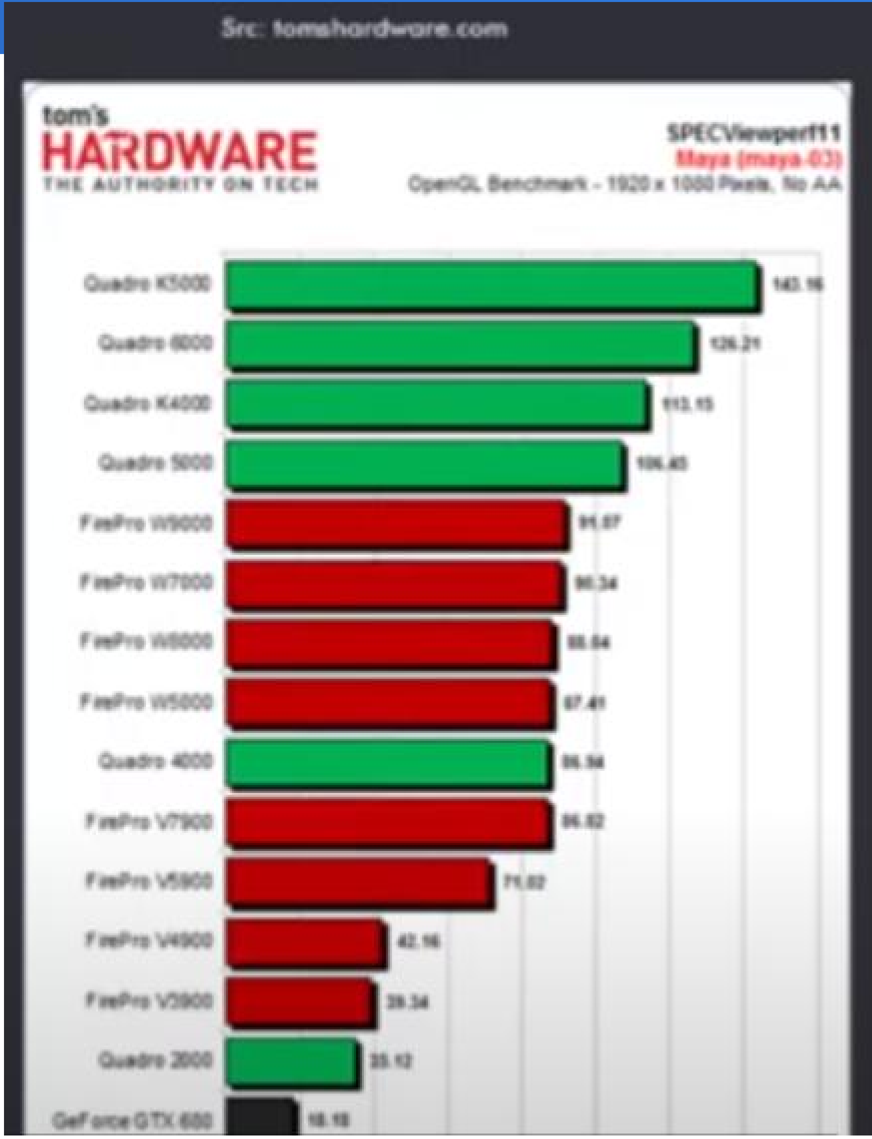
Benchmark name by SPEC generation

	SPEC2017	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
GNU C compiler						gcc
Perl interpreter				perl		espresso
Route planning			mcf			li
General data compression	XZ		bzip2		compress	eqntott
Discrete Event simulation - computer network		omnetpp	vortex	go	sc	
XML to HTML conversion via XSLT		xalancbmk	gzip	ijpeg		
Video compression	X264	h264ref	eon	m88ksim		
Artificial Intelligence: alpha-beta tree search (Chess)	deepsjeng	sjeng	twolf			
Artificial Intelligence: Monte Carlo tree search (Go)	leela	gobmk	vortex			
Artificial Intelligence: recursive solution generator (Sudoku)	exchange2	astar	vpr			
		hmmer	crafty			
		libquantum	parser			
Explosion modeling		bwaves				fpppp
Physics: relativity		cactuBSSN				tomcatv
Molecular dynamics		namd				doduc
Ray tracing		povray				nasa7
Fluid dynamics		lbm				spice
Weather forecasting		wrf			swim	matrix300
Biomedical imaging: optical tomography with finite elements	parest	gamess		apsi	hydro2d	
3D rendering and animation	blender			mgrid	su2cor	
Atmosphere modeling	cam4	milc	wupwise	applu	wave5	
Image manipulation	imagick	zeusmp	apply	turb3d		
Molecular dynamics	nab	gromacs	galgel			
Computational Electromagnetics	fotonik3d	leslie3d	mesa			
Regional ocean modeling	roms	dealII	art			
		soplex	equake			
		calculix	facerec			
		GemsFDTD	ammp			
		tonto	lucas			
		sphinx3	fma3d			
			sixtrack			

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.

Some other SPEC Benchmarks

- **Graphics benchmarks**
  - SPEC viewperf: OpenGL 3D rendering performance
  - SPECapc: apps making extensive use of graphics
- **Server apps have significant I/O activity**
  - SPECSFS: file server benchmark
  - SPECWeb: web server benchmark

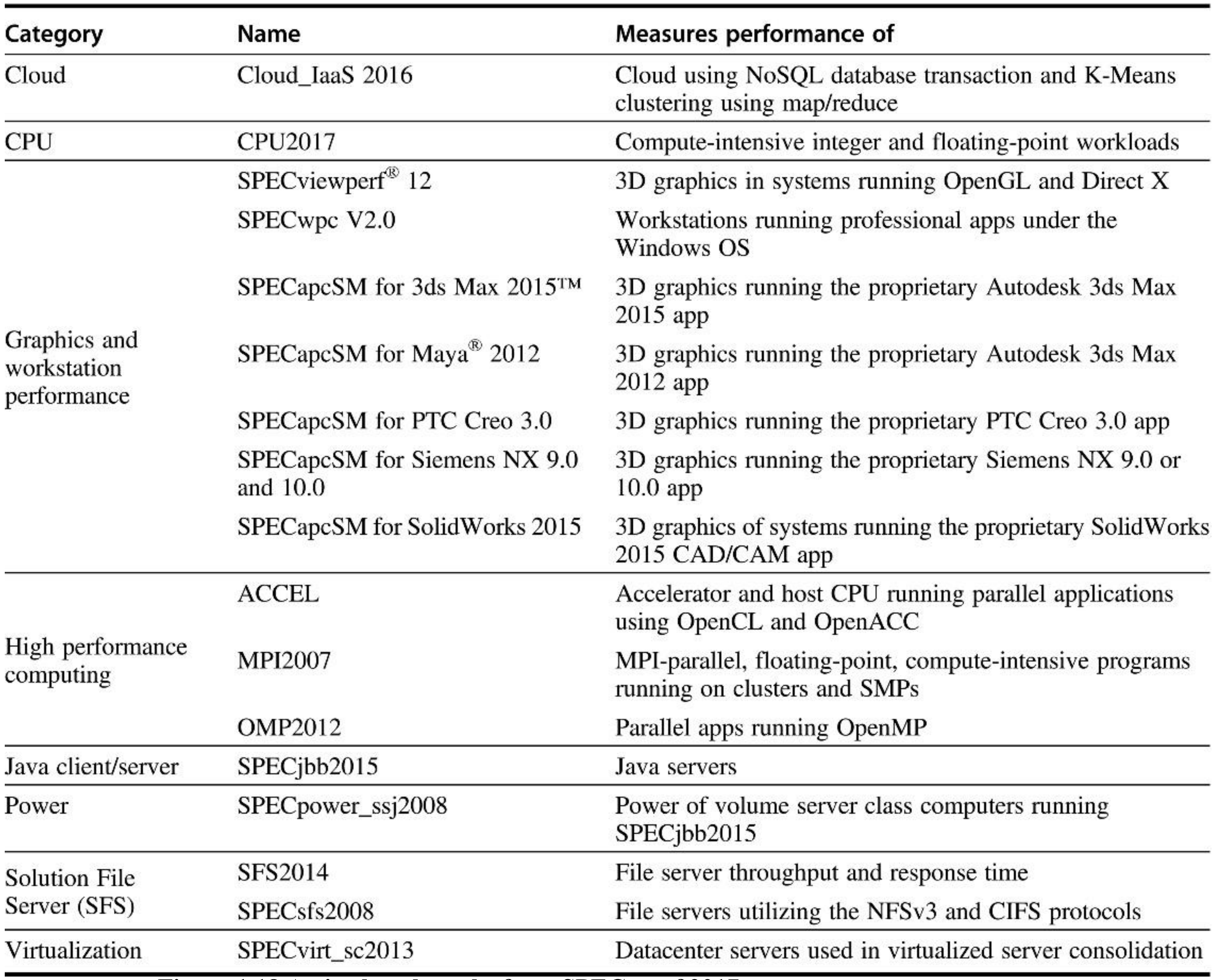


Embedded Benchmarks

- **与desktop、Server 测试集相比 较少**
- **测试程序反应了嵌入式系统应用**
- **例如：Embedded Microprocessor Benchmark Consortium (EEMBC)**

- **EEMBC来源于五类应用**
  – Automotive/industrial
  – Comsumer
  – Networking
  – Office automation
  – Telecommunications

Benchmark suites Summary



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

- **如果仅用一个测试程序来比较机器性能**
  - 例如： Response time
- **当使用多个测试程序来评估CPU的性能**
  - 如何综合评价CPU的性能?
- **例如:**

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

从性能角度出发： CPU A, CPU B, CPU C 选哪个?

性能的综合评价

- **如果测试程序的运行占比不同，如何评价?**
- **得到综合性能的两种方法**
  - 加权的算术平均
    - `SUM(Ti)/n` 或 `SUM (Wi×Ti)/n`
  - 将执行时间归一化到一个参考机器，然后计算平均性能: Execution_time_ratio
    - SPEC采用这种方法(SPECRatio)

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

Src: https://pixabay.com/en/balance-scale-silhouette-justice-147053/

加权算术平均

$$WeightedMean = \sum_{i=1}^{n} Weight_i \times Time_i$$

	CPU A	CPU B	CPU C
Program P1 (secs)	1	10	20
Program P1 (secs)	1000	100	20
Arithmetic mean	500.5	55	20
Weighted mean(W1 = 0.8, W2=0.2)	200.8	28	20
Weighted mean(W1 = 0.9, W2=0.1)	100	19	20

Test Yourself

- 我们需购置一台计算机
- 我们用P1和P2两个测试程序来测试三台机器的性能
- 测试结果如下：

	C1	C2	C3
Program P1 (secs)	1	10	100
Program P1 (secs)	100	100	1

- 什么情况下C1是最好的选择？(P1和P2的运行占比?)
- 什么情况下C1是最好的选择？(w1=?, w2=?)
- 什么情况下C1是最好的选择？(w1=?, w2=?)

归一化执行时间

- **以某一参考机器为基准，归一化执行时间**

$$ExecutionTimeRatio = \frac{ExecutionTime_{reference}}{ExecutionTime_{new}}$$

- **SPEC使用这种方法，称为SPECRatio**
- **注意：**
  - 综合性能不采用算术平均
  - 采用几何平均得到综合性能

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

SPEC Ratio

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

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

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

归一化执行时间后的算术平均举例

- **注意：不适合采用算术平均**

	CPU A	CPU B	CPU C
Program P1 (secs)	1	10	20
Program P2 (secs)	1000	100	20

	Normalized to A			Normalized to B		
	A	B	C	A	B	C
Program P1	1.0	0.1	0.05	10.0	1.0	0.5
Program P2	1.0	10.0	50	0.1	1.0	5.0
Arithmetic mean	1.0	5.05	**25.025**	**5.05**	1.0	2.75

Geometric Mean

- **使用几何平均**
  - 与选用的参考机器无关
  - 统计意义上的综合性能

	Normalized to A			**Normalized to B**		
	A	B	C	A	B	C
Program P1	1.0	0.1	0.05	10.0	1.0	0.5
Program P2	1.0	10.0	50	0.1	1.0	5.0
Arithmetic mean	1.0	5.05	**25.025**	**5.05**	1.0	2.75
Geometric mean	1.0	1.0	**1.58**	**1.0**	1.0	**1.58**

几何平均的两个重要特性

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

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

- 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

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

小结：定量分析基础

- **性能度量**
  - 响应时间 (response time)
  - 吞吐率 (Throughput)
  - 其他指标：可用性，可缩放性，每瓦的性能，.....
- **Computer architectus focus on CPU time**
- **CPU Time = 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定律用来度量加速比（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

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
- **散热设计功耗（Thermal Design Power (TDP)）**
  - 在最糟糕、最坏情况下的功耗。散热解决方案的设计必须满足这种散热设计功耗。
  - 表达了设备的功耗特征，主要用于电源和冷却系统的设计
- **TDP和CPU功耗的关系?**
  - CPU的功耗很大程度上是对主板提出的要求，要求主板能够提供相应的电压和电流;
  - TDP是对散热系统提出要求，要求散热系统能够把CPU发出的热量散掉，即：TDP是要求CPU的散热系统必须能够驱散的最大总热量。
  - TDP值一定比CPU满负荷运行时的发热量大一点。
- **降低频率可导致功耗下降**

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
- **举例: 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
- **答案: Energy consumption of A = 1.2 × 0.7 = 0.84 of B**
  - Processor A consumes less energy than B (more energy-efficient)

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 × Voltage^2**
  - 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 × Voltage^2× 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**

举例：动态能耗和功耗

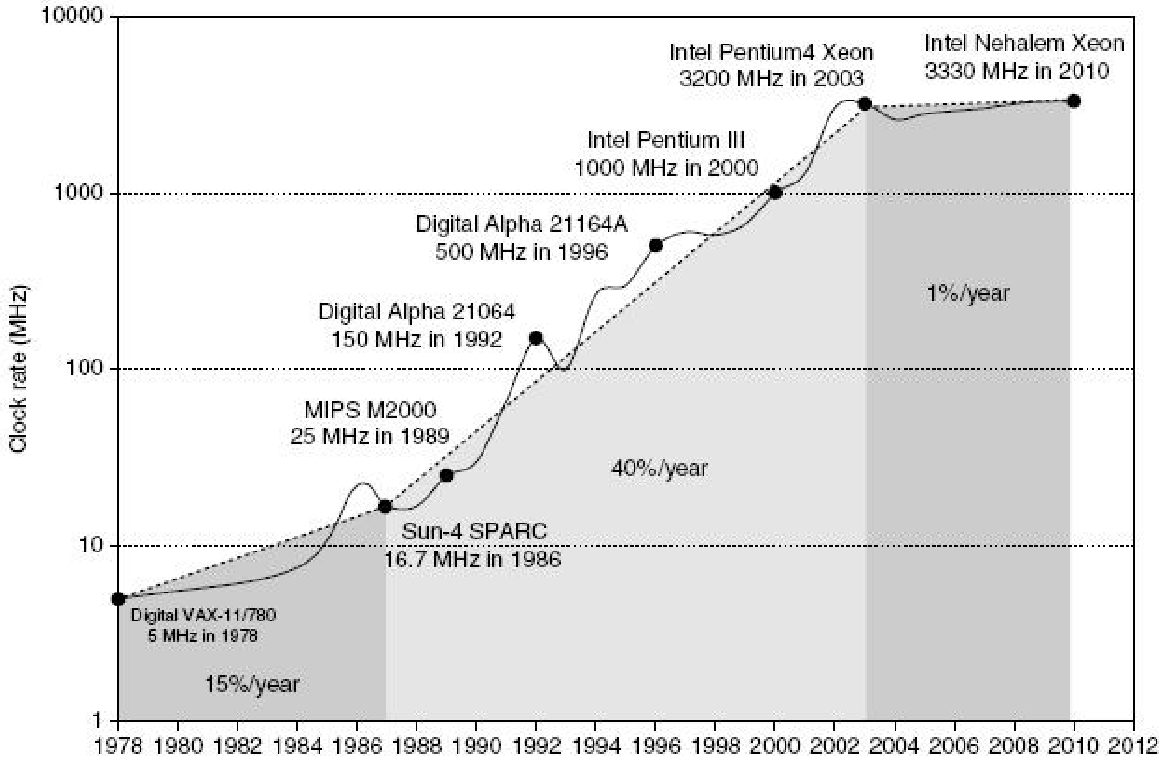
- 今天的一些微处理器具有动态地调整电压和频率的机制（DVFS）。假设电压降低10%，频率降低15%，对动态能耗和动态功耗有什么影响?
- Answer:
  - 10% reduction in Voltage ->$\text{Voltage}_{\text{new}}$= 0.90 $\text{Voltage}_{\text{old}}$
  - 15% reduction in Frequency ->$\text{Frequency}_{\text{new}}$= 0.85 $\text{Frequency}_{\text{old}}$

$$\frac{\text{Energy}_{\text{new}}}{\text{Energy}_{\text{old}}} = \frac{\text{Voltage}_{\text{new}}{}^2}{\text{Voltage}_{\text{old}}{}^2} = (0.90)^2 = 0.81$$

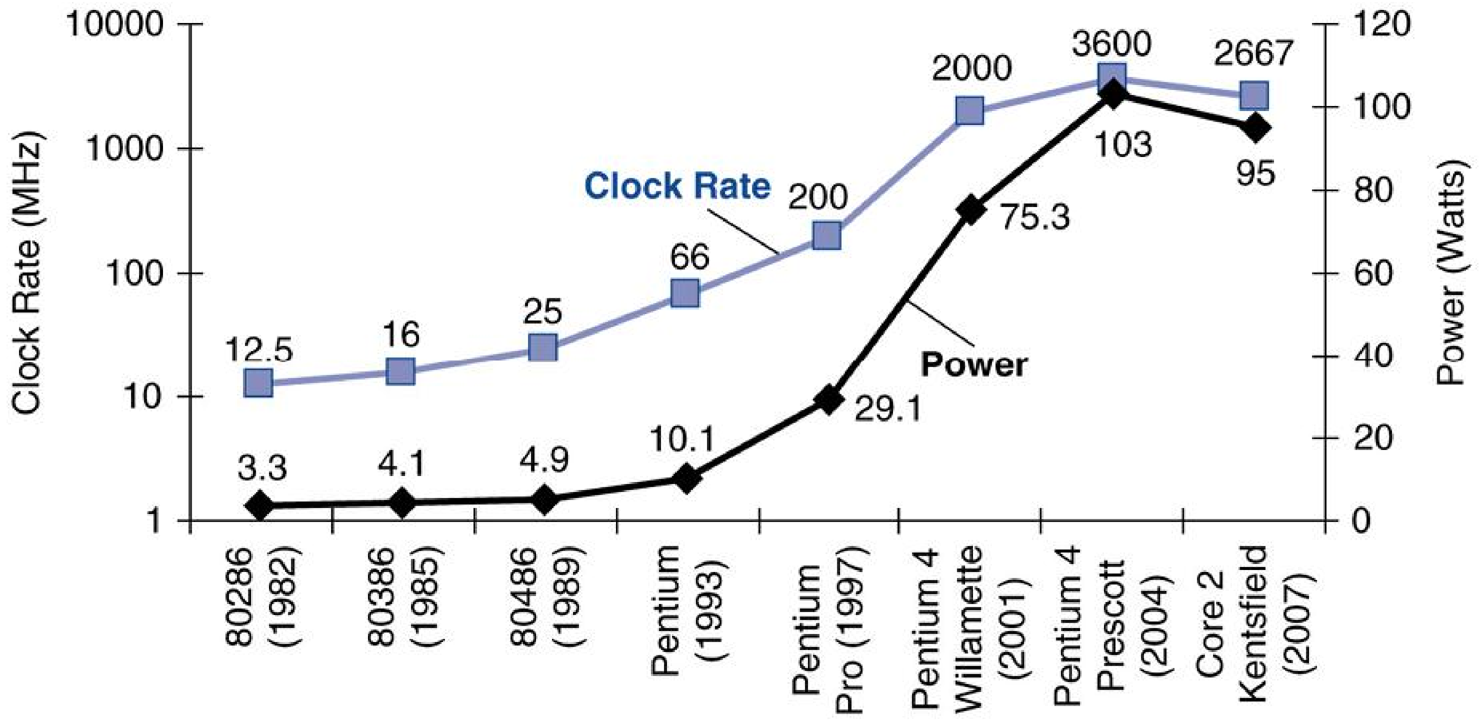
$$\frac{\text{Power}_{\text{new}}}{\text{Power}_{\text{old}}} = 0.81 \times \frac{\text{Frequency}_{\text{new}}}{\text{Frequency}_{\text{old}}} = 0.81 \times 0.85 = 0.6885$$

Trends in Clock frequency

- **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**



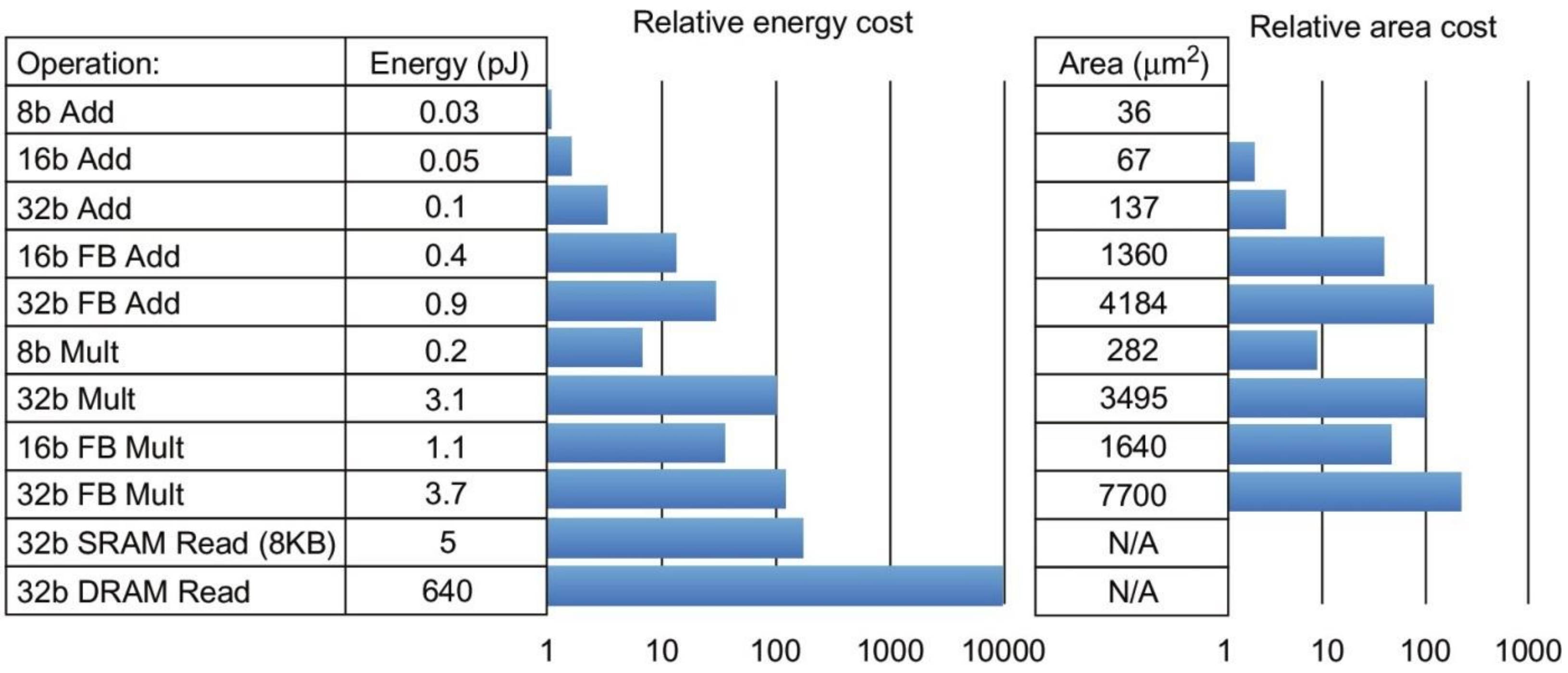
Trends in Power & Clock Frequency



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

$\boxed{30X}$ $\boxed{5V \rightarrow 1V}$ $\boxed{1000X}$

运算与访存部件的能耗及占用面积比较



Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FB Add	0.4	1360
32b FB Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FB Mult	1.1	1640
32b FB Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A

Energy numbers are from Mark Horowitz *Computing's Energy problem (and what we can do about it)*. ISSCC 2014
Area numbers are from synthesized result using Design compiler under TSMC 45nm tech node. FP units used DesignWare Library.

Figure 1.13 Comparison of the energy and die area of arithmetic operations and energy cost of accesses to SRAM and DRAM. [Azizi][Dally]. Area is for TSMC 45 nm technology node.

TIPS：1、存储器访问与运算相比：能耗高2到4个数量级
2、DRAM访问不仅速度慢，而且能耗高
3、SRAM访问与DRAM访问相比，速度快，能耗低2个数量级
4、浮点运算与定点运算相比，能耗高，占用的面积大

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)**
  - 当在较高频率运行安全时，先以较高频率运行一段时间，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

低功耗技术-DVFS

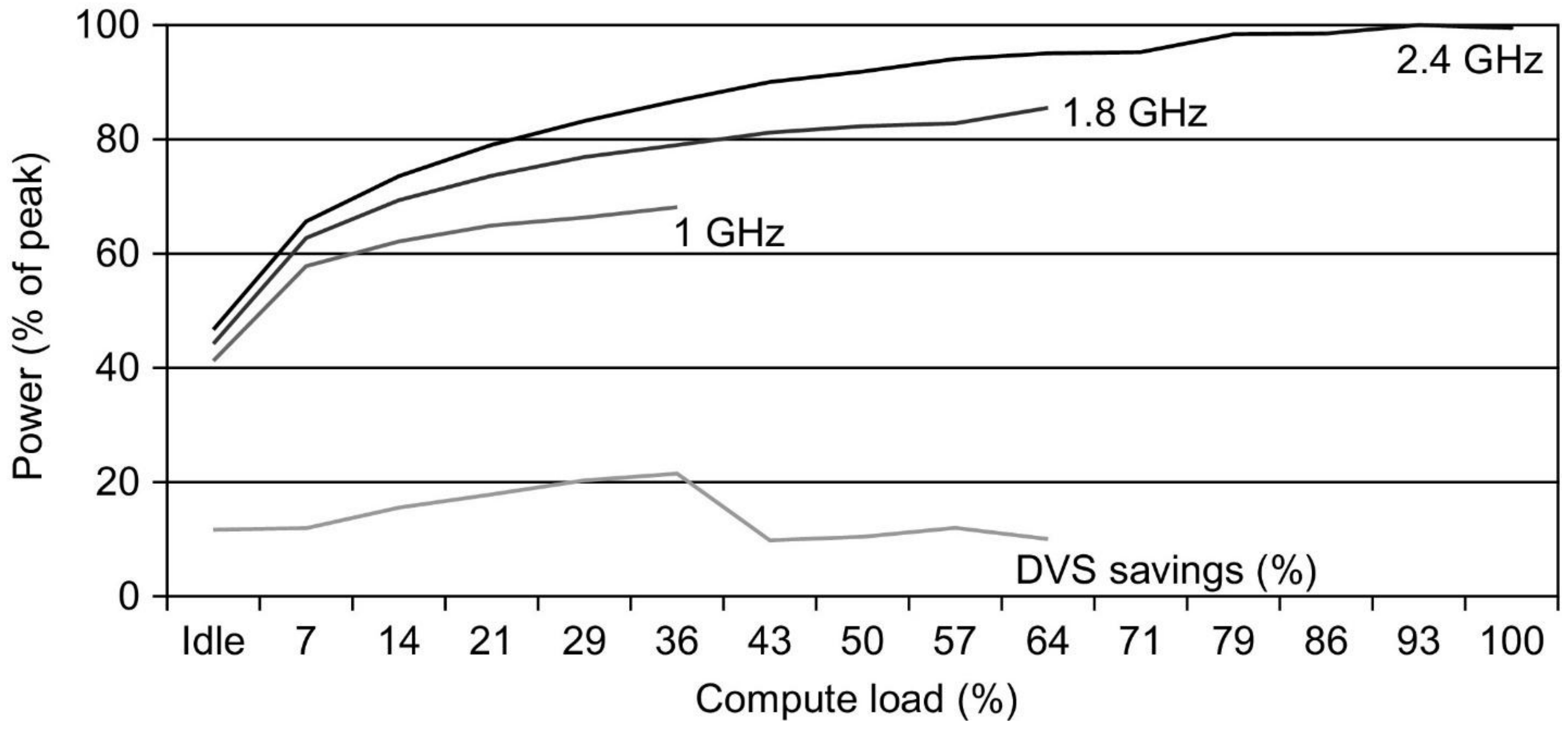


Figure 1.12 Energy savings for a server using an AMD Opteron microprocessor, 8 GB of DRAM, and one ATA disk. At 1.8 GHz, the server can handle at most up to two-thirds of the workload without causing service-level violations, and at 1 GHz, it can safely handle only one-third of the workload (Figure 5.11 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**
  - 随着晶体管尺寸的减少漏电流的大小在增加
- **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
- **Power/Ground Gating: 通过切断不使用模块的电源或地减少漏电流**
  - To inactive modules to control the loss of leakage current

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**
  - 例如：A chip might be limited to 120 watts (cooling + power supply)
- **Power Consumed = Dynamic Power + Static Power**
  - 晶体管导通和截止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)

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"：**

```
ExTime(Y)              Performance(X)
---------      =       --------------
ExTime(X)              Performance(Y)
```

本章小结(续)

- **Amdahl's 定律:**

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

- **CPI Law:**

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

- **执行时间是计算机系统度量的最实际，最可靠的方式**

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：系统配置越高，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。

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**