计算机体系结构

周学海
xhzhou@ustc.edu.cn
0551-63606864
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
- **Great Ideas in Computer Architecture**
  - Design for Moore’s Law
  - Abstraction to Simplify Design
  - Make the Common Case Fast
  - Dependability via Redundancy
  - Memory Hierarchy
  - Performance via Parallelism/Pipelining/Prediction

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
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 in quarter mil
- **2013 Type D school bus**
  - 54 passengers, quarter mile time? http://www.youtube.com/watch?v=KwyCoQuhUNA
- **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 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!

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

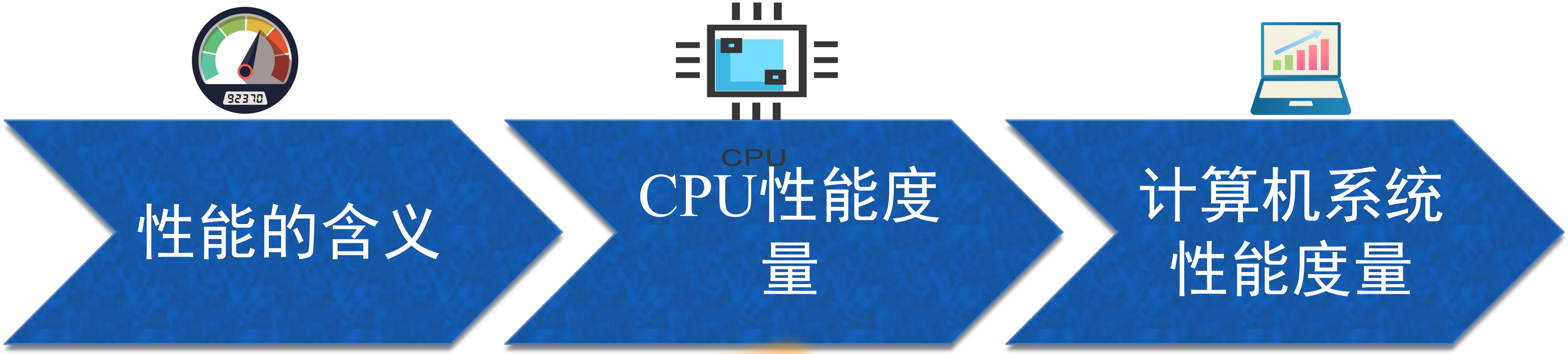
- 某程序运行在X系统上

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

- X 性能是Y的n倍”　是指

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

1.3 定量分析基础



性能的含义

CPU性能度量

计算机系统性能度量

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**

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

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

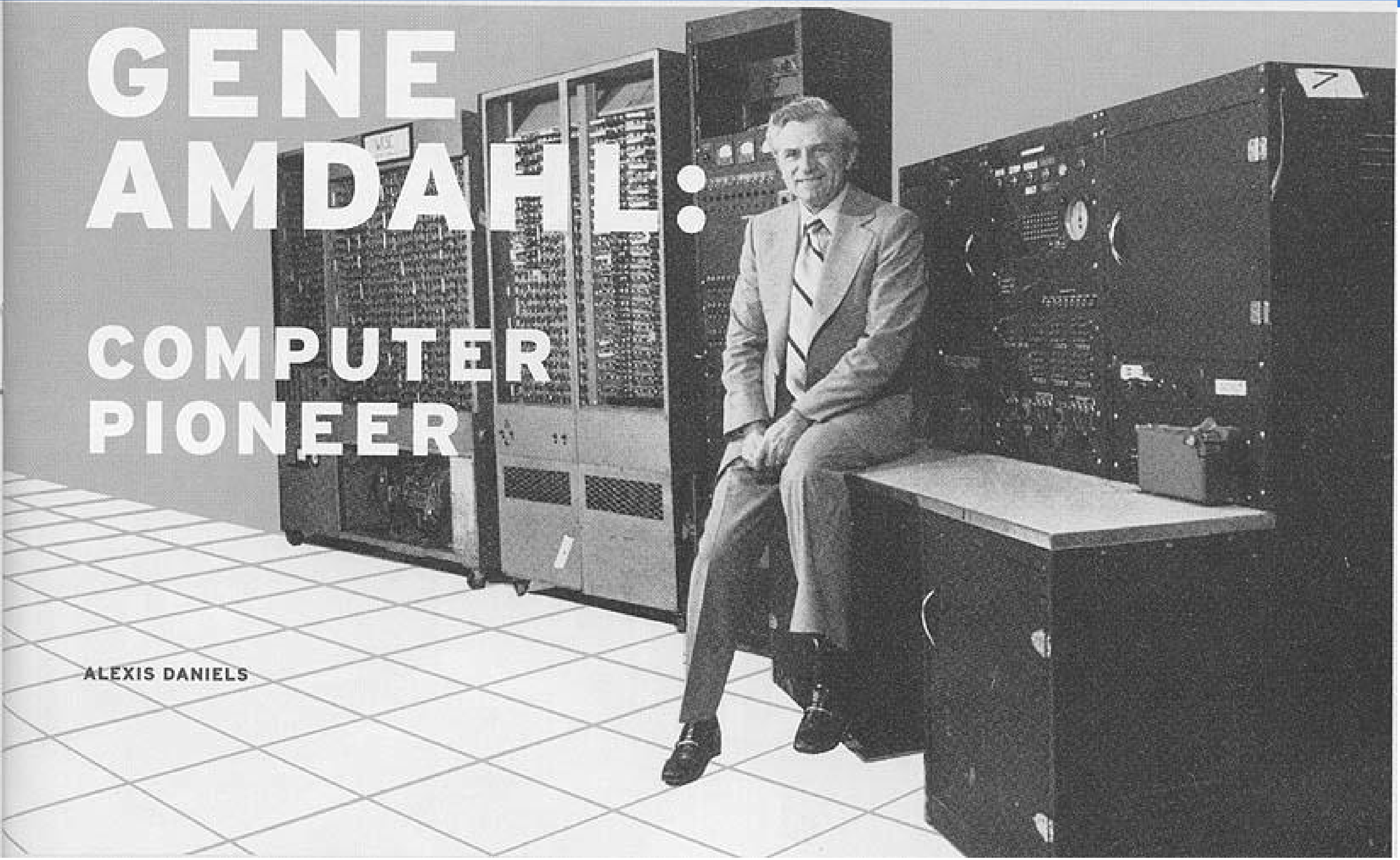
$$\text{CPI} = \sum_{i=1}^{n}(CPI_i \times Feq_i)$$

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	

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



GENE AMDAHL:

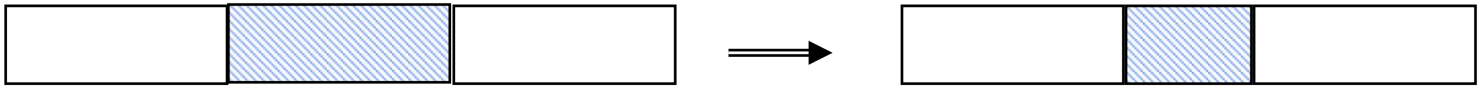
COMPUTER PIONEER

ALEXIS DANIELS

Amdahl's Law

- 假设对机器的部件进行了改进（加速比的概念）

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

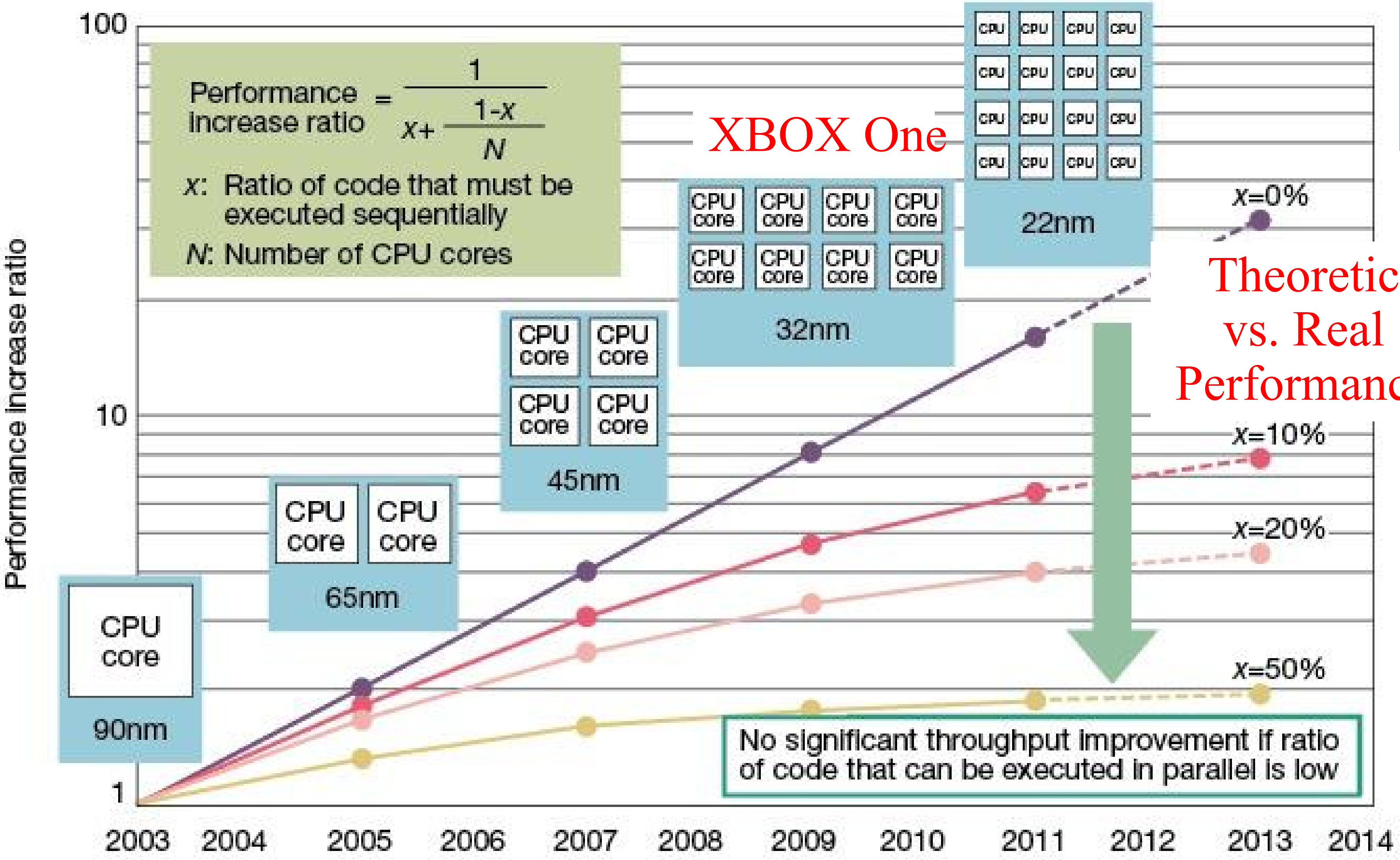


- 假设可改进部分E在原来的计算时间所占的比例为F，而部件加速比为S，任务的其他部分不受影响，则

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

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

- 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)



$$\text{Performance increase ratio} = \frac{1}{x + \frac{1-x}{N}}$$

x: Ratio of code that must be executed sequentially

N: Number of CPU cores

No significant throughput improvement if ratio of code that can be executed in parallel is low

XBOX One

Theoretic vs. Real Performance

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

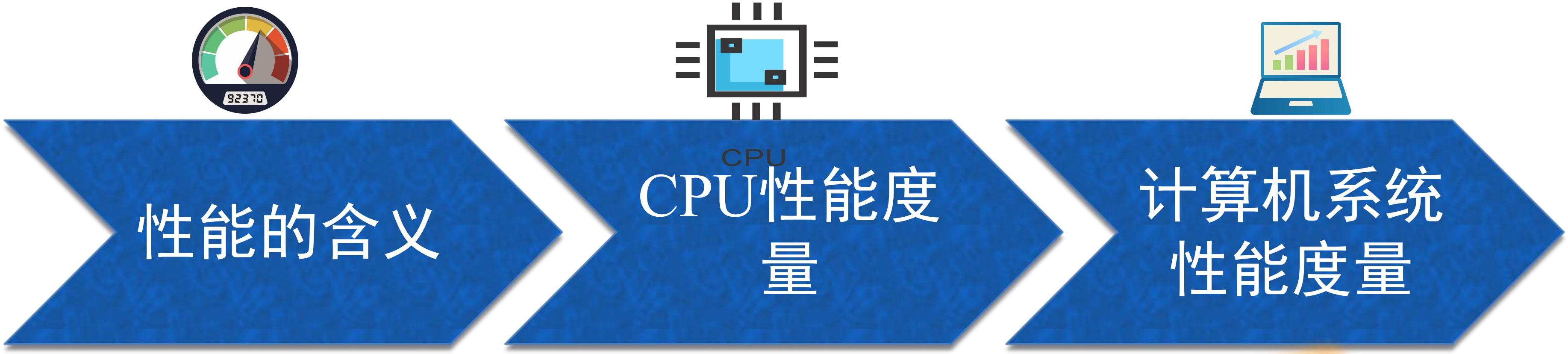
- **假设给定一体系结构硬件不支持乘法运算，乘法需要通过软件来实现。在软件中做一次乘法需要200个周期，而用硬件来实现只要4个时钟周期。如果假设在程序中有10%的乘法操作，问整个程序的加速比？如果有40%的乘法操作，问整个程序的加速比又是多少？**
- **假设一计算机在运行给定的一程序时，有90%的时间用于处理某一类特定的计算。现将用于该类计算的部件性能提高到原来的10倍。**
  - 如果该程序在原来的机器上运行需100秒，那么该程序在改进后的机器上运行时间是多少?
  - 新的系统相对于原来的系统加速比是多少?
  - 在新的系统中，原来特定的计算占整个计算的比例是多少?

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）
    - 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，每个处理器执行的任务仍然相同
  - Gustafson–Barsis’s Law （古斯塔夫森定律）
  - Speedup = N + (1-N)s

1.3 定量分析基础



性能的含义

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：每秒百万条指令数**
  - $\mathrm{MIPS} = \mathrm{IC}/(\mathrm{CPI}*\mathrm{IC}*T*10^6)=1/(\mathrm{CPI}*T*10^6)$
  - 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：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）**

(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。

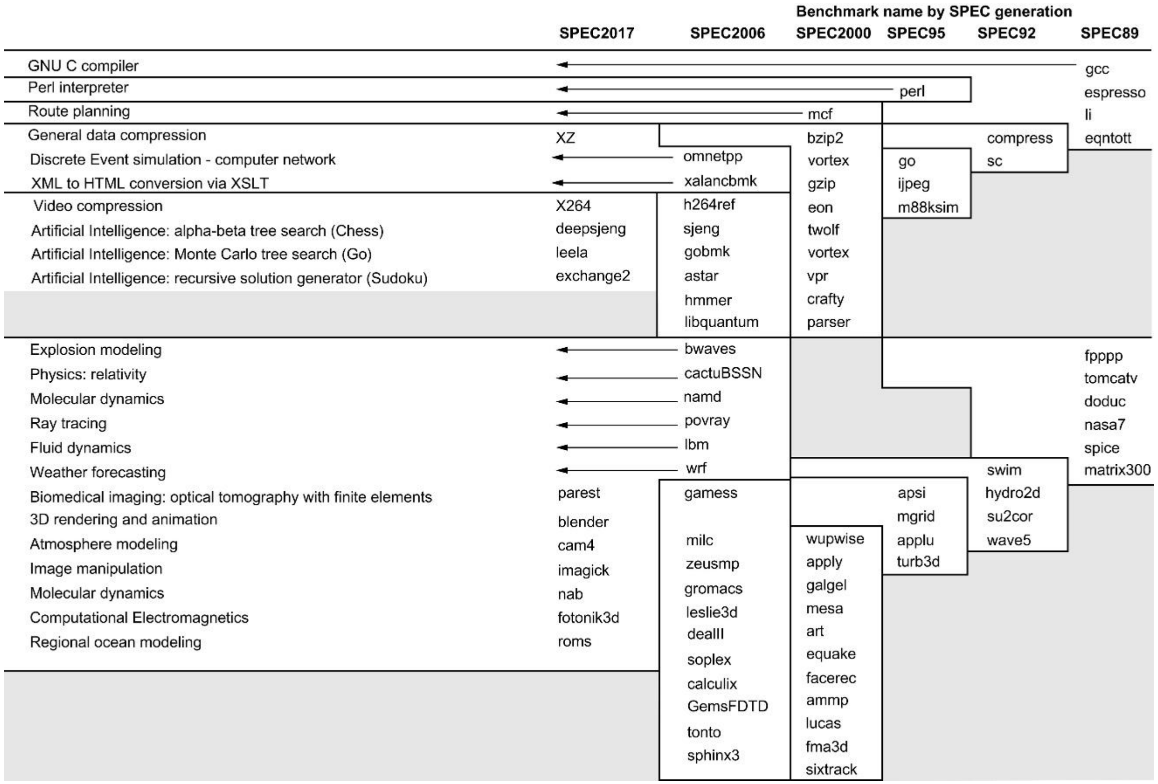
(4) 小测试程序（toy programs)：一般在100行以内。

(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

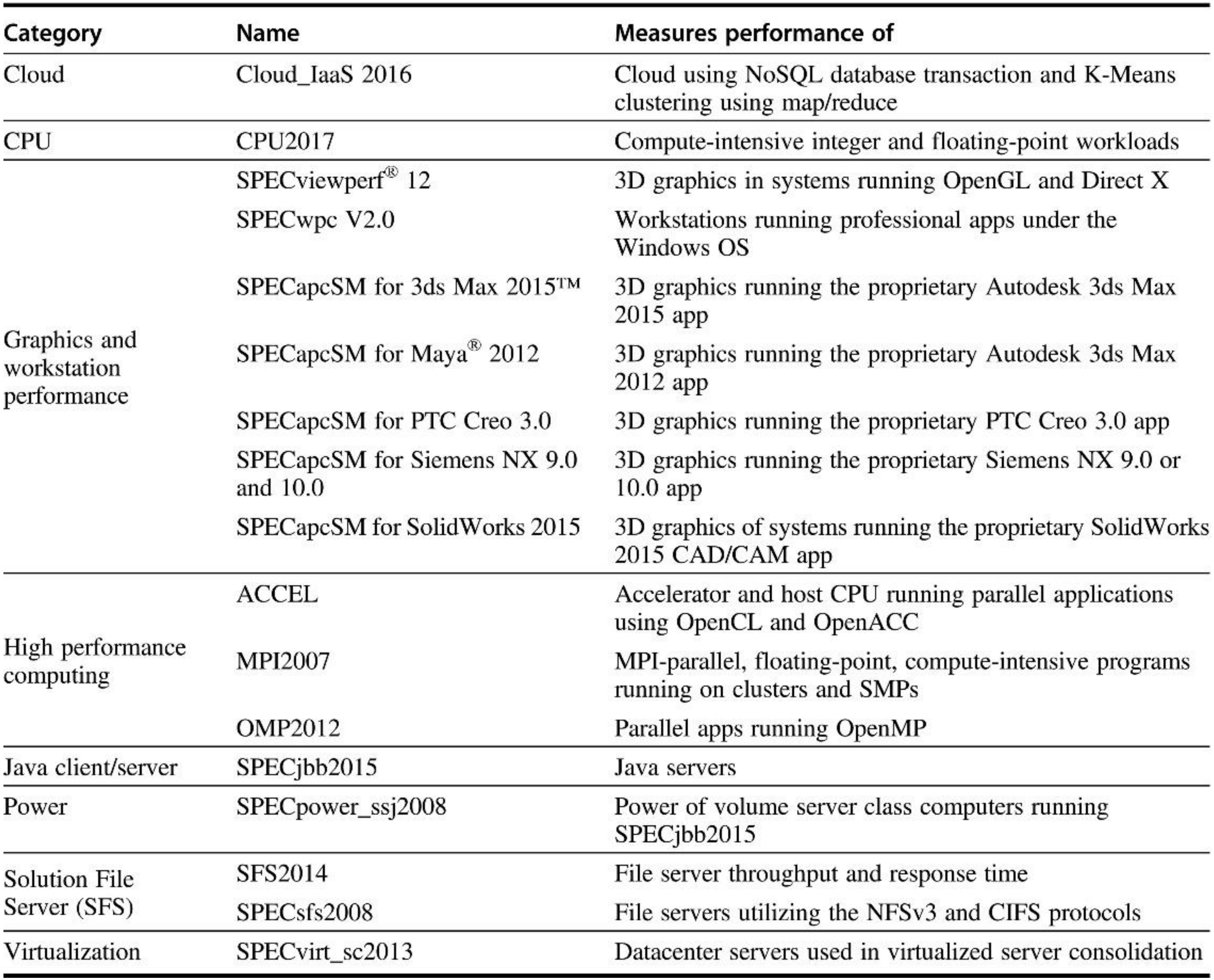
- **Embedded Microprocessor Benchmark Consortium (EEMBC)**
- **Desktop Benchmarks**
  - SPEC2017
  - SPEC2006
  - SPEC2000
  - SPEC 95
  - SPEC 92
  - SPEC 89
- **Server Benchmarks**
  - Processor Throughput-oriented benchmarks (基于SPEC CPU benchmarks->SPECrate
  - SPECSFS, SPECWeb
  - Transaction-processing (TP) benchmarks (TPC-A, TPC-C, …)
- **…..**

Standard Performance Evaluation Corporation (www.spec.org)



	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.



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)

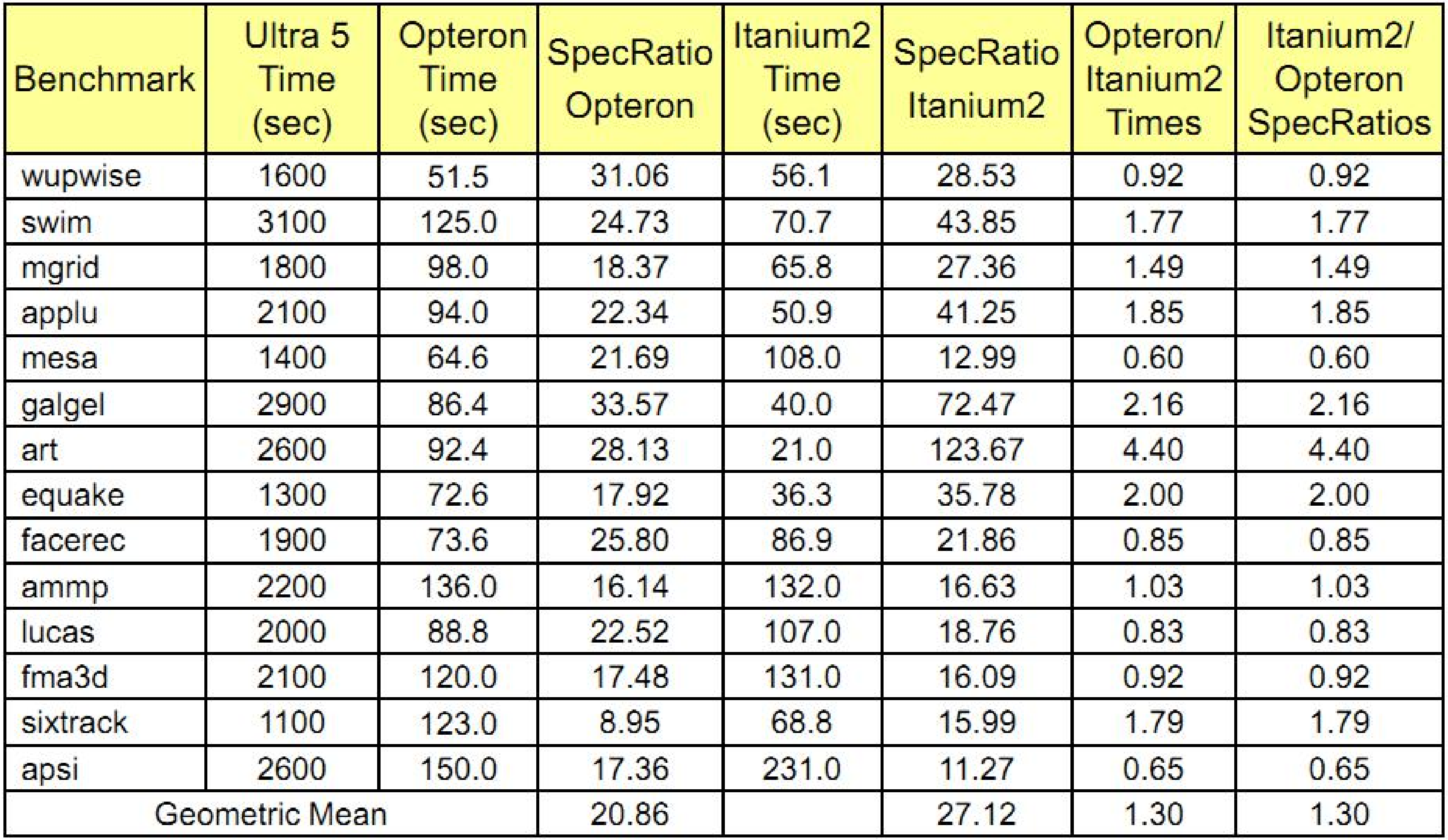
SPEC 性能综合

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

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

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

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

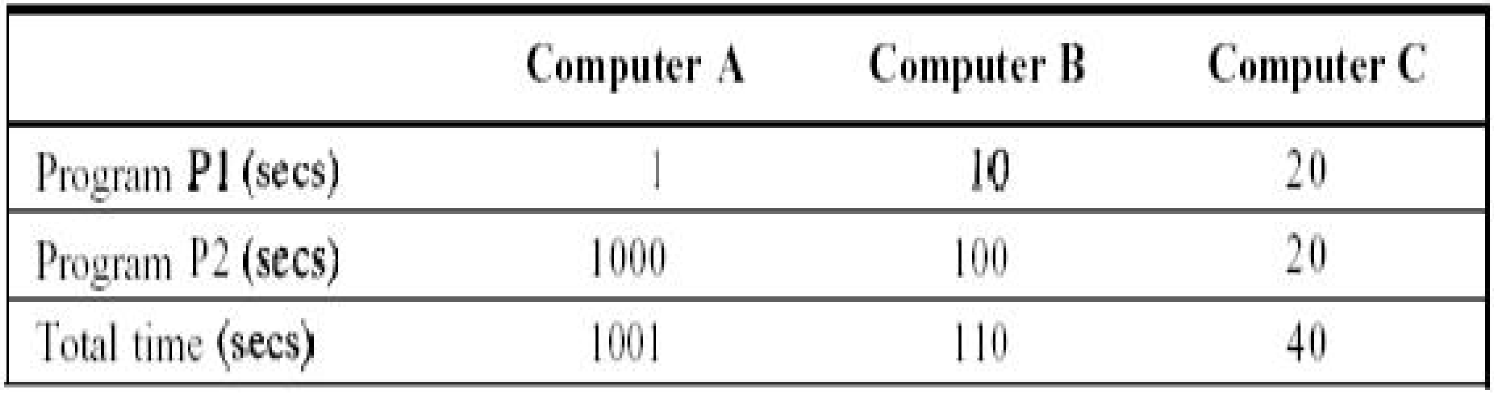
几何平均的两个重要特性

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

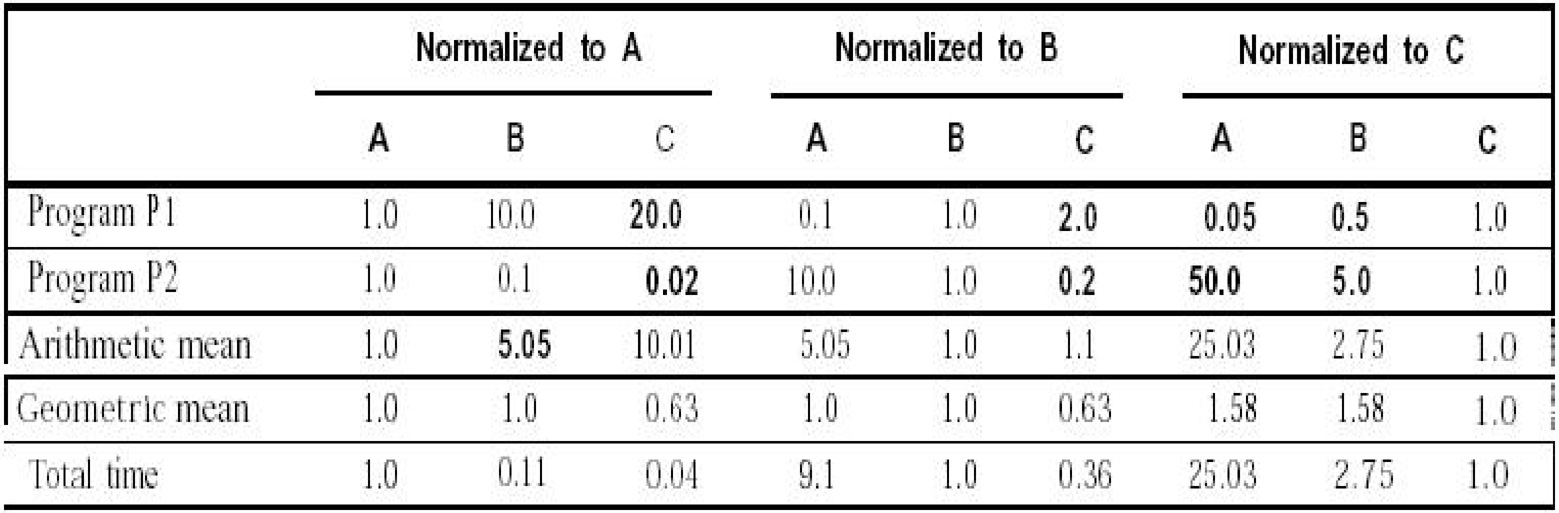
$$= \sqrt[n]{\prod_{i=1}^{n} \frac{\dfrac{\text{Execution time}_{\text{reference}_i}}{\text{Execution time}_{A_i}}}{\dfrac{\text{Execution time}_{\text{reference}_i}}{\text{Execution time}_{B_i}}}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Execution time}_{B_i}}{\text{Execution time}_{A_i}}} = \sqrt[n]{\prod_{i=1}^{n}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

- 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

几何平均的两个重要特性

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

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

- 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

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
- **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)

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

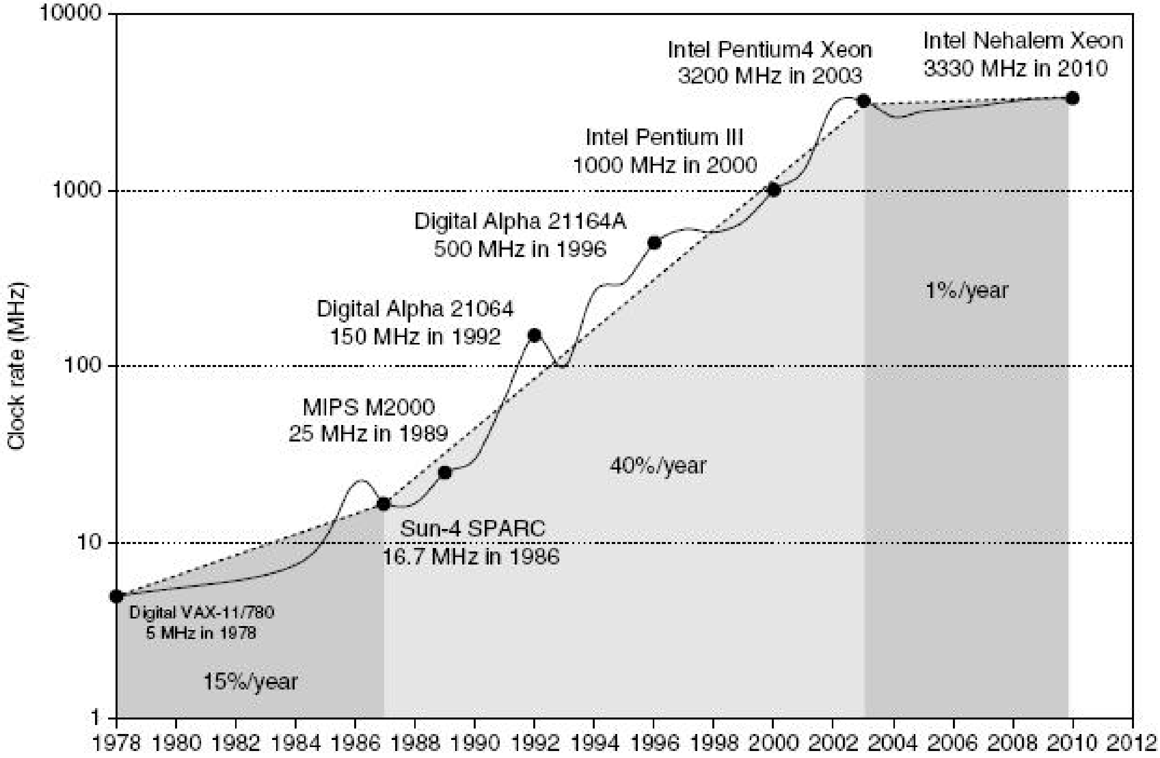
- Some microprocessors today have adjustable voltage and clock frequency. Assume 10% reduction in voltage and 15% reduction in frequency, what is the impact on dynamic energy and dynamic power?
- Answer:
  - 10% reduction in Voltage ->$\text{Voltage}_{\text{new}}$= 0.90 $\text{Voltage}_{\text{old}}$
  - 15% reduction in Frequency ->$\text{Frequency}_{\text{new}}$= 0.85 $\text{Frequency}_{\text{old}}$

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

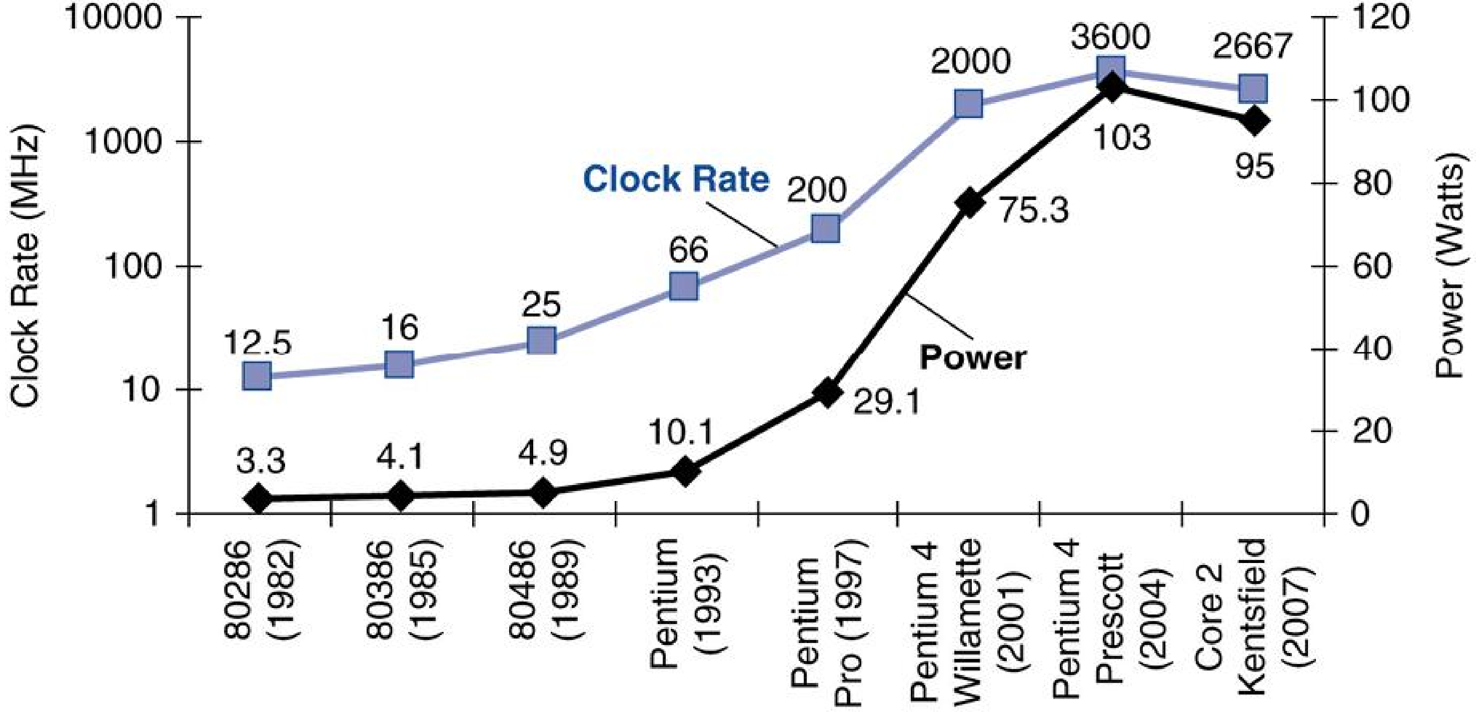
$$\frac{\text{Power}_{\text{new}}}{\text{Power}_{\text{old}}} = 0.81 \times \frac{\text{Frequency}_{\text{new}}}{\text{Frequency}_{\text{old}}} = 0.81 \times 0.85 = 0.6885$$

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



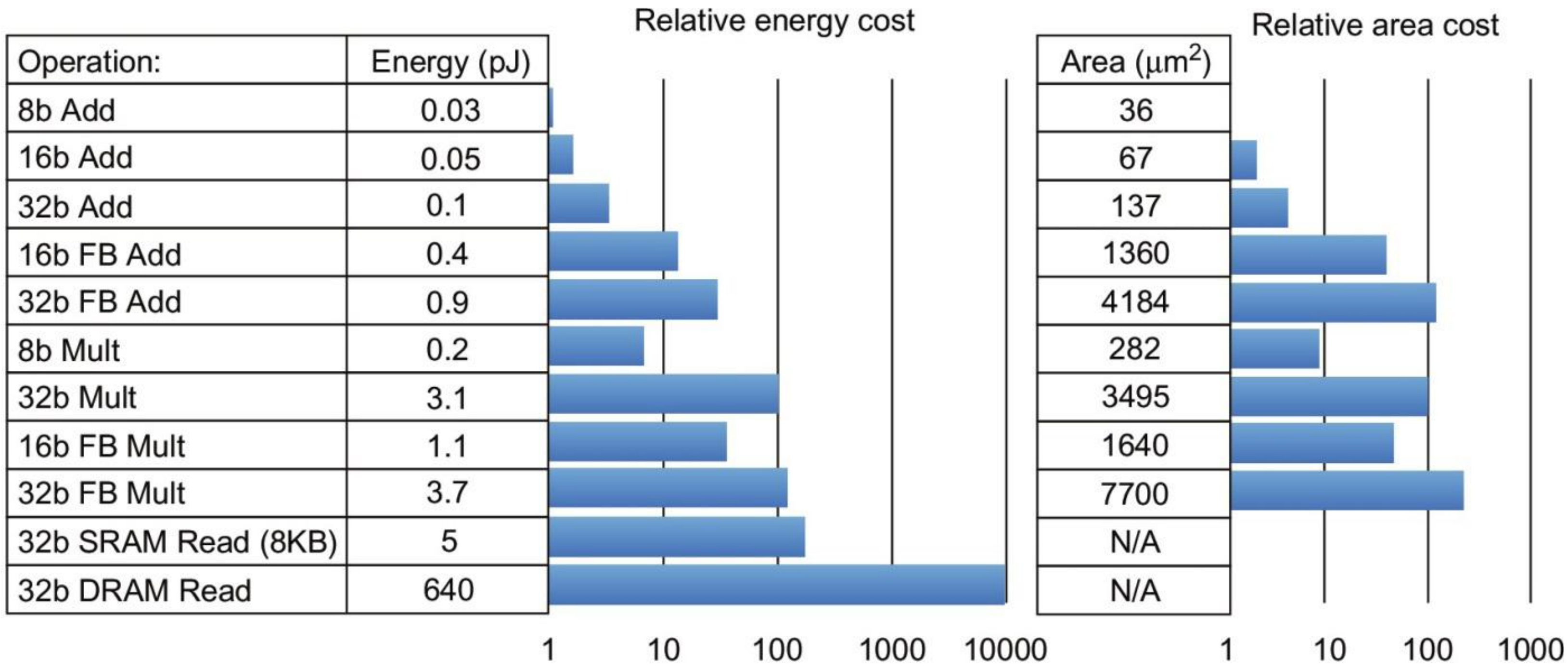
$$\text{Power} \propto \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

30X

5V→1V

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.

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

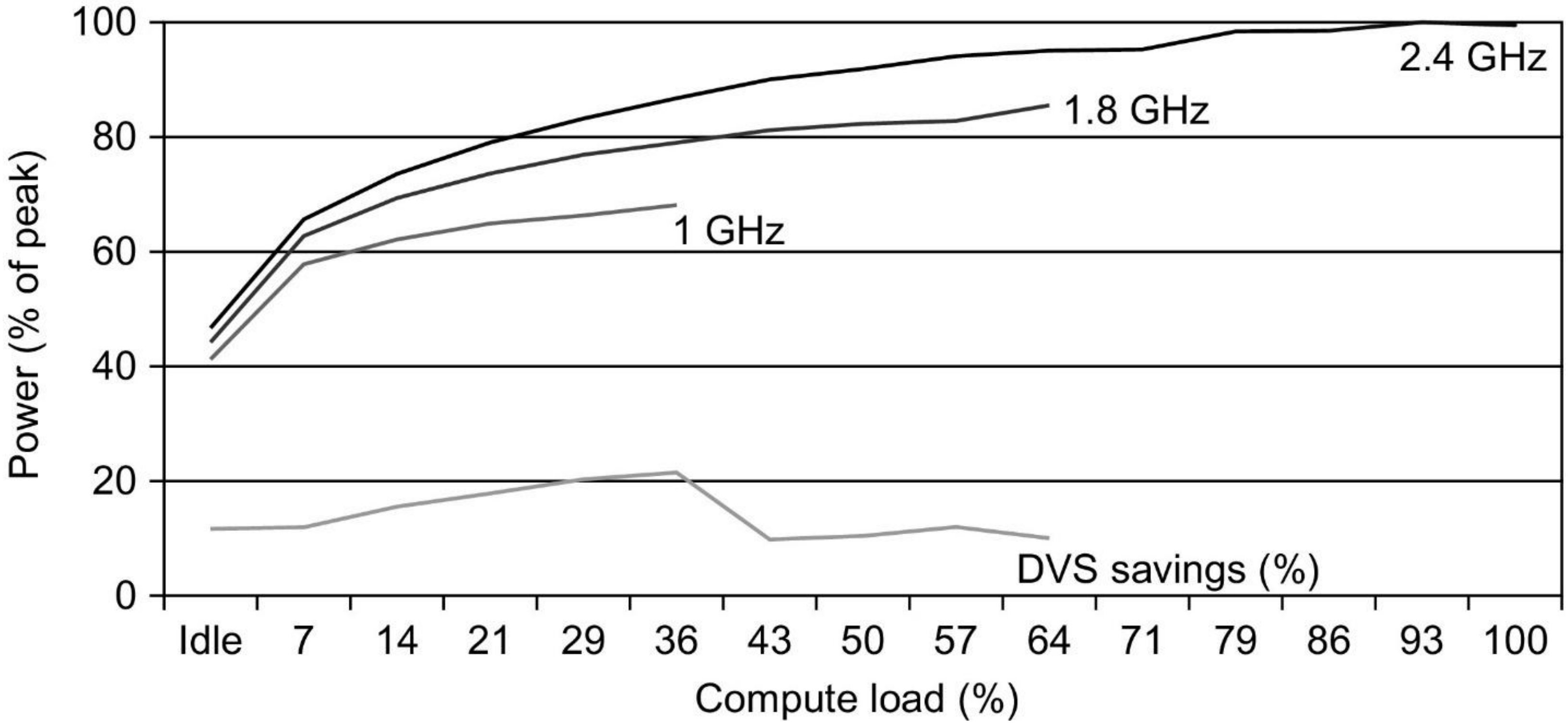


Figure 1.12 Energy savings for a server using an AMD Opteron microprocessor, 8 GB of DRAM, and one ATA disk. At 1.8 GHz, the server can handle at most up to two-thirds of the workload without causing service-level violations, and at 1 GHz, it can safely handle only one-third of the workload (Figure 5.11 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 Gating: 通过切断供电减少漏电流**
  - 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**
  - 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 * Delay2
- **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:**

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

- **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**

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。

Example

- **Suppose we have two implementations of the same ISA**
- **For a given program**
  - Machine A has a clock cycle time of 250 ps and a CPI of 2.0
  - Machine B has a clock cycle time of 500 ps and a CPI of 1.2
- **Which machine is faster for this program, and by how much?**

- **Problem: A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: class A, class B, and class C, and they require one, two, and three cycles per instruction, respectively.**
  - The first code sequence has 5 instructions: 2 of A, 1 of B, and 2 of C
  - The second sequence has 6 instructions: 4 of A, 1 of B, and 1 of C
- **Compute the CPU cycles for each sequence. Which sequence is faster?**
- **What is the CPI for each sequence**