量化设计与评估

南开大学网络空间安全学院 计算机学院

性能度量

- 如何评估计算机系统的性能?
  - 运行时间：运行程序有多快?
  - 吞吐量：每天能运行多少程序?
  - ……

- 对比两台计算机的性能
  - 计算机 x 比计算机 y 快 n 倍

$$\frac{execution\ time(y)}{execution\ time(x)} = n$$

  - 性能表示为运行时间的倒数
    - 运行时间越短，性能越高（越好）

$$n = \frac{execution\ time(y)}{execution\ time(x)} = \frac{\frac{1}{execution\ time(x)}}{\frac{1}{execution\ time(y)}} = \frac{performance(x)}{performance(y)}$$

运行时间

- 挂钟时间
  - 观测到的程序运行时间，也称为响应时间或实耗时间
  - 影响因素
    - 系统负载
    - I/O延迟
    - 操作系统开销
- CPU时间
  - 运行特定程序花费的时间
    - User time：程序本身花费的时间
    - System time：操作系统花费的时间
  - 不包含等待 I/O 和运行其它程序的时间，但包含操作系统运行时间

- 方法：运行一段代码来比较两台计算机系统的性能。

代码内容

- 代码内容应该反映购买者对计算机系统的功能需求
  - 与购买该计算机系统的用途密切相关
- 问题
  - 比较计算机系统性能时，需要运行什么程序?
  - 如何使用基准程序来进行评价?
- 基准程序(benchmark)
  - 生产商用来确定它们所生产计算机系统的相对性能
- 挑战
  - 购买者需要的是与他的功能需求相关的基准程序

基准程序

- 测量性能的基准程序的最佳候选是实际使用的应用程序
  - 购买后计算机系统运行的 就是 测试性能时运行的
  - 试图运行比实际使用的应用程序简单得多的测试程序会导致性能陷阱

- 早期基准程序的类型
  - 核心(kernel)：实际应用程序的关键小片段
    - 矩阵乘法
  - 简单程序(toy programs)：节选自初级程序设计作业
    - 快速排序
  - 综合基准(synthetic benchmarks)：为了匹配实际应用程序的配置和行为而开发的“假”程序
    - Dhrystone：评估处理器运算能力的最常见基准程序之一

存在的问题

- 基准测试游戏：编译程序生产者与计算机架构师合谋，使得计算机系统在这些“替身”程序上的表现优于在实际应用程序上的表现。

- 在运行条件上造假
  - 特定基准程序的编译器标志
    - 程序的非法改善
    - 在其它计算机上的性能下降
  - 是否允许修改源代码
    - 不允许修改
    - 允许修改，但基本上不可能修改
    - 允许修改，但输出结果不变

基准程序包

- 基准程序包(benchmark suites)：针对各种应用程序的常用处理器性能测试程序
  - 优势：任何一个基准程序的弱点都由其它基准程序来弥补
- 台式机基准程序
  - 处理器密集型：SPEC CPU2017
  - 图形密集型
- 服务器基准程序
  - 处理器吞吐量：SPECrate
  - 文件服务器和Java服务器：SPECSFS
  - 面向事务处理：TPC基准程序
- 嵌入式基准程序
  - EEMBC: **E**lectronic Design News **E**mbedded **M**icroprocessor **B**enchmark **C**onsortium

性能结果

- 性能测试报告的关键：具有重复性或再现性
- 汇总测量数据
  - 算术均值
    - 各个程序运行时间的算术均值或加权算术均值
    - 权值的意义？
  - 几何均值
    - 归一化运行时间：SPECRatio

$$SPECRatio_A = \frac{execution\ time_{ref}}{execution\ time_A}$$

$$\frac{performance_A}{performance_B} = \frac{\frac{execution\ time_{ref}}{execution\ time_A}}{\frac{execution\ time_{ref}}{execution\ time_B}} = \frac{SPECRation_A}{SPECRation_B}$$

性能结果

- 几何均值

$$Geometric\ mean = \sqrt[n]{\prod_{i=1}^{n} sample_i} = \sqrt[n]{\prod_{i=1}^{n} SPECRatio_i}$$

- 几何均值的性质
  - 比值的几何均值等于几何均值的比值
  - 几何均值的比值等于性能比值的几何均值（暗示与参考计算机选择无关）
- 几何均值的缺点
  - 对于工作负载，归一化运行时间的几何均值导致明显的不真实结论
  - 几何均值“奖励”简单的改进
    - 参考计算机：$P1 = 2s, P2 = 10000s$
    - 测试计算机：$P1 = 1s, P2 = 5000s$
    - 两种改进的几何均值是相同的，但是实现在 $P2$ 上的改进要困难得多。

题外话：三种均值

- 算术均值：适合绝对数值

$$\frac{1}{n}\sum_{i=1}^{n} execution\ time_i$$

- 几何均值：适合相对数值

$$\sqrt[n]{\prod_{i=1}^{n} normalized\ execution\ time_i} = \sqrt[n]{\prod_{i=1}^{n} \frac{execution\ time_{ref}}{execution\ time_i}}$$

- 调和均值：适合比率值

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{rate_i}}$$

理想的性能测试步骤

1. 测量实际工作负载
2. 建立实际应用程序与基准程序之间的相似度
3. 按照相似度确定每个基准程序结果的权值
4. 计算加权算术均值

- 关键：了解你对计算机系统的真实需求
  - 不要相信别人总结的所有结果
  - 尽可能做出你自己的总结

躲避陷阱

- 从错误的应用领域选择基准程序
- 从未知的应用领域选择基准程序
- 使用简单的基准程序
- 与所关心特性不匹配的基准程序
  - 关心大容量Cache却选择SPEC基准程序
- 草率地确定基准程序规模
- 草率地抽取或构造基准程序
- 简单案例太多或太少

错误观点

- 使用时钟频率或单个基准程序包的性能来判断具有相同指令集体系结构的两种处理器之间的相对性能
  - 性能的比值与频率不是线性的
- 计算机系统的最佳设计是对主要目标优化而不考虑具体实现
  - 上市时间
  - 错误设计概率
- 评估系统性能或性价比时，忽略软件成本
  - 软件费用可能占总成本的“大头”

计算机设计的量化原理

- 利用并行性：提高性能的最重要方法之一
  - 系统级并行性——提高吞吐量性能
    - 扩展性：扩展内存和处理器数目
    - 数据级并行性：使用多存储系统来并行化数据读/写操作
  - 指令级并行性——提高性能的关键
    - 指令流水操作：指令重叠执行
  - 数据级并行性——数字化设计细节
    - 组关联Cache使用多个存储体
    - 算术逻辑使用先行进位
- 局部性原理
  - 程序倾向于重复使用它们最近使用过的数据和指令
  - 局部性类型
    - 时间局部性，如分支指令预测
    - 空间局部性，如Cache块结构

计算机设计的量化原理

- 计算机系统设计的最重要和最普遍原则是

Make the Common Case fast

- 聚焦常见情形(Common Case)
  - 聚焦能源、资源配置和性能方面的常见情形
    - 处理器的取指和译码单元
  - 常见情形相比不常见情形更简单，可以更快地完成

- 关键
  - **What**：确定常见情形是什么
  - **How much**：优化常见情形可以对性能有多少提升

Amdahl定律

加速比(speedup)

- 通过系统优化而获得的性能改进

$$speedup = \frac{perf_{enhancement}}{perf_{w/o\ enhancement}} = \frac{exec\ time_{w/o\ enhancement}}{exec\ time_{enhancement}}$$

Amdahl定律

- 决定因素
  - $fraction_{enhanced}$：系统优化前，运行时间中将被优化那部分所占的比例
  - $speedup_{enhanced}$：系统优化后，被优化部分的加速比
- 优化后的运行时间

$$exec\ time_{enhanced} = exec\ time \times \left((1 - fraction_{enhanced}) + \frac{fraction_{enhanced}}{speedup_{enhanced}} \right)$$

Amdahl定律

- 整体加速比

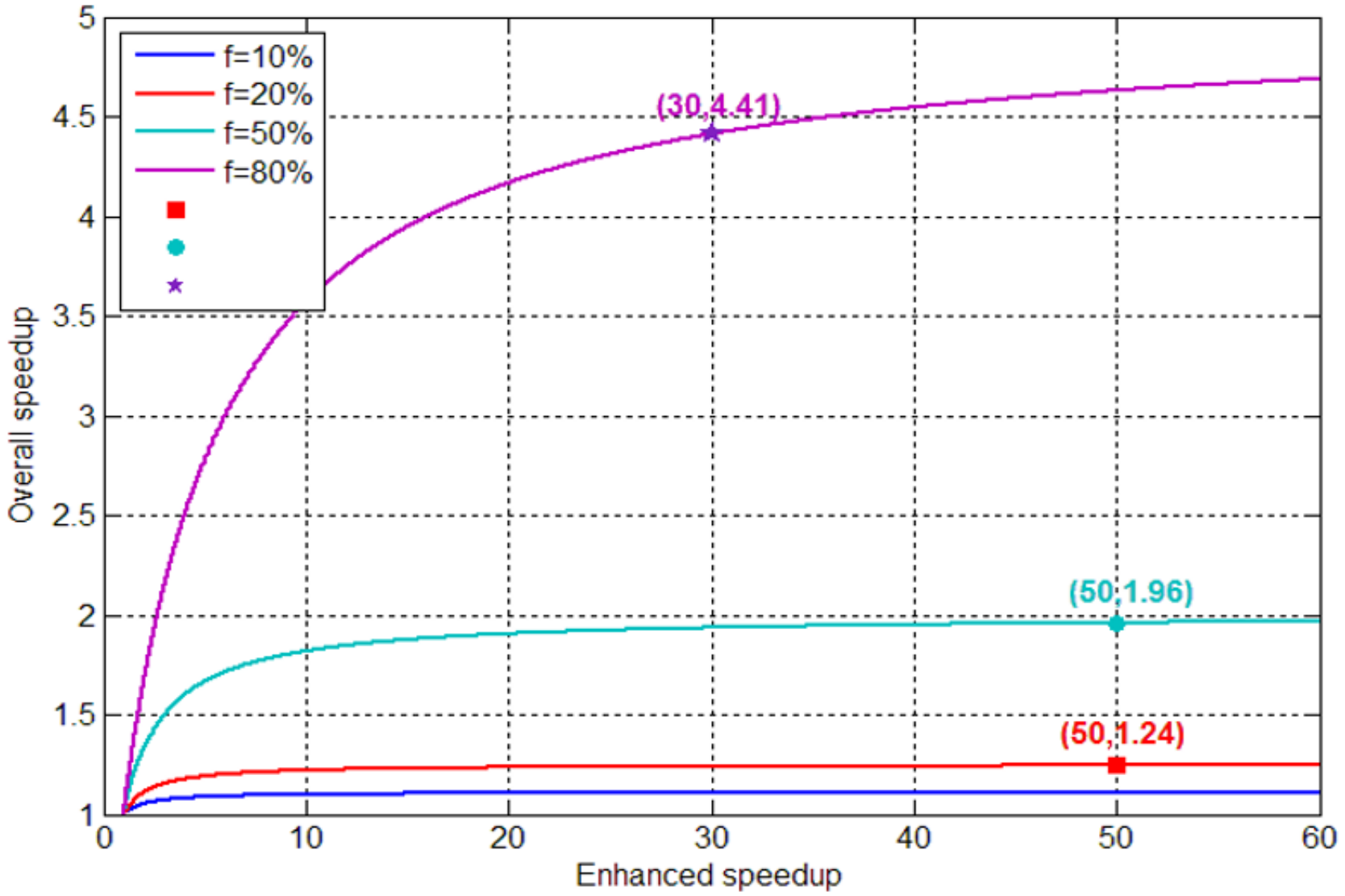
$$speedup_{overall} = \frac{1}{(1 - fraction_{enhanced}) + \frac{fraction_{enhanced}}{speedup_{enhanced}}}$$

- 例：希望通过更换处理器来改良web服务的性能。新处理器在web服务应用中的计算比老处理器快 10 倍。假设，老处理器 40% 时间用于计算，60% 时间等待 I/O 操作，那么通过更换新处理器带来的整体加速比是多少?
  - 答:

$$\because fraction_{enhanced} = 0.4, speedup_{enhanced} = 10$$

$$\therefore speedup_{overall} = \frac{1}{(1-0.4)+\frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$$

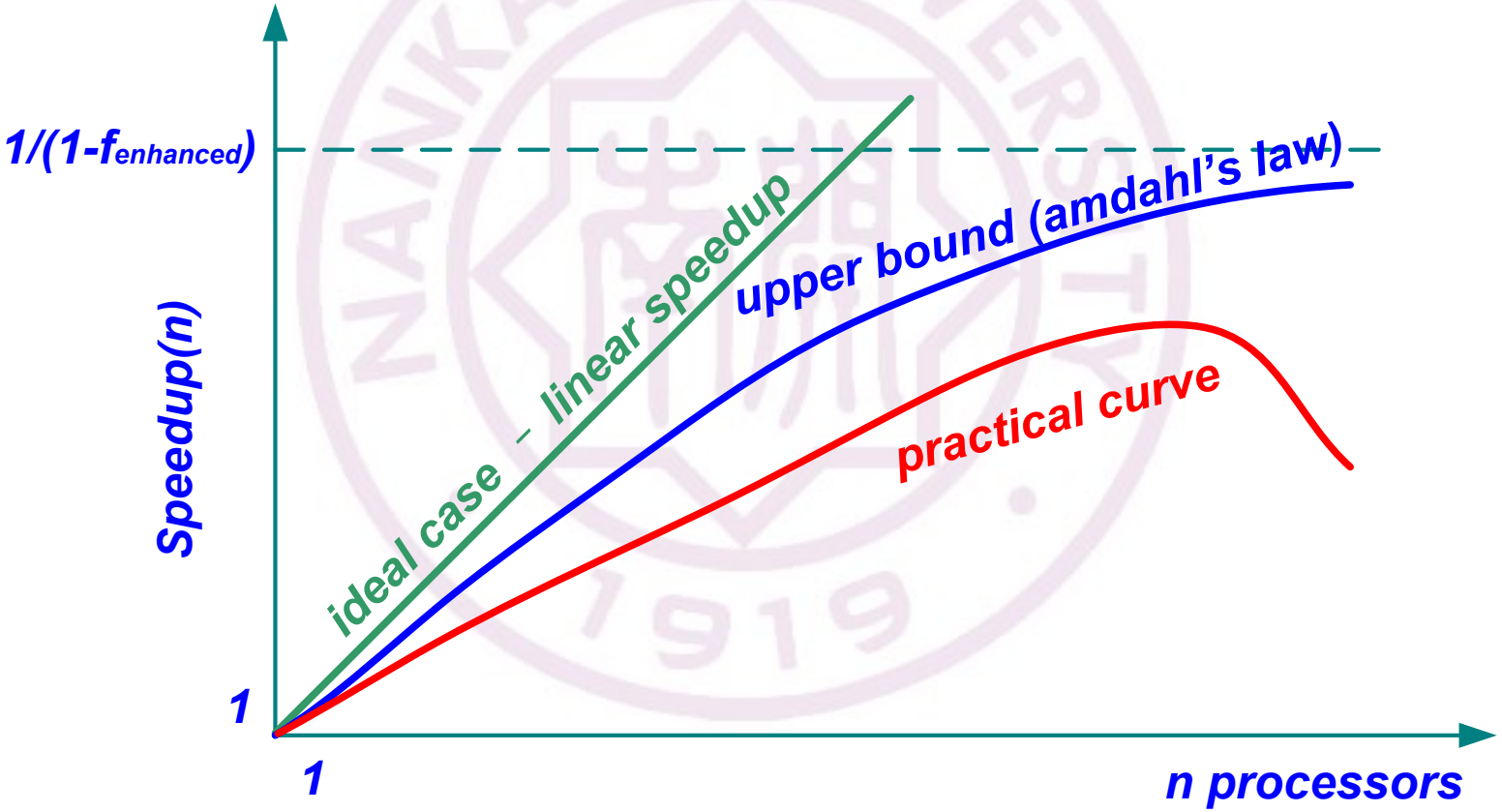
Amdahl定律


f=10%
f=20%
f=50%
f=80%
(30,4.41)
(50,1.96)
(50,1.24)
Overall speedup
Enhanced speedup
1
1.5
2
2.5
3
3.5
4
4.5
5
0
10
20
30
40
50
60

Amdahl定律的启示

- 收益递减规则
  - 加速比上限

$$speedup_{enhanced} \leq \frac{1}{1 - fraction_{enhanced}}$$



Amdahl定律的应用

- 主要用途：比较两个设计方案的优劣
- 例：图形应用程序使用大量的浮点数平方根(FPSQR)运算。假设 FPSQR 计算花费了图形基准程序 20% 的运行时间，而浮点数计算占图形基准程序运行时间的 50%。
  - 方案一：通过使用更好的硬件，改进 FPSQR 性能 10 倍

$$speedup_1 = \frac{1}{(1-0.2)+\frac{0.2}{10}} = \frac{1}{0.82} \approx 1.22$$

  - 方案二：通过购买更好的处理器，改进所有浮点数运算 1.6 倍

$$speedup_2 = \frac{1}{(1-0.5)+\frac{0.5}{1.6}} = \frac{1}{0.8125} \approx 1.23$$

深入了解 Amdahl 定律

- 计算公式

$$speedup = \frac{1}{(1-f) + \frac{f}{s}}$$

- 当 $f \to 0$ 时，$speedup = 1$；
- 当 $f \to 1$ 时，$speedup = s$；
- 当 $s \to \infty$ 时，$speedup = 1/(1-f)$；
- 当 $s = 1$ 时，$speedup = 1$。

CPU 性能方程

- 所有计算机系统组成中都包含一个固定频率的时钟
- 程序运行所需的 CPU 时间为

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

- 指令计数(Instruction Count, IC)
  - 程序运行的指令数目
- 指令周期数(Cycles Per Instruction, CPI)
  - 时钟指令数(Instruction Per Clock, IPC)是 CPI 的倒数

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

CPU 性能方程

- CPU 性能方程

$$CPU\ time = \frac{IC \times CPI}{Clock\ rate}$$

  - 表示为测量单位形式

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

- 三个参数均与体系结构特性有关，彼此不是独立的。
  - 时钟频率(Clock rate)：硬件技术和组成
  - 平均指令周期数(CPI)：组成和指令集体系结构
  - 指令计数(IC)：指令集体系结构和编译技术
- 增强方法
  - 增强某个部分，对另外两个部分有较小或可预测的影响。

CPU 性能方程

- 如果
  - 程序包含 n 种不同的指令，IC_i 表示第 i 种指令的数目
  - CPI_i 是第 i 种指令的平均时钟周期数
- 则
  - 运行该程序的 CPU 周期数为

$$CPU\ clock\ cycles = \sum_{i=1}^{n} IC_i \times CPI_i$$

有

$$CPU\ time = \left(\sum_{i=1}^{n} IC_i \times CPI_i\right) \times \frac{1}{clock\ rate}$$

$$CPI = \frac{\sum_{i=1}^{n} IC_i \times CPI_i}{instruction\ count} = \sum_{i=1}^{n} \frac{IC_i}{instruction\ count} \times CPI_i$$

  - CPI 是每种指令 CPI_i 的加权均值，权值由程序中指令的频度决定。

CPU 性能方程

- 例：
  - FP操作（不同于FPSQR）的频度为 25%，FP操作的平均 CPI 为 4.0，其它指令的平均 CPI 为 1.33，FPSQR的频度为 2%，FPSQR的 CPI 为 20。
  - 两种增强方案可选
    - 选项1：降低 FPSQR 的 CPI 为 2
    - 选项2：降低所有 FP 操作的平均 CPI 为 2.5
- 问题：哪种选项更好?
  - CPU 性能方程

$$CPU\ time = \frac{IC \times CPI}{clock\ rate}$$

  - 由于两个选项都不会改变 IC 和时钟频率，故

$$speedup_1 = \frac{CPI_{original}}{CPI_{option1}}, speedup_2 = \frac{CPI_{original}}{CPI_{option2}}$$

CPU性能方程

- 问题转换为求解每种选项的 CPI
- 本题中包含三种不同类型的操作（FP，FPSQR和其它）

$$CPI = CPI_{FP} \times 0.25 + CPI_{FPSQR} \times 0.02 + CPI_{other} \times 0.73$$

- 有

$$CPI_{original} = 4.0 \times 0.25 + 20 \times 0.02 + 1.33 \times 0.73 = 2.3709$$

$$CPI_{option1} = 4.0 \times 0.25 + 2 \times 0.02 + 1.33 \times 0.72 = 2.0109$$

$$CPI_{option2} = 2.5 \times 0.25 + 2.5 \times 0.02 + 1.33 \times 0.73 = 1.6459$$

$$speedup_{option1} = \frac{2.3709}{2.0109} \approx 1.1790$$

$$speedup_{option2} = \frac{2.3709}{1.6459} \approx 1.4405$$

- 结论：选项2具有更高的加速比（与使用Amdahl定律获得相同结果）

确定参数

- 所有方程都包含CPU time，clock rate，IC和CPI

- 指令计数(IC)
  - 只查看源代码很难确定，因为程序的运行路径是不可预测的
  - 可通过运行配置文件，其记录着完整的指令流
    - 处理器中的硬件计数器
    - 检测运行来增减软件计数器
    - 解释器仿真运行
- 指令平均周期数(CPI)
  - 对于简单系统，可通过查询每种指令的实现方式和每种指令的周期数目来确定 CPI；
  - 实际系统要复杂得多，处理器在某种情形下会减少指令的周期数，而存储系统也会引入许多延迟周期。

谬误

- 谬误：应该避免的错误理念或误解
- 多处理器是一颗银弹
  - 银弹：某种策略、技术或技巧可以极大提高程序员的生产力
  - 近几年多处理器每代只增加2核，性能改进主要靠程序员来开发并行性
- 提升性能的硬件改进也会提高能效，最坏情形下是持平
  - 2011年，Esmaeilzadeh等人在 2.67GHz Intel Core i7上使用Turbo模式运行SPEC2006。当时钟频率增加到2.94GHz时
    - 性能增加1.07倍
    - 能耗（焦耳）增加1.37倍
    - 功耗（瓦特小时）增加1.47倍
- 基准程序无限期有效
  - 许多因素影响基准程序的有用性，它将随时间变化。
  - 编译器优化目标基准程序

谬误

- 因为磁盘的额定平均故障时间为 1,200,000 小时或近 400 年，所以磁盘几乎从来不出故障。
  - 制造商提供的 MTTF 数据是假设定期更换磁盘的前提下

陷阱

- 陷阱：容易犯的错误，一般来说，原则的泛化是有条件的。
- 所有指数定律都必须结束
  - Dennard 缩放比例定律：被阈值电压终止
  - 磁盘容量：每年 30%~100% 降到每年 5%
  - 摩尔定律：
    - DRAM 容量增长预期
    - 只剩四家生产最先进逻辑芯片的厂商
    - 11nm（ITRS提出）和3nm（Intel和TSMC提出）可能是极限
- 成为Amdahl定律的牺牲品
  - 我们应该在花费精力提速之前，首先衡量一下后果是否令人失望
- 单点失效
  - 类似木桶原理，“一环软弱，全链不强”，可以通过冗余来解决。

陷阱

- 故障检测降低了可用性
  - 并非所有操作都需要正确执行

诚信 创新 实践