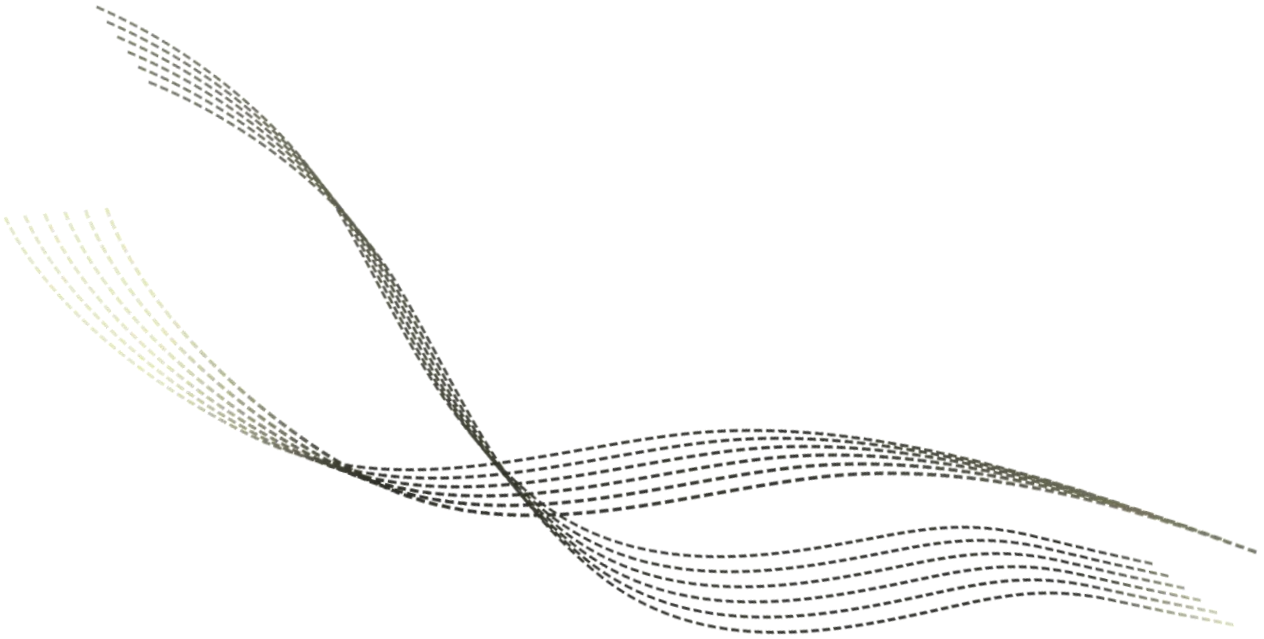
高级计算机体系结构

Advanced Computer Architecture



性能分析与评测

沈明华

目录

CONTENTS

PART 01 量化分析

■ 背景

- 组装了一台计算机
  - 我如何知道这台计算机性能有多强，计算有多快?
  - 我如何与他人计算机进行性能比较?

- 需要一个可以精确量化的指标

■ 评估指标

□ 评估指标

- 典型指标
  - ➢ 事件频率 (Events frequency)、间隔时长 (Interval durations)、规格参数 (Parameter sizes)
- 优秀指标应具备的特征
  - ➢ 支持无歧义的精确比较
  - ➢ 易于建模
  - ➢ 有意义且可操作性强
- 例如
  - ➢ 越高越好的指标：计算速度、每秒浮点数操作数、并行加速比
  - ➢ 越低越好的指标：成本、功耗、CPI、内存延迟
  - ➢ 其他指标：利用率

■ 阿姆达尔定律 (Amdahl's Law)

- 在并行计算领域，假设
  - 有比例为f的代码是可并行化的
  - 其余$1-f$的代码只能串行执行
  - 可并行化部分的加速比是线性的
- n个处理器的有效加速比

$$\frac{S(n)}{S(1)} = \frac{1}{1-f+\frac{f}{n}} \qquad \lim_{n\to\infty} \frac{S(n)}{S(1)} = \frac{1}{1-f}$$

- 阿姆达尔定律给出理论加速比
  - 理论加速比由代码中不能并行的部分决定

■ CPI（每指令时钟周期数）

- 假设
  - IC_i为第i类指令的数量，CPI_i为第i类指令的平均时钟周期数

$$\text{平均 } CPI = \sum \frac{IC_i \times CPI_i}{Instruction\ count}$$

$$CPU\ time = (\sum IC_i \times CPI_i) \times clock\ cycle\ time$$

- 示例：
  - 假设浮点操作占总指令的25%，这类指令平均需要4个时钟周期，其他指令占75%，平均需要1.33个时钟周期
  - 则平均CPI为：

$$CPI = (4 \times 25\%) + (1.33 \times 75\%) = 2.0$$

■ 内存性能

- 每指令缺失数

$$\frac{\text{缺失数}}{\text{指令数}} = \text{缺失率} \times \frac{\text{内存访问次数}}{\text{指令数}}$$

- 平均内存访问时间

$$\text{平均内存访问时间} = \text{命中时间} + \text{缺失率} \times \text{缺失惩罚}$$

- 平均内存访问时间 (多级缓存)

$$\text{平均内存访问时间} = \text{命中时间}_{L1} + \text{缺失率}_{L1} \times \text{缺失惩罚}_{L1}$$

$$\text{缺失惩罚}_{L1} = \text{命中时间}_{L2} + \text{缺失率}_{L2} \times \text{缺失惩罚}_{L2}$$

PART 02 性能评估

■ 背景

- 确定评估指标内容，然后设计实验获得对应指标数据
  - 例如评价一台计算机的吞吐量、完成任务的用时
    - 有些数据无法直接测定，需要利用公式间接获得
  - 评价一台计算机的功耗
    - 评估计算机的平均耗电量，从而估计运营成本等等

■ 利特尔定律 (Little's Law)

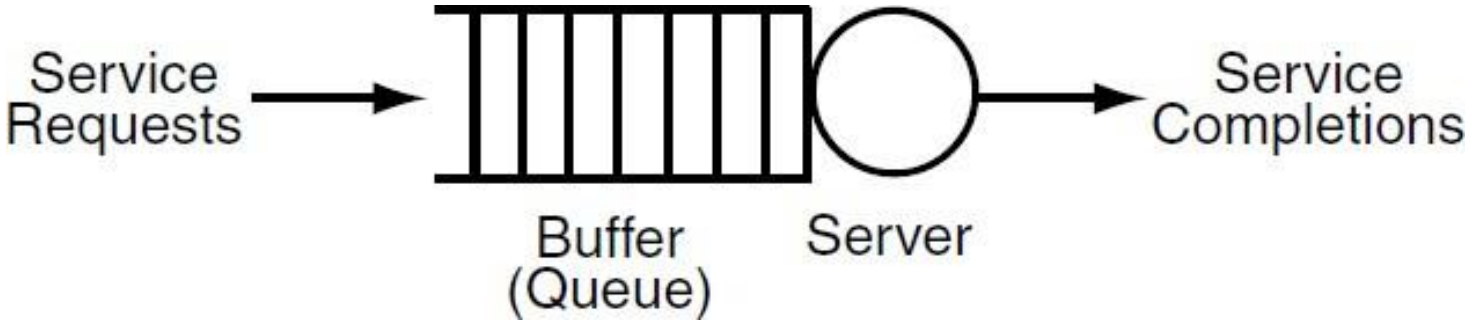
- 假设在稳定系统中
  - λ是平均作业到达率 (单位时间内进入系统的作业数量)
  - W是单个作业在系统中的平均停留时间
- 系统队列的平均长度:

$$L = \lambda W$$

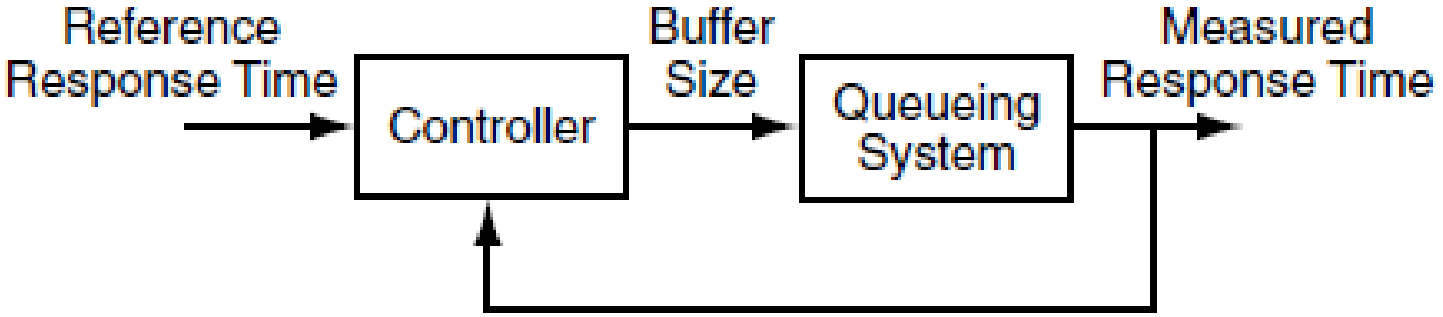
- 示例:
  - 假设每分钟有100个请求进入系统
  - 每个请求平均的停留时间为$W = 30s$
  - 系统队列平均长度$L = 100 \times 0.5 = 50$

■ 系统响应时间评估

- 排队系统被广泛应用于计算系统

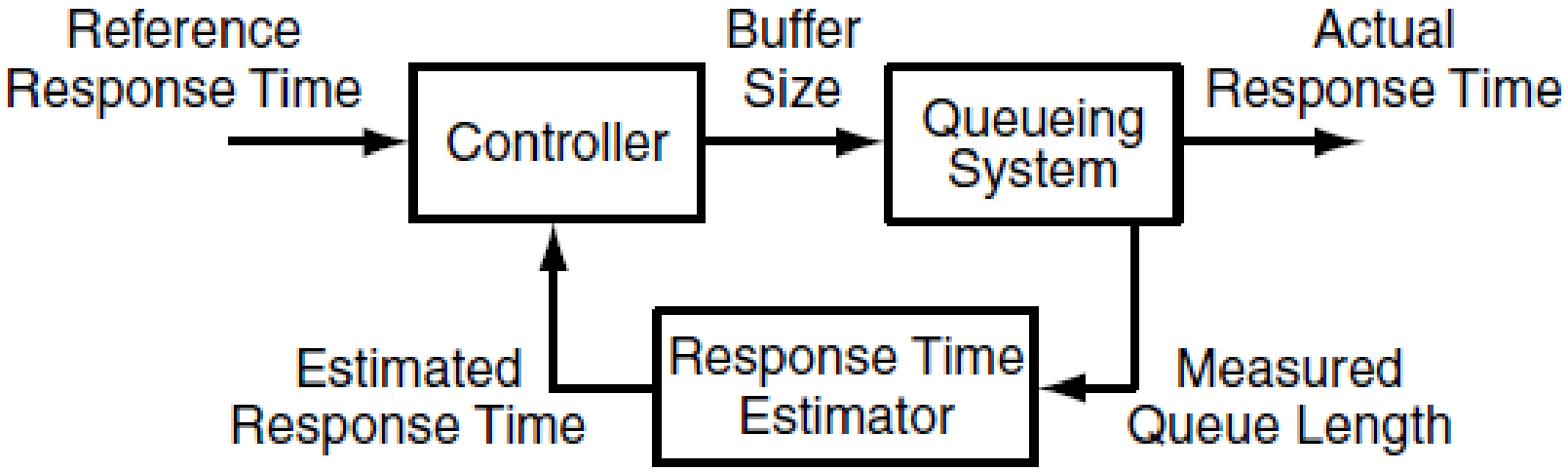


- 排队系统的存在，使得准确测量系统响应时间十分有挑战性
  - 因为无法获知排队时长，所以准确测量可能不切实际或者非常耗时



■ 系统响应时间评估

- 不能准确测量系统响应时间
  - 只能通过间接测量的方式
  - 使用传感器实时测量当前队列长度和任务到达速率
  - 使用利特尔定律$L/\lambda = W$反向推导系统响应时间



■ 功耗评估

□ 动态功耗

- C: 负载电容
- V: 供电电压 (通常低于1.5V)
- A: 活动因子， $A \in [0,1]$
- f:工作频率

$$P = a \times CV^2Af$$

□ 频率f通常与电压呈线性关系，即$f = kV$

- 因此动态功耗通常与供电电压呈立方关系

计算机速度与功耗关系

- CPU功耗与运行频率呈立方关系
  - CPU频率提升时，功耗会被立方级别放大
- 计算机性能高低通常由CPU频率决定
- 那么计算机性能和功耗也呈现立方关系吗?
  - 实验结果显示计算机功耗与性能的关系几乎是线性的
    - 一方面受限于CPU的电压以及运行频率不能无限增加
    - 另一方面这个动态功耗的结论不适用于计算机所有组件

■ 计算机功耗估计

- 广泛使用线性估计模型估计计算机功耗
  - P_{dyn}：动态功耗
  - P_{idle}：空闲功耗 (可以视为静态功耗)
  - U：系统利用率

$$P_{total} = P_{dyn} \cdot U + P_{idle}$$

- 示例
  - 假设CPU平均利用率为30%，系统全负荷时的最大功耗需求为180W，空闲功耗为100W
  - 平均功耗为$100 + (180 - 100) \times 0.3 \ = \ 124W$

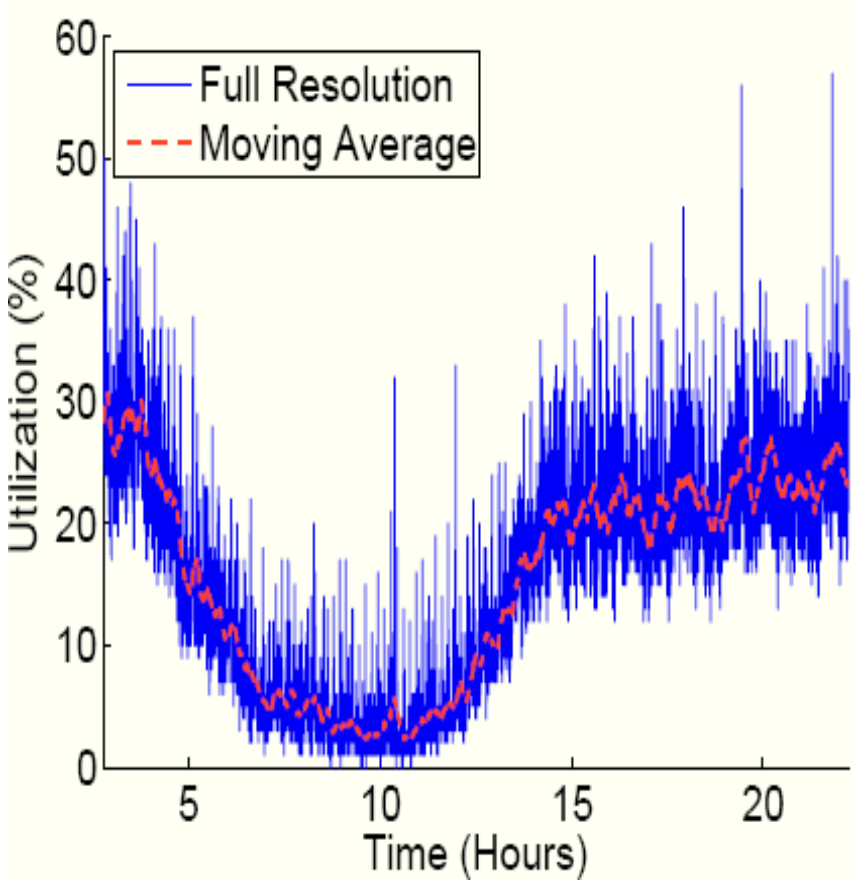
■ 计算机功耗估计

□ 优势

- 能对整个系统的平均功耗进行全面评估
- 将计算机CPU的利用率与功耗关联
- 当计算机数量很多时，估计结果相当准确

□ 劣势

- 无法预测峰值功耗



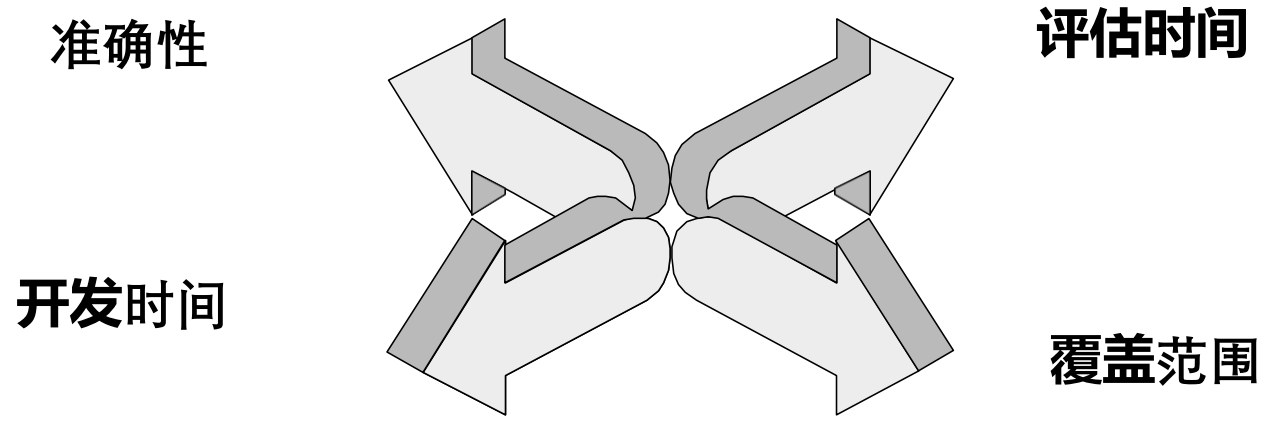
PART 03 架构仿真

■ 背景

- 架构越来越复杂，依赖芯片实体测试架构性能的做法成本高
  - 芯片设计人员需要不断迭代架构设计，从而找到功耗、性能、成本的最优解
  - 在此期间需要不断评估当前架构的各项指标
  - 利用仿真模拟器快速获得当前设计的各项指标可以显著降低成本
  - 人们开发各种各样的模拟器
    - 成为人们探索下一代处理器设计方向的重要工具

■ 计算机架构评估工具

- 模拟器的四个相互制约的因素
  - 准确性：模拟模型对真实系统的还原程度
  - 评估时间：运行一次模拟所需的时间
  - 覆盖范围：可探索的设计空间比例 (不同参数、应用场景...)
  - 开发时间：开发模拟器的时间
- 从原理上可以分为三种模拟器类型



■ 第一种：Functional 仿真

- 只对指令集架构的功能进行建模
  - 不考虑执行时延、硬件时钟等
  - 本质上是指令集仿真器，只关注指令的功能
- 主要应用于设计验证
  - 验证设计的正确性而不是评估系统性能
- 可以生成指令调用的堆栈信息
  - 可以作为其他模拟工具的输入

■ 第二种：Trace-Driven仿真

- 将程序执行指令和指令地址输入微体系结构时序模拟器
  - 将功能仿真和时序仿真解耦，两者可独立开展

- 缺点
  - Trace文件包含海量指令/地址信息，存储压力大
  - 分支预测错误时，路径模拟不准确
    - Trace文件中不会包含分支预测错误后被作废的指令，这些指令可能影响了缓存行或分支预测器

■ 第三种：Execution-Driven仿真

- Execution-Driven仿真
  - 实际应用中主流的仿真方法
  - 结合功能仿真与时序仿真
    - 既关注指令功能的正确性，也关注指令的执行时间、硬件的时序细节等
  - 相较于Trace-Driven仿真，准确性更高
  - 逻辑更复杂，增加模拟器开发和仿真耗时
- 代表工具
  - SimpleScalar、GEMS、Simics、M5、PTLSim...

■ 仿真方式对比

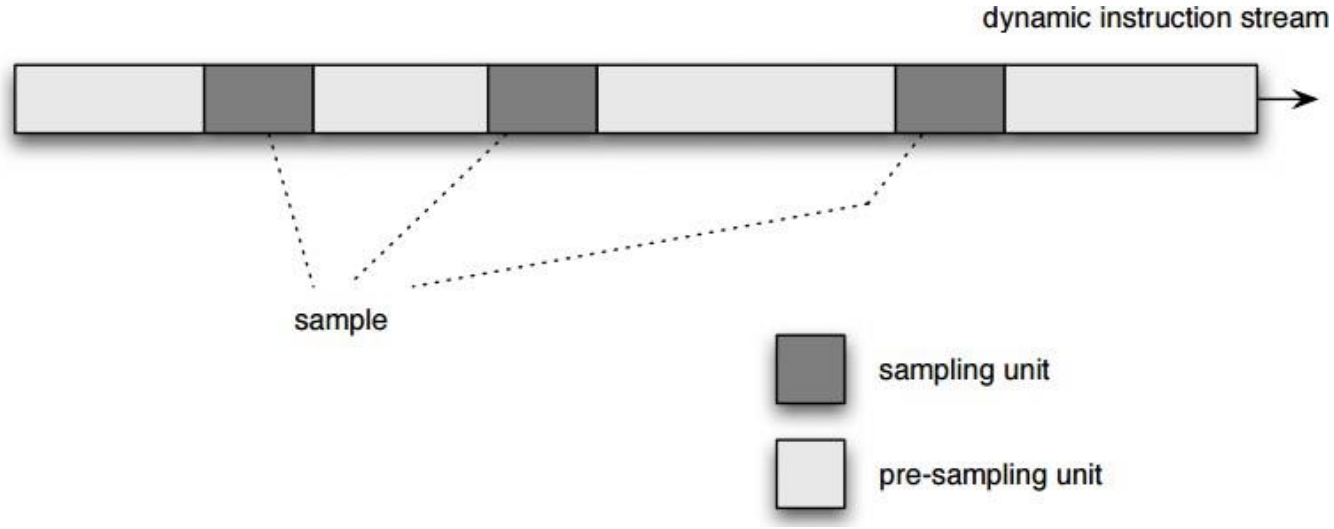
□ 全系统仿真

- Trace-Driven仿真、Execution-Driven仿真
- 模拟整个计算机系统的运行，包含硬件行为
- 更多仿真工具
  - http://pages.cs.wisc.edu/~arch/www/tools.html

	Functional 仿真	Trace-Driven 仿真	Execution-Driven 仿真
开发时间	极好	差	极差
评估时间	好	差	极差
准确度	极好	很好	极好
覆盖率	差	极好	极好

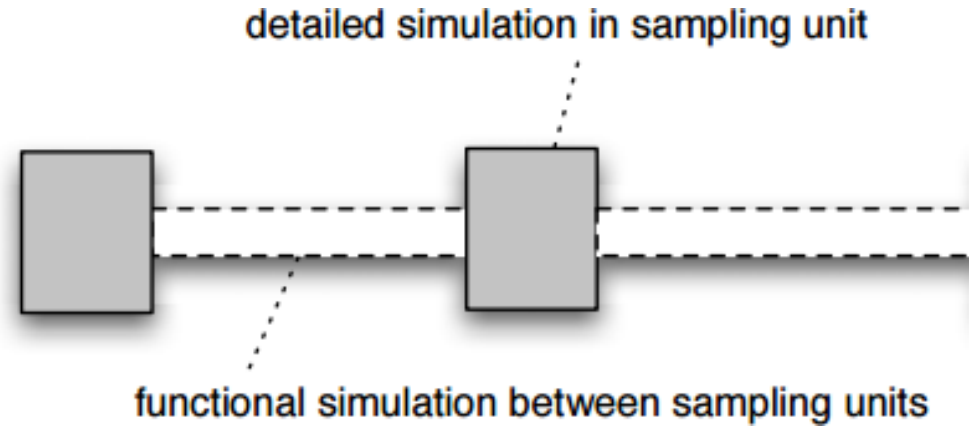
■ 模拟器加速

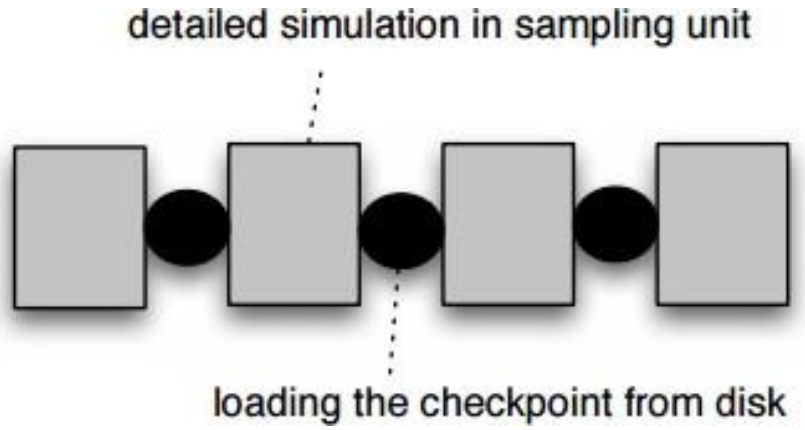
- 周期精确 (circle-accurate) 仿真效率低
  - 需要逐周期精确仿真微架构的运行细节
- 采样仿真
  - 只采样少部分动态指令进行周期精确的仿真
  - 需要准确提供采样单元的起始状态 (ASI, architecture starting image)
    - ASI包括寄存器状态、内存状态...



■ 模拟器加速

- 快速推进
  - 利用functional 仿真快速构建架构状态
  - 仿真时切换functional 仿真与Execution-Driven仿真兼顾效率和准确度
- 添加检查点
  - 在采样单元执行前，保存寄存器、内存状态
  - 运行该采样单元时只需加载预先保存的状态数据，再更新仿真器，无需重头开始仿真





PART 04 工作负载设计

■ 背景

- 工作负载
  - 是用户在机器上运行的程序与操作系统命令的组合
- 工作负载优先选择
  - 真实应用：实际场景中运行的程序
  - 修改后的应用：对真实应用进行适配调整
- 某些场景可使用简单程序作为工作负载
  - 快速测试
  - 基础性能验证

■ 基准测试套件

- 基准测试套件
  - 将多个基准测试(工作负载)整合，定量衡量计算机系统在各类应用场景下的性能
- 过去流行的基准测试套件
  - SPEC 95, SPEC CPU 2000, SPEC CPU 2006 …
  - TPC-A, TPC-C, TPC-W…
  - LINPACK
- 新兴基准测试套件
  - CloudSuite 3.0 (http://cloudsuite.ch/)
  - Rodinia (http://www.cs.virginia.edu/~skadron/wiki/rodinia/)
  - Hibench (https://github.com/intel-hadoop/HiBench)

■ 基准测试套件 CloudSuite

- **数据分析**
  - 基于维基百科数据集，运行朴素贝叶斯分类器
- **数据缓存**
  - 基于推特数据集，搭建推特缓存服务器
- **数据服务**
  - 采用雅虎云服务基准测试
- **图分析**
  - 对大规模数据集进行图分析
- **内存分析**
  - 基于用户-电影评分数据，在内存中运行协同过滤算法
- **网页搜索**
  - 基于Apache Solr搜索引擎框架
- **网页服务**
- **媒体流**

■ 工作负载设计

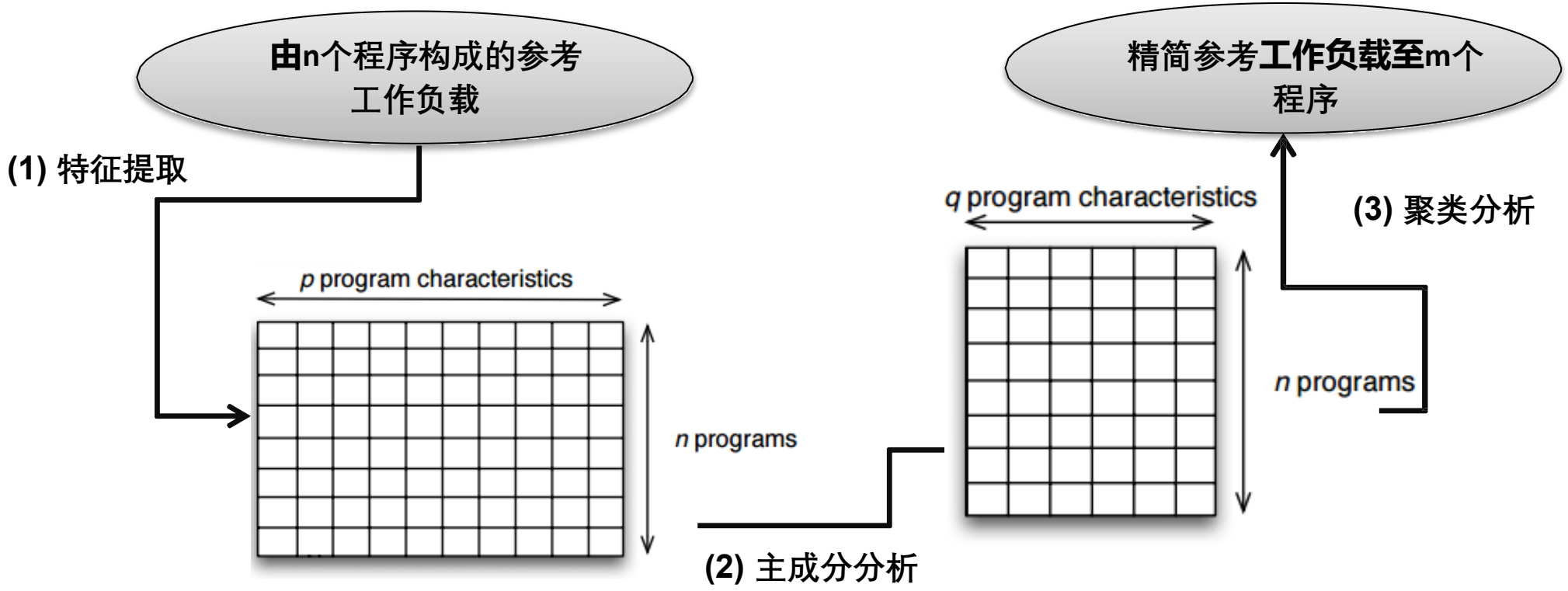
□ 运用以下策略科学设计工作负载

- 工作负载特征分析
  - ➢ 通过硬件监视器、模拟器等工具，分析基准测试程序的运行规律
- 主成分分析
  - ➢ 成熟的统计数据分析技术
  - ➢ 将多个可能关联的变量转换为少量不相关的主成分
- 聚类分析
  - ➢ 使用K-means等聚类算法挑选最具多样性的基准测试集
  - ➢ 使基准测试尽可能覆盖更多不同场景

■ 工作负载设计

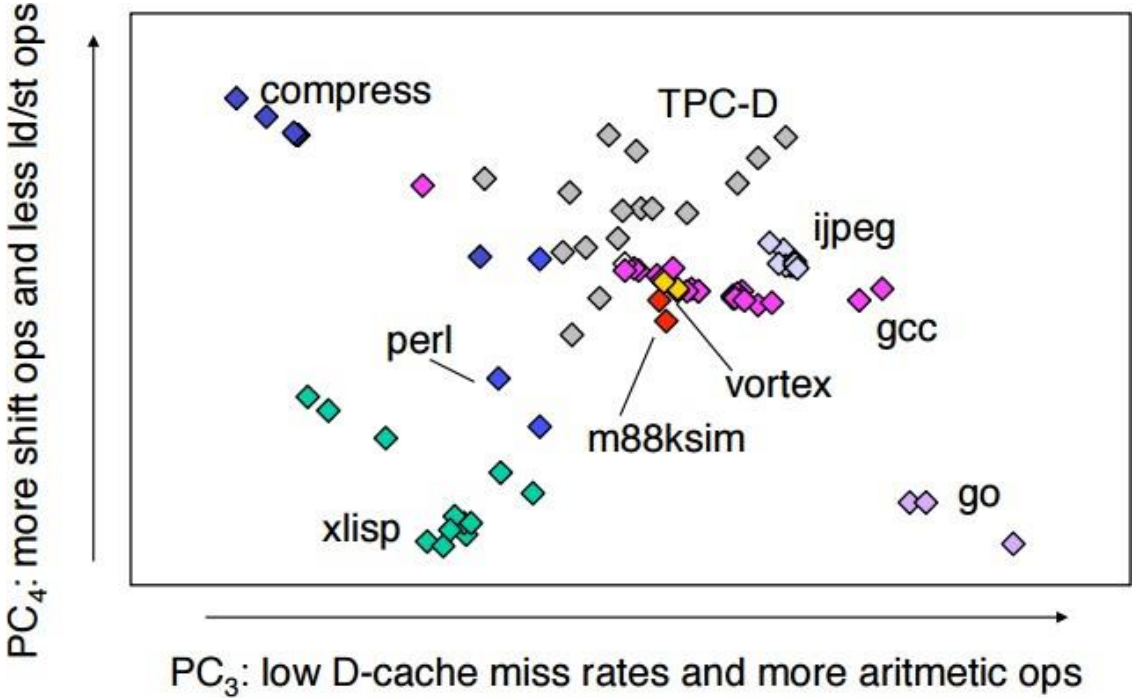
□ 构建精简但有代表性的工作负载

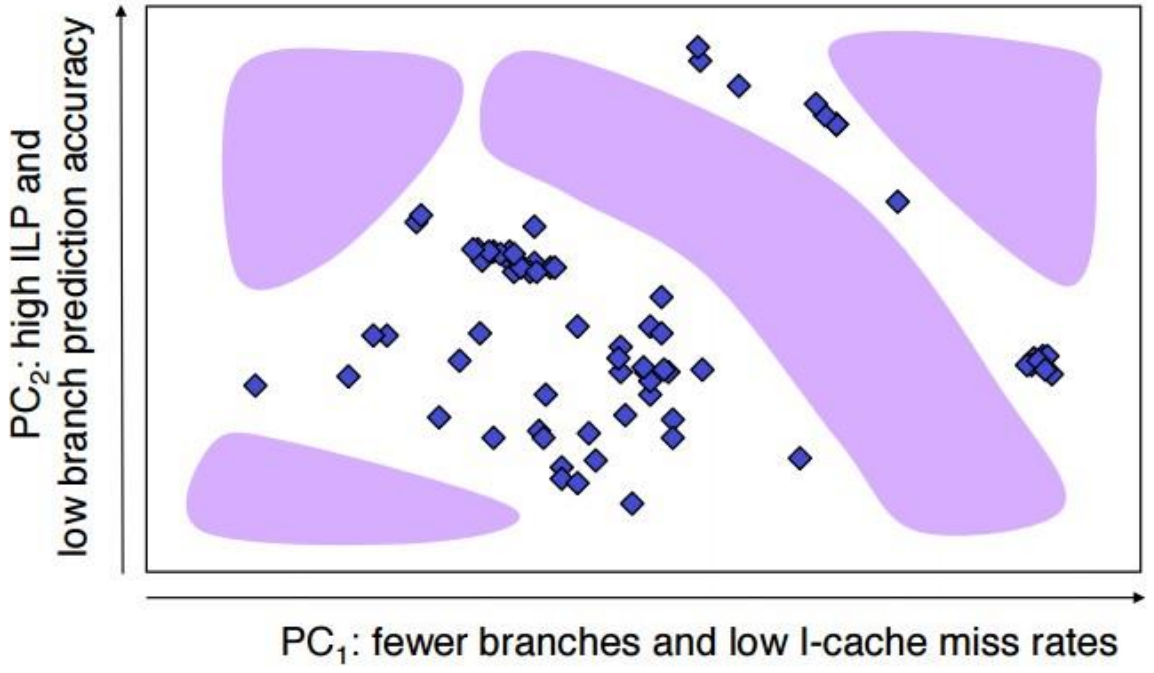
- 基于主成分分析、聚类分析等方法
- 减少需要测试的程序数量，但保留整体工作负载的代表性



■ PCA-Based设计的应用

- 工作负载分析
  - 可视化工作负载空间
  - 分析基准测试的差异
- 精简工作负载：构建代表基准测试的小型集合





■ 多程序基准测试

- 单程序基准测试相对简单
  - 由程序和一组特定输入组成
- 复杂多程序基准测试
  - 每个程序的运行时间可能不同
  - 多个程序同时运行时，程序之间互相影响
    - 更加贴近真实情况
  - 多程序基准测试的问题
    - 负载不均衡
    - 存在资源竞争

■ 多程序基准测试

- 一个多程序基准测试可以表示为一个四元组

$$\langle \mathcal{B}, \mathcal{T}, \mathcal{F}, \mathcal{S}_0 \rangle.$$

- B - 一组单程序基准测试集合
- T - 基准测试的终止条件
- F - 选择函数，当某个CPU核心空闲时，决定从B中选哪个单程序基准测试在该核心上运行，负责程序调度
- S - F的初始状态

■ 性能评估

- SPEC比率
  - 将执行时间与参考计算机进行归一化，从而比较不同计算机的性能

$$SPEC\text{比率} = \frac{\text{参考计算机执行时间}}{\text{额定执行时间}}$$

$$\frac{SPEC\text{比率}A}{SPEC\text{比率}B} = \frac{\text{执行时间}A}{\text{执行时间}B} = \frac{\text{性能}A}{\text{性能}B}$$

- 使用几何平均值计算平均值，SPEC比率是一个比值而非绝对执行时间

■ 性能评估

- 选择适当的平均值
  - 取决于评估指标的计算方法
- 调和平均值的适用场景
  - 指标可以通过A/B计算
  - A在各个基准测试中的权重相同

$$\frac{\sum_{i=1}^{n} A_i}{\sum_{i=1}^{n} B_i} = \frac{n \cdot A}{\sum_{i=1}^{n} B_i} = \frac{n}{\sum_{i=1}^{n} \frac{B_i}{A}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{A/B_i}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{A_i/B_i}} = HM(A_i/B_i)$$

- 算术平均值
  - 指标可以通过A/B计算
  - B在各个基准测试中的权重相同

$$\frac{\sum_{i=1}^{n} A_i}{\sum_{i=1}^{n} B_i} = \frac{\sum_{i=1}^{n} A_i}{n \cdot B} = \frac{1}{n} \sum_{i=1}^{n} \frac{A_i}{B} = \frac{1}{n} \sum_{i=1}^{n} \frac{A_i}{B_i} = AM(A_i/B_i)$$

■ 系统吞吐量

- 归一化周转时间 (NTT)
  - T_i^{MP}：多程序基准测试执行时间
  - T_i^{SP}：单程序基准测试执行时间

$$NTT_i = \frac{T_i^{MP}}{T_i^{SP}}$$

- 平均归一化周转时间 (ANTT)

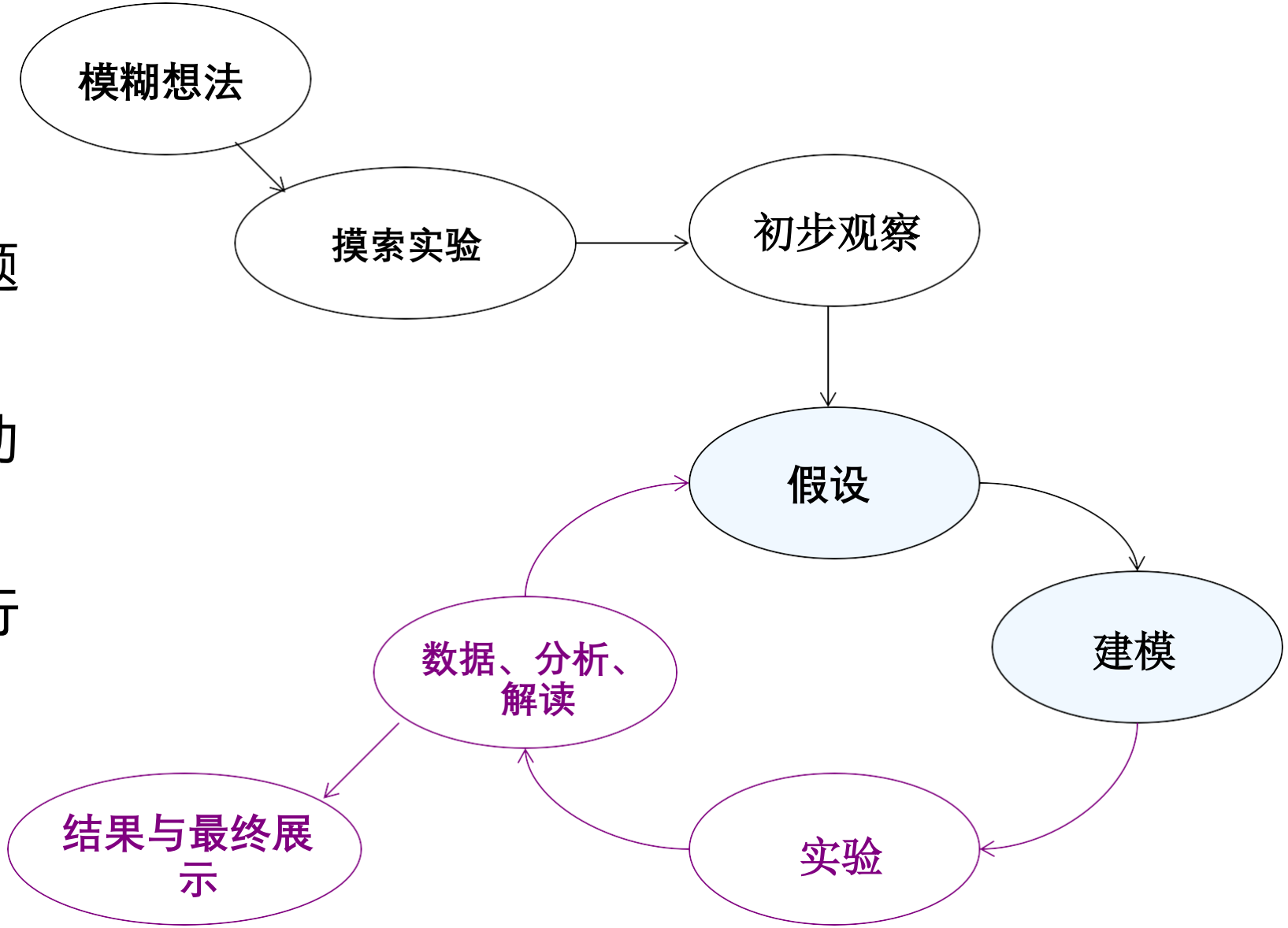
$$ANTT = \frac{1}{n}\sum NTT_i$$

- 每周期指令数(IPC)吞吐量

$$IPC\text{吞吐量} = \frac{1}{n}\sum IPC_i$$

■ 实验的生命周期

1. 理解问题、明确问题框架与目标
2. 确定哪些要素能帮助回答问题
3. 识别影响研究对象行为的指标



■ 练习

□ 例题

- 假设有一个4GHz的单核处理器，执行一个标准测试程序。程序包含的指令类型如下表所示，问平均CPI是多少？
  - CPI=Clock per instruction
- 缓存的平均访问周期数是多少？
- 若只有算术和逻辑部分可以充分并行，用双核处理器的加速比？四核？极限加速比是多少？

指令	CPI	指令占比	指令	CPI	指令占比
算术和逻辑	1	60%	跳转	4	12%
缓存命中时访存	2	18%	缓存失效时访存	8	10%

■ 练习

- 程序包含的指令类型如下表所示，问平均CPI是多少
  - 答：平均CPI为2.24
  - $1 \times 60\% + 4 \times 12\% + 2 \times 18\% + 8 \times 10\% = 2.24$

指令	CPI	指令占比	指令	CPI	指令占比
算术和逻辑	1	60%	跳转	4	12%
缓存命中时访存	2	18%	缓存失效时访存	8	10%

■ 练习

- 缓存的平均访问周期数是多少?
  - 答：平均访问周期数为4.14
  - 这道题缓存命中率为$18/(10+18) \approx 64.29\%$
    - 有时题目直接提供命中率
  - 由此可知平均访问周期$2 \times 64.29\% + 8 \times 35.71\% = 4.1426$
    - 值得注意的是，缓存失效时访存已经包含失效惩罚。注意审题，给出失效惩罚还是失效访存时间
    - 利用课上提到的公式计算应为$2 + 6 * 35.71\% = 4.1426$，其中6是失效惩罚

指令	CPI	指令占比	指令	CPI	指令占比
算术和逻辑	1	60%	跳转	4	12%
缓存命中时访存	2	18%	缓存失效时访存	8	10%

■ 练习

- 若只有算术和逻辑部分可以充分并行，用双核处理器的加速比？四核？极限加速比是多少？
  - 答：双核加速比1.43，四核加速比1.81，极限加速比为2.5
  - 利用Amdahl定律$\frac{1}{1-+\frac{f}{n}}$，其中$f = 0.6$，n用$2,4$代入计算可以得到结果

指令	CPI	指令占比	指令	CPI	指令占比
算术和逻辑	1	60%	跳转	4	12%
缓存命中时访存	2	18%	缓存失效时访存	8	10%

感谢！