北京大学计算机学院本科生课程

计算机组成与系统结构实习 Review 5

北京大学微处理器研发中心

易江芳

2022-10-31

主要内容

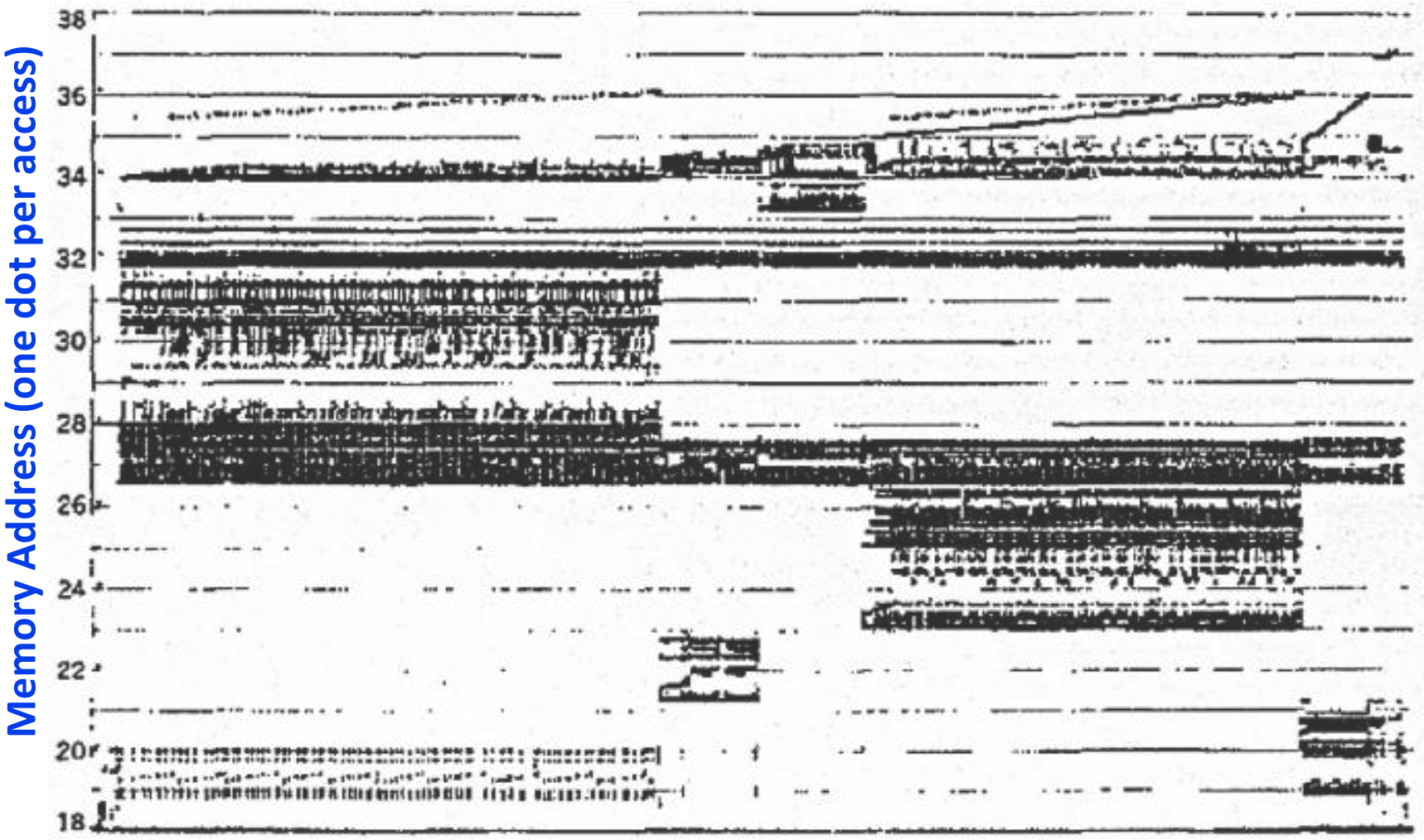
- 高速缓存（Cache）
  - 基本原理
  - 基本概念
  - 性能公式
  - 替换策略

Cache的基本原理

- 层次化的存储结构
  - 离cpu越近，速度越快
  - 离cpu越远，容量越大

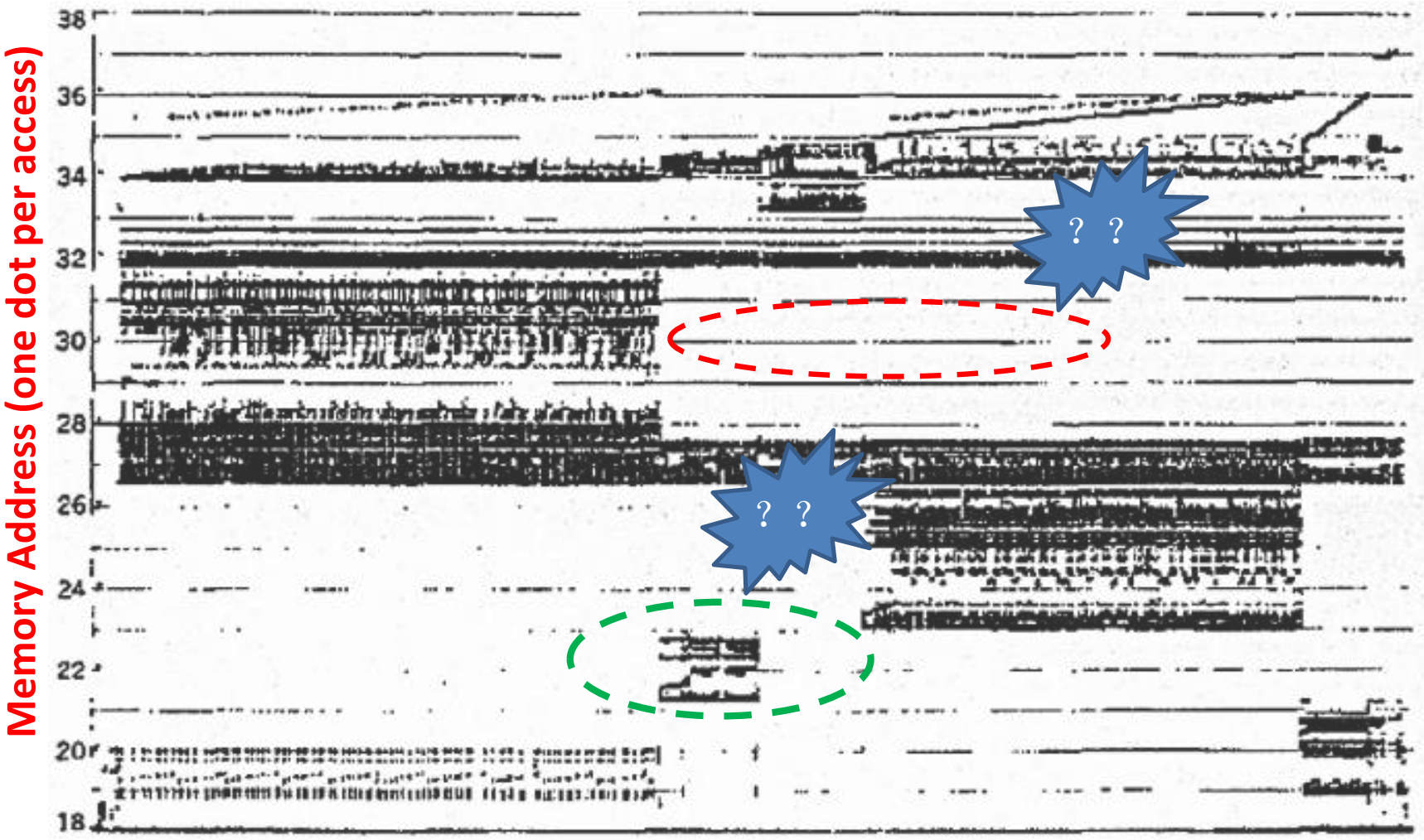
- 局部性原理
  - 时间局部性
  - 空间局部性

真实的访存模式



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

时间局部性 和 空间局部性

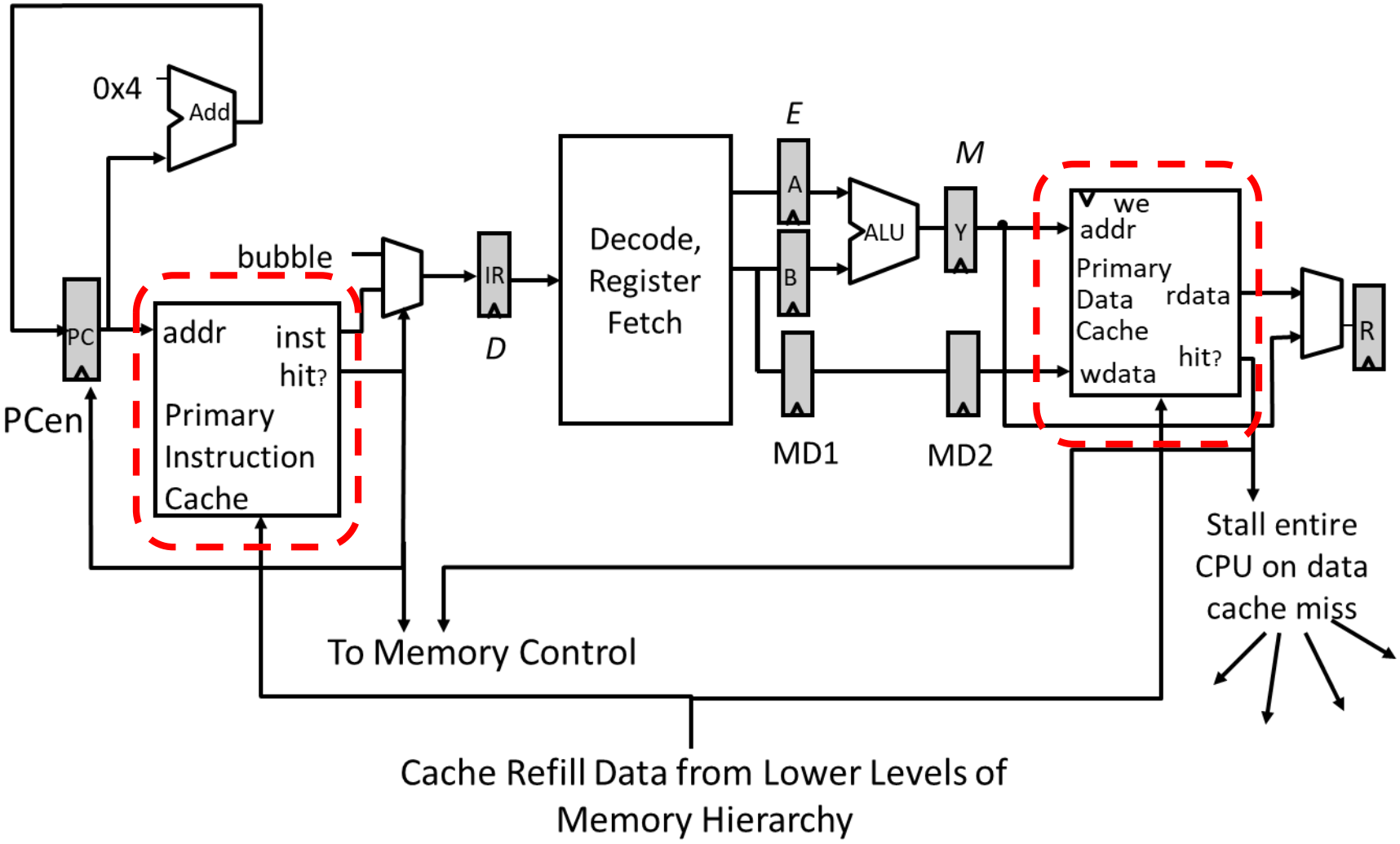


Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

基本概念

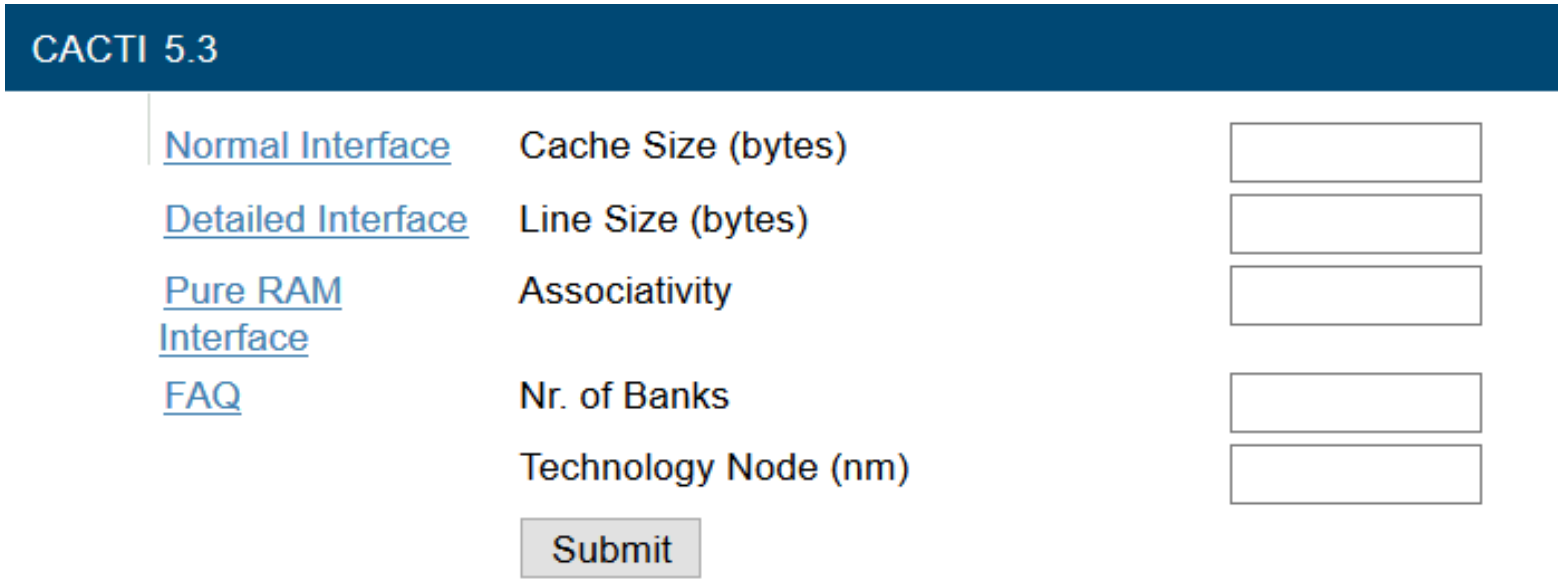
- 组（set），路（way），块/行（block/line）
  - 例如，四路组相联
  - V = set * way * blocksize
- 组相联/ 直接映射/ 全相联
- 命中hit/ 失效miss，命中率=1- 失效率
- 写策略
  - 写命中：写回（write back）和写穿透（write through）
  - 写失效：写分配（write-allocate ）or 写不分配（no-write-allocate）
  - 一般组合：write through+no-write-allocate，write-back+ write-allocate
- 三种失效（3C）
  - 义务Compulsory/ 容量Capacity / 冲突Conflict
- 替换
  - 替换策略：随机，FIFO，LRU等

CPU-Cache 相连（以5级流水线为例）



CACTI: 开源的Cache模型

https://www.hpl.hp.com/research/cacti/



CACTI 肯定不是最准确的，但课程或模拟应该够了 ☺

性能公式

- Average memory access time (AMAT) =
  Hit time + Miss rate x Miss penalty

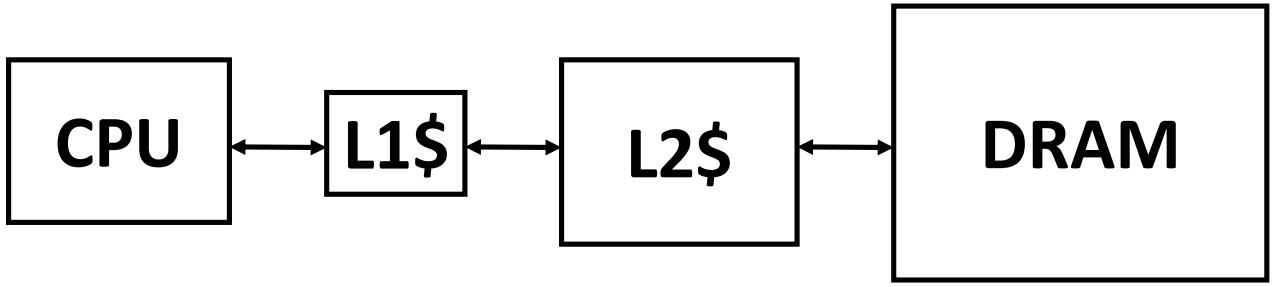
- 如何改善Cache性能
  - 减少 Hit time
  - 减少 Miss rate
  - 减少 Miss penalty

- 即使最大容量的Cache，也要使得L1$ 命中时间在1-3周期内

参数对Cache性能的影响

- 容量越大
  + reduces capacity and conflict misses
  - hit time will increase
- 相联度越高
  + reduces conflict misses
  - may increase hit time
- 块大小越大
  + reduces compulsory and capacity (reload) misses
  - increases conflict misses and miss penalty

存储层次的性能指标



- Average memory access time (AMAT) = Hit time + **Miss rate** x Miss penalty

- Local miss rate = misses in cache / accesses to cache
- Global miss rate = misses in cache / all accesses

- Misses per instruction = misses in cache / number of instructions

示例

- 假设：L1$ 命中时间 2 cycles，local miss rate 20%；L2$ 命中时间 15 cycles, global miss rate 5%；访问主存时间 100 cycles.

- Q1: L2$ 的local miss rate?
  - Local miss rate = 5% / 20% = 0.25 = 25%
- Q2：该存储系统的AMAT?
  - AMAT=2+20%x15+5%x100=10 (using global miss rates)
  - 或 AMAT = 2+20%x(15+25%x100)=10
- Q3：想通过增加L3$来降低AMAT到8 cycles或更低，已知L3$的 local miss rate为30%，L3$的命中时间最大为多少?
  - 2+20%x(15+25%x(H+30%x100)) ≤8 ，计算后可得H≤30.

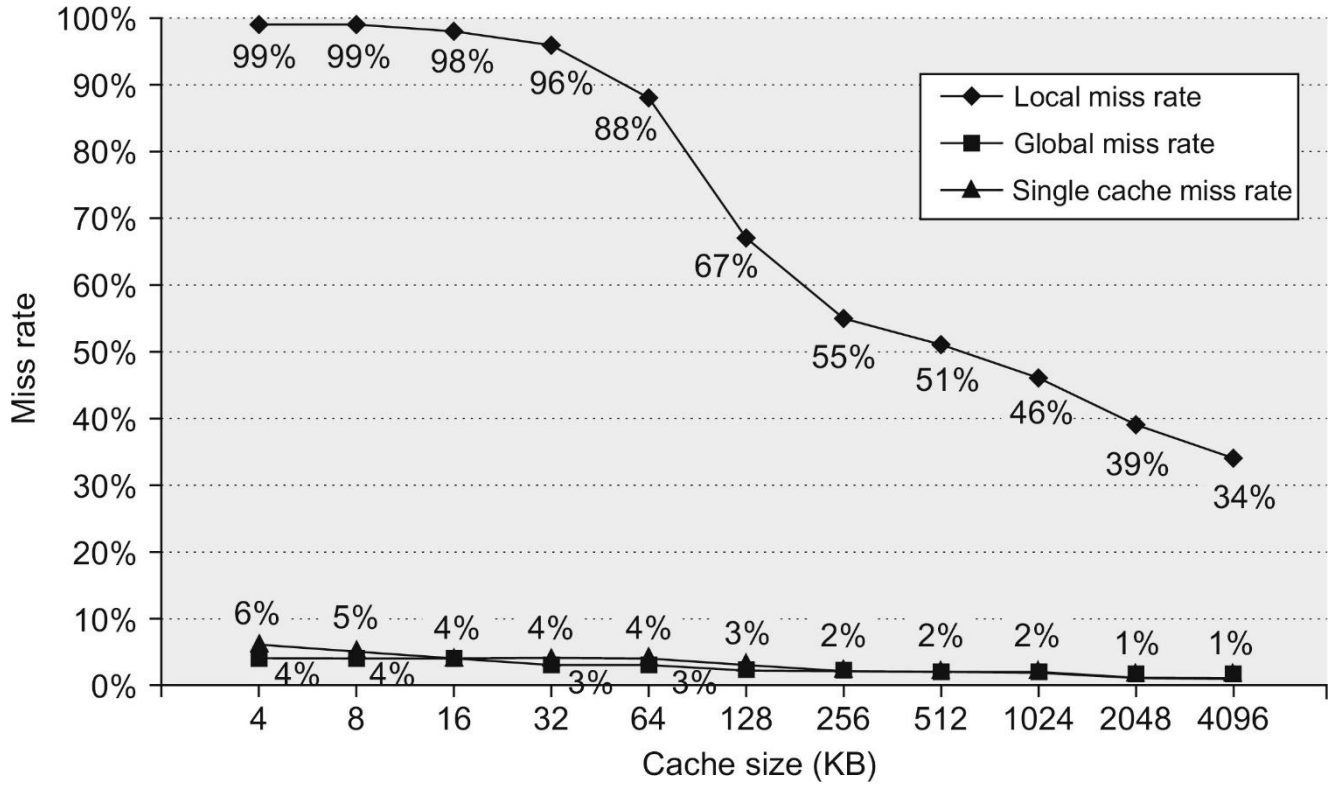


Figure B.14 Miss rates versus cache size for multilevel caches. Second-level caches *smaller* than the sum of the two 64 KiB first-level caches make little sense, as reflected in the high miss rates. After 256 KiB the single cache is within 10% of the global miss rates. The miss rate of a single-level cache versus size is plotted against the local miss rate and global miss rate of a second-level cache using a 32 KiB first-level cache. The L2 caches (unified) were two-way set associative with replacement. Each had split L1 instruction and data caches that were 64 KiB two-way set associative with LRU replacement. The block size for both L1 and L2 caches was 64 bytes. Data were collected as in Figure B.4.

L2 Cache对L1 Cache的影响

- 如果存在 L2$，则 L1$ 容量可以小一些
  - 可以减少 L1$ 的命中时间
  - 可以减少 L1$ 的失效代价
  - 可以减少存储访问的平均能耗

- 如果存在 L2$，L1$ 可以使用简单的write through策略
  - L2$ 采用write back策略，可以让写操作尽量在芯片内部
  - 没有脏数据回写，简化流水线控制
  - 简化一致性协议处理
  - 简化L1$ 中的错误恢复机制，校验码可从L2$ 中获得

Inclusion Policy

- L2$ 和 L1$ 的关系

- Inclusive 多级Cache
  - 下一级Cache包含上一级Cache的数据
  - 一致性协议只需要检查下一级Cache

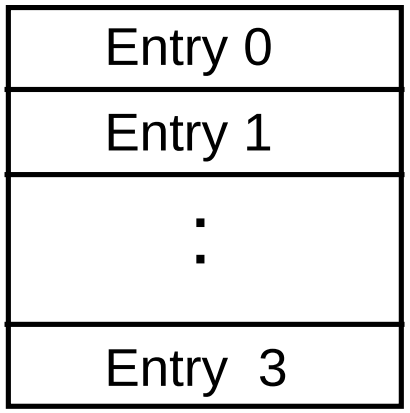
- Exclusive 多级Cache
  - 上一级Cache包含下一级Cache 没有的数据
  - 发生miss时，两级Cache中的数据进行交换
  - AMD的Athlon中使用，64KB L1$，256KB L2$

替换策略

- 常用替换策略
  - 随机替换
  - LRU替换
  - FIFO
- 其他替换策略
  - LRU (Least Recently Used), pseudo
  - LFU (Least Frequently Used)
  - NMRU (Not Most Recently Used)
  - NRU（Not Recently Used）
  - Pseudo
  - Optimal
  - …

简单的例子

- 全相联Cache，共有4个 Block

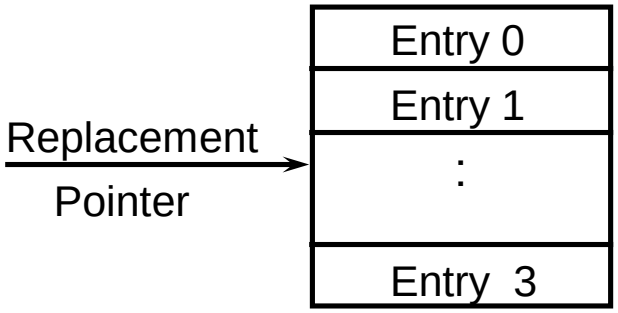


- 访问序列如下：3 2 3 1 2 0 3 1
- 如果使用 LRU, 则下一个被替换的是 Block 2
- 如果使用 LFU, 则下一个被替换的是 Block 0

LRU的不足

- 对于streaming类的访问序列：X0, X1,…,Xn
  - 没有时间上的重用，LRU无法挖掘有效信息
- 重用距离（reuse distance）
  - 连续两次访问相同地址的时间间隔
- Cache thrashing（颠簸）：a[i] = b[i] + c[i]
  - 重用距离 > a（相联度）
  - 相当于程序无缓存的使用存储
  - 原因是不合理的向量对齐（vector alignment），所有的向量都映射到了相同的cache位置
  - 解决方法：改变数据排布，不对齐；改变替换算法
- Pseudo-LRU 可能会有点帮助

一个简单的伪LRU实现



- 接上例，含有4个表项的全相联Cache
- 使用指针指向下一个要被替换的表项
- 每处理完毕一次Cache访问，指针都指向下一个表项
- 全相联Cache没有冲突失效
- 如果发生冲突失效，如何实现伪LRU算法?

基于二叉树的伪LRU实现

- 每个cache set有一个N位的标识，称为 PLRU 位，N= #way -1
- 初始时，所有 PLRU位 被设置为0。例如，b0b1b2=000
- 如果set中任一数据块无效，则替换它；否则根据b0的取值选择b1或者b2
- ……
- 根据算法替换相应数据块

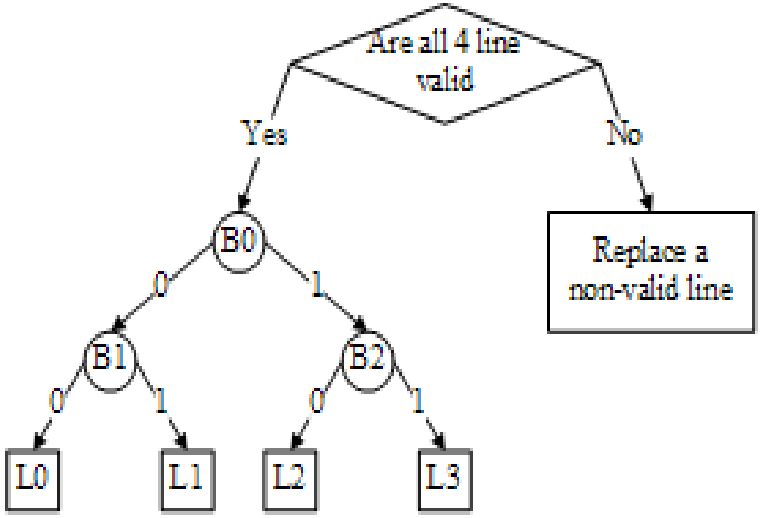


图 2-15 Tree-Based PLRU 替换算法

更新PLRU

- 更新时机：
  - 替换 cache行时，更新PLRU位
  - Cache行命中时，更新PLRU位
- 更新规则如下表所示：

Current Access	New State of the PLRU Bits		
	B0	B1	B2
L0	1	1	No Change
L1	1	0	No Change
L2	0	No Change	1
L3	0	No Change	0

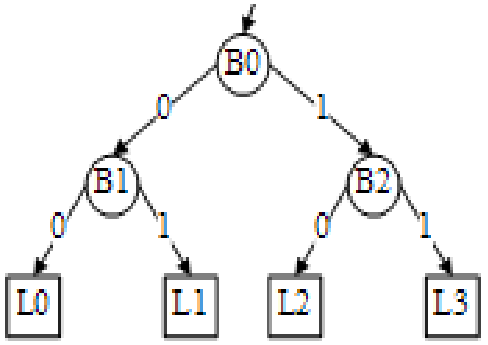
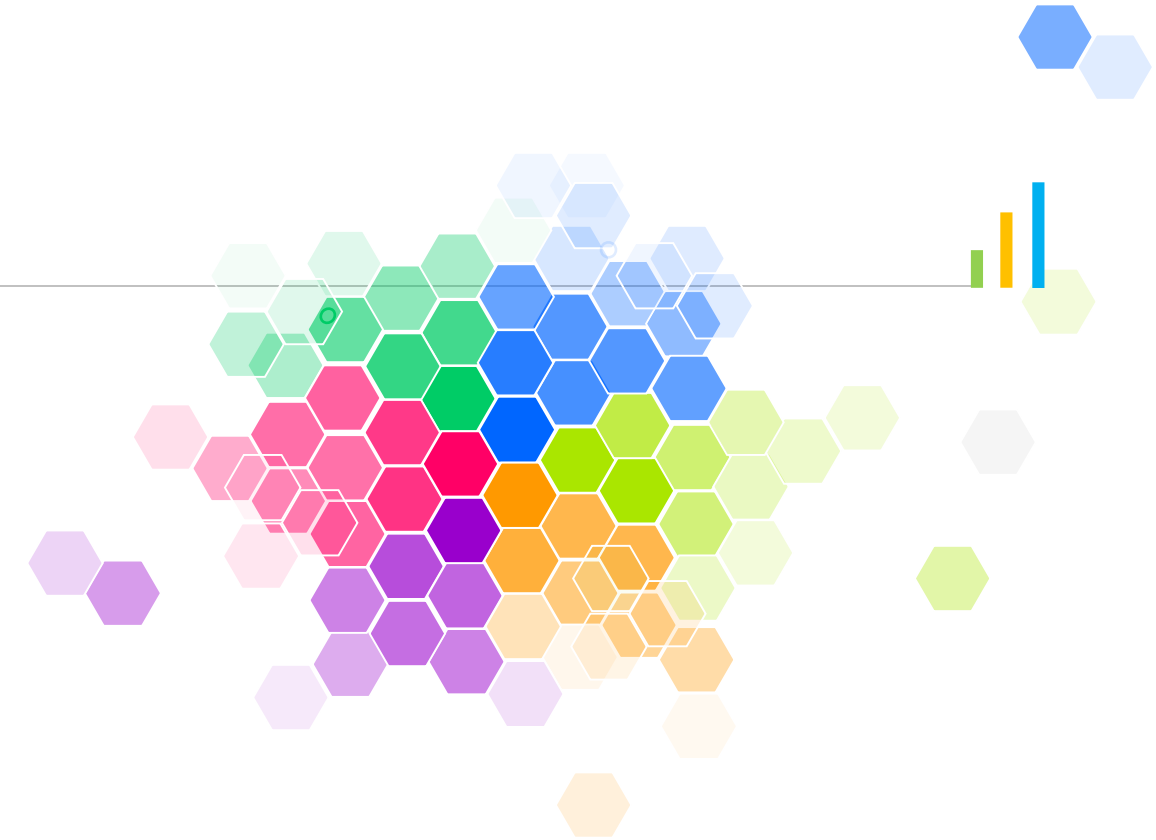


图 2-15 Tree-Based PLRU 替

假设访问序列如下：
3 2 3 1 2 0 3 1

小结

- Cache是层次化存储的一部分
- 利用了程序的局部性原理
- Cache容量：V = set * way * blocksize
- Average memory access time (AMAT) =
  Hit time + Miss rate x Miss penalty
- 层次化Cache可以有效的改善失效代价
- 对于Cache内容的管理：替换策略
  - 本质上是计算数据的重用距离
  - 不同的替换策略是对重用距离的不同使用方式



欢迎提问