存储器的层次结构

一. 为什么存储器要多层？ ① CPU 处理器性能和存储系统性能之间存在巨大差异（存储墙 Memory Wall）。访存时间太长，严重制约计算机性能提升。

② 系统应用规模不断扩大，需要更大存储空间支撑程序运行。

③ 容量、速度、价格不可兼得

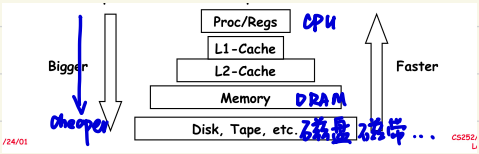
SRAM 速度快 管多、体积大、价格高

DRAM（主存主要用） 速度太慢

磁盘 速度慢

单种不可能 → 利用多种存储器件，取长补短，层次开

二. 存储结构



Cache 在 CPU 和主存之间

尽量让 CPU 多访问 Cache

时间局部性 当前指令不久会用，放入 Cache

空间局部性 相邻指令会用，放入 Cache

现在 CPU 中 Cache 占很大面积

Cache - 主存层次 弥补主存速度不足 速度 ns 比 - CPU 可直接访问 Cache 但不直接访问主存/辅存

主存 - 辅存层次 弥补主存容量不足 速度 ms 比 --

三. 设计存储结构时需要考虑的四个问题

Review: Four Questions for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level? *(Block placement)*
  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level? *(Block identification)*
  - Tag/Block
- Q3: Which block should be replaced on a miss? *(Block replacement)*
  - Random, LRU
- Q4: What happens on a write? *(Write strategy)*
  - Write Back or Write Through (with Write Buffer)

1/24/01 CS252/Kubiatowicz Lec 3.12

就是能写的 RISC-V 中的 tag

1. 当把一个块调入高一层(靠近CPU)存储器时，可以放在哪些位置上？
   (映象规则 调入块可以放在哪些位置) 从主存到 Cache
2. 当所要访问的块在高一层存储器中时，如何找到该块？
   (查找算法 如何在映象规则规定的候选位置查找) CPU 查询一个块
3. 当发生失效时，应替换哪一块？ 块不在高一层中，将块调入 Cache
   (替换算法 规定的候选位置均被别的块占用) 但位置都被占用
4. 当进行写访问时，应进行哪些操作？
   (写策略 如何处理写操作)

性能分析 Cache Performance

Cache平均访问时间=命中率+失效率×失效开销

$$AMAT = HitTime + MissRate \times MissPenalty$$
$$= (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}) + (HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data})$$

（但有可能Cache会影响CPU，所以最终最准确为CPU时间）

'01

- Miss-oriented Approach to Memory Access:

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty \right) \times CycleTime$$

IC：指令数；$CPI_{Execution}$：CPU执行周期数（←命中的算在执行周期里）；MemAccess：访存次数；Inst：指令数 =IC；MissRate：失效率；MissPenalty：失效开销；CycleTime：时钟周期时间

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemMisses}{Inst} \times MissPenalty \right) \times CycleTime$$

MemMisses：访存失效次数

  - $CPI_{Execution}$ includes ALU and Memory instructions

- Separating out Memory component entirely 将存储分开算
  - AMAT = Average Memory Access Time
  - CPI_{ALUOps} does not include memory instructions

IC：指令条数

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

CPI_{AluOps}：CPU运行时间；CycleTime：时钟周期时间

$$AMAT = HitTime + MissRate \times MissPenalty$$

HitTime：命中率；MissRate：失效率；MissPenalty：失效开销

Cache平均访问时间 $= (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}) + (HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data})$

1/24/01　　CS252/Kubiatowicz Lec 3.9

提高Cache性能的三种方法

Review: Improving Cache Performance

1. Reduce the miss rate, — 降低Cache失效率的方法
2. Reduce the miss penalty, or — 减少Cache失效开销
3. Reduce the time to hit in the cache. — 减少命中时间

映象规则：主存的块放到Cache的哪些位置

1、全相联映象：主存中的任一块可以被放置到Cache的任一位置。 实现复杂，利用稿
2、直接映象：主存中的每一块只能被放置到Cache中唯一的一个位置 实现简单利用额
　　例 主存中第i位，映象到Cache中，$j = i \bmod M$（Cache大小）
3、组相联：主存中的每一块可以被放置到Cache中唯一一个组的任意中的任何一个位置。

1. Reduce the miss rate

(1) reasons: 三种失效(3C)

Reducing Misses

Classifying Misses: 3 Cs

- *Compulsory*—The first access to a block is not in the cache, so the block must be brought into the cache. Also called *cold start misses* or *first reference misses*.
  (Misses in even an Infinite Cache)
- *Capacity*—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
  (Misses in Fully Associative Size X Cache)
- *Conflict*—If block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called *collision misses* or *interference misses*.
  (Misses in N-way Associative, Size X Cache)

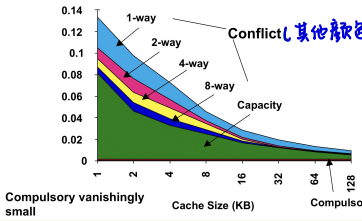
More recent, 4th "C":

- *Coherence* - Misses caused by cache coherence.

CS252/Kubiatowicz Lec 3.14

(1) 强制性失效 (Compulsory miss)
当第一次访问一个块时，该块不在 Cache 中， 需 从下一级存储器中调入 Cache ，这就是强制性失效 (冷启动失效，首次访问失效)
(2) 容量失效 (Capacity miss) 如果程序执行时所需的块不能全部调入 Cache 中，则当某些块被替换后，若又重新被访问，就会发生失效。这种失效称为容量失效。
(3) 冲突失效 (Conflict miss) 在组相联或直接映象 Cache 中，若太多的块映象到同一组 (块) 中，则会出现该组中某个块被别的块替换 (即使别的组或块有空闲位置) ，然后又被重新访问的情况。这就是发生了冲突失效。(碰撞失效，干扰失效)

3Cs Absolute Miss Rate (SPEC92)



(其他颜色代表冲突失效)

(绿色代表容量失效)

(红色部分代表强制性失效)

由左图可得出结论

2. 三种失效所占的比例 (SPEC92) 图示1(绝对值)

可以看出：
(1) 相联度越高，冲突失效就越少；
(2) 强制性失效不受Cache容量的影响，但容量失效却随着容量的增加而减少；强制性失效和容量失效不受相联度的影响.
(3) 表中的数据符合2:1的Cache经验规则，即大小为N的直接映象Cache的失效率约等于大小为N/2的两路组相联Cache的失效率。

```
# 2: 1 Cache Rule
miss rate 1-way associative cache size X
= miss rate 2-way associative cache size X/2
```

由此想到方法

How Can Reduce Misses?

- 3 Cs: Compulsory, Capacity, Conflict
- In all cases, assume total cache size not changed:
- What happens if:

1. 1) Change Block Size:
   Which of 3Cs is obviously affected?

当块变大时，不命中率先减小后增大(在缓存总容量不变的前提下)

一开始不命中率减小：块变大，减少了容量不命中(空间局部性利用变多)

后来不命中率增大：块变大意味着行数(块数)减少，增加了冲突不命中的可能(时间局部性利用变少)

2. 2) Change Associativity:
   Which of 3Cs is obviously affected?

增加相关度(以增加命中时间为代价.超过8意义不大)

增加组数可以减少冲突不命中，但是会增加在组内并行搜索的时间(增加 T_{hit})和构造电路的成本

3. 3) Change Compiler: 增加容量
   Which of 3Cs is obviously affected?

构造一个又大又快的相联高速缓存很困难且很昂贵，所以一般全相联高速缓存只适合做小的高速缓存，过大的高速缓存的组数不能一味的追求大

(许多增加时效的方法会增加失效并减少命中时间!)