

I am a slow walker, but never walk backwards. 我走得很慢，但是我从来不会后退

### 第13周作业 第四章 6-16

6. 如果采用地址高位作索引，可能会映射在同一组中。如果采用中位，则会映射在不同组的缓存块中  
 e.g. 2个组，每组一行，每个块32个字节，地址8位。

则地址可分为 索引  $s = 1\text{bit}$ , 偏移  $b = 5\text{bit}$ , 标记  $t = 2\text{bit}$

设结构体  $X$  有 32 个字节，分成 2 个结构体  $X[2]$

若中位索引:  $\&X[0] = 0 = 00^t 0^s b^b 00000$  两者分别存在组 [0] 组 [1] 中

$\&X[1] = 1 = 00^t 1^s 00000$

若高位索引:  $\&X[0] = 0^s 00^t b^b 00000$  两者都会存放组 [0] 中

$\&X[1] = 0 01 00000$

7. 虚拟地址的页偏移会不变的移动到物理地址的页偏移

$$8.(1) \frac{3\%N \times 110 + 97\%N \times 1}{N} = 4.27 \text{ 周期}$$

$$1\text{GB} = 1024 \times 1024 \text{KB} = 2^{20} \text{KB}$$

$$(2) \frac{64\text{KB}}{1\text{GB}} \times 1 + \left(1 - \frac{64\text{KB}}{1\text{GB}}\right) \times 110 = \frac{1}{2^4} + 110 \times \left(1 - \frac{1}{2^4}\right) = 109.99 \text{ 周期}$$

- (3) 利用局部性原理，可以将时间、空间上有关联的数据提前加载到缓存中，大幅提高缓存命中率，如 Q 所示。另外由于缓存容量较小，若随机加载数据，则缺失率很高，基本没有提升速度的作用。

$$(4) 1 \times x\% + 110 \times (1-x\%) \leq 105 \quad x \geq 4.58\%$$

9.



录取名单尚未确定，一切伟大正在形成。

9. 地址位bit	缓存大小KB	块大小Byte	相联度	组数量	索引位数	行首位数	偏移位数
32	4	64	2	32	5	21	6
32	4	64	8	8	3	23	6
32	4	64	全	1	0	25	6
32	16	64	1	256	8	18	6
32	16	128	2	64	6	19	7
32	64	64	4	16	8	18	6
32	64	64	16	64	6	20	6
32	64	128	16	32	5	20	7

$$(1) (1-P_1) \times 0.22 + 100.22 \times P_1 \leq (1-P_2) \times 0.52 + 100.52 \times P_2$$

要求  $P_1 - P_2 \leq 0.003$

$$(2) (1-P_1) \times 0.22 + (k+1) \times 0.22 P_1 \leq (1-P_2) \times 0.52 + (k+1) \times 0.52 P_2$$

要求  $22kP_1 \leq 30 + 52kP_2$

$$11. (1) 直接计算: 1001 \% 16 = 9 ; 1005 \% 16 = 13 ; 1021 \% 16 = 13$$

进位: 0x  
1045 \% 16 = 5 ; 1305 \% 16 = 9 ; 12005 \% 16 = 5  
~~进位16位数!!!~~ 65285 \% 16 = 5 , 替换4次 — 替5次

$$(2) 2进制: 1001 \% 8 = 1 ; 1005 \% 8 = 5 ; 1021 \% 8 = 5$$

$$1045 \% 8 = 5 ; 1305 \% 8 = 1 ; 12005 \% 8 = 5$$

65285 \% 8 = 5 替换3次

$$(3) 4进制: 1001 \% 4 = 1 ; 1005 \% 4 = 1 ; 1021 \% 4 = 1$$

$$1045 \% 4 = 1 ; 1305 \% 4 = 1 ; 12005 \% 4 = 1$$

65285 \% 4 = 1 替3次



在高中学习中，与其用很多参考书，不如扎实实用一本好的参考书，充分吸收其中的营养。

——甘肃文科高考状元

(4) 设  $i = 1 \dots 100$ , 缺  $\leq 0$

12. 磁盘中共有  $156 \div 16 = 16$  个块。数据的每一位为 32bit, 4Byte,

若运行  $100 \times 96$  次

一个块 16 字节，故一个块可以在 4 个数组中分配

~~array[0~3] → 块 0      array[4~7] : 块 1      ...      array~~

若用 A：第一次想写  $array[0]$ , 缺。把它读来，在进组 0 位置。第 2, 3, 4 次写不缺

第 5 次想写  $array[4]$ , 缺。—— 在进组 1 位置。第 6, 7, 8 次写不缺

29                  28                  存进组 0 位 1      30, 31, 32

61                  60                  存进组 1 位 2      62, 63, 64

第 15 次想写  $array[64]$ , 缺。存进组 0 位 1

93                  array[92]                  存进组 1 位 1      93, 94, 95 不缺

$i=0$  时，写 96 次，缺 24 次

$i=1 \sim 100$  时，写 96 次，缺 24 次，因为 LRU 算。下次写  $array[0]$  时会存进组 0 位 2。

Miss Rate = 25%

若用 B：第一次想写  $array[0]$ , 缺。存进组 0。第 2, 3, 4 次不缺

61                  array[60]                  存进组 15.      62, 63, 64

93                  12                  7

$i=0$  时，写 96 次，缺 24 次

$i=1 \sim 100$  时，写 96 次，缺 8 次      Miss Rate =  $\frac{24 + 99 \times 8}{96 \times 100} = 16.75\%$



射门，球不进的几率是50%；不射门，球不进的几率是100%

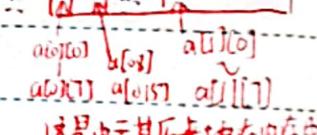
```
B. for (int i=0; i<128; i++)
{
    for (int j=0; j<64; j++)
    {
        A[i][j] = A[i][j]+1;
    }
}
```

14(1) 整型数组的一个元素为4字节，一块里可以存8个元素。 $4K \div 32B = 128$ 个块

优化前： $i=0, j=0 \sim 128$ 时，每次读时缺，写时不缺

因为直接映射

128个块



这是由于其未块在内存中

② 下次读写  $a[0 \sim 127][1 \sim 7]$  时，都缺

13次

接下来7组也一样。

一共缺  $128 \times 8 = 1024$  次

的地址冲突

优化后：相当于有几块就缺几次：一共缺  $8 \times 128 = 1024$  次

优化

$a[0][0] \sim a[0][1]$	$a[0][8] \sim a[0][15]$	...	$a[0][56] \sim a[0][63]$
$a[1][0] \sim a[1][7]$			
...			
$a[127][0] \sim a[127][7]$			

(2) 优化：1024次

优化：1024次

(3) 必要的：换块时带来的缺失

优化：1块即可  $\rightarrow 32B$

优化：4KB 16KB



把脸迎向阳光，就不会有阴影。

15. 写分配：写缺失时缓存会分配出一个块用于向其中写入数据，导致块替换。

$a[i][j]$	Input				Output			
110	311	312	313		310	311	312	313
111	miss	m	h	m	miss	m	m	m
112	m	h	m	h	m	m	m	m
113	m	m	h	m	m	m	m	m
114	m	h	m	h	m	m	m	m

16. int input[2][128], sum=0;

一个块可存4个数组中的元素

for (int i=0; i<128; ++i)

16组，每组2个位置

{   sum = sum + input[0][i] \* input[1][i]}

LRU策略

(1) $in[0][0] \sim in[0][3]$	$in[0][4] \sim in[0][7]$	...	$in[0][24] \sim in[0][27]$
$in[1][0] \sim in[1][3]$		...	$in[1][24] \sim in[1][27]$

组0  $in[0][0] \sim in[0][3]$        $in[0][4] \sim in[0][7]$        $in[1][0] \sim in[1][3]$        $in[1][4] \sim in[1][7]$

组1  $in[0][4] \sim in[0][7]$

...

组15  $in[0][60] \sim in[0][63]$        $in[0][124] \sim in[0][127]$        $in[0][64] \sim in[0][67]$        $in[1][64] \sim in[1][67]$

一共读  $128 \times 2 = 256$  次。读  $in[0][0], in[1][0]$  都缺。剩下读  $in[0][1 \sim 3]$  都不缺。

以后每次读也类似。8次读新 2次缺 命中率 = 75%



泪水和汗水的化学成分相似，但前者只能为你换来同情，后者却可以为你赢得成功。

- (2) 若增加 Cache 大小 可以假设现在变成了 32 组，每组 2 个位置，命中率不变  
(3) 假设块大小变为 32 个字节，每块可装 8 个元素  $512 \div 32 = 16$ ，8 组，每组 2 个位置

组 0:  $m[0][0] - m[0][7]$        $m[0][8] - m[0][15]$        $m[1][0] - m[1][7]$

组 7:  $m[0][64] - m[0][71]$        $m[0][72] - m[0][79]$        $m[1][64] - m[1][71]$

可以提高命中率

