

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

eg. 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 $s=16\text{bit}$, 偏移 $=16\text{bit}$, 标记 $=2\text{bit}$

设结构体 X 有32个字节, 创建结构体 $X[2]$

若中位索引: $\&X[0] = 0 = \overset{t}{00} \overset{s}{0} \overset{b}{00000}$

两者分别存在组[0] 组[1]中

$\&X[1] = 1 = 00 \ 1 \ 00000$

若高位索引: $\&X[0] = \overset{s}{0} \overset{t}{00} \overset{b}{00000}$

两者都会存放在组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

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

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

$$(2) \frac{64\text{KB}}{1\text{GB}} \times 1 + (1 - \frac{64\text{KB}}{1\text{GB}}) \times 110 = \frac{1}{2^{14}} + 110 \times (1 - \frac{1}{2^{14}}) = 109.99 \text{ 周期}$$

(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, 如图A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。

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

$$x \geq 4.58\%$$

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	bit 索引位数	bit 标签位数	bit 编程位数
32	4	64	2	32	5	21	6
32	4	64	8	8	3	23	6
32	4	64	全	1	0	26	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

10 (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) 直接寻址: $1001 \% 16 = 9$; $1005 \% 16 = 13$; $1021 \% 16 = 13$
 注意: 0x... $1045 \% 16 = 5$; $1305 \% 16 = 9$; $12005 \% 16 = 5$
 就是16位数!!! $65285 \% 16 = 5$; 替换4次——替5次

(2) 2路: $1001 \% 8 = 1$; $1005 \% 8 = 5$; $1021 \% 8 = 5$
 $1045 \% 8 = 5$; $1305 \% 8 = 1$; $12005 \% 8 = 5$
 $65285 \% 8 = 5$; 替换3次

(3) 4路: $1001 \% 4 = 1$; $1005 \% 4 = 1$; $1021 \% 4 = 1$
 $1045 \% 4 = 1$; $1305 \% 4 = 1$; $12005 \% 4 = 1$
 $65285 \% 4 = 1$; 替换3次



在高中学习中，与其用很多参考书，不如扎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) 读: $100 \times 2 = 1 \dots$ 的 51, 替换 0 以

12. 缓存中共有 $156 \div 16 = 16$ 个块。数组的每一位为 32 bit, 4 byte,
共运行 100×96 次

一个块 16 字节, 故一个块可以存 4 个数组中的数据

$\text{array}[0-3] \rightarrow$ 块 0 $\text{array}[4-7] \rightarrow$ 块 1 \dots array

若用 A: 第一次想写 $\text{array}[0]$, 缺。把它读来, 存在组 0 位置 1。第 2, 3, 4 次写, 不会缺。
第五次想写 $\text{array}[4]$, 缺。—— 存在组 1 位置 1, 第 6, 7, 8 次写, 不缺。

29 28 组 7 位 1 30 31 32

61 60 组 7 位 2 62 63 64

第 15 次把写 $\text{array}[64]$ 缺。

存在组 0 位 1

93 $\text{array}[92]$

存在组 7 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 策。下次写 $\text{array}[0]$ 时会存在组 0 位 2。

Miss Rate = 25%

若用 B: 第一次想写 $\text{array}[0]$, 缺。存在组 0 第 2, 3, 4 次不缺。

61 $\text{array}[60]$ 存在组 15 62, 63, 64

93 92 7

$i=0$ 时, 写 96 次, 缺 24 次。

$i=1 \sim 100$ 时, 写 96 次, 缺 8 次。

Miss Rate = $\frac{24 + 99 \times 86}{96 \times 100} = 16.75\%$



射门，球不进的几率是50%；不射门，球不进的几率是100%

```

13. 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。一个块里可以存8个元素。 $4k \div 32B = 128$ 个块

优化前: $i=0, j=0 \sim 127$ 时, 每次读时缺, 写时不缺

① 只读写 $a[0 \sim 127][1 \sim 7]$ 时, 都不缺

接下来7组也一样。

一共缺 $128 \times 8 = 1024$ 次

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

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

因为直接映射

128个块

$a[0][0] \quad a[0][8] \quad a[0][56]$
 $a[0][1] \quad a[0][9] \quad a[0][57]$
 $a[0][2] \quad a[0][10] \quad a[0][58]$
 $a[0][3] \quad a[0][11] \quad a[0][59]$
 $a[0][4] \quad a[0][12] \quad a[0][60]$
 $a[0][5] \quad a[0][13] \quad a[0][61]$
 $a[0][6] \quad a[0][14] \quad a[0][62]$
 $a[0][7] \quad a[0][15] \quad a[0][63]$

这是由于其后来块在内存中的地址是连续的

(2) 优化前: 1024次

优化后: 1024次

(3) 必要的: 换块时带来的缺失

优化后: 1个块即可 $\rightarrow 32B$

优化前: ~~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

$a[i][i]$	input				output			
得到	310	311	312	313	310	311	312	313
310	miss	m	h	m	miss	m	m	m
311	m	h	m	h	m	m	m	m
312	m	m	h	m	m	m	m	m
313	m	h	m	h	m	m	m	m

16. $\text{int input}[2][128], \text{sum}=0;$ $\text{for}(\text{int } i=0; i<128; i++)$ $\{ \text{sum} = \text{sum} + \text{input}[0][i] \times \text{input}[1][i]; \}$

一个块可存4个数组中的元素

16组, 每组2个位置

LRU策略

(1) $\text{in}[0][0] \sim \text{in}[0][3] \quad \text{in}[0][4] \sim \text{in}[0][7] \quad \dots \quad \text{in}[0][124] \sim \text{in}[0][127]$ $\text{in}[1][0] \sim \text{in}[1][3] \quad \dots \quad \text{in}[1][124] \sim \text{in}[1][127]$ 组0 $\text{in}[0][0] \sim \text{in}[0][3] \quad \text{in}[0][4] \sim \text{in}[0][7] \quad \text{in}[1][0] \sim \text{in}[1][3] \quad \text{in}[1][4] \sim \text{in}[1][7]$ 组1 $\text{in}[0][4] \sim \text{in}[0][7] \quad \dots$ 组15 $\text{in}[0][60] \sim \text{in}[0][63] \quad \text{in}[0][64] \sim \text{in}[0][67] \quad \text{in}[0][68] \sim \text{in}[0][71] \quad \text{in}[0][72] \sim \text{in}[0][75]$ 一共读 $128 \times 2 = 256$ 次, 读 $\text{in}[0][0], \text{in}[0][4]$ 都缺, 剩下读 $\text{in}[0][1] \sim \text{in}[0][3]$ 都不缺

以后每4次读自也类似, 8次读新 2次缺 命中率 = 75%



让每

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 32 组，每组 2 个位置，命中率不变

(3) 假设把块大小变为 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 $in[0][0] - in[0][7]$

$in[0][8] - in[0][15]$

$in[1][0] - in[1][7]$

组 7 $in[0][56] - in[0][63]$

$in[0][128] - in[0][135]$

$in[1][56] - in[1][63]$

可以提高命中率

