

6. 这种设计是为了提高缓存的效率和性能。

采用高位作为标签的目的在于可以保证每个数据块在缓存中的唯一性，高位包含了更多的位数，因此可以提供更大标签空间，避免了不同数据块之间的标签冲突。标签冲突会导致数据块被错误替换或检索，降低了缓存的性能。

将中间位作为组索引的这样是为了提高缓存命中率，组索引决定了数据块在缓存中组的位置，而缓存的每个组内有3个数据块，通过使用中间位作为组索引，可以增加组的数量，从而提高缓存的容量和并发度。

7. 可以确保相同的地址在物理内存、虚拟内存之间进行一致的映射，获取地址时不需进行额外的地址转换。

8. 1). $97\% \times 1 \times 110 \times 3\% = 0.97 + \frac{3.3}{35} = 4.27$

2). L1大小为64 KB，则平均缓存命中率 $\frac{1}{214}$
 $1 \times \frac{1}{214} + 110 \times (1 - \frac{1}{214}) = 110 - \frac{109}{214} \approx 110$

3). 由于局部性原理，程序并不是随机访问所有数据，而是集中于访问缓存中的数据。

4). $1 \times \lambda + (1 - \lambda) \times 110 < 105$

$\Rightarrow \lambda > \frac{5}{109}$

编号	地址位数 Bit	缓存大小 KB	块大小 Byte	相联度	组数量	组索引位数 Bit	标签位数 Bit
9. 1	32	4	64	2	2 ⁵	5	21 6
	32	4	64	8	2 ³	3	23 6
	32	4	64	全	1	0	26 6
	32	16	64	1	2 ⁸	8	18 6
	32	16	128	2	2 ⁶	6	19 7
	32	64	64	4	2 ⁸	8	18 6
	32	64	64	16	2 ⁶	6	20 6
	32	64	128	16	2 ⁵	5	20 7

10. (1) $t_A = 0.22 \times (1 - p_1) + 100p_1 + 0.22p_1$
 $t_B = 0.52 \times (1 - p_2) + 100p_2 + 0.52p_2$
 $t_A < t_B \Rightarrow 99. \quad p_1 - p_2 < \frac{3 \times 10^4}{100} \times 0.003$

(2) $t'_A = 0.22 \times (1 - p_1) + kp_1 \times 0.22$
 $t'_B = 0.52 \times (1 - p_2) + kp_2 \times 0.52$
 $t'_A < t'_B \quad 0.22p_1 - 0.52p_2 < \frac{1}{k-1} \times 0.3$

11. 直接映射 $0x1001 \rightarrow 0x1021$

5次块替换 $0x1005 \rightarrow 0x1045 \rightarrow 0x1305 \rightarrow 0x2005 \rightarrow 0x4005$

2路组相联 $\left\{ \begin{array}{l} 0x1001 \\ 0x1021 \end{array} \right\} \left\{ \begin{array}{l} 0x1005 \\ 0x1045 \end{array} \right\} \rightarrow \left\{ \begin{array}{l} 0x1305 \\ 0x2005 \end{array} \right\} \rightarrow \left\{ \begin{array}{l} 0x4005 \\ 0x8005 \end{array} \right\}$
 3次块替换

4路组相联, 1次块替换, 8路组相联, 0次块替换

12. 缓存A组数量为 $\frac{256}{16} \div 2 = 2^3 = 8$

缓存B组数量为 $\frac{256}{16} = 2^4 = 16$

块大小16字节, 共128bit, 存放4个32位数据

缓存缺失率为 25%

13. 将 $\text{for (int } j=0; j<128; j++) \{$
 $\quad \text{for (int } i=0; i<64; i++) \{$
 $\quad \quad A[j][i] = A[j][i] + 1;$

14. (1) 块大小32字节 $\frac{4KB}{32B} = 2^7$ 个块, 1个块可容纳 8个 $A[j][i]$

优化前缓存缺失率 $\frac{128 \times 64}{8} = 8 \quad \frac{2^7}{8} = 16$

当 $i=0$ 时, j 每重复16次为一个周期, 每个周期16次缓存缺失

$i=1, 2, 3, 4, 5, 6, 7$
 访问重复与 $i=0$ 情况相同, 故缺失 $16 \times \frac{128}{16} \times \frac{64}{8} = 2^{10}$ 次

优化后. $j=0$ $i=0$ 缺失 $i=1, 2, 3, 4, 5, 6, 7$ hit.
 $i=8$ 缺失 $i=9, 10, 11, 12, 13, 14, 15$ hit.

故 j 的每个循环内, 缺失 $\frac{64}{8} = 8$ 次.

j 的每个循环, 缺失情况相同.

缺失 $8 \times 128 = 2^{10}$ 次.

(2) 优化前. 64×128 个数据需 $\frac{64 \times 128}{8} = 2^{10}$ 个块.

$i=0$ 每次 j 重复 $128 \div 8 = 16$ 次, 不会产生缓存缺失, 缺失 2^7 次.

$i=1, 2, 3, 4, 5, 6, 7$ 每次 j 重复, 不会产生缓存缺失.

$i=8 \sim 15$ 同 $i=0$ 情况, 不会产生缓存缺失, 每次 j 循环都会产生缺失.

可知缓存缺失 $\frac{64}{8} \times 2^7 = 2^{10}$ 次.

优化后. $j=0$ $i=0$ 缺失 $i=1, 2, 3, 4, 5, 6, 7$ hit.

$i=8$ 缺失 $i=9, 10, 11, 12, 13, 14, 15$ hit.

故同 (1) 优化后, 缺失 $8 \times 128 = 2^{10}$ 次.

(3) 优化前 $\frac{64 \times 128}{8} \times 32 = 2^6 \times 2^7 \times 2^2 = 32768$

优化后 32768 .

15 个块, 16 个数据, input 数组

output 数组

0 1 2 3

0 1 2 3

0 miss hit hit hit

miss miss miss miss

1 miss hit hit hit

miss miss miss miss

2 miss hit hit hit

miss miss miss miss

3 miss hit hit hit

miss miss miss miss

16. (1) $\frac{512}{16} = 2^5$ 个块. 2^4 个组 1 个块存放 4 个数据

$i=0$ 时缺失 2 次. 共缺失 $\frac{128 \times 2}{4} = 20$ 次.

故命中率为 $1 - \frac{20}{128} = \frac{15}{16} = \frac{1}{2}$

(2) 不能. 增加缓存大小仍有大量的强制缺失.

以及循环时无法利用之前在好的数据. 缺失次数不变.

B) 可以. 增加块大小. 一次读写缺失存入的数据更多.

减少了以后缺失的发生.