


大(更可能出现这种情况),那么每次对 $a[i][j]$ 的访问都会不命中!

| $a[i][j]$ | $j=0$ | $j=1$ | $j=2$ | $j=3$ | $j=4$ | $j=5$ | $j=6$ | $j=7$ |
|-----------|-------|-------|--------|--------|--------|--------|--------|--------|
| $i=0$ | 1 [m] | 5 [m] | 9 [m] | 13 [m] | 17 [m] | 21 [m] | 25 [m] | 29 [m] |
| $i=1$ | 2 [m] | 6 [m] | 10 [m] | 14 [m] | 18 [m] | 22 [m] | 26 [m] | 30 [m] |
| $i=2$ | 3 [m] | 7 [m] | 11 [m] | 15 [m] | 19 [m] | 23 [m] | 27 [m] | 31 [m] |
| $i=3$ | 4 [m] | 8 [m] | 12 [m] | 16 [m] | 20 [m] | 24 [m] | 28 [m] | 32 [m] |

较高的不命中率对运行时间可以有显著的影响。例如,在桌面机器上, `sumarray-rows` 运行速度比 `sumarraycols` 快 25 倍。总之,程序员应该注意他们程序中的局部性,试着编写利用局部性的程序。

 **练习题 6.17** 在信号处理和科学计算的应用中,转置矩阵的行和列是一个很重要的问题。从局部性的角度来看,它也很有趣,因为它的引用模式既是以行为主(row-wise)的,也是以列为主(column-wise)的。例如,考虑下面的转置函数:

```

1  typedef int array[2][2];
2
3  void transpose1(array dst, array src)
4  {
5      int i, j;
6
7      for (i = 0; i < 2; i++) {
8          for (j = 0; j < 2; j++) {
9              dst[j][i] = src[i][j];
10         }
11     }
12 }
```

假设在一台具有如下属性的机器上运行这段代码:


- `sizeof(int)==4`。
- `src` 数组从地址 0 开始, `dst` 数组从地址 16(十进制)开始。
- 只有一个 L1 数据高速缓存,它是直接映射的、直写和写分配的,块大小为 8 个字节。
- 这个高速缓存总的大小为 16 个数据字节,一开始是空的。
- 对 `src` 和 `dst` 数组的访问分别是读和写不命中的唯一来源。

A. 对每个 `row` 和 `col`,指明对 `src[row][col]` 和 `dst[row][col]` 的访问是命中(h)还是不命中(m)。例如,读 `src[0][0]` 会不命中,写 `dst[0][0]` 也不命中。

| dst数组 | | |
|-------|----|----|
| | 列0 | 列1 |
| 0行 | m | |
| 1行 | | |

| src数组 | | |
|-------|----|----|
| | 列0 | 列1 |
| 0行 | m | |
| 1行 | | |

B. 对于一个大小为 32 数据字节的高速缓存重复这个练习。

 **练习题 6.18** 最近一个很成功的游戏 `SimAqarium` 的核心就是一个紧密循环(tight loop),它计算 256 个海藻(algae)的平均位置。在一台具有块大小为 16 字节($B=16$)、整个大小为 1024 字节的直接映射数据缓存的机器上测量它的高速缓存性能。定义如下:

```

1  struct algae_position {
2      int x;
3      int y;
4  };
```