第六章家庭作业

——117300825 李大鑫

6.25

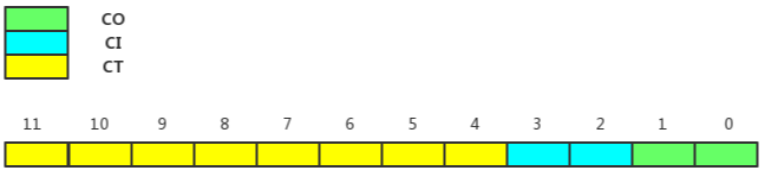
高速缓存	m	C	B	E	S	t	s	b
1	32	1024	4	4	64	24	6	2
2	32	1024	4	256	1	30	0	2
3	32	1024	8	1	128	22	7	3
4	32	1024	8	128	1	29	0	3
5	32	1024	32	1	32	22	5	5
6	32	1024	32	4	8	24	3	5

（m 是物理地址位数，S 是组数，E 是相联度，B 是每个高速缓存块的字节数，t 为标记位数，高速缓存大小 C=SxExB，b 是块偏移位数）

6.29

A:

分析：根据 B=4 确定 b=2，根据 S=4 确定 s=2，根据 m=12，得 t=8。划分如下图：



（本题书上题目给的图有问题，12 位地址，图上画了 13 位）

B:

读 0x834：划分之后得到 CT=83，CI=1，CO=0，因为此时有效位是 0，所以这次读操作不命中，需要向下一级存储中读取数据到 Ct=83，CI=1 的行，这时候设置有效位为 1。

写 0x836：划分之后得到 CT=83，CI=1，CO=2，所以这次写操作命中，读出的值是上次读 0x834 之后得到的，所以未知。

读 0xFFD：划分之后得到 CT=FF，CI=3，CO=1，所以这次写操作命中，读出的值是 0xC0。

操作	地址	命中?	读出的值（或者未知）
读	0x834	不命中	-
写	0x836	命中	未知
读	0xFFD	命中	0xC0

6.33

0x16C8 0x178A 0x16C9 0x16CA 0x16CB 0x1788 0x1789 0x178B

6.37

函数	N=64	N=60
sumA	25%	25%
sumB	100%	25%
sumC	50%	25%

分析：题目中给出 4KB 缓存是直接映射缓存，块大小为 16B（应该是每行？），首先直接映射缓存代表每个组只有一行（E=1）的高速缓存。所以我们可以得知这个 cache 可以存放 4x1024/16=256 个 block，每个 block 16 个字节可以存放 4 个 int，所以 cache 一共可以放 1024 个 int。

1）当 N=64 的时候，因为 cache 一共可以存放 1024 个 int，刚好存放 16 行的 64x64 int 数组，所以从数组到 cache 的映射情况是：每 16 行一组，共 4 组，每组映射到 cache 的位置是重叠的，因此产生替换。

sumA：两个 for 循环顺序读 aij，因为 block 的大小可以放 4 个 int，所以每隔 4 个 int 会产生一次 miss，所以缓存不命中率是 1/4，25%;

sumB：先列后行访问 a 数组，访问列的时候每 16 行完成，后 16 行就会对 cache 中存储的数据进行替换，这样每次访问一个 aij，都不会命中，也就是不命中率为 100%。

sumC: 先列后行访问 a 数组，每次访问的 blocking 是 2x2，每次访问一个 blocking，结合替换的情况，都会有两次 miss，所以不命中率是 1/2，50%。

2）当 N=60 的时候，给数组元素按照先行后列的顺序进行编号 0..3599，根据 cache 可以存放 1024 个 int，对 60x60 int 数组到 cache 的映射情况进行分组：0..1023，1024..2047，2048..3072，3072..3600。发生重叠替换的情况：两个元素的编号分别是 xy，当(x-y)%1024=0 的时候会发生替换。

sumA：顺序读 aij，每个 4 个产生一次 miss，所以缓存不命中率是 25%。

sumB：先列后行，主要观察列循环的时候是否会发生替换的情况，当循环第一列的时候，元素的编号是 0,60,120…3540，任意两个元素之间编号之差为 60 的倍数，所以取模 1024 不可能为 0，而且 cache 大小足够，所以不会发生替换的情况，当对第 1..3 行进行列循环的时候，因为 cache 已经有存了，所以皆命中。忽略对于剩下每 4 列开始的时候是否有命中的讨论（影响相对较小），我们假设它们都不命中，所以**大概**不命中率为 1/4，25%。

sumC：先列后行，每次访问的 blocking 是 2x2，根据 sumB 的讨论以及假设，**大概**不命中率应该与 sumB 相同为 25%。

6.41

分析：

直接映射高速缓存有 64KB，每行 4 个字节，同时 sizeof(struct pixel)=4B，所以缓存可以存放 64K/4=16K=16384 个 pixel。映射时每 16384 个为 1 组。

数组访问的顺序是先列后行，然后对 rgba4 个域进行分别赋值，观察一列的访问是否会发生替换，数组大小是 640x480，与 6.37 的 N=60 的 sumB 思路相同，我们会发现访问一列的时候，任意两个元素的**编号**（编号规则相同）之差都是 480 的倍数，所以对 16384 取模一定不为 0，所以不会发生替换。所以对字节的访问是每 4 个字节的写发生一次 miss，所以 25%的写不会命中。

6.45

实现尽可能快的、未知大小的矩阵转置函数。

1）利用 blocking 技术，设置一个 Bsize*Bsize 大小的 blocking，以 blocking 为单位进行循环，每次循环对 blocking 内部元素进行转置。主要是为了充分利用 cache 中存储的内容减少读 src 数组的 miss 数目（具体思想与 CacheLab trans 实验相同，此处不展开）。

2）因为目标矩阵是行宽相等为 dim 的矩阵，所以我们可以通过特殊处理对角线替换情况来减少 miss 的数目。

以下是代码、对两种转置函数的时间测试、对 transposeB 的正确性测试，数组大小为 1024x1024，Bsize 为 8（运行时增大了程序栈的大小）。

代码：

```
#include <stdio.h>
#include <time.h>

#define Bsize 8
void transposeA(int *dst, int *src, int dim)
{
    int i, j;
    for (i = 0; i < dim; ++i)
    {
        for (j = 0; j < dim; ++j)
        {
            dst[j*dim + i] = src[i*dim +j];
        }
    }
}
void transposeB(int *dst, int *src, int dim)
{
    int rowBlock, colBlock;
    int r, c;
    int temp = 0, d = 0;
    for(colBlock = 0; colBlock < dim; colBlock += Bsize)  {
        for(rowBlock = 0; rowBlock < dim; rowBlock += Bsize)  {
            for(r = rowBlock; r < rowBlock + Bsize; r++)  {
```

```
            for(c = colBlock; c < colBlock + Bsize; c++)  {
                if(r != c)  {
                    dst[c*dim+r] = src[r*dim+c];
                }  else  {
                    temp = src[r*dim+c];
                    d = r;
                }
            }
            if (rowBlock == colBlock)  {
                dst[d*dim+d] = temp;
            }
        }
    }
}

int main() {
    int dim = 1024;
    int allsize = dim*dim;
    int src[allsize],dst[allsize];
    for(int i=0;i<allsize;i++) {
        src[i] = i;
    }
    clock_t start,finish;
    double duration;
    start = clock();
    transposeA(dst,src,dim);
    finish = clock();
    printf("transposeA duration: %ld\n",finish-start);
    start = clock();
    transposeB(dst,src,dim);
    finish = clock();
    printf("transposeB duration: %ld\n",finish-start);
    puts("after transpose B , test mattrix dst:");
    for(int i=0;i<dim;i++) {
        for(int j=0;j<dim;j++) {
            if (dst[i*dim+j] != src[j*dim+i]) {
                puts("dst answer erro :)");
                return 0;
            }
        }
    }
    puts("right answer!");
    return 0;
```

```
}
```

代码测试结果:

```
E:\CodeTraining\ProgramingLanguageTest\C\csapp_chapter6\cmake-build-debug\csapp_chapter6.exe

transposeA duration: 18

transposeB duration: 7

after transpose B , test mattrix dst:

right answer!
```