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 |

6.29

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CT | CT | CT | CT | CT | CT | CT | CI | CI | CI | CO | CO |

（1）读0x834：经过划分之后CT=83，CI=1，CO=0，有效位是 0，所以这次读操作不命中，此时需要读取下一级存储，读取数据到 Ct=83，CI=1 的行，这时候设置有效位为 1。

（2）写0x836：经过划分之后CT=83，CI=1，CO=2，写操作命中，读出的值是上次读 0x834 之后得到的，写的结果未知。

（3）读 0xFFD：经过划分之后CT=FF，CI=3，CO=1，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

0x1788 ~ 0x178B

0x16C8 ~ 0x16CB

6.37

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

分析：

4KB直接映射缓存，块大小为 16B，每个组E=1。可知此cache可以存放4x1024/16=256个block，int占四个字节，每个block的16个字节可以存放4个int，所以此cache一共可以放1024个int。

（1）当 N=64 。每组映射到 cache 的位置是重叠的，产生替换：cache 一共可以存放1024个int，刚好可以存放16行的64x64 int数组，从数组到 cache 的映射情况是每16行一组，共4组。

sumA：

两个for循环顺序读 a[i][j] , block的大小可以放4个int，所以每隔4个int会产生一次 miss ,缓存不命中率是25% 。

sumB：

先列后行访问数组a。在访问列的过程中，每完成16行，后16行就会替换cache中存储的数据，每次访问一个a[i][j]，都不会命中，也就是不命中率为100% 。

sumC：

先列后行访问数组a 。每次访问的blocking是2x2，由于替换的情况，访问一次blocking，都会有两次 miss，所以不命中率是50% 。

（2）当 N=60 。将元素编号为0-3599，当两个元素的编号之差为1024的整数倍时，发生重叠替换。

cache可以存放 1024 个int，由此对60x60 int数组到 cache 的映射情况每1024个分为一组。

sumA：

顺序读 a[i][j]，每个 4 个产生一次 miss，缓存不命中率是 25%。

sumB：

先列后行。替换主要发生在列循环中。循环第一列时，任意两个元素之间编号之差为 60\*n，不可能是1024的整数倍，且对于cache不会发生替换。当对第1..3行进行列循环的时候，全部命中。对于剩下每4列开始时，假设都不命中，所以大概不命中率为25%。

sumC：

先列后行。每次访问的 blocking是2x2，大概不命中率应该与sumB相同为 25%。

6.41

直接映射高速缓存64KB，每行4个字节，struct pixel的大小为4B，缓存可以存放64K/4=16K=16384个。映射时1组16384个pixel。先列后行访问，再对rgba4个域进行分别赋值。对字节的访问是每4个字节的写发生一次miss。数组大小是 640x480，发现访问一列的时候，任意两个元素的编号之差都是 480 的倍数，不可能是16384的整数倍，所以不会发生替换。25%的写不会命中。

6.45

尽可能快+未知大小

（1）blocking技术，减少读src数组时的miss

（2）处理对角线，减少dim数组的对角线替换

void transposeB(int \*dst, int \*src, int dim)

{

int RB, CB;

int R, C;

int temp = 0, D = 0;

for (CB = 0; CB < dim; CB += B\_s)

for (RB = 0; RB < dim; RB += B\_s)

for (R = RB; R < RB + B\_s; R++)

{

for (C = CB; C < CB + B\_s; C++)

{

if(R!=C)

dst[C\*dim + R] = src[R\*dim + C];

else

{

temp = src[R\*dim + C];

D = R;

}

}

if (RB == CB)

dst[D\*(dim + 1)] = temp;

}

}

测试函数：

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define size\_d 1024

#define size\_a size\_d\*size\_d

int main() {

int src[size\_a], dst[size\_a];

clock\_t start1, start2, finish1, finish2;

for (int i = 0; i < size\_a; i++)

src[i] = i;

start1 = clock();

transposeA(dst, src, size\_d);

finish1 = clock();

start2 = clock();

transposeB(dst, src, size\_d);

finish2 = clock();

printf("The decrease of time is %d\n", (int)((start1 - finish1) - (start2 - finish2)));

printf("The result after transposeB is ");

for (int i = 0; i < size\_d; i++)

for (int j = 0; j < size\_d; j++)

if (dst[i \* size\_d + j] != src[j \* size\_d + i]) {

printf("wrong \n");

return 0;

}

printf("right \n");

system("pause");

return 0;

}

测试结果：

