**CSAPP 实验之 cache lab**

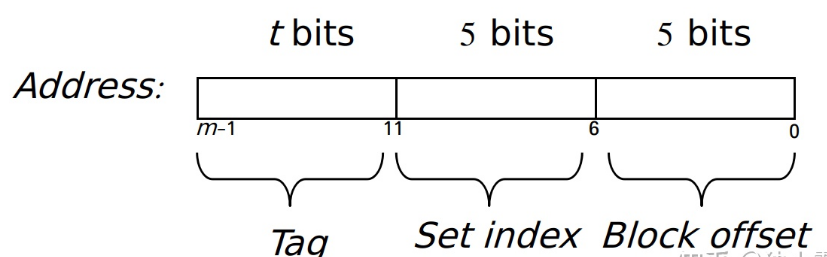
## **Part B**

**预备内容**

首先，安装valgrind。

linux> sudo apt-get install valgrind

s = 5, E = 1, b = 5。S=2^5=32，每组一行，B=2^5=32。  
cache有32组，每组一行，每行可以储存32字节的数据，int类型为4字节，所以每个缓存中的每个数据块可以保存8个int元素。



### **32\*32**

在writeup 中提示，可以试一试分块的方法。具体的做法是，先把整个矩阵分成几大块，之后以“块”为单位进行转置。一个数据块可以保存8个int元素，存一行需要4个set，整个cache可以存8行，则将矩阵分成8\*8的块。在每一块中，从A中读一行，再写到B一列。代码如下：

if (M == 32)

for (int j = 0; j < M; j += 8)

for (int i = 0; i < N; i++) {

int n1 = A[i][j];

int n2 = A[i][j+1];

int n3 = A[i][j+2];

int n4 = A[i][j+3];

int n5 = A[i][j+4];

int n6 = A[i][j+5];

int n7 = A[i][j+6];

int n8 = A[i][j+7];

B[j][i] = n1;

B[j+1][i] = n2;

B[j+2][i] = n3;

B[j+3][i] = n4;

B[j+4][i] = n5;

B[j+5][i] = n6;

B[j+6][i] = n7;

B[j+7][i] = n8;

}

mark@mark-VirtualBox:~/桌面/cachelab-handout$ make

gcc -g -Wall -Werror -std=c99 -m64 -O0 -c trans.c

gcc -g -Wall -Werror -std=c99 -m64 -o test-trans test-trans.c cachelab.c trans.o

gcc -g -Wall -Werror -std=c99 -m64 -O0 -o tracegen tracegen.c trans.o cachelab.c

# Generate a handin tar file each time you compile

tar -cvf handin.tar csim.c trans.c

csim.c

trans.c

mark@mark-VirtualBox:~/桌面/cachelab-handout$ ./test-trans -M 32 -N 32

Function 0 (2 total)

Step 1: Validating and generating memory traces

Step 2: Evaluating performance (s=5, E=1, b=5)

func 0 (Transpose submission): hits:1766, misses:287, evictions:255

Function 1 (2 total)

Step 1: Validating and generating memory traces

Step 2: Evaluating performance (s=5, E=1, b=5)

func 1 (Simple row-wise scan transpose): hits:870, misses:1183, evictions:1151

Summary for official submission (func 0): correctness=1 misses=287

TEST\_TRANS\_RESULTS=1:287

当然这是还有优化空间的，首先，对于对角线两侧的矩阵块，已经最优了。但是对于对角线上的矩阵块，还存在优化空间。就是先把A矩阵块中的数据先直接赋值到B的对应位置，然后在B上进行转置。

for (int i = m; i < m + 8; ++i){

// 必须整行搬运，这种写法相当于完全展开内层循环 \_0 = A[i][n];\_1 = A[i][n + 1];\_2 = A[i][n + 2];\_3 = A[i][n + 3];

\_4 = A[i][n + 4]; \_5 = A[i][n + 5]; \_6 = A[i][n + 6];\_7 = A[i][n + 7];

B[i][n] = \_0; B[i][n + 1] = \_1;B[i][n + 2] = \_2; B[i][n + 3] = \_3;

B[i][n + 4] = \_4;B[i][n + 5] = \_5;B[i][n + 6] = \_6;B[i][n + 7] = \_7;

}

// 原址转置！

for (int i = m; i < m + 8; ++i)

for (int j = n + (i - m + 1); j < n + 8; ++j)

if (i != j){

\_0 = B[i][j];

B[i][j] = B[j][i];

B[j][i] = \_0;

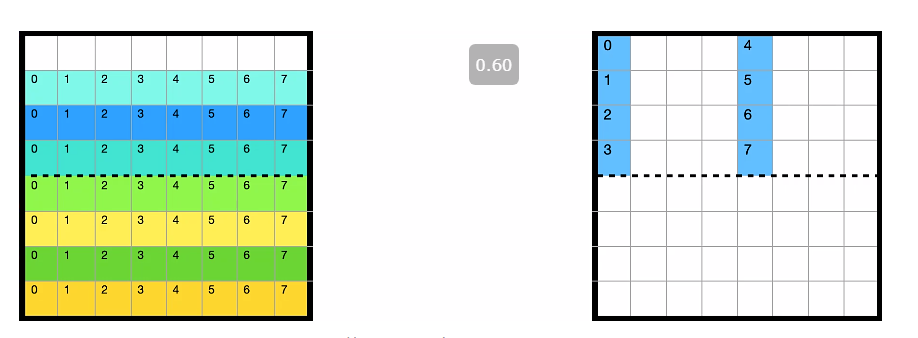
}

misses：259。

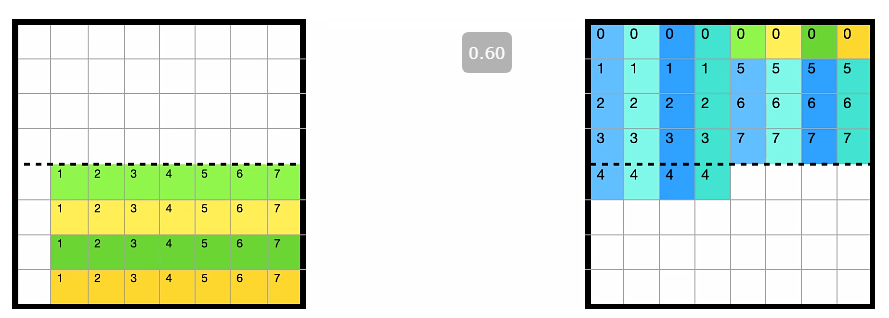
### **64\*64**

还是先向上面一样，把整个矩阵分成若干 8 × 8 的块，发现效果很不好。可以发现在矩阵块中前四行和后四行的set值是一样的，导致miss提高。如果换成4\*4的块，就浪费了数据块后八个位置，似乎miss仍然非常高。

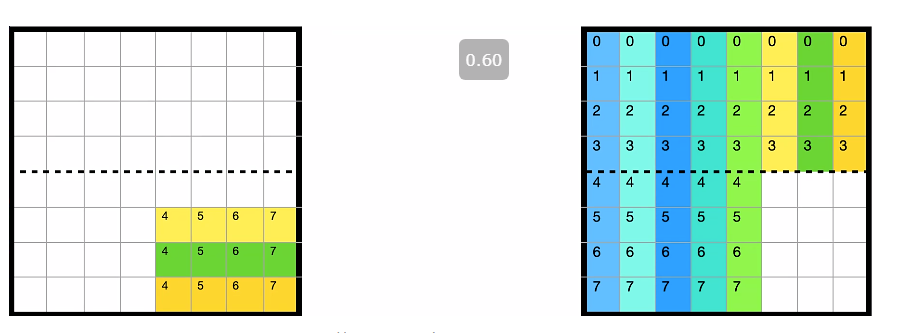
所以还是在8\*8的分块的基础上，进行改变。为了充分利用数据块，所以将A矩阵块一行的数据存到变量里，同时为了充分利用B的矩阵块，不是直接将这一行赋给B的一列，而是存在前四行的第一列和第四列。这样就不会存在在给一列后四位赋值时驱逐前四行的缓存。



如此循环存储前四行之后，再将A矩阵块后四行按列存变量，再将存在B矩阵块非正确位置的数据按行存变量。接着把存入变量的A中数据分配给刚才的B部分，把B部分的数据分配给正确的位置。



最后，把A中右下角的4\*4块按行分配给B中。



if (M == 64) {

for (int j = 0; j < M; j += 8)

for (int i = 0; i < N; i += 8) {

for (int x = i; x < i + 4; x++) {

int n1 = A[x][j];

int n2 = A[x][j+1];

int n3 = A[x][j+2];

int n4 = A[x][j+3];

int n5 = A[x][j+4];

int n6 = A[x][j+5];

int n7 = A[x][j+6];

int n8 = A[x][j+7];

B[j][x] = n1;

B[j+1][x] = n2;

B[j+2][x] = n3;

B[j+3][x] = n4;

B[j][x+4] = n5;

B[j+1][x+4] = n6;

B[j+2][x+4] = n7;

B[j+3][x+4] = n8;

}

for (int y = j; y < j + 4; y++) {

int n1 = A[i+4][y];

int n2 = A[i+5][y];

int n3 = A[i+6][y];

int n4 = A[i+7][y];

int n5 = B[y][i+4];

int n6 = B[y][i+5];

int n7 = B[y][i+6];

int n8 = B[y][i+7];

B[y][i+4] = n1;

B[y][i+5] = n2;

B[y][i+6] = n3;

B[y][i+7] = n4;

B[y+4][i] = n5;

B[y+4][i+1] = n6;

B[y+4][i+2] = n7;

B[y+4][i+3] = n8;

}

for (int y = j + 4; y < j + 8; y++) {

int n1 = A[i+4][y];

int n2 = A[i+5][y];

int n3 = A[i+6][y];

int n4 = A[i+7][y];

B[y][i+4] = n1;

B[y][i+5] = n2;

B[y][i+6] = n3;

B[y][i+7] = n4;

}

}

}

mark@mark-VirtualBox:~/桌面/cachelab-handout$ ./test-trans -M 64 -N 64

Function 0 (2 total)

Step 1: Validating and generating memory traces

Step 2: Evaluating performance (s=5, E=1, b=5)

func 0 (Transpose submission): hits:9074, misses:1171, evictions:1139

Function 1 (2 total)

Step 1: Validating and generating memory traces

Step 2: Evaluating performance (s=5, E=1, b=5)

func 1 (Simple row-wise scan transpose): hits:3474, misses:4723, evictions:4691

Summary for official submission (func 0): correctness=1 misses=1171

TEST\_TRANS\_RESULTS=1:1171

### **61\*67**

首先，同上，分成8\*8的块发现miss超了，然后增大块，分成16\*16的块，发现miss是2002

for (int m = 0; m < M; m += 16)

for (int n = 0; n < N; n += 16) {

for (int j = m; j < m + 16 && j < M; j++)

for (int i = n; i < n + 16 && i < N; i++)

B[j][i] = A[i][j];

}

}

mark@mark-VirtualBox:~/桌面/cachelab-handout$ ./test-trans -M 61 -N 67

Function 0 (2 total)

Step 1: Validating and generating memory traces

Step 2: Evaluating performance (s=5, E=1, b=5)

func 0 (Transpose submission): hits:6177, misses:2002, evictions:1970

Function 1 (2 total)

Step 1: Validating and generating memory traces

Step 2: Evaluating performance (s=5, E=1, b=5)

func 1 (Simple row-wise scan transpose): hits:3756, misses:4423, evictions:4391

Summary for official submission (func 0): correctness=1 misses=2002

TEST\_TRANS\_RESULTS=1:2002

后来发现先遍历行，后遍历列，miss就是1995，因为行最后是多出三行，而列是多出13列。所以miss减少。不知道再怎么优化了。

综上，代码如下;

char transpose\_submit\_desc[] = "Transpose submission";

void transpose\_submit(int M, int N, int A[N][M], int B[M][N]){

if (M == 32)

for (int j = 0; j < M; j += 8)

for (int i = 0; i < N; i++) {

int n1 = A[i][j];

int n2 = A[i][j+1];

int n3 = A[i][j+2];

int n4 = A[i][j+3];

int n5 = A[i][j+4];

int n6 = A[i][j+5];

int n7 = A[i][j+6];

int n8 = A[i][j+7];

B[j][i] = n1;

B[j+1][i] = n2;

B[j+2][i] = n3;

B[j+3][i] = n4;

B[j+4][i] = n5;

B[j+5][i] = n6;

B[j+6][i] = n7;

B[j+7][i] = n8;

}

else if (M == 64) {

for (int j = 0; j < M; j += 8)

for (int i = 0; i < N; i += 8) {

for (int x = i; x < i + 4; x++) {

int n1 = A[x][j];

int n2 = A[x][j+1];

int n3 = A[x][j+2];

int n4 = A[x][j+3];

int n5 = A[x][j+4];

int n6 = A[x][j+5];

int n7 = A[x][j+6];

int n8 = A[x][j+7];

B[j][x] = n1;

B[j+1][x] = n2;

B[j+2][x] = n3;

B[j+3][x] = n4;

B[j][x+4] = n5;

B[j+1][x+4] = n6;

B[j+2][x+4] = n7;

B[j+3][x+4] = n8;

}

for (int y = j; y < j + 4; y++) {

int n1 = A[i+4][y];

int n2 = A[i+5][y];

int n3 = A[i+6][y];

int n4 = A[i+7][y];

int n5 = B[y][i+4];

int n6 = B[y][i+5];

int n7 = B[y][i+6];

int n8 = B[y][i+7];

B[y][i+4] = n1;

B[y][i+5] = n2;

B[y][i+6] = n3;

B[y][i+7] = n4;

B[y+4][i] = n5;

B[y+4][i+1] = n6;

B[y+4][i+2] = n7;

B[y+4][i+3] = n8;

}

for (int y = j + 4; y < j + 8; y++) {

int n1 = A[i+4][y];

int n2 = A[i+5][y];

int n3 = A[i+6][y];

int n4 = A[i+7][y];

B[y][i+4] = n1;

B[y][i+5] = n2;

B[y][i+6] = n3;

B[y][i+7] = n4;

}

}

}

else {

for (int m = 0; m < M; m += 16)

for (int n = 0; n < N; n += 16) {

for (int j = m; j < m + 16 && j < M; j++)

for (int i = n; i < n + 16 && i < N; i++)

B[j][i] = A[i][j];

}

}

}

mark@mark-VirtualBox:~/桌面/cachelab-handout$ ./driver.py

Part A: Testing cache simulator

Running ./test-csim

Your simulator Reference simulator

Points (s,E,b) Hits Misses Evicts Hits Misses Evicts

3 (1,1,1) 9 8 6 9 8 6 traces/yi2.trace

3 (4,2,4) 4 5 2 4 5 2 traces/yi.trace

3 (2,1,4) 2 3 1 2 3 1 traces/dave.trace

3 (2,1,3) 167 71 67 167 71 67 traces/trans.trace

3 (2,2,3) 201 37 29 201 37 29 traces/trans.trace

3 (2,4,3) 212 26 10 212 26 10 traces/trans.trace

3 (5,1,5) 231 7 0 231 7 0 traces/trans.trace

6 (5,1,5) 265189 21775 21743 265189 21775 21743 traces/long.trace

27

Part B: Testing transpose function

Running ./test-trans -M 32 -N 32

Running ./test-trans -M 64 -N 64

Running ./test-trans -M 61 -N 67

Cache Lab summary:

Points Max pts Misses

Csim correctness 27.0 27

Trans perf 32x32 8.0 8 287

Trans perf 64x64 8.0 8 1171

Trans perf 61x67 10.0 10 2002

Total points 53.0 53