

# Cache3 实验报告

【10%】实验 test-trans, 以 M32N32 矩阵为例, 并通过 csim-ref 详细选项 (-v) 在缓存跟踪 trace.f 文件中观察结果。

(1) 输入指令 make。

```
zhangjiwei@zhangjiwei-virtual-machine:~/cachelab-handout$ make
# Generate a handin tar file each time you compile
tar -cvf zhangjiwei-handin.tar csim.c trans.c
csim.c
trans.c
```

(2) 输入指令 ./test-trans -M 32 -N 32 测试 test-trans。

```
zhangjiwei@zhangjiwei-virtual-machine:~/cachelab-handout$ ./test-trans -M 32 -N 32
Function 0 (2 total)
Step 1: Validating and generating memory traces
Validation error at function 0! Run ./tracegen -M 32 -N 32 -F 0 for details.
Skipping performance evaluation for this function.

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
```

(3) 输入指令 ./csim-ref -v -s 5 -E 1 -b 5 -t trace.f1 > f0.txt 冲突测试 f0 文件, 结果输出到文件 f0.txt。

【10%】编写 csim.c 处理本条命令行同时输出显示标记位和组号。

在 csim.c 的主函数中增加语句: printf("setBits is %x, tagBits is %x\n", setBits, tagBits), 用来输出显示标记位和组号。

```
if(opt[0]=='I')
    continue;
int setBits = getSet(addr, s, b);
int tagBits = getTag(addr, s, b);
printf("setBits is %x, tagBits is %x\n", setBits, tagBits);
```

【20%】实验 test-trans 以 M4N4 为例, 分析示例函数 miss 过多的原因。

(1) 输入指令 ./csim-ref -v -s 5 -E 1 -b 5 -t trace.f1 > f0.txt 冲突测试 f0 文件, 结果输出到文件 f0.txt, 保留组号和标记位。

```
zhangjiwei@zhangjiwei-virtual-machine:~/cachelab-handout$ ./test-trans -M 4 -N 4
Function 0 (2 total)
Step 1: Validating and generating memory traces
Validation error at function 0! Run ./tracegen -M 4 -N 4 -F 0 for details.
Skipping performance evaluation for this function.

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:15, misses:22, evictions:19

Summary for official submission (func 0): correctness=0 misses=2147483647
TEST_TRANS_RESULTS=0:2147483647
```

(2) 详细分析: misses 数位 22, 驱逐数为 19。

- a) A 数组访问 A[0][0] 冷不命中，第 4 个 block 被装入数组 A 的数据。
- b) B 数组访问 B[0][0] 虽然 B 数组要被赋值的 A[0][0] 已经在第 4 个 block 中，但是数组 B 和数组 A 的地址显然不同，所以标记位不同，所以冲突产生驱逐，驱逐第 4 个块，第 4 个 block 被装入数组 B 的数据。
- c) A 数组访问 A[0][1] 原理同步骤 b 产生了冲突并驱逐，第 4 个 block 被装入数组 A 的数据。
- d) B 数组访问 B[1][0] 原理同步骤 b 产生了冲突并驱逐，第 4 个 block 被装入数组 B 的数据。
- e) A 数组访问 A[0][2] 原理同步骤 c 产生了冲突。
- f) B 数组访问 B[2][0] 在这里产生变化 B 将数据装载入第 5 个 block，冷不命中。
- g) A 数组访问 A[0][3] 标志位相同，因此 hit，其他与上述分析类似。

综上 miss 最主要是因为冲突不命中，转置操作不断地因为访问数组 A 和 B 而替换 block 中的数据，因此我们可以通过设置临时变量一次访问块中的多个元素，减少此类 miss。

**【20%】实验 test-trans 以 M32N32 为例，并分析示例函数 miss 过多的原因。**

(1) 输入指令 ./csim-ref -v -s 5 -E 1 -b 5 -t trace.f1 > f0.txt 冲突测试 f0 文件，结果输出到文件 f0.txt。

```
zhangjiwei@zhangjiwei-virtual-machine:~/cachelab-handout$ ./csim-ref -v -s 5 -E 1 -b 5 -t trace.f1 > f0.txt
zhangjiwei@zhangjiwei-virtual-machine:~/cachelab-handout$ ./test-trans -M 32 -N 32

Function 0 (2 total)
Step 1: Validating and generating memory traces
Validation error at function 0! Run ./tracegen -M 32 -N 32 -F 0 for details.
Skipping performance evaluation for this function.

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=0 misses=2147483647
TEST_TRANS_RESULTS=0:2147483647
```

(2) 详细分析：

- a) 原始提供的转置操作，还是会和第 3 问一样产生许多冲突不命中。
- b) 在 32\*32 矩阵，跨过第 8 行之后的操作，因为超过了 block 的上限，会覆盖到原来位置的数，再次产生冲突不命中。
- c) 进行一些分析，决定采取分块的策略，在该情况下，因为数组 B 每次都会换块，因此数组 A 基本都能命中，所以产生 miss 所以采用分块的策略，转置完一个 8\*8 行，再转置下一个 8\*8 行。

**【20%】分别按 4 和 8 分块编写 transpose\_submit() 代码，记录 test-trans 以 M32N32 矩阵下 Miss 数目结果；进一步编写代码重新处理相同下标的对角线上元素来再次优化，列表记录数组 B 第一个 8\*8 块写操作的命中情况。**

(1) 4 分块代码和 miss 结果

```
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    int i, j, x, y;
    if( (M == 32) && (N == 32) )
    {
        for(i = 0; i < N; i = i + 4)
        {
            for(j = 0; j < M; j = j + 4)
            {
                for(x = i; x < i+4; x++)
                {
                    for(y = j; y < j+4; y++)
                    {
                        B[y][x] = A[x][y];
                    }
                }
            }
        }
    }
}
```

```
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:1566, misses:487, evictions:455
```

## (2) 8 分块代码和 miss 结果

```
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    int i, j, x, y;
    if( (M == 32)&&(N == 32) )
    {
        for(i = 0; i < N; i = i + 8)
        {
            for(j = 0; j < M; j = j + 8)
            {
                for(x = i; x < i+8; x++)
                {
                    for(y = j; y < j+8; y++)
                    {
                        B[y][x] = A[x][y];
                    }
                }
            }
        }
    }
}
```

```
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:1710, misses:343, evictions:311
```

## (2) 对角线优化代码和 miss 结果

为减少冲突 定义 8 个临时变量 直接存储一个 block 中的所有数据

```
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    int i, j, x, y;
    int tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
    if( (M == 32)&&(N == 32) )
    {
        for(i = 0; i < N; i = i + 8)
        {
            for(j = 0; j < M; j = j + 8)
            {
                if(i == j)
                {
                    for(int x = i; x < i+8; x++)
                    {
                        tmp0 = A[x][j]; tmp1 = A[x][j+1];
                        tmp2 = A[x][j+2]; tmp3 = A[x][j+3];
                        tmp4 = A[x][j+4]; tmp5 = A[x][j+5];
                        tmp6 = A[x][j+6]; tmp7 = A[x][j+7];
                        B[j][x] = tmp0; B[j+1][x] = tmp1;
                        B[j+2][x] = tmp2; B[j+3][x] = tmp3;
                        B[j+4][x] = tmp4; B[j+5][x] = tmp5;
                        B[j+6][x] = tmp6; B[j+7][x] = tmp7;
                    }
                }
                else
                {
                    for(x = i; x < i+8; x++)
                    {
                        for(y = j; y < j+8; y++)
                        {
                            B[y][x] = A[x][y];
                        }
                    }
                }
            }
        }
    }
}
```

```
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
```

【20%】分析采用分块技术后 miss 改善的原因。按 4 分块编写 transpose\_submit() 代码，记录 test-trans 以 M32N32 矩阵下 miss 数目的结果；以两个 4\*4 为例分析 hit 增多的原因。注意：加载 8\*4 的矩阵 A，存储到 4\*8 的矩阵 B。

### (1) 4\*4 矩阵下 miss 数目的结果

```
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:1566, misses:487, evictions:455
```

### (2) 8\*4 矩阵下 miss 数目的结果

```
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:1710, misses:343, evictions:311
```

### (3) 采用分块技术之后 miss 改善的原因。

在 4\*4 分块没有充分利用好 block 的 8 个数据，所以在 8\*4 分块中有很多冲突不命中。