20170801020 9-计科 1702 魏一鹏-日志 2.6

1. 实验 test-trans，以 M32N32 为例，并通过 csim-ref 详细选项（-v）在缓存中跟踪 trace.f 文件中观察结果（10%）

```
yihuahuahua@ubuntu:~/桌面/cash/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:870, misses:1183, evictions:1151

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=1183

TEST_TRANS_RESULTS=1:1183
```

上图，make 后输入指令./test-trans -M 32 -N 32,对 test-trans 进行了测试

```
yihuahuahua@ubuntu:~/桌面/cash/cachelab-handout$ ./csim-ref -v -s 5 -E 1 -b 5 -
t trace.f0 > f0.txt
```

上图，输入指令./csim-ref -v -s 5 -E 1 -b 5 -t trace.f0 > f0.txt,对 f0 文件进行了冲突测试，并将详细结果输出到文件 f0.txt

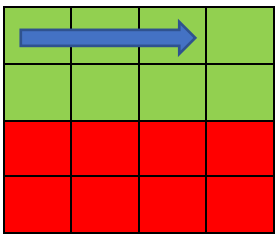
2. 编写 csim.c 处理本条命令行同时输出显示标记位和组号（10%）

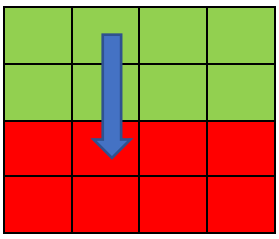
```
        continue;
    int setBits = getSet(addr, s, b);
    int tagBits = getTag(addr, s, b);
    printf("setBits is %x, tagBits is %x\n", setBits, tagBits);

    if(isVerbose == 1)
        printf("%s %x,%d  ", opt, addr, number);
```

如上图，在 csim.c 的主函数中增加一条语句：printf("setBits is %x, tagBits is %x\n", setBits, tagBits);从而在使用此文件时，可以输出显示标记位和组号。

3. 实验 test-trans 以 M4N4 为例，分析示例函数 miss 过多的原因。（20%）

首先在 s5E1b5 情况下，一个 block 有 32 个字节，int 为 4 字节，因此一次缓存可以装 8 位元素。同时，我们应该明确，数组 a 是横向移动数据，数组 b 是纵向移动数据，效果如下（左图为块 A，右图为块 B，同一颜色表示同一个 block）：





输入指令./csim -v -s 5 -E 1 -b 5 -t trace.f0 > f0.txt,对 f0 文件进行了冲突测试，并将详细结果输出到文件 f0.txt，其中保留了组号和标记位

```
setBits is 4, tagBits is e28
S 38a08c,1 miss
setBits is 5, tagBits is e28
L 38a0a0,8 miss
setBits is 4, tagBits is e28
L 38a084,4 hit
setBits is 4, tagBits is e28
L 38a080,4 hit
setBits is 4, tagBits is c28
L 30a080,4 miss eviction
setBits is 4, tagBits is d28
S 34a080,4 miss eviction
setBits is 4, tagBits is c28
L 30a084,4 miss eviction
setBits is 4, tagBits is d28
S 34a090,4 miss eviction
setBits is 4, tagBits is c28
L 30a088,4 miss eviction
setBits is 5, tagBits is d28
S 34a0a0,4 miss eviction
setBits is 4, tagBits is c28
L 30a08c,4 hit
```

```
hits:15 misses:22 evictions:20
```

我们可以看到了详细的操作与最后的结果，发现 misses 数为 22，但是驱逐数为 20，通过图示对详细操作进行分析：

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 数组要被赋取的值 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，驱逐了第 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]，原理和步骤 b 一样，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]，原理和步骤 b 一样，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 一样产生冲突。

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。

其他的与上述分析类似，限于篇幅不做过多阐述。

我们通过上述分析可以发现，miss 最主要是因为冲突不命中，在示例方法中，转置操作不断地因为访问数组 A 和 B 而换了 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，这样可以减少此类 miss

4. 实验 test-trans 以 M32N32 为例，并分析示例函数 miss 过多的原因。(20%)

对于 32*32 的矩阵，一个 block 可以容纳 8 个数据，因此一行 4 个 block，32 个 block 可以最多容纳 8 行的数据。

输入指令./csim -v -s 5 -E 1 -b 5 -t trace.f0 > f0.txt,对 f0 文件进行了冲突测试，并将详细结果输出到文件 f0.txt，可以得到如下截图，只放上了部分。

```
2047 L 30b074,4 hit
2048 S 34af7c,4 miss eviction
2049 L 30b078,4 hit
2050 S 34affc,4 miss eviction
2051 L 30b07c,4 hit
2052 S 34b07c,4 miss eviction
2053 S 38a08d,1 miss eviction
2054 hits:870 misses:1183 evictions:1151
```

通过最开始的讲述，结合具体操作，对其进行一些分析：

- 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
- 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据，再次产生冲突不命中。

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 主要是因为问题 2，所以采用分块的策略，转置完了一个 8*8 行，再转置下一个 8*8 行

5. 分别按 4 和 8 分块编写 transpose_submit()代码，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。(20%)

- 4 分块（左）和 8 分块（右）
  - 代码

```
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];
            }
        }
    }
}
```

```
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];
            }
        }
    }
}
```

  - Miss 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
```

```
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
```

- 继续优化对角线

  我们发现，对角线以外的元素，可以好好地不驱逐地转置，但是位于对角线处的元素，就如同 4*4 的矩阵一样，A 转置向 B 中存在同块的冲突，因此定义 8 个临时变量，直接存储一个 block 中的所有数据，这样就减少了冲突。

  - 代码

```
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];
                }
```

  - Miss 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:1766, misses:287, evictions:255
```

- 列表记录 B 数组第一个 8*8 块的写操作命中（m 为 miss，h 为 hit）

m	h	h	h	h	h	h	h
m	m	h	h	h	h	h	h
m	h	m	h	h	h	h	h
m	h	h	m	h	h	h	h
m	h	h	h	m	h	h	h
m	h	h	h	h	m	h	h
m	h	h	h	h	h	m	h
m	h	h	h	h	h	h	m

因为第一列都是开辟新 block，都是冷不命中，对角线则是 AB 冲突，所以 miss

6. 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（20%）

   - 4*4miss

```
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
```

   - 8*4miss

```
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:1686, misses:367, evictions:335
```

   - 改善的原因

     显然，4*4 分块，在一块中，并没有充分利用好 block 的 8 个数据，所以会 8*4 多冲突不命中很多。