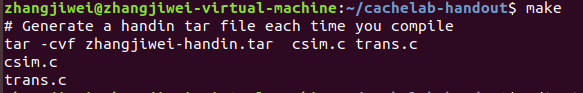
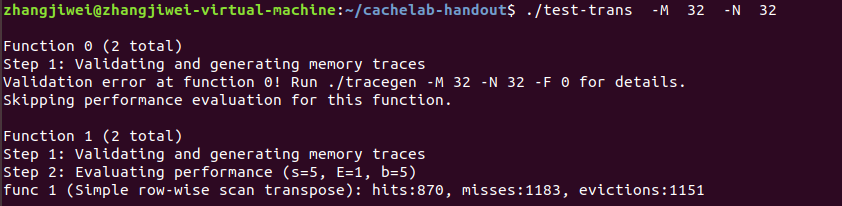
# 实验报告

**【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。



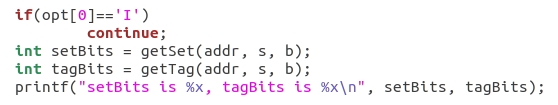
1. 输入指令./test-trans -M 32 -N 32 测试 test-trans。



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。

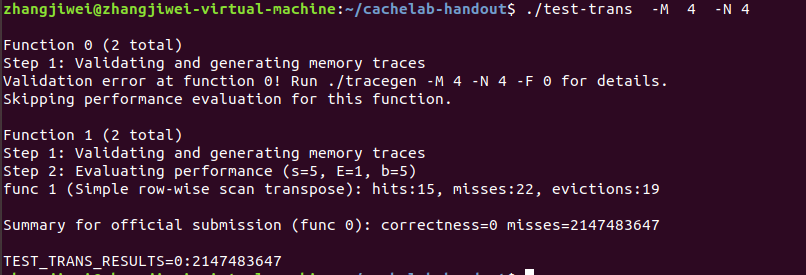
**【10%】编写csim.c处理本条命令行同时输出显示标记位和组号。**

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。



**【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。



1. 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。

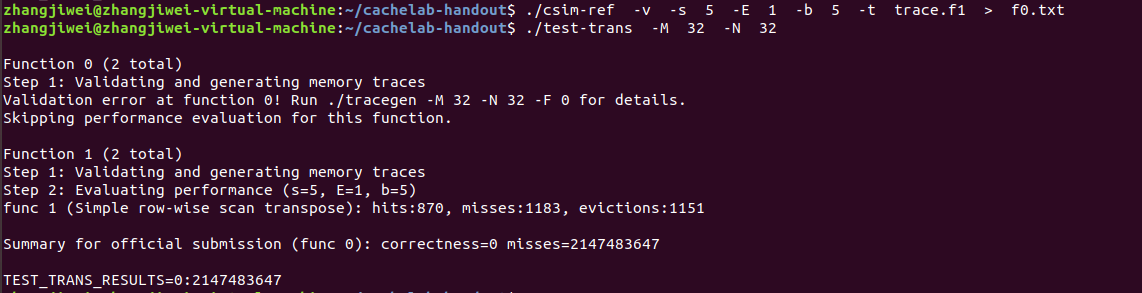
f) B数组访问B[2][0] 在这里产生变化B将数据装载入第5个block，冷不命中。

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。



（2）详细分析：

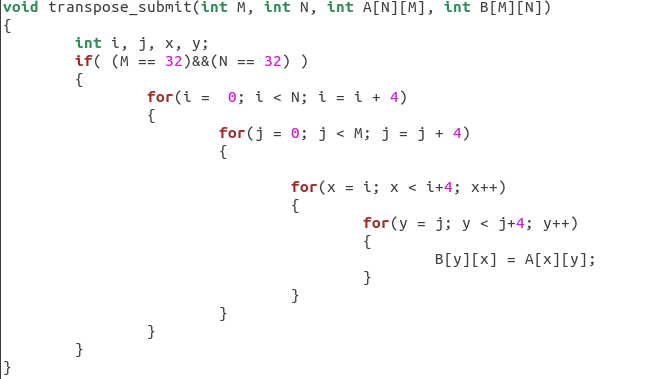
a)原始提供的转置操作，还是会和第3问一样产生许多冲突不命中。

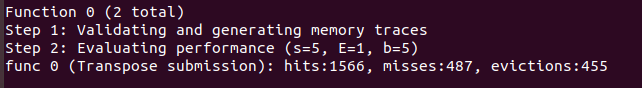
b)在32\*32矩阵，跨过第8行之后的操作，因为超过了block的上限，会覆盖到原来位置的数，再次产生冲突不命中。

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。

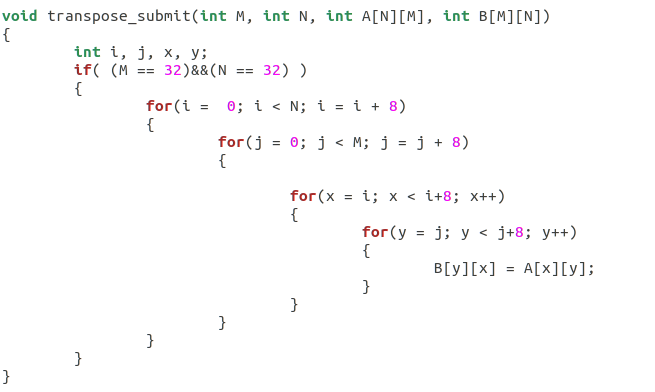
**【20%】分别按4和8分块编写transpose\_submit()代码，记录test-trans以M32N32矩阵下Miss数目结果；进一步编写代码重新处理相同下标的对角线上元素来再次优化，列表记录数组B第一个8\*8块写操作的命中情况。**

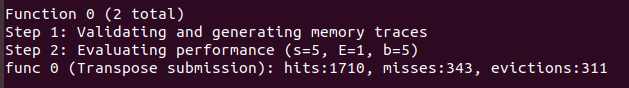
（1）4分块代码和miss结果





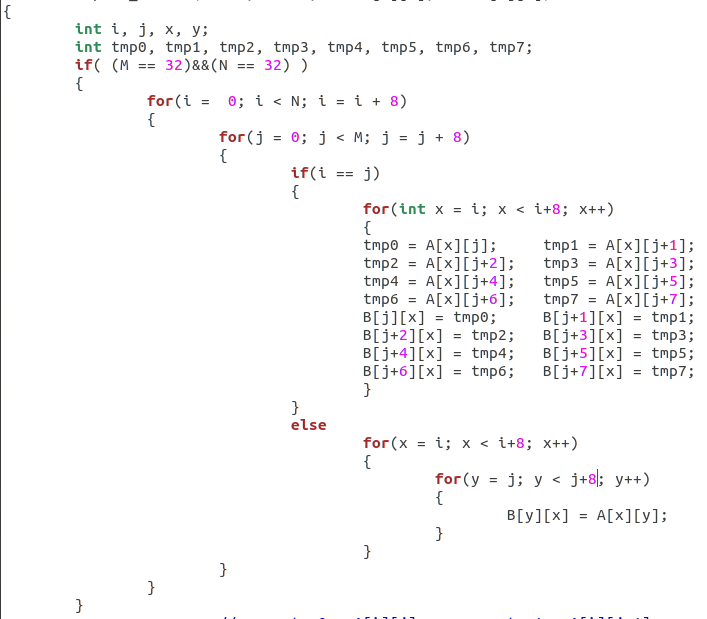
（2）8分块代码和miss结果

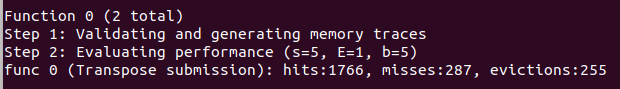




1. 对角线优化代码和miss结果

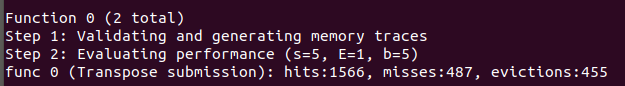
为减少冲突 定义8个临时变量 直接存储一个block中的所有数据



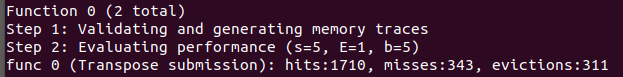


**【20%】分析采用分块技术后miss改善的原因。按4分块编写transpose\_submit()代码，记录test-trans以M32N32矩阵下miss数目的结果；以两个4\*4为例分析hit增多的原因。 注意：加载8\*4的矩阵A，存储到4\*8的矩阵B。**

（1）4\*4矩阵下miss数目的结果



（2）8\*4矩阵下miss数目的结果



（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分块中有很多多冲突不命中。