

华东师范大学软件学院实验报告

实验课程：计算机系统	年级：2023 级本科	实验成绩：
实验名称：Lab4 – Cache Lab	姓名：张梓卫	
实验编号：(4)	学号：10235101526	实验日期：2024/05/13
指导老师：肖波	组号：	

目录	
一 实验简介	1
二 实验前置准备	2
三 Part A 实验内容	2
1 csim-ref 命令行参数	2
2 模拟缓存行	3
3 代码实现	3
3.1 分配内存	3
3.2 判断缓存状态	4
3.3 执行 eviction 操作	4
3.4 执行读取操作	5
3.5 执行写入缓存操作	5
3.6 计数器	5
3.7 打印内存数据	5
3.8 主要执行函数	6
3.9 驱动函数	7
3.10 基本参数	8
4 实验结果	9
四 Part B 实验内容	9
1 简介	9
2 实验内容	10
2.1 Part B.1: 32 × 32 矩阵转置	10
2.2 Part B.2: 64 × 64 矩阵转置	11
2.3 Part B.3: 61 × 67 矩阵转置	13
五 实验总结	14
六 附录	14

一 实验简介

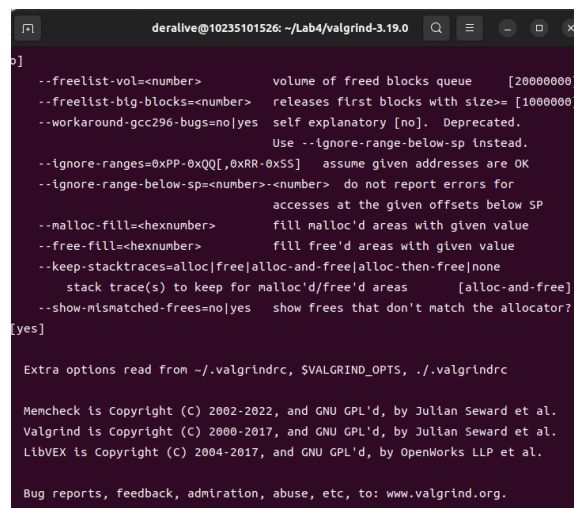
本实验是 CSAPP 第四章的实验，主要是学习了解程序的内存分配和释放机制。通过实验，学习如何使用 valgrind 工具来检测程序中的内存错误。本实验主要包括以下内容：

- Part A: 写一个使用 LRU 策略的 cache 模拟器
- Part B: 优化转置矩阵算法, 由 cache 模拟器处理后得到的 miss 数尽量小。

本实验的实验环境为 Ubuntu 20.04.2 LTS, 实验报告使用 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 撰写。

二 实验前置准备

- 解压 valgrind 包: `tar -jxvf valgrind-3.19.0.tar.bz2`
- 进入 valgrind 目录: `cd valgrind-3.19.0`
- 初始化配置: `./configure`
- 编译并安装: `make / & / sudo make install`
- 检查是否安装成功: `valgrind -h` 显示 valgrind 的参数及提示, 说明安装成功



```

deralive@10235101526: ~/Lab4/valgrind-3.19.0
[yes]
--freelist-vol=<number>          volume of freed blocks queue   [20000000]
--freelist-btg-blocks=<number>   releases first blocks with size=> [1000000]
--workaround-gcc296-bugs=no|yes  self explanatory [no].  Deprecated.
                                Use --ignore-range-below-sp instead.
--ignore-ranges=0xPP-0xQQ[,0xRR-0xSS]  assume given addresses are OK
--ignore-range-below-sp=<number>-<number> do not report errors for
                                accesses at the given offsets below SP
--malloc-fill=<hexnumber>        fill malloc'd areas with given value
--free-fill=<hexnumber>          fill free'd areas with given value
--keep-stacktraces=alloc|free|alloc-and-free|alloc-then-free|none
                                stack trace(s) to keep for malloc'd/free'd areas [alloc-and-free]
--show-mismatched-frees=no|yes   show frees that don't match the allocator?
[yes]

Extra options read from ~/.valgrindrc, $VALGRIND_OPTS, ~/.valgrindrc

Memcheck is Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
Valgrind is Copyright (C) 2000-2017, and GNU GPL'd, by Julian Seward et al.
LibVEX is Copyright (C) 2004-2017, and GNU GPL'd, by OpenWorks LLP et al.

Bug reports, feedback, admiration, abuse, etc, to: www.valgrind.org.

```

图 1: Valgrind 安装成功

三 Part A 实验内容

1 csim-ref 命令行参数

我们先要知道 csim-ref 的部分命令, 以便于后续进行实验

- -h: 可选的帮助标志, 用于打印使用情况信息
- -v: 显示跟踪信息的可选详细标志
- -s <s>: 设置的索引位数 ($S = 2s$ 是设置的数量)
- -E <E>: 关联性 (每组行数)
- -b : 块位数 ($B = 2b$ 是块大小)
- -t <tracefile>: 要重播的 valgrind 跟踪的名称
- Usage: `./csim-ref [-hv] -s <s> -E <E> -b -t <tracefile>`

2 模拟缓存行

首先，我们先了解 cache 的结构：

- cache 一般分为 S 组，每组有 E 块
- 每块结构为一个有效位 v，一个标志位 tag，一个数据块 data
- cache 地址分为三部分：标志位 tag，组索引 s，块偏移 b

根据缓存行相关的知识，我们定义了一个结构体，用于模拟缓存行的结构。

```
1 typedef struct {
2     int valid; // 有效位
3     ulong tag; // 标志位
4     clock_t time; // 时间
5 } CacheLine;
```

Struct Definition

通过二维数组的模拟，我们可以模拟 cache 的行为。

根据官方文档中的介绍，可以用 getopt() 函数来解析参数。

```
1 int main(int argc, char** argv){
2     int opt,x,y;
3     /* looping over arguments */
4     while((opt = getopt(argc, argv, "x:y:")) != -1){ // getopt() 函数解析参数
5         /* determine which argument it's processing */ // 确定正在处理的参数
6         switch(opt) { // 判断参数
7             case 'x':
8                 x = atoi(optarg); // 将参数转换为整数
9                 break;
10            case 'y':
11                y = atoi(optarg);
12                break;
13            default:
14                printf("wrong argument\n"); // 参数错误
15                break;
16        }
17    }
18 }
```

Getopt Function

3 代码实现

首先要分配内存，然后初始化 cache 行，最后模拟 cache 的行为。

3.1 分配内存

```
1 // 为缓存动态分配内存
2 CacheHead CacheInit(int S, int E) {
3     CacheHead cache; // 定义缓存
4     cache = calloc(1 << S, sizeof(CacheSet)); // 分配内存
5
6     if (cache == NULL) {
7         printf("Fail to allocate memory for cache.\n"); // 分配内存失败
8         exit(EXIT_FAILURE);
9     }
10
11     int i = 0; // 初始化
12     for (i = 0; i < 1 << S; i++) {
13         if ((cache[i] = calloc(E, sizeof(CacheLine))) == NULL) { // 分配内存
```

```

14     printf("Fail to allocate memory for cache.\n");
15     exit(EXIT_FAILURE); // 分配内存失败
16 }
17 }
18
19 for (i = 0; i < 1 << S; i++) {
20     int j;
21     for (j = 0; j < E; j++) {
22         cache[i][j].valid = 0; // 有效位
23     }
24 }
25 return cache;
26 }

```

Allocate Memory

3.2 判断缓存状态

```

1 // 判断缓存状态，是否有效，标记匹配
2 int CacheJudge(CacheHead cache, ulong index, ulong tag) {
3     int i;
4     int fullFlag = 1; // 标记是否满
5     int matchFlag = 0;
6     for (i = 0; i < globalOptions.associativity; i++) {
7         if (cache[index][i].valid == 0) { // 有效位为0
8             fullFlag = 0;
9         }
10        if (cache[index][i].tag == tag && cache[index][i].valid == 1) { // 有效位为1
11            matchFlag = 1;
12        }
13    }
14    if (matchFlag == 1) // 匹配
15        return CACHED;
16    if (fullFlag == 1)
17        return NEED_EVICT; // 需要驱逐
18    else
19        return NO_MATCH; // 不匹配
20 }

```

Judge The Situation Of Cache

3.3 执行 eviction 操作

```

1 // 判断缓存状态，是否有效，标记匹配
2 void CacheEvict(CacheHead cache, ulong index, ulong tag) { // 驱逐
3     int firstLine = 0, i = 0;
4     clock_t firstCachedTime = cache[index][i].time; // 时间
5     for (i = 0; i < globalOptions.associativity; i++) { // 遍历
6         if (cache[index][i].time < firstCachedTime) { // 时间比较
7             firstCachedTime = cache[index][i].time; // 更新时间
8             firstLine = i;
9         }
10    }
11    CacheLine *target = cache[index] + firstLine; // 目标
12    target->tag = 0; // 标志位
13    target->time = 0; // 时间
14    target->valid = 0; // 有效位
15 }

```

Eviction

3.4 执行读取操作

```
1 // 执行读取的操作
2 void CacheTouch(CacheHead cache, ulong index, ulong tag) {
3     int touchLine = 0; // 读取
4     while (cache[index][touchLine].tag != tag) { // 标志位不匹配
5         touchLine++; // 读取下一行
6     }
7     cache[index][touchLine].time = clock(); // 更新时间
8 }
```

Cache Touch

3.5 执行写入缓存操作

```
1 // 写入缓存
2 void CacheInsert(CacheHead cache, ulong index, ulong tag) {
3     int freeLine = 0, i;
4     for (i = 0; i < globalOptions.associativity; i++) {
5         if (cache[index][i].valid == 0) // 有效位为0
6             break; // 跳出循环
7         freeLine++; // 空行
8     }
9     CacheLine *target = cache[index] + freeLine; // 目标
10    target->tag = tag; // 标志位
11    target->valid = 1; // 有效位为1
12    target->time = clock(); // 更新时间
13 }
```

Cache Insert

3.6 计数器

```
1 // 计数器, 增加 hit, miss 和 eviction 的数量
2 void Adder(int type, int num) {
3     int v = globalOptions.verboseFlag; // 详细标志
4     switch (type) {
5     case ADD_EVICT:
6         totalEvictCount += num; // 驱逐
7         if (v && num != 0) // 详细标志
8             printf("eviction ");
9         break;
10    case ADD_HIT:
11        totalHitCount += num; // 命中
12        if (v && num != 0) // 详细标志
13            printf("hit ");
14        break;
15    case ADD_MISS:
16        totalMissCount += num; // 未命中
17        if (v && num != 0) // 详细标志
18            printf("miss ");
19    }
20 }
```

Counter

3.7 打印内存数据

```

1 // 逐字节以 16 进制打印内存数据
2 void printByte(bytept h, int len) {
3     int i;
4     for (i = 0; i < len; i++)
5         printf("%.2x ", h[i]); // 逐字节打印
6     printf("\n");
7 }

```

Print Byte

3.8 主要执行函数

```

1 void Execute(CacheHead cache, char type, ulong address, int len) {
2     ulong index = (address << globalOptions.tagBits) >> (MACHINE_BITS - globalOptions.setIndexBits); // 索引
3     ulong tag = address >> (globalOptions.blockBits + globalOptions.setIndexBits); // 标志
4     int status = CacheJudge(cache, index, tag); // 判断
5     if (globalOptions.verboseFlag == 1) {
6         if (globalOptions.superVerboseFlag == 1) {
7             printf("\n[address:] ");
8             printByte((bytept)&address, sizeof(long)); // 打印内存数据
9             printf("[index:] ");
10            printByte((bytept)&index, sizeof(long));
11            printf("[tag:] ");
12            printByte((bytept)&tag, sizeof(long));
13            printf("(Decimal)[index: %ld, tag: %ld]\n—————", index, tag);
14        }
15        else {
16            printf("(Decimal)[index: %ld, tag: %ld] ———", index, tag);
17        }
18    }
19    switch (status) { // 判断状态
20    case CACHED:
21        CacheTouch(cache, index, tag); // 读取
22        if (type == 'M') {
23            Adder(ADD_HIT, 1); // 命中
24            Adder(ADD_HIT, 1);
25        } else {
26            Adder(ADD_HIT, 1);
27        }
28        break;
29    case NO_MATCH:
30        CacheInsert(cache, index, tag); // 写入
31        if (type == 'M') {
32            Adder(ADD_MISS, 1);
33            Adder(ADD_HIT, 1);
34        } else {
35            Adder(ADD_MISS, 1);
36        }
37        break;
38    case NEED_EVICT:
39        CacheEvict(cache, index, tag); // 驱逐
40        CacheInsert(cache, index, tag); // 写入
41        if (type == 'M') {
42            Adder(ADD_MISS, 1);
43            Adder(ADD_EVICT, 1);
44            Adder(ADD_HIT, 1);
45        }
46        else {
47            Adder(ADD_MISS, 1);
48            Adder(ADD_EVICT, 1);
49        }
50        break;

```

```

51     default:
52         printf("Unknown error.\n");
53         exit(EXIT_FAILURE);
54     }
55     if (globalOptions.verboseFlag == 1) {
56         printf("\n");
57     }
58 }

```

Execute Function

3.9 驱动函数

```

1 // 读取参数，打开文件
2 int main(int argc, char *args[]) {
3     char ch;
4     while ((ch = getopt(argc, args, optString)) != -1) {
5         switch (ch) {
6             case 's':
7                 if (atoi(optarg) < 0) {
8                     printf("Unvalid input for <s>. Try Again.\n"); // 无效输入
9                     exit(EXIT_FAILURE); // 退出
10                }
11                globalOptions.setIndexBits = atoi(optarg); // 索引位数
12                break;
13             case 'E':
14                 if (atoi(optarg) < 0) {
15                     printf("Unvalid input for <E>. Try Again.\n"); // 无效输入
16                     exit(EXIT_FAILURE); // 退出
17                }
18                globalOptions.associativity = atoi(optarg); // 关联性
19                break;
20             case 'b':
21                 if (atoi(optarg) < 0) {
22                     printf("Unvalid input for <b>. Try Again.\n"); // 无效输入
23                     exit(EXIT_FAILURE); // 退出
24                }
25                globalOptions.blockBits = atoi(optarg); // 块位数
26                break;
27             case 't':
28                 globalOptions.traceDir = optarg; // 跟踪文件
29                 break;
30             case 'v':
31                 globalOptions.verboseFlag = 1; // 详细标志
32                 break;
33             case 'h':
34                 usage();
35                 exit(EXIT_FAILURE); // 退出
36             case 'V':
37                 globalOptions.verboseFlag = 1; // 详细标志
38                 globalOptions.superVerboseFlag = 1; // 详细标志
39                 break;
40             default:
41                 usage(); // 使用
42                 exit(EXIT_FAILURE); // 退出
43                 break;
44         }
45     }
46     globalOptions.tagBits = MACHINE_BITS - globalOptions.blockBits - globalOptions.setIndexBits; // 标志位
47
48     FILE *traceFile = fopen(globalOptions.traceDir, "r"); // 打开文件
49     if (traceFile == NULL) {
50         printf("Fail to open file: %s\n", globalOptions.traceDir); // 打开文件失败

```

```

51     exit(EXIT_FAILURE);
52 }
53 CacheHead cache = CacheInit(globalOptions.setIndexBits, globalOptions.associativity); // 初始化
54 char traceLine[32];
55 while (fgets(traceLine, 32, traceFile) != NULL) { // 读取文件
56     char mode; // 模式
57     ulong address; // 地址
58     int len; // 长度
59     sscanf(traceLine, "%c %lx,%d", &mode, &address, &len); // 读取
60     if (mode == 'I')
61         continue;
62     if (globalOptions.verboseFlag == 1) {
63         printf("%c %lx,%d ", mode, address, len); // 打印
64     }
65     Execute(cache, mode, address, len); // 执行
66 }
67 printSummary(totalHitCount, totalMissCount, totalEvictCount); // 打印结果
68 free(cache);
69 return 0;
70 }

```

Main

3.10 基本参数

```

1 #define MACHINE_BITS 64 // 机器位数
2 #define NEED_EVICT -1 // 需要驱逐
3 #define NO_MATCH -2 // 不匹配
4 #define CACHED 1 // 命中
5 #define ADD_HIT 1 // 增加命中
6 #define ADD_MISS 2 // 增加未命中
7 #define ADD_EVICT 3 // 增加驱逐
8
9 int totalMissCount = 0; // 未命中
10 int totalHitCount = 0; // 命中
11 int totalEvictCount = 0; // 驱逐
12
13 typedef unsigned long ulong; // 无符号长整型
14 typedef unsigned char *bytept; // 无符号字符型指针
15 const char *optString = "s:E:b:t:hVv";
16
17 struct globalOptions {
18     int setIndexBits; // 索引位数
19     int associativity; // 关联性
20     int blockBits; // 块位数
21     int verboseFlag; // 详细标志
22     int tagBits; // 标志位
23     int superVerboseFlag; // 详细标志
24     char *traceDir; // 跟踪文件
25 } globalOptions;
26
27 struct result {
28     int hit; // 命中
29     int miss; // 未命中
30     int evict; // 驱逐
31 };
32
33 typedef struct {
34     int valid;
35     ulong tag;
36     clock_t time;
37 } CacheLine;
38

```



```
39 typedef CacheLine *CacheSet; // 缓存行
40 typedef CacheSet *CacheHead; // 缓存头
```

Parameter Definition

4 实验结果

我们使用 `csim - ref` 来测试我们的代码：将在 Windows 下编译好的文件放入 Linux 系统中，然后使用 `make clean` 和 `make` 来编译，最后使用 `./test - csim` 运行测试，结果如下：

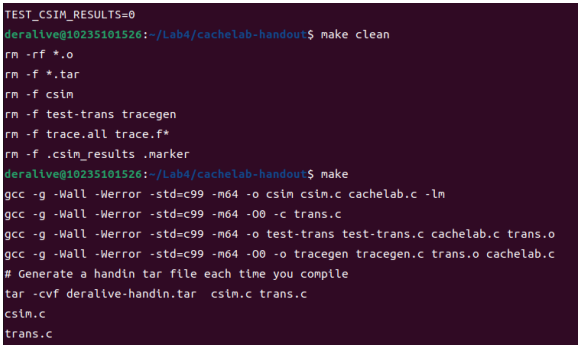


图 2: Makefile 编译成功

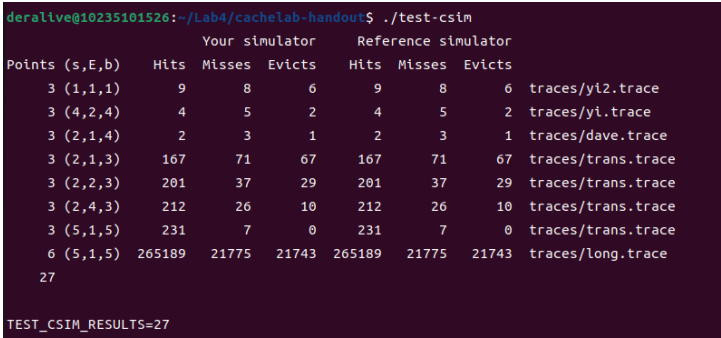


图 3: 跑分结果

四 Part B 实验内容

1 简介

按照官方文档的说明，需要在 `trans.c` 中写入一个优化的矩阵转置函数。尽可能地降低不命中率。使用命令：`./test - trans - M < rol > -N < col >` 可以查看这一转置函数的不命中数。生成的 `trace.fi` 文件还可以利用 PART A 写的缓存模拟器检查命中情况。

5.2.1 Performance (26 pts)

For each matrix size, the performance of your `transpose_submit` function is evaluated by using `valgrind` to extract the address trace for your function, and then using the reference simulator to replay this trace on a cache with parameters ($s = 5, E = 1, b = 5$).

Your performance score for each matrix size scales linearly with the number of misses, m , up to some threshold:

- 32×32 : 8 points if $m < 300$, 0 points if $m > 600$
- 64×64 : 8 points if $m < 1,300$, 0 points if $m > 2,000$
- 61×67 : 10 points if $m < 2,000$, 0 points if $m > 3,000$

Your code must be correct to receive any performance points for a particular size. Your code only needs to be correct for these three cases and you can optimize it specifically for these three cases. In particular, it is perfectly OK for your function to explicitly check for the input sizes and implement separate code optimized for each case.

图 4: 官方文档简介

2 实验内容

2.1 Part B.1: 32×32 矩阵转置

我们研究最简单的一个矩阵转置函数：

```
1 int temp; A[32][32], B[32][32];
2 for (int i = 0; i < N; i++) {
3     for (int j = 0; j < M; j++) {
4         tmp = A[i][j];
5         B[j][i] = tmp;
6     }
7 }
```

Matrix Transpose

这是最简单的矩阵转置函数，它具有超过 1000 的 *Miss* 数量。我们可以通过优化这个函数来减少 *Miss* 数量。

所以可以利用矩阵分块的思想。每一行数组都可以被存入 4 个缓存行中，一共有 32 个缓存行，所以每过 8 行就会出现一次和前面相同的组索引，发生 *miss* 和 *eviction*。所以考虑将 32×32 的矩阵分成 16 个 8×8 的矩阵，每一次都将一行的 8 个 *int* 分别存储进 $t1 - t4$ 。即，将矩阵划分成如下结构：

1	2	3	4
5	6	7	8
9	10	11	12

表 1: Cache Struct (其中每一个小块都是 8×8)

```
1  /*
2  * transpose_submit - This is the solution transpose function that you
3  *   will be graded on for Part B of the assignment. Do not change
4  *   the description string "Transpose submission", as the driver
5  *   searches for that string to identify the transpose function to
6  *   be graded.
7  */
8  char transpose_submit_desc[] = "Transpose submission";
9  void transpose_submit(int M, int N, int A[N][M], int B[M][N])
10 {
11     if (N == 32 && M == 32)
12     {
13         int i, j, k;
14         int t1, t2, t3, t4, t5, t6, t7, t8;
15         for (i = 0; i < 32; i += 8)
16         {
17             for (j = 0; j < 32; j += 8)
18             {
```

```

19     for (k = 0; k < 8; k++)
20     {
21         t1 = A[i + k][j];
22         t2 = A[i + k][j + 1];
23         t3 = A[i + k][j + 2];
24         t4 = A[i + k][j + 3];
25         t5 = A[i + k][j + 4];
26         t6 = A[i + k][j + 5];
27         t7 = A[i + k][j + 6];
28         t8 = A[i + k][j + 7];
29         B[j][i + k] = t1;
30         B[j + 1][i + k] = t2;
31         B[j + 2][i + k] = t3;
32         B[j + 3][i + k] = t4;
33         B[j + 4][i + k] = t5;
34         B[j + 5][i + k] = t6;
35         B[j + 6][i + k] = t7;
36         B[j + 7][i + k] = t8;
37     }
38 }
39 }
40 }
41 }

```

Matrix Transpose

实验结果如下所示：

```

deralive@10235101526:~/Lab4/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:1765, misses:288, evictions:256

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:869, misses:1184, evictions:1152

Summary for official submission (func 0): correctness=1 misses=288

TEST_TRANS_RESULTS=1:288

```

图 5: Part B.1 结果

2.2 Part B.2: 64×64 矩阵转置

大小变成了 64×64 ，每过 4 行就会出现一次冲突的情况。

所以可以先分成 8×8 的块，然后再把 8×8 的块分成 4 个 4×4 的块。

我们这里需要优化的不是程序运行的时间，而是 cache 的命中率。具体是每一个 8×8 块中，分别移动 4×4 的块，这样可以减少冲突。

- 将 8×8 fence 分成 4×4 的 fence，再将角块分为 $A B C D$;
- 将 A 区的第一行缓存到 A' 和 C' 的第一列
- 全部存到 A' 和 C' 后，再将 C' 的第一列存到 $Temp1 \& Temp2 \& Temp3 \& Temp4$ 变量当中
- 手动交换 C' 区和 B' 区的各个数值

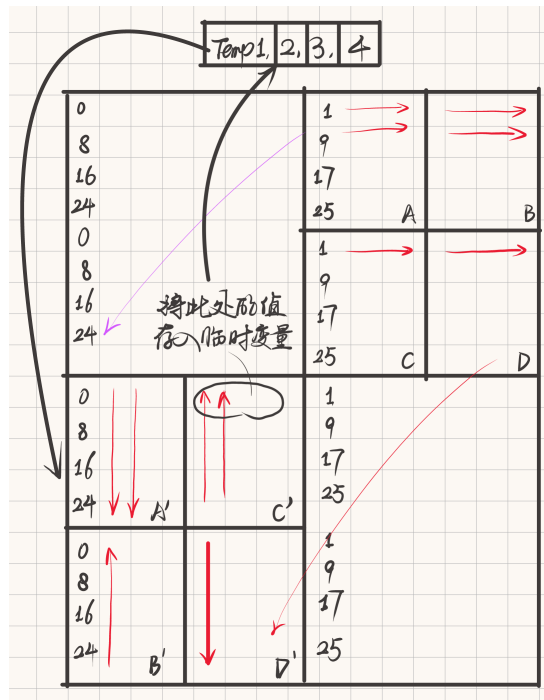


图 6: Analysis Process

```

1  else if (N == 64 && M == 64)
2  {
3      int t0, t1, t2, t3, t4, t5, t6, t7;
4      for (int i = 0; i < N; i += 8)
5      {
6          for (int j = 0; j < M; j += 8)
7          {
8              for (int k = i; k < i + 4; k++)
9              {
10                 t0 = A[k][j];
11                 t1 = A[k][j + 1];
12                 t2 = A[k][j + 2];
13                 t3 = A[k][j + 3];
14                 t4 = A[k][j + 4];
15                 t5 = A[k][j + 5];
16                 t6 = A[k][j + 6];
17                 t7 = A[k][j + 7];
18                 B[j][k] = t0;
19                 B[j + 1][k] = t1;
20                 B[j + 2][k] = t2;
21                 B[j + 3][k] = t3;
22                 B[j + 0][k + 4] = t7;
23                 B[j + 1][k + 4] = t6;
24                 B[j + 2][k + 4] = t5;
25                 B[j + 3][k + 4] = t4;
26             }
27             for (int h = 0; h < 4; h++)
28             {
29                 t0 = A[i + 4][j + 3 - h];
30                 t1 = A[i + 5][j + 3 - h];
31                 t2 = A[i + 6][j + 3 - h];
32                 t3 = A[i + 7][j + 3 - h];
33                 t4 = A[i + 4][j + 4 + h];
34                 t5 = A[i + 5][j + 4 + h];
35                 t6 = A[i + 6][j + 4 + h];
36                 t7 = A[i + 7][j + 4 + h];
37                 B[j + 4 + h][i + 0] = B[j + 3 - h][i + 4];

```

```

38         B[j + 4 + h][i + 1] = B[j + 3 - h][i + 5];
39         B[j + 4 + h][i + 2] = B[j + 3 - h][i + 6];
40         B[j + 4 + h][i + 3] = B[j + 3 - h][i + 7];
41         B[j + 3 - h][i + 4] = t0;
42         B[j + 3 - h][i + 5] = t1;
43         B[j + 3 - h][i + 6] = t2;
44         B[j + 3 - h][i + 7] = t3;
45         B[j + 4 + h][i + 4] = t4;
46         B[j + 4 + h][i + 5] = t5;
47         B[j + 4 + h][i + 6] = t6;
48         B[j + 4 + h][i + 7] = t7;
49     }
50 }
51 }
52 }

```

Matrix Transpose

实验结果如下所示：

```

deralve@10235101526:~/Lab4/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:9001, misses:1244, evictions:1212

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:3473, misses:4724, evictions:4692

Summary for official submission (func 0): correctness=1 misses=1244

TEST_TRANS_RESULTS=1:1244
deralve@10235101526:~/Lab4/cachelab-handout$

```

图 7: Part B.2 结果

2.3 Part B.3: 61×67 矩阵转置

这里的矩阵大小是 61×67 ，这个矩阵的大小是一个奇数，所以不能被整除。而且也不是 8 的倍数，所以在行与行之间没有特别明显的冲突不命中的关系。可以尝试用分块矩阵的方式优化。经过尝试 8×8 的分块和 16×16 的分块后，发现使用 16×16 的分块方式可以将 *Miss* 数降低到 2000 以下。

代码如下所示：

```

1 else {
2     int i, j, k, h;
3     for (i = 0; i < N; i += 16) {
4         for (j = 0; j < M; j += 16) {
5             for (k = i; k < i + 16 && k < N; k++) {
6                 for (h = j; h < j + 16 && h < M; h++) {
7                     B[h][k] = A[k][h];
8                 }
9             }
10        }
11    }
12 }

```

Matrix Transpose

实验结果如下所示：

```

deralive@10235101526:~/Lab4/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:6186, misses:1993, evictions:1961

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:3755, misses:4424, evictions:4392

Summary for official submission (func 0): correctness=1 misses=1993

TEST_TRANS_RESULTS=1:1993

```

图 8: Part B.3 结果

五 实验总结

通过这次实验，我学习了如何通过优化矩阵转置函数来提高缓存的命中率。通过分块的方式，我们可以将不命中率降低到 2000 以下。这次实验让我更深入地了解了缓存的工作原理，也让我对缓存的优化有了更深的认识。更重要的是，我学会了 LaTeX，我会继续使用它来写实验报告。

六 附录

实验的源代码如下所示：

```

1  /*
2  * trans.c - Matrix transpose B = A^T
3  *
4  * Each transpose function must have a prototype of the form:
5  * void trans(int M, int N, int A[N][M], int B[M][N]);
6  *
7  * A transpose function is evaluated by counting the number of misses
8  * on a 1KB direct mapped cache with a block size of 32 bytes.
9  */
10 #include <stdio.h>
11 #include "cachelab.h"
12
13 int is_transpose(int M, int N, int A[N][M], int B[M][N]);
14
15 /*
16 * transpose_submit - This is the solution transpose function that you
17 * will be graded on for Part B of the assignment. Do not change
18 * the description string "Transpose submission", as the driver
19 * searches for that string to identify the transpose function to
20 * be graded.
21 */
22 char transpose_submit_desc[] = "Transpose submission";
23 void transpose_submit(int M, int N, int A[N][M], int B[M][N])
24 {
25     if (N == 32 && M == 32)
26     {
27         int i, j, k;
28         int t1, t2, t3, t4, t5, t6, t7, t8;
29         for (i = 0; i < 32; i += 8)
30         {
31             for (j = 0; j < 32; j += 8)
32             {
33                 for (k = 0; k < 8; k++)
34

```

```

35         t1 = A[i + k][j];
36         t2 = A[i + k][j + 1];
37         t3 = A[i + k][j + 2];
38         t4 = A[i + k][j + 3];
39         t5 = A[i + k][j + 4];
40         t6 = A[i + k][j + 5];
41         t7 = A[i + k][j + 6];
42         t8 = A[i + k][j + 7];
43         B[j][i + k] = t1;
44         B[j + 1][i + k] = t2;
45         B[j + 2][i + k] = t3;
46         B[j + 3][i + k] = t4;
47         B[j + 4][i + k] = t5;
48         B[j + 5][i + k] = t6;
49         B[j + 6][i + k] = t7;
50         B[j + 7][i + k] = t8;
51     }
52 }
53 }
54 }
55 else if (N == 64 && M == 64)
56 {
57     int t0, t1, t2, t3, t4, t5, t6, t7;
58     for (int i = 0; i < N; i += 8)
59     {
60         for (int j = 0; j < M; j += 8)
61         {
62             for (int k = i; k < i + 4; k++)
63             {
64                 t0 = A[k][j];
65                 t1 = A[k][j + 1];
66                 t2 = A[k][j + 2];
67                 t3 = A[k][j + 3];
68                 t4 = A[k][j + 4];
69                 t5 = A[k][j + 5];
70                 t6 = A[k][j + 6];
71                 t7 = A[k][j + 7];
72                 B[j][k] = t0;
73                 B[j + 1][k] = t1;
74                 B[j + 2][k] = t2;
75                 B[j + 3][k] = t3;
76                 B[j + 0][k + 4] = t7;
77                 B[j + 1][k + 4] = t6;
78                 B[j + 2][k + 4] = t5;
79                 B[j + 3][k + 4] = t4;
80             }
81             for (int h = 0; h < 4; h++)
82             {
83                 t0 = A[i + 4][j + 3 - h];
84                 t1 = A[i + 5][j + 3 - h];
85                 t2 = A[i + 6][j + 3 - h];
86                 t3 = A[i + 7][j + 3 - h];
87                 t4 = A[i + 4][j + 4 + h];
88                 t5 = A[i + 5][j + 4 + h];
89                 t6 = A[i + 6][j + 4 + h];
90                 t7 = A[i + 7][j + 4 + h];
91                 B[j + 4 + h][i + 0] = B[j + 3 - h][i + 4];
92                 B[j + 4 + h][i + 1] = B[j + 3 - h][i + 5];
93                 B[j + 4 + h][i + 2] = B[j + 3 - h][i + 6];
94                 B[j + 4 + h][i + 3] = B[j + 3 - h][i + 7];
95                 B[j + 3 - h][i + 4] = t0;
96                 B[j + 3 - h][i + 5] = t1;
97                 B[j + 3 - h][i + 6] = t2;
98                 B[j + 3 - h][i + 7] = t3;

```

```

99         B[j + 4 + h][i + 4] = t4;
100         B[j + 4 + h][i + 5] = t5;
101         B[j + 4 + h][i + 6] = t6;
102         B[j + 4 + h][i + 7] = t7;
103     }
104 }
105 }
106 }
107 else
108 {
109     int i, j, k, h;
110     for (i = 0; i < N; i += 16)
111     {
112         for (j = 0; j < M; j += 16)
113         {
114             for (k = i; k < i + 16 && k < N; k++)
115             {
116                 for (h = j; h < j + 16 && h < M; h++)
117                 {
118                     B[h][k] = A[k][h];
119                 }
120             }
121         }
122     }
123 }
124 }
125
126 /*
127  * You can define additional transpose functions below. We've defined
128  * a simple one below to help you get started.
129  */
130
131 /*
132  * trans - A simple baseline transpose function, not optimized for the cache.
133  */
134 char trans_desc[] = "Simple row-wise scan transpose";
135 void trans(int M, int N, int A[N][M], int B[M][N])
136 {
137     int i, j, tmp;
138
139     for (i = 0; i < N; i++)
140     {
141         for (j = 0; j < M; j++)
142         {
143             tmp = A[i][j];
144             B[j][i] = tmp;
145         }
146     }
147 }
148
149 /*
150  * registerFunctions - This function registers your transpose
151  * functions with the driver. At runtime, the driver will
152  * evaluate each of the registered functions and summarize their
153  * performance. This is a handy way to experiment with different
154  * transpose strategies.
155  */
156 void registerFunctions()
157 {
158     /* Register your solution function */
159     registerTransFunction(transpose_submit, transpose_submit_desc);
160
161     /* Register any additional transpose functions */
162     registerTransFunction(trans, trans_desc);

```



```
163 }
164
165 /*
166  * is_transpose — This helper function checks if B is the transpose of
167  *     A. You can check the correctness of your transpose by calling
168  *     it before returning from the transpose function.
169  */
170 int is_transpose(int M, int N, int A[N][M], int B[M][N])
171 {
172     int i, j;
173
174     for (i = 0; i < N; i++)
175     {
176         for (j = 0; j < M; ++j)
177         {
178             if (A[i][j] != B[j][i])
179             {
180                 return 0;
181             }
182         }
183     }
184     return 1;
185 }
```

Coding in C