Lab4 - Cache Lab Report

`csim.c`

实验思想：采用封装的思想，提高函数的复用性，有助于理解总体代码。

前置函数：定义在`cachelab.h`：

```c
/*
 * printSummary - This function provides a standard way for your cache
 * simulator * to display its final hit and miss statistics
 */
void printSummary(int hits,  /* number of  hits */
                  int misses, /* number of misses */
                  int evictions); /* number of evictions */
```

首先我们先构思出建立一个缓存需要哪些部分。这包括：

- 缓存的基本数据结构
- 从命令行获得缓存`s` `E` `b`和读写的`t`信息
- 初始化缓存
- 获取文件读写信息
- 访问缓存
- 增加`LRU`次数
- 释放Cache

于是我们先搭建总体框架。其中函数的返回值和参数暂时不完善，我们之后按需求修改。

```c
#include "cachelab.h"

uint64_t hits = 0;
uint64_t misses = 0;
uint64_t evictions = 0;

typedef struct {
    int valid;
    uint64_t tag;
    uint64_t lru;
} Line;

typedef struct {
    Line *lines;
} Set;

typedef struct {
    Set *sets;
    uint64_t s;
    uint64_t E;
    uint64_t b;
} Cache;

void getCommandInfo(uint64_t s, uint64_t E, uint64_t b) {

}

Cache initCache() {

}

void accessCache(Cache *cache, uint64_t address) {

}

void updateLRU(Cache *cache) {

}

void getFileInfo(char *tracefile) {

}

void freeCache(Cache *cache) {

}
```

```c
int main(int argc, char *argv[]) {

    printSummary(hits, misses, evictions);

}
```

接下来我们逐个分析。

`getCommandInfo`

这个函数的目的是为了读取命令行输入的指令，获得缓存 `s` `E` `b` 和读写的 `t` 信息。

```
Usage: ./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>
```

我们可以使用 `getopt`。这个头文件一般只在Linux下使用。`getopt.h` 必须要包括，否则会报错，原因未知。

```c
#include <unistd.h>
#include <getopt.h>
int getopt(int argc, char * const argv[], const char *optstring);
```

特殊变量:

- `optarg`：当前选项的参数值（如 -b value 中的 value）
- `optind`：下一个要处理的 argv 索引
- `opterr`：设为 0 可禁止错误信息输出

`optstring` 规则:

- 字符：表示允许的选项（如 "ab:c"）
  - 'a'：无参数
  - 'b:'：需要参数
  - 'c'：无参数

这个函数很明显比较适合使用在 `main` 函数中，所以我们把 `getCommandInfo` 的功能放在 `main` 函数中。

```c
int main(int argc, char *argv[]) {
    uint64_t s;
    uint64_t E;
    uint64_t b;
    char tracefile[fileNameLength];

    char opt;
    while ((opt = getopt(argc, argv, "s:E:b:t:")) != -1) {
        switch (opt) {
            case 's':
                s = atoi(optarg);
                break;
            case 'E':
                E = atoi(optarg);
                break;
            case 'b':
                b = atoi(optarg);
                break;
            case 't':
                strcpy(tracefile, optarg);
                break;
        }
    }

    printSummary(hits, misses, evictions);

}
```

这样我们就完成了获取命令行指令的函数。

`initCache`

我们已经获得了`s` `E` `b`信息，我们可以开始初始化缓存了。我们需要给cache的`s` `E` `b`赋值，并且开辟内存空间，而且需要使每一路的`valid` `tag` `lru`都设置为0。

```
Cache initCache(uint64_t s, uint64_t E, uint64_t b) {
    int S = 1 << s;

    Cache cache;
    cache.s = s;
    cache.E = E;
    cache.b = b;

    cache.sets = (Set *)malloc(S * sizeof(Set));
    for (int i = 0; i < S; i++) {
        cache.sets[i].lines = (Line *)malloc(E * sizeof(Line));
        for (int j = 0; j < E; j++) {
            cache.sets[i].lines[j].valid = 0;
            cache.sets[i].lines[j].tag = 0;
            cache.sets[i].lines[j].lru = 0;
        }
    }

    return cache;
}
```

在`main`函数中新增：

```
Cache cache;
cache = initCache(s, E, b);
```

`getFileInfo`

```
 <操作类型> <地址>,<大小>
 L 10,4
 S 18,4
 L 20,4
```

我们只需要处理`L`(load) `S`(store) `M`(modify load/store)，不需要理会`L`。其中`M`进行了两次的内存访问。

为了安全地从文件中读取`uint64_t`，可以如下操作：

```
#include <inttypes.h>
fscanf(fp, " %c %" SCNx64 ",%d", &opt, &address, %size) == 3
```

完整的函数如下：

```c
void getFileInfo(char *tracefile, Cache *cache) {
    FILE *fp = fopen(tracefile, "r");
    if (fp == NULL) {
        return;
    }

    char opt;
    uint64_t address;
    int size;

    while (fscanf(fp, " %c %" SCNx64 ",%d", &opt, &address, &size) == 3) {
        switch (opt) {
            case 'L':
                accessCache(cache, address);
                break;
            case 'S':
                accessCache(cache, address);
                break;
            case 'M':
                accessCache(cache, address);
                accessCache(cache, address);
                break;
        }
        updateLRU(cache);
    }
}
```

在`main`函数加上：

```c
getFileInfo(tracefile, &cache);
```

接下来我们需要完成`accessCache`和`updateLRU`，后者较为简单，我们从易到难。

`updateLRU`

实现很简单，每次进行了一次操作都需要更新`valid`位置不为0的`LRU`，以便于我们找到使用最不频繁的路进行替换。

```
void updateLRU(Cache *cache) {
    int S = 1 << cache->s;
    int E = cache->E;

    for (int i = 0; i < S; i++) {
        for (int j = 0; j < E; j++) {
            if (cache->sets[i].lines[j].valid) {
                cache->sets[i].lines[j].lru++;
            }
        }
    }
}
```

accessCache

这个函数比较复杂。首先我们需要处理传入的地址，区分出`tag` `index` `block_offset`，只不过对于这个实验`block_offset`可以不考虑。之后访问相对应的缓存路，分为三类情况：

- **命中了**：比对`tag`并且确认`valid`不为0。再把缓存的`lru`设置为0，表示其刚刚被访问了，`hits++`。
- **未命中且有空的路**：加载到空的路中，`tag`更新，`valid = 1`，`misses++`。
- **未命中且没有空的路**：加载到`lru`最大的路中，`tag`更新，`valid = 1`，`lru = 0`，`evictions++`，`misses++`。

```c
void accessCache(Cache *cache, uint64_t address) {
    uint64_t s = cache->s;
    uint64_t b = cache->b;

    uint64_t tag = address >> (s + b);

    uint64_t mask = UINT64_MAX;
    mask >>= 64 - (s + b);
    uint64_t temp = address & mask;

    uint64_t index = temp >> b;

    int isEmpty = 0;
    for (int i = 0; i < cache->E; i++) {
        if (cache->sets[index].lines[i].valid && cache->sets[index].lines[i].tag == tag) {
            cache->sets[index].lines[i].lru = 0;
            hits++;
            return;
        }

        if (cache->sets[index].lines[i].valid == 0) {
            isEmpty = 1;
        }
    }

    if (isEmpty) {
        for (int i = 0; i < cache->E; i++) {
            if (cache->sets[index].lines[i].valid == 0) {
                cache->sets[index].lines[i].valid = 1;
                cache->sets[index].lines[i].tag = tag;
                misses++;
                return;
            }
        }
    } else {
        uint64_t MaxLRU = 0;
        for (int i = 0; i < cache->E; i++) {
            MaxLRU = cache->sets[index].lines[i].lru > MaxLRU ? cache->sets[index].lines[i].lru
        }

        for (int i = 0; i < cache->E; i++) {
            if (MaxLRU == cache->sets[index].lines[i].lru) {
                cache->sets[index].lines[i].lru = 0;
                cache->sets[index].lines[i].tag = tag;
                misses++;
                evictions++;
```

```
                return;
            }
        }
    }
}
```

`freeCache`

释放内存即可，逐层释放。

```
void freeCache(Cache *cache) {
    int S = 1 << cache->s;
    for (int i = 0; i < S; i++) {
        free(cache->sets[i].lines);
    }
    free(cache->sets);
}
```

在 `main` 函数加上：

```
freeCache(&cache);
```

完整代码

```c
#include "cachelab.h"
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include <inttypes.h>
#include <stdio.h>
#include <getopt.h>

#define fileNameLength 1000

uint64_t hits = 0;
uint64_t misses = 0;
uint64_t evictions = 0;

typedef struct {
    int valid;
    uint64_t tag;
    uint64_t lru;
} Line;

typedef struct {
    Line *lines;
} Set;

typedef struct {
    Set *sets;
    uint64_t s;
    uint64_t E;
    uint64_t b;
} Cache;

Cache initCache(uint64_t s, uint64_t E, uint64_t b) {
    int S = 1 << s;

    Cache cache;
    cache.s = s;
    cache.E = E;
    cache.b = b;

    cache.sets = (Set *)malloc(S * sizeof(Set));
    for (int i = 0; i < S; i++) {
        cache.sets[i].lines = (Line *)malloc(E * sizeof(Line));
        for (int j = 0; j < E; j++) {
```

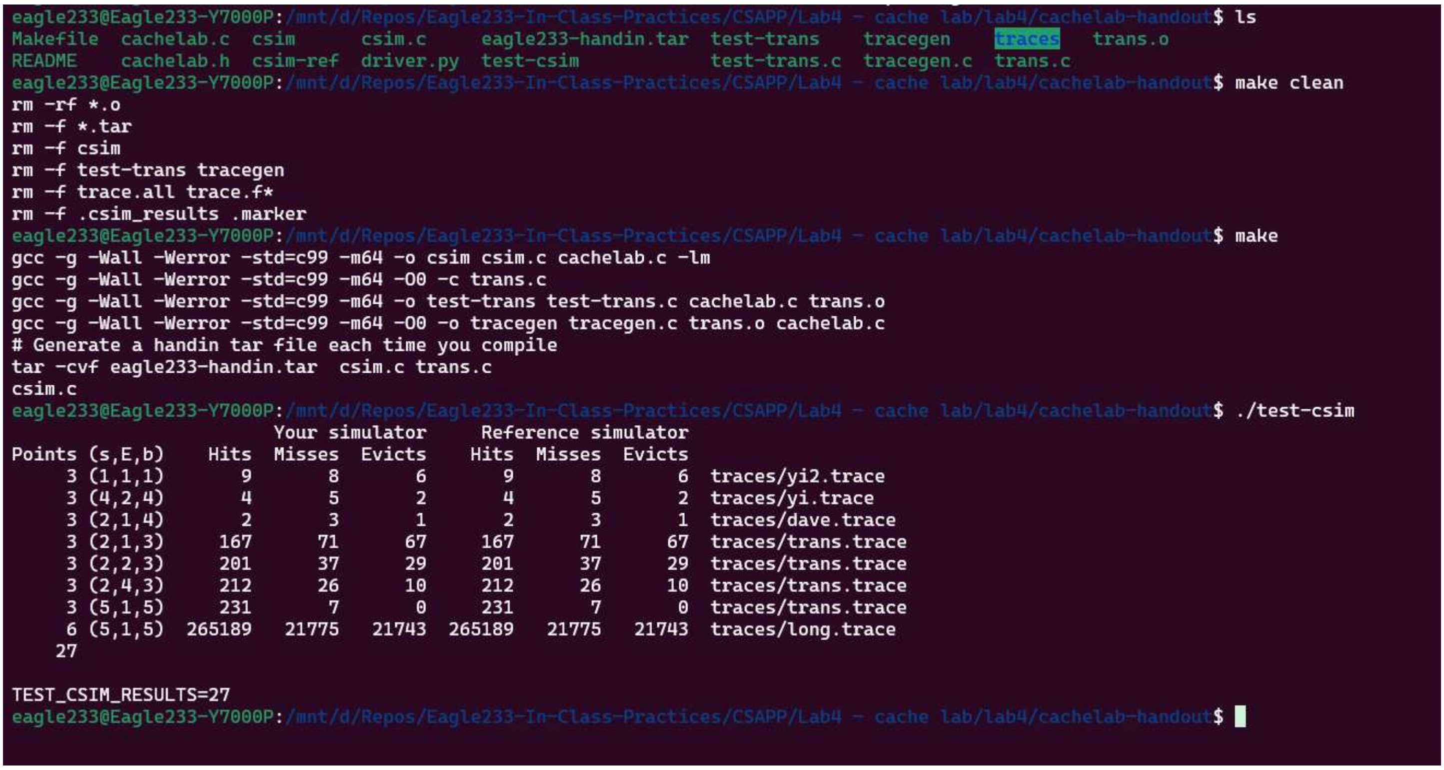
```c
            cache.sets[i].lines[j].valid = 0;
            cache.sets[i].lines[j].tag = 0;
            cache.sets[i].lines[j].lru = 0;
        }
    }

    return cache;
}

void accessCache(Cache *cache, uint64_t address) {
    uint64_t s = cache->s;
    uint64_t b = cache->b;

    uint64_t tag = address >> (s + b);

    uint64_t mask = UINT64_MAX;
    mask >>= 64 - (s + b);
    uint64_t temp = address & mask;

    uint64_t index = temp >> b;

    int isEmpty = 0;
    for (int i = 0; i < cache->E; i++) {
        if (cache->sets[index].lines[i].valid && cache->sets[index].lines[i].tag == tag) {
            cache->sets[index].lines[i].lru = 0;
            hits++;
            return;
        }

        if (cache->sets[index].lines[i].valid == 0) {
            isEmpty = 1;
        }
    }

    if (isEmpty) {
        for (int i = 0; i < cache->E; i++) {
            if (cache->sets[index].lines[i].valid == 0) {
                cache->sets[index].lines[i].valid = 1;
                cache->sets[index].lines[i].tag = tag;
                misses++;
                return;
            }
        }
    } else {
        uint64_t MaxLRU = 0;
        for (int i = 0; i < cache->E; i++) {
```

```
            MaxLRU = cache->sets[index].lines[i].lru > MaxLRU ? cache->sets[index].lines[i].lru
        }

        for (int i = 0; i < cache->E; i++) {
            if (MaxLRU == cache->sets[index].lines[i].lru) {
                cache->sets[index].lines[i].lru = 0;
                cache->sets[index].lines[i].tag = tag;
                misses++;
                evictions++;
                return;
            }
        }
    }
}

void updateLRU(Cache *cache) {
    int S = 1 << cache->s;
    int E = cache->E;

    for (int i = 0; i < S; i++) {
        for (int j = 0; j < E; j++) {
            if (cache->sets[i].lines[j].valid) {
                cache->sets[i].lines[j].lru++;
            }
        }
    }
}

void getFileInfo(char *tracefile, Cache *cache) {
    FILE *fp = fopen(tracefile, "r");
    if (fp == NULL) {
        return;
    }

    char opt;
    uint64_t address;
    int size;

    while (fscanf(fp, " %c %" SCNx64 ",%d", &opt, &address, &size) == 3) {
        switch (opt) {
            case 'L':
                accessCache(cache, address);
                break;
            case 'S':
                accessCache(cache, address);
                break;
```

```
            case 'M':
                accessCache(cache, address);
                accessCache(cache, address);
                break;
        }
        updateLRU(cache);
    }
}

void freeCache(Cache *cache) {
    int S = 1 << cache->s;
    for (int i = 0; i < S; i++) {
        free(cache->sets[i].lines);
    }
    free(cache->sets);
}

int main(int argc, char *argv[]) {
    uint64_t s;
    uint64_t E;
    uint64_t b;
    char tracefile[fileNameLength];

    char opt;
    while ((opt = getopt(argc, argv, "s:E:b:t:")) != -1) {
        switch (opt) {
            case 's':
                s = atoi(optarg);
                break;
            case 'E':
                E = atoi(optarg);
                break;
            case 'b':
                b = atoi(optarg);
                break;
            case 't':
                strcpy(tracefile, optarg);
                break;
        }
    }

    Cache cache;
    cache = initCache(s, E, b);

    getFileInfo(tracefile, &cache);
```

```
    freeCache(&cache);

    printSummary(hits, misses, evictions);
}
```

跑分截图



```
eagle233@Eagle233-Y7000P:/mnt/d/Repos/Eagle233-In-Class-Practices/CSAPP/Lab4 - cache lab/lab4/cachelab-handout$ ls
Makefile  cachelab.c  csim      csim.c     eagle233-handin.tar  test-trans    tracegen    traces   trans.o
README    cachelab.h  csim-ref  driver.py  test-csim            test-trans.c  tracegen.c  trans.c
eagle233@Eagle233-Y7000P:/mnt/d/Repos/Eagle233-In-Class-Practices/CSAPP/Lab4 - cache lab/lab4/cachelab-handout$ make clean
rm -rf *.o
rm -f *.tar
rm -f csim
rm -f test-trans tracegen
rm -f trace.all trace.f*
rm -f .csim_results .marker
eagle233@Eagle233-Y7000P:/mnt/d/Repos/Eagle233-In-Class-Practices/CSAPP/Lab4 - cache lab/lab4/cachelab-handout$ make
gcc -g -Wall -Werror -std=c99 -m64 -o csim csim.c cachelab.c -lm
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 eagle233-handin.tar  csim.c trans.c
csim.c
eagle233@Eagle233-Y7000P:/mnt/d/Repos/Eagle233-In-Class-Practices/CSAPP/Lab4 - cache lab/lab4/cachelab-handout$ ./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

TEST_CSIM_RESULTS=27
eagle233@Eagle233-Y7000P:/mnt/d/Repos/Eagle233-In-Class-Practices/CSAPP/Lab4 - cache lab/lab4/cachelab-handout$
```

`trans.c`

实验思路：先分块后优化

根据实验指导书，我们知道：`s = 5, E = 1, b = 5`，我们重点观察`b = 5`，因为这意味着我们的缓存一个组（全相联）最多可以缓存`32bits == 4bytes`，刚好是8个`int`。

我们在`tracegen.c`中发现：

```c
/* Markers used to bound trace regions of interest */
volatile char MARKER_START, MARKER_END;

static int A[256][256];
static int B[256][256];
static int M;
static int N;
```

矩阵定义的大小是`256*256 == 65536`个`int`，刚好是缓存大小的整数倍。这意味着我们原始的转置函数：

```
/*
 * trans - A simple baseline transpose function, not optimized for the cache.
 */
char trans_desc[] = "Simple row-wise scan transpose";
void trans(int M, int N, int A[N][M], int B[M][N])
{
    int i, j, tmp;

    for (i = 0; i < N; i++) {
        for (j = 0; j < M; j++) {
            tmp = A[i][j];
            B[j][i] = tmp;
        }
    }

}
```

会使得矩阵`A` `B`重复使用同一块缓存的同一块区域，造成抖动极大。我们要尽可能减少冲突不命中和容量不命中，冷不命中是无法避免的。

我们的缓存共有32组，每组可以缓存8个整型，接下来分别分析题目要求的三种情况。

32 × 32

这个矩阵的每一行都需要4组(32/8)缓存，缓存一共可以容纳8行。我们可以使用**8 × 8**分块的方式来转置。原因如下：

- **空间局部性好**：每次访问都刚好读取八个元素进入缓存。
- **避免冲突未命中**：由于每个分块距离都较远，不容易发生冲突未命中。

可以写出如下的代码：

```c
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 k = 0; k < M; k += 8) {
            for (int l = 0; l < M; l += 8) {
                for (int i = k; i < k + 8; i++) {
                    for (int j = l; j < l + 8; j++) {
                        B[j][i] = A[i][j];
                    }
                }
            }
        }
    }
}
```

得到如下结果:

```
./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:1709, misses:344, evictions:312

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=344

TEST_TRANS_RESULTS=1:344
```

离答案要求的300并不遥远。我们想到实验允许我们使用12个局部变量,因此可以进行优化。我们注意到,当我们的分块矩阵在对角线上的时候,时常会发生**冲突不命中**。我们可以用局部变量保存冲突的元素,保证对角线上的分块矩阵不容易冲突。由于我们在循环之中已经用了4个变量,因此我们还能使用8个变量进行优化。

这样处理之后,由于我们预先访问了分块矩阵一行的元素,这里只会开销一个miss。之后每一行访问矩阵 `B`,各自出现8次misses,这样就会有 `(1 + 8) * 8 = 72` 次misses,相比于之前的一定是大幅减小,因为之前的方法要交替访问 `A` `B`,misses的次数要多得多。

```c
char transpose_submit_desc[] = "Transpose submission";
void transpose_submit(int M, int N, int A[N][M], int B[M][N]) {
    int i, j;
    if (M == 32) {
        for (int k = 0; k < M; k += 8) {
            for (int l = 0; l < M; l += 8) {
                if (k == l) {
                    for (i = k; i < k + 8; i++) {
                        int temp1 = A[i][l];
                        int temp2 = A[i][l + 1];
                        int temp3 = A[i][l + 2];
                        int temp4 = A[i][l + 3];
                        int temp5 = A[i][l + 4];
                        int temp6 = A[i][l + 5];
                        int temp7 = A[i][l + 6];
                        int temp8 = A[i][l + 7];

                        B[l][i] = temp1;
                        B[l + 1][i] = temp2;
                        B[l + 2][i] = temp3;
                        B[l + 3][i] = temp4;
                        B[l + 4][i] = temp5;
                        B[l + 5][i] = temp6;
                        B[l + 6][i] = temp7;
                        B[l + 7][i] = temp8;
                    }
                } else {
                    for (i = k; i < k + 8; i++) {
                        for (j = l; j < l + 8; j++) {
                            B[j][i] = A[i][j];
                        }
                    }
                }
            }
        }
    }
}
```

```
./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
```

64 × 64

对于这个矩阵，一次只能存四组了。因此我们可能会想到**4 × 4**的方法来分块。但这样很明显效率极其低下，因为有大量的缓存是没有被使用的，每组缓存中一半的缓存都是不被使用的。为了节省时间，这里不再去尝试**4 × 4**和**8 × 8**的结果。

对于分块矩阵，我们是否能对其再次分块？我们可以先分为**8 × 8**，再在内部分为**4 × 4**。不过这样的代码会比较不容易写，我们需要先对每个**4 × 4**的矩阵转置，再在 `B` 中交换第1、3象限的矩阵，这样才能完成矩阵的转置。我们先：

```
if (M == 64) {
    int k, l, i, j, x, y;
    for (k = 0; k < M; k += 8) {
        for (l = 0; l < M; l += 8) {
            for (i = k; i < k + 8; i += 4) {
                for (j = l; j < l + 8; j += 4) {
                    for (x = i; x < i + 4; x++) {
                        for (y = j; y < j + 4; y++) {
                            B[y][x] = A[x][y];
                        }
                    }
                }
            }
        }
    }
}
```

然后发现我们的代码超时了：

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

Function 0 (2 total)
Step 1: Validating and generating memory traces
Error: Program timed out.
TEST_TRANS_RESULTS=0:0
```

猜测原因是因为实验在`/mnt`下运行，把实验转移到Linux磁盘内试试。

```
eagle233@Eagle233-X16:~/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:6305, misses:1892, evictions:1860

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=1892

TEST_TRANS_RESULTS=1:1892
```

说明我们还需要进行优化。考虑到上一个矩阵利用变量进行优化的思想，我们要确保减少冲突不命中，那么就要优先使用刚刚使用的缓存。我们不仅仅要对$\mathbf{4 \times 4}$的转置进行优化，还要对第1、3象限的交换进行优化。

在处理$\mathbf{4 \times 4}$的一块时不能让`A`和`B`的数据在缓存里直接冲突，得提前规划好。我们分步读取`A`块，先把上半部分存好写入`B`块的临时位置，然后当读取`A`块下半部分时，我们边读边把`B`块右上角临时放的数据巧妙地挪到左下角该去的地方，这个挪动的顺序和读取`A`的方式都很讲究，都是为了利用那些刚刚访问的缓存块，避免冲突，最大化缓存效率。

```c
if (M == 64) {
    for (int i = 0; i < N; i += 8) {
        for (int j = 0; j < M; j += 8) {
            for (int k = i; k < i + 4; ++k) {
                int temp0 = A[k][j];
                int temp1 = A[k][j + 1];
                int temp2 = A[k][j + 2];
                int temp3 = A[k][j + 3];
                int temp4 = A[k][j + 4];
                int temp5 = A[k][j + 5];
                int temp6 = A[k][j + 6];
                int temp7 = A[k][j + 7];

                B[j][k]     = temp0;
                B[j + 1][k] = temp1;
                B[j + 2][k] = temp2;
                B[j + 3][k] = temp3;

                B[j][k + 4]     = temp7;
                B[j + 1][k + 4] = temp6;
                B[j + 2][k + 4] = temp5;
                B[j + 3][k + 4] = temp4;
            }

            for (int l = 0; l < 4; ++l) {
                int temp0 = A[i + 4][j + 3 - l];
                int temp1 = A[i + 5][j + 3 - l];
                int temp2 = A[i + 6][j + 3 - l];
                int temp3 = A[i + 7][j + 3 - l];
                int temp4 = A[i + 4][j + 4 + l];
                int temp5 = A[i + 5][j + 4 + l];
                int temp6 = A[i + 6][j + 4 + l];
                int temp7 = A[i + 7][j + 4 + l];

                B[j + 4 + l][i]     = B[j + 3 - l][i + 4];
                B[j + 4 + l][i + 1] = B[j + 3 - l][i + 5];
                B[j + 4 + l][i + 2] = B[j + 3 - l][i + 6];
                B[j + 4 + l][i + 3] = B[j + 3 - l][i + 7];

                B[j + 3 - l][i + 4] = temp0;
                B[j + 3 - l][i + 5] = temp1;
                B[j + 3 - l][i + 6] = temp2;
                B[j + 3 - l][i + 7] = temp3;

                B[j + 4 + l][i + 4] = temp4;
                B[j + 4 + l][i + 5] = temp5;
```

```
                B[j + 4 + l][i + 6] = temp6;
                B[j + 4 + l][i + 7] = temp7;
            }
        }
    }
}
```

```
eagle233@Eagle233-X16:~/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
```

61 × 67

这一题对miss的要求很低，因此我们可以大胆分一个比较大的块。经过测试，选取 **16 × 16**。

要注意矩阵 `A` 是N行M列，不要搞混行列。

```
if (M == 61) {
    for (int i = 0; i < M; i += 16) {
        for (int j = 0; j < N; j += 16) {
            for (int k = i; k < i + 16 && k < M; k++) {
                for (int l = j; l < j + 16 && l < N; l++) {
                    B[l][k] = A[k][l];
                }
            }
        }
    }
}
```

```
eagle233@Eagle233-X16:~/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
```

虽然有点含糊，但是通过要求还是很简单的。

完整代码

```
void transpose_submit(int M, int N, int A[N][M], int B[M][N]) {
    if (M == 32) {
        int i, j, k, l;
        int temp1, temp2, temp3, temp4, temp5, temp6, temp7, temp8;
        for (k = 0; k < M; k += 8) {
            for (l = 0; l < M; l += 8) {
                if (k == l) {
                    for (i = k; i < k + 8; i++) {
                        temp1 = A[i][l];
                        temp2 = A[i][l + 1];
                        temp3 = A[i][l + 2];
                        temp4 = A[i][l + 3];
                        temp5 = A[i][l + 4];
                        temp6 = A[i][l + 5];
                        temp7 = A[i][l + 6];
                        temp8 = A[i][l + 7];

                        B[l][i] = temp1;
                        B[l + 1][i] = temp2;
                        B[l + 2][i] = temp3;
                        B[l + 3][i] = temp4;
                        B[l + 4][i] = temp5;
                        B[l + 5][i] = temp6;
                        B[l + 6][i] = temp7;
                        B[l + 7][i] = temp8;
                    }
                } else {
                    for (i = k; i < k + 8; i++) {
                        for (j = l; j < l + 8; j++) {
                            B[j][i] = A[i][j];
                        }
                    }
                }
            }
        }
    }

    if (M == 64) {
        for (int i = 0; i < N; i += 8) {
            for (int j = 0; j < M; j += 8) {
                for (int k = i; k < i + 4; ++k) {
                    int temp0 = A[k][j];
                    int temp1 = A[k][j + 1];
                    int temp2 = A[k][j + 2];
```

```
                int temp3 = A[k][j + 3];
                int temp4 = A[k][j + 4];
                int temp5 = A[k][j + 5];
                int temp6 = A[k][j + 6];
                int temp7 = A[k][j + 7];

                B[j][k]     = temp0;
                B[j + 1][k] = temp1;
                B[j + 2][k] = temp2;
                B[j + 3][k] = temp3;

                B[j][k + 4]     = temp7;
                B[j + 1][k + 4] = temp6;
                B[j + 2][k + 4] = temp5;
                B[j + 3][k + 4] = temp4;
            }

            for (int l = 0; l < 4; ++l) {
                int temp0 = A[i + 4][j + 3 - l];
                int temp1 = A[i + 5][j + 3 - l];
                int temp2 = A[i + 6][j + 3 - l];
                int temp3 = A[i + 7][j + 3 - l];
                int temp4 = A[i + 4][j + 4 + l];
                int temp5 = A[i + 5][j + 4 + l];
                int temp6 = A[i + 6][j + 4 + l];
                int temp7 = A[i + 7][j + 4 + l];

                B[j + 4 + l][i]     = B[j + 3 - l][i + 4];
                B[j + 4 + l][i + 1] = B[j + 3 - l][i + 5];
                B[j + 4 + l][i + 2] = B[j + 3 - l][i + 6];
                B[j + 4 + l][i + 3] = B[j + 3 - l][i + 7];

                B[j + 3 - l][i + 4] = temp0;
                B[j + 3 - l][i + 5] = temp1;
                B[j + 3 - l][i + 6] = temp2;
                B[j + 3 - l][i + 7] = temp3;

                B[j + 4 + l][i + 4] = temp4;
                B[j + 4 + l][i + 5] = temp5;
                B[j + 4 + l][i + 6] = temp6;
                B[j + 4 + l][i + 7] = temp7;
            }
        }
    }
}
```

```c
    if (M == 61) {
        for (int i = 0; i < N; i += 16) {
            for (int j = 0; j < M; j += 16) {
                for (int k = i; k < i + 16 && k < N; k++) {
                    for (int l = j; l < j + 16 && l < M; l++) {
                        B[l][k] = A[k][l];
                    }
                }
            }
        }
    }
}
```

两个文件的跑分截图

```
eagle233@Eagle233-X16:~/cachelab-handout$ python2 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         288
Trans perf 64x64           8.0         8        1244
Trans perf 61x67          10.0        10        1993
          Total points    53.0        53
eagle233@Eagle233-X16:~/cachelab-handout$
```
