CacheLab 实验报告

涂奕腾 2020201018

1. Part1

主要分为以下几个部分：

(1)cache 结构的定义和初始化：

```
typedef struct {
    int valid;
    unsigned tag;
    int timestamp;
}line;

line** cache;

void init(){
    cache = (line**)malloc(sizeof(line*) * S);
    for(int i = 0; i < S ; ++i)
        *(cache + i) = (line*)malloc(sizeof(line) * E);
    for(int i = 0; i < S; ++i)
        for(int j = 0; j < E; ++j){
            cache[i][j].valid = 0;
            cache[i][j].tag = 0;
            cache[i][j].timestamp = 0;
        }
}
```

(2)解析命令行输入

调用<getopt>库函数解析即可。

```
void getoptions(int argc, char* argv[]){
    int opt;
    while((opt = getopt(argc,argv,"hvs:E:b:t:")) !=-1){
        switch(opt){
        case 'h':
            printHelp();
            exit(0);
        case 'v':
           ver = 1;//ver 表示是否输出信息
           break;
        case 's':
            s=atoi(optarg);
            S =  1<<s;
            break;
        case 'E':
            E = atoi(optarg);
```

```
            break;
        case 'b':
            b = atoi(optarg);
            break;
        case 't':
            fp = fopen(optarg, "r");
            if(fp == NULL){
               fprintf(stderr, "Open file failed");
               exit(-1);
            }
            break;
        default:
            break;
        }
    }
}
```

(3)测试数据输入

这里我采用的是逐行按照字符串处理，其中 I 操作加载指令不用管，L 数据加载和 S 数据储存更新 cache 一次，M 数据修改更新 cache 两次。

```
void solve(){
    ++timestamp;
    char opt;
    unsigned address;
    int siz;
    int tmp = sscanf(Str," %c %x,%d", &opt, &address, &siz);
    if(!tmp) return;
    if(ver) printf("%s", Str);
    switch(opt){
        case 'L':
            update(address);
            break;
        case 'M':
            update(address);
        case 'S':
            update(address);
            break;
        default:
            break;
    }
    if(ver) printf("\n");
}

void input(){
    while(fgets(Str, 256, fp)){
       if(Str[strlen(Str) - 1] == '\n') Str[strlen(Str) - 1] = '\0';
```

```
        solve();
    }
}
```

(4)模拟 cache

每次更新时先读取组号 s 和标记位 tag，在组中查找是否有满足条件的标记位：

①若有效位为 1 且标记位相符，则找到命中的位置；

②若有效位为 0，则找到一个空位置；

③若有效位为 1，则找时间戳最小的替换位置。

对于①(命中)增加一次 hit 次数，更新时间戳即可；

否则不命中，增加 miss 次数。此时若有空位置，直接占据；否则冲突，增加 eviction 次数再替换数据。

```
void update(unsigned address){
    unsigned setaddress = (address >> b) & (0xffffffff >> (32 - s));
    unsigned tagaddress = address >> (s + b);
    int Hit = -1, Empty = -1, Rep = -1;
    for(int i = 0; i < E ; ++i){
        if(!~Hit && cache[setaddress][i].valid &&
cache[setaddress][i].tag == tagaddress) Hit = i;
        if(!~Empty && !cache[setaddress][i].valid) Empty = i;
        if(cache[setaddress][i].valid && (!~Rep ||
cache[setaddress][Rep].timestamp > cache[setaddress][i].timestamp)) Rep =
i;
    }
    if(~Hit){
        ++hit;
        if(ver) printf(" hit");
        cache[setaddress][Hit].timestamp = timestamp;
    }
    else{
        ++miss;
        if(ver) printf(" miss");
        if(~Empty){
            cache[setaddress][Empty].valid = 1;
            cache[setaddress][Empty].tag = tagaddress;
            cache[setaddress][Empty].timestamp = timestamp;
        }
        else{
            if(!~Rep){
                printf("LRU Error\n");
               exit(-1);
            }
            ++eviction;
            if(ver) printf(" eviction");
            cache[setaddress][Rep].tag = tagaddress;
            cache[setaddress][Rep].timestamp = timestamp;
```

```
        }
    }
}
```

2. Part2

知 s=5, E=1, b=5，即有 32 组直接映射高速缓存，每个 block 大小为 32bytes，可容纳 8 个 int 类型的整数。

(1)M=32, N=32

画出简单的 cache 与内存对应位置示意图：

M=32,N=32					
cache	0	1	2	…	31
	A[0][0]-A[0][7]	A[0][8]-A[0][15]	A[0][16]-A[0][23]	…	A[7][24]-A[7][31]
	A[8][0]-A[8][7]	A[8][8]-A[8][15]	A[8][16]-A[8][23]	…	A[15][24]-A[15][31]
	A[16][0]-A[16][7]	A[16][8]-A[16][15]	A[16][16]-A[16][23]	…	A[23][24]-A[23][31]
	A[24][0]-A[24][7]	A[24][8]-A[24][15]	A[24][16]-A[24][23]	…	A[31][24]-A[31][31]
	B[0][0]-B[0][7]	B[0][8]-B[0][15]	B[0][16]-B[0][23]	…	B[7][24]-B[7][31]
	B[8][0]-B[8][7]	B[8][8]-B[8][15]	B[8][16]-B[8][23]	…	B[15][24]-B[15][31]
	B[16][0]-B[16][7]	B[16][8]-B[16][15]	B[16][16]-B[16][23]	…	B[23][24]-B[23][31]
	B[24][0]-B[24][7]	B[24][8]-B[24][15]	B[24][16]-B[24][23]	…	B[31][24]-B[31][31]

对 A 数组逐行访问共产生 32*4=128 次 miss，对 B 数组逐列访共产生 32*32 次 miss，显然无法满足要求。根据 ppt 的提示，我们使用分块的方法，限制写入 B 矩阵的行数，充分利用 B 矩阵在缓存中的部分。

由于 A 矩阵在第 9 行出现了冲突，为了使统一矩阵在缓存中的内容不被相互替换，我们尝试 8*8 分块每个块的大小为 8*8，miss 次数为 8。分析冲突次数：

1.对于非对角线上的块：每次转置 A、B 两矩阵的 miss 次数为 8*2=16。

2.对于对角线上的块：

①转置对角线上元素 A[i][i]和 B[i][i]时，A[i]会被 B[i]取代，**随后复制 A[i][i+1]时 A[i]又会取代 B[i]**,这样就会产生两次多余的 miss(除了每个块中第一行 B[i]为第一次加载，最后一行 A[i]不需要重新加载，分别只产生一次多余 miss)，额外产生 2*6+2=14 次 miss

②B[i]被 A[i]取代后，除了最后一行外，**在下一行 A[i+1][i]转置时需要重新加载 B[i]**，又额外产生 7 次 miss。

故转置一个对角线上的块总 miss 次数为 16+14+7=37。

由上述得到，直接分块产生的总 miss 数为 37*4+16*12=340，还达不到要求。原因是对角线上的块中缓存相互替换的次数较多，我们可以考虑用本地变量存下 A[i]的一行后再复制给 B，由于本地变量储存在寄存器上，这样就减少了缓存块的替换次数，即 A[i]和 B[i]不需重新加载(上文中两处**加粗**部分)，这样一来，一个对角线上的块 miss 次数为 37-2*7=23，总 miss 次数为 23*4+16*12=284，满足要求。

```
for(int i = 0; i < M; i += 8){
```

```
    for(int j = 0; j < N; j += 8){
        for(int k = i; k < i + 8; k++){
            a0 = A[k][j];
            a1 = A[k][j+1];
            a2 = A[k][j+2];
            a3 = A[k][j+3];
            a4 = A[k][j+4];
            a5 = A[k][j+5];
            a6 = A[k][j+6];
            a7 = A[k][j+7];
            B[j][k] = a0;
            B[j+1][k] = a1;
            B[j+2][k] = a2;
            B[j+3][k] = a3;
            B[j+4][k] = a4;
            B[j+5][k] = a5;
            B[j+6][k] = a6;
            B[j+7][k] = a7;
        }
    }
}
```

实际运行结果 miss 次数为 287，满足要求。查阅资料后得知，多出来的 3 次 miss 可能是程序进行了 3 次额外内存访问产生的固定偏差。

```
[2020201018@work122 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:1766, misses:287, evictions:255

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=287
```

在此基础上，我们对程序进行进一步改进。联想到课本上矩阵乘法相关内容，我们可以将 A，B 均按行处理，只复制元素不转置，等到块中内容复制完成后，再在 B 的块里面完成转置，这样就能完全消除对角线上 A[i]与 B[i]的相互替换，且在转置时缓存块中正好存储了 8 行 B 的内容，转置过程不会产生 miss。这样总 miss 次数为 16*16=256(+3=259)

```
for(int i = 0; i < M; i += 8){
    for(int j = 0; j < N; j += 8){
```

```
        for(int k = i, l = j; k < i + 8; k++, l++){
            a0 = A[k][j];
            a1 = A[k][j+1];
            a2 = A[k][j+2];
            a3 = A[k][j+3];
            a4 = A[k][j+4];
            a5 = A[k][j+5];
            a6 = A[k][j+6];
            a7 = A[k][j+7];
            B[l][i] = a0;
            B[l][i + 1] = a1;
            B[l][i + 2] = a2;
            B[l][i + 3] = a3;
            B[l][i + 4] = a4;
            B[l][i + 5] = a5;
            B[l][i + 6] = a6;
            B[l][i + 7] = a7;
        }
        for(int k = 0; k < 8; k++){
            for(int l = k + 1; l < 8; l++){
                a0 = B[k + j][l + i];
                B[k + j][l + i] = B[l + j][k + i];
                B[l + j][k + i] = a0;
            }
        }
    }
}
```

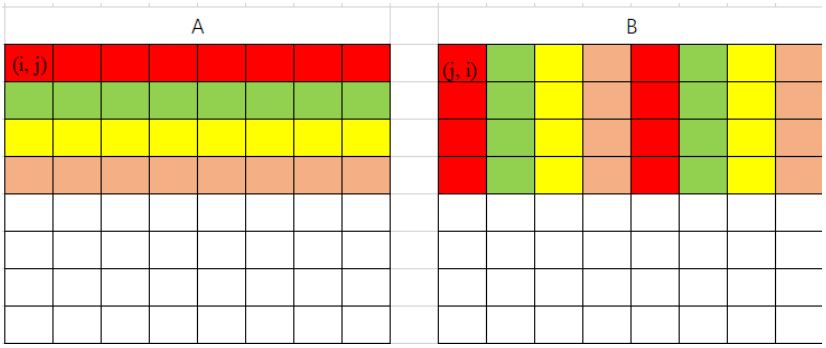
```
[2020201018@work122 cachelab-handout]$ ./test-trans -M 32 -N 32

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:3586, misses:259, evictions:227

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=259
```

(2)M=64, N=64

在 64*64 的情况下，一个 cache 可以容纳矩阵的 4 行，如果直接按照 8*8 的分块，一个分块内就会产生冲突，于是我们将前四行和后四行分别处理。

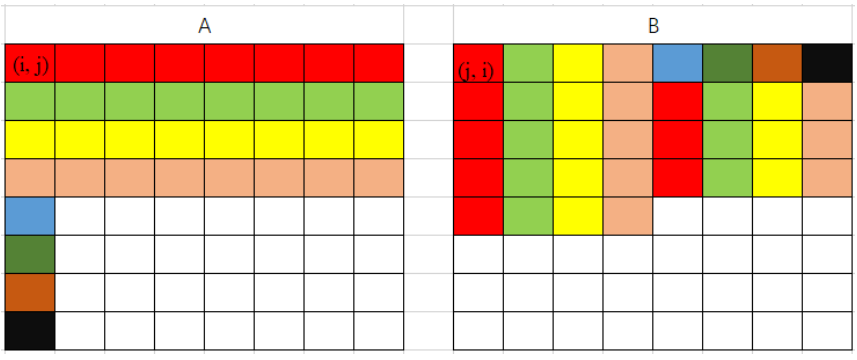
cache	0	1	2	…	31
	A[0][0]-A[0][7]	A[0][8]-A[0][15]	A[0][16]-A[0][23]	…	A[3][56]-A[3][63]
	A[4][0]-A[4][7]	A[4][8]-A[4][15]	A[4][16]-A[4][23]	…	A[7][56]-A[7][63]
	…	…	…	…	…

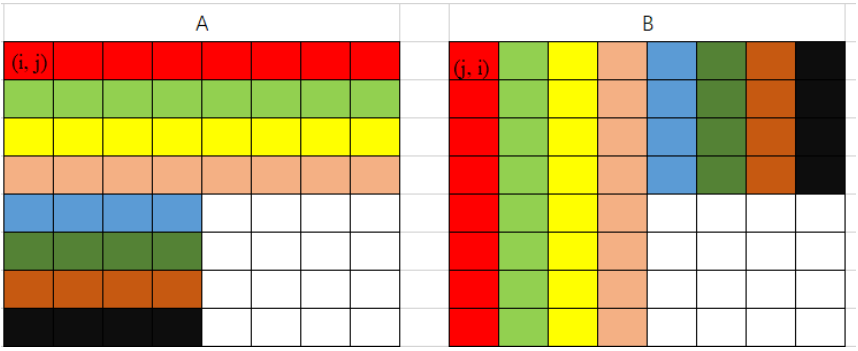
1.对于前四行，我们先按照如图的方式将其复制到 B 的前四行中以减少后四行载入增加 miss 次数：




```
for(int k = i;k < i + 4; k++){
    a0 = A[k][j];
    a1 = A[k][j + 1];
    a2 = A[k][j + 2];
    a3 = A[k][j + 3];
    a4 = A[k][j + 4];
    a5 = A[k][j + 5];
    a6 = A[k][j + 6];
    a7 = A[k][j + 7];
    B[j][k] = a0;
    B[j + 1][k] = a1;
    B[j + 2][k] = a2;
    B[j + 3][k] = a3;
    B[j][k + 4] = a4;
    B[j + 1][k + 4] = a5;
    B[j + 2][k + 4] = a6;
    B[j + 3][k + 4] = a7;
}
```

2.对于后四行的前四列，逐行转置：






```
for(int k = j; k < j + 4; k++){
    a0 = A[i + 4][k];
    a1 = A[i + 5][k];
    a2 = A[i + 6][k];
    a3 = A[i + 7][k];
    a4 = B[k][i + 4];
    a5 = B[k][i + 5];
    a6 = B[k][i + 6];
    a7 = B[k][i + 7];
    B[k][i + 4] = a0;
    B[k][i + 5] = a1;
    B[k][i + 6] = a2;
    B[k][i + 7] = a3;
    B[k + 4][i] = a4;
    B[k + 4][i + 1] = a5;
    B[k + 4][i + 2] = a6;
    B[k + 4][i + 3] = a7;
}
```

3.对于后四行的后四列，同 1 直接转置即可

```
for(int k = i + 4; k < i + 8; k++){
    a0 = A[k][j + 4];
    a1 = A[k][j + 5];
    a2 = A[k][j + 6];
    a3 = A[k][j + 7];
    B[j + 4][k] = a0;
    B[j + 5][k] = a1;
    B[j + 6][k] = a2;
    B[j + 7][k] = a3;
}
```

```
[2020201018@work122 cachelab-handout]$ ./test-trans -M 64 -N64

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:9066, misses:1179, evictions:1147

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:3474, misses:4723, evictions:4691

Summary for official submission (func 0): correctness=1 misses=1179
```

(3)M=61, N=67

2000 次的 miss 要求很松，类似(1)，我使用的是 18*18 普通分块。

```
for(int i = 0; i < M; i += 18){
    for(int j = 0; j < N; j += 18){
        for(int k  = j; k < j + 18 && k < N; ++k){
            for(int l = i; l < i + 18 && l < M; ++l){
                a0 = A[k][l];
                B[l][k] = a0;
            }
        }
    }
}
```

```
[2020201018@work122 cachelab-handout]$ ./test-trans -M 61 -N 67

Function 0 (2 total)
Rhythmbox idating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:6354, misses:1825, evictions:1793

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:3756, misses:4423, evictions:4391

Summary for official submission (func 0): correctness=1 misses=1825
```

最终结果：

```
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         259
Trans perf 64x64           8.0         8        1179
Trans perf 61x67          10.0        10        1825
          Total points    53.0        53
```