Project 2: Understanding Cache Memories

516030910125 Yining Liu jadyn24@outlook.com

1. Introduction

This lab consists of two parts.

Task of Part A is to implement a cache simulator with C. The simulator should be able to take a *valgrind* memory trace as input and simulate its hit/miss/evict behavior as a cache, then output the number of hits, misses, and evictions.

Task of Part B is to optimize a small matrix transpose function, aiming to minimize the number of cache misses. Particularly, the code only needs to be correct for 3 specific cases, so optimize the function for them with different strategies is enough.

2. Experiments

2.1 Part A

2.1.1 Analysis

Simulations for structure and behaviors of a cache are both needed.

For structure, use a 3-level *struct* to simulate *"cache-set-line"*, and record the number of sets in the cache and the number of lines in each set for further convenience.

For behaviors, it should be able to initialize, accept input parameters and deal with different instructions with LRU replacement policy. So, the simulator must detect misses and update LRU number timely. Notice that only behaviors is concerned, so we can implement functions of different instructions in almost the same way. "-*v*" is also implemented to track the trace.

Difficult points are distributed down to details, including methods of calculating set index and tag, finding a new minimum when evictions happen and arranging outputs for *v*-module.

2.1.2 Code

```
#include "cachelab.h"
#include <stdio.h>
#include <unistd.h>
#include <getopt.h>
#include <stdlib.h>


typedef struct{
    int valid;
    int tag;
    int LRU;
}Line;
```

```c
typedef struct{
    Line* lines;
}Set;

typedef struct {
    int setNum;
    int lineNum;
    Set* sets;
}Cache;

Cache cache;
int hits,misses,evictions;

int findMin(int set);
void initCache(int s,int E,int b);
void sim(int addr,int size,int set,int tag ,int verbose);

int main(int argc, char** argv){
    int s,E,b,verbose=0;
    int addr,size;
    int set,tag;
    char opt;
    char *tracefile;

    //read-in parameters
    while((opt = getopt(argc,argv,"vs:E:b:t:"))!=-1){
        switch(opt){
            case 'v': verbose = 1; break;
            case 's': s = atoi(optarg); break;
            case 'E': E = atoi(optarg); break;
            case 'b': b = atoi(optarg); break;
            case 't': tracefile = optarg;
        }
    }

    //init
    initCache(s,E,b);
    //operating
    freopen(tracefile,"r",stdin);  //open tracefile
    while(scanf("%c %x,%d",&opt,&addr,&size) != EOF){
         if(opt=='L'||opt=='S'||opt=='M'){  //ignore "I"
            set = (addr>>b)&((1<<s)-1);  //get set index
            tag = addr>>(s+b);           //get tag

         if(verbose==1) printf("%c %x,%d ",opt,addr,size);  //%x for hex
         if(opt=='L'||opt=='S') sim(addr,size,set,tag,verbose);//load or store
         else if(opt=='M') {
            sim(addr,size,set,tag,verbose);    //load
            sim(addr,size,set,tag,verbose);    //store
         }
         if(verbose==1) printf("\n");
         }
    }

    //print result
    printSummary(hits,misses,evictions);

    return 0;
}

void sim(int addr,int size,int set,int tag ,int verbose){
    int i,j;
    int miss = 1,full = 1;

    //miss or not
    for(i=0;i<cache.lineNum;i++){
        if(cache.sets[set].lines[i].valid==1&&cache.sets[set].lines[i].tag==tag)
{  //hit!
            miss = 0;
            //update LRU
            cache.sets[set].lines[i].LRU = 2333;
            for(j=0;j<cache.lineNum;j++){
                if(j!=i) cache.sets[set].lines[j].LRU--;
            }
        }
    }
```

```
    if(!miss){
        hits++;
        if(verbose == 1) printf("hit ");
    }
    else{
        misses++;
        if(verbose == 1) printf("miss ");

        //this set: full or not
        for(i=0;i<cache.lineNum;i++){
            if(cache.sets[set].lines[i].valid==0){
                full = 0;
                break;
            }
        }
        if(full){
            i = findMin(set); ////eviction happens
            evictions++;
            if(verbose == 1) printf("eviction ");
        }
        cache.sets[set].lines[i].valid = 1;
        cache.sets[set].lines[i].tag = tag;
        cache.sets[set].lines[i].LRU = 2333;
        for(j=0;j<cache.lineNum;j++){
            if(j!=i) cache.sets[set].lines[j].LRU--;
        }
    }
}

// find new minimum LRU
int findMin(int set){
    int i;
    int minIdx = 0, minLRU=2333;

    for(i=0;i<cache.lineNum;i++){
        if(cache.sets[set].lines[i].LRU<minLRU){
            minIdx = i;
            minLRU = cache.sets[set].lines[i].LRU;
        }
    }
    return minIdx;
}

void initCache(int s,int E,int b){
    int i ,j;

    cache.setNum = 2 << s;
    cache.lineNum = E;
    cache.sets = (Set *)malloc(cache.setNum * sizeof(Set));

    for(i=0;i<cache.setNum;i++){
        cache.sets[i].lines = (Line *)malloc(E*sizeof(Line));
        for(j=0;j<E;j++){
            cache.sets[i].lines[j].valid = 0;
            cache.sets[i].lines[j].LRU = 0;
        }
    }
}
```

2.1.3 Evaluation

Test of verbose mode with *yi.trace*, identical to performance of *csim-ref*.

```
lynx@lynx-virtual-machine:~/Desktop/proj2$ ./csim -v -s 4 -E 1 -b 4 -t traces/yi.trace
L 10,1 miss
M 20,1 miss hit
L 22,1 hit
S 18,1 hit
L 110,1 miss eviction
L 210,1 miss eviction
M 12,1 miss eviction hit
hits:4 misses:5 evictions:3
```

Run *driver.py* to compare with *csim-ref*, get identical results.

```
lynx@lynx-virtual-machine:~/Desktop/proj2$ ./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
```

2.2 Part B

2.2.1 Analysis

The evaluation standard for Part B is not very intuitive only with the reference of *simple row-wise scan transpose*, so get down to a quite straight-forward strategy at first to see what happens.

Divide and conquer: for a 32*32 matrix, spilt it into four 8*8 blocks and do the row-wise transpose. Variable *i, j, m, n* for index, and tuple *tmp* of size 8 for temps. Result in 289 misses for the first try (full marks).

For size 64*64, strategy above is not efficient enough with the performance of about 3,000 misses. Divide an 8*8 further into four 4*4 blocks. To make full use of every loading, do row-wise transpose with the upper-left block and the upper-right block independently, then transpose upper-right with upper-left, and do transpose with lower-right block at last. Result in 1277 misses with this strategy (full marks).

61*67 is easier, almost the same to a 32*32 matrix. Only need to pay attention to the overflow of index, because neither 61 nor 67 is a multiple of 8. Result in 1891 misses (full marks).

2.2.2 Code

```
char transpose_submit_desc[] = "Transpose submission";
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    int i,j,m,n;
    int tmp[8];

    //32*32 = 4*(8*8), simply use tmps to avoid conflicts between A & B
    if(M==32&&N==32){
        for(i=0;i<N;i+=8){
            for(j=0;j<M;j+=8){
                for(n=i;n<i+8;n++){
                    for(m=0;m<8;m++){
                        tmp[m] = A[n][j+m];
                    }
                    for(m=0;m<8;m++){
                        B[j+m][n] = tmp[m];
                    }
                }
            }
        }
    }
```

```
    //64*64: most complicated among 3 cases.
    if(M==64&&N==64){
        for(i=0;i<N;i+=8){
            for(j=0;j<M;j+=8){
                for(n=i;n<i+4;n++){
                    for(m=0;m<8;m++){
                        tmp[m] = A[n][j+m];
                    }
                    for(m=0;m<4;m++){
                        B[j+m][n] = tmp[m];
                        B[j+m][n+4] = tmp[m+4];  //upper right
                    }
                }
                for(m=j;m<j+4;m++){
                    for(n=4;n<8;n++){
                        tmp[n] = B[m][i+n];
                    }
                    for(n=4;n<8;n++){
                        B[m][i+n] = A[i+n][m];
                    }
                    for(n=0;n<4;n++){
                        B[m+4][i+n] = tmp[4+n]; //evictions here
                    }
                }
                for(n=i+4;n<i+8;n++){  //lower right
                    for(m=j;m<j+4;m++){
                        B[m+4][n] = A[n][m+4];
                    }
                }
            }
        }
    }

    //61*67: similar to 32*32, set blocks to 16*8, see if n<67 in loops
    if(M==61&&N==67){
        for(i=0;i<N;i+=16){
            for(j=0;j<M;j+=8){
                for(n=i;n<i+16&&n<N;n++){
                    for(m=0;m<8;m++){
                        tmp[m] = A[n][j+m];
                    }
                    for(m=0;m<8;m++){
                        B[j+m][n] = tmp[m];
                    }
                }
            }
        }
    }
}
```

2.2.3 Evaluation

Run *driver.py*, all results meet the standard.

```
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         289
Trans perf 64x64           8.0         8        1277
Trans perf 61x67          10.0        10        1891
          Total points    53.0        53
```

3. Conclusion

3.1 Problems

Most obstacles in Part A happened due to my carelessness. Logic for this program is quite clear but lots of details need to be implemented properly. In the meanwhile, it is inefficient to debug in linux with gedit. It is rather time consuming than brain consuming.

Divide and conquer is a first thought for Part B and would not take a long time to implement. Only the 64*64 matrix need to make full use of all the variables, which takes some time to adjust my strategy.

3.2 Achievements

Strength of project solution: Part A – intuitive, also simulate the structure.
Part B – easy to understand.

Things learned: cache memory has a great quite impact on the performance.