2016-12146 이세리

Understanding Cache Memories

0. Introduction

This lab helped me understand the impact of cache memories on the performance of C programs. This lab consists of two parts. The first part, I wrote a small C program that simulates the behavior of cache memory. In the second part, I optimized a small matrix transpose function, with the goal of minimizing the number of cache misses.

cf) Eviction policies of Cache

The first row of Matrix A evicts the first row of Matrix B. Caches are memory aligned. Matrix A and B are stored in memory at addresses such that both the first elements align to the same place in cache! Diagonal elements evict each other.

Matrices are stored in memory in a row major order. If the entire matrix can't fit in the cache, then after the cache is full with all the elements, it can load. The next elements will evict the existing elements of the cache.

cf) getopt()

-when getopt returns -1, indicating no more options are present, the loop terminates.

-a switch statement is used to dispatch on the return value from getopt. In typical use, each case just sets a variable that is used later in the program.

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int
main (int argc, char **argv)
{
  int aflag = 0;
  int bflag = 0;
  char *cvalue = NULL;
  int index;
  int c;

  opterr = 0;
```

```
  while ((c = getopt (argc, argv, "abc:")) != -1)
    switch (c)
      {
      case 'a':
        aflag = 1;
        break;
      case 'b':
        bflag = 1;
        break;
      case 'c':
        cvalue = optarg;
        break;
      case '?':
        if (optopt == 'c')
          fprintf (stderr, "Option -%c requires an argument.\n",
optopt);
        else if (isprint (optopt))
          fprintf (stderr, "Unknown option `-%c'.\n", optopt);
        else
          fprintf (stderr,
                   "Unknown option character `\\x%x'.\n",
                   optopt);
        return 1;
      default:
        abort ();
      }

  printf ("aflag = %d, bflag = %d, cvalue = %s\n",
          aflag, bflag, cvalue);

  for (index = optind; index < argc; index++)
    printf ("Non-option argument %s\n", argv[index]);
  return 0;
}
```

---

1. Writing a cache simulator

My job for Part 1 was to fill in the csim.c file so that it takes the same command line arguments and produce the identical output as the reference simulator.

Valgrind memory traces have the following form:

[space] operation address, size

The operation field denotes the type of memory access: “I” is the instruction load, “L” data load, “S” data store, and “M” data modify. There is never a space before “I” but there is a space before each “M”, “L”, and “S”. The address field specifies a 64-bit hexadecimal memory address. The size field specifies the number of bytes accessed by the operation.

2016-12146 이세리

The cache simulator in csim.c takes a valgrind memory trace as input, simulates th hit/miss behavior of a cache memory on this trace, and outputs a total number of hits, misses and evictions. The cache simulator uses the LRU replacement policy when choosing which cache line to evict.

The simulator works correctly for arbitrary s, E, and b. This means that I needed to allocate storage for my simulator's data structures using the malloc function. Also, I ignored all instruction cache accesses because this lab is only interested in data cache performance.

For each test case, outputting the correct number of cache hits, misses and evictions gives me full credit for that test case. Each data load or store operation can cause at most one cache miss. The data modify operation is treated as a load followed by a store to the same address. Thus, a data modify operation can result in two cache hits, or a miss and a hit plus a possible eviction.

The reference simulator takes the following command-line arguments:

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

-s : number of set index bits

-E: Associativity, number of lines per set

-b: number of block bits

—t

1) parse command line arguments using getopt()

2) compute S, E, and B from command line arguments

```
for (int opt; (opt = getopt(argc, argv, "s:E:b:t:")) != -1;) {
    switch (opt) {
            /*2)compute s, E, b from command line arguments*/
        case 's':
            set_index_bits = atoi(optarg);
            cache.set_num = 2 << set_index_bits;
            break;

        case 'E': //number of line per set
            cache.line_num = atoi(optarg);
            break;

        case 'b':
            block_bits = atoi(optarg);
            break;

        case 't': // Input filename
            if (!(file = fopen(optarg, "r"))) {
                return 1;
            }
            break;
```

```
        default://otherwise, it is an unknown option
            return 1;
    } //end of switch

} //end of for
```

3) initialize your cache simulator

```
/*3) initialize your cache simulator*/
cache.sets = malloc(sizeof(set_t) * cache.set_num);
for (int i = 0; i < cache.set_num; i++) {
    cache.sets[i].lines = calloc(sizeof(line_t), cache.line_num);
}
```

4) replays the given trace file against your cache simulator and count the number of hits, misses, and evictions

```
void simulate(int addr) {
    … 중간생략 …
    //select set for set[set_index]
    set_t *set = &cache.sets[set_index];

    /*check if cache hit*/
    for (int i = 0; i < cache.line_num; i++) {
        line_t* line = &set->lines[i];

        // Check if the cache line is valid
        if (!line->valid) {
            continue;
        }
        // Compare tag bits
        if (line->tag != tag) {
            continue;
        }

        /*cache hit*/
        hit_count++;
       //update cache
        return;
    }
    /*********************/

    /*it is cache miss*/
    miss_count++;

    /*check for cache emtpy line*/
    for (int i = 0; i < cache.line_num; i++) {
        line_t* line = &set->lines[i];

        if (line->valid) {
            continue;
        }

        line->valid = true;
        line->tag = tag;
       //update cache
        return;
    }
    /***************************/

    /*it is cache eviction*/
    eviction_count++;

    /*look for least recently used cache line*/
    for (int i = 0; i < cache.line_num; i++) {
```

```
        line_t* line = &set->lines[i];

        if (line->time) { //time is [0,E) if it is not zero move on!
            continue;
        }

        line->valid = true;
        line->tag = tag;
       //update cache
        return;
    }
    /******************************************/
}
```

5) free allocated memory

```
/*5) free allocated memory*/
for (int i = 0; i < cache.set_num; i++) {
    free(cache.sets[i].lines);
}
free(cache.sets);
```

6) output the hit and miss statistics for the autograder using printSummary()

```
printSummary(hit_count, miss_count, eviction_count);
```

---

2. Optimizing Matrix Transpose

I wrote a transpose function in trans.c that causes as few cache misses as possible. For part 2, the correctness and performance of transpose_submit function is evaluated on three different-sized output matrices:

-32 X 32 (m < 300) , 64 X 64 (m < 1300) , 61 X 67 (m < 2000)

For each matrix size, the performance of 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)

But my code only needed to be correct for these three cases so I optimized it specifically for these three cases. I explicitly checked for input sizes and implemented separate code optimized for each case.

The autograder takes the matrix size as input. It uses valgrind to generate a trace of each registered transpose function. It then evaluates each trace by running the reference simulator on a cache with parameters written above.

I used the blocking method, which divides the matrix into sub-matrices. Size of sub-matrix depends on cache block size, cache size, input matrix size.

```
######Blocked Matrix Multiplication######

c = (double *) calloc(sizeof(double), n*n);

void mmm(double *a, double *b, double *c, int n) {
```

```
int i, j, k;
for (i = 0; i < n; i+=B)
    for (j = 0; j < n; j+=B)
        for (k = 0; k < n; k+=B)
        /* BXB mini matrix multiplications */
        for (i1 = i; i1< i+B; i++)
            for (j1 = j; j1< j+B; j++)
                for (k1 = k; k1< k+B; k++)
                    c[i1*n+j1] += a[i1*n+k1]*b[k1*n+j1]
```

suggest largest possible block size B, but limit 3B**2 < C! You get 1 kilobytes of cache, directly mapped (E =1), block size is 32 bytes (b = 5), there are 32 sets (s = 5).

1) if i = j => A[i][j] = B[i][j] results in eviction

my solution: do the i == j calculation at the end

```
/**32 X 32 ***************************************************************/
if (M == 32 && N == 32){
    /block size 8*8*/
    for(l=0; l < 4; l++){
        for(k=0; k < 4; k++){
            for (i = 8*l; i < 8*l+8; i++){
                for (j = 8*k; j < 8*k+8; j++) {
                    if(i!=j)
                        B[j][i]=A[i][j];
                    else{
                        tmp=A[j][j];
                        same = j;
                    }
                }
                if(k==l) B[same][same] = tmp;
            }
        }
    }

}
/**32 X 32 ***************************************************************/
```

2) divide the matrix into 8*8 blocks, and do calculation separately.

```
/**64 X 64 ***************************************************************/
else if (M==64 && N == 64){
    for(i=0; i<64; i+=8){
        for(j=0; j<64; j+=8){
            for(k=0; k<4; k++){
                l = A[i+k][j+0];
                tmp = A[i+k][j+1];
                same = A[i+k][j+2];
                remainder = A[i+k][j+3];
                sets_num = A[i+k][j+4];
                f = A[i+k][j+5];
                g = A[i+k][j+6];
```

```
                h = A[i+k][j+7];
                B[j+0][i+k+0] = l;
                B[j+0][i+k+4] = f;
                B[j+1][i+k+0] = tmp;
                B[j+1][i+k+4] = g;
                B[j+2][i+k+0] = diag;
                B[j+2][i+k+4] = h;
                B[j+3][i+k+0] = remainder;
                B[j+3][i+k+4] = sets_num;
            }

            l = A[i+4][j+4];
            tmp = A[i+5][j+4];
            diag = A[i+6][j+4];
            remainder = A[i+7][j+4];
            sets_num = A[i+4][j+3];
            f = A[i+5][j+3];
            g = A[i+6][j+3];
            h = A[i+7][j+3];

            B[j+4][i+0] = B[j+3][i+4];
            B[j+4][i+4] = l;
            B[j+3][i+4] = sets_num;
            B[j+4][i+1] = B[j+3][i+5];
            B[j+4][i+5] = tmp;
            B[j+3][i+5] = f;
            B[j+4][i+2] = B[j+3][i+6];
            B[j+4][i+6] = diag;
            B[j+3][i+6] = g;
            B[j+4][i+3] = B[j+3][i+7];
            B[j+4][i+7] = remainder;
            B[j+3][i+7] = h;

            for(k=0;k<3;k++){

                l = A[i+4][j+5+k];
                tmp = A[i+5][j+5+k];
                diag = A[i+6][j+5+k];
                remainder = A[i+7][j+5+k];
                sets_num = A[i+4][j+k];
                f = A[i+5][j+k];
                g = A[i+6][j+k];
                h = A[i+7][j+k];

                B[j+5+k][i+0] = B[j+k][i+4];
                B[j+5+k][i+4] = l;
                B[j+k][i+4] = sets_num;
                B[j+5+k][i+1] = B[j+k][i+5];
                B[j+5+k][i+5] = tmp;
                B[j+k][i+5] = f;
                B[j+5+k][i+2] = B[j+k][i+6];
                B[j+5+k][i+6] = diag;
                B[j+k][i+6] = g;
                B[j+5+k][i+3] = B[j+k][i+7];
                B[j+5+k][i+7] = remainder;
                B[j+k][i+7] = h;

            }

        }
    }


}
```

3) naive approach works

```
/**61 X 67 ***************************************************************/
for (i = 0; i < N; i+=8) {
```

```
		for (j = 0; j < M; j+= 8) {
			for (k = j;(k<j+8) && (k<M);++k) {
				for (l = i; (l<i+8)&&(l<N); ++l) {
					B[k][l] = A[l][k];
				}
			}
		}
	}
/**61 X 67 ****************************************************************/
```

---

3. Conclusion

중간고사2에서 cache에 관한 문제가 나왔지만 잘 풀지 못해서 cache에 대해 내가 잘 알지 못하고 있다는 생각을 하고 있었는데, 마침 cache에 관련한 lab이 나와서 cache에 대해 더 배워볼 수 있는 기회가 생겨 기뻤다. 특히 Part 2의 matrix transpose function을 cache miss가 최대한 안일어나도록 짜는 것은 computer architecture 시간에 배웠지만 한번도 스스로 구현해볼 기회가 없었는데 이번 기회에 3가지 다른 matrix size에 대해서 cache-friendly한 code를 직접 짜볼 수 있어서 좋았다. 단순히 block으로 나누는 것에서 끝나는 것이 아니여서 조금 힘들었다. 이런 식으로 코드를 짤 수 있다면 performance도 고려하는 코드를 짜는 최강의 프로그래머가 될 수 있을 것 같다.

```
seri@ubuntu:~/Downloads/cachelab$ ./driver.py
Part A: Testing cache simulator
Running ./test-csim
                        Your simulator     Reference simulator
Points (s,E,b)    Hits  Misses  Evicts    Hits  Misses  Evicts
     6 (1,1,1)       9       8       6       9       8       6  traces/yi2.trace
     6 (4,2,4)       4       5       2       4       5       2  traces/yi.trace
     6 (2,1,4)       2       3       1       2       3       1  traces/dave.trace
     6 (2,1,3)     167      71      67     167      71      67  traces/trans.trace
     6 (2,2,3)     201      37      29     201      37      29  traces/trans.trace
     6 (2,4,3)     212      26      10     212      26      10  traces/trans.trace
     6 (5,1,5)     231       7       0     231       7       0  traces/trans.trace
    12 (5,1,5)  265189   21775   21743  265189   21775   21743  traces/long.trace
    54
```

```
Cache Lab summary:
                        Points   Max pts      Misses
Csim correctness          54.0        54
Trans perf 32x32          10.0        10         287
Trans perf 64x64          10.0        10        1299
Trans perf 61x67          16.0        16        1931
          Total points    90.0        90
```