1 Part A: Writing a Cache Simulator

Solution The source code of data cache simulator with least recently used (LRU) cache level policy is presented on listing 1.

```c
#define _GNU_SOURCE
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <getopt.h>
#include "cachelab.h"

const char *options = "h v s: E: b: t:";
const char *help = "Usage: ./csim-ref [-hv] -s <num> -E <num> -b <num> -t <file>\
Options:\
  -h         Print this help message.\
  -v         Optional verbose flag.\
  -s <num>   Number of set index bits.\
  -E <num>   Number of lines per set.\
  -b <num>   Number of block offset bits.\
  -t <file>  Trace file.\
\
Examples:\
  linux>  ./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace\
  linux>  ./csim-ref -v -s 8 -E 2 -b 4 -t traces/yi.trace\n";

typedef struct cacheLineNode
{
    int tag;
    struct cacheLineNode *next;
} CacheLineNode;

typedef struct set
{
    int currentSize;
    int maxSize;
    CacheLineNode *head;
    CacheLineNode *tail;
} Set;

int hits;
int misses;
int evictions;
int blockOffsetMask;
int setNumberMask;
int dCacheSetBits;
int dCacheAssociativity;
int dCacheBlockBits;
Set *dCacheSets;
int dCacheVerboseMode;

int isFull(const Set *set)
{
    return set->currentSize == set->maxSize;
}

void evict(Set *set)
{
    CacheLineNode *evicted = set->head;
    if (set->head->next == NULL)
    {
        set->head = NULL;
        set->tail = NULL;
    }
    else
    {
        set->head = set->head->next;
    }

    set->currentSize--;

```

Listing 1 (Cont.): csim.c

```c
    free(evicted);

    evictions++;
    if (dCacheVerboseMode)
        fprintf(stdout, "eviction ");
}

void load(Set *set, int tag)
{
    if (isFull(set))
        evict(set);

    CacheLineNode *newLine = (CacheLineNode *)malloc(sizeof(CacheLineNode));
    newLine->tag = tag;
    newLine->next = NULL;

    if (set->tail == NULL)
    {
        set->head = newLine;
        set->tail = newLine;
    }
    else
    {
        set->tail->next = newLine;
        set->tail = newLine;
    }

    set->currentSize++;
}

int isCached(Set *set, int tag)
{
    int hit = 0;
    CacheLineNode *prev = NULL;
    CacheLineNode *current = set->head;
    while (current != NULL)
    {
        if (current->tag == tag)
        {
            hit = 1;
            if (current->next == NULL)
            {
                break;
            }

            if (prev != NULL)
            {
                prev->next = current->next;
            }
            else
            {
                set->head = current->next;
            }

            set->tail->next = current;
            set->tail = current;
            current->next = NULL;
            break;
        }

        prev = current;
        current = current->next;
    }

    if (hit)
    {
        hits++;
        if (dCacheVerboseMode)
            fprintf(stdout, "hit ");
    }
    else
    {
        misses++;
```

Listing 1 (Cont.): csim.c

```c
        if (dCacheVerboseMode)
            fprintf(stdout, "miss ");
    }

    return hit;
}

int createRightBitMask(int numberOfBits)
{
    int mask = 0;
    while (numberOfBits)
    {
        mask ^= 1 << --numberOfBits;
    }

    return mask;
}

void initializeDCache(int setBits, int associativity, int blockBits, int verboseMode)
{
    hits = 0;
    misses = 0;
    evictions = 0;
    dCacheSetBits = setBits;
    dCacheAssociativity = associativity;
    dCacheBlockBits = blockBits;
    dCacheVerboseMode = verboseMode > 0 ? verboseMode : 0;
    setNumberMask = createRightBitMask(dCacheSetBits);
    blockOffsetMask = createRightBitMask(dCacheBlockBits);

    int setNumber = 1 << dCacheSetBits;
    dCacheSets = (Set *)malloc(setNumber * sizeof(Set));
    for (int i = 0; i < setNumber; i++)
    {
        dCacheSets[i].currentSize = 0;
        dCacheSets[i].maxSize = dCacheAssociativity;
        dCacheSets[i].head = NULL;
        dCacheSets[i].tail = NULL;
    }
}

void parseAddress(unsigned int address, int *setNumber, int *tag, int *blockOffset)
{
    *blockOffset = address & blockOffsetMask;
    unsigned int cutOffset = address >> dCacheBlockBits;
    *setNumber = cutOffset & setNumberMask;
    *tag = cutOffset >> dCacheSetBits;
}

void dCacheOperate(Set *set, int tag)
{
    if (!isCached(set, tag))
    {
        load(set, tag);
    }
}

int dCacheSimulate(char command, int address, int bytes)
{
    if (dCacheVerboseMode)
    {
        fprintf(stdout, "%c %x,%d ", command, address, bytes);
    }

    int setNumber;
    int tag;
    int offsetNumber;
    parseAddress(address, &setNumber, &tag, &offsetNumber);
    Set *set = &dCacheSets[setNumber];
    switch (command)
    {
```

Listing 1 (Cont.): csim.c

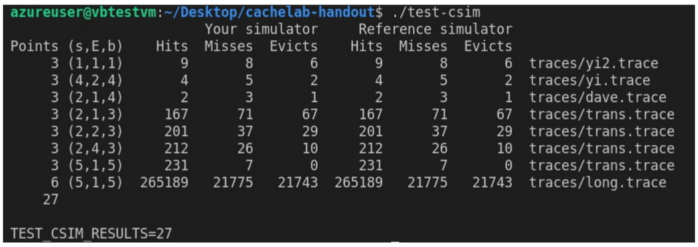
```c
    case 'S':
    case 'L':
    {
        dCacheOperate(set, tag);
        break;
    }
    case 'M':
    {
        dCacheOperate(set, tag);
        dCacheOperate(set, tag);
        break;
    }
    default:
        return -1;
    }

    if (dCacheVerboseMode)
        fprintf(stdout, "\n");

    return 0;
}

int parseIntegerOption(char option, const char *optionValue, int *value)
{
    if (sscanf(optionValue, "%d", value) == 0)
    {
        fprintf(stderr,
                "Invalid value %s for option -%c, integer required.\n",
                optionValue,
                option);

        return 0;
    }

    return 1;
}

int main(int argc, char **argv)
{
    opterr = 0;
    int optionVerboseMode = 0;
    int optionSetBits = 0;
    int optionAssociativity = 0;
    int optionBlockBits = 0;
    char *optionTraceFile = NULL;
    int c;
    while ((c = getopt(argc, argv, options)) != -1)
        switch (c)
        {
        case 'h':
            fprintf(stderr, "%s", help);
            return 0;
        case 'v':
            optionVerboseMode = 1;
            break;
        case 's':
            if (!parseIntegerOption(c, optarg, &optionSetBits))
                return -1;
            break;
        case 'E':
            if (!parseIntegerOption(c, optarg, &optionAssociativity))
                return -1;
            break;
        case 'b':
            if (!parseIntegerOption(c, optarg, &optionBlockBits))
                return -1;
            break;
        case 't':
            optionTraceFile = optarg;
            break;
        case '?':
            if (optopt == 's' ||
                optopt == 'E' ||
```

Listing 1 (Cont.): csim.c

```c
                optopt == 'b' ||
                optopt == 't')
                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();
        }

    if (optionSetBits == 0 ||
        optionAssociativity == 0 ||
        optionBlockBits == 0 ||
        optionTraceFile == NULL)
    {
        fprintf(stderr, "Missing required argument...");
        fprintf(stderr, "%s", help);
    }

    initializeDCache(optionSetBits, optionAssociativity, optionBlockBits, optionVerboseMode);

    FILE *fp = fopen(optionTraceFile, "r");
    if (fp == NULL)
    {
        fprintf(stderr, "No such file or directory - %s", optionTraceFile);
        return -1;
    }

    size_t max_length = 32;
    char *line = (char *)malloc(max_length * sizeof(char));
    while (getline(&line, &max_length, fp) != -1)
    {
        if (!isspace(line[0]))
            continue;

        char parsedCommand;
        int parsedAddress;
        int parsedBytes;
        sscanf(line, " %c %x,%d", &parsedCommand, &parsedAddress, &parsedBytes);
        dCacheSimulate(parsedCommand, parsedAddress, parsedBytes);
    }

    fclose(fp);

    printSummary(hits, misses, evictions);
    return 0;
}
```

Listing 1: csim.c

The program was run against the reference simulator. The screenshot of the test benchmark run is presented on figure 1.



```
azureuser@vbtestvm:~/Desktop/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
```

Figure 1: Benchmark results

2 Part B: Optimizing Matrix Transpose

The solution includes three different transpose functions for matrices with sizes:

- 32 × 32
- 64 × 64
- 32 × 32

The full source code of trans.c is show on listing 2.

```c
#include <stdio.h>
#include "cachelab.h"

int is_transpose(int M, int N, int A[N][M], int B[M][N]);
void trans32(int M, int N, int A[N][M], int B[M][N]);
void trans64(int M, int N, int A[N][M], int B[M][N]);
void transX(int M, int N, int A[N][M], int B[M][N]);
char transpose_submit_desc[] = "Transpose submission";
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    if (N == 32 && M == 32)
    {
        trans32(M, N, A, B);
    }
    else if (N == 64 && M == 64)
    {
        trans64(M, N, A, B);
    }
    else
    {
        transX(M, N, A, B);
    }

    return;
}

void transX(int M, int N, int A[N][M], int B[M][N])
{
    for (int fourthI = 0; fourthI < N; fourthI += 8)
    {
        for (int fourthJ = 0; fourthJ < M; fourthJ += 8)
        {
            for (int sub = 0; sub < 8; sub += 4)
            {
                int i = fourthI;
                int iEnd = i + 8 > N ? N : i + 8;
                for (; i < iEnd; i++)
                {
                    int k = fourthJ + sub;
                    int kEnd = k + 4 > M ? M : k + 4;
                    for (; k < kEnd; k++)
                    {
                        B[k][i] = A[i][k];
                    }
                }
            }
        }
    }

    return;
}

void trans64(int M, int N, int A[N][M], int B[M][N])
{
    for (int fourthI = 0; fourthI < 64; fourthI += 8)
    {
        for (int fourthJ = 0; fourthJ < 64; fourthJ += 8)
        {
            for (int sub = 0; sub < 8; sub += 4)
            {
                for (int i = 0; i < 8; i++)
                {
```

Listing 2 (Cont.): csim.c

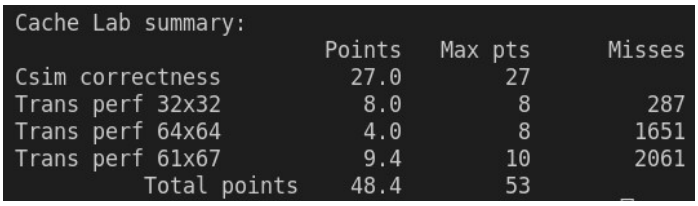
```c
                    int iEf = i + fourthI;
                    int a0 = A[iEf][fourthJ + sub];
                    int a1 = A[iEf][fourthJ + 1 + sub];
                    int a2 = A[iEf][fourthJ + 2 + sub];
                    int a3 = A[iEf][fourthJ + 3 + sub];
                    B[fourthJ + sub][iEf] = a0;
                    B[fourthJ + 1 + sub][iEf] = a1;
                    B[fourthJ + 2 + sub][iEf] = a2;
                    B[fourthJ + 3 + sub][iEf] = a3;
                }
            }
        }
    }

    return;
}

void trans32(int M, int N, int A[N][M], int B[M][N])
{
    for (int fourthI = 0; fourthI < 32; fourthI += 8)
    {
        for (int fourthJ = 0; fourthJ < 32; fourthJ += 8)
        {
            for (int i = 0; i < 8; i++)
            {
                int iEf = i + fourthI;
                int a0 = A[iEf][fourthJ];
                int a1 = A[iEf][fourthJ + 1];
                int a2 = A[iEf][fourthJ + 2];
                int a3 = A[iEf][fourthJ + 3];
                int a4 = A[iEf][fourthJ + 4];
                int a5 = A[iEf][fourthJ + 5];
                int a6 = A[iEf][fourthJ + 6];
                int a7 = A[iEf][fourthJ + 7];
                B[fourthJ][iEf] = a0;
                B[fourthJ + 1][iEf] = a1;
                B[fourthJ + 2][iEf] = a2;
                B[fourthJ + 3][iEf] = a3;
                B[fourthJ + 4][iEf] = a4;
                B[fourthJ + 5][iEf] = a5;
                B[fourthJ + 6][iEf] = a6;
                B[fourthJ + 7][iEf] = a7;
            }
        }
    }

    return;
}

/*
 * registerFunctions - This function registers your transpose
 *     functions with the driver.  At runtime, the driver will
 *     evaluate each of the registered functions and summarize their
 *     performance. This is a handy way to experiment with different
 *     transpose strategies.
 */
void registerFunctions()
{
    /* Register your solution function */
    registerTransFunction(transpose_submit, transpose_submit_desc);
}

/*
 * is_transpose - This helper function checks if B is the transpose of
 *     A. You can check the correctness of your transpose by calling
 *     it before returning from the transpose function.
 */
int is_transpose(int M, int N, int A[N][M], int B[M][N])
{
    int i, j;

    for (i = 0; i < N; i++)
    {
```

Listing 2 (Cont.): csim.c

```c
        for (j = 0; j < M; ++j)
        {
            if (A[i][j] != B[j][i])
            {
                return 0;
            }
        }
    }
    return 1;
}
```

Listing 2: csim.c

The screenshot of the full lab benchmark run is presented on figure 2.



```
Cache Lab summary:
                        Points   Max pts      Misses
Csim correctness          27.0        27
Trans perf 32x32           8.0         8         287
Trans perf 64x64           4.0         8        1651
Trans perf 61x67           9.4        10        2061
          Total points    48.4        53
```

Figure 2: Final lab results