Contents

0.1 Objective

The purpose of this lab is to understand the working of cache memories by developing a small C program that simulates the behavior of a cache memory. This task aims to strengthen understanding of cache systems and enhance C programming skills. The simulator tracks cache hits, misses, and evictions for different cache configurations based on given memory traces.

0.2 Tools and Technologies

- Programming Language: C
- GNU Toolchain

0.3 Implementation

The implementation involves writing a cache simulator that takes a Valgrind memory trace as input, simulates cache hit/miss behavior, and outputs the number of hits, misses, and evictions. The simulator adheres to the least-recently used (LRU) cache replacement policy.

0.3.1 Cache Parameters

The cache is configured with parameters s, E, and b, where:

- s - Number of set index bits, determining the number of sets $S = 2^s$.
- E - Associativity, indicating the number of lines per set.
- b - Block offset bits, determining the block size $B = 2^b$.

0.3.2 Simulator Overview

The simulator parses the memory trace file and processes each memory access (load/store/modify) to determine whether it results in a hit, miss, or eviction.

Key Functions

- **initCache()** - Allocates memory for the cache based on s, E, and b.
- **accessData()** - Core function that simulates cache access and updates hit, miss, and eviction counts.
- **freeCache()** - Frees the dynamically allocated memory for the cache.
- **printSummary()** - Prints the summary of hits, misses, and evictions.

0.4 Results

The following figure shows the results of running the cache simulator against the provided memory traces. As shown in the screenshot, the outputs of the custom cache simulator match the reference simulator's output across all test cases.

```
miraj@DESKTOP-M45A623:~$ cd /mnt/c/Users/welcome\ computers/Downloads/cachelab-handout
miraj@DESKTOP-M45A623:/mnt/c/Users/welcome computers/Downloads/cachelab-handout$ ls
Makefile  README  cachelab.c  cachelab.h  csim  csim-ref  csim.c  driver.py  test-csim  traces
miraj@DESKTOP-M45A623:/mnt/c/Users/welcome computers/Downloads/cachelab-handout$ python3 driver.py
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
     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
     9 (5,1,5)  265189   21775   21743  265189   21775   21743  traces/long.trace
    48


MAXIMUM POINTS=48


Cache Lab summary:
                        Points   Max pts      Misses
Csim correctness          48.0        48
miraj@DESKTOP-M45A623:/mnt/c/Users/welcome computers/Downloads/cachelab-handout$
```

Figure 1: Cache simulator results compared with the reference simulator

As seen in the figure, the number of hits, misses, and evictions produced by our simulator are consistent with the reference simulator for all trace files. This validates the correctness of the implementation.

0.5 Self-Evaluation

The cache simulator was tested thoroughly using various provided traces, and the correctness of the results was verified by comparing them with the reference simulator output. The total points achieved are the maximum of 48, confirming the simulator's accuracy.

0.5.1 Correctness

The simulator correctly handled cache hits, misses, and evictions across all test cases. For example, in the case of trace `trans.trace`, the cache demonstrated 67 hits, 71 misses, and 67 evictions, matching the reference results exactly.

0.5.2 Performance

The simulator successfully implemented the least-recently used (LRU) policy, which ensures that the least recently accessed cache lines are replaced first. This policy was key to obtaining the correct number of evictions in scenarios where the cache was full.