**EECS**

**University of Tennessee**

**Computer Systems Organization– CS 530**

**Lab 1 – Cache Simulator**

**Submitted by:**

Mahendra Duwal Shrestha

[mduwalsh@vols.utk.edu](mailto:mduwalsh@vols.utk.edu)

**Introduction**

**Background**

In this lab, we were required to develop program in any one of the programming languages to simulate cache behavior. The objective of this lab was to simulate cache behavior and observe differences in performance with various configurations of cache parameters. The program should support multi-level cache configuration with L2 and L3 as optional layers of cache that could be configured by user. The program should also support changeable configuration of write method to be used in cache and other parameters such as block size, word size, memory access (read/write) time and, hit time, associativity, number of cache lines in each levels of cache. All time are in cycles.

**Overview**

The cache simulator program was developed and language chosen to develop it was C++. Only standard library in C++ was used to develop it. User can easily configure cache levels and structure of cache in each level through command argument during execution of program. Number of trace addresses to be read or write in the memory can be read and traced through this simulator but it has to be in defined pattern in a file. The trace file should contain read/write addresses in hexadecimal format in left side and read or write designated by ‘R’ or ‘W’ respectively on right side with blank space being the delimiter.

The cache simulator program developed can support both write methods, write through and write back. Write allocate policy was selected for both write methods. Block size is assumed fixed for all levels of cache memory for simplification of transfer of block from one level to another. Levels of cache memory, word size, block size, memory access time are taken as command line input arguments. Also individual cache parameters number of lines in a cache, associativity of cache and hit time of cache are taken as command line inputs. Based on configuration of cache memory, the program outputs statistics for each level of cache like read/write access, read/write misses, read/write hits and read/write miss rate and read/write hit rate. It calculates average memory access time (AMAT) for read access made and write access made separately and then as combined.

**Assumptions**

Certain conditions were assumed for simplification of simulation during development of code.

* Cache is inclusive; if a block is found in higher level memory, it is always found on lower level memory
* Write through and write back both policies use write allocate policy
* Least recently used block would get replaced within a set if one block needs to get replaced (LRU algorithm)
* While using LRU algorithm in cache write back policy, dirty block gets replaced even if another block is not dirty within a same set and the dirty block was least recently used
* Block size for all levels of cache are assumed to be same
* Main memory read time and write time are equal
* Read and write to same level of cache takes same time
* Hit time covers for both read and write to cache with in a same time

**AMAT Calculation**

For write through, write allocate policy:

On a read-hit, let RH be the time needed (price) to get the data from this cache to one level higher. The size of data we are getting is the size that a higher level needs.

On a read-miss, we pay RH plus we pay some time RLL to read one block (block size is for this cache) from the lower level (this will include the time for a lower level to find this block).

On a write-hit, we pay some time WH to get the data from one level higher to this cache, plus we pay some time WLL to get the same piece of data from this cache to one level lower (this will include time for a lower level to deal with this write). The size of data we are transferring is the size that a higher level needs to write.

On a write-miss, we pay WH + WLL plus we pay some time RLL (same price that we paid on a read miss) to read one block (block size is for this cache) from the lower level.

Read\_time\_for\_this\_cache will be R = RH+miss\_rate\*RLL

Write\_time\_for\_this\_cache will be W = WH + WLL+ miss\_rate\*RLL

For write back, write allocate policy:

On a read-hit, let RH be the time needed (price) to get the data from this cache to one level higher. The size of data we are getting is the size that a higher level needs.

On a read-miss, we pay RH plus we pay some time E to evict one block from this cache to a lower level plus we pay RLL to read one block from the lower level (this will include the time for a lower level to find this block). If the block we are evicting is not dirty, then the E is zero, otherwise E is some time D which is the time needed to write a block one level lower. In all those cases a block size is for this cache.

On a write-hit, we pay time WH to get the data from one level higher to this cache. The size of data we are transferring is the size that a higher level needs to write.

On a write-miss, we pay WH, plus we pay some time E to evict one block from this cache to a lower level plus we pay WRL to write data in to lower level and read one block from the lower level (this will include the time for a lower level to find this block). If the block we are evicting is not dirty, then the E is zero, otherwise E is some time D which is the time needed to write a block one level lower. In all those cases a block size is for this cache.

Read\_time\_for\_this\_cache will be R = RH+miss\_rate\*(RLL + E) =

RH+miss\_rate\*(RLL + %dirty\*D)

Write\_time\_for\_this\_cache will be W = WH+miss\_rate\*(WRL + E) =

WH+miss\_rate\*(WRL + %dirty\*D)

For simplicity, assumption of RLL and WRL both are equivalent to hit time was made.

**Test**

The code was tested against the sample trace files that were provided to us and program executed successfully showing correct number of write accesses and read accesses made to cache for all possible configuration. However, those contain large number of addresses to follow correctness hits and misses. Therefore a file ‘test1.txt’ with fewer sample trace addresses was used to test it where number of hits and misses to be occurred for given configuration was known.

test1.txt with 20 total address accesses, where 8 are write and 12 are read, was tested against cache memory with 3 levels with following configurations.

main memory access time=100 cycles; word size=4 bytes; block size=4 words; lines in L1 cache=10; associativity L1=4; hit time L1=1 cycle; lines in L2 cache=20; associativity L1=4; hit time L1=10 cycles; lines in L3 cache=40; associativity L1=4; hit time L1=20 cycles;

For write back method,

L1 read access=12; L1 read hits=6; L1 read misses=6; L1 read hit rate=0.5; L1 read miss rate=0.5;

L1 write access=8; L1 write hits=6; L1 write misses=2; L1 write hit rate=0.75; L1 write miss rate=0.25;

L2 read access=6; L2 read hits=0; L2 read misses=6; L2 read hit rate=0; L2 read miss rate=1;

L2 write access=2; L2 write hits=0; L2 write misses=2; L2 write hit rate=0; L2 write miss rate=1;

L3 read access=6; L3 read hits=0; L3 read misses=6; L3 read hit rate=0; L3 read miss rate=1;

L3 write access=2; L3 write hits=0; L3 write misses=2; L3 write hit rate=0; L3 write miss rate=1;

Dirty blocks removed for all caches were 0;

This was as expected.

read AMAT = 81.5; write AMAT=41.25;

AMAT=65.4 cycles;

For write through method,

L1 read access=12; L1 read hits=6; L1 read misses=6; L1 read hit rate=0.5; L1 read miss rate=0.5;

L1 write access=8; L1 write hits=6; L1 write misses=2; L1 write hit rate=0.75; L1 write miss rate=0.25;

L2 read access=6; L2 read hits=0; L2 read misses=6; L2 read hit rate=0; L2 read miss rate=1;

L2 write access=2; L2 write hits=0; L2 write misses=2; L2 write hit rate=0; L2 write miss rate=1;

L3 read access=6; L3 read hits=0; L3 read misses=6; L3 read hit rate=0; L3 read miss rate=1;

L3 write access=2; L3 write hits=0; L3 write misses=2; L3 write hit rate=0; L3 write miss rate=1;

This was as expected.

read AMAT = 81.5; write AMAT=171.25;

AMAT=117.4 cycles;

read AMAT was same for both write back and write method because dirty blocks removed for read misses using write back policy was 0 so it didn't have to write any dirty block to lower level memory as miss penalty during read miss.

However, there is significance difference in write AMAT. First of all for write back, it didn't have to write any dirty blocks to lower level memory on write miss as miss penalty and secondly for write through, it had to write data to all levels of memory even for write hits.

**Discussion**

From this lab, we learnt that cache shows different behavior for under different configurations and performance also changes accordingly. Access patterns for write back and write through policy are different. Write back reduces number of write traffic across memory levels by using dirty bit when block is updated and the block with dirty bit set is written to lower level only when the block gets evicted. Whereas, write back writes data to all levels of memory so that all memory level contains up-to-date data. Because of this behavior difference, average memory access times for these two write policies are different.

**Usage of code**

Usage: ***./cache\_sim trace\_file no\_of\_cache\_levels write\_method mem\_access\_time word\_size block\_size cache1\_lines\_no cache1\_associativity cache1\_hit\_time cache2\_lines\_no cache2\_associativity cache2\_hit\_time cache3\_lines\_no cache3\_associativity cache3\_hit\_time*** trace\_file: name of file containing byte addresses trace

no\_of\_cache\_levels: how many levels of cache

write\_method: it can be either '0' for write through or '1' for write back

mem\_access\_time: access time for main memory

word\_size: size of word in bytes

block\_size: size of block in number of words

cache1\_lines\_no: number of sets or lines in 1st level cache

cache1\_associativity: associatvity for 1st level cache

cache1\_hit\_time: time to access 1st level cache

other parameters are not needed if no\_of\_cache\_levels is entered 1; if no\_of\_cache\_levels entered 2, then provide cache2\_lines\_no, cache2\_associativity, cache2\_hit\_time parameters similar to 1st level cache; if no\_of\_cache\_levels entered 2, then provide cache3\_lines\_no, cache3\_associativity, cache3\_hit\_time for 3rd level cache similar to 1st level cache.

All time are in number of CPU cycles.