|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | |  | |  | |  | |  | |
| L3 Cache Simulator Report  ECE 485 / Microprocessor System Design |
| Carmen Ciobanu Deborah Denhart Jeremiah Franke  12/2/2014 |

Table of Contents

L3 Cache Simulator Report 1

ECE 485 / Microprocessor System Design 1

1. Purpose 1

2. Cache System Architecture 1

3. Assumptions 2

3.1. L3 Cache 2

3.2. L2 cache 2

3.3 Shared Memory 2

4. Design 2

4.1 Trace File Parser 2

4.2 Commands Algorithms 3

4.3 Coherence Protocol (MESIF) 3

4.4. Line Replacement Policy (Pseudo LRU) 3

4.5. L2-L3 Inclusivity 3

4.6. Write Buffer 3

4.7. Memory 3

4.8. Statistics 3

5. Test Plan 4

5.1. Test Plan 4

5.2. Test Cases 4

6. Simulation Results 4

6.1. Trace Files 4

6.2. Results 4

6.3. Usage Statistics 4

# L3 Cache Simulator Report

## ECE 485 / Microprocessor System Design

## 1. Purpose

The purpose of the program is to simulate the last level cache (L3) for a processor that can be used with up to three other processors in a shared memory system.

## 2. Cache System Architecture

The system uses multilevel caches:

* Separate L1 instruction and data caches
* Unified L2 instruction and data caches
* Unified L3 instruction and data caches

Insert picture of caches

The L3 caches and memory are communicating on the FSB (Front Side Bus). The L1 and L2 are probably on the same chip as the CPU. L2 might communicate with L3 through a Back Side Bus, or L3 might also be on the same chip as the CPU.

The L1 and L2 caches as well as the DRAM shared memory are not modeled in this program. The program allows for flexible L3 cache configuration, by taking command line arguments. No command line arguments are taken for L2 cache configuration, but several assumptions are made and will be detailed in the Assumptions section of the report.

## 3. Assumptions and Design Considerations

### 3.1. L3 Cache

* The maximum L3 cache size is 4294965248 bytes
* The possible L3 line sizes (in bytes) are: 32, 64, 128, 256, 512, 1024, 2048
* L3 number of sets must be in the range: 1 to 65535
* L3 number of lines must be in the range: 1 to 2097151
* L3 associativity will be one of: 4, 8, 16, or 32
* Data is not stored in the cache in this simulation
* Only tags, state bits and LRU bits are stored in the cache in this simulation
* L3 is a write-back cache (implicit for MESIF implementation)

### 3.2. L2 cache

* L2 cache parameters (associativity, line size etc.) will change depending on the changes in the L3 cache parameters in such a way that it is possible to maintain inclusivity with both L3 and L1 caches
* The ratio of L3 line size to L2 line size is 2:1
* L2 is write-through the first time a line is modified

### 3.3 Shared Memory

* The size of the L3 line is equal to the number of bytes the DIMM can provide in a burst event
* Only one memory access will be needed to bring in an L3 cache line from DRAM

### Other

* In case of a write command (1), we are assuming all physical addresses generated are pointing to a line that has the permission of being written
* Since in this simulation addresses are generated manually and stored in the trace file, and not translated from a TLB, which contains W bits, extra care will be needed in associating addresses and commands. (maybe keep a R/W bit for each line in the cache????)

## 4. Design

### 4.1 Trace File Parser

Stuff

### 4.2 Commands Algorithms

#### 4.2.1. Commands 0, 2

Command 0 = Read from

Command 2 = Read from

We decided that command 0 would be equivalent to command 2, since at level 2 the instruction and data caches are unified. The actions taken as a result of receiving any of the 0 or 2 commands are identical.

The only notable difference between the two is that since lines containing instructions cannot be written, the snoop result gotten from the other processor will never be a HITM. This of course does make any difference, since HITM and HIT are always asserted simultaneously.

Command 0 Algorithm:

Check tags for all lines in set

// Tags match, we get the line we want

// -----------------------------------------------------

if tag == MATCH

// Line is invalid - Cache MISS

// -------------------------------------------

if MESIF state == INVALID

Increment number of misses for statistics

Increment number of accesses for statistics

Increment number of reads for statistics

Update MESIF state

MESIF related stuff

Bus op = Read cache line from memory (send in address to read\_memory stub code)

Write tag bits

Write pseudo LRU bits (send in flag CACHE\_MISS)

Send message to L2

" Sending x bytes starting at address y to L2" (mention number

of bytes sent since L2 line is half the size of L3 line)

// Line is valid & tags match - cache HIT

// --------------------------------------

else

Increment number of hits for statistics

Increment number of accesses for statistics

Increment number of reads for statistics

Update MESIF state

Write pseudo LRU bits

Send message to L2

" Sending x bytes starting at address y to L2" (mention number

of bytes sent since L2 line is half the size of L3 line)

// Tags dont match any of the line tags - Cache Miss

// --------------------------------------------------

else if tag == NO MATCH

Increment number of misses for statistics

Increment number of accesses for statistics

Increment number of reads for statistics

Select line to evict and write pseudo LRU bits (send in flag EVICT\_LINE)

Capture return to see if line is to be evicted or not

If no line to be evicted

it means we are using an invalid line to put the new data in so no need to write it to memory

else

Check MESIF state

if MESIF state == MODIFIED

we need to write dirty line to memory

else

line was not dirty so it can be overwritten

Update MESIF state

MESIF related stuff

Bus op = Read cache line from memory (send in address to read\_memory stub code)

Write tag bits

Send message to L2

" Sending x bytes starting at address y to L2" (mention number

of bytes sent since L2 line is half the size of L3 line)

If line evicted in L3

send eviction notice to L2 (send addresses for multiple lines if cache line sizes are different)

else

no eviction notice sent

#### 4.2.2. Command 1

Command 1 = Write from…

A write request coming from the L2 cache can be interpreted in one of three ways:

1. CPU write request to L1, L1 cache miss, L2 cache miss
2. L2 writes evicted line to L3
3. L2 writes modified line to L3 behaving as a write through cache the first time a line is modified

Since there is not enough granularity in the provided commands to capture all of these three different behaviors, we decided command 1 will be interpreted as a CPU write request to L1, followed by a cache miss in both L1 and L2. This behavior is more useful in emitting the correct bus operation and maintaining coherence with the other L3 caches.

#### 4.2.3. Command 3

#### 4.2.4. Command 4

#### 4.2.5. Command 5

#### 4.2.6. Command 6

#### 4.2.7. Command 8

Command 8 = clear all

Command 8 clears all tag, MESIF and LRU bits, and implicitly invalidates all lines.

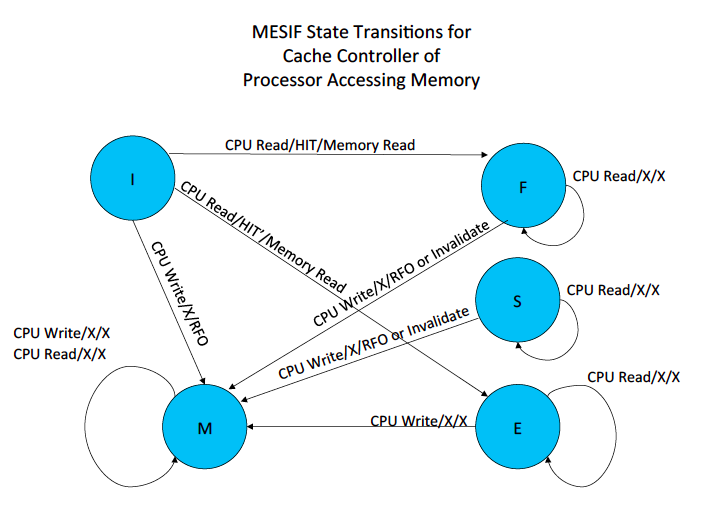
#### 4.2.8. Command 9

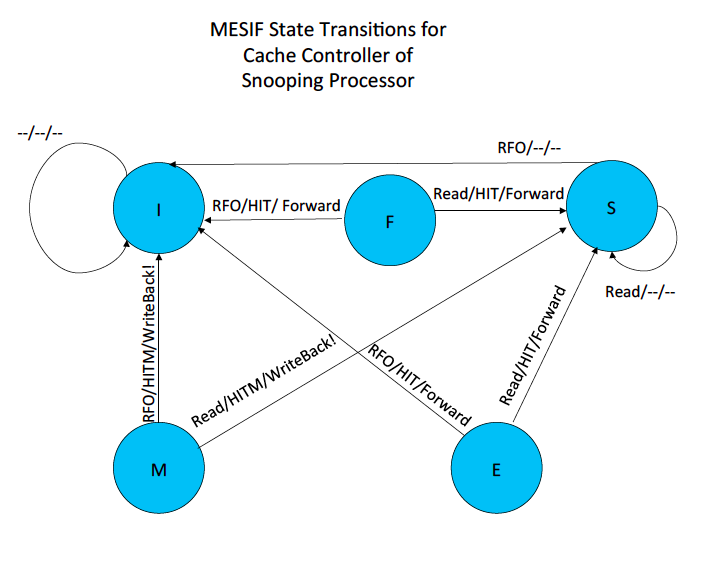
Command 9 = print valid lines

Command 9 displays the tag and MESIF bits of all the valid lines and the pseudo LRU bits of the sets containing at least one valid line.

### 4.3 Coherence Protocol (MESIF)

Stuff





### 4.4. Line Replacement Policy (Pseudo LRU)

Stuff

### 4.5. L2-L3 Inclusivity

Stuff

### 4.6. Write Buffer

Stuff

### 4.7. Memory

Stuff

### 4.8. Statistics

Stuff

## 5. Test Plan

### 5.1. Test Plan

Stuff

### 5.2. Test Cases

Stuff

## 6. Simulation Results

### 6.1. Trace Files

Stuff

### 6.2. Results

Stuff

### 6.3. Usage Statistics

Bibliography

Last Name, F. (Date). Dolor Sit Amet. *Lorem Ipsum*, 1 - 10.

Last Name, F. (Date). *Lorem Ipsum Dolor Sit Amet.* City: Publisher.

Last Name, F. (Date). Lorem Ipsum Dolor Sit Amet. *Duis sed elit ante*, pp. 10-20.