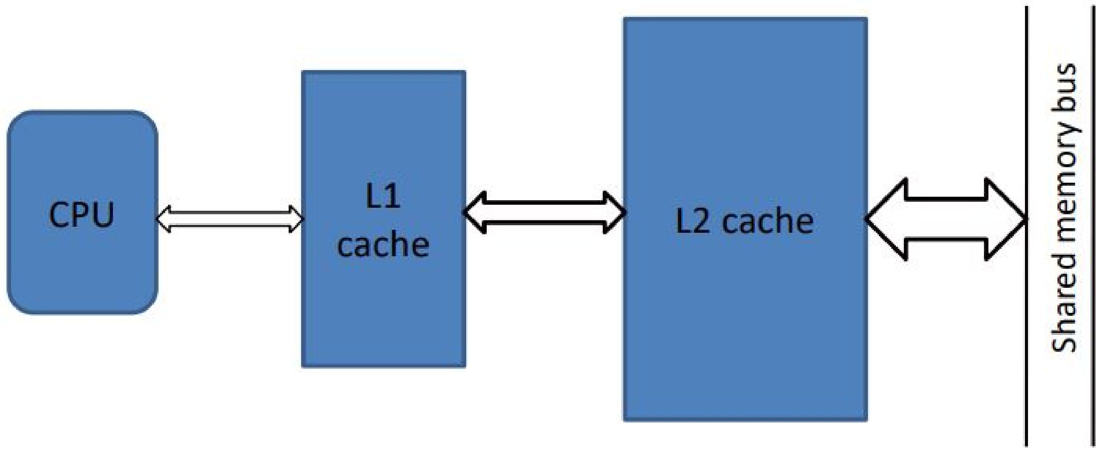
FINAL PROJECT REPORT

Last Level Cache Simulation



Sandeep Chintakunta Venkata

Todd Townsend

Ross Wegter

Fall 2019
Team 2

INTRODUCTION

Our Lower Level cache(L2) has a total capacity of 16MB, uses 64-byte lines, and is 8-way set associative. A write allocate policy is employed and uses the MESI protocol to ensure cache coherence. The replacement policy is implemented with a pseudo-LRU scheme. A higher level cache(L1) uses 64-byte lines, is 64 byte aligned, and is 4-way set associative. It employs a write-once policy, the first line write is write through; subsequent writes to the same line are write-back. Our system maintains inclusivity and returns snoop results returned by the lower level cache in response to our lower level cache bus operations. Our design has repeatable and easily changed results so that our simulated cache behaves properly for all return values.

DESIGN

Design is categorized into the following

- Main: Inputs for a design control and configuration files.
- Cache_memory: Structures to point the cache lines and sets
- Cache_control
  - Parameterized debug level, line_size, associativity and no of cache_lines
  - Reads the trace file line by line and do the respective operation
- Bus_operation
  - Contains all the communications LLC does with system bus and to the next level cache
- Cache_operation
  - Decodes the address into sets and tags based on input parameters
  - Initialize the cache by allocating the required memory as per the config.
  - Read requests: Determines cache operation and performs read and read with writes. Directs evicts if needed.Updates MESI states and PLRU bits.
  - Write requests: Updates PLRU bits and MESI states. Call read with write if needed.
  - All Snoop requests: Calls updateMESIsnoop by passing respective snoop.
  - Reset of the cache: Invalidates all the cache lines in all the sets.
  - Print all the information of the cache line in each set if all lines are valid.
  - Update MESI state for transaction:This changes the state when the read and write transactions occurred and also performs relevant bus operation.

    - Update MESI State for snoop requests: This is used to update the MESI state for all the snoop requests And also invalidates and flush when required.
    - UpdatePLRU: Updates bits of an abstracted Least Recently Used Tree per given way of cache line tag. 0 is for left traversal, 1 for right traversal.
    - Which way to evict: If else branching to simulate traversal of abstracted Pseudo LRU directed opposite the PLRU bits.
    - Flush : This is used when flush data when required and sends the data to the DRAM through bus.
    - Void way: This is used for invalidating or evicting the way
- Cache_performence
    - Counters: We are using counters to validate number of operations.
    - Hit and miss ratios : calculate the performance values based on Hits/Misses/Eviction counters
    - AMAT: Calculating AMAT by assuming HIT as 5 cycles and MISS as 100 cycles
- Logger
    - Displays all the logs based on the debug mode levels
        - Level 0: Displays the response of operation & performance results
        - Level 1: Displays all the Bus operations.
        - Level 2: Displays basic details of cache operation
        - Level 3: Displays all details of cache operation
        - Level 4: Displays all the performance data information
        - Level 5: Displays all the control mechanism details

COMMUNICATION

LLC needs to communicate with the higher level cache as well as the BUS we take care all this communications during the state change keeping in the mind of multi level cache. We need to get the data from L1 whenever our LLC cache MESI state is modified as L1 can have updated data.

TESTING

1. trace_mesi.file to verify all the MESI state transitions and bus operations during it
    a. For MESI transaction change checking following scenarios.
        i. I-E-E-M , I-S-S-M-M, I-M
    b. For MESI snoop requests change the following scenarios covers all.

            i. M-S-S-I, M-I, E-S-I

2. Trace_plru.file to verify access and replacement of Pseudo LRU
    a. Read operations to fill each way of a single set until full, then subsequent reads again into each of the filled ways to evict once from each. PLRU bits display in sequential order from root, counting up from the left to right leaves. Evicts from a full set show evict way order of 1, 5, 3, 7, 2, 6, 4, 8.
3. Trace_op89.file to verify the operation 8 and 9 to reset and print cache.
    a. Made sure after reset this is not printed when print operation is executed
4. Cc1.din used to compare performance for different cache lines.

COUNTERS

- Hit Counts: Elaborate various types of hit counts
    - IreadHitCount, DreadHitCount, DwriteHitCount;
- Miss Counts with empty cache line
    - IreadMissCount, DreadMissCount, DwriteMissCount;
- Evict Counts when eviction is caused.
    - IreadMissEvictCount, DreadMissEvictCount, DwriteMissEvictCount;
- Total individual Operational Counts
    - IreadCount, DreadCount, DwriteCount;

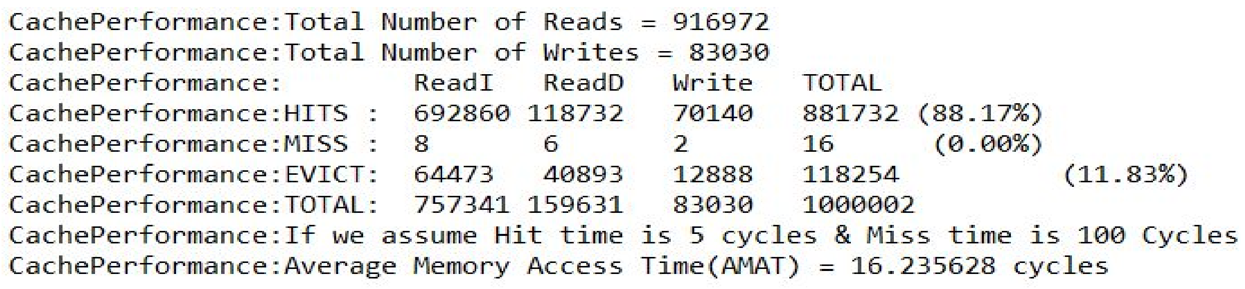
CODE REFERENCE:

We has a team worked on version control for our development to sync and effectively work on the final project here is our repository details

- **https://github.com/cvsandeep/MSD-Cache.git**

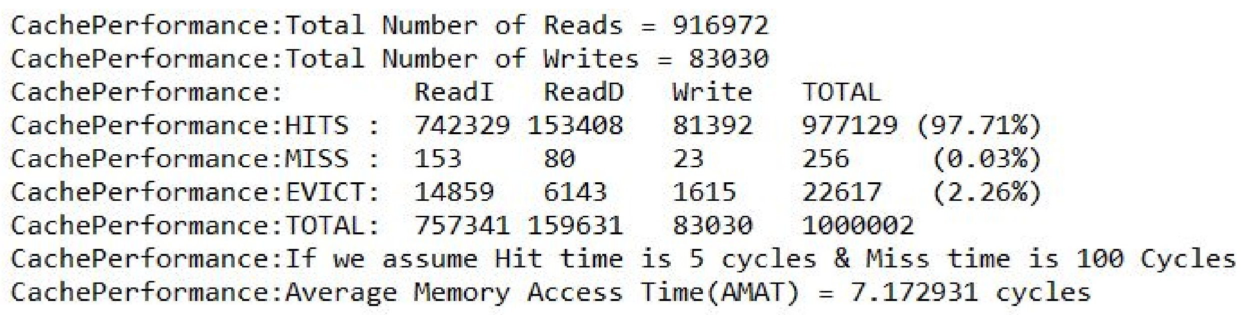
PERFORMANCE

Cache Lines = 16



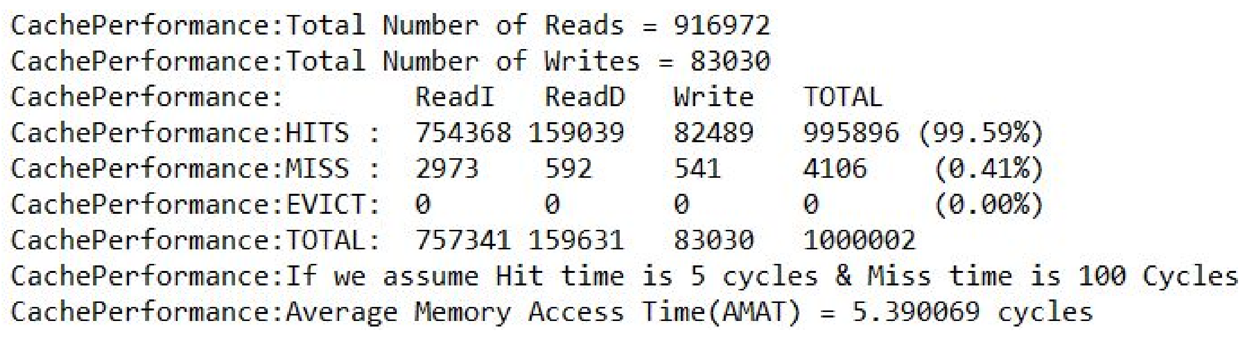
```
CachePerformance:Total Number of Reads = 916972
CachePerformance:Total Number of Writes = 83030
CachePerformance:         ReadI   ReadD   Write   TOTAL
CachePerformance:HITS :   692860 118732   70140   881732 (88.17%)
CachePerformance:MISS :   8       6       2       16      (0.00%)
CachePerformance:EVICT:   64473   40893   12888   118254          (11.83%)
CachePerformance:TOTAL:   757341 159631   83030   1000002
CachePerformance:If we assume Hit time is 5 cycles & Miss time is 100 Cycles
CachePerformance:Average Memory Access Time(AMAT) = 16.235628 cycles
```

Cache Lines = 256



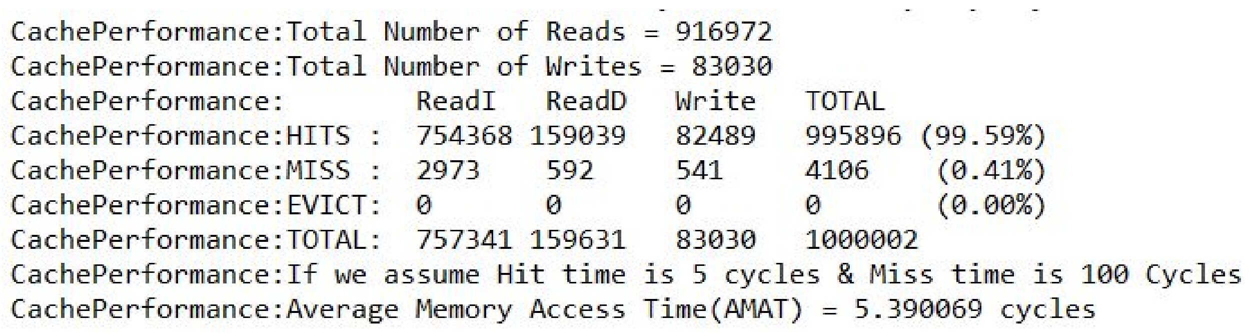
```
CachePerformance:Total Number of Reads = 916972
CachePerformance:Total Number of Writes = 83030
CachePerformance:         ReadI   ReadD   Write   TOTAL
CachePerformance:HITS :   742329 153408   81392   977129 (97.71%)
CachePerformance:MISS :   153     80      23      256     (0.03%)
CachePerformance:EVICT:   14859   6143    1615    22617   (2.26%)
CachePerformance:TOTAL:   757341 159631   83030   1000002
CachePerformance:If we assume Hit time is 5 cycles & Miss time is 100 Cycles
CachePerformance:Average Memory Access Time(AMAT) = 7.172931 cycles
```

Cache Lines = 32,768



```
CachePerformance:Total Number of Reads = 916972
CachePerformance:Total Number of Writes = 83030
CachePerformance:         ReadI   ReadD   Write   TOTAL
CachePerformance:HITS :   754368 159039   82489   995896 (99.59%)
CachePerformance:MISS :   2973    592     541     4106    (0.41%)
CachePerformance:EVICT:   0       0       0       0       (0.00%)
CachePerformance:TOTAL:   757341 159631   83030   1000002
CachePerformance:If we assume Hit time is 5 cycles & Miss time is 100 Cycles
CachePerformance:Average Memory Access Time(AMAT) = 5.390069 cycles
```

Cache Lines = 262,144



```
CachePerformance:Total Number of Reads = 916972
CachePerformance:Total Number of Writes = 83030
CachePerformance:         ReadI   ReadD   Write   TOTAL
CachePerformance:HITS :   754368 159039   82489   995896 (99.59%)
CachePerformance:MISS :   2973    592     541     4106    (0.41%)
CachePerformance:EVICT:   0       0       0       0       (0.00%)
CachePerformance:TOTAL:   757341 159631   83030   1000002
CachePerformance:If we assume Hit time is 5 cycles & Miss time is 100 Cycles
CachePerformance:Average Memory Access Time(AMAT) = 5.390069 cycles
```