Sreeramamurthy Tripuramallu
903057502

ECE 3056 Assignment 5 Report

## A) Data Structures

There are two primary data structures used to implement the cache, both of which are structs. The first struct defines each block of the cache, it consists of a four parameters: valid, dirty, tag, and LRUTime. These values correspond to the valid bit, dirty bit, tag value, and the amount of time spent on the LRU stack (LRUTime). The second struct represents a line, which corresponds to a pointer to a block array.

```
typedef struct {
    int valid;
    int dirty;
    int tag;
    int LRUTime;
} block;

typedef struct {
    block* lineBlock;
} line;
```

Figure 1: Source Code Representing Data Structures (block and line) for Cache

## B) Function to Parse Address

```
int numBits = calcPower(blockSize);
int numIndexBits = calcPower(cacheSize/(blockSize*numWays));
int tagValue = physical_addr >> (numBits + numIndexBits);
int index = physical_addr >> numBits;
int notTag = (tagValue) << numIndexBits;
notTag = ~notTag;
index = index & notTag;
```

Figure 2: Source Code to Parse Physical Address

```
int calcPower(int num) {
    int power = 0;
    while (num != 1) {
        num /= 2;
        power++;
    }
    return power;
}
```

Figure 3: Source Code for New Method that Calculates Number of Bits

A separate function was not created to parse the physical address; rather, only a new method (calcPower) was created to calculate the number of bits of the block and index, by doing a log base 2 operation on the input value. The number of bits for the block is simply the output of the calcPower with blockSize as the input. Also, the number of bits for the index is the output of calcPower with the cacheSize divided by the product of the blockSize and

associativity as the input. Upon calculating these two values, the tag value is then computed by shifting the physical address with the sum of the number of block bits and index bits. Lastly, the index is determined by using shifting and masking operations on bit sizes of the block and index, as well as the tag value.

**C) Implementation of cache LRU Stack**

The LRU Stack is implemented by using a globalTime counter variable. When the cache is accessed and there is a hit, the LRUTime for the block is updated to the globalTime, then the globalTime is incremented. If there is a miss and an empty block in the cache, the address needs to be added to the cache, and the LRUTime value of block gets set to the globalTime, and the globalTime is incremented. Lastly, if there is a miss and no empty block, then the block with the minimum LRUTime is found and evicted. Overall, the higher the LRUTime value of the block, the more recent the corresponding address has been accessed.

**D) Analysis From Section 3**

As the line size increased, both *trace.bubble* and *trace.merge* exhibited similar results. Initially, the miss rate decreased till a line size of 512; once the cache line became 1024 the miss rate increased. In turn, the optimal line configuration for a 128Kbyte unified set associative cache is 512. Seen in the two tables and plots, once the line size goes beyond 512, the miss rate begins to increase.

Table 1: Results for *trace.bubble* with varying line size

| Line Size | 128 Kbyte | Associativity | Accesses | Hits | Misses | Writebacks | Overall Miss Rate |
|---|---|---|---|---|---|---|---|
| 32 | 131072 | 4 | 6322343 | 6306878 | 15465 | 4470 | 0.002446087 |
| 64 | 131072 | 4 | 6322343 | 6312721 | 9622 | 2763 | 0.001521904 |
| 128 | 131072 | 4 | 6322343 | 6315673 | 6670 | 1807 | 0.001054989 |
| 256 | 131072 | 4 | 6322343 | 6317146 | 5197 | 1312 | 0.000822005 |
| 512 | 131072 | 4 | 6322343 | 6317712 | 4631 | 1115 | 0.000732482 |
| 1024 | 131072 | 4 | 6322343 | 6317475 | 4868 | 1119 | 0.000769968 |
| 2048 | 131072 | 4 | 6322343 | 6315169 | 7174 | 1538 | 0.001134706 |

Table 2: Results for *trace.merge* with varying line size

| Line Size | 128 Kbyte | Associativity | Accesses | Hits | Misses | Writebacks | Overall Miss Rate |
|---|---|---|---|---|---|---|---|
| 32 | 131072 | 4 | 7678430 | 7657295 | 21135 | 8228 | 0.002752516 |
| 64 | 131072 | 4 | 7678430 | 7665634 | 12796 | 4799 | 0.001666487 |
| 128 | 131072 | 4 | 7678430 | 7669628 | 8802 | 3010 | 0.001146328 |
| 256 | 131072 | 4 | 7678430 | 7671662 | 6768 | 2076 | 0.00088143 |
| 512 | 131072 | 4 | 7678430 | 7672564 | 5866 | 1631 | 0.000763958 |
| 1024 | 131072 | 4 | 7678430 | 7672354 | 6076 | 1575 | 0.000791308 |
| 2048 | 131072 | 4 | 7678430 | 7670144 | 8286 | 1882 | 0.001079127 |

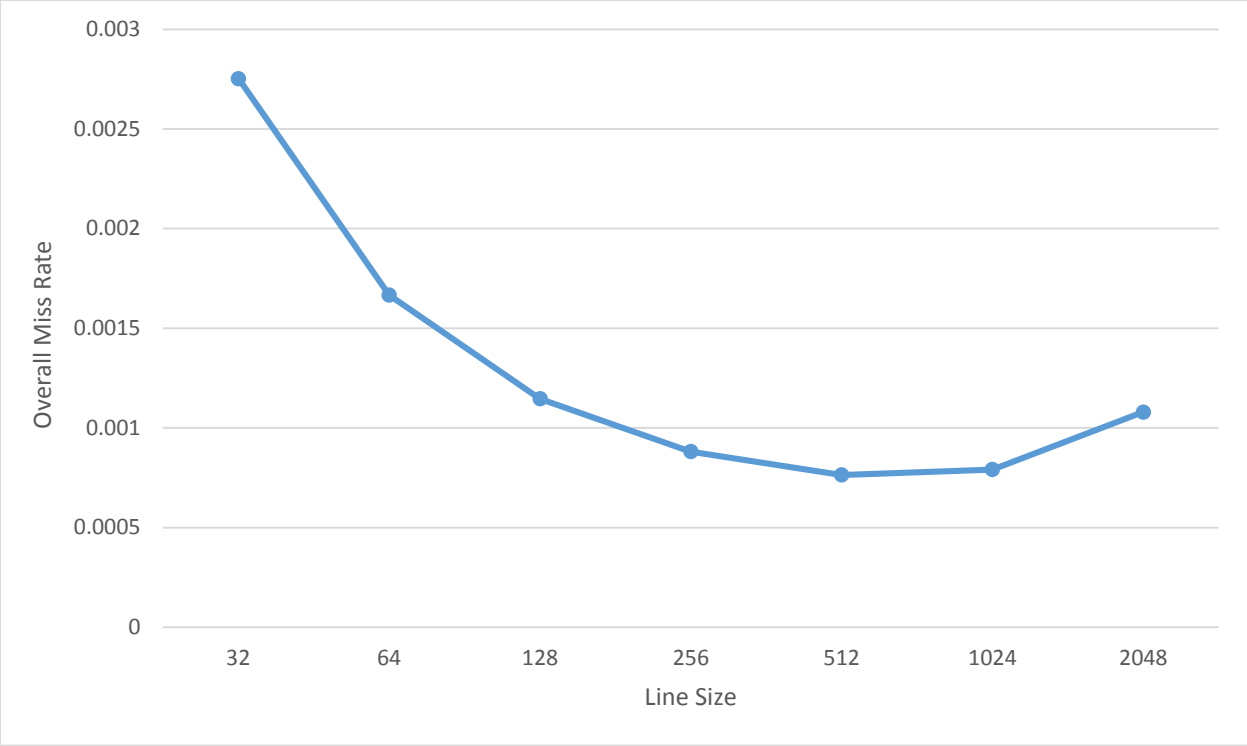


Figure 1: Plot of the effect of line size on miss rate for 128Kbyte unified set associative cache for *trace.bubble*

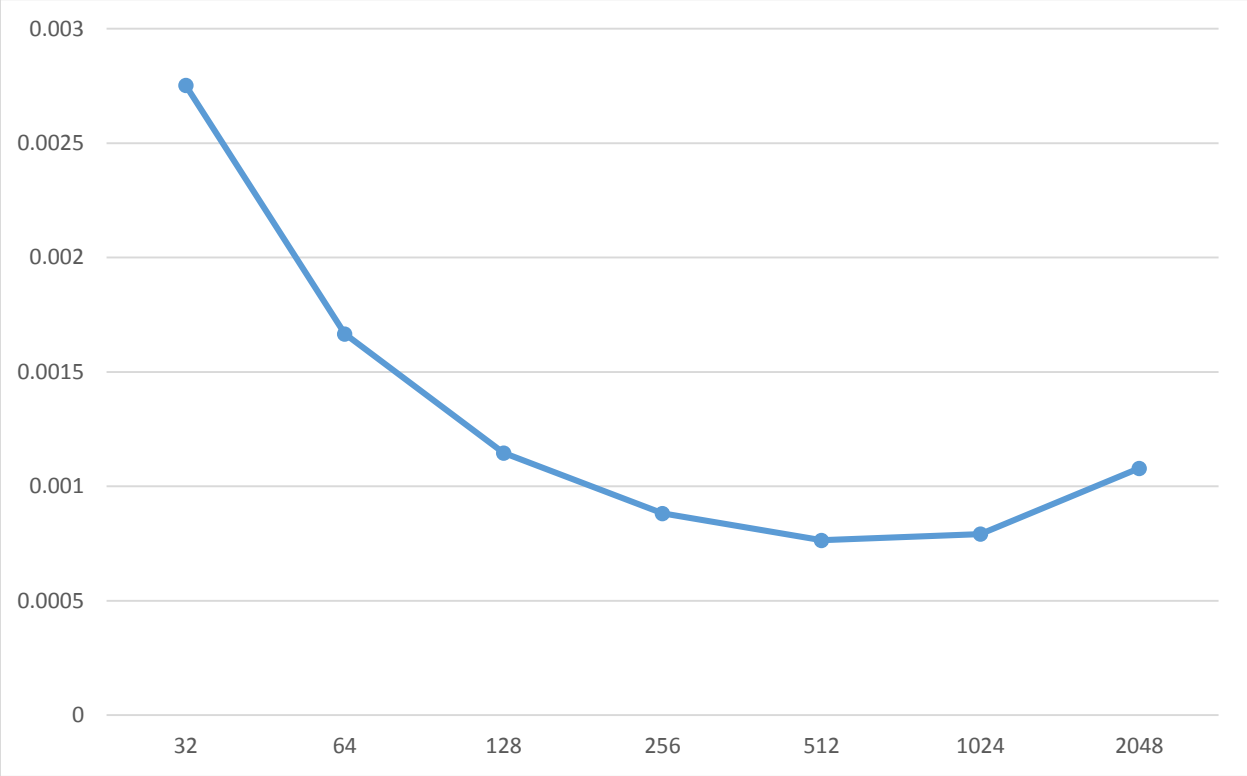


Figure 2: Plot of the effect of line size on miss rate for 128Kbyte unified set associative cache for *trace.merge*