Computer System Design Lab: CS4110

Ishu Dharmendra Garg - CS13B060 Sanchit Agrawal - CS13B061

Cache Address Translation Scheme & Replacement Strategy in Intel Processors

Introduction

Caches: Caches are memory structures used in CPUs to reduce the average memory access time. They work on the principle of locality of reference which states that if memory address A is accessed at time T, then A and addresses close to it will likely be accessed in time T + ΔT. Hence when a memory location is requested, an entire block containing that memory location is fetched and stored in the cache. Since the cache has lower access time than the RAM, the average memory access time decreases.

Cache Structure: The cache can be modeled as a 2-D array where the rows are referred to as sets, and the columns are called ways. Each cell of the cache holds a block of memory.

Cache Address Translation Scheme: This term refers to how blocks in the physical memory (RAM) are mapped to blocks in the cache. One common address translation scheme is to map a block #B to the set #S if B = S (mod N) (here N is the number of sets in the cache). However, in general, any hash function can be used to map blocks to the cache sets.

Cache Replacement Strategy: In the case of a cache miss, the missing block is fetched from the memory and placed in the corresponding set. If the set is full, one of the blocks from the set must be replaced by the incoming block. The block which is replaced is known as the victim block. The policy which dictates how the victim block is chosen is known as the cache replacement strategy. The most common cache replacement strategy is the Least Recently Used (LRU) policy, under which the block which was least recently used is the victim block.

Understanding the behavior of Last Level Caches in Intel Processors

The assignment requires us to infer the cache address translation scheme, and the cache replacement strategies used in Intel processors.

The cache used in modern computer is typically set associative. A cache with associativity A is also called an A-way set-associative cache. The organization of a cache is specified by the following parameters:

A: The associativity of cache (also the number of ways).

B: The block (also called cache line) size in bytes.

N: The number of cache sets

$W=B * N$: The cache capacity in bytes in one way.

$S=A * W$: The cache capacity in bytes.

For the L3 cache on our system, these values are:

A = 16

B = 64 B

N = 4K

W = 256KB

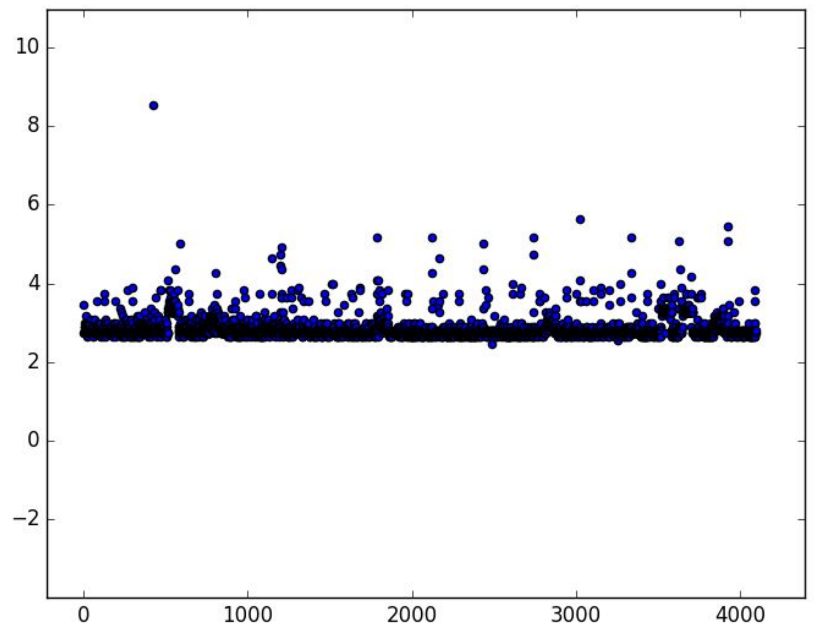
S = 4MB

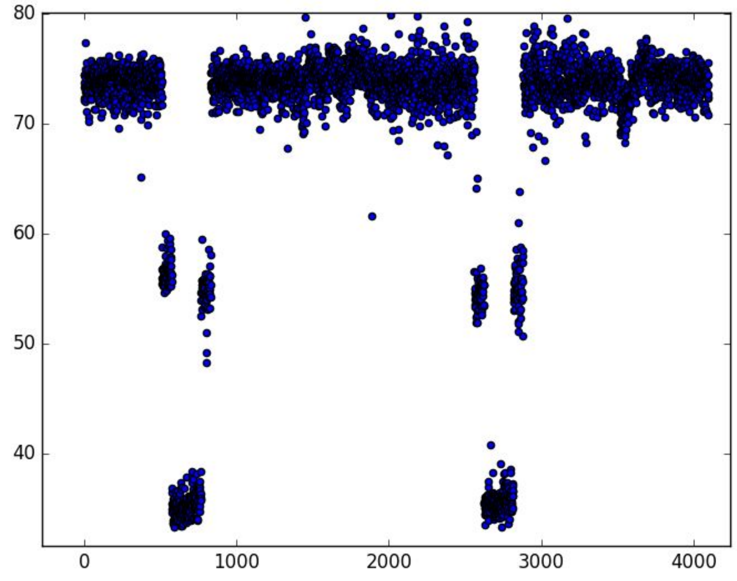
We perform the experiments explained below in order to determine the translation scheme, and the replacement policy.

Experiment 1:

```
Base = memAlloc (2∗S);
assoc = 1;
while ( assoc < maxAssoc ) {
    set = 0;
    while ( set < N) {
        init ( Base + set , assoc );
        repeatAccess ( Base + set );
        printtime ();
        set = set + 1;
    }
    assoc = assoc + 1;
}
```

The main idea in this test is: at each assoc, the program traverses all the cache sets, and on each set it repeatedly and sequentially accesses a list with assoc items. In the innermost loop, the function init initializes the list with assoc items. The addresses for those items have the offset starting at the Base + B ∗ set and differ by n ∗ W. The repeatAccess executes the item access and record the access time. To eliminate the interferences come from system, this function repeatedly accesses the list for 1000 times. And we directly read the time stamp counter to get the exact execution time.





Here the results are shown for assoc = 10 for the first figure and assoc = 48 for the second figure.

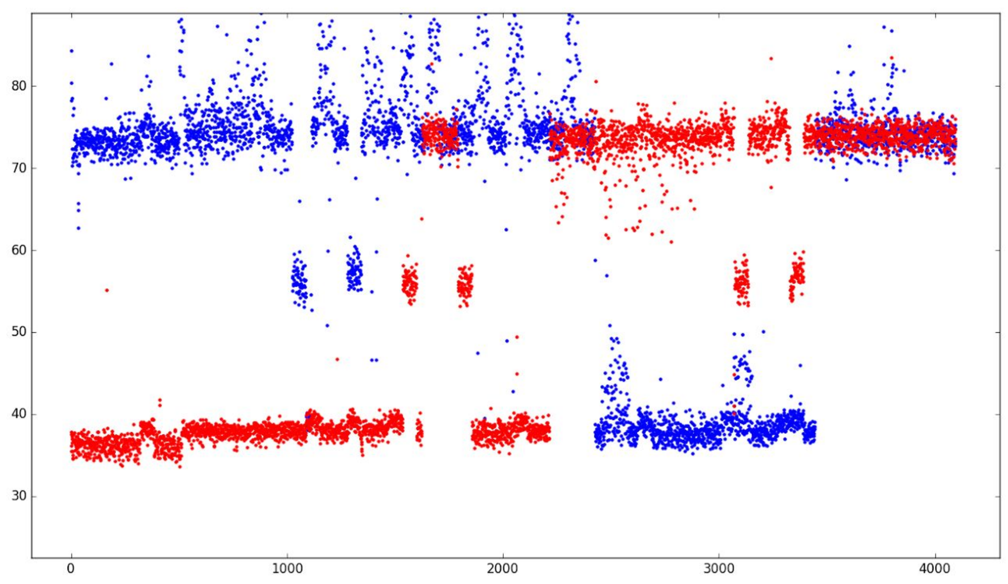
We observe that for assoc <= A, there are no cache misses, because the entire list can fit in the cache sets. This shows that the cache follows the % N rule for address translation.

For assoc > A, the list cannot fit in the cache set, and every read is necessarily a miss, if we assume the plain LRU policy. However, this is not what is observed. While most sets have a miss on every read, some sets still perform well. These sets might be following the Hit Promotion Policy, in which a newly written block has LRU status instead of MRU status, along with the Set Dueling based Dynamic Insertion Policy.

Under the SDDIP mechanism, there are two policies which dynamically compete to be the dominant policy on the cache. The Set Dueling mechanism dedicates a small number of sets to each of the two competing policies. The policy that incurs fewer misses on the dedicated sets is used for the remaining follower sets. Thus under SDDIP, most sets become adaptive to the access patterns. This mechanism is designed with the principle that the cache behavior can be approximated by sampling a small number of sets on the cache.

Experiment 2:

This is similar to experiment 1, but this time we have a fixed assoc value, and we iterate through the sets two times. The first time we go from the lowest set to the highest set and the next time from the highest set to the lowest set.



The above plot is shown for assoc = 50. The red plot shows the reverse scan, and the blue plot shows the forward scan.

We can clearly see that there are a large number of follower sets, interspersed with dedicated sets in between. When a large number of dedicated sets following the same policy as the dominant policy are encountered, then the policy is flipped.

Initially the blue plot follows the LRU policy, encountering a large number of misses. In between it encounters some dedicated MRU sets, until it finally encounters dedicated LRU sets, which flip the dominant strategy to MRU. Finally, towards the end, the strategy switches to LRU.

The same path can be retraced in reverse while following the red plot.

Thus we can see how the processor switches between two policies depending on which is performing better.

Conclusions

Through these two experiments, we verify that the Intel processor follows %N cache address translation, and a Set Dueling based Dynamic Insertion Policy which chooses the dominant strategy based on which leads to lesser cache misses. The two policies are:

1. LRU replacement where newly assigned blocks have MRU status.
2. LRU replacement where newly assigned blocks have LRU status until they are referenced by a READ request.

The way the policies are flipped is by setting aside some dedicated sets which follow a fixed policy (irrespective of the dominant policy followed by the follower sets). This provides a way of sampling the cache sets in order to choose a strategy.