REPORT

-by Prakash A(CS20B061),Rudra Laxmi Kanth(CS20B

Reverse Engineering On L1 Cache Memory - Assignment - 4

The assignment 4 is on the concept of measuring latency of hit and miss in a cache , Here we are using the _rdtsc(); a built in assembly function in C language which is used in measuring the latency time .

but we can also use inline assembly language of _rdtsc(), which is available or shown in : https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf

Here we are using _rdtsc(); function instead of inline assembly language for better accuracy since an inline assembly rdtsc and rdtscp instruction take more time for execution(in newer computer OS).

And in the C program we are using _mm_lfence() to make sure a memory barrier is created where we don't want the insruction to get executed via cache.(This is done to prevent anyother instruction or operation used in the C program to get into cache). And for more accurate measurement we have used (void)*((volatile*)) which is used for accessing the variable or array without casting it. we have also used _mm_clflush to make sure that the after a miss in L1 cache it doesnot get hit in L2 cache.

How to see Actual specifications of cache in a Computer

The L1 cache specifications present in our computer was 320 KB , which can be done by executing following command :
ctrl + shift + esc -> taskmanager -> go to performance -> where we can see all the cache specifications.

Explaination Or Methodology of How the code works to find Block Size.

The L1 cache block Size we got using reverse engineering is 64B.
We did this by fact that everytime we incure a miss in cache the processor takes a chunk or a block of data from main memory into the cache , so using this fact , we used an array access latency to calculate the block Size. The first miss in the cache is when accessing array[0] element , so that time we are not calculating since it is not useful. After that first miss , we would have a contiguous values in the cache , but the number of values depends on the block Size. so now to find out the block size we access the array elements starting from 0 to some value say 20 and calculate the latency time it takes for this access.

If the access latency value is near to the latency time of array[0] (That is , we are calculating the latency for arrray[0] after the first miss .) then the particular element is in the same L1 cache , but is suddenly the access time of an element increases that means we are incurring a miss in cache and going to main memory , this implies the particular value is not in the same block , Let us say at value n (n>0) we are incuring a miss and high latency then block size is n*(no fo bytes used for one array element).

Here we are using integer array elements , so Size is 4 bytes and the value we got a miss was at n = 16 , therefore we can infer that the block size = 16*4 = 64 Bytes .

In the code we are running and testing how much latency it takes to access and array element , by the data's or latency displaced below we can see that all the array elements from array[0] to array[15] takes nearly same time , this implies they are in same block , but after that at array[16] , there is large increase in latency , this means there is a miss in cache and therefore array[16] is not in the block .

cache latency of a[0] is 7

cache latency of a[1] is 8

cache latency of a[2] is 7

cache latency of a[3] is 8

cache latency of a[4] is 6

cache latency of a[5] is 8

cache latency of a[6] is 19

cache latency of a[7] is 29

cache latency of a[8] is 23

cache latency of a[9] is 7

cache latency of a[10] is 8

cache latency of a[11] is 5

cache latency of a[12] is 5

cache latency of a[13] is 9

cache latency of a[14] is 8

cache latency of a[15] is 4

cache latency of a[16] is 142

cache latency of a[17] is 11

cache latency of a[18] is 10

cache latency of a[19] is 11

cache latency of a[0] is 7

cache latency of a[1] is 6

cache latency of a[2] is 16

cache latency of a[3] is 3

cache latency of a[4] is 20

cache latency of a[5] is 12

cache latency of a[6] is 12

cache latency of a[7] is 5

cache latency of a[8] is 6

cache latency of a[9] is 5

cache latency of a[10] is 8

cache latency of a[11] is 4

cache latency of a[12] is 17

cache latency of a[13] is 4

cache latency of a[14] is 5

cache latency of a[15] is 6

cache latency of a[16] is 52

cache latency of a[17] is 61

cache latency of a[18] is 67

cache latency of a[19] is 7

The above ones are data or Numbers we got from Running the code , If we see clearly the Value of cache latency get increased at array[16] position suddenly , this is due to the miss in latency we explained before , Therefore we can conclude that the block of L1 cache can contain only 16 elements array[0...15] of 4 bytes (integer) this infers that block size - 64 B.

How to find Associativity of Cache

From the above code we can find the block size of the L1 cache , now to find the associativity of cache , we are going to load or access the different elements which are starting positions of a block , therefore array[0], array[16], array[32].... and so on. This is done so that one can see how many block a set in L1 cache can hold which is equal to associativity.
After bringing the blocks to cache one by one we should check cache latency of elements belonging to different blocks , so if we bring one block more than no of associativity , than a lesser frequent block should have got evicted , that means cache latency of an element when accessed should have cache latency larger than when other elements of other blocks are accessed , so if no of block brought when larger cache latency occurs = n ,then no of associativity = n-1. Generally associativity of a cache will be 4 or 8 , so we have check for this 2 only.

In calculating the associativity , in the code first we have accessed array[0], array[16] ... array[48], by doing so we will bring 16 elements as a block to L1 cache , thus we got 4 blocks into the L1 cache , and after that we are accessing array[0], it show a less value , which means it is L1 cache only , after that we are accessing a new block array[64] which brings a 5 th block it to L1 cache , if L1 cache was a 4 way associativity then one of the previous 4 block must get evicted and the access latency of evicted block must have increased , but it didnt happen , so the associativity of L1 cache > 4 , so we have check for associativity 8. In the we have we calculated the latency for all the first 4 block when 5th block is added , the latency time was nearly the same , that means , no block got evicted , next we checked for associativity 8 , when 9th block was added to cache, one of got evicted , that gave rise to high increase in latency , for one of the element which we can see from below data .

cache latency of a[0] is 381

latency - 4

cache latency of a[0] is 188

cache latency of a[16] is 163

cache latency of a[32] is 147

cache latency of a[48] is 172

latency - 8

cache latency of a[0] is 147

cache latency of a[16] is 209

cache latency of a[32] is 122

cache latency of a[48] is 139

cache latency of a[64] is 174

cache latency of a[80] is 114

cache latency of a[96] is 304

cache latency of a[112] is 196

cache latency of a[0] is 471

latency - 4

cache latency of a[0] is 178

cache latency of a[16] is 118

cache latency of a[32] is 184

cache latency of a[48] is 129

latency - 8

cache latency of a[0] is 618

cache latency of a[16] is 129

cache latency of a[32] is 128

cache latency of a[48] is 186

cache latency of a[64] is 139

cache latency of a[80] is 164

cache latency of a[96] is 186

cache latency of a[112] is 630

From the above data we can see that , the cache latency of the first 4 elements didnt change much when 5th block got added , where when 9th block got added , in first data , array[96] or array[96..111] block got evicted which showed large latency , similary in second data array[112] or array[112...127] block got evicted which showed large latency , this implies when 9th block gets added to L1 cache , one block is getting evicted.

Therefore the no of ways associativity of L1 cache is 8.

The each integer is of 32 bits = 4 bytes
we have 16 integers in one block
therefore block size = 16 * 4
= 64

When the 9th block got added one of a block got evicted when is evident from the data
therefore associativity = 8

Thus justified/proved.