Assingnment 4

cs20b070 Sasubilli Yuvan cs20b029 Garapati Tej Pavan

March 2, 2022

1 Finding Cache Block Size

1.1 Measuring the latency

a)The rdtsc (Read Time-Stamp Counter) instruction is used to determine how many CPU ticks took place since the processor was reset. Loads the current value of the processor's time-stamp counter into the EDX:EAX registers. It is commonly used as a timing defense (anti-debugging technique)

b)The RDTSCP instruction is not a serializing instruction, but it does wait until all previous instructions have executed and all previous loads are globally visible. 1. But it does not wait for previous stores to be globally visible, and subsequent instructions may begin execution before the read operation is performed.

c)We should put the instructions given in the intel mannual and access the memory location each time so the cycleslow will be calculated at the start and cycleslow1 will be calculated at the end on processors that support the Intel 64 architecture, the high order 32 bits of each of RAX, RDX, and RCX are cleared So we did left shift after the cycleslow and cycleslow1 are calculated, So for the elapsed time we can take the difference of the start and finish.

d)The cpuid instruction can force all previous instructions to finish

1.2 Actual specifications of the L1 cache present in my system (Lenove Thinkpad intel i5 10th Gen)

The command used to find the actual specifications of the L1 cache present in your system is
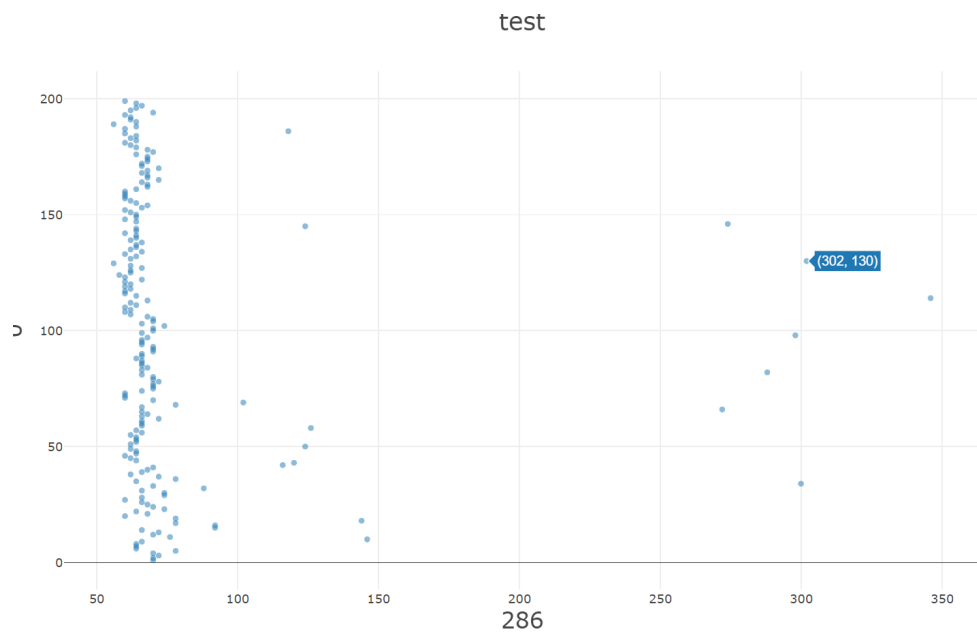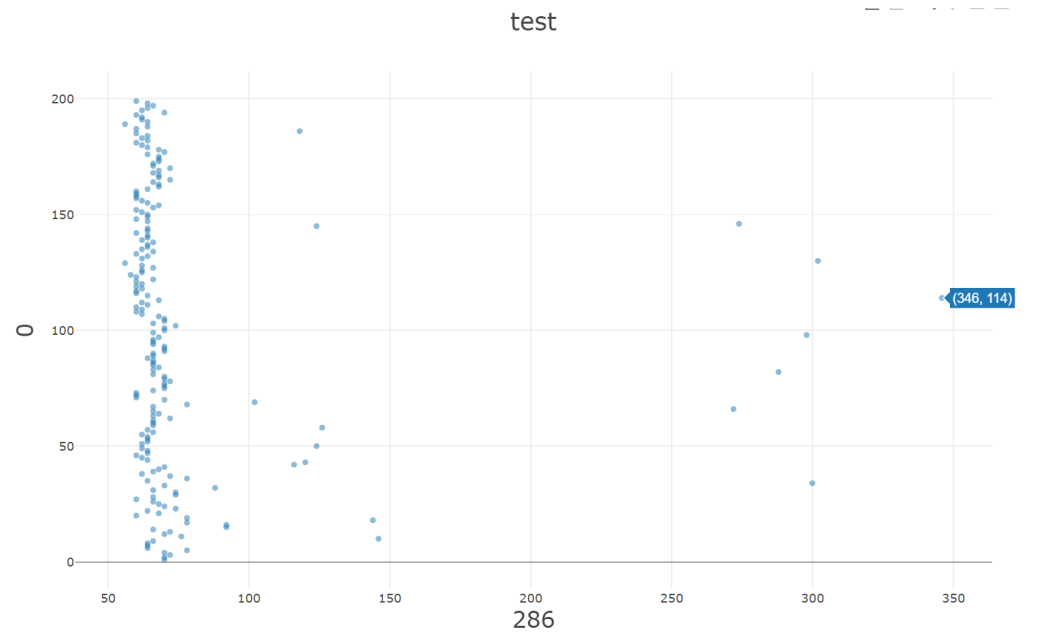**getconf -a | grep CACHE**

```
sasubilli@sasubilli-VirtualBox: ~
Selecting previously unselected package giblib1:amd64.
Preparing to unpack .../giblib1_1.2.4-12_amd64.deb ...
Unpacking giblib1:amd64 (1.2.4-12) ...
Selecting previously unselected package scrot.
Preparing to unpack .../archives/scrot_1.2-2_amd64.deb ...
Unpacking scrot (1.2-2) ...
Setting up libid3tag0:amd64 (0.15.1b-14) ...
Setting up libimlib2:amd64 (1.6.1-1ubuntu0.1) ...
Setting up giblib1:amd64 (1.2.4-12) ...
Setting up scrot (1.2-2) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
sasubilli@sasubilli-VirtualBox:~$ getconf -a | grep CACHE
LEVEL1_ICACHE_SIZE              32768
LEVEL1_ICACHE_ASSOC             8
LEVEL1_ICACHE_LINESIZE          64
LEVEL1_DCACHE_SIZE              32768
LEVEL1_DCACHE_ASSOC             8
LEVEL1_DCACHE_LINESIZE          64
LEVEL2_CACHE_SIZE               262144
LEVEL2_CACHE_ASSOC              4
LEVEL2_CACHE_LINESIZE           64
LEVEL3_CACHE_SIZE               6291456
LEVEL3_CACHE_ASSOC              12
LEVEL3_CACHE_LINESIZE           64
LEVEL4_CACHE_SIZE               0
LEVEL4_CACHE_ASSOC              0
LEVEL4_CACHE_LINESIZE           0
sasubilli@sasubilli-VirtualBox:~$ scrot
```

1.3   Finding Cache Block Size Via Reverse Engineering

a)Define an int array of size 200 and initialise two unsigned integer variables start and finish to measure the cyclestime by rdtsc and rdtscp registers respectively.
b) using mmclflush operation we ensured that the cache is free by accessing each element in the int array and we created a temporary variable and ran the loop for 200 iterations
c) We then gave temp = a[i] in between the asm volatile instructions used to measure the time
d) We removed the higher order 32 bits and equated them to start and finish times and calculated the elapsed time by etime = start − finish
e)We used file handling to get the output as a csv file and used csv to plot software to get the i values (corresponding) index of memory accesses when we observe a significant jump
f)From the concept we can understand that the jump is observed when a block is completely filled and we incurred a miss in cache
g)Here if we observe the values of the jumps we found that the difference in memory access is 16 (for int)
h)Interference : int occupies 4 bytes so 16*4 = **64** bytes which is the size of the block

test

200

150

O

100

50

0

50   100   150   200   250   300   350

(346, 114)

286

test

200

150

100

50

0

50   100   150   200   250   300   350

(302, 130)

286

1.4   Justification

The actual cache block size = **64** bytes which is matched from the experiment.
Here we observe a first miss so we observe 286 at the first iteration which is a

compulsory miss and the other times were very nearer but this was observed after running mulitple times and the same wasn't observed after every run. Then when the jumps start there is a jump after every 16 iterations starting from 18,34,50,66,... and in the diagram you can find 114 th and 130 th iterations so the cache block size is **64** from the experiment !!! and it **matches** with my intel i5 processor :)

2 Finding Associativity of Cache

## 2.1 Finding Number of sets from the grep command

b)Define an int array of size 1024 and initialise two unsigned integer variables start and finish to measure the cyclestime by rdtsc and rdtscp registers respectively.

b) using mmclflush operation we ensured that the cache is free by accessing each element in the int array and we created a temporary variable and ran the loop for 200 iterations

c) Repeat the same process above but for 1024 iterations of an integer array

d) We got the block size to be **64B** and the L1 data cache size is **32768B** and the line size is **64** which means 64 sets from this we get we know that the cache blocks that are brought for LOAD requests to addresses 0, 512, 1024, 1536,2048 , etc... go to the same cache set. Every time, when we bring a new cache block to a cache set, we initiate access to all the blocks that are already brought in earlier to the cache set and observe the access latency. If the access latencies are almost the same, indicating that the cache blocks are present in the cache set. So **number of ways is 512B/64B (size of cache block) = 8**

e)We can see that the jumps are observed at 64 intervals (64*8 here long long int) taken are giving the same time
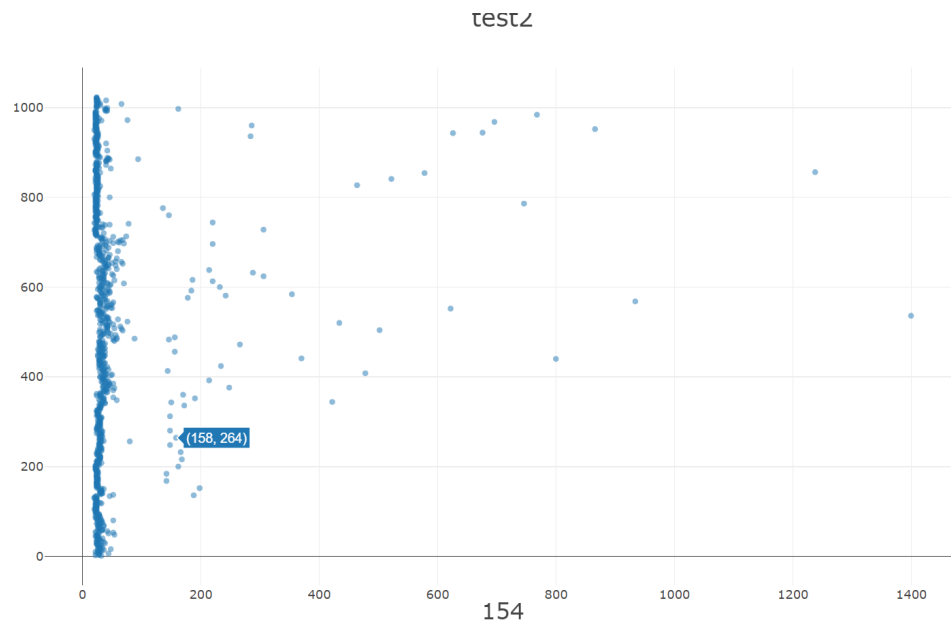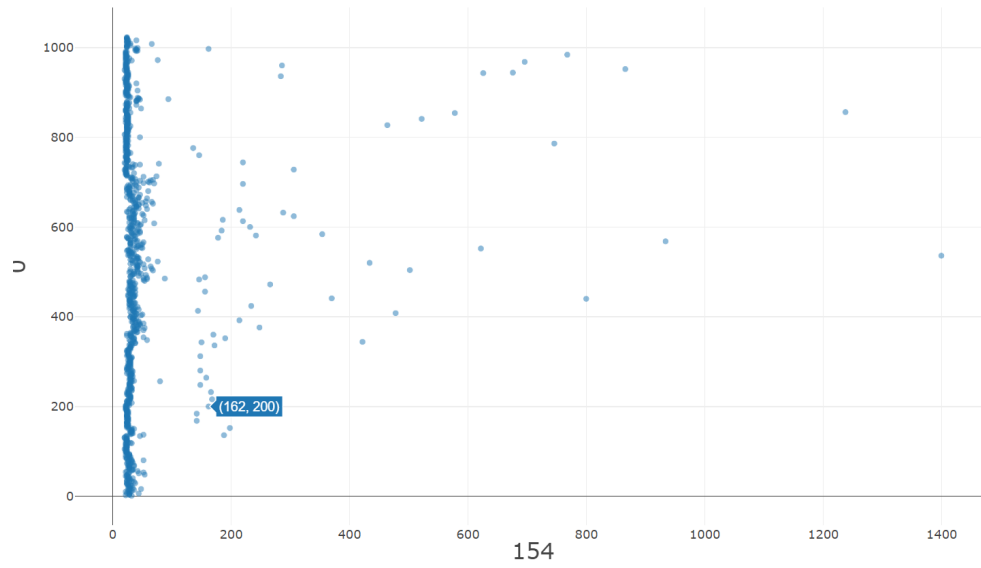
2.2   Justification

The time taken for both **200th** iteration is 158 and **264th** iteration is 162 which is almost similar and for every jump which has an interval of **64 B (long long int)** is same which means for every **512** we are going to the same set So the set

associativity is **8** from the experiment !!! and This also **matches** with my intel
i5 processor :)