CS6600: Computer Architecture
Assignment 1

Arjun Menon Vadakkeveedu
EE18B104

August 22, 2021

1 Approach

The goal of the assignment is to estimate the block size and set associativity of the L1 cache on a Desktop PC. For this task, the evaluation program was run on the i5-8250U processor by Intel.
To evaluate the block size, a sufficiently large byte addressable array (char datatype) was allocated in the heap memory and initialised. The blocks are then flushed from L1 cache using the CLFLUSH instruction of the x86 ISA. Since the block size is not known, the address of each byte was passed to the instruction to ensure no block from the char array is contained in L1 cache. Once all bytes are evicted from the cache, the elements of the array are addressed one-by-one starting from the base address.

If the accessed byte is absent in the cache, the block containing the byte is brought into the cache resulting in higher access latency. The latency for accessing the byte is evaluated using the RDTSC instruction. The above exercise of flushing the blocks followed by accessing them byte-by-byte is repeated multiple times to reduce the effects of noise.
Noises may arise from the cache usage characteristics of background processes and software and hardware interrupts that may add delays to the execution of the program. Also, in order to avoid Out-of-Order execution while executing the RDTSC instruction, the execution is serialised by performing the CPUID instuction before RDTSC. By design, the CPUID instruction is constrained to be performed in a serialised manner. A drawback of this solution is that the reported number of cycles is not the true memory access latency; however we only require the difference in access latency to estimate the cache parameters.

To find the cache associativity, we must first identify the blocks that get mapped to the same set. Since the number of sets is also unknown, we access blocks with relative addresses that are multiples of the cache size - the cache size is found using the **lscpu** command in Linux systems. For the i5-8250U processor, the L1 data cache has a size of 32K. Bytes with addresses of the form *base_addr*, *base_addr + 32K*, *base_addr + 2*32K* up to *base_addr + (R-1)*32K* are accessed, with R chosen to be large enough to populate the set with blocks containing these bytes. In doing so, the P blocks containing the latest accessed

bytes occupy the set, where P is the associativity of the L1 cache.
Next, the bytes are accessed in the reverse order, starting with address *base_addr + (R-1)*32K* up to address *base_addr*. Blocks containing addresses *base_addr + (R-1)*32K* to *base_addr + (R- P)*32K* will be met with minimal latency due to cache hit, while all subsequent accesses would incur at least the miss penalty of L1 cache miss. This difference in access times is used to estimate the set associativity.

2 Results

Looking at Figure 1, peaks in access latency are observed at uniform intervals of 64 bytes. At the same time, there is a considerable effect of noise at the beginning (peak at 2) and for indices after about the first 512 bytes. In addition, the peak access latency drops after the first few peaks with some of the peaks being non-apparent. This may be due to a hardware pre-fetching optimisation employed by the processor after seeing the access pattern.
The best estimate of the block size is hence 64 bytes. This matches with the value reported in the processor's datasheet.

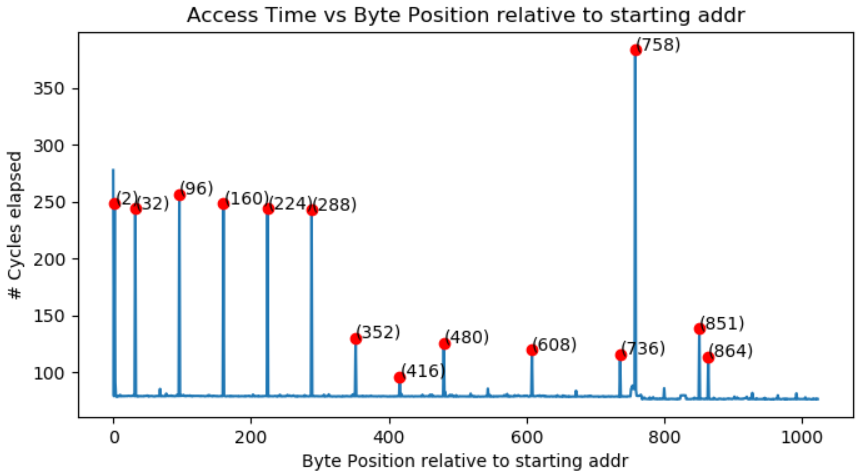

Figure 1: Access Latency vs Index rel. to Base Address of accessed byte

Figure 2 describes the latency behaviour when multiple blocks mapped to the same set are accessed one after the other. The step at index 8 clearly indicates that the L1 cache is 8-way associative.

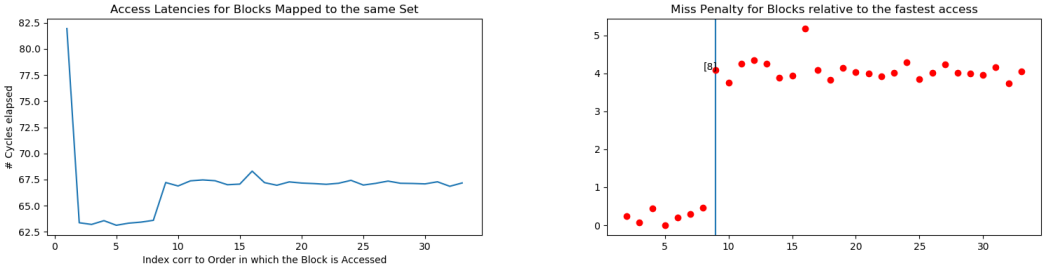

Figure 2: Latency for accessing blocks mapped to the same set; (right) excess penalty incurred for accesses not contained in set

---

⋆⋆⋆