CS6600 Computer Architecture (Jul-Nov 2021)
Assignment-1

Karthikeyan R EE18B015
Nithin Babu EE18B021

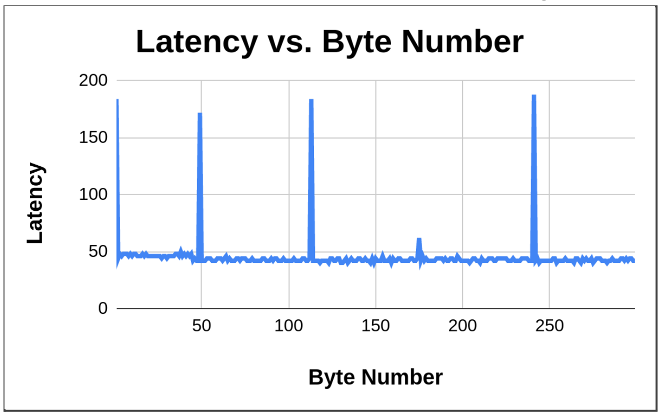
Identifying cache block size

To identify the block size of a cache, the steps followed in *blocksize.c* are:

- Dynamically allocate a large chunk of memory of type 'char' using *malloc()*
- Flush the cache using *_mm_clflushopt()* function included in *x86intrin.h* header file
- Access the first 300 bytes of the allocated memory consecutively and measure the access latency time using *_rdtsc()* function included in *x86intrin.h* header file.

Observations are:

- The initial byte will be a definite miss as the cache is empty and hence we see a high latency time for the initial byte. But the consecutive bytes will be hits resulting in low latency times, since the entire block will be brought into the cache. These low latency times are observed till we reach the end of a cache block, after which accessing the next byte will again have a higher latency.
- Doing this for around 300 bytes, will give us high latencies for the bytes placed at memory locations that are multiples of block size width away from the initial byte.
- In our system, we found out this width to be 64 by plotting the recorded latency data and observing the distance between peaks in the plot.
- **Hence, our cache block size is 64 Bytes**



Identifying cache associativity

We utilize the following data to determine our cache associativity:

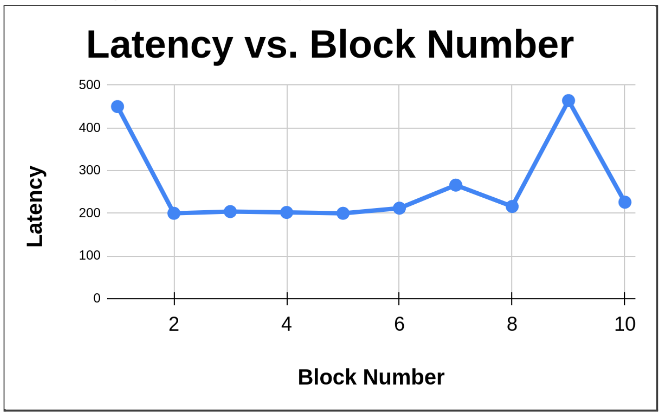
- L1 Cache size of our system using *lscpu* command in linux (128KB in our system)
- Block size of our cache is determined as explained in the previous section. This is 64 Bytes in our system. Hence, in our system cache will have a total of 2048 blocks.

To identify the associativity of the cache, the steps followed in *associativity.c* are:

- Assume the associativity to be either 2-way, 4-way or 8-way (as these are divisible by 2048). This will result in the number of sets to be 1024, 512, and 256 respectively. Set these values appropriately in the file using macros.
- Dynamically allocate a large chunk of memory of type 'char' using *malloc()*. Flush the cache using *_mm_clflushopt()* function.
- We then access only those bytes that belong to different blocks, but belong to the same set by keeping their set index same. Latency time to access these bytes are recorded using *__rdtsc()* function.

Observations are:

- Once a set is fully occupied, accessing a new byte in a different block belonging to the same set will result in a very high latency because placing this new block in cache will also involve evicting an already existing block from the set using a block replacement policy.
- We assume a cache associativity and record access latencies. We assume a cache associativity of 2 way, 4 way and 8 way to record access latency.
- If we observe a high access latency for the *(associativity+1)th* block, this assumed associativity will be the correct associativity of our cache.
- For eg., if we assume 4-way associative cache and if we get high access latency for the 5th block, then the cache indeed would be 4-way associative.
- In our system, we were able to observe a high access latency for the 9th block when 8-way associativity is assumed. **This confirms the 8-way associativity of our cache.**



Note:

To improve interpretability of our latency data:

- Prefetching was disabled using *noprefetch* pragma
- Compiler optimizations were switched off
- *sfence* and *lfence* blocking instructions were used