An Introduction to Cache Organizations

Suresh Purini, IIIT-H

A *cache* is a high speed memory that sits transparently between the processor and the main memory, buffering instructions or data that the processor is likely to access in the near future. Caches are useful to bridge the speed mismatch between the processor and the main memory. Caches will be effective when the application programs exhibit good *spatial* or *temporal locality* property. It has to be noted that all programs does not show good locality property, especially those that use pointer based data structures like trees and graphs.

There are broadly 3-different ways of organizing a cache memory: *direct mapped*, *fully associative* and *set associative* cache organizations. We shall understand the structure of different cache organizations assuming the following parameters.

- Size of main memory is $M = 2^m$.
- Size of cache is $C = 2^c$.

In all the cache organizations, main memory and cache are logically divided into a collection of blocks of size $B = 2^b$. So the total number of main memory and cache blocks are respectively given by $t_m = 2^{m-b}$ and $t_c = 2^{m-c}$.

1. **Direct Mapped Cache Organization** In this cache organization, a main memory block l gets mapped to the cache block $l \ mod \ t_c$. Since multiple main memory blocks can possibly map to the same cache block we need extra *tag* bits to identify which main memory block is occupying a given cache block.

2. **Fully Associative Cache Organization** In this cache organization a main memory block can reside in any of cache blocks. The main memory block occupying a particular cache block is identified by using *tag* bits. This is the most flexible cache organization but expensive to realize in hrdware.

3. **K-way Set Associative Cache Organization** In this cache organization, the cache is divided into a collection of sets with each set containing k-consecutive cache blocks. If there are t_s sets available in total, then a main memory block l gets mapped to the set $l \ mod \ t_s$ and within the set, the main memory block can occupy any of the cache blocks. Again here we need *tag* to exactly identify which main memory block is occupying which cache block.

Direct mapped cache organization offers fast look-up time but no flexibility in main memory block placement. This could result in *conflict misses*. In fully associative cache organization there is full flexibility in main memory block placement but the fast hardware look-up table needed would be expensive. The advantage however is all the cache misses are either *compulsory* or *capacity misses*. K-way set associative mapping strikes a trade-off between these two extreme approaches.

Apart from the cache orgnization there are few other policies that have to be decided while designing caches. During a *write* memory access should we allocate a cache block to a main memory block or not. If we allocate a cache block on a *write miss*, it is called *write allocate* policy. Otherwise it is called *write no allocate* policy. The next design decision is, whether to simultaneously update the cache and memory on a *write* memory access or not. If we simulataneously update the cache and main memory, then it is called as a *write through* policy. On the otherhand, if we update the main memory at the time of cache block eviction, then it is called as a *write back* policy.

In k-way associative mapping and fully associative mapping, when all the candidate cache blocks are full, we have to evict a cache block to promote a new main memory block not already present in cache. There are many possible strategies for *cache eviction* like random, FIFO, LRU etc. All these design parameters will have an impact on the eventual performance of a cache organization.