The Comparison Between Three Kind of Cache Mapping Methods

- ## Direct Mapped Cache:

The direct mapped cache is the simplest form of cache and the easiest to check for a hit. Since there is only one possible place that any memory location can be cached, there is nothing to search; the line either contains the memory information we are looking for, or it doesn't.

Unfortunately, the direct mapped cache also has the worst performance, because again there is only one place that any address can be stored. Let's look again at our 512 KB level 2 cache and 64 MB of system memory. As you recall this cache has 16,384 lines (assuming 32-byte cache lines) and so each one is shared by 4,096 memory addresses. In the absolute worst case, imagine that the processor needs 2 different addresses (call them X and Y) that both map to the same cache line, in alternating sequence (X, Y, X, Y). This could happen in a small loop if you were unlucky. The processor will load X from memory and store it in cache. Then it will look in the cache for Y, but Y uses the same cache line as X, so it won't be there. So Y is loaded from memory, and stored in the cache for future use. But then the processor requests X, and looks in the cache only to find Y. This conflict repeats over and over. The net result is that the hit ratio here is 0%. This is a worst case scenario, but in general the performance is worst for this type of mapping.

- ## Fully Associative Cache:

The fully associative cache has the best hit ratio because any line in the cache can hold any address that needs to be cached. This means the problem seen in the direct mapped cache disappears, because there is no dedicated single line that an address must use. However (you knew it was coming), this cache suffers from problems involving searching the cache. If a given address can be stored in any of 16,384 lines, how do you know where it is? Even with specialized hardware to do the searching, a performance penalty is incurred. And this penalty occurs for *all* accesses to memory, whether a cache hit occurs or not, because it is part of searching the cache to determine a hit. In addition, more logic must be added to determine which of the various lines to use when a new entry must be added (usually some form of a "least recently used" algorithm is employed to decide which cache line to use next). All this overhead adds cost, complexity and execution time.

- N-Way Set Associative Cache:

The set associative cache is a good compromise between the direct mapped and set associative caches. Let's consider the 4-way set associative cache. Here, each address can be cached in any of 4 places. This means that in the example described in the direct mapped cache description above, where we accessed alternately two addresses that map to the same cache line, they would now map to the same cache *set* instead. This set has 4 lines in it, so one could hold X and another could hold Y. This raises the hit ratio from 0% to near 100%! Again an extreme example, of course. As for searching, since the set only has 4 lines to examine this is not very complicated to deal with, although it does have to do this small search, and it also requires additional circuitry to decide which cache line to use when saving a fresh read from memory. Again, some form of LRU (least recently used) algorithm is typically used.

Conclusion :

Cache Type	Hit Ratio	Search Speed
Direct Mapped	Good	Best
Fully Associative	Best	Moderate
N-Way Set Associative, N>1	Very Good, Better as N Increases	Good, Worse as N Increases

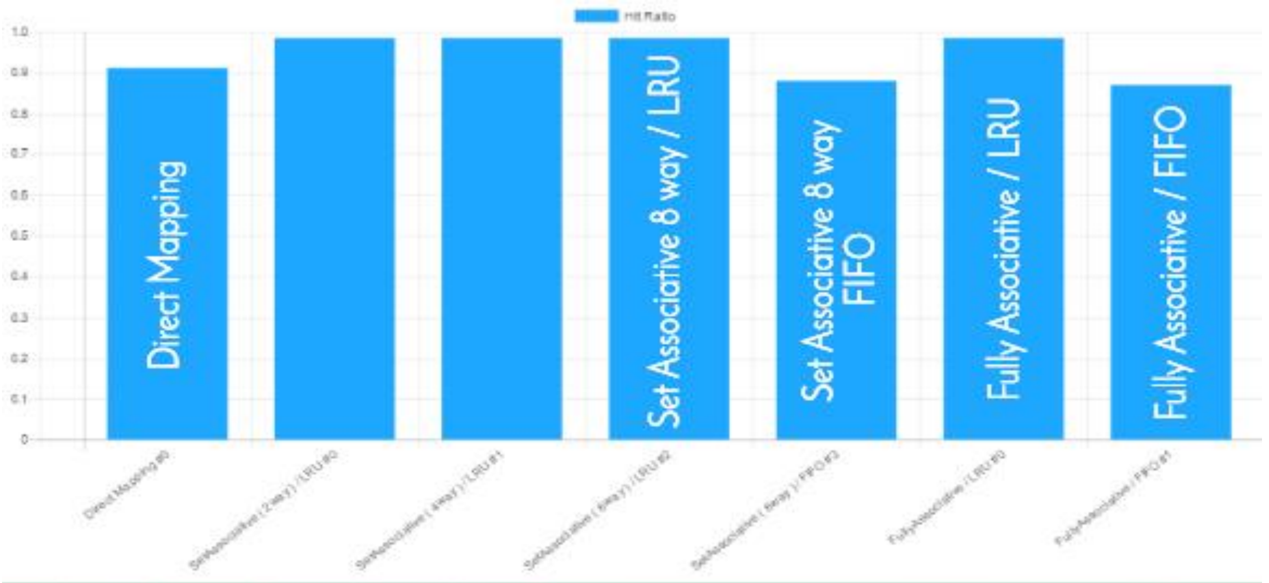
In The Real World :

the **direct mapped** and **set associative** caches are by far the most common. Direct mapping is used more for level 2 caches on motherboards, while the higher-performance set-associative cache is found more commonly on the smaller primary caches contained within processors.

The Comparison Between Cache Replacement Algorithms (LRU vs. FIFO)

No perfect caching policy exists because it would require knowledge of the future (how a program will access memory). But, some are measurably better than others in the common memory access pattern cases. This is the case with LRU. Because if your process is frequently accessing a page, you really don't want it to be paged to disk, even if it was the very first one you accessed. On the other hand, if you haven't accessed a memory page for several days, it's unlikely that you'll do so in the near future.

Here is a simple Analysis with I.U.S.T Online Cache Simulator


Hit Ratio
Direct Mapping
Set Associative 8 way / LRU
Set Associative 8 way FIFO
Fully Associative / LRU
Fully Associative / FIFO
Direct Mapping #0
SetAssociative (2way) / LRU #0
SetAssociative (4way) / LRU #1
SetAssociative (8way) / LRU #2
SetAssociative (8way) / FIFO #3
FullyAssociative / LRU #0
FullyAssociative / FIFO #1