Cache organization and Optimization

Sumit Yadav; Usha Verma & chhavi Bhardwaj
(ryadav1918@gmail.com),
(usha.verma1991@gmail.com),
(chhavi985@gmail.com)

Abstract –

Cache memory is the fastest and most expensive memory. As we know computer programmer always want unlimited amount of fast memory, thus a small amount of cache memory is used which stores frequently used data. That data is stored in cache in form of blocks, because of cache organization. Different techniques are used for cache organization. To reduce the miss rate and miss time and to increases hit ration different optimization techniques are used. Different cache organization and optimization methods are discussed.

1. INTRODUCTION

Cache memory is the fastest and most expensive memory.

Computer programmer always want unlimited amount of fast memory. An economical solution to that desire is a memory hierarchy, which takes advantage of locality and cost- performance of memory technology. This principle, plus the guide line that smaller hardware can be made faster, led to hierarchies based on memories of different speeds and sizes. The memory system consists of a hierarchy of storage elements. Excluding the register set, the cache has the shortest access time, or latency, of all the levels of the storage system, and the highest bandwidth.

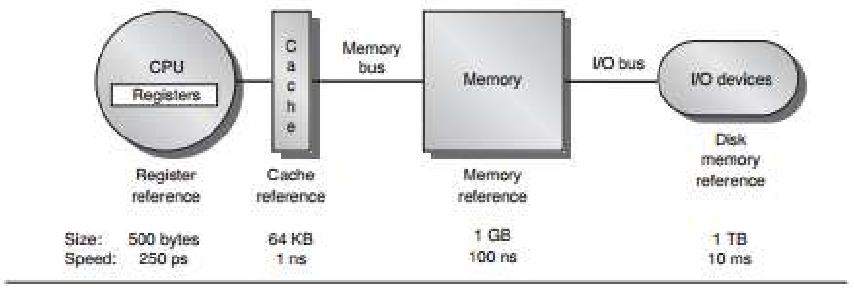


Fig.:- The levels in a typical memory hierarchy in embedded, desktop, and server computers.

Cache memory works as intermediate between CPU and main memory. As we know our CPU works at a very fast speed but compare to that main memory work at a very slow speed. Thus while fetching the data from main memory CPU had to sit idle. That is waste of time, thus throughput is low. So cache memory is used between the CPU and main memory. Cache memory stores the frequently used data in it, so that CPU can retrieve the data at a very fast speed.

Since fast memory is expensive, a memory is organized into several levels-each smaller, faster, and more expensive per byte than the next lower level. The goal is to provide a memory system with cost per byte as low as the cheapest level of the memory and speed almost as fast as the fastest level.

The importance of the memory hierarchy has increased with advances in performances of processors.

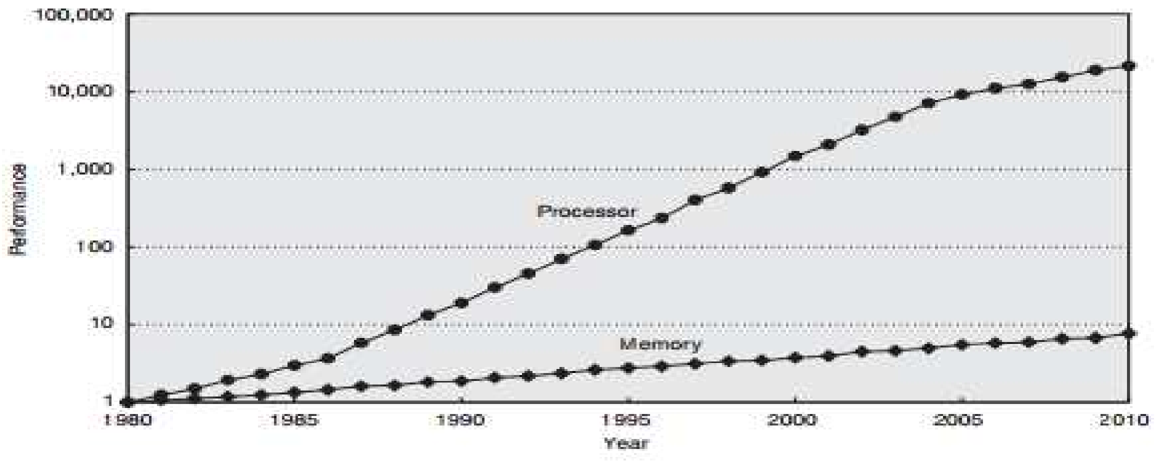


Fig:- Starting with 1980 performance as a baseline, the gap in performance between memory and processors is plotted over time.

2. CACHE ORGANIZATION

A cache may be organized to fetch on demand or to prefetch data. The former organization, usually referred to as demand fetch organization, is the most commonly used.

A demand fetch cache brings a new memory locality into the cache only when a processor reference is not found in the current content. The prefetch cache attempts to anticipate the locality about to be requested by the processor and thus prefetches it into the cache.

When a word is not found in the cache, the word must be fetched from the memory and placed in the cache before continuing. Multiple words, called a block (or line), are moved for efficiency reasons. Each cache block includes a tag to see which memory address it corresponds to.

Cache hits:- Processor references that are found in the cache are called cache hits.

Cache misses:- References that are not found in the cache are called cache misses.

When a cache miss occurs, cache control mechanism fetches the missing data from cache memory.

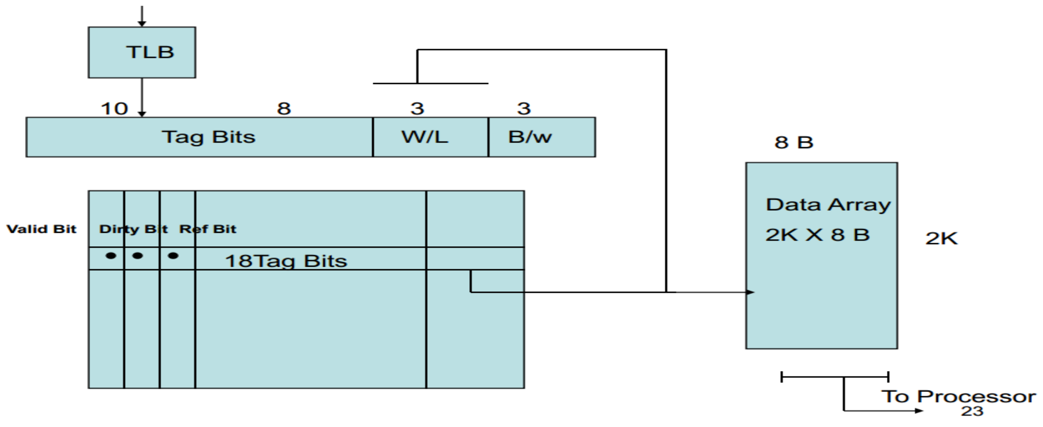
3. TYPES OF CACHE ORGANIZATION

Cache treats main memory as a set of blocks. As the cache size is much smaller than main memory so the numbers of cache lines are very less than the number of main memory blocks. So a procedure is needed for mapping main memory blocks into cache lines. cache mapping scheme affects cost and performance. There are three methods in block placement-

- **Direct Mapped Cache (**Each address has a specific each address has a specific place in the cache**).**
- **Fully Associative Mapped Cache (**Search the entire cache for an address**).**
- **Set Associative Mapped Cache (**Each address can be in any of a small set of cache locations**)**

1) Fully Associative Mapped Cache

In **fully associative mapping** when a request is made to the cache, the requested address is compared in a directory against all entries in the directory. If the requested address is found, the corresponding location in the cache is fetched and returned to the processor, otherwise a cache miss occurs.

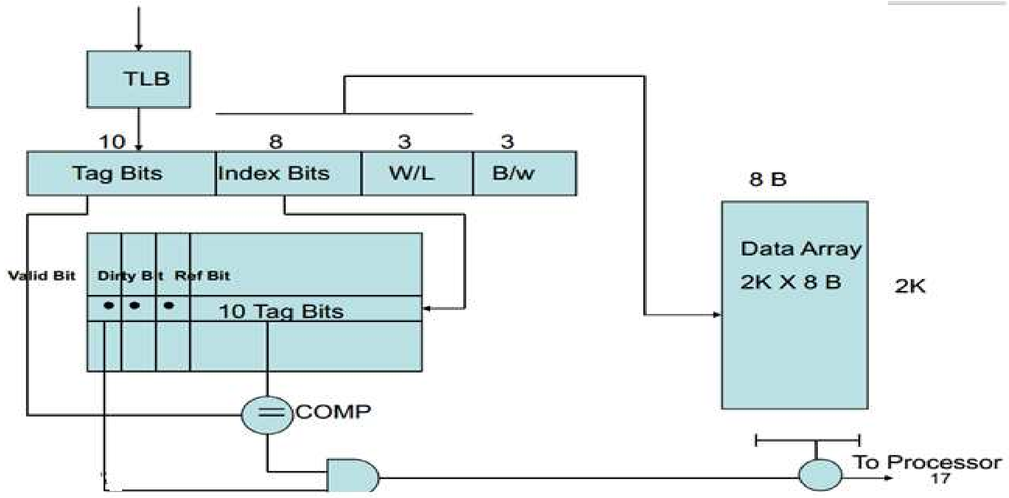


Steps –

1. Translate VPN (virtual page number) to RPN (real page number) with TLB.
2. Access the cache directory, then the cache.
3. Access cache with address from the directory index and offset bits.
4. If compare valid, enable data to go to processor.

2) Direct Mapped Cache

In **direct mapping** lower order line address bits are used to access the directory. Since multiple line addresses map into the same location in the cache directory, the upper address bits must be compared with the directory address to ensure a hit. If a comparison is not valid, the result is a cache miss.



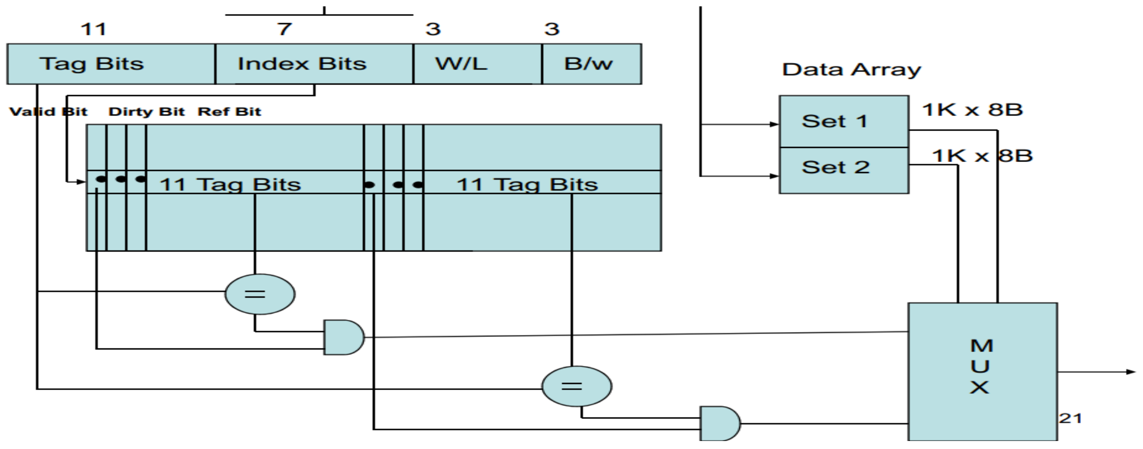
Steps –

1. Translate VPN (virtual page number) to RPN (real page number) with TLB.
2. Access cache array and directory simultaneously and compare tag with directory entry to ensure correct line is accessed.
3. Access cache array with index and offset bits.
4. If compare valid, enable data to go to processor.

3) Set Associative Mapped Cache

In **set associative mapping,** cache operates in a fashion somewhat similar to the direct mapped cache. Bits from the line address are used to address a cache memory. However, now there are multiple choices: two, four or more complete line addresses maybe present in the directory. Each of these line addresses corresponds to a location in a sub cache. The collection of these sub-cache forms the total cache array.

In a set associative cache, as in the direct mapped cache, all of these sub arrays can be accessed simultaneously, together with the cache directory. If any of the entry in the cache directory matches the reference address, and there is a hit, that particular sub cache array is selected and out gated back to the processor. While selection in the out gating process adds somewhat to the cache access time, the set associative cache access time is generally better than that of the associative mapped cache.



Steps –

1. Translate VPN (virtual page number) to RPN (real page number) with TLB.
2. Access cache array sets and cache directory entries to ensure correct line is in cache.
3. Compare tags from directory with tag address bits.
4. If compare valid, select corresponding set and MUX data to processor.

Still, from an access time consideration alone, the direct mapped cache provides the fastest processor access to cache data for any given size cache.

4. ISSUES

One measure of the benefits of different cache organizations is miss rate. Miss rate is simply the fraction of cache accesses that result in a miss—that is, the number of accesses that miss divided by the number of accesses.

To gain insights into the causes of high miss rates, which can inspire better cache designs, the three Cs model sorts all misses into three simple categories:

■ Compulsory—The very first access to a block cannot be in the cache, so the block must be brought into the cache. Compulsory misses are those that occur even if you had an infinite cache.

■ Capacity—If the cache cannot contain all the blocks needed during execution of a program, capacity misses (in addition to compulsory misses) will occur because of blocks being discarded and later retrieved.

■ Conflict—If the block placement strategy is not fully associative, conflict misses (in addition to compulsory and capacity misses) will occur because a block may be discarded and later retrieved if conflicting blocks map to its set.

miss rate can be a misleading measure for several reasons. Hence, some designers prefer measuring misses per instruction rather than misses per memory reference (miss rate). These two are related:

Misses = miss rate*memory =
accesses miss rate* memory
accesses
Instruction Instruction count
instruction

(It’s often reported as misses per 1000 instructions to use integers instead of fractions.) For speculative processors, we only count instructions that commit.
The problem with both measures is that they don’t factor in the cost of a miss. A better measure is the average is the average memory access time:

Average memory access time= hit time + miss rate * miss penalty

Where hit time is the time to hit in the cache and Miss penalty is the time to replace the block from memory (that is, the cost of a miss). Average memory access time is still an indirect measure of performance; although it is a better measure than miss rate, it is not a substitute for execution time.

5. ADVANCED OPTIMIZATIONS OF CACHE PERFORMANCE

The average memory access time formula above gives us three metrics for cache optimizations: hit time, miss rate, and miss penalty.

- Advanced cache optimizations are into the following categories:
- Reducing the hit time: small and simple caches, way prediction, and trace Caches
- Increasing cache bandwidth: pipelined caches, multibank caches, and non blocking caches.
- Reducing the miss penalty: critical word first and merging write buffers
- Reducing the miss rate: compiler optimizations
- Reducing the miss penalty or miss rate via parallelism: hardware prefetching and compiler prefetching

Small and Simple Caches to Reduce Hit Time

A time-consuming portion of a cache hit is using the index portion of the address to read the tag memory and then compare it to the address. Smaller hardware can be faster, so a small cache can help the hit time. It is also critical to keep an L2 cache small enough to fit on the same chip as the processor to avoid the time penalty of going off chip.

The second suggestion is to keep the cache simple, such as using direct mapping. One benefit of direct- mapped caches is that the designer can overlap the tag check with the transmission of the data. This effectively reduces hit time.

Way prediction to reduce hit time

Another approach reduces conflict misses and yet maintains the hit speed direct-mapped. In way prediction, extra bits are kept in the cache to predict the way, or block within the set of the next cache access. This prediction means the multiplexer is set early to select the desired block and only a single tag comparison is performed that clock cycle in parallel with reading the cache data. A miss results in checking the other blocks for matches in the next clock cycle.

Added to each block of a cache are block predictor bits. The bits select which of the blocks to try on the next cache access. If the predictor is correct, the cache access latency is the fast hit time. If not, it tries the other block, changes the way predictor, and has a latency of one extra clock cycle. Simulations suggested set prediction accuracy is in excess of 85% for a two-way set, so way prediction saves pipeline stages more than 85% of the time. Way prediction is a good match to speculative processors, since they must already undo actions when speculation is unsuccessful. The Pentium 4 uses way prediction.

Trace Caches to Reduce Hit Time

A challenge in the effort to find lots of instruction-level parallelism is to find enough instructions every cycle without use dependencies. To address this challenge, block in trace cache contain dynamic traces of the executed instructions rather than static sequences of instructions as determined by layout in memory. Hence, the branch prediction is folded into the cache and must be validated along with the addresses to have a valid fetch.

Pipelined Cache Access to Increase Cache Bandwidth

This optimization is simply to pipeline cache access so that the effective latency of a first-level cache hit can be multiple clock cycles, giving fast clock cycle time and high bandwidth but slow hits.

Critical Word First and Early Restart to Reduce Miss Penalty

This technique is based on the observation that the processor normally needs just one word of the block at a time. This strategy is impatience: Don’t wait for the full block to be loaded before sending the requested word and restarting the processor. Here are two specific strategies:

- Critical word first—Request the missed word first from memory and send it to the processor as soon as it arrives; let the processor continue execution while filling the rest of the words in the block.
- Early restart—Fetch the words in normal order, but as soon as the requested word of the block arrives, send it to the processor and let the processor continue execution.

Compiler Optimizations to Reduce Miss Rate

Thus far, our techniques have required changing the hardware. This next technique reduces miss rates without any hardware changes.

This magical reduction comes from optimized software. The increasing performance gap between processors and main memory has inspired compiler writers to scrutinize the memory hierarchy to see if compile time optimizations can improve performance. Once again, research is split between improvements in instruction misses and improvements in data misses.

Hardware Prefetching of Instructions

Instruction prefetch is frequently done in hardware outside of the cache. Typically, the processor fetches two blocks on a miss: the requested block and the next consecutive block. The requested block is placed in the instruction cache when it returns, and the prefetched block is placed into the instruction stream buffer. If the requested block is present in the instruction stream buffer, the original cache request is canceled, the bloch is read from the stream buffer, and the next prefetch request is issued.

6. REFERENCES

[1]. Michael J. Flynn- Computer Architecture: Pipelined and Parallel Processor Design

[2]. John L. Hennessy-Stanford University", "David A. Patterson- University of California at Berkeley"--Computer Architecture: A Quantitative Approach, Fourth Edition

[3]. Cache mapping-http://williams.comp.ncat.edu/comp375/CacheMapping.pdf

[4]. CPU cache"-http://en.wikipedia.org/wiki/CPU_cache

[5]. cache organization-http://www.cs.utexas.edu/~fussell/courses/cs429h/lectures/Lecture_19-429h.pdf

[6]. Andreas Moshovos- Advanced Computer Architecture-http://www.eecg.toronto.edu/~moshovos/ACA05/001-intro.pdf

[7]. virtual lab-IIT Kharagpur, computer organization and Architecture- http://virtual-labs.ac.in/labs/cse10/ac.html

[8]. Cache organization-http://ecee.colorado.edu/~ecen2120/Manual/caches/cache.html