Cache and Its Performance

Preeti Chhabra, Radhika Gogia, Rupa Kumari
Dronacharya College of Engineering, Gurgaon, India
chhabra.preeti28@gmail.com[1]
gogia.radhika13@gmail.com[2]
rupasingh252@gmail.com[3]

Abstract-

This paper presents a basic idea of cache memory and its performance. 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. The goal that an effective memory system will have to achieve is that the effective access time that the processor sees is very close to t_0, the access time of the cache. Here in this paper, we will first see an overview and then we we will study how to improve the performance of a cache by various methods.

***Keywords*- Hit Ratio, Miss ratio, Miss rate, Miss penalty, performance**

I. INTRODUCTION

A CPU cache is a cache used by the central processing unit (CPU) of a computer to reduce the average time to access data from the main memory. The cache is a smaller, faster memory which stores copies of the data from frequently used main memory locations. Most CPUs have different independent caches, including instruction and data caches, where the data cache is usually organized as a hierarchy of more cache levels (L1, L2 etc.). Basic performance metrics for cache are :

Hit ratio(h)= Number of memory references that hit in the cache / total number of memory references
Typically h = 0.90 to 0.97
Miss ratio m = 1 –h

II. CACHE PERFORMANCE OVERVIEW

When the processor needs to read from or write to a location in main memory, it first checks whether a copy of that data is in the cache. If so, the processor immediately reads from or writes to the cache, which is much faster than reading from or writing to main memory.
Average memory access time is a useful measure to evaluate the performance of a memory-hierarchy configuration.
Average memory access time:
AMAT = Hit Time + Miss Rate × Miss Penalty

There are three ways to improve cache performance:
1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache

A. MISS RATE REDUCTION

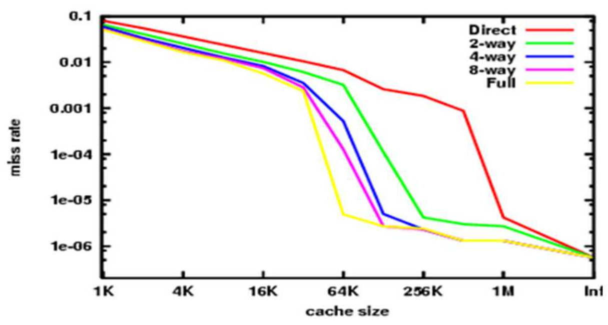
We can classify the misses as 4Cs :

- Compulsory : The first access to a block is not in the cache, so the block must be brought into the cache. Also called *cold start misses* or *first.*

- *Capacity*—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieve.

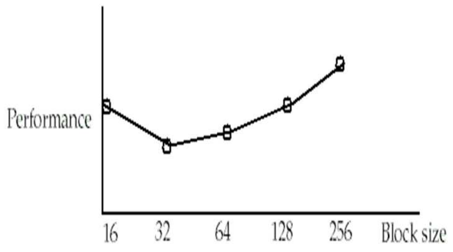
- *Conflict*—If block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called *collision misses* or *interference misses*.

- *Coherence* - Misses caused by cache coherence: data may have been invalidated by another processor or I/O device.



There are various ways to reduce the miss rate:

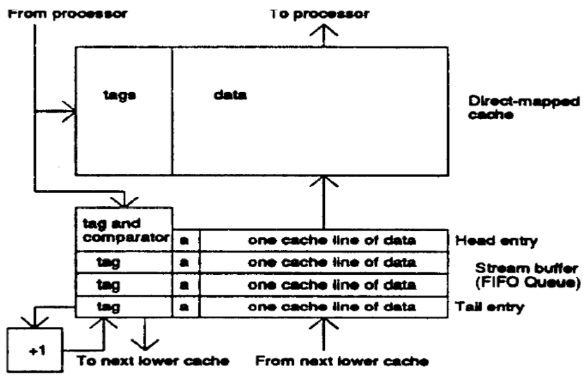
1) *REDUCE MISSES VIA LARGER BLOCK SIZE*



The performance curve is U-shaped because Small blocks have a higher miss rate and Large blocks have a higher miss penalty (even if miss rate is not too high).

2) *REDUCING MISSES BY HARDWARE PREFETCHING OF INSTRUCTIONS & DATA*

In this an extra block is placed in “stream buffer”. After a cache miss, the stream buffer initiates fetch for *next* block. But it is not allocated into cache to avoid “pollution”. On miss, it check the stream buffer in parallel with cache. It relies on having extra memory Bandwidth.



3) *REDUCING MISSES BY SOFTWARE PRE-FETCHING DATA :*

Data prefetching means loading the data into register. Cache pre-fetch means load into cache. Special prefetching instructions cannot cause faults. Pre-fetching comes in two flavours:

a) *Binding Pre-fetch* : Requests load directly into register.

b) *Non-Binding Pre-fetch*: Load into cache.

4) *Reducing Misses by Compiler Optimizations:*

This can be done by the following techniques:

a) Merging Arrays: Improve spatial locality by single array of compound elements vs. 2 arrays.
b) Permuting a multidimensional array : Improve spatial locality by matching array layout to traversal order.

c) Loop Interchange : change nesting of loops to access data in order stored in memory.
d) Loop Fusion: Combine 2 independent loops that have same looping and some variables overlap
e) Blocking: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows.

B) MISS PENALTY REDUCTION

1) Write Policy 1:Write-Through vs Write-Back

- In a write-through cache, every write to the cache causes a write to main memory.
- Alternatively, in a write-back or copy-back cache, writes are not immediately mirrored to the main memory. Instead, the cache tracks which locations have been written over .The data in these locations are written back to the main memory only when that data is evicted from the cache. For this reason, a read miss in a write-back cache may sometimes require two memory accesses to service: one to first write the dirty location to memory and then another to read the new location from memory.

2) Write Policy 2: Write-allocate vs Non-Allocate

- In write allocate, we allocate a new cache line. It means that we have to do a "read miss" to fill in rest of the cache line.
- In write non-allocate or write around, simply send write data through to underlying memory/cache. In this, we do not allocate new cache line.

3) REDUCING THE MISS PENALTY

a) READ PRIORITY OVER WRITE ON MISS

If a system has a write buffer, writes can be delayed to come after reads. The system must, however, be careful to check the write buffer to see if the value being read is about to be written. A simple method of dealing with this problem is stall reads until the write buffer is empty. However, this method increases the read miss penalty considerably since the write buffer in write-through is likely to have blocks waiting to be written.

b) NON BLOCKING CACHES TO REDUCE STALLS ON MISSES:

A **non-blocking** cache, in conjunction with out-of-order execution, can allow the CPU to continue executing instructions after a data cache miss. The cache continues to supply hits while processing read misses(hit under miss). The instruction needing the missed data waits for the data to arrive). Complex caches can even have multiple outstanding misses (miss under miss).

c) EARLY RESTART AND CRITICAL WORD FIRST

This strategy does not require extra hardware (like the previous two techniques). It optimizes the order in which the words of a block are fetched and when the desired word is delivered to the CPU.

a) Early Restart : With early restart, the CPU gets its data (and thus resumes execution) as soon as it arrives in the cache without waiting for the rest of the block.

b)Critical Word First : Instead of starting the fetch of a block with the first word, the cache can fetch the requested word first and then fetch the rest afterward. In conjunction with early restart , this reduces the miss penalty by allowing the CPU to continue execution while most of the block is still being fetched. Also called *wrapped fetch* and *requested word first.*

C) REDUCE THE TIME TO HIT IN THE CACHE

1) FAST HITS BY AVOIDING THE ADDRESS TRANSLATION

Sending the virtual address to cache is Virtually Addressed cache. Every time process is switched logically must flush the cache; otherwise get false hits. Cost is time to flush + “compulsory” misses from empty cache. It deals with aliases.

a)Solution to Aliases

- HW guarantees that every cache block has unique physical address
- SW guarantees that lower n bits must have same address; as long as covers index field & direct mapped, they must be unique; called *page coloring.*

b)Solution to cache flush

- Add *process identifier tag* that identifies process as well as address within process

2)FAST WRITES ON MISSES VIA SMALL SUBBLOCKS

If most writes are 1 word, subblock size is 1 word, & write through then always write subblock and tag immediately.

a)Tag match and valid bit already set :

Writing the block was proper and nothing lost by setting valid bit again.

b) Tag match and valid bit not set:

The tag match means that this is the proper block; writing the data into the subblock makes it appropriate to turn the valid bit on.

c) Tag mismatch:

This is a miss and will modify the data portion of the block. Since write-through cache, no harm was done; memory still has an up-to-date copy of the old value. Only the tag to the
address of the write and the valid bits of the other subblock need be changed because the valid bit for this subblock has already been set.

3) FAST HIT TIMES VIA PIPELINED WRITES

Pipeline tag check and update cache as separate stages; current write tag check & previous write cache update. If shade is “Delayed Write Buffer”; it must be checked on reads; either complete write or read from buffer.

REFERENCES

[1] Jouppi, N. P. (1990, May). Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Computer Architecture, 1990. Proceedings., 17th Annual International Symposium on* (pp. 364-373). IEEE.

[2] Lam, M. D., Rothberg, E. E., & Wolf, M. E. (1991). The cache performance and optimizations of blocked algorithms. *ACM SIGOPS Operating Systems Review*, *25*(Special Issue), 63-74.

[3] Jin, H., Frumkin, M., & Yan, J. (1999). *The OpenMP implementation of NAS parallel benchmarks and its performance* (Vol. 192). Technical Report NAS-99-011, NASA Ames Research Center.

[4] Zhang, X. Y., Zhang, Q., Zhang, Z., Song, G., & Zhu, W. (2004). A construction of locality-aware overlay network: mOverlay and its performance. *Selected Areas in Communications, IEEE Journal on*, *22*(1), 18-28.