2nd IEEE International Conference on Engineering and Technology (ICETECH), 17th & 18th March 2016, Coimbatore, TN, India.

An Overview of Modern Cache Memory and Performance Analysis of Replacement Policies

Swadhesh Kumar
Dept. of Computer Science and Engg.
MMM University of Technology
Gorakhpur, Uttar Pradesh
swadheshkumar@gmail.com

Dr. P K Singh
Dept. of Computer Science and Engg.
MMM University of Technology
Gorakhpur, Uttar Pradesh
topksingh@gmail.com

***Abstract*—Memory hierarchy in current generation computers is formed by keeping registers inside, cache on or outside the processor and virtual memory on Hard disk. The principle of locality of reference is used to make memory hierarchy work efficiently. In recent years various advances have been made to improve the cache memory performance on the basis of hit rate, latency, speed, replacement policies and energy consumption. Cache replacement policy is one of the important design parameter which affects the overall processor performance and also become more important with recent technological moves towards highly associative cache. This paper yields a survey of current generation processors on the basis of various factors effecting cache memory performance and throughput. The main focus of this paper is the study and performance analysis of the cache replacement policies on the basis of simulation on several benchmarks.**

Keywords—Miss rate, Hit rate, Cache Performance, Replacement Policy, LRU, LFU, FIFO, RANDOM

I. INTRODUCTION

Cache, a fast semiconductor memory is a place to store frequently used subset of data or instruction from relatively slower memory. It avoids having to go to main memory every time when this same information is required. The table given below shows the caching hierarchy [1]

The principal of locality of reference is used to get data or instructions of program. At one time the processor accesses a small portion of address space. Cache memory performance is calculated on the basis of miss rate, miss penalty, and average access time. Miss Rate is defined as the fraction of memory accesses that are not found in the cache while hit rate is the fraction of memory access found in the cache. Miss Penalty is defined as the total number of cycles CPU is stalled for a memory access determined by the sum of Cycles (time) to replace a block in the cache, upper level and Cycles (time) to deliver the block to the processor. Average Access Time and CPU execution time is calculated as:

Average Access Time = HT x HR + MP x MR

CPU Execution Time = (CCC + MSC) x CCT

MSC = Number of Misses x MP

= IC x (Misses / Instructions) x MP

= IC x [(Memory Access / Instructions)] x MR x MP

here, HT= Hit Time, HR= Hit Rate

MP= Miss Penalty, MR= Miss Rate,

CCC= CPU Clock Cycles, MSC= Memory Stall Cycles

CCT= clock cycle time

Table1: Caching Hierarchy

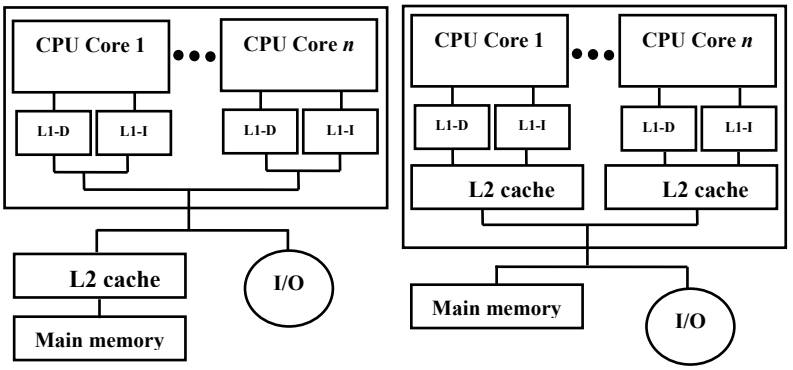
Cache Type	What Cached	Where Cached	Latency in cycles	Managed By
Registers	4-byte word	CPU registers	0	Compiler
TLB	Address translation	On-Chip TLB	0	Hardware
L1 cache	32-byte block	On-Chip L1	1	Hardware
L2 cache	32-byte block	On-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main Memory	100	Hardware + OS
Buffer Cache	Parts of files	Main Memory	100	OS
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

978-1-4673-9916-6/16/$31.00 ©2016 IEEE

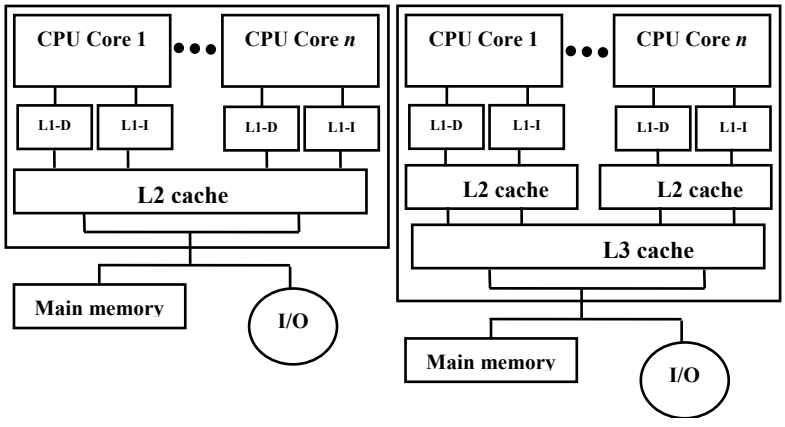
Number of cycles for memory read and memory write can be different similarly Miss penalty to read can be different from write.

Memory Stall Clock Cycles = (Memory read stall cycles + Memory write stall cycles)

The memory system in uniprocessor design was a simple component, made up of few levels of cache memory, which supply data and instruction to single processor. But with multicore processors, caches are just a component of memory system, here the other parts include consistency model, cache coherence support and intra chip interconnection. In multicore processors each core of a processor consists of all the components of an individual processor such that registers, ALU, pipeline hardware, control unit and L1 instruction and data caches. In addition to multiple numbers of cores, multicore chips also include L2 and sometimes L3 cache also [2]. There are mainly three variables in multicore organization are the number core on the chip, levels of cache memory and amount of the shared cache memory.



(a) Dedicated L1 cache (b) Dedicated L2 cache



(c) Shared L2 cache (d) Shared L3 cache

Here, the figure (a) is the organization with dedicated L1 data and instruction cache found in early multicore chips and is still seen in embedded chips. The example of this multicore organization is ARM11 MPCore. The figure (b) is the organization with dedicated L2 cache is one with no on chip cache sharing. The example of this multicore organization is AMD Opteron. The figure (c) is the organization with shared L2 cache is used in the Intel Core Duo. The figure (d) is the organization with shared L3 cache is used in the Intel Core i7 [3].

II. STRATEGY OF CACHE DESIGN

The three main units of a processor are instruction, execution, and storage unit. The instruction unit is responsible for organizing instructions of a program to be fetched and decode. The execution unit perform logical, arithmetic operations and execute instructions. The storage unit establishes interface through a temporary storage between other two units. The essential components of storage unit are cache memory, translator, and Translation Look-aside Buffer (TLB). Address Space Identifier Table (ASIT), a Buffer Invalidation Address Stack (BIAS) and write through buffers may also be available in storage unit. Technology has made it possible to fabricate millions transistors on a single chip because of which a small portion is needed to make a powerful processor. To minimize inter-chip data transfers, on-chip memory is placed inside the processor. Table1 shows cache design strategy and specifications of a various recent processors launched by Intel and AMD. There are various mapping techniques for determining cache organization. A mapping technique is used to map large number of main memory blocks into few lines of the cache memory and tag bits within every cache line examine which block of main memory is currently available in a particular cache line. Out of three mapping approaches i.e. direct mapping, associative mapping and set associative mapping, set associative caches are considered best because of better hit rate and less access time. But beyond a certain limit, increasing cache size has more of an impact than increasing associativity. Replacement algorithm plays an important role in the design of cache memory because it makes decision to select a particular line of cache memory is to be replaced with the desired main memory block. First in First Out (FIFO), Least Frequently Used (LFU) and Least Recently Used (LRU) are algorithms used for making such decision. Least Recently Used (LRU) is the most effective algorithm because it is easy to implement, according to this more recently used words are likely to be referenced again. Write Caching, Write Back and Write Through are three main caching policies that decide how consistency is maintained between cache lines and corresponding main memory blocks. In write back policy write operations are made to cache only, the main memory is updated only when the corresponding cache line is flushed from cache memory. In write through policy write operations are performed to main memory along with the cache memory. The write back policy can result in inconsistency if two caches hold same line, and line is updated in one cache, then the other cache will unknowingly holds an invalid value. Inconsistency can also occur with the write-through policy, unless the other caches monitor to memory traffic or get direct notification of update. So the considerable traffic is generated in both write through and write back policy, a single bit error of any of these cannot be tolerated unless Error Correcting Code (ECC) is provided.

978-1-4673-9916-6/16/$31.00 ©2016 IEEE

Write caching is a mix of write through caching and write back caching policy where a small full associative cache is placed behind the write through cache.

TABLE 2: SPECIFICATIONS OF AMD AND INTEL PROCESSORS

Processor	Clock Speed in MHZ	Bus Speed	Clock Multiplier	Micro-Architecture	Data Width in bits	Cores	Threads	L1 Cache in KB	L2 Cache In K/MB	L3 Cache in MB	Mapping, Replacement Algo & Write Policy	No. of Cache	Thermal Design Power in Watts
AMD Processors													
Opteron 6180	2500	3200 MHZ	X	K 10	64	12	12	I: 12X64 D:12X64	12X512	2X6	NPA	3	140
Phenom X4 9950	2600	533* MHZ	13	K 10	64	4	4	I: 4X64 D: 4X64	4X512	2 shared	NPA	3	140
Athelon x4 760	3800	4100 MHZ	X	Pile-driver	64	4	4	I:2X64 D: 2X16	2X2MB	Nil	NPA	2	100
A10 6800K	4100	4400 MHZ	X	Pile-driver	64	4	4	I:2X64 D: 4X16	2X2MB	Nil	NPA	2	100
*Legends: I-Instruction Cache, D-Data Cache; *: 533 MHz, Memory controller; One 2000 MHz 16-bit Hyper Transport link.*													
Intel Processors													
Core i3 2100T	2500	5 GT/s DMI	25	Sandy Bridge	64	2	4	I: 2X32 D: 2X32	2x256	3 shared	NPA	3	35
Core i5 760	2800	2.5 GT/s DMI	21	Nehalem	64	4	4	I: 4X32 D: 4X32	4x256	8 shared	NPA	3	95
Core i7 875K	2933	2.5 GT/s DMI	22	Nehalem	64	4	8	I: 4X32 D: 4X32	4x256	8 shared	NPA	3	95
Core i7 990x*	3467	6.4 GT/s DMI	26	Westmere#	64	6	12	I: 6X32 D: 6X32	6x256	12 shared	NPA	3	130
*Legends: I-Instruction Cache, D-Data Cache; #: Nehalem (Westmere), *: Extreme Edition, NPA: Not Publically Available.*													

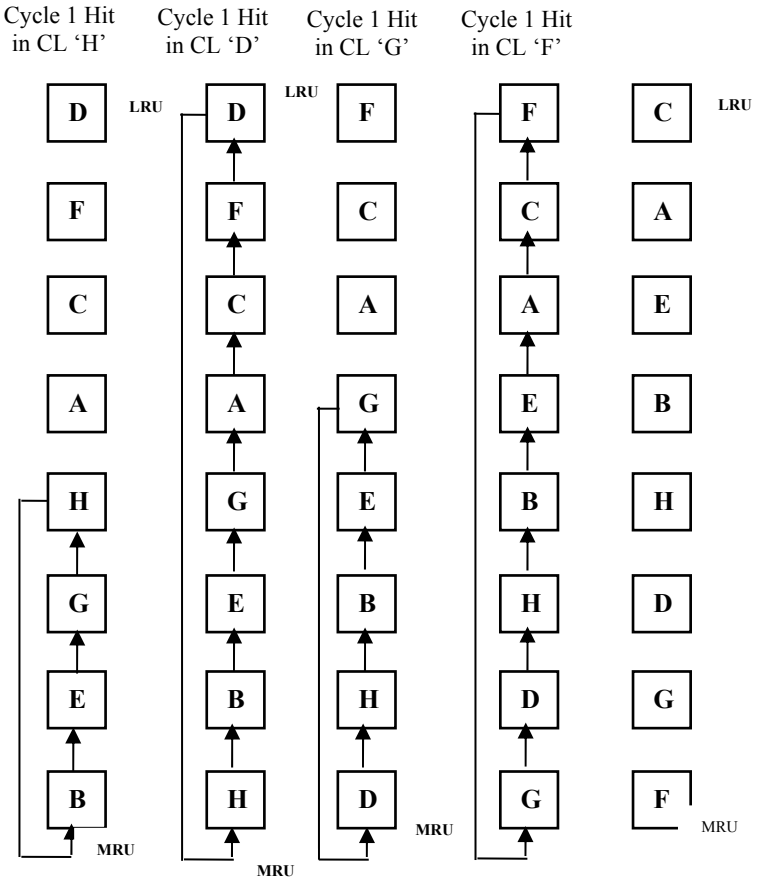
Mostly, Level 1 cache consists of Level 1 data cache and Level 1 instruction cache. When the first on chip cache made an appearance then some designs consisted of a single cache to store references to both instructions and data. Recently, it has become common to split cache memory into two parts one for instructions and the other for data. When the processor attempts to fetch an instruction from main memory then first it consults with the L1 instruction cache similarly, when the processor attempts to fetch data from main memory then first it consults with the L1 data cache. The main advantage of a unified cache is higher hit rate than split caches because it balances load between data and instruction fetches automatically. In unified cache only one cache design and implementation needed. On the other hand the split cache removes problem of contention between fetch/decode and execution unit. This contention can reduce performance by interfering with instruction pipeline. But, many implementation attribute of processors like replacement algorithm, mapping function and write policies are not publicly available. Intel x86 processors and AMD processors employ a direct-mapped Level 1 cache, and Level 2 cache between 2 to 4 way set associative. The L3 and higher level caches could be between 16-way to 64-way set associative. Most of them use LRU (least recently used replacement policy), and a write-back cache.

III-CACHE REPLACEMENT POLICIES

Current generation processors include multiple levels of cache memory and the high associativity [4] has made it important to re-check the effectiveness of various cache replacement policies. In cache memory, when all the lines in a set of cache become full and a new block from main-memory needs to be placed in cache, then the cache controller has to be discard a line from cache set and replace it with the new block from main-memory. The modern processors employ cache replacement policies such as LRU (Least Recently Used)[5], Random[6], FIFO(First in First Out)[4], LFU(Least Frequently Used). In all these policies, except Random, determine which block of cache memory to replace by looking only at the past references. Least Recently Used replacement needs a number of status bits to maintain record of each cache block accessed. As the set-associativity increases, the number of these bits also increases. Random replacement policy can be used to reduce the complexity and cost of LRU replacement policy but at the expense of performance. Recent studies describe cache design space with relatively finite associativity, and consider only true Least Recently Used replacement policy [7].

The Least Recently Used replacement policy uses access pattern of a program memory to predict that most recently accessed cache line will most likely to be accessed again, and the cache line which has been Least Recently Used (a block in the set that is in cache for the longest and having no reference to it) will be replaced by cache controller. The LRU stack is maintained as-

978-1-4673-9916-6/16/$31.00 ©2016 IEEE



Although the Least Recently Used policy is efficient, it does need a number of bits to maintain a record for each block, contains details such that when a block is accessed before. In LRU policy, each time when a cache hit or miss occurs then the block shifting in LRU stack requires more time and power. Random cache replacement policy can be used to reduce the complexity and cost of LRU replacement policy. Random replacement policy chooses a candidate block to be discarded randomly from all the cache lines in the set. This policy does not need to keep any information of access history. It has been used in ARM processors for its simple design. In LFU cache replacement policy a counter is assigned to each cache block, which loaded in cache memory. For each reference of block the counter associated is incremented by one. When the cache is full and has a new block to be inserted, then the block with the lowest counter is removed.

III- EVALUATION METHODOLOGY

We have used SMP3.0, a trace driven simulator for the performance analysis of major cache replacement policies [8]. Trace driven simulator is a cost effective technique of performance evaluation of computer system design, specially for cache design, TLB, and paging system. In this paper, we have used some SPEC92 Benchmarks such as: HYDRO, NASA7, CEXP, MDLJD, EAR, COMP, WAVE, SWM and UPCOMP for the performance analysis of replacement policies.

IV- SIMULATION SETUP

Number of Processors = 1
Cache Coherence Protocol = MESI
Bus Arbitration = LFU
Word Wide (bits) = 16
Blocks in Main Memory = 8192
Block size = 32 bytes
Main Memory size = 256 K Bytes
Blocks in Cache = 128
Cache size = 4KB
Mapping = 8 way- set associative
Writing Strategy = Write Back
Replacement Policies = RANDOM, FIFO, LFU, LRU

TABLE 3: EXPERIMENTAL RESULTS

BENCHMARK	REPLACEMENT POLICIES			
	RANDOM	FIFO	LFU	LRU
WAVE	88.124	89.116	89.408	89.262
CEXP	99.545	99.545	99.545	99.545
COMP	90.214	90.61	90.65	90.65
EAR	95.554	96.157	96.157	96.157
HYDRO	90.55	91.067	91.067	91.067
MDLJD	92.61	92.915	93.23	93.27
NASA 7	90.296	90.674	90.674	90.674
SWM	90.415	89.855	90.65	90.34
UCOMP	90.377	90.926	90.962	90.962

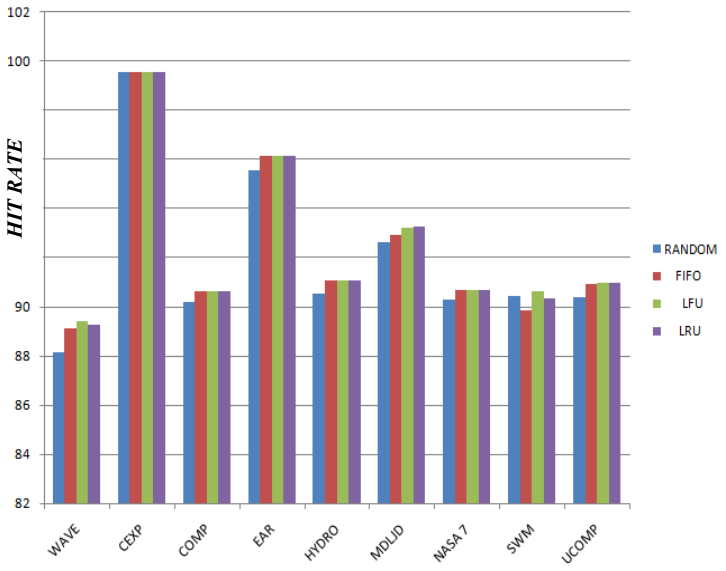


Table 3 shows the experimental results of various replacement policies in the form of Hit Rate on various Benchmarks. We found that LRU Replacement Policy has better performance than others across almost all the Benchmarks used for

analysis, but there are some exceptions also as: for COMP Benchmark LFU and RANDOM Policies are slightly better than LRU. In general, the LRU replacement policy has less number of cache misses than the LFU, RANDOM and FIFO for most of the benchmarks.

V- CONCLUSION

In this paper, we have studied some of the recent advances made in cache design to improve access time of memory management unit, latency and energy consumption. This paper describes the cache replacement policies in the form of their performance analysis on various Benchmarks. The analysis of experimental results shows that LRU is the most scalable cache replacement policy. From this survey, we found that speed of processors growing continuously than decrease in memory latency, so eliminating cache misses is much important for the overall performance of the Processor. Since caches today becoming more set associative, so cache replacement policies will gain more significance.

References

[1]. C. Kozyrakis, "Advanced Caching Techniques." 2008.

[2]. Swadhesh Kumr, Dr. PK Singh, "An Overview of Hardware Based Cache Optimization Techniques," International Journal of Advance Research in Science and Engg, vol. no. 4, Special Issue (01), September 2015

[3]. Gavrichenkov, "First Look at Nehalem Microarchitecture", November2008,http://www.xbitlabs.com/articles/cpu/display/nehalem microarchitecture.html.

[4]. Intel Xscale-Core ,Developer's Manual,December 2000, http://developer.intel.com

[5]. Ackland B., Anesko D., Brinthaupt D., Daubert S.J, Kalavade A.., Knoblock J., Micca E., Moturi M., Nicol C.J., O'Neill J.H., Othmer J., Sackinger E., Singh K. J.,Sweet J., Terman C. J., and Williams J., "A Single-Chip,1.6 Billion, 16-b MAC/s Multiprocessor DSP," IEEE Journal of Solid-state circuits, Vol. 35, No. 3, March 2000, pp. 412-423.

[6]. David A. Patterson, John L. Hennessy. "Computer organization and design: the hardware/software interface". 2009.

[7]. Cantin J. F, Hill M. D., Cache Performance of the SPEC CPU2000Benchmarks,http://www.cs.wisc.edu/multifacet/misc/spec2000 cachedata

[8]. Miguel A. VegaRodríguez, Juan M. SánchezPérez,Juan A. GómezPulido "An Educational Tool for Testing Caches on Symmetric Multiprocessors".Microprocessors and Microsystems, Elsevier Science, vol. 25, no. 4, pp. 187194. June 2001. ISSN 01419331.

978-1-4673-9916-6/16/$31.00 ©2016 IEEE