Enhancing Cache Performance Based on Improved Average Access Time

Jasim. A. Ghaeb

***Abstract*—**A high performance computer includes a fast processor and millions bytes of memory. During the data processing, huge amount of information are shuffled between the memory and processor. Because of its small size and its effectiveness speed, cache has become a common feature of high performance computers. Enhancing cache performance proved to be essential in the speed up of cache-based computers. Most enhancement approaches can be classified as either software based or hardware controlled. The performance of the cache is quantified in terms of hit ratio or miss ratio. In this paper, we are optimizing the cache performance based on enhancing the cache hit ratio. The optimum cache performance is obtained by focusing on the cache hardware modification in the way to make a quick rejection to the missed line's tags from the hit-or miss comparison stage, and thus a low hit time for the wanted line in the cache is achieved. In the proposed technique which we called Even- Odd Tabulation (EOT), the cache lines come from the main memory into cache are classified in two types; even line's tags and odd line's tags depending on their Least Significant Bit (LSB). This division is exploited by EOT technique to reject the miss match line's tags in very low time compared to the time spent by the main comparator in the cache, giving an optimum hitting time for the wanted cache line. The high performance of EOT technique against the familiar mapping technique FAM is shown in the simulated results.

***Keywords*—**Caches, Cache performance, Hit time, Cache hit ratio, Cache mapping, Cache memory.

Symbols

C	Number of characters per a line's tag/or memory field's tag
EOT	Even- Odd Tabulation
FAM	Fully Associative Mapping
DM	Direct Mapping
CPU	Central Processing Unit
D	Number of lines in the cache
v	Number of sets in the cache
a	Number of lines per set
LSB	Least Significant Bit
t	Number of bits per tag
w	Word field bits

I. Introduction

SINCE the early development of computer systems, there has been a growing need for faster and more powerful computer systems. This motivated the researchers in the areas of hardware and software development of computer systems. In order to alleviate the impact of the growing gap between CPU speed and main memory performance, today's computer architectures implement hierarchical memory structures [1].

Jasim. A. Ghaeb is with the Hashemite University, Department of Electrical Engineering, P.O. Box 150459, Zarqa 13115, Jordan. Email:gaebja@hu.edu.jo

The idea behind this approach is to hide both the low main memory bandwidth and the latency of main memory accesses as well as to provide a large amount of real memory at an economical price. Although, there was a speed enhancement for both CPU and main memory, the speed gap between them has widened.

Moving further away from the CPU, the layers of memory successively become larger and slower. The memory components which are located between the processor core and main memory are called cache memories or caches. They are intended to contain copies of main memory blocks to speed up accesses to frequently needed data. The next lower level of the memory hierarchy is the main memory which is large but comparatively slow. The external memory such as hard disk drives or remote memory components in a distributed computing environment represent the lower end of any common hierarchical memory design [2], [3].

The good overall performance of a computer system cannot be achieved without good cache performance. Based on this change on computer architecture, techniques have been designed to minimize instruction count to improve CPU performance may not achieve a good performance unless take into consideration cache performance [4].

In this paper, we explore the potential performance gains that cache conscious design offers in understanding and improving the performance. We develop a novel technique which we called Even- Odd Tabulation (EOT) to enhance the cache performance in terms of reducing the hit time. The cache line's tags are tabulated into two groups: even line's tags and odd line's tags. Depending on the line number that is looking for, the cache line's tags of opposite LSB are undesired tags and rejected a way directly before going to the complete and long C-characters comparison. By this approach, the line's tag of a missed match LSB does not pass to the C-character comparison stage and there is no waste of time. Thus the desired cache line is located quickly, leading to maximum hit ratio.

This paper is structured as follows. In Section II, we will introduce some fundamental cache characteristics, including a brief discussion of conventional elementary cache optimization techniques. Section III presents our proposed technique to improve the cache hit time. In Section IV shows the simulation results and explains the performance of the proposed technique against the conventional methods. Finally, Section V concludes the paper work.

II. Cache Design

Typically, a memory hierarchy contains a rather small number of registers on the chip which are accessible without delay. Furthermore, a small cache, usually called level one (L1) cache, is placed on the chip to ensure low latency and

high bandwidth. The L1 cache is often split into two separate parts; one keeps data, the other instructions. The latency of on-chip caches is commonly one or two cycles. The L1 caches are usually backed up by a level two (L2) cache. Currently, L2 cache memories are typically located on-chip as well; e.g., in the case of Intel's Itanium CPU. The off-chip caches are much bigger and provide data with lower bandwidth and higher access latency [5], [6]. The cache design has a direct effect on the cost and performance of the computer system. The various designs explored in this section have different addressing mechanisms in terms of cost and speed but have one common feature that every cache is divided into d-lines each containing 2^W-words. The address generated by the CPU is formed of two components, a line index *b* and a word offset *f* within the line. When such an address is generated, the cache is first checked for the presence of the requested line *b*, if found, then the offset *f* is used to fetch the needed data. Otherwise the specified line *b* must be fetched from the lower level of memory. All cache approaches are based on selecting where to store a particular block and how to locate it for a memory request. Hence, cache organizations are classified according to the various ways the memory blocks are assigned to the cache and thus such an assignment usually adhere to a particular mapping technique [7]-[10]. In the following subsections, we will survey a number of conventional cache designs. Each design will have a different mapping strategy that will map the memory blocks to the cache lines. In order to highlight the performance of the proposed EOT technique, a comparison is made with the conventional Fully Associative Mapping Technique (FAM).

A. Aspects of Cache Architectures

Data within the cache are stored in cache lines. A cache line holds the contents of a contiguous block of main memory. If data requested by the processor are found in a cache line, it is called a cache hit. Otherwise, a cache miss occurs. The contents of the memory block containing the requested word are fetched from a lower memory layer and copied into a cache line. For this purpose, another data item must typically be replaced. Therefore, in order to guarantee low access latency, the question into which cache line the data should be loaded and how to retrieve them must be handled efficiently. In the next subsections, we will brief the well-known techniques and introduce the privileges and drawbacks of each.

1. Direct Mapping (DM)

In respect of hardware complexity, the cheapest approach to implement block placement is direct mapping [11]- [13]; its function maps any block in the main memory into only one possible cache line. The cache line is marked by a tag value to distinguish a particular block of data from the other blocks that can locate in that line.

Direct mapped caches are fast, simple and inexpensive to implement. Moreover, direct mapped cache has been among the most popular cache architectures in the past and is still very common for off-chip caches. However, its main disadvantage is the frequent occurrence of conflict. Conflict occurs while executing a program task that requires several variables that resides in two or more blocks that map to the same cache line.

2. Fully Associative Mapping (FAM)

In contrast to the simple mapping for a direct mapping cache, an associative cache allows any of the memory blocks to be mapped to any of the cache lines [8], [14]. Such flexibility allows for better utilization of the cache without conflict. In this design, the line numbers of the current cache contents are placed in an associative cache directory. The block is searched for in the cache directory. If the CPU finds its match, then the data is in the cache and the corresponding block is accessed. A mismatch results in a cache miss, and a fetch is issued to main memory. Since the block could be placed in any cache line, one of the resident blocks is overwritten. The choice of replacement block may cause future cache misses.

Direct mapped and fully associative caches can be seen as special cases of k-way set-associative caches of *v*-sets and *d*-lines per a set. At $v = d$, a direct mapped cache is a one-way set-associative cache, whereas a fully associative cache is *d*-way one set-associative. In a fully associative cache and in a *d*-way set-associative cache, a memory block can be placed into several alternative cache lines.

Computer architects have recently focused on increasing the set associatively of on-chip caches. A k-way set-associative cache is characterized by a higher hardware complexity, but usually implies higher hit rates [14]- [16]. The d-cache lines of k-way set-associative cache are grouped into *v*-sets. The contents of any memory block can be placed into any cache line of the corresponding set. This design is a compromised design that avoids the problem of conflicts and the dependency of the cycle time on the cache size. For a given CPU address, the address is obtained as in the direct mapping cache: *B Mod L* where *L* is the number of sets. Once the required set is determined, the desired block *B* is associatively searched for among the members of the set. When a cache miss occurs, replacement decisions take into account only members of the set where the miss occurred, not the whole cache. Conflict will rarely occur since two blocks accessed at the same time from the same correspondence set may reside in different blocks of a set. Thus, the set associative mapping cache organization has most of the speed advantage of the direct mapping cache and much of the flexibility of the full associative cache, both at a moderate cost. Due to its cost-performance edge, set associative cache design has been selected by many computer manufactures when implementing a cache for their computer systems.

3. Sector Mapping Cache (SMC)

In sector mapping, the main memory and the cache are both divided into sectors; each sector is composed of a number of blocks [17]. Any sector in the main memory can map into any sector in the cache and a tag is stored with each sector in the cache to identify the main memory sector address. However, a complete sector is not transferred to the cache or back to the main memory as one unit. Instead, individual blocks are transferred as required. On cache sector miss, the required

block of the sector is transferred into a specific location within one sector.

Sector mapping might be regarded as a fully associative mapping scheme with valid bits, as in some microprocessor caches. Each block in the fully associative mapped cache corresponds to a sector, and each byte corresponds to a sector block.

B.Cache Performance Model

Typical design and optimization techniques attempt to reduce the number of instructions that are executed leading to a high speed computer system. Many attempts had been carried out to improve the cache memory performance. Serhan and Abdel-Hag modified the set associative mapping in a way to increase the set size virtually by allowing interleaving processes to make use of empty lines in cache and not overwrite the cache lines by each other [9]. Spjuth proposed the elbow caching to improve the skewed-associative caching, which depends on the efficiency of data movement between alternate positions in the cache [18].

The performance of a cache can be quantified in terms of the hit and miss rates, the cost of a hit, and the miss penalty, where a cache hit is a memory access that finds data in the cache and a cache miss is one that does not.

For cache reading, the cost of a cache hit is roughly the time to access an entry in the cache. The miss penalty is the additional cost of replacing a cache line with one containing the desired data. Due to the principle of locality, there are a number of accesses to items in the block that is brought into cache, leading to faster overall access time. The fraction of the total number of blocks that are missed in the cache and need to access main memory is the miss ratio. Higher hit-rates provide a high cache performance. Designers use average memory access time as a way to measure cache performance. It is the average time to access memory considering both hits and misses and the frequency of different access which affects the performance. For two memory levels, the average access time (Tav) is determined in terms of cache hit and cache miss ratios and access times for cache and main memory [6]. It is given by:

$$Tav = (\text{Cache hit ratio}) \times (\text{Hit time}) + (\text{Cache miss ratio}) \times (\text{Miss Penalty}) \quad (1)$$

Since the speeds of the actual memory used will be improved independently, most effort in cache design is spent on fast control and decreasing the miss rates. We can classify misses into three categories, compulsory misses, capacity misses and conflict misses. Compulsory misses are when data is loaded into the cache for the first time (e.g. program startup) and are unavoidable. Capacity misses are when data is reloaded because the cache is not large enough to hold all the data no matter how we organize the data. All other misses are conflict misses which will occur because of a line may be discarded and later retrieved if too many lines map to its set in the case of direct mapped or set-associative cache. These misses are also called collision or interference misses.

III. Proposed Methodology

In this paper, we investigate the benefits of optimizing cache performance and focusing exclusively on enhancing the cache hit ratio; since the cache performance can be classified in terms of cache hit and cache miss ratios. The cache control comparator and its technique play a crucial role in cache hit time and thus in improving the cache performance. In our proposed EOT technique, the line's tags are assumed in two groups; even line's tags and odd line's tags depending on their Least Significant Bit (LSB). Each line's tag is C-characters size and each character is 4-bits. Before feeding the line's tag of C-characters to the main cache comparator for a complete C-character comparison, the line's tag is passed through the even-or odd comparator for filtering. By this even-or odd comparator, all the cache's tags of opposite LSB to the memory address's tag that is looking for will be rejected directly and quickly before entering the complete and long C-character comparison stage. Therefore, a lot of miss match line's tags are discarded quickly before going to the main cache comparator, leading to an optimum hit time for the wanted cache line.

It is not necessary for all the cache line's tags fetched from the main memory to be divided equally between the even and odd values. Thus depending on the current values of the line's tags in the cache, the performance of the EOT technique can be categorized in four cases:

i. Case- one: *Equal hit ratio*

If the current values of the line's tags in the cache come equally between the even and odd values, then the hit ratio will be the same for locating an even line's tag or odd line's tag.

ii. Case- two: *High hit ratio*

If the current line's tags of opposite LSB to the one that is seeking for in the cache are coming more, the hit time is low and a high hit ratio is obtained. This is due to the high speed rejection of the miss match- LSB line's tags by the even-or odd comparator.

iii. Case- three: *Worse hit ratio*

This is the rarely case. There is no improvement in the hit ratio if the current values of the line's tags in the cache are all coming in even or odd order and the cache line that is looking for in even or odd order, respectively.

iv.Case- four: *Uncounted hit ratio*

If the current values of the line's tags come randomly in the cache, the line hit time is unknown and depends on the random distribution of the tags values in the cache.

A.Even- Odd Tabulation Technique (EOT)

For mapping the main memory blocks into cache lines, the memory address word is divided into two fields; the tag field of t-bits and word field of w-bits. The tag value of the memory address that is looking for needs to be compared with all line's tags in the cache to grasp the desired cache line for a word access. In EOT technique, before feeding the line's tags to the main cache comparator of a counted time comparison, a discriminated stage is added for even tags and odd tags. This stage is implemented by adding the even-or odd comparator as shown in Fig.1.

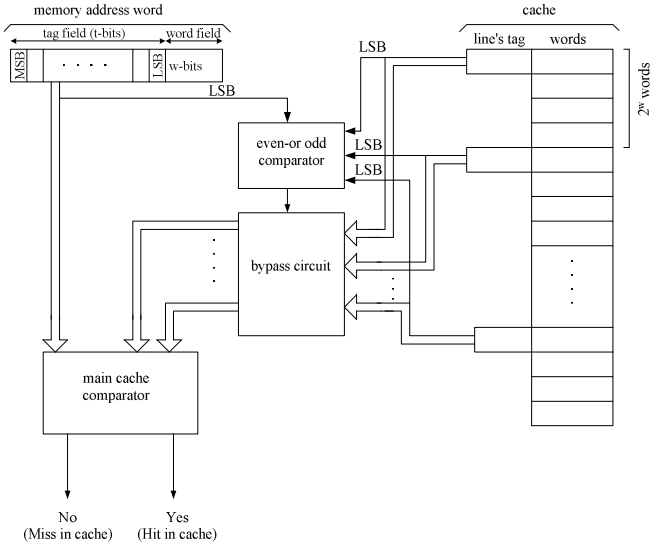

Fig. 1 Even- Odd Tabulation Technique

B. Pseudo code for EOT Technique

A java code is written to implement the EOT technique and determine the cache line hit time at different memory sizes. The Pseudo code is given in Appendix (A). It includes the following steps:

1. Step one

Take the tag value of the memory address of C-characters that is looking for and the cache line's tag of C-characters and feed them to the even-or odd comparator.

2. Step two

Compare the LSBs of both; memory address's tag and cache line's tag. If there is a mismatch between them, the cache line's tag is rejected directly. Otherwise, the line's tag is pushed forward to the main cache comparator.

3. Step three

Compare the passed- line's tag with the memory address's tag at the main cache comparator for a complete C-characters comparison.

4. Step four

Continue on comparison process until reaching the cache line's tag of C-matches. It is the wanted one, so break the comparison.

IV. Results and Discussions

The performance of the proposed technique EOT is determined and compared with the well known technique FAM. The simulation results are based on cache hit time and investigated for different sizes of cache and main memory. Fig.2 shows the cache hit time for the EOT technique against that of FAM technique. The cache is 16k line size and their lines are taken randomly from a main memory of 4M byte. The performance of EOT and FAM are simulated for 20 line's tags generated randomly from the cache. The high performance of EOT technique for cache hit time is arisen strongly. It is at average improvement of 36.57% compared to FAM technique. Fig.3 shows the performance of EOT against that of FAM technique, for 20 line's tags selected randomly from a 32k line cache. The cache lines are also taken randomly from a main memory of 16M byte. The average improvement in cache hit time by EOT compared to FAM is 40.57%.

In Fig.4, the current values of the line's tags are brought and forced equally between even and odd values for 16k line cache. Due to the direct and quick rejection of the half line's tags by the even-or odd comparator, the EOT technique has shown a high performance for cache hit time. Its average improvement compared to FAM is 45.45%.

Fig. 5 shows the worst case at where the current values of the line's tags in the cache are all forced in even order or in odd order and the cache line that is looking for is in even order or in odd order respectively. In practice, it is a rarely case. The even- odd comparator in EOT technique becomes without avail and it caused a regression of 4.16%.

The results summarizes that a powerful performance for a cache hit time is obtained with the proposed technique EOT compared to the familiar FAM technique.

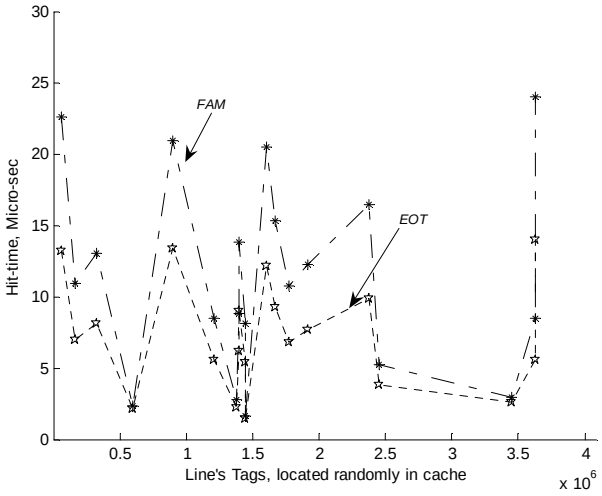

Fig. 2 Hit time in a 16k-line cache

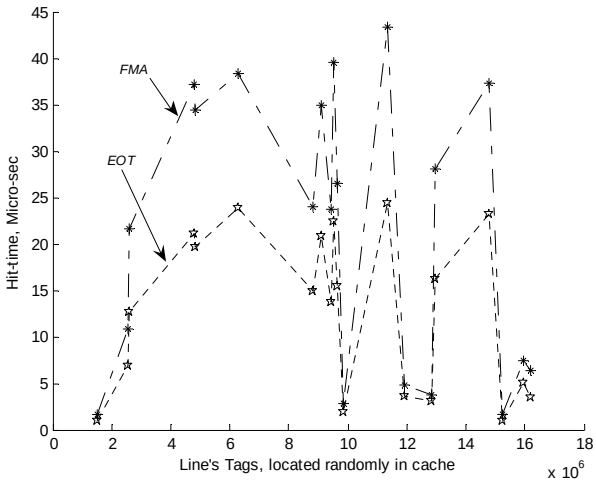

Fig.3 Hit time in a 32k-line cache

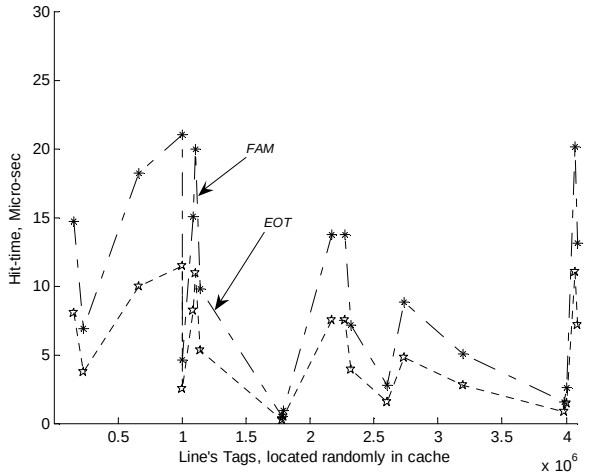

Fig.4 Hit time in 16k-line cache for equally even and odd line's tags

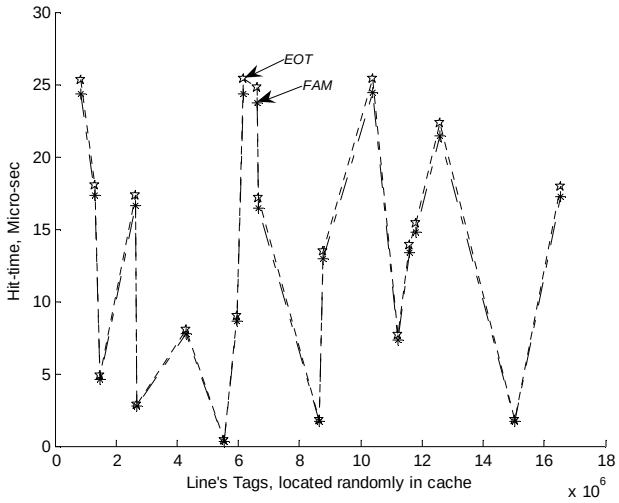

Fig.5 Hit time in a 16k-line cache for forced even order or odd order line's tags

V. Conclusions

Cache performance optimization yields to significant execution speedups. While some techniques are based on optimizing compilers, others are based on hardware. Future computer architecture trends further motivate research efforts focusing on memory hierarchy optimizations. In this paper, we presented optimum hardware cache architecture to enhance the performance of the cache based on high hit ratio. Because there are fewer cache lines than the main memory blocks, the cache line's tags can be come onto unequally two groups of tags; even line's tags and odd line's tags. So the proposed EOT approach exploited the LSB of the tag field in the main memory address to distinguish between the match tags and miss match tags in the cache. The even-or odd comparator compares only the LSBs of the cache line's tag and the memory field tag and rejects any miss match cache line in very low time. Consequently, only the matched cache line's tags are passed to the main cache comparator for a long and complete C-character comparison. In this way a lot of miss match line's tags are grasped and rejected quickly in the even-or odd comparator and they do not need to go to the main cache comparator. Building on that, there is no waste of time and a minimum time for cache line hitting is reached.

The performance of the proposed EOT technique is simulated and compared to the well known FAM technique, for different cache sizes; 16k line and 32k line. The results have shown that the new EOT technique has achieved a high performance for a line hit time in the cache compared to the familiar FAM technique.

Appendix (A): Pseudo code for EOT technique

Find a cache-line's tag using EOT technique;
Initialize time of one bit comparison = TB μs;
Initialize time of one hex-digit comparison = TH μs;
Initialize No. of Characters per a tag = C;
Initialize the "Bit-Counter" = zero;
Initialize the "Character-Counter" = zero;
for each line's tag on cache *do*
    Initialize the "Comparator-Counter" = zero;
    Increment "Bit-Counter";
    if line's tag and address's tag are matched for LSB *then*
        for each hex-digit on line's tag *do*
            Increment "Character- Counter";
            if hex-digits of both Line's tag and address's tag are matched *then*
                Increment "Comparator-Counter";
            endif;
        endfor;
        if Comparator- Counter equal C *then*
            Cache line' tag is located
            Time = (Bit- Counter × TB) + (Character- Counter ×TH);
            Break;
        endif;
    endif;
endfor;

References

[1] U. Meyer, p. Sanders and j. Sibeyn, algorithms for memory hierarchies: advanced lectures, springer-verlag, berlin, heidelberg 2003.

[2] M. Kowarschik and C. Weiß, "An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms," Lecture Notes in Computer Science Vol. 2625, pp. 213-232, Springer, 2003.

[3] S. Mamagkakis, D. Atienza , C. Poucet , F. Catthoor , D. Soudris and J. M. Mendias, "Custom Design of Multi-Level Dynamic Memory Management Subsystem for Embedded Systems," Proceedings of the IEEE Workshop on Signal Processing Systems (SIPS'04), vol. 1, No. 1, 2004, pp. 170-175, New York: IEEE Press, 2004.

[4] C. Baloukas, et. al, "Optimization Methodology of Dynamic Data Structures Based on Genetic Algorithms for Multimedia Embedded Systems," Elsevier, the journal of systems and software, Vol. 82, pp.590–602, 2009.

[5] H. Eichenbaum, "Memory Systems," Wiley Interdisciplinary Reviews: Cognitive Science, 1: 478–490. doi: 10.1002/wcs.49, 2010.

[6] M. D. Hill and A. J. Smith, "Evaluating Associatively in CPU Caches," IEEE Transactions on Computers, Vol. 38(12), pp. 1612-1630, 1989.

[7] T. M. Chilimbi, M. D. Hill and James R. Larus, “Cache-Conscious Structure Layout,” Proceedings of the ACM SIGPLAN Conference on programming language design and implementation, ISBN:1-58113-094-5, PLDI ,1999.
[8] W. Stallings, Computer Organization and architecture, 7th Edition, Prentice Hall, 2006.
[9] S. I. Serhan and H. M. Abdel-Haq, “Improving Cache Memory Utilization,” World academy of science, engineering and technology, Vol. 26, pp. 299-304, 2007.
[10] N. P. Topham and A. GonZalez, “Randomized Cache Placement for Eliminating Conflicts,” IEEE Transactions on Computers, Vol. 48, No.2, pp. 185-192, 1999.
[11] J. H. Bae and C. M. Kyung, “A Supplementary Scheme for Reducing Cache Access Time,” IEICE Trans. on inf. and systems, Vol. E79-d, No. 4, pp. 385-389, 1996.
[12] N. P. Jouppi, Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers, IEEE Proceeding, 17th Annual International Symposium on Computer Architecture, Seattle, WA, USA, pp. 364-373, 28-31 May 1990.
[13] S. J. E Wilton and N. P. Jouppi, “CACTI: An Enhanced Cache Access and Cycle Time Model,” IEEE Transactions on solid state circuits, Vol. 31, No. 5, pp. 677-688, 1996.
[14] P.F. Lin, “A 0.8-V 128Kb Four-Way Set-Associative Two-Level CMOS Cache Memory Using Two-Stage Wordline/Bitline-oriented Tag-Compare (WLOTC/BLOTC) Scheme,” IEEE Journal of Solid-State Circuits Vol. 37, No. 10, pp. 1307-1311, 2002.
[15] Ruud Van DerPas “Memory Hierarchy in Cache Based System," Sun Microsystems, Inc, part .No. 817-0742-10, 2002.
[16] P. Palsodkar, A. Deshmukh, P. Bajaj and A. G. Keskar, An Approach for Four Way Set Associative Multilevel CMOS Cache Memory, Lecture Notes in Computer Science, Vol. 4692, pp. 740-746, DOI: 10.1007/978-3-540-74819-9_91, 2007.
[17] J.B. Rothman and A.J. Smith, Sector cache design and performance, IEEE Proceedings. 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000. pp.124-133, San Francisco,CA,2000,doi: 10.1109/MASCOT.2000.876437.
[18] M. Spjuth, Refinement and Evaluation of the Elbow Cache, master's thesis, Department of computer systems, Uppsala University, Sweden, 2002.