British Journal of Mathematics & Computer Science

British Journal of Mathematics & Computer Science
12(3): 1-13, 2016, Article no.BJMCS.20700
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

Individual Character Comparison Technique for Improving the Internal Memory Performance

Jasim A. Ghaeb[1*]

[1]*Faculty of Engineering, Philadelphia University, P.O.Box 19392, Amman, Jordan.*

Article Information

DOI: 10.9734/BJMCS/2016/20700
Editor(s):
(1) Qiang Duan, Information Sciences & Technology Department, The Pennsylvania State University, USA.
Reviewers:
(1) Kexin Zhao, University of Florida, USA.
(2) Sarminah Samad, Universiti Teknologi Mara, Malaysia.
(3) Anand Nayyar, KCL Institute of Management and Technology, Punjab, India.
Complete Peer review History: http://sciencedomain.org/review-history/11712

Original Research Article

Received: 04 August 2015
Accepted: 10 September 2015
Published: 07 October 2015

Abstract

The efficiency of the cache mapping technique depends on how the cache lines are organized and the way that is used to look for and hit the target cache line. In this paper, an efficient technique is proposed to obtain a significant improvement in average hit time of a line in the cache. The paper presents Distributive Comparison Approach (DCA) that significantly minimizes the hit time and improve cache hit ratio. The efficient of DCA is based on how the cache lines are compared and picked up the coveted one leading to a low cache hit ratio. In DCA, the cache line is assigned by multi tags where each individual tag is only one character. Then, instead of one line tag of complete characters per a comparison cycle, the comparator is flushed by multi tags of different lines in the cache. Also the cache lines that are come from the main memory classified into two groups; even and odd line's tags to reject the unwanted lines form the multi-tag comparison. These two procedures practically speed up the repelling of misfit tagged lines and consequently the hitting of the target line in the cache. Simulation results show that the DCA outperforms well-known mapping techniques including FAMT and SMT.

Keywords: Cache memory; memory hierarchy; cache hit ratio; two-level memory; cache mapping.

1 Introduction

The growing gap between processor speeds and memory speeds is resulting in increasingly expensive cache misses, underscoring the need for sophisticated cache hierarchy techniques. The well organized cache hierarchy improves the cache hit ratio and minimizes cache misses. In the meanwhile, the increased demand for the higher transfer data rates between subsystems in most of today multimedia applications is achieved

*Corresponding author: E-mail: gaebja@yahoo.com;

through the utilization of cache memory as a fast data bus/buffer between communication entities. The cache can improve performance of the computer system if the application software is planned carefully for utilizing the cache [1,2].

During data processing by computers huge amount of information is shuffled between the memory and processor. The large amount of information needs a greater capacity memory of a smaller cost per bit and a lower average access time. Intuitively, the principle of capacity makes sense. A small-capacity memory requires a lower access time. Obviously, it is easier and faster to look for a location in 1 Kbyte memory than in 1Mbytes memory. The high access time and low cost per unit are required for a computer memory. The speed of a memory depends on its ability to keep up with the processor. For efficient data transfer and a lower data access time, the main computer memory (usually large and slow) is supplemented by a small and a high speed memory called cache [3]. It is the faster memory available due to its high technology fabrication and its small size. The upper-level memory or cache is used as a temporary storage of data chunks shuffled between the lower-level memory and the processor. For a computer system of memory hierarchy of two levels or more, the system first copies the data needed by the processor from lower-level memory into cache and then from the cache into the local memory of the processor. Comparatively, the upper-level memory of a lower capacity has a lower access time compared to the lower-level memory of large capacity [4]. In order to run different applications on embedded systems efficiently the memory subsystem needs to be optimized [5,6].

Employing the memory hierarchy, the average access time is increased and the number of accesses to that memory is decreased as the memory goes far from the processor. Therefore, it is necessary to organize the data such that the percentage of accesses to cache is more than the main memory. When a memory reference is addressed, a first trial is made to find the word in the cache. If this attempt succeeds a quick access is obtained, otherwise, the particular block of references is copied from the lower-level memory into cache and the processor will access the memory words via the cache.

The cache helps in organizing the movement of data transfer between main memory and processor registers to improve the performance. Data transferred from lower-level memory to the cache is held in terms of block-sized units [7]. This transfer process is similar to the process of column to row transposition of the word locations. The size of low-level memory is larger compared to the cache size, therefore, the cache lines cannot be uniquely and constantly pointed to a particular block [8]. Obviously during the execution of a program, one of the blocks of data has to be removed from the cache to allow a new block to fetch from the main memory. Many mapping algorithms had been proposed to map the blocks of low-level memory into cache lines. All these techniques have tried to attain a low cache hit time, a low cache miss ratio and no cache conflicts leading to improved cache access efficiency [9,10,11]. An even or odd technique is proposed by [12] to improve the cache performance. In this technique, the line's tags are passed through one-bit comparator to speed up the rejection of the cache lines of opposite least significant bit to the target cache line. This technique is useful if the current values of the line's tags come equally between the even values and odd values. But there is no improvement in the cache hit ratio if all or more of the current values of the line's tags in the cache are coming in even or odd order and the cache line that is looking for is also in even or odd order.

In this paper, an efficient technique is proposed to improve the cache hit time. The proposed DCA divides the original tag of a cache line into multi tags; where each tag is reduced to one character. Therefore, instead of feeding the control logic by one line-tag of complete C sub-tags to complete one comparison per a time, the proposed technique can fetch the control logic circuit by C line's sub-tags to execute C comparisons per a time. The proposed alteration does not need to wait for a complete comparison of a line tag, and can repel many lines per a comparison that their tags do not match. Thus the time required to reach the target line is minimized and consequently the cache hit time is improved. Simulation results for the proposed DCA scheme and FAMT and SMT schemes at different cache sizes are presented later in the paper. These results show that DCA algorithm outperforms FAMT and SMT offering significant reduction in cache hit time.

The paper is organized as follows: Section 2 introduces the average access time for assigning of a word in the cache. In Section 3, we introduce some fundamentals of cache organization. In Section 4, the well-known cache mapping techniques are discussed in detail. The proposed technique to improve the cache hit time is presented in Section 5. Section 6 presents the simulation results and shows the performance of the proposed technique against the well-known schemes. Finally, Section 7 provides conclusions and work summary.

2 Average Access Time

From conjectural point of view, the principle of locality for items of data makes sense. According to the principle of locality, there are number of accesses to items in the block that is brought into cache, leading to faster overall access time. The fraction of all memory accesses in the cache is the cache hit ratio. 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. For two memory levels, the average access time is determined in terms of cache hit and cache miss ratios and access times for cache and main memory [13]. It is given by:

$$T_{av} = H \times T_c + M \times (T_c + T_m) \tag{1}$$

Where;

$T_{av:}$ average access time of a memory word.

H: cache hit ratio.

M: cache miss ratio, (M= 1-H).

T_c and T_m: cache access time and main memory access time, respectively.

Equation (1) ignores the time required by the cache control logic to determine if the word is in the cache or not. It may be rewritten as below:

$$T_{av} = H \times T_c + (1-H)(T_c + T_m) \tag{2}$$

$$T_{av} = T_c + (1-H)T_m \text{ , so;}$$

$$T_{av} = T_c + MT_m \tag{3}$$

3 Well-Known Cache Mapping Techniques

The power of the cache mapping technique depends on how the cache lines are arranged in the cache and how dynamically the cache lines are tracked for a high opportunity of line hitting and then a low hit time.

To highlight the performance of the proposed technique, a comparison is made with the fully associative mapping technique and the set-associative mapping of K-line. In subsection below, the fully associative mapping technique and set-associative mapping technique are discussed in more detail.

3.1 Fully Associative Mapping Technique (FAMT)

The direct mapping technique gives a fixed cache line value for each block of the main memory, but with different tags [14,15]. The disadvantage of this technique is manifested in the replacement of a cache line that still contains data needed shortly afterwards The Fully Associative Mapping Technique (FAMT) overcomes the potential disadvantage of line hit accompanied with the direct mapping, by allowing each block in the main memory to be located into any line in the cache. In other words, there is elasticity as to which cache line to swap when a new data has to be loaded from main memory, and thus the hit ratio in the cache will be high. The FAMT solves the problem by considering the main memory address completely as tag field instead of a tag and line field [16]. The “*r*” bits of the line field and “*t*” bits of the tag field are reduced into one field called tag field, as shown in Fig. 1.

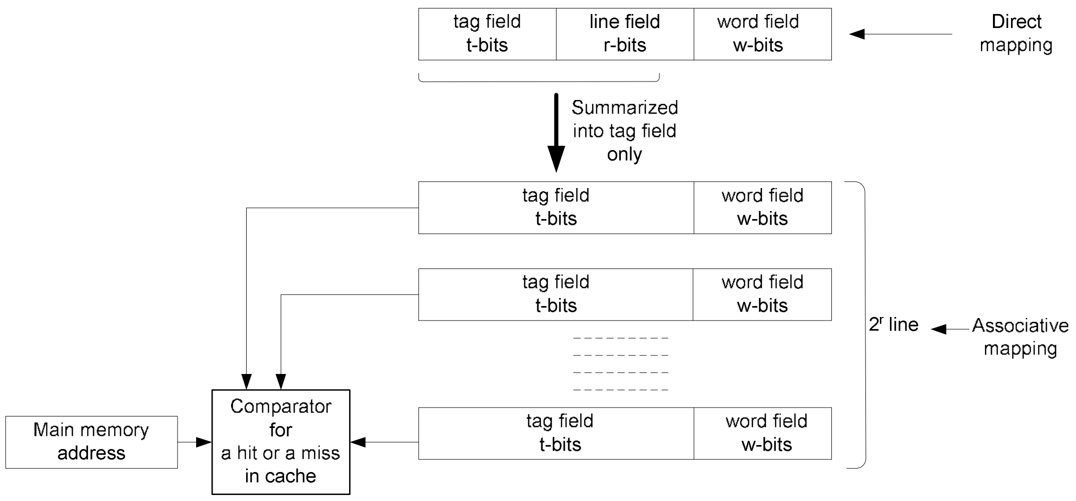

Fig. 1. The fully associative mapping technique (FAMT)

3.2 Set-Associative Mapping Technique (SMT)

The set associative mapping technique (SMT) allows a limited number of blocks, with the same index and different tags, to be in the cache. Therefore SMT can be considered as a compromise between fully associative and direct mapping schemes. SMT divides the cache into many associative mapped caches, each of which is called a set [17,18]. Each set consists of K-line to be referred as “K-way set associative mapping”. Responding to an incoming address, the cache control logic first determines which set has to be involved, then inside the selected set the associative mapping is implemented to appoint the line that is looking for in the cache, as shown in Fig. 2. For a single set, the set associative mapping is reduced to fully associative mapping and the control logic compares the K-tags for each line and does not need to index the set number. For a single line per set, the set associative mapping is reduced to direct mapping and the cache control logic needs to index the corresponding cache line for a comparison. The set associative mapping technique reduces the number of comparisons and the bit size of the line-tags in each set but it increases the time required for indexing the cache sets.

4 Multi-Tag Technique (MTT)

When a block of data needs to be read from its assigned line in the cache, it is necessary to brand its data to be discriminated from other data. The tag-comparator in the cache control logic carries out this work. The tag-comparator plays a crucial role in cache hit time, and thus in cache hit ratio. Instead of feeding the control logic by one line-tag per comparison as in the FAMT, multi sub-tags of many lines are flushed to the control logic by the proposed MTT. In this case, there are always *C* sub-tags comparisons rather than just one tag comparison per a time. The control logic receives *C* sub-tags from different cache lines and compares them simultaneously. These *C* sub-tags are coming from different lines of the same order (n). By comparing *C* sub-tags with the corresponding character (Y_n) of the main memory address, the line of a misfit sub-tag will be discarded directly and does not need to compare the rest of its multiple sub-tags. The number of sub-tags that are discarded from the current comparison is compensated by new sub-tags from the cache in the next cycle of comparison. The line-tag that has full *C* matches of comparison is the one that is looking for. The MTT increases the number of sub-tags per one comparison, leading to a short time for the line hitting and then a high cache hit ratio is achieved. The subsections below present the detail of the MTT algorithm.

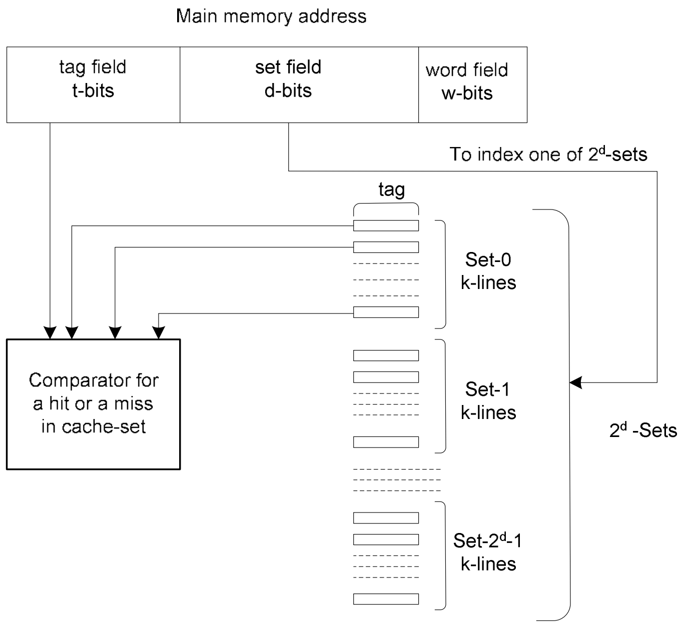

Fig. 2. The set-associative mapping technique (SMT)

4.1 MTT Algorithm

The number of lines in the cache is assumed 'D-lines', each line tag is distinguished by a '*C* sub-tags'. The length of each sub-tag (X_{nm}) is four-bit number. The tag-order in the cache is assumed 'm' and the sub-tag order in the tag is 'n'. The comparator of *C* length in the control logic is feeding from cache by a cache word of *C* length. This cache word has the same sub-tag order for all different lines, as shown in Fig. 3. The cache word is compared with the corresponding sub-tag in the main memory address. The “m” in X_{nm} is varied for all characters in the cache-word array. It can be any tag number in the cache, depending on how many tags are discarded from the current comparison and what their orders are.

4.2 MTT Algorithm Sequence

The algorithm sequence used by MTT for low tag hitting ratio is shown in Fig. 4. It consists of the following steps:

- **Step.1** Start to feed the cache control logic by the cache word. It consists of the Least Significant Sub-tags (LSSs) of the first *C* tags in the cache (X_{1m}, X_{1m}, X_{1m} . . . until *C* length). This cache word is compared with the main memory address word. The memory address word is a *C* repeated of the LSSs of the main memory address.

- **Step.2** If there is a match between the memory word sub-tag and any cache line sub-tag, keep this line for next cycle of comparison.
- **Step.3** If there is a misfit for any sub-tag, discard its line and feed the cache word with a sub-tag of a new line. The sub-tag order of the new line must be the same order of the current sub-tag in the cache word.

Continue in comparison and discard any line of a misfit sub-tag. The one that is reaching the C continuous matches is the cache line that is being sought and the comparison has to be stopped.

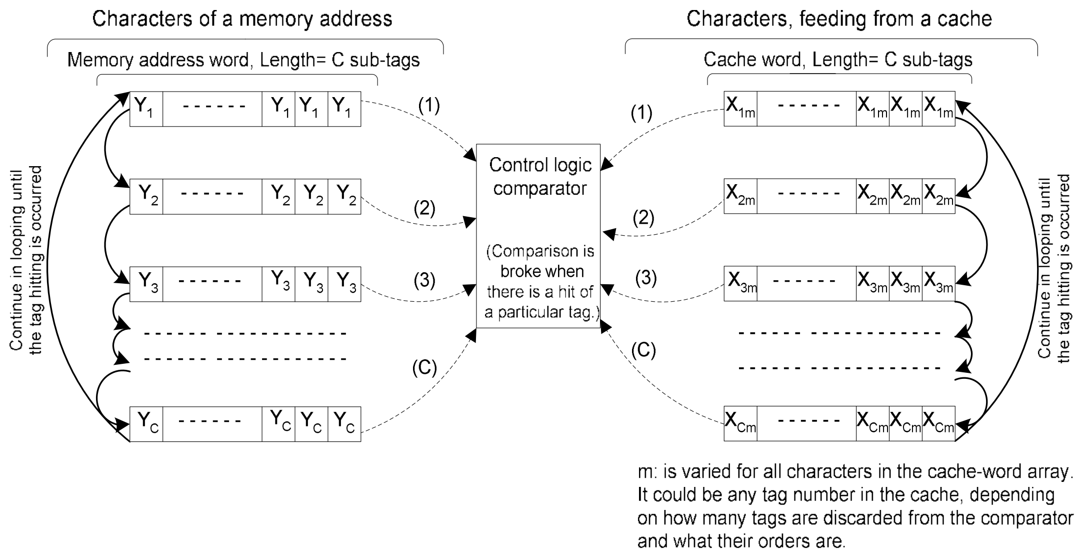
Characters of a memory address
Memory address word, Length= C sub-tags
Characters, feeding from a cache
Cache word, Length= C sub-tags
Control logic comparator
(Comparison is broke when there is a hit of a particular tag.)
Continue in looping until the tag hitting is occurred
(1)
(2)
(3)
(C)
m: is varied for all characters in the cache-word array. It could be any tag number in the cache, depending on how many tags are discarded from the comparator and what their orders are.

Fig. 3. The multi-tag technique (MTT)

4.3 Even-or-Odd Division Technique (EODT)

Obviously, the address of a corresponding line in the cache is tagged by an even value or odd value. This truth simplifies and speeds up the rejection lines of opposite Least Significant Bit (LSB) to the cache line value that is looking for. Therefore, one bit comparison stage is added before the MTT comparison stage, to improve the cache hit ratio. The EODT works effectively if the current values of the cache line's tags of opposite LSB to the one that is looking for are coming more or equally. But EODT gives a worse hit ratio if all or more of the current values of the line's tags in the cache are coming in even or odd order and the cache line that is looking for is also in even or odd order.

4.4 Distributive Comparison Approach (DCA)

The use of EODT makes direct and quick rejection for the misfit LSB cache lines from the cache control logic, as mentioned in Sec 4.3.

Instead of feeding the control logic by one line-tag per comparison as in the traditional methods, multi sub-tags of many lines are flushed to the control logic by the proposed MTT. In this case, there are always *C* sub-tags comparisons rather than just one tag comparison per a time, leading to a short time for the line hitting and then a high cache hit ratio is achieved. The using of MTT speeds up the number of discarded lines of misfit tags from the cache control logic, as mentioned in Sec 4.

The goal of this work is to minimize the cache memory hit time which affects significantly the overall performance of system. Employing MTT and EODT, a Distributive Comparison Approach (DCA) is achieved for an optimum cache hit ratio as we will see in the simulation results.

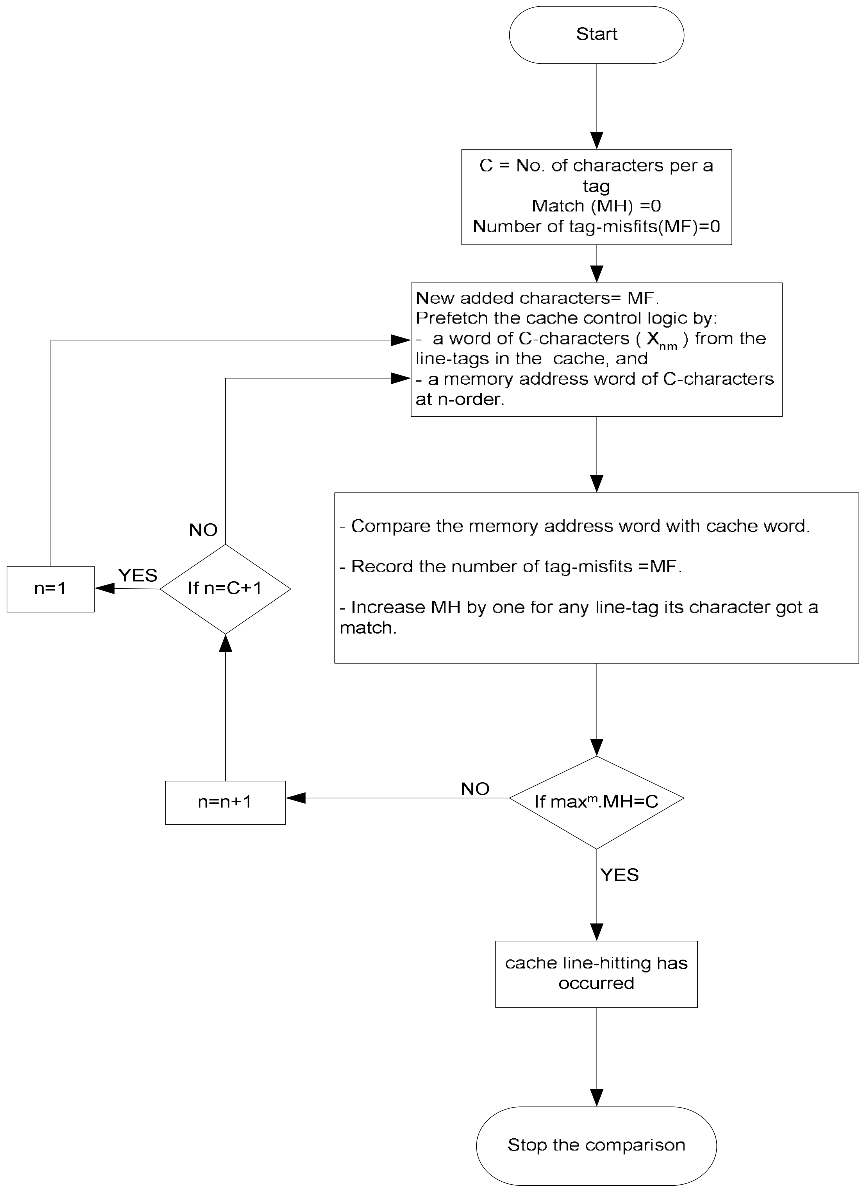
Start
C = No. of characters per a tag
Match (MH) =0
Number of tag-misfits(MF)=0
New added characters= MF.
Prefetch the cache control logic by:
- a word of C-characters (X_{nm}) from the line-tags in the cache, and
- a memory address word of C-characters at n-order.
- Compare the memory address word with cache word.
- Record the number of tag-misfits =MF.
- Increase MH by one for any line-tag its character got a match.
NO
YES
n=1
If n=C+1
NO
n=n+1
If max^m.MH=C
YES
cache line-hitting has occurred
Stop the comparison

Fig. 4. Algorithm sequence applied by MTT for a low hit ratio in the cache

5 Results and Discussion

The counted time taken by the cache control logic to compare the main memory tag with the cache line tags in order to hit the target line in the cache is the tag hit time. A java program is written to select the lines in the cache randomly and to look for and hit them. The performance of the proposed DCA is benchmarked against that of FAMT, SMT and EODT for two different cache sizes. Two main memories of 4 Mbytes and 16 Mbytes along with their corresponding cache memories of 16 K-line and 32 K-line are used to evaluate the performance of the different techniques mentioned previously, for cache hit ratio. The blocks are fed from the main memory into the cache and their lines located randomly in the cache. Table 1 shows the hit times spent by the proposed DCA technique and other techniques FAMT and MTT, to read lines located randomly in 32 k-line cache of main memory size of 4 Mbytes. The high performance of the proposed DCA is appeared clearly compared to FAMT and MTT. It spent a very low time to hit a line in the cache. Fig. 5 shows the hit time that was spent for hitting a line in cache for the well-known technique FAMT and our proposed MTT technique. The main memory size is taken 4 Mbytes and its cache is 16 K-line. The cache lines are located in random tags; and twenty line tags are generated randomly for simulation. The power of the proposed MTT technique is appeared clearly as it is given by the very low time to hit a tagged line of data in the cache. The average time that is taken for hitting the twenty line tags in the cache is 12.92170 μs by FAMT and 2.3795 μs by MTT. For system clock of 5 MHz, the number of machine cycles taken by MTT for the tag matching are 12 whereas by FAMT are 64. An improvement of 81.58% in average hit time is attained by MT. Fig. 6 shows the hit time spent by the FAMT and MTT to read a tagged line in 32 K-line cache which is loaded from a 16 Mbytes main memory. It is clear that MTT overcome FAMT for reading a tagged line in the cache. This is due the efficient performance of MTT by achieving 82.19% improvement in average hit time.

Fig. 7 shows the performance of FAMT, EODT and MTT for hitting a line in the 32 K-line cache. It is clear that MTT outperforms FAMT and EODT due to its quick rejection of misfit cache lines to reach the target line. The simultaneous employing of both MTT and EODT establishes the DCA which is provided an optimum cache hit ratio as shown in Fig. 8. Fig. 9 shows the time taken by the well known mapping technique FAMT and the proposed DCA. It is shown that a high performance is achieved with DCA due to its high response of rejection of non-conforming tags from the control logic. This high performance is achieved by employing both MTT and EODT simultaneously to establish the DCA technique. For 4-line per a set of 32 K-line cache memory, the performances of SMT and the proposed DCA techniques are shown in Fig. 10. The DCA has produced a lower cache hit time compared to SMT. The average time that is taken for hitting a tag in the cache is 3.75 ns by SMT and 1.0275ns by DCA. It is clear that there is an improvement of 72.60% in average hit time attained by DCA.

The results conclude that the DCA has showed a high performance for cache hit ratio compared to the well-known techniques: FAMT, SMT and EODT.

Table 1. Hit times spent by the proposed DCA technique and other techniques FAMT and MTT, to read lines located randomly in 32 k-line cache and main memory size of 4 Mbytes

Tag vales of lines located randomly in cache (Hexadecimal)	Hit time (μs) by		
	FAMT	MTT	DCA
009412	46.4085	10.2605	8.2480
01998F	9.2130	4.6520	1.6350
088E77	13.1100	5.3410	2.3295
0AA6C6	38.7510	9.9030	6.8920
21803B	37.8720	9.7395	6.7350
2BE5EB	31.1385	9.0112	5.5437
2E1CB5	35.7315	9.3650	6.3585
31289E	36.1470	9.4380	6.4292
319A51	32.5275	8.9540	5.7840
36FD13	37.7760	9.2953	6.7230
4E0F98	38.3865	9.2750	6.8150
5ADC60	40.8255	11.7052	7.2617

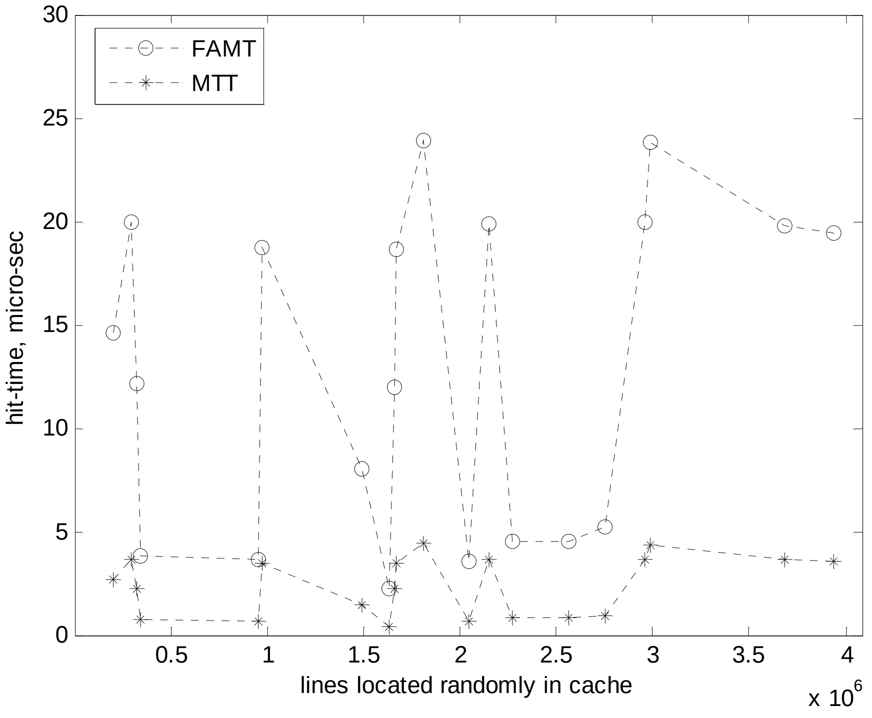

Fig. 5. Hit times for reading a line in 16 K-line cache. Main memory Size is 4 Mbytes

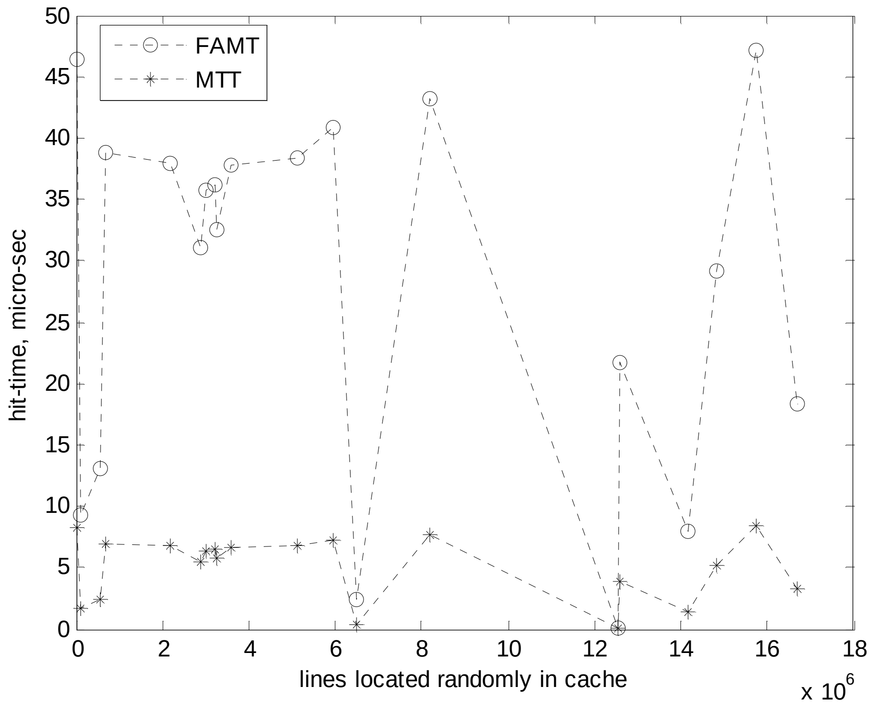

Fig. 6. Hit times for reading a line in 32 K-line cache. Main memory size is 16 Mbytes

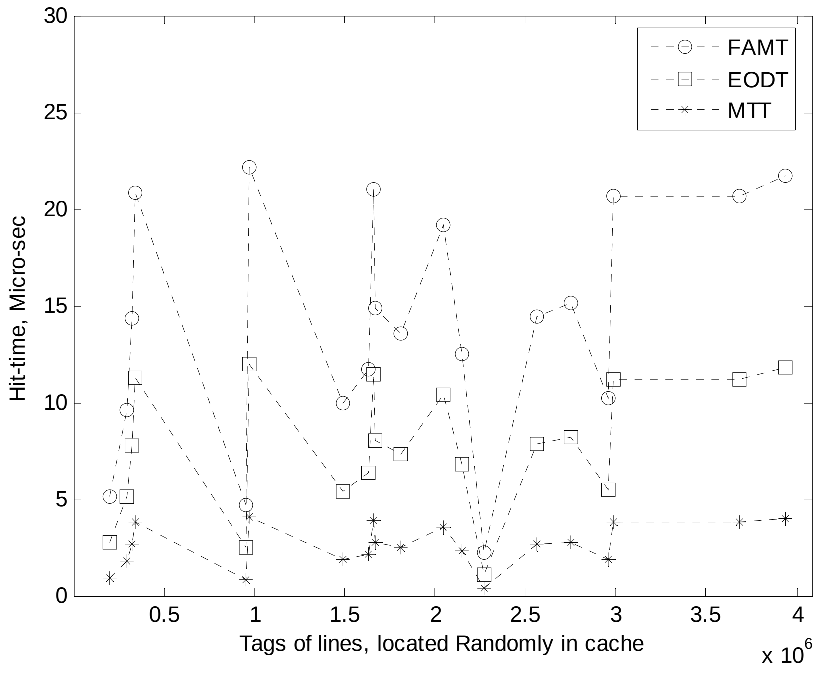

Fig. 7. Hit times for reading a line in 32 K-line cache taken by FAMT, EODT and MTT. Main memory size is 16 Mbytes

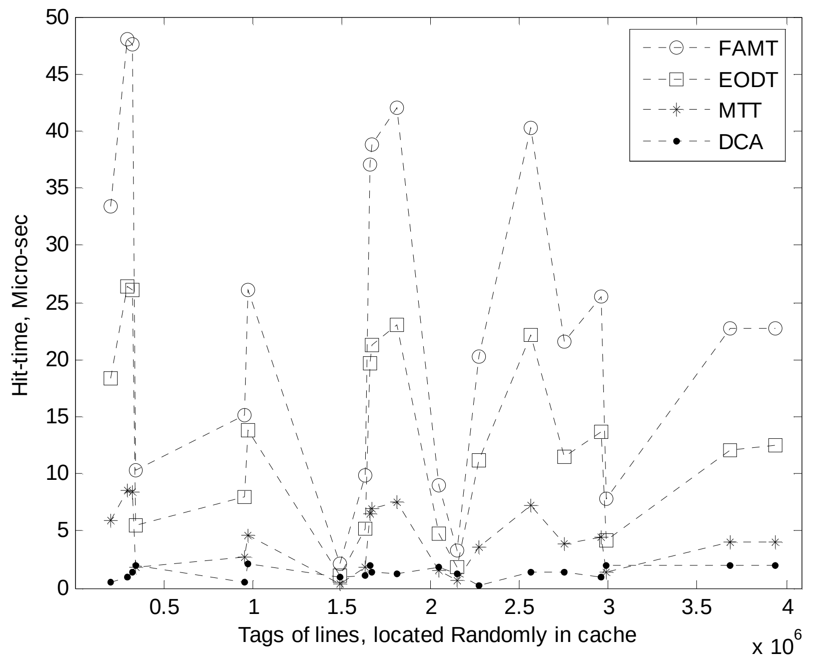

Fig. 8. The simultaneous employing of both MTT and EODT to establish DCA for reading a line in 16 K-line cache. Main memory size is 4 Mbytes

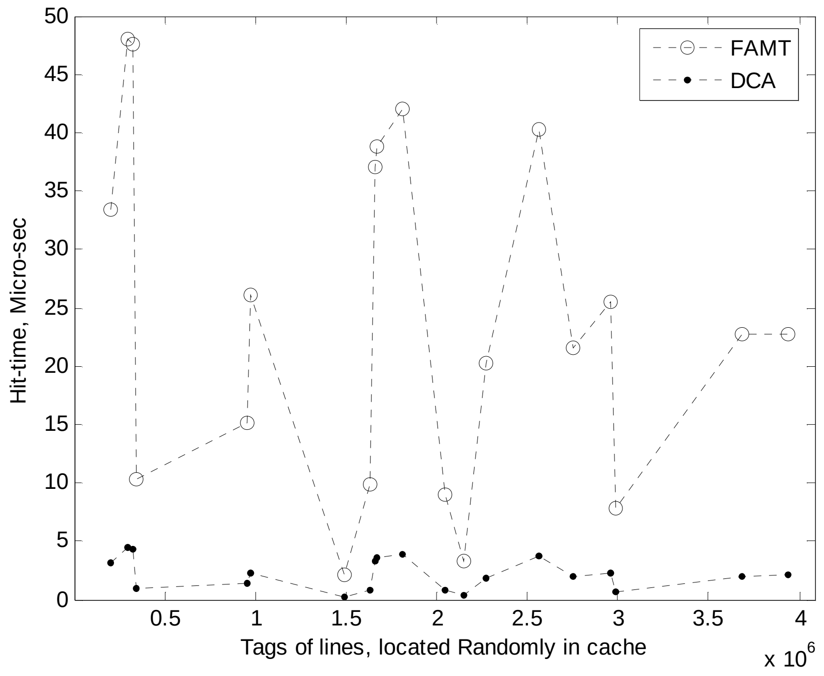

Fig. 9. Hit Times for reading a line in 32 K-Line Cache spent by FAM and DCA for reading a line in 32 K-line cache. Main memory size is 16 Mbytes

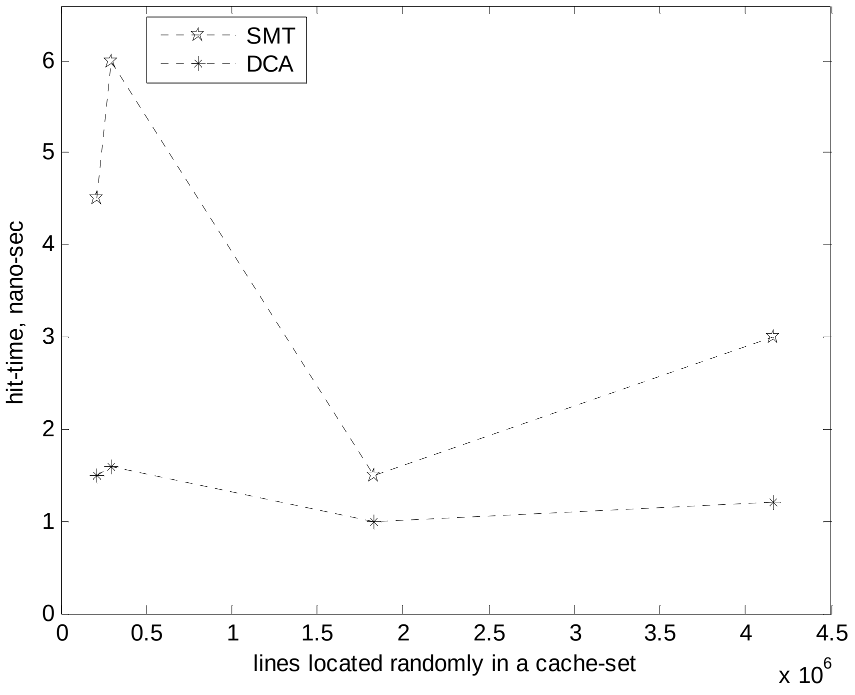

Fig. 10. Hit times for reading a line in 32 K-line cache. Main memory size is 16 Mbytes

6 Conclusions

In this paper, the proposed technique makes an individual character comparison for many cache line-tags simultaneously in the way to reject any line-tag that its character does not match. The performance of the DCA to improve the cache hit ratio, and consequently improve the overall system speed is benchmarked against that of FAMT and SMT schemes. The simulation results for two different cache sizes clearly depict a significant improvement in average hit time of 81.58% and 64.46% against FAMT and SMT respectively.

The index process together with the comparison process makes a delay in the average access of a line in the cache. Further work can be made to improve both index and comparison processes for a low average hit time.

Competing Interests

Author has declared that no competing interests exist.

References

[1] Wang W, Tafil D. Performance enhancement on microprocessors with hierarchical memory system for solving large space linear systems. The International Journal of Supper Computing Applications. 1999;13:63-79.

[2] Yehuda A, David D, Adam M. Cache index-aware memory allocation. In: 10th International Symposium on Memory Management; San Jose, CA, USA: 4-5 June; 2011. DOI: 10.1145/2076022.1993486.

[3] Gagged G, Paresh R, Madarkar J. Survey on hardware based advanced technique for cache research in computer science optimization for RISC based system architecture. International Journal of Advanced and Software Engineering. 2013;3-9:156-160.

[4] Mamagkakis S, Atienza D, Poucet F, Catthoor D, Soudris JM. Custom design of multi-level dynamic memory management subsystem for embedded systems. In: IEEE Workshop on Signal Processing Systems; New York, USA: IEEE. 2004;170-175.

[5] Baloukas C, Risco-Martin JL, Atienza D, Poucet C, Papadopoulos L. Optimization methodology of dynamic data structures based on genetic algorithms for multimedia embedded systems. Journal of Systems and Software. 2009;82:590–602.

[6] Eichenbaum H. Memory systems. Wiley Interdisciplinary Reviews: Cognitive Science. 2010;1: 478–490.

[7] Chilimbi T, Hill M, Larus R. Cache-conscious structure layout. In: ACM SIGPLAN Conference on Programming Language Design and Implementation; New York, NY, USA. 1999;13-24.

[8] Stallings W. Computer organization and architecture: Designing for performance. 7th Edition: Prentice Hall; 2006.

[9] Serhan S, Abdel-Haq H. Improving cache memory utilization. World Academy of Science, Engineering and Technology. 2007;26:299-304.

[10] Topham N, GonZalez A. Randomized cache placement for eliminating conflicts. IEEE Transactions on Computers. 1999;4-2:185-192.

[11] Bae J, Kyung M. A supplementary scheme for reducing cache access time. Trans. on Information and Systems. 1996;E79-d:385-389.

[12] Ghaeb J. Enhancing cache performance based on improved average access time. In: ICCCISE International Conference on Computer, Communication and Information Sciences and Engineering; 25-26 April; Paris, France. 2012;815-820.

[13] Hill M, Smith A. Evaluating associatively in CPU caches. IEEE Transactions on Computers. 1989; 38-12:1612-1630.

[14] Jouppi N. Improving direct- mapped cache performance by the addition of a small fully-associative cache and prefetch Buffers. In: 17th Annual International Symposium on Computer Architecture; WA, USA: IEEE. 1990;364-373.

[15] Wilton S, Jouppi N. CACTI: An enhanced cache access and cycle time model. IEEE Transactions on Solid State Circuits. 1996;31-5:677-688.

[16] Singh JA. Model of workloads and its use in miss-rate prediction for fully associative aches. EEE Transactions on Computers. 1992;41-7:811-825.

[17] Lin P A. 0.8-V 128Kb four-way set-associative two-level CMOS cache memory using two-stage word line/bit line-oriented tag-compare (WLOTC/BLOTC) scheme. IEEE Journal of Solid-State Circuits. 2002;37-10:1307-1311.

[18] Palsodkar P, Deshmukh A, Bajaj P, Keskar A. An approach for four way set associative multilevel CMOS cache memory. Lecture Notes in Computer Science. 2007;4692:740-746.