IOSR Journal of Electronics and Communication Engineering (IOSRJECE)
ISSN : 2278-2834 Volume 1, Issue 6 (July-Aug 2012), PP 31-37
www.iosrjournals.org

Vlsi Design of Cache Compression in Microprocessor Using Pattern Matching Technique

M. Nisha Angeline [1], Prof. S. Valarmathy [2], S. K. Manikandan [3], Prof. C. Palanisamy[4]

[1](*Asst.Professor, Velalar College of Engineering and Technology,*
[2](*H.O.D/ECE, Bannariamman Institute of Technology, Sathy,*
[3](*Asst.professor, Velalar College of Engineering and Technology,*
[4](*H.O.D/IT, Bannariamman Institute of Technology, Sathy,*

Abstract: *Microprocessors speeds have been increasing faster than speed of off-chip memory. When multi processors are used in the system design, more processors require more accesses to memory. Thus it raises a 'wall' between processor and memory. Accessing off-chip memory takes an order of magnitude more time than accessing an on-chip cache, two orders of magnitude more time than executing an instruction. Cache compression presents the challenge that the processor speed has to be improved but it should not substantially increase the total chip's power consumption. This Cellular Automata (CA) based pattern matching architecture has number of novel features tailored for the application. The compression is based on pattern matching and dictionary matching and if the pattern matches, the dictionary matching has to be bypassed. The compressor is composed of Pattern matching and Priority Unit. In this paper modified Priority unit is proposed. By this method the speed and the power can be improved without affecting the performance of system cache.*

Keywords*: Cellular Automata (CA), Dictionary Matching (DM), Pattern matching (PM), Priority Selection Unit (PSU).*

I. Introduction

Memory latencies have long been a performance bottleneck in modern computers. In fact, with each technology generation, microprocessor execution rates outpace improvements in main memory latencies. Memory latencies are thus having increasing impact on overall processor performance. The rift between processor and memory speeds is alleviated primarily by using caches. Today's microprocessors have on-chip cache hierarchies incorporating multiple megabytes of storage. Increasing the size of on-chip caches can greatly increase processor performance. However, the amount of on-chip storage cannot be increased without bound. Caches already consume most of the die area in high performance microprocessors. The fabrication costs of larger die, and ultimately limited by semiconductor manufacturing technology. Practical cache sizes are constrained by the increased Memory bandwidth is also a scarce resource in high-performance systems. Several researches used hardware based compression to increase effective memory size, reduce memory address and bandwidth and increase effective cache size.

This paper addresses the increasingly important issue of controlling off-chip communication in computer systems in order to maintain good performance and energy efficiency. The ongoing move to chip-level multiprocessors (CMPs) is further increasing the problem; more processors require more accesses to memory, but the performance of the processor-memory bus is not keeping pace. Techniques that reduce off-chip communication without degrading the performance have the potential to solve this problem. Cache compression is one such technique. Cache compression presents several challenges. First, decompression and Compression must be extremely fast: a significant increase in cache hit latency will overwhelm the advantages of reduced cache miss rate. Then Secondly the hardware should occupy little area compared to the corresponding decrease in the physical size of the cache, and should not substantially increase the total chip power consumption. Finally, cache compression should not increase power consumption substantially. The above requirements prevent the use of high-overhead compression algorithms .A faster and lower-overhead technique is required. Section II explains the related work regarding the Microprocessor Cache Compression. Section III explains the proposed work.

II. Review Of Literature

2.1 FREQUENT PATTERN COMPRESSION

This compression scheme [4] builds on significance-based compression schemes. It is also based on the observation that some data patterns are frequent and also compressible to a fewer number of bits. For example, many small-value integers can be stored in 4, 8 or 16 bits, but are normally stored in a full 32-bit word (or 64-bits for 64-bit architectures). These values are frequent enough to merit special treatment, and storing them in a more compact form can increase the cache capacity. In addition, special treatment is also given to runs of zeros since they are very frequent. The insight behind FPC is that we want to get most of the benefits of dictionary-

based schemes, while keeping the per - line overhead at a minimum. The FPC compresses on a cache line basis. Each cache line is divided into 32-bit words (e.g., 16 words for a 64-byte line).These patterns are: a zero run (one or more all-zero words), 4-bit sign-extended (including one-word zero runs), one byte sign-extended, one half words sign-extended, one half word padded with a zero half word, two byte sign-extended half words, and word consisting of repeated bytes (e.g. "0x20202020", or similar patterns that can be used for data initialization). These patterns are selected based on their high frequency in many of our integer and commercial benchmarks. A word that doesn't match any of these categories is stored in its original 32-bit format.

Frequent pattern compression, compresses cache lines at the L2 level by storing common word patterns in a compressed format. Patterns are differentiated by a 3-bit prefix. Cache lines are compressed to predetermined sizes that never exceed their original size to reduce decompression overhead. The drawback of this method is that there is no register-transfer-level hardware implementation or FPGA implementation of FPC, and therefore its exact performance, power consumption, and area overheads are unknown.

2.2 RESTRICTIVE COMPRESSION TECHNIQUE

With the CMOS scaling trends and slow scaling of wires as compared to the transistors, the cache access latencies will increase in the future microprocessors. To prevent the increasing latencies from affecting the cache throughput, the L1 caches are small-sized and their accesses are pipelined. small-sized L1 data caches can result in significant performance degradation due to increased miss rates. Compression techniques can be used to increase the L1 data cache capacity. However, these compression techniques cannot alter the byte-offset of the memory reference, to avoid any increase in the cache access latency. Restrictive cache compression techniques [2] do not require updates to the byte-offset, and hence result in minimal, if any, cache access latency impact. The basic technique AWN compresses a cache block only if all the words in the cache block are of small size. The compressed cache blocks are then packed together in a single physical cache block. The AWN technique requires minimal additional storage in the cache and results in a 20% increase in the cache capacity.

The AWN technique is extended by providing some additional space for the upper half-word AHS of a few normal-sized words in a cache block, with an aim to convert them into narrow blocks. Further the AHS technique is extended to AAHS so that the number of upper half-words used by a cache block can vary depending on the requirement. AHS and AAHS techniques increase the cache capacity by about 50%, while incurring a 38% increase in the storage space required, compared to a conventional cache. However, these techniques still do not impact the cache access latency. To reduce the additional tag requirements (which is inevitable with any cache compression technique), it is proposed to reduce the number of additional tag bits provided for the additional cache blocks to be packed in a physical cache block, reducing the overhead of the AHS techniques to about 30%.

The byte-offset of each word in the cache block will depend on the size of the words before it. This will require recalculating the byte-offset to read a word from the block. Therefore, it is imperative that any compression technique that is applied to L1 caches should not require updates to the byte-offset. Such compression techniques can be called as restrictive compression techniques. The drawback of this technique is that it cannot alter the byte-offset of the memory reference, to avoid any increase in the cache access latency.

2.3 INDIRECT INDEX CACHE WITH COMPRESSION

Indirect index cache [3] allocates variable amount of storage to different blocks, depending on their compressibility. The Indirect Index Cache with Compression is based on the IIC. The basic IIC consists of a data array containing the cache blocks and a tag store which contains the tags for these blocks. Each IIC tag entry holds a pointer to the data block with which it is currently associated. This indirection provides the ability to implement a fully associative cache.

Replacements in the IIC are managed by a software algorithm running on an embedded controller or as a thread on the main CPU. Our primary algorithm, called Generational Replacement (GEN), maintains prioritized pools (queues) of blocks; periodically, referenced blocks are moved to higher priority pools and unreferenced blocks are moved to lower priority pools. Replacements are selected from the unreferenced blocks in the lowest priority pool. To reach the highest priority pool, a block must be referenced regularly over an extended period of time; once there, it must remain unreferenced for a similarly long period to return to the lowest priority pool. This algorithm thus combines reference regency and frequency information with a hysteresis effect, while relying only on block reference bits and periodic data structure updates. To ensure that adequate replacement candidates are available to deal with bursts of misses, the algorithm can identify multiple candidates per invocation and maintain a small pool of replacement blocks. These blocks are used by hardware to handle incoming responses, while GEN works in the background to keep this pool at a predetermined level. The Figure.1 illustrates the pipelined data cache read with byte-offset adjustment. The IIC was originally designed to utilize large on-chip caches to achieve better performance over traditional LRU caches. The main drawback of this technique is that one cannot reliably determine whether the architectural schemes are beneficial without, a cache compression algorithm and hardware implementation are designed and evaluated for effective

system-wide compression ratio, hardware overheads, and interaction with other portions of the cache compression system.

2.4 SELECTIVE COMPRESSION TECHNIQUE

The Selective compression [6] means that a data block is compressed only if its compression ratio is less than a specific compression threshold value. This technique can reduce the decompression overhead because decompression process is required only for the compressed data blocks. In addition, data expansion problem can be easily solved by compressing only well-compressible blocks. As the threshold value becomes lower, the average compression ratio of whole data blocks increases, so that the compression benefits may be reduced. However, the measurement of tradeoff between decompression time and compression ratio over the variation of threshold value shows that the selective compression technique can provide performance advantage by reducing the impact of decompression overhead without meaningful loss of compression ratio.

The main drawback of this technique is that one cannot reliably determine whether the architectural schemes are beneficial without, a cache compression algorithm and hardware implementation are designed and evaluated for effective system-wide compression ratio, hardware overheads, and interaction with other portions of the cache compression system.

2.5 X-MATCH COMPRESSION ALGORITHM

X-Match is a dictionary-based compression algorithm that has been implemented on an FPGA [5]. It matches 32-bit words using a content addressable memory that allows partial matching with dictionary entries and outputs variable-size encoded data that depends on the type of match. To improve coding efficiency, it also uses a move-to-front coding strategy and represents smaller indexes with fewer bits. Although appropriate for compressing main memory, such hardware usually has a very large block size (1 KB for MXT and up to 32 KB for X-Match), which is inappropriate for compressing cache lines. It is shown that for X-Match and two variants of Lempel-Ziv algorithm, i.e., LZ1 and LZ2, the compression ratio for memory data deteriorates as the block size becomes smaller [5]. For example, when the block size decreases from 1KBto 256 B, the compression ratio for LZ1 and X-Match increase by 11% and 3%. It can be inferred that the amount of increase in compression ratio could be even larger when the block size decreases from 256 B to 64 B. In addition, such hardware has performance, area, or power consumption costs that contradict its use in cache compression. For example, if the MXT hardware were scaled to a 65 nm fabrication process and integrated within a 1 GHz processor, the decompression latency would be 16 processor cycles, about twice the normal L2 cache hit latency.

2.6 C-PACK COMPRESSION ALGORITHM

C-Pack targets on-chip cache compression. It permits a good compression ratio even when used on small cache lines. The performance, area, and power consumption overheads are low enough for practical use. This contrasts with other schemes such as X-match which require complicated hardware to achieve an equivalent effective system-wide compression ratio. Prior work in cache compression does not adequately evaluate the overheads imposed by the assumed cache compression algorithms. C-Pack is twice as fast as the best existing hardware implementations potentially suitable for cache compression. For FPC to match this performance, it would require at least 8* the area of C-Pack. The main drawbacks of this technique are that Compression is not performed in all the cases. When the total number of compressed bits exceeds the uncompressed line size, the content of the backup buffer is selected. Here the backup buffer has the actual 64 bit uncompressed word. Compression is made in the ratio of 2:1.Due to its efficiency it cannot be expanded. Frequently used instructions are compressed and stored in FIFO.When that particular instruction leaves the dictionary it re-enters the dictionary in uncompressed form and it has to be again compressed. Hence the memory is not properly used.

Table .1 Pattern encoding for the proposed Method

Code	Pattern
1 0	11 00
In case of 01(or)10 it is matched with words in dictionary	
W X	If 10 = 01 (or) 01 =10 If 10 = 00 (or) 11 (or) 01 (or) 01 = 00 (or) 11 (or) 10

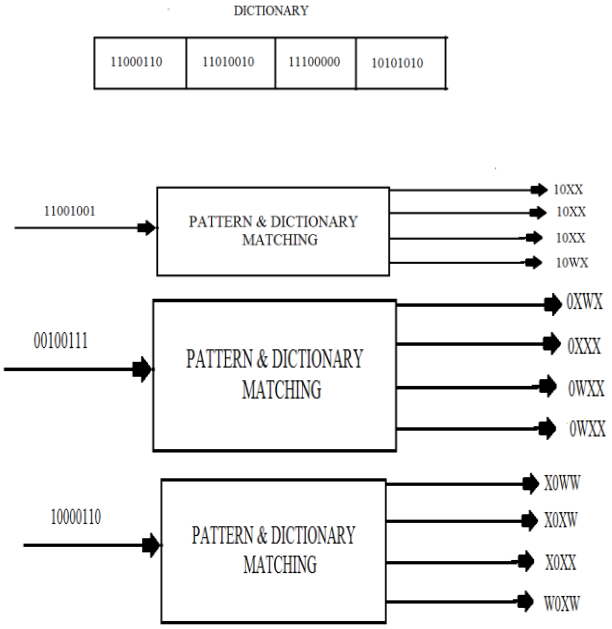

Figure 1 Compression example

III. Proposed Architectures

3.1 COMPRESSION ARCHITECTURE 2:1 RATIO

In the proposed 2:1 architecture shown in Figure 2 is having two pipeline stages:

1)Pipeline stage1: The first pipeline stage performs pattern and dictionary matching on eight uncompressed words in parallel. Each of the eight 8 bits will match with the instruction in dictionary and produces four 4 bit words. After dictionary matching they are concatenated .The output W1, W2, W3 and W4 each has 32 bit and it is fed to the next pipeline stage.

2) Pipeline stage 2: The second pipeline stage performs the selection of word from W1, W2, W3, and W4 depending on the priority. The first priority is give to all one's, Second priority will be give to all zero's, third priority will be given to the combination of zero's and one's, next priority will be given to matched bytes and the least priority will be given to unmatched bytes. This is possible only by counting the number of zero's, one's, matched bytes and unmatched bytes. Counter 0, counter 1 and counter W counts the number of zero's, one's and the matched bytes. Not of matched bytes give the number of unmatched bytes. Later they are selected based on their priority. This work is performed by the comparators. If both the inputs to the comparator has same numbers of zeros, ones and matched bytes then the first input to the comparator will be the output of the comparator. The output of the selection unit will be the compressed 32 bit word. Thus a 64 bit word is compressed to 32 bit word with a compression ratio of 2:1.For example when the input pattern is 11001001 and the dictionary patterns are 11000110, 11010010, 11100000 and 10101010 then the output patterns will be 10xx, 10xx, 10xx and 10wx.If the input pattern given is 10100101111000010110000101100001111 10001011100010110100111000 01 the output pattern obtained after compression is 1x0w1100w100w10011w0w1w0w1x01w0x which is compressed in the ratio of 2:1.

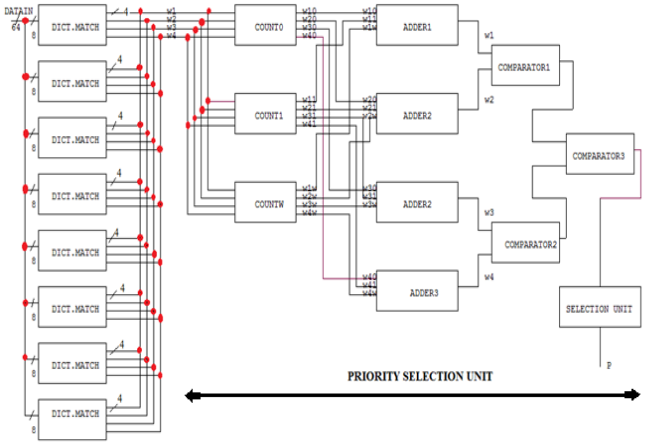

Figure 2. Compression architecture (2:1)

3.2 COMPRESSION ARCHITECTURE 4:1 RATIO

The proposed architecture can be enhanced by implementing the compression with the compression ratio of 4:1.Figure.3 illustrates the hardware compression process. The compressor is decomposed into two pipeline stages.

1) Pipeline stage1: The first pipeline stage performs pattern and dictionary matching on four uncompressed words in parallel. Each of the four 16 bits will match with the instruction in dictionary and produces four 4 bit words. After dictionary matching they are concatenated .The output W1, W2, W3 and W4 each has 16 bit and it is fed to the next pipeline stage.

2) Pipeline stage 2: The second pipeline stage performs the selection of word from W1, W2, W3, and W4 depending on the priority. The first priority is give to all one's, Second priority will be give to all zero's, third priority will be given to the combination of zero's and one's, next priority will be given to matched bytes and the least priority will be given to unmatched bytes. This is possible only by counting the number of zero's, one's, matched bytes and unmatched bytes. Counter 0, counter 1 and counter W counts the number of zero's, one's and the matched bytes. Not of matched bytes give the number of unmatched bytes. Later they are selected based on their priority. This work is performed by the comparators. If both the inputs to the comparator has same numbers of zeros, ones and matched bytes then the first input to the comparator will be the output of the comparator. The output of the selection unit will be the compressed 16 bit word. Thus a 64 bit word is compressed to 16 bit word with a compression ratio of 4:1.For example when the input pattern is 11001001 and the dictionary patterns are 11000110, 11010010, 11100000 and 10101010 then the output patterns will be 10xx, 10xx, 10xx and 10wx.If the input pattern given is 1010010111100001011000010110000111110001011100010110100111000001 the output pattern obtained after compression is 1x0w1100w100w10011w0w1w0w1x01w0x which is compressed in the ratio of 4:1.

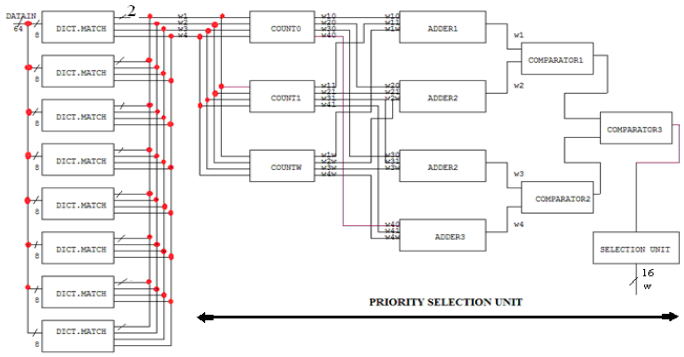

Figure.3 Compressor architecture (4:1 compression ratio)

The decompression of the proposed 2:1 architecture will be a lossless method where as the decompression of the proposed 4:1 architecture is somewhat tedious process. Even though it can be made easy, the decompression of the proposed 2:1 and 4:1 architecture is my future work which is not stated in this paper. My only objective now is to design a compressor architecture which compresses the input pattern without any loss in data. My future work is to design an appropriate hardware cache compressor architecture which produces the input pattern as such after decompression without or with a minimum loss in data.

3.3 PRIORITY SELECTION UNIT (PSU)

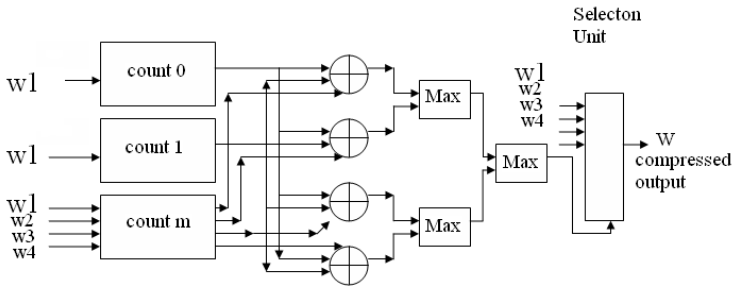

Figure.4 Modified PSU

In the compressor architecture, Priority selection unit plays an important role in this work. Figure 4 shows the modified PSU used in the work. It selects the maximum matched word from the pattern generated by the Pattern matching unit. In the previous work [7], the word is selected by counting the ones, zeros and matched bits. The counting of ones and zeros are same for all the words. Therefore, counting the ones and zeros

can be carried out for single word. Due to this, delay will be reduced and the speed can be increased. By using this logic, new PSU is proposed in the paper which is shown in Figure.5.

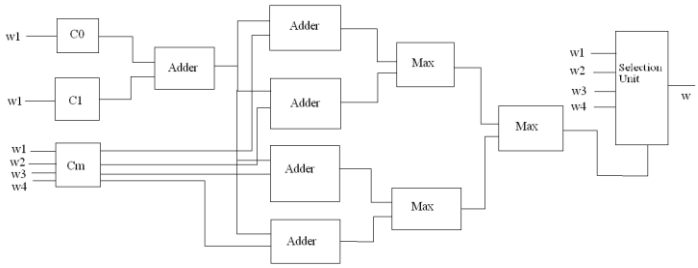

Figure.5. Proposed PSU

In the previous work, the counting of 0 and counting of one are same. Therefore, the addition of ones and zeros are added first. Then this output is again added with counting of matched bits. Due to this, area occupied will be less compared to the previous works.

IV. Results & Discussions

We simulated our design using Model Sim version 10.We have put two simulation results. One shows the 2:1 compression output shown if Figure.6, while the other one is the 4:1 compression output shown in Figure.7.

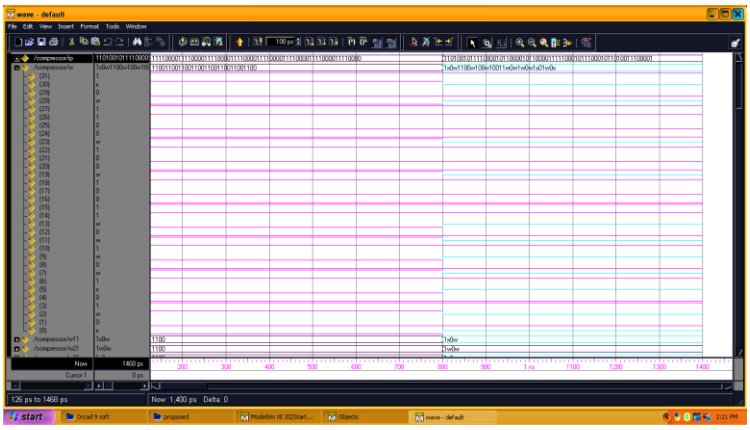

Figure.6 Compressor output (2:1 compression ratio)

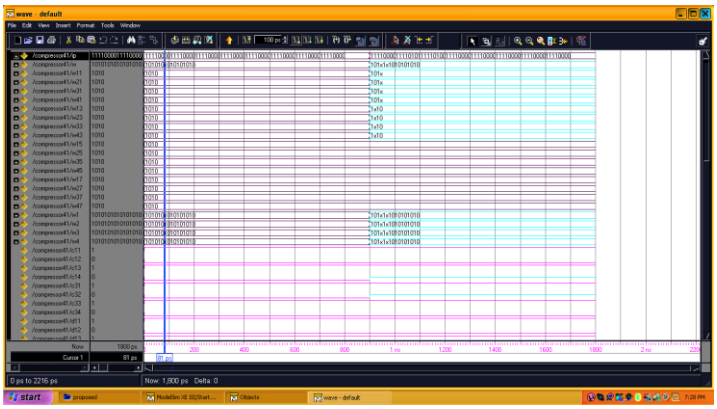

Figure.7 Compressor output (4:1 compression ratio)

Table.2 Compression Ratio comparison

COMPRESSION RATIO (%)	MXT	FPC	X-MATCH	C-PACK	P-MATCH(2:1 COMPRESSION RATIO)	P-MATCH(4:1 COMPRESSION RATIO)
	71.7	68.18	58.64	61.4	50	25

We compared our algorithm to several other hardware compression designs namely C-pack, FPC, X-match, and MXT, that may be considered for cache compression. The Table.2 indicates the compression ratio which varies from algorithm to algorithm. P-Match is being the best and MXT being the worst.

V. Conclusions

This paper has proposed and evaluated an algorithm: P-match for cache compression that honors the special constraints this application imposes. The algorithm is based on pattern matching and partial dictionary coding. Its hardware implementation permits parallel compression of multiple words without degradation of dictionary match probability. Although the proposed hardware implementation mainly targets online cache compression, it can also be used in other high-performance lossless data compression applications with few or no modifications. The proposed hardware implementation targets on the compression with the compression ratio of 2:1 and 4:1.Future work is to design an appropriate hardware for the decompression unit of the above two architectures.

VI. Scope For Further Research

This work can be extended for any ratio and it is expected to retrieve the data with minimum error. The decompression architectures can be designed for cache memory. The architecture can also be modified to get a reduced area with good performance.

VII. Acknowlegments

The authors acknowledge the contributions of the students, faculty and Velalar College of Engineering and Technology, the Research Center and would like to particularly thank Bannari Amman Institute of Technology, Sathy for help in the design of test circuitry, and for tools support. The authors also thank the anonymous reviewers for their thoughtful comments that helped improve this paper**.** The authors would like to thank the anonymous reviewers for their constructive critique from which this paper greatly benefited.

References

[1]. A.Deepa, M.Nisha Angeline and C.N. Marimuthu, " P-Match: A Microprocessor Cache Compresion Algorithm", 2nd International Conference on Intelligent Information Systems and Management (IISM'11), July 14- 16, P.No.98, 2011.
[2]. Xi Chen, Lei Yang, Robert P. Dick (2010) "C-Pack: A High – performance Microprocessor Cache Compression Algorithm
[3]. A. Alameldeen and D. A. Wood, (2004) "Frequent pattern compression: A significance-based compression scheme for 12 caches," Dept. Comp. Scie. , Univ. Wisconsin-Madison, Tech. Rep. 1500.
[4]. E.G.Hallnor and S.K.Reinhardt, (2004) "A compressed memory hierarchy using an indirect index cache," in Proc. Workshop Memory Performance Issues, pp. 9–15.
[5]. J.-S. Lee et al., (1999) "Design and evaluation of a selective compressed memory system," in Proc. Int. Conf. Computer Design, pp. 1
[6]. J. L. Nunez and S. Jones, "Gbit/s lossless data compression hardware," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 11, no. 3, pp.499–510, Jun. 2003
[7]. P. Pujara and A. Aggarwal, (2005) "Restrictive compression techniques to increase level 1 cache capacity," in Proc. Int. Conf. Computer Design, pp. 327–333.