World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:2, 2007

Improving Cache Memory Utilization

Sami I. Serhan, and Hamed M. Abdel-Haq

Open Science Index, Computer and Information Engineering Vol:1, No:2, 2007 publications.waset.org/11933/pdf

***Abstract*—**In this paper, an efficient technique is proposed to manage the cache memory. The proposed technique introduces some modifications on the well-known set associative mapping technique. This modification requires a little alteration in the structure of the cache memory and on the way by which it can be referenced. The proposed alteration leads to increase the set size virtually and consequently to improve the performance and the utilization of the cache memory. The current mapping techniques have accomplished good results. In fact, there are still different cases in which cache memory lines are left empty and not used, whereas two or more processes overwrite the lines of each other, instead of using those empty lines. The proposed algorithm aims at finding an efficient way to deal with such problem.

***Keywords*—**Modified Set Associative Mapping, Locality of Reference, Miss Ratio, Hit Ratio, Cache Memory, Clustered Behavior, Index Address, Tag Field, Status Field, and Complement of Index Address.

I. Introduction

A cache memory is a high-speed, relatively small memory that represents a critical point in computer systems, since it eliminates the gab between a CPU and main memory speed. It has an access time smaller than that for main memory. So, the average access time is decreased in a great deal. Every time the CPU generates a reference for a specific word, the cache memory will be accessed to get that word if it is there; otherwise, the main memory is accessed to retrieve the line containing that word. Then, the line is stored in a vacant place in the cache memory. If no such vacant place is found in the cache memory for that line, a replacement algorithm (such as LRU) is used to store that line in a suitable cache memory location [14].

In fact, a cache memory is considered a good tool that makes use of locality of reference property that states the program references to the memory at any given interval of time tend to be confined within a few localized areas in the memory. In such cases, the cache memory will be referenced more frequently without needing to return to main memory [13].

When a cache memory is referenced to retrieve a certain word, there will be two possibilities: a hit that means the word is found in the cache memory or a miss if it is not found. The efficiency of a cache memory is usually measured using the hit ratio. A hit ratio equals to the number of hits divided by the total number of cache references [13] [14].

A cache memory is divided into blocks of words referred to usually as lines. A part of the address of the requested word (called tag) is stored together with the line. This facilitates and accelerates the process of accessing cache to find out whether a line already exists in it or not .This process is usually termed as cache interrogation [6]. In addition, each line (or set) in the cache is given an identifier, called index; the CPU address is accordingly divided into index and tag. Tags are used to distinguish between lines [15].

Mapping is the transformation of data from the main memory to the cache memory. Because the cache memory is smaller than the main memory, mapping is very important in that transformation, and there are three types of mapping: direct mapping, fully associative mapping and set associative mapping [14].

In direct mapping, the number of lines in the main memory is divided by the number of lines in the cache. The result is that every n main memory lines will compete for one cache line [15].

In a fully associative cache, any line in the main memory can be found any where in the cache so no restriction exists. Flexibility is maximized, and the cost is maximized as well, since the number of comparators needed is equal to the number of lines in the cache. Moreover, the miss ratio is minimized [13].

A set associative cache is an optimal approach of both, in which the cache is partitioned into sets. Each set contains the same number of lines where the number of comparators needed is equivalent to the number of lines per set. The hit ratio is less than that the case of the fully associative mapping, since the lines compete on sets although the line might be found any where in its set. The line size, set size and number of sets are cache parameters that affect the hit ratio [14].

In some cases, some processes overwrite each others in a cache memory when they reference addresses having the same index addresses and different tag fields, whereas, other sets remain idle and rarely referenced. References acting in this manner is said to behave in a clustered manner. This paper introduces a modification of the cache memory structure and logical treatment to gain benefits from those idle lines. Improving the hit ratio and reducing the memory average access time are also focused.

II. Previous Work

Different efforts have been carried out to improve the cache memory efficiency and utilization. Several literature have addressed the cache conflicts [2][11]. Agarwal and Pudar[3] suggested column associativity for improving direct-mapped caches. Seznec and Bodin[4] pioneered the work on skewed-associative caches. Some works on skewed caches are presented in some of their papers [8][9][6][12].

Mathias Spjuth, Martin Karlsson and Erik extended a skewed cache organization with a relocation strategy [1]. They achieved a miss ratio that is comparable to the miss

Authors are with Computer Science Department, University of Jordan, Jordan.

Open Science Index, Computer and Information Engineering Vol:1, No:2, 2007 publications.waset.org/11933/pdf

ratio of an 8-way set-associative cache, while consuming up to 48% less dynamic power. They concluded also that an optimal selective allocation algorithms based on knowledge about the future, can drastically increase the effectiveness of a cache. The effectiveness is further enhanced if the allocation candidates are temporarily held in a small staging cache before making the allocation decision. Spjuth[7] showed that by extending the elbow cache idea to n relocation steps, a further reduction in the miss ratio can be achieved.

Another research in cache modeling was done by Ramachandran [5], who suggested an ideal cache model. He also considered a class of caches, called random-hashed caches. In his work he mentioned: "We shall look at analytical tools to compute the expected conflict-miss overhead for random-hashed caches, with respect to fully associative LRU caches. Specifically, [5] achieved good upper bounds on the conflict-miss overhead for random-hashed set-associative caches". In [1] G. E. Suh, S. Devadas, and L. Rudolph introduced an analytical cache model for multiprocessing environments. They defined a model that takes the miss ratio curves for each process, the time quantum per process and the total cache size as input. The trace driven simulation using this model provided accurate results. Y. Yu, K. Beyls and E. D'Hollander in [10] introduced a cache visualizer that takes some cache snapshots and used them to study the behavior of the cache rather than constructing a cache model. This paper aims at improving the performance of the set associative mapping method.

III. Modification of the Set Associative Mapping Algorithm

The set associative mapping is an improvement over the direct mapping organization in that each word in the cache can store two or more words of memory under the same index address. Each data word is stored together with its tag and the number of tag data items in one word of cache forms a set. A modification on the set associative mapping algorithm is proposed in this paper to solve some problems arise in some circumstances as discussed below.

A. The Proposed Algorithm

The proposed algorithm aims at increasing the set size virtually, by allowing interleaving processes to make use of empty lines in cache and not to overwrite the cache lines by each other. Naturally, a strict mechanism should be found to retrieve the words stored in cache using that algorithm. So, a two-bit status field is associated with each word in the cache memory that acts as in Table I.

TABLE I
Specification of Status Field

Value	Meaning
00	Empty words
01	The word referenced using the index address
10	The word referenced using the complement of the index address
11	For future use

For each address generated by the CPU, the index and tag fields are calculated. Then, if one of the words in the set that the index address references has the same tag value as the calculated tag and the status field equals to (01), then we have a hit, otherwise, we take the seventh complement of the index address represented in octal and do the previous steps but with the status field equals to (10). If the word is found there, then we have a hit, else we have a miss and should bring the block that corresponds to the CPU-generated address and check if there is an empty line (status field = 00) in the set referenced by the index address or by its complement. If an empty line is found, we replace it with that block and set the status field to (01) if we use the index address itself or (10) if we use its complement. If no empty line is found, then a replacement algorithm is used to replace one of the words referenced using the index address or its complement with the block retrieved from main memory. The proposed algorithm is shown in Fig. 1.

```
T  : Tag
I  : Index Address
L  : Cache Memory Line
L.T   : Tag of a Line
comp.: the complement of
LRU  : Least Recently Used Replacement Algorithm

Input: a set of CPU-generated addresses, C (cache)
Output: Hit and Miss percentage
Hit = 0
Miss = 0

For each Address
  Found = 'false'
  Calculate I (index address)
  Calculate T (Tag)
  For each line in the set referenced by I (L)
    If L.T = T AND L.S=01 then
      Hit = Hit + 1
      Found = 'true'
      Exit for
    End if
  End

  If not Found then
    For each line in the set referenced by the comp. of I (L)
      If L.T = T AND L.S=10 then
        Hit = Hit + 1
        Found = 'true'
        Exit for
      End if
    End
  End if

// if the line was not found neither in normal place nor in the place
//obtained through the complement
   If not Found then
     Miss = Miss + 1
     Retrieve data from the main Memory
     Space = returnEmptySpaceIn(I)
     If space <> -1 then
          Put data in C (I), and set the Tag and status field to (01)
     Else
          Space = returnEmptySpaceIn(comp. I)
          If space <> -1 then
            Put data in C (comp. I), and set the Tag and status field to (10)
          Else
            Use the replace Algorithm (ex. LRU)
          End if
     End if
   End if
End
```

Fig. 1 The Modified set associative mapping algorithm

Open Science Index, Computer and Information Engineering Vol:1, No:2, 2007 publications.waset.org/11933/pdf

B. An Illustrative Example

Consider a main memory with 32K words of 12 bits each. The cache memory uses the proposed set associative mapping organization and can store 512 sets with two words for each set.

In Fig. 2, a part of that cache memory is illustrated. Four eight-word lines are brought from the main memory and stored in that cache with the tags (22, 54, 66, and 43). Each line is identified with its tag number (as line-tag_no.) for discussion purposes. The generated addresses are in octal representation.

In the initialization phase (when the computer is restarted), the status fields are set to (00), i.e. Empty lines. Assume that three processes, with the same index address, started (22110, 54110, and 66110)$_8$. Now, Notice the following processes cache memory allocation:

- The first process generates the address $(22110)_8$, and looks for a line with a tag $(22)_8$, but it finds the sets referenced by the index address $(110)_8$ and the sets referenced by its seventh complement $(667)_8$ are all empty (note that they are four empty lines, two referenced by the index address and two referenced by its complement), see Fig. 2. So, it decides to bring the line that corresponds to that address from the main memory and puts it in the cache (as in line-22).The status field is set to (01) because we use the index address not its complement.
- Next, the second process generates the address $(54110)_8$, and performs the previous operations. It decides to put the brought line in line-54 in the cache memory (note that there are three empty available lines in the cache memory referenced by the index address and its complement).
- Then, the third process generates the address $(66110)_8$, but note that the sets referenced by the index address $(110)_8$ are reserved. Here, the importance of the proposed algorithm appears, because the brought line does not replace the existing lines using a replacement algorithm, but it uses the empty sets referenced by the complement of the index address $(667)_8$ to store the line (so, no miss happens). Moreover, the status field should be set to (10) to help in retrieving it later on.
- The fourth process generates the address $(43660)_8$, and looks for an empty word referenced by the index address $(660)_8$. It finds such line, so it stores the line brought from the main memory in that empty line, and sets the status field to (01) because the index address is used not its complement.

Assume that a new process is started and generates an address with an index part equals to $(110)_8$. At this moment, it is known that the sets referenced by the index address and its complement have no empty lines. So, a replacement algorithm should be used to replace an old line with the new one. Here, the replacement algorithm will be applied on four lines (not on two as in the normal set associative mapping), so the percentage of replacing a line which will be referenced later will be reduced.

C. The Implementation Work

The object-based Visual Basic 6 programming language, Visual Studio Enterprise Edition, was used to implement the proposed set associative mapping algorithm. The main form contains three tabs. The first tab has options to decide the number of processes that is needed in the simulation process, to determine the address space of each process in main memory, and to find the empty spaces in it. The second tab displays the number of processes that access the modified set associative cache at a certain time, the block retrieved from the main memory in miss case, the hit ratio, the miss ratio, and graphical representation for hit and miss case. The last tab has the same features of the previous tab but related to the normal set associative cache.

An object-oriented approach is used in writing the code. The simulation application contains a class for processes, main memory, cache memory with different properties that could be updated for each class, such as, address space, block size, set size… etc. The code size is about 1100 lines. The experiments were run on an IBM compatible PC, running under windows XP professional operating system.

The system could be put in a clustering mode according to the entered address to each process. Then, "Start Processing" is issued and the result will appear in the next tabs. Note that, there is a timer that allows the processing state to be continued with switching between processes.

IV. Experiments

The experiments are split into two parts: the first one focuses on the cache which has a clustering behavior (in which each process overwrites the reserved space of each other). The second part is performed using a generated random number of space addresses for each process to reflect the real behavior. Here, when the processes allocate locations in the cache memory with different indices, the hit ratio will not be affected even if the number of processes is changed.

address	Status Field	Tag Field	Content Field	Status Field	Tag Field	Content Field
110	01	22	6521	01	54	2213
111	01	22	5534	01	54	4413
112	01	22	3333	01	54	7777
113	01	22	6547	01	54	6354
114	01	22	6771	01	54	6547
115	01	22	5631	01	54	1110
116	01	22	5555	01	54	4420
117	01	22	7777	01	54	6214
⋮	⋮	⋮	⋮	⋮	⋮	⋮
660	10	66	2213	01	43	6663
661	10	66	6354	01	43	6771
662	10	66	1110	01	43	5555
663	10	66	4413	01	43	6521
664	10	66	6214	01	43	3333
665	10	66	6663	01	43	4420
666	10	66	4413	01	43	6547
667	10	66	5534	01	43	5631

Figure 2: a part of the modified cache memory. Notice that a 2-bit status field was added to each word in that cache, and the addresses (667,666, ...) are the ones complement of the addresses (110,111, ...) respectively.

A. Part One

In this part, seven processes, which have the same index address (that is used to a reference set in the cache memory), are used. The simulation process was done using different

Open Science Index, Computer and Information Engineering Vol:1, No:2, 2007 publications.waset.org/11933/pdf

numbers of those processes with different set sizes as discussed below.

Experiment 1:

In Fig. 3, the set size equals to one, and the simulation is done on different numbers of processes (from 1 to 7).

Note here that the set size equals one in the normal cache, so it is a direct mapping cache memory. Using the proposed algorithm it appears like 2-word set-associative normal cache, because another location, obtained by the complement of the index address, will be used. The average improvement in hit ratio is 2.83%.

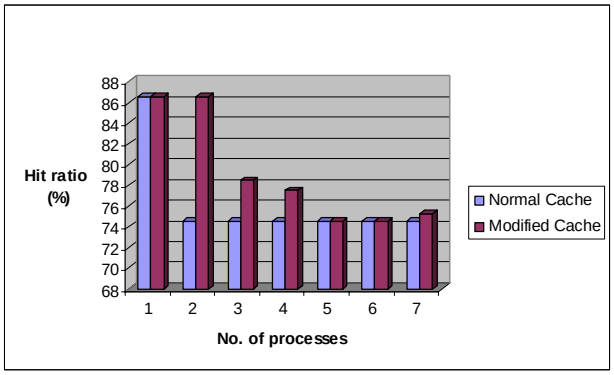


Fig. 3 The number of processes versus hit ratio. (Set size = 1)

Experiment 2:

According to this experiment, the set size is considered 2 (i.e., each process will compete on two words using the normal cache memory, while in the modified algorithm, each process will compete on four words). So, the number of misses will be decreased (see Fig. 4).

The average improvement in hit ratio here is 4.02%. Note also that the modified cache memory with the set size equals to one acts the same as the normal cache in this experiment (set size equals two). This is due to the proposed algorithm which tries to duplicate the set size of the cache memory.

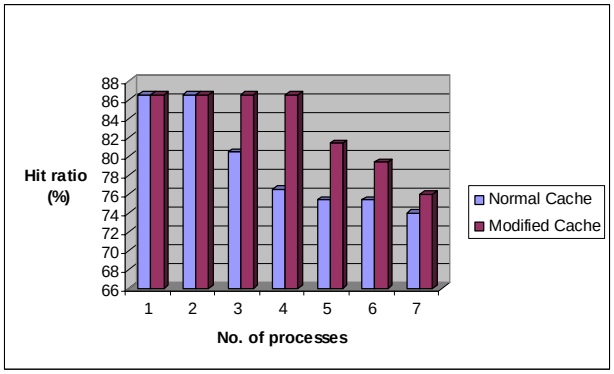


Fig. 4 The number of processes versus hit ratio. (Set size = 2)

Experiment 3:

Finally, in this experiment, the cache set size is considered with 3 words. The efficiency on the modified cache memory didn't decrease despite of the increment of processes in clustering behavior up to six processes as shown in Fig. 5. This is because the set size here is 3 and it appears as if it is of a size equals six (virtually duplicated) in the proposed approach. The average improvement in hit ratio is 4.62%

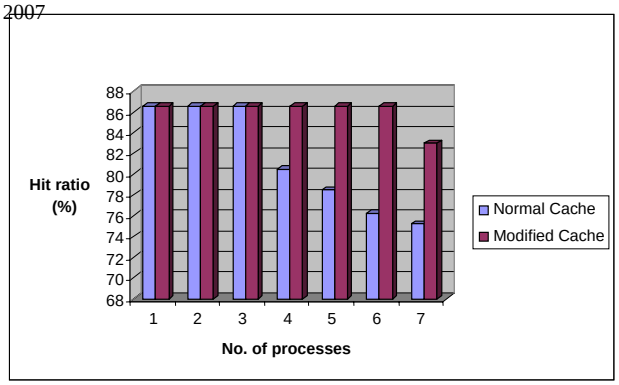


Fig. 5 The number of processes versus hit ratio. (Set size = 3)

B. Part Two

This part contains three experiments with different set sizes and the same number of processes.

Experiment 4:

This experiment uses three processes with randomly chosen address spaces and the experiment is done on different set sizes as shown in Fig. 6. The average improvement in hit the ratio is 3.03%.

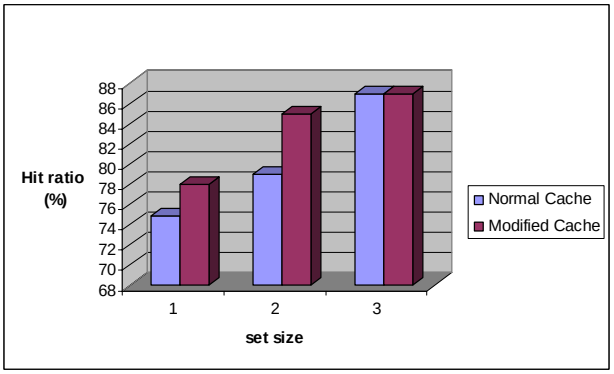


Fig. 6 Set size versus hit ratio. (Number of processes = 3)

Experiment 5:

In Fig. 7, four processes with random address spaces are chosen with different set sizes. The average improvement in the hit ratio is 3.56%.

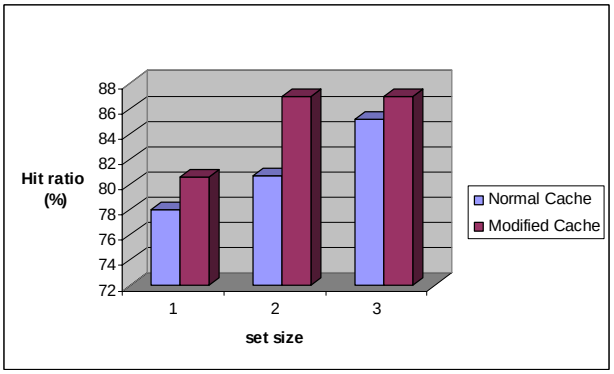


Fig. 7 Set size versus hit ratio. (Number of processes = 4)

Experiment 6:

In this experiment, five processes with different address spaces running on different set sizes (see Fig. 8). The average improvement in the hit ratio is 1.45%

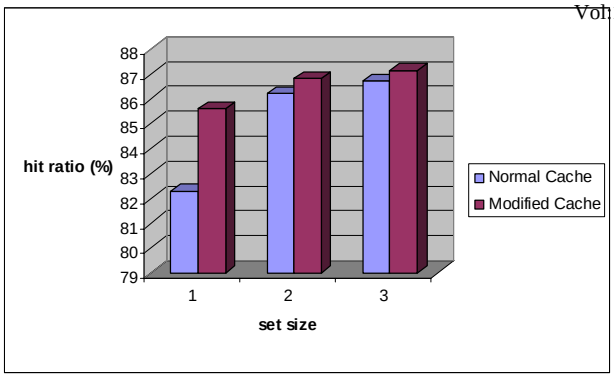


Fig. 8 Set size versus hit ratio. (Number of processes =5)

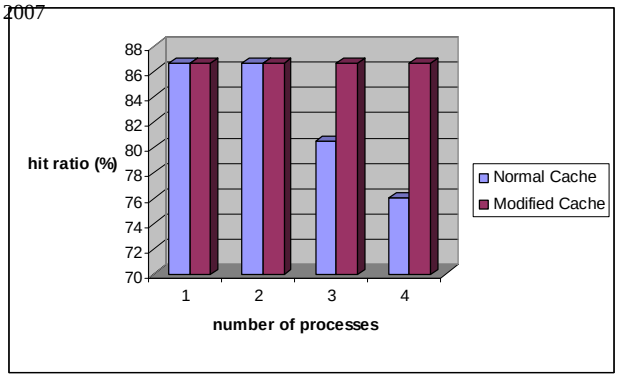


Fig. 19 Number of processes versus hit ratio in repetitive structure. (Set size = 2)

V. Experiments for Special Cases

Experiments were done on the following two special cases:

- The first experiment represents a special case in which the CPU switches among processes quickly, due to the excessive number of I/O interrupts. So, the number of applying the replacement algorithm is increased, and the miss ratio is high if the set size is not large enough. Here, the proposed algorithm expanded the set size virtually. So, the quick switching among processes did not affect the hit ratio. The normal set associative mapping became inefficient because of that case. The average improvement in the hit ratio is 33.15%

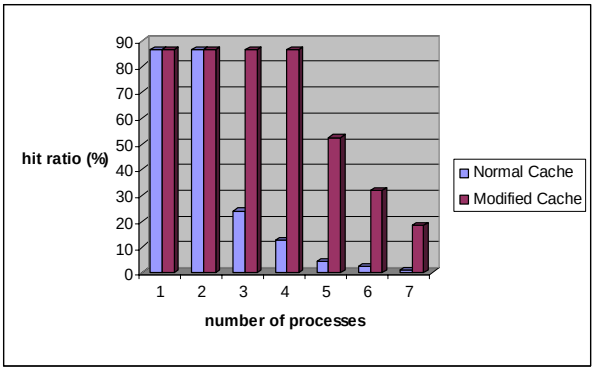


Fig. 9 Number of processes versus hit ratio. (Set size = 2)

- The second experiment illustrates the effect of the proposed set associative mapping algorithm on the repetitive programming structures, in which several instructions are executed many times (implemented using jump instructions). Imagine the case in which two or more processes with address spaces having relatively the same index address. If they are allowed to be executed simultaneously (and that is possible) using the normal set associative mapping, the overhead resulting from replacing the lines of each other will be huge. So, using the proposed algorithm, each process will work on separate lines in the cache memory and that will avoid the clustering behavior. As shown in Fig. 10, four tests have been accomplished using different numbers of processes with the same set size (set size equals two). Each of the used processes has a repetitive structure with the same index address. The average improvement in the hit ratio is 4.175%.

By observing the above graphs, you can see that the hit ratio values in part two are different than in part one for the same set size and the same number of processes. This is because the address space in part one was selected to reflect the interleaving behavior of processes, while in part two it was selected randomly.

VI. Conclusion

In this paper, a modification on a set associative mapping algorithm was proposed. It tries to duplicate the set size of the cache memory indirectly or virtually. In other words, instead of leaving the sets referenced by the complement of the index address empty and not used, we could consider them as parts of the sets referenced by the index address itself. It is right that we couldn't guarantee that we will find the other side set (set referenced by the complement of index address) empty. But we could guarantee that the possibility of replacing a block that will be used in the next time will be decreased because the number of words that will be put under the replacement algorithm will be duplicated. It also prevents interleaving processes from overwriting each other, but it makes use of empty lines in the cache memory. Space utilization has been improved, since the cache memory size is not increased physically by considering some empty lines of the cache memory as a part of the referenced set. The proposed modified algorithm has improved the hit ratio by a value up to 10 % in average.

References

[1] Mathias Spjuth, Martin Karlsson and Erik, "Cache Memory Design Trade-offs for Current and Emerging Workloads". Licentiate Thesis 2003-009, Department of Information Technology, Uppsala University, September 2003.

[2] M.D. Hill. "Aspects of cache Memory and Instruction Buffer Performance",A PhD thesis presented to the University of California, Berkeley 1987.

[3] A. Agarwal and S.D. Puadar, "Column-Associative Caches: A Technique for Reducing the Miss Ratio of Direct-Mapped Caches", In Proceedings of the 20th International Symposium on Computer Architecture, UK, pages 179-190, May 1993.

[4] A.Seznec and F. Bodin, "Skewed-associative caches", In Proceedings of PARLE '93, Munich, pages 305-316, June 1993.

[5] Ramachandran S.1999, An algorithmic theory of caches http://216.239.51.100/search?q=cache:KmPHLKLvJlkC:supertech.lcs.mit.edu/~sridhar/thesis.ps+%22cache+model%22&hl=en&ie=UTF-8.

[6] A.Seznec, "A New Case for Skewed-Associativity", A technical Report No. 1114, IRISA-INRIA, Campus de Beaulieu, July 1997.

[7] M. Spjuth, "Refinement and Evaluation of the Elbow Cache", Master's thesis, School of Engineering, Uppsala University, Sweden, April 2002.

[8] L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: A scalable

Open Science Index, Computer and Information Engineering Vol:1, No:2, 2007 publications.waset.org/11933/pdf

Architecture Based on Single-Chip Multiprocessing", In Proceeding of the 27th Annual ISCA, USA, pages 149-160, 2000.

[9] A. Seznec, "A Case for Two-way Skewed Associative Caches", In Proceedings of the 20th International Symposium on Computer Architecture, USA , pages 169-178, May 1993.

[10] Yu Y. , K. Beyls and E. H. D'Hollander , "Visualizing the Impact of the Cache on Program Execution", http://citeseer.nj.nec.com/502364.html, 2001.

[11] Nagel P. Topham and Antoio GonZalez, "Randomized Cache Placement for Eliminating Conflicts", IEEE Transactions on Computers, Vol. 48, No.2, pages 185-192, 1999.

[12] Hans Vandierendonck and Koen De Bosschere, "Trade-offs for Skewed- Associative Caches", Proceedings of the International Conference in Parallel Computing (PARCO), Germany, pages 467-474, September 2003.

[13] William Stallings, Computer Organization and Architecture, seventh edition, Prentice Hall, 2006.

[14] Barry Wilkinson, Computer Architecture, second edition, Prentice Hall, 1996.

[15] Linda Null and Julia Lobur, The essentials of Computer Organization and Architecture, second edition, Jones and Bartlett, 2006.

Open Science Index, Computer and Information Engineering Vol:1, No:2, 2007 publications.waset.org/11933/pdf