A novel method of dynamic permanent caching with resourceful built up and imperative access

Ijaz Ali Shoukat
Department of Computer Science
College of Computer & Information Sciences
King Saud University, Riyadh, KSA
ishoukat@ksu.edu.sa

Mohsin Iftikhar
Department of Computer Science
College of Computer & Information Sciences
King Saud University, Riyadh, KSA
miftikhar@ksu.edu.sa

Abstract **–** Caching is built up each time when the machine starts up according to user's application usability. This built up procedure escalates the efficiency of application's usage for the next coming access of same application. This cache history is vanished when power is switched off. Most of the time, a user uses the same common applications in his/her daily routine, particularly during the working hours. Moreover, against every new start up, the user bears the penalty of cache rebuilt to achieve better and efficient access because on first time access, the application is opened without caching which results more time to open rather than the second time after the cache has already built up.

This paper proposes a novel idea of intelligent and permanent caching which can build up dynamically and can be stored permanently in one part of cache chip according to user's application usability. This paper describes an algorithm that how CPU can build a dynamic and intelligent Electrically Editable Permanent Cache (EEPC) according to the probability of user's application usage in a computer machine. For proposed EEPC, we implement a *Probability Calculation Table* (PCT) by reusing available *compression techniques* and through introducing some basic change in cache storage policy. On the other hand L1 and L2 implementation is also the part of this study; for example, *History Table* (HT) is maintained for L1 and L1 is physically addressed to the main memory. Whereas, L2 is virtually addressed with the main memory and fully associative to handle misses. We mainly focus on Cache design enhancement, improvement in cache speed and implementation issues.

Key Words **-** PCT, HT, EEPC, Compression, L1, L2, Graphically Proposed Architecture, Intelligent Permanent Cache Storage Algorithm & Technique

I. INTRODUCTION

Cache memory is efficient and tiny chip resides between main memory and CPU or within the CPU of computer. There are two major types of cache memory

- L1 Cache
- L2 Cache

But in some new processors L3 cache is also exits. Where L1 cache is small SRAM which is the part of the processor and fast enough. L2 cache is additional kind of cache memory in order to increase cache capacity and to overcome the issues of address translation. L2 exits between L1 and main memory. Here some basic cache terminologies are described below which must be known for full understating of caching discussion.

A. Cache Hit: Cache hit is the requested action from cache memory to get desire contents (address, index, tag, and data) without reading the main memory [1].

B. Cache Miss

If we hit the cache to get our desire contents but the desire contents (address, index, tag, data) are not available in the cache and as a result desire contents are taken from main memory this is called cache miss [1].

C. Write Through Cache

If we write the desire contents from cache by writing them into main memory at the same time in such a way desire contents reside in both (cache and RAM) [2].

D. Write Back / Copy Back

In case of successful cache hit desire contents are only written to cache memory while in case of cache miss the desire contents are written into main memory [2].

E. Cache Memory Mapping / Types

The construction of cache memory depends upon the mapping action which helps to take recently used or requested data from RAM to the processor for performing some necessary actions. Cache chip holds the most recently used contents (Address or both address and data) in it so that there may be a chance, the processor requires same data again that it has used earlier. Different approaches were used to map the data from main memory to the cache. There are different types of cache memory are available. Some of them are as follows [3].

a. Fully Associative Mapping [3]

By taking both, data and full address from Random Access Memory (RAM), the contents are stored simply in cache memory, this approach is said to be fully associative mapping. Each cache location and incoming memory address is simultaneously compared in order to find the requested data, if compare operation is successful then it is called *'hit'* and the desire contents are transferred to the processor for performing necessary operations. In case of un-successful hit the request is forward to RAM for purpose of getting data. When data is found then a copy is saved in cache before providing it to the processor. This procedure of un-successful hit is known as ' *Cache Miss'*. To store the un-necessary complete address with its data is space consuming as cache memory is small in size. On the other hand its main advantage is that no transformation of address

algorithm is required for manipulation of addresses because it stores complete address of data.

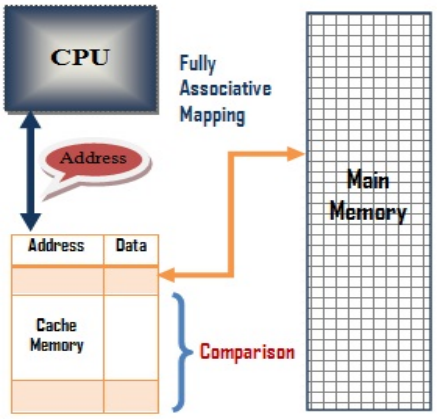

Fig. 1 Fully associative mapping

b. Direct Mapping [3]

The cache memory location, which we want to use, is directly addressed via (physical or virtual address) then it is called direct cache mapping. In this type of memory contains built in "cache replacement policy" on the available cache line. Each location of the cache memory holds contents at specified location address based on lower half bits of an address of RAM. The remaining higher bits are stored with the data in the cache memory to complete the identification of cached data. Direct mapping follows the mapping scenario in such a way, every address (CPU or RAM) is partitioned into two designated parts (1) index made up by lower significant bits and (2) tag made up by higher significant bits.

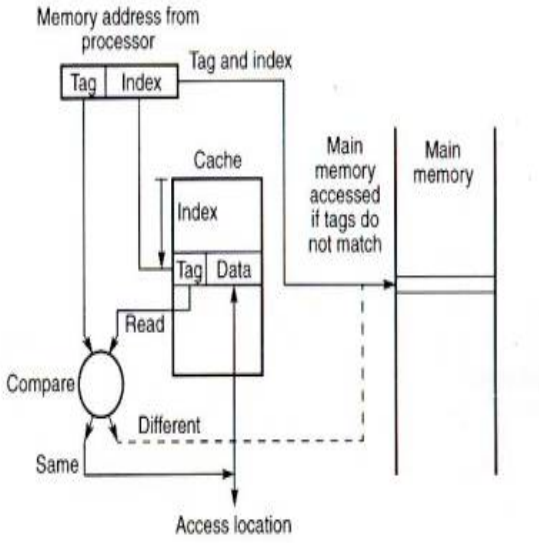

Fig. 2 Direct mapping

This type of memory is simple and efficient because it can have various indexes in cache chip at the same time as same indices are reside on the same location.

c. Set Associative Mapping [3]

Various kind of direct mapped cache is the base for the development of Set Associative Mapping, therefore, each direct mapped cache behaves like a set. Its development follows the rule; two direct mapped caches are needed for two ways set associative cache. A committee of blocks is consists of same index and variable tags in cache chip. As a full address, most significant bits of RAM are marked where the block address is the next coming least significant bits for the set in a block. This address mapping scheme is regarded as bit selection as shown in the Fig. 3 below.

0 1 2 3 4…	… 25783 …	… 10 38….
Sets	Block	Tag

Fig. 3 Memory address division

This type of memory has the ability to reduce thrashing but thrashing cannot be completely handled due to algorithm's own functions.

d. Sector Mapping [3]

The process of dividing the RAM and Cache into logical sectors is called sector mapping. Each sector consists of more than one block but RAM to Cache mapping is done via one to one map rule. It follows the working functionality of Fully Associative Mapping in such a way a tag resides in cache chip to located the RAM address. Its diagram is shown in Fig.4 below.

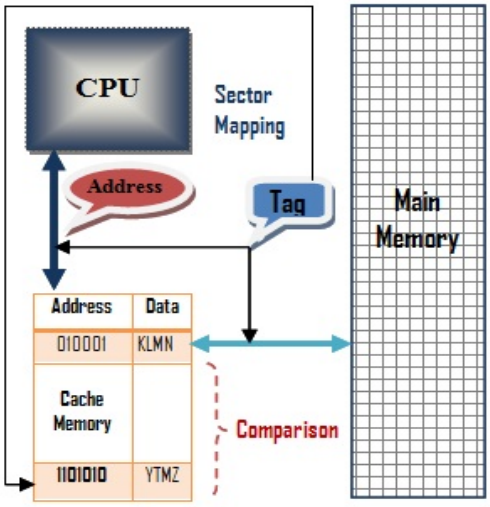

Fig. 4 Sector Mapping

II. IMPLEMENTATION ISSUES OF CACHE

The speed of main memory and CPU has a valuable gap as processor is fast more and main memory is slow more, therefore there should be criteria in which both are synchronized with respect to their speed in order to get optimized performance. This need caused the presence of cache memory. Fast accessing of main memory is made possible with the use of cache memory in such a way the most recently used application's addresses are stored in cache and at the time of need these addresses are firstly located in the cache. if the address is present in the cache it is selected quickly and data is accessed efficiently. But if the address is not located in the cache then firstly these are located in the memory and then data accessing procedure will start therefore, as a result, time is lost. It means cache should be used and it must be efficient and large to little more in capacity with limited acceptable cost. Where the cache overcame the speed inconsistency there is at the same time it generates some issues of implementations in terms of access and address translation procedure.

According to author of research paper referenced at [4], access delay is due to the access of address tag and address mapping. But actually, the delay is some more rather than the above estimation because another kind of delay i.e. to access data from memory or storage is also there. Other kind of fast memory named as cache memory is associated with the CPU in the form of two dimensional arrays. The first dimension is the set of steps used to access address tag and other is to perform address mapping. The ability of cache memory is estimated with the following factors

1. Cache hit time.
2. Cache miss rate and miss penalty.

Because while accessing or hitting the cache may address translation is need but it depends upon the architectural design of processor and cache memory. Some Arithmetic Logic Unit (ALU) design based on real physical address generating capability and some generates virtual address, Therefore in case of real address there is no need of address mapping but in case of virtual address, address mapping is needed. Virtual address generation has two problems:-

1. Some time is wasted for translation procedure that caused the lack of efficiency.

2. "The most serious drawback of the virtual address cache is that multiple virtual addresses may be mapped to the same physical address, i.e. synonyms may occur "[4].

Synonyms problem can be controlled as follows. Either synonyms should not be occurred or index bit that is used to select the cache should be same for both virtual and physical address [4] [5] [6].

Similarly, cache misses cause some extra delay in cache performance. Following are the possible misses in cache.

A. Compulsory Miss

Very first cache block reference generates a compulsory miss.

B. Capacity Miss

When block misses in write to the cache due to cache low capacity rather than the block is called capacity miss.

C. Conflict Miss

When block removes or overwrites into cache due to the mapping of other block into the same set. The necessary miss cannot be removed or minimized but the other two misses can be minimized to make the cache performance more efficient. The capacity misses can be avoided by using I/O approach [7].

D. Measuring the Cache performance Mathematically

The general mathematical cache memory performance formula becomes as follows

ACAT = Cache hit_time + Cache(Miss_rate x Miss_penalty) [8]

Where,

ACAT= Average Cache Access Time

Hit_time takes about 0.25 – 1.0 ns. i.e. 1 clock cycle

Miss_penalty takes about 75 – 100 clock cycles

Miss_rate is cache dependent!

III. LITERATURE REVIEW

From 1989 to 2001 research focuses on cache performance with different approaches in order to get optimized efficiency of cache with affordable cost. But every time when any attempt is made for this purpose, it results some other delays or disadvantages. Therefore, efficiency and disadvantages are compensated and optimized cache efficient could not be achieved at optimized level. Here some comparison of adopted strategies is represented with some other parameters in tabular form below. We made this table in the light of various research paper and slides.

A. Research implementation comparison [4] [8] [9] [10]

TABLE 1. Implementation Comparison

Idea Parameters	Strategy adopted	Advantages	Disadvantages	Cost
Larger cache size	Introduce L2 & L3 cache	Fewer capacity misses Miss penalty decrease	Longer access time Efficiency is lacked	High
Grater cache associativity		Fewer conflict misses decrease miss rates	Longer access time Complexity increase Efficiency is lacked	consistent
Physical Address cache		No" multiple" address mapping fear	Address translation delay involved	medium
Virtual Address cache		Delay decrease in successful condition, Absence of address does not generate miss problems because it searched under other virtual address.	Synonyms problem (multiple virtual address mapping with physical address)	
Write-through policy (for cache only)		No write-back time penalty, easier write-miss handling	Wasted memory bandwidth, Longer access time Efficiency is lacked	Consistent
Reduce Miss rate	Larger Block size, Higher associatively, compiler optimization	Miss penalty is minimized Less Access time	Complexity increase	medium
Reduce Miss Penalty	Multi level cache Critical word first and early restart	Less access time	Miss penalty increased	high
Reduce Hit Time	Small and simple cache Avoid address Translation	Faster access	Low Capacity	low
L2 Compression [8]	Frequent Pattern Compression Algorithm [11]	Cache capacity increased 29 – 75% L2 Miss ration reduced by 9 -24% Overall 17% speed is up	Due to decompression L2 access latency increased	Medium
Parallelism	pipelining	Improve access time	Dependency and conflicts generation	Medium

B. Comparison of Cache Mapping Techniques [12] [13] [13]

TABLE 2. Cache Mapping Comparison

Cache Types	Hit Ratio	Searching	Advantages / Disadvantages
Direct Mapped	*****	*******	Fast Simple One to one address map Maximum misses conflicts
Fully Associative	*******	****	No address translation Any sector can map any where No conflict misses Expensive More space is required Searching is slow
N-Way Set Associative, N>1	********** Good when n increase	***** Worse as N Increases	Compromised solution Conflict misses are greatly reduced.

Hence, the above comparison Table - 2 shows that the set associative mapping techniques is better comprised way to handle the clash of conflict misses as well as the gap between CPU and memory speed. The problem with set associative mapping is the worst case of search from large no. of increasing data entries; therefore, for searching point of view direct mapping is more suited. But each mapping techniques has its own value.

C. Available cache in different processors

The different cache sizes which are implemented according to processor's manufactures are shown in a Table -3 [14] [15].

TABLE 3

Processor	L1, L2, L3 size
Ultra SPARC IV	64KB-D, 32KB-I, 16MB
PA-RISC 8800	1.5MB, 32MB, NA
Opteron	128KB, 1MB, NA
Xeon	16KB, 512KB, 4MB
Power5	64KB, 1.5MB, 36MB
Itanium2	32KB, 256KB, 6-9MB
Athlon XP	128 KB, 256 KB
Athlon XP+	128 KB, 512 KB
Pentium 4 (I)	20 KB, 256 KB
Pentium 4 (II, "Northwood")	20 KB, 512 KB
Athlon 64	128 KB ,512 KB
Athlon 64 FX	128 KB , 1024 KB
Pentium 4 (III, "Prescott")	28 KB, 1024 KB

Above Table 3 contains collectively all three caches without mentioning, which cache is on the chip and which of them is off the CPU chip.

Now we analyze the L1 and L2 cache size ***"on the Chip"***

TABLE 4 [16]

Processor	Date of Introduction	On-chip Cache Memory	
		L1	L2
8086	1978	--	--
80286	1982	--	--
80386	1985	--	--
80486	1989	8K B	--
Pentium	1993	16K B	--
Pentium Pro	1995	16K B	256 / 512 KB
Pentium II	1997	32K B	256 / 512 KB
Pentium III	1999	32K B	512 KB

Moreover, we analyze the L2 cache place in different Processor Models in a table- 5 as described below **[15]**

TABLE 5.

CPU	L2 Place
Pentium, K5, K6	External, on the Motherboard
Pentium Pro	Internal, in the CPU
Pentium II, Athlon	External, in a module close to the CPU
Celeron (1st generation)	None
Celeron (later gen.), Pentium III, Athlon XP, Duron, Pentium 4	Internal, in the CPU

D. Problem and need

The summary of research from1989 to 2010 follows the cache memory research which revolves around the above mention parameters as shown in the first column of the comparison Table - 1. So in a single statement, we can say that the research of cache memory is done to improve speed and capacity with affordable cost. But this paper introduces some other ideas as follows:-

- Not too much costly.
- Suggested size is 20 KB to 128 KB.

1. Still no research idea is put forward for storing and modifying the desire contents permanently into the cache chip based on probability calculation table (PCT by introducing some basic change in the implementation of cache architecture.
2. Similarly, there is no idea also put forward to store the cache contents dynamically in expert way according the situation and behavior of user with respect to the use of application. Recently just the idea of access cache is introduced and implemented in the HP-PA7200 that manage the cache by using FIFO buffers based on observation and exits in fully associative L1 cache off side the CPU chip [17]. The problem in this idea is delay which requires every time to make the FIFO buffer when the system will start.

Therefore, there is need to propose some new cache implementation architecture which can introduce intelligent factor in caching to overcome the cache implementation issues as described in the compression Table 1.

IV. METHODOLOGY OF PROPOSED CACHE IMPLEMENTATION

We propose a little change in cache architecture as well as also some basic change in cache filling scheme according the probability of usages of application on computer machine. The probability can be calculated statistically by seeing the most no. of uses of any application by giving them some numeric values which can be stored in proposed cache named as Electrically Editable Permanent Cache (EEPC). The probability calculation table (PCT) will be able to change dynamically time to time and will be overwritten according to the highest probability of application use. In case of managing PCT only CPU utilization is increased to some extent which is negligible because CPU is much faster and mostly remains free into measurable extent. This EEP cache will contains the following characteristics,

- Will be the part of the CPU and directly coordinate with it.
- Will be able to store mostly used application's addresses permanently on the base of PCT.
- Will be able to change or over write according to PCT like Electronically Editable Programmable Read only Memory (EEPROM).
- It will directly coordinate with RAM and it will be able to store desire address in full format to save address translation time.
- Faster than RAM due small size. Even it will respond efficiently than L1 and L2 because L1 and L2 require one step cost more than EEPC.

CPU also contains L1 cache (128 KB to 1024 KB) on chip and L2 (2 MB to desire MB) cache off chip. Where L1 is direct mapped with main memory for faster access and manage History Table (HT) of address according to its capacity. L1 cache will be compressed by using Frequent Pattern Compression Algorithm [11] and the other comparison technique i.e. "Dictionary base compression algorithm" as discussed by the study [18]. On the other hand the entries that exit in EEP Cache can not be duplicated in L1 cache in order to save the cache memory space and conflicts. **L2** cache will contain the following characteristics:-

- Larger Block size.
- Higher associatively.
- Virtually address with main memory.
- L2 cache will be compressed.

A. Propose Cache Architecture with CPU

Here the proposed cache framework is described graphically with CPU and Main Memory as shown in Fig 5. This proposed model is able to manage cache in intelligent manner to some extent as shown in Fig. below. Major change which we introduced in this model is the EEPC which can maximize the performance (high speed, instant access). L1 and L2 idea is the same as previously implemented in currently available computer machines with only a little change of mapping and compression techniques in order to optimize the overall performance of caching. Moreover we introduce a new heuristic of storing and managing history table in L1 cache which actually provides base for building up the EEPC in expert way according to individual user's probability of application usability. This probability is managed by a probability table in EEPC.

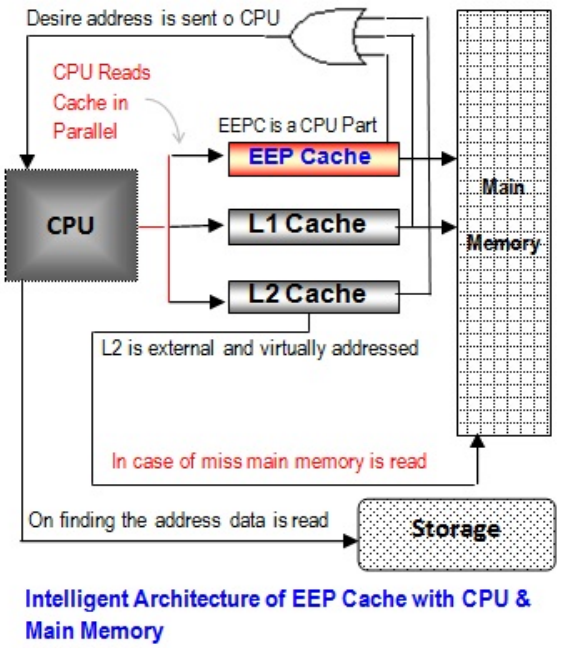

Fig. 5. EEPC Architecture

B. Working Methodology Algorithm

Begin

{

Req_Add as binary Signal,
PCT as binary object
HT as binary object

Create or overwrite and compress PCT Buffer by calculating the occurrence of application use and store permanently in EEPC with highest value until EEP cache is not full. In overwrite case least value will be overwrite

If EEPC size is full then manage HT runtime with some low probability value without duplication. HT is write back when any entry is removed from EEPC ' s PCT buffer.

On Request of Req_Add

CPU reads EEPC, L1, L2

If Req_Add is found in EEPC
{
Return to CPU
Block L1, L2 request action
}

Else If Req_Add is not found in EEPC OR found in L1

{

Block EEPC & L2 request action

}

Else if Req_Add is not found in EEPC AND L2 OR found in L2
{
Return to CPU
Block EEPC & L1 request action
}

Else //when complete Miss Occurs
{
Read Req_Add from Main Memory and fill the L2 and also write HT if space is available.
}

End if
End if
End if
End if
}
END

V. IMPLEMENTATION RESULTS

TABLE 6. Implementation Results

Factors	EEPC Role	L1 Role	L2 Role
Very first hit on system start up after shut down.	Most probably succeeded Performance **Cost = 3** CPU →EEPC→RAM	Fail	Fail
2nd hit when caching will have been built up.	In case of successful hit its access is faster than L1 and L2 Performance **Cost = 3** CPU →EEPC→RAM	No need of L1 and time will be saved	No need of L2 and time will be saved
In case of EEPC miss	Fail L1 most probably will be used and compensate it. So in case of Fail cost be the same as L1	If L1 is successful Performance **Cost=4** CPU →L1→Address Translation→RAM	If L1 Fail which is most rare case for a new kind of application Performance **Cost= 5** CPU→L2→HDD→Address Translation→RAM
Searching	Fast as small size and no address translation due to full address storage.	Medium due to large size as compare to EEPC and address translation.	Low
CPU over hedge	Low	Medium	High
Table Managing cost	PCT cost is Negligible due to small size and fast speed of CPU.	Little more than normal L1 for HT	High cost as there is need to update both HT and PCT as well as EEPC writing.
Overall performance	Best and its performance cost is consistent.	Better	Good

The above results claim that the implementation of EEPC can decrease the access cost of cache hits and search. Furthermore, it can provide the facility of caching on early restart of computer machine where the other caching techniques are failed. Hence, the use EEPC is able to improve overall cache performance as described in Table 6.

VI. CONCLUSION

We have shown that the basic change in the cache storage can improve the cache performance, for example, a *History Table* (HT) is maintained for L1 and it is physically addressed to the main memory, whereas L2 is virtually addressed with the main memory and fully associative to handle misses. The cache is partitioned into three parts, (1) EEPC is the first part, which stores mostly used user's applications permanently; (2) L1 is the second part, which stores history of user's application results, and (3) L2 is the third part, which behaves like normal cache as in current computer systems. The implementation of proposed cache memory architecture can save the very first penalty of user when user starts the machine and opens up any application. The permanent caching can make a faster and instant possible access against every regular application. The presented cache memory architecture along with the proposed algorithm is smart and efficient to a large extent because caching can be built up according to user's choice of applications which is handled with History Table and Probability Calculation Table. As a matter of fact, the proposed cache architecture will provide a new platform for the researchers and scientists to introduce intelligent factors in cache memory. By using our proposed architecture in new version of caches, the user experience can be enhanced with a fast and instant access. We have provided the framework to utilize the benefits of three kinds of cache mapping techniques at the same time. The implementation results (Table – 6) indicate that the proposed Electrically Editable Permanent Cache (EEPC) is an efficient, fast and more consistent as compared to L1 and L2. Collectively the suggested architecture of EEPC, L1 and L2 clearly seems to be an improved scheme for cache memory architecture.

REFERENCES

[1] P. Koopman, "Cache Organization", Memory System Architecture, 18-548/15-548, cited at http://www.ece.cmu.edu/~ece548/handouts/04cachor.pdf, visited on May 25, 2010, (1998).

[2] A. K. Christopher, Cache Coherence in Distributed Systems, Western Research Laboratory 100 Hamilton Avenue Palo Alto, California 94301 USA, (1987).

[3] V. *D. P.* Ruud, "Memory Hierarchy in Cache-Based Systems", Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95045 U.S.A, 650 960-1300, Part No. 817-0742-10, 11/20/02, Revision A Edition: November (2002).

[4] J. K.Peir., W. H. Windsor, A.J Smith, *"Implementation Issues in Modern Cache Memory"*, Computer Science Division (EECS), University of California, Berkeley, California 94720, (1998).

[5] K H. Inoue, Nonogaki, T. Urakawa, K. Shimizu., Plural virtual address space processing system," US Patent No. 4145738, March 20, (1979).

[6] S. Bederman, "Cache Management System Using Virtual and Real Tags in The Cache Directory," IBM Tech. Disc., 21(11), , pp. 4541, April (1979)

[7] G. Golan, "Summary for the seminar-Analysis of algorithms in hierarchical memory", (2004).

[8] Old Dominion University Slides – *"Memory Hierarchy Design"*

[9] B. Parham, *"Book- Computer Architecture"*, *Microprocessors to Supercomputers*, Oxford University Press, New York, (ISBN 0-19-515455-X), (2005).

[10] R. Alaa, Alameldeen and David. A, *"Adaptive Cache Compression for High Performance Processors"*, wood Computer Sciences Department, University of Wisconsin-Madison, (2004).

[11] L. Benini, D. Bruni, A. Macii and E. Macii, "Hardware Assisted Data Compression For Energy Minimization In Systems With Embedded Processors", Pages (449 -453), put forward in IEEE 2002 design Automation and test in Europe", (2002).

[12] P. Koopman, Associativey- Memory System Architecture, Pages(18-548/15-548), Cragon. (1998).

[13] C. M. Kozierok. (2001), The PC Guide, Site Version: 2.2.0 - Version Date: April 17, 2001 , cited at: (http://www.PCGuide.com).

[14] P. Ghosh, "ASE112*:* Adaptive Server Enterprise Performance Tuning on Next Generation Architecture presentation Slides". (2004).

[15] Chapter 11. The L2 cache cited at: http://www.karbosguide.com/books/pcarchitecture/chapter11.htm

[16] Chapter Four CPU Architecture, *"Art of Assembly Language"*, Cited at http://webster.cs.ucr.edu/AoA/Windows/HTML/CPUArchitecture.html#1013234 , visited on February 02, 2010.

[17] G. Kurpanek, K. Chan, J. Zheng, E. DeLano andW. Bryg, "PA7200: A PA-RISC Processor with Integrated High Performance MP Bus Interface", *COMPCON Digest of Papers*, pp. 375-382 , Feb 1994

[18] A. F. Briglia, A. Bezerra. (Nokia Institute of Technology) , L. Moiseichuk, (Nokia Multimedia, OSSO) and H. Gupta(VMware Inc.), "Evaluating effects of cache memory compression on embedded systems".

AUTHOR'S PROFILE

Ijaz Ali Shoukat
Research Scholar - Department of Computer Science, King Saud University, Riyadh, KSA.
M-Phil / MSCS – GC University, Lahore, Pakistan.
BSC.(Hons) – GC University, Faisalabad, Pakistan.
Member of **(IACSIT)** - International Association of Computer Science and Information Technology.

Mohsin Iftikhar, PhD
Assistant Professor - Department of Computer Science, King Saud University, Riyadh KSA.
PhD (CS) - School of IT, The University of Sydney, Australia.
M. Engg. Sc. (Telecommunications) - The University of New South Wales, Australia.
B.Sc Electrical Engg. University of Engineering and Technology, Lahore, Pakistan.