REDUNDANT CACHE DATA EVICTION IN A MULTI-CORE ENVIRONMENT

Muthukumar S[1] and P. K. Jawahar[2]
[1]Associate Prof., Deptt. of Computer Science and Engg., SVCE, Chennai, India
[2]Prof., Deptt. of Electronics and Communication Engg., BSA University, Chennai, India

ABSTRACT

Replacement algorithms designed for the multi-level cache found in Chip Multi Processors (CMP), in specific, heterogeneous multi-core processors, might result in data redundancy across the cache levels for certain data sets. The presence of redundant data not only uses up extra space but also results in performance degradation of the multi-core processor. Our paper takes up a review on a technique for the optimization of such processors for a better performance. At a fine-grained level we propose to redesign the replacement strategy adopted in various levels of cache to minimize redundancy which in turn will escalate the performance and efficiency of multi-core processors to greater heights. The techniques proposed apply directly to a single core and can also be adopted across all the available cores to boost up the effective memory utilization factor.

KEYWORDS: *HMP, Cache, Redundancy, LRU, MRU, replacement pair, miss rate.*

I. INTRODUCTION

Multi-core processors, in computer architecture have an edge over the others because they serve as an important tool in promoting parallelism thereby facilitating parallel execution which increases the degree of utility to a much greater extent. Heterogeneous Multi-core Processors (HMP) have proved to be the perfect choice over their homogeneous counterparts as they possess a unique advantage of stepping-up the system parameters – throughput, access time, efficiency of execution. These processors have been extensively used in mobile phone applications, graphics processing units (GPUs) etc. Though advantageous, these processors face many issues like data redundancy across various levels of cache, memory contention which tend to deter their overall performance.

Memory management plays a crucial role in determining the performance of such Multi-Core processors. Cache memory is widely used in HMPs to enhance the system performance. The most widely found architecture is that every core in a HMP environment has a private L1 cache and all the available cores share a relatively larger L2 cache (which in some cases can also be referred to as the shared Last Level Cache) [17]. The L1 cache can be further divided into 'Instruction Cache' and 'Data Cache'. Cache memories have been fine tuned over the years to improve the performance. Various methods for evaluating their performance parameters, such as DEFCAM [3], have been proposed, which bases its evaluation on fault tolerant models. When multiple copies of the same data reside across various levels of the cache, it not only consumes space but also results in considerable performance degradation.

In this paper, we report a comprehensive study of techniques to minimize cache conflicts and to improve the uniformity of cache accesses and reduce redundancy. The rest of the paper is organized as follows:

Section II – Related work describes about the various methods and techniques that will help boost performance when it comes to cache memory access. It also talks about the existing replacement strategies.

Section III – This section gives a brief overview of heterogeneous multi-core processors.

Section IV – Section IV describes how cache memory can have impact in the overall system performance.
Section V – In this section we present the proposed idea in a detailed manner.
Section VI – This section, which is the Results and Discussion section, substantiates the introduced idea by an example with proper reference to appropriate figures.
Section VII – This section summarises the paper.
Section VIII – Section VIII discusses about how the idea can be taken forward in the future to obtain better results.
Following this section, there is a list of captions for the figures that has been used in this paper to demonstrate the idea. The list of references follows this section and the actual figures forms the final section.

II. RELATED WORK

HMP can lead to high performance computing as discussed above, but in order to effectively utilize the entire power of HMP, applications must be properly scheduled to appropriate cores. Over utilization or under utilization might result, if accurate scheduling is not done. For example when a taxing application is assigned to a core which is already heavily loaded, it will result in lesser throughput (over utilization). Similarly if a very small task is scheduled to an idle core, it will also result in performance degradation (under utilization). Various methods have been adopted, which monitor applications' behavior to get a clear idea as to which core they needs to be assigned to, based on certain metrics like the miss rate of Last-Level Cache (LLC) etc [10,11].
Memory allocation plays a vital role in deciding the performance of HMP. When it comes to HMP, methods have been proposed by Hasitha Muthumala et al for judicious allocation of memory with respect to media processing applications [5], which are said to reduce the processing time considerably. Not all applications will have high hit rates. This is mainly because every application has its own data requirements. While one application might effectively utilize the data present in cache, another application may not. So there is no single method to bring uniformity in cache access for all applications. The scheme which is best suited for that particular scenario has to be adopted dynamically [5, 6]. Deviating from the traditional technique of having a shared L2 cache, Kakoee et al have come up with the concept of shared a L1 cache targeted mainly towards tightly coupled processor architectures [12]. The number of processors in this case is significantly high (16 or more). The multi banked cache architecture used here is proven to have improved the space overhead and processor performance by a considerable margin.
When there are many processor cores, cache line protection becomes an important issue. Huang Z et al have proposed a method [15] where protection of a cache line is based on its generation time rather than the CPU core ID to which it belongs. This is achieved using the live-time protected counter that decides whether a given cache line is 'dead' (stale) or 'alive' (up to date). Power consumption and energy efficiency [8, 14] are challenging issues in a multi-core environment. The paper by Fang Juan and Lei Ding discusses a cache reconfigurable method [14] to reduce the power consumption without compromising on the performance.
Lot of studies and researches has been conducted by various researchers in the paradigm of cache replacement. In general cache replacement refers to the process that takes place when the cache becomes full and certain existing objects must be replaced to make way for the new ones [17]. Cache replacement occurs right after encountering a miss. Numerous Replacement techniques have come up over the years. The most commonly used replacement algorithms are the LRU, MRU, LFU etc. A brief overview of each of them is given below. The paper by Khan S et al proposes a cache segmentation method [13], which can be adopted in the shared last level cache. The results have shown that this method has outperformed traditional algorithms like and LRU in 2MB and 8MB sized last level caches.

2.1. Least Recently Used (LRU):

LRU [17], as the name suggests, replaces the objects that have not been accessed for the longest period of time. This algorithm is quite simple to understand and it has been used extensively because of its

very good performance. The data structure used to implement this algorithm is basically a stack, where insertion and deletion takes place at opposite ends.

2.2. Least Frequently Used (LFU):

LFU [17] is similar to LRU but the selection of objects depends on the number of times the object has been accessed in the recent past (in other words its frequency of access). It replaces the objects with fewer access counts and keeps objects with higher access counts. This algorithm has a major drawback. The principle of locality of reference might be violated. A new object, which was recently added to the cache, has more probability of getting replaced during a cache miss.

2.3. Most Recently Used (MRU):

This algorithm contradicts LRU. This algorithm is used to deal with the case such as sequential scanning access pattern, where most of the recently accessed pages are not reused in the near future [17]. So it is better to replace the new objects in those scenarios rather than replacing the old ones which have more probability of being accessed in the future.

There are also other replacement algorithms apart from these commonly employed techniques. One such notable technique was proposed by Carole-Jean Wu and Margaret Martonosi. It is the Adaptive Timekeeping Replacement (ATR) algorithm [4], which tracks the time interval since last access to a cached data, taking into consideration the application priorities and memory characteristics and makes the replacement based on the information that has been gathered. It has been proven to have improved the performance of parallel workloads in a multi-core environment. Obviously any given algorithm cannot be efficient in all possible ways. Every algorithm has its own drawbacks.

Data redundancy in cache has been proven to have improved miss penalty if manoeuvred diligently [2]. Marios Klenthous and Yiannakis Sazeides have demonstrated this concept. When a reference is made to a block and if there is a miss encountered in the cache, the content of the block might be present in the same level under a different tag, which the cache is generally unaware of. Generally this duplication is undesirable. But it can be utilized wisely by the cache by looking-up for the data in the same level instead of searching the lower levels in case of a miss, thus reducing the miss penalty. In general, Data duplication leads to wastage of memory which will rule out the possibility of accommodating different data. So if the available redundancy is not made use of in a judicious fashion (as specified above), it will result in significant degradation of the system performance over a period of time.

III. HETEROGENEOUS MULTI-CORE PROCESSORS – OVERVIEW

3.1. Heterogeneous Processor Cores:

The HMP integrates various types of cores with different complexities into a single chip, and facilitates higher utility by addressing issues like throughput and efficiency for various workloads by relating the application's needs with the resources available for execution. Heterogeneity in processors can be exploited effectively to enhance performance and to make it energy efficient [8]. Parallelization is an important advantage in an HMP. Workloads are split between the cores which are then executed in parallel to boost up the throughput [1]. In a heterogeneous multi-core environment, the execution time of a software task depends on the processor core it is executed upon. For example, a software task performing computer graphics such as simulating physics, runs much faster on a graphics processor than on a normal processor. Stated alternatively, a software task with many branches and no inherent parallelism runs much faster on a normal processor than on a graphics processor.

Contention for memory is also an important issue and needs to be dealt with efficiently. When two or more data items opt for the same block in the cache, there arises a memory contention. Diligent algorithms must be in place to decide which data item needs to go into the cache. Memory contention is not only a problem in on chip memories but also in an off chip scenario. Proper memory models, that make decisions based on the problem size, their frequency of occurrence, etc needs to be adopted in order to understand memory contention issues [7]. With the advent of multi-core processors, simulations have become a tougher task. Normal trace-driven memory simulations consume a lot of

time. Methods have been discussed to alleviate this problem by splitting independent cache simulations across multiple nodes which can now run in parallel and then combining the results effectively [9].
In order to optimize the performance of the processor, proper allocation of software tasks, pertaining to the application's requirements is essential. Refining the existing replacement policies in cache levels, by a better alternative will answer all issues and consequently raise the performance standards.

IV. CACHE AND PERFORMANCE

The data that is being stored in a cache are values that have been computed earlier or copies of original values that are stored elsewhere. Cache performance greatly affects the overall performance of the processor. There are many terms that govern the efficiency when it comes to cache, like hit rate, miss rate etc. The most important cache performance measuring metric is as follows:

4.1. Average Memory Access Time and Processor Performance

In a processor, many issues like memory reference instructions, contention due to I/O devices cause stalls which will reduce the overall performance of the processor. The CPU stalls during misses and the memory stall time is strongly correlated to the average memory access time [17]. Thus,
Average Memory Access Time = Hit Time + Miss Rate x Miss Penalty

4.2. Multi-level cache

It is an advantage to have multiple levels of cache in a processor as it will subsequently increase the hit rate. Generally, there are two levels of cache namely L1 and L2. The order of proximity to the processor is inversely proportional to the size of the various cache levels and directly proportional to the access time of the cache.
Thus, the smallest cache level, L1 which is the closest, has the minimum access time. The L2 cache size will be slightly higher compared to its L1 counterpart but the time taken to fetch data from it will be high as well. In certain cases, an additional level L3 is present to serve the same purpose. Figure.1 shows the memory architecture of a single core in a multi-core environment. L1 cache is integrated within the processor chip whereas L2 is kept away from it. DRAM refers to the primary memory of the system. When a data item misses in L1 it gives rise to the terms L1 miss rate and miss penalty. So is the case with L2 as depicted in the Figure.1

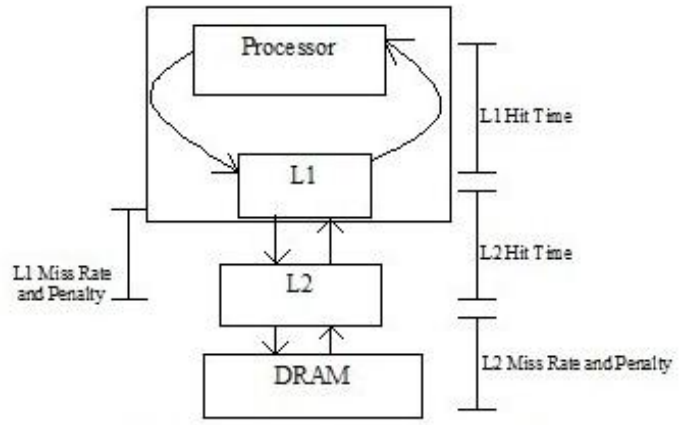

Figure 1. Multi-Level Cache Representation Corresponding to a Single Core

V. PROPOSED CHANGES

We have taken up a multi-core environment consisting of two levels of cache as represented in Figure. 1. When the CPU does not get the desired data in L1, it accesses L2 [17]. The replacement algorithm which works efficiently on L1 might produce sub-optimal performance if it is applied on L2 also simultaneously. The reason for this can be found in the following sections. Researches are currently

being conducted to find an efficient "pair of replacement algorithms" that can be used in L1 and L2, by applying it in various test case scenarios like merge sort, etc

To see why the usage of same algorithm across different cache levels might produce sub-optimal performance, let us assume that the replacement algorithm used in both L1 and L2 caches be the same (say LRU). Now consider the case where L1 encounters a miss on a particular data item, the processor then proceeds to search for the object in L2 cache, if found, it is immediately accessed and the least recently used block in L1 cache is replaced with this block of data which is then subsequently forwarded to the processor for further processing [17]. Any reference to the same data item will now hit in L1 and the access time is greatly minimized.

In a scenario where both L1 and L2 cache encounters a miss, the data is fetched from the next level of memory and is stored in both L2 and L1.As a result we end up in duplicating the same data in both the levels. At first sight one might argue that the data be forwarded directly to L1 and then to processor to eliminate duplication. But according to temporal locality, this data item might be accessed by the processor in the near future. If it ends up being evicted from L1 in future (to make way for new data items) and if there is no copy of it in the relatively larger L2 also, then it has to be fetched again from the main memory which induces a serious miss penalty. Having said that, data duplication is also something that cannot be tolerated. If this continues over a period of time, duplicate data will start flooding the cache resulting in performance bottleneck.

So we recommend using LRU algorithm in Level 1 cache and MRU algorithm in Level 2 Cache. This pair reduces miss rate significantly and results in improving the hit rate as well. More importantly it reduces data redundancy across multiple levels of cache. Fewer amounts of data will be duplicated across the various levels of cache, because our method removes duplication automatically whenever a miss is encountered in both L1 and L2.

To make it clearer, consider the case where both L1 and L2 encounter a miss on a particular data item (which caused the problem in our previous case). The data is fetched from the memory and it replaces the least recently used block in L1 and the most recently used block in L2. As we can clearly see that there is data redundancy similar to our previous case. This state will persist till the next data item arrives. This redundancy is also desirable owing to the reason that was stated above. But this cannot be allowed to exist for a longer period of time. So now when a new data item arrives, again it replaces the least recently used block in L1 and the most recently used block in L2. In L2 this MRU block will be the one which we saw earlier and which has a duplicate copy in L1 cache. Thus the redundant data in L2 is now removed. The concept has been clearly illustrated via an example in Figure. 2A, 2B, 3A, 3B.

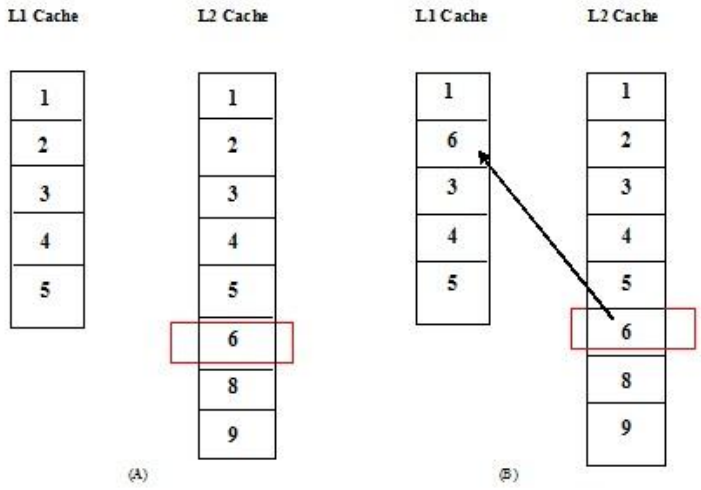

Figure. 2 (A) The Processor Requests for Data Item '6' which is Not Present in L1 but present in L2 cache. (B) Requested Data Item '6' is promoted from L2 to L1 cache and subsequently sent to the Processor for Further Processing

VI. RESULTS AND DISCUSSION

In Figure. 2A, 2B, 3A, 3B the first set of blocks indicate L1 cache and the larger block in the right depicts L2 cache. Both of them are loaded with some arbitrary data.

If both the caches adopt LRU replacement strategy.

Case 1:

Processor requests for the data item '6'. As it can be observed from Figure 2(A), the requested item is not present in L1 cache and thus the request is forwarded onto the L2 cache. Here, a hit is encountered and the data item is copied to the L1 cache (considering the temporal locality principle) as shown in Figure. 2(B) and is subsequently forwarded to the processor. The Least Recently Used block in L1 cache is assumed to be '2' in this case. This is the block which is replaced by '6'.

Case 2:

Processor requests for a data item '7'. The requested item is not present in L1 cache and thus the request is forwarded onto the L2. L2 cache also encounters a miss and the request is forwarded to the main memory. The fetched data '7' is put in L2 cache and another copy is placed in L1 cache as illustrated in Figure 3(A). Assume that the Least Recently Used blocks in L1 and L2 are '1' and '4' respectively. As both the caches use LRU policy, there arises a duplication of data item 6 (which is circled in Figure. 3(A)). Further it can be clearly seen that if this trend continues, every data item will end up being duplicated across L1 and L2 cache.

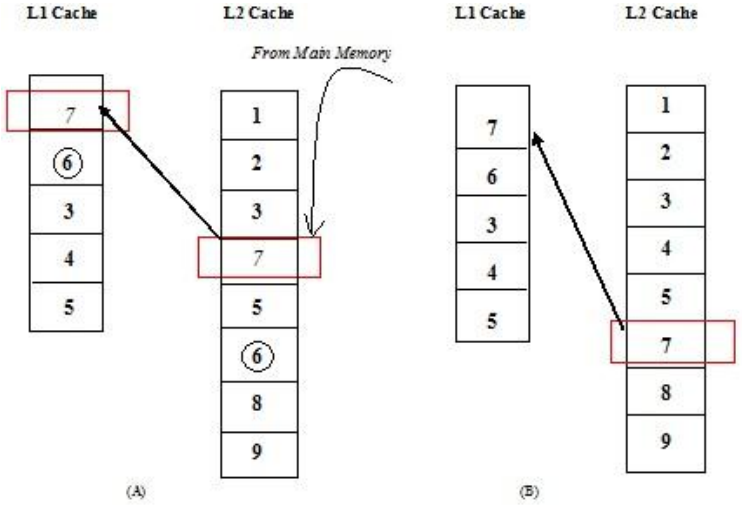

Figure 3. (A) Processor Requests for a New Data Item '7' (absent in both the caches) which is fetched from the Main Memory and put in L2 cache. It is then promoted to L1 cache and finally to Processor. Data Item '6' is circled to indicate Duplication (here both the caches adopt LRU Policy). (B) Now L1 adopts LRU and L2 adopts MRU policy. Processor Requests for a New Data Item '7' which is fetched from the Main Memory and put in L2 cache. It is then promoted to L1 cache and finally to Processor. No Duplication of Data Item '6' (in L2 cache) as it is replaced by '7' due to MRU policy.

Consider a scenario where both caches adopt a different 'pair of replacement policies'. Let L1 use LRU and L2 use MRU.

Case 1:

The processor requests for a specific data item, which misses in L1 cache but hits in L2 cache. The data is then forwarded to L1 cache and the Least Recently Used block is replaced with this new incoming data item, which is then subsequently forwarded to the processor for further processing.

Case 2:

Now when the processor requests for a data item which is not present in both the caches (in our example it is '7'), it is fetched from the main memory and replaces the Most Recently used data block

(in our case it is '6' because that was the one which was accessed in the previous step). Figure. 3(B) shows that the data item '6' is replaced by '7' in L2 where as in L1 the Least Recently Used block '1' is replaced by '7'. We can see that this method does abide by the temporal locality principle as it leaves a copy of '6' untouched in L1 which might be accessed in the near future. Now we can observe that the duplicate copy of '6' in L2 has vanished and is replaced by '7'.

On a long term basis, this method will evict many such duplicate data from L2 cache. And most importantly not compromising on the temporal locality principle. At any given point of time, the recently accessed data is guaranteed to be present in L1 cache. Only its duplicate in L2 is replaced after some time to pave way for new incoming data. The memory utilization and processor performance is enhanced considerably over a period of time.

VII. CONCLUSION

Data duplication can be harmful particularly in cache memory architecture. This is because of the very limited size allocated to cache memories. When it comes to multi-core processor, each core will have its own cache memory [17] and duplication across the cache levels will slow down the system and defeat the reason for opting for a multi-core environment. With the advent of banked cache architecture [12], space utilization may have improved a little but still there is a vast scope for improvement. This stresses the importance of an efficient replacement algorithm at the cache level. LRU and MRU by themselves perform well individually but when any one of them is adopted across multiple levels of cache, it might result in data duplication.
From the above mentioned illustration, one can see that cache data duplication can be minimized greatly if the suggested replacement pair is implemented across the cache levels.
L1 Cache: LRU algorithm
L2 Cache: MRU algorithm
In a scenario where a data fetch from L2 has been made, following a miss in L1, 2 copies of same data exists on both L1 and L2. By adopting MRU in L2, when a new data item arrives immediately to L2 from secondary memory, it replaces the most recently used element in L2 thereby removing the previously present duplicate copy. This pair can be adopted in all cache memories across all the available cores, which will help to witness a significant improvement in the overall system performance.

VIII. FUTURE WORK

The replacement pair suggested in this paper will help to reduce redundancy with respect to certain data sets. But finding out an optimal replacement pair that will work fine for all the cases still remains a challenge. To make it achievable, the data access pattern of cache can be studied, which in turn will help to choose an appropriate replacement pair in a dynamic fashion, instead of fixing it statically at the beginning. There are prediction techniques available, like RRIP [16**]**, but they are targeted in finding out a single replacement algorithm that is optimal at that given point of time. The same idea can be followed to select an optimal replacement pair in multi-level cache architecture, thereby elevating the level of utility of multi-core processors to a new dimension.

REFERENCES

[1]. Felipe L. Madruga, Henrique C. Freitas, and Philippe O. A. Navaux, (2010) "Parallel Shared-Memory Workloads Performance on Asymmetric Multi-core Architectures", Institute of Informatics, Federal University of Rio Grande do Sul, UFRGS Porto Alegre,Brazil, 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, pp 163-169.

[2]. Marios Klenthous and Yiannakis Sazeides, University of Cyprus, (2011) "CATCH-A Mechanism For Dynamically Detecting Cache-Content-Duplication in Instruction Caches", ACM transactions on Architecture and Code Optimization, vol. 8,no.3,Article 11.

[3]. Hyunjin Lee, Sangyeun Cho and Bruce R. Childers, (2011) "DEFCAM:A Design and Evaluation Framework for Defect-Tolerant Cache Memories", University of Pittsburgh, ACM transactions on Architecture and Code Optimization, vol. 8,no.3,Article 17.

[4]. Carole-Jean Wu and Margaret Martonosi, (2011) "Adaptive Time Keeping Replacement : Fine-Grained Capacity Management for Shared CMP Caches", Princeton University, ACM transactions on Architecture and Code Optimization, vol. 8,no.3,Article 17.

[5]. Hasitha Muthumala WaidyaSooriya, Yosuke Ohbayashi, Masanori Hariyama, Michitaka Kameyama, (2011) "Memory Allocation Exploiting Temporal Locality for Reducing Data –Transfer bottlenecks in Heterogeneous Multi-core Processors", IEEE transactions on Circuits and Systems for Video Technology, vol.21, no.10, pp 1453-1466.

[6]. Izuchukwu Nwachukwu, Krishna Kavi, Fawibe Ademola and Chris Yan, (2011) "Evaluation of Techniques to improve Cache Access Uniformities", University of North Texas, International Conference on Parallel Processing, pp 31-40.

[7]. Bogdan Marius TUDOR, Yong Meng TEO, (2011) "Understanding off-Chip Memory Contention of Parallel Programs in Multi-core Systems", National University of Singapore, Simon SEE, NVIDIA International Conference on Parallel Processing,pp 602-611.

[8]. Vinay Saripalli, Guangyu Sun, Asit Mishra, Yuan Xie, Suman Datta and VijayKrishnan Narayanan, Fellow, (2011) "Exploiting Heterogeneity for Energy Efficiency in Chip MultiProcessors", IEEE emerging and selected topics in Circuits and Systems , vol.1, no.2.

[9]. Georgios Keramidas, Nikolaos Strikos Stefanos Kaxiras, (2011) "Multi-core Cache Simulations using Heterogeneous Computing on General Purpose and Graphics Processors", Industrial Systems Institute, Patras, Greece, 14th Euromicro Conference on Digital System Design, pp 270-273.

[10]. Shouqing Hao, Qi Liu, Longbing Zhang and Jian Wang, (2011) "Processes Scheduling on Heterogeneous Multi-core Architecture with Hardware Support", Institute of Computing Technology, Sixth IEEE International Conference on Networking, pp 236-241.

[11]. Mathias Jacqueline, (2011) "Memory aware Algorithms and Scheduling Techniques from Multi-Core Processors to Petascale Supercomputers", France, IEEE International and Parallel and Distributed Systems, pp 2038-2041.

[12]. Kakoee, Mohammad Reza, Vladimir Petrovic and Luca Benni, (2012) "A Multi-Banked Shared-L1 Cache Architecture for Tightly Coupled Processor Clusters", Finland, IEEE International Symposium on System On Chip (SOC), pp 1-5.

[13]. Khan S.M, Zhe Wang and Jimmez D.A, (2012) "Decoupled Dynamic Cache Segmentation", IEEE High Performance Computing Architecture, pp 1-12.

[14]. Fang Juan and Lei Ding, (2012) "The Shared Cache Reconfigurable Method for Low Power Consumption in CMPs", IEEE Symposium on Robotics and Applications (ISRA), pp 171-173.

[15]. Huang Z, Mingfa Z and Limin X, (2012) "LvtPPP: Live-Time Protected Pseudo-Partitioning of Multi-Core Shared Caches", IEEE Transactions on Parallel and Distributed Systems, Issue 99, pp 1.

[16]. Aamer Jaleel, Kelvin B Theobald, Simon C Steely and Joel Emer, (2010) "High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP)", ACM Proceeedings of 37th Annual International Symposium on Computer Architecture (ISCA), vol 38, Issue 3, pp 60.

[17]. John.L.Hennessy and David.A.Patterson, "Computer Architecture A Quantitative Approach", Elsevier, 423 pages, 4th Edition, 2009.

AUTHORS

Muthukumar S received the BE degree in Electronics and Communication Engineering and the ME degree in Applied Electronics from Bharathiar University, Coimbatore in 1989 and 1992 respectively. He is currently pursuing the Ph.D. degree from BSA University, Chennai. His current research interests include Multi-Core Processor Architectures, High Performance Computing and Embedded Systems.

P K Jawahar received the BE degree in Electronics and Communication Engineering from Bharathiar University, Coimbatore in 1989 and MTech degree in electronics and communication engineering from Pondicherry University 1998. He received the Ph.D. degree in Information and Communication Engineering from Anna University, Chennai in 2010. His current research interests include VLSI, Embedded Systems and Computer Networks.