# MEMORY CUSTOMIZATION IN MULTIPROCESSOR SYSTEMS–ON–CHIP

A THESIS
Submitted in partial fulfillment of the
requirements for the award of the degree of

DOCTOR OF PHILOSOPHY
IN
COMPUTER SCIENCE & ENGINEERING
BY

SHAILY MITTAL
(Enrollment No.: 106208)

Department of Computer Science & Engineering and Information Technology,
Jaypee University of Information Technology, Waknaghat, Solan–173234,
Himachal Pradesh, INDIA

APRIL 2014

# MEMORY CUSTOMIZATION IN MULTIPROCESSOR SYSTEMS–ON–CHIP

A THESIS
Submitted in partial fulfillment of the
requirements for the award of the degree of

DOCTOR OF PHILOSOPHY
IN
COMPUTER SCIENCE & ENGINEERING
BY

SHAILY MITTAL
(Enrollment No.: 106208)

Department of Computer Science & Engineering and Information Technology,
Jaypee University of Information Technology, Waknaghat, Solan–173234,
Himachal Pradesh, INDIA

APRIL 2014

# ACKNOWLEDGEMENTS

I sincerely express my gratitude to my advisor, Dr. Nitin, for consistent guidance in this research work. I am fortunate to have blessings of almighty, brought closer to my advisor, is always interested in exploring new research directions and ideas with deep vision, encouragement and dimension. He gave freedom in performing research activities and ensured that my various contents are neatly tied together. He has personal touch and support to all his students. One can easily feel friend and philosopher in his disposition.

This research has benefited from the ideas and suggestions of many well wishers. Special thanks to Management, Jaypee Sewa Sansthan especially to Dr. Yajulu Medury, COO– Jaypee Education System & founder Vice Chancellor, Brig. Balbir Singh (Retd.), Director and Prof.(Brig.) S.P.Ghera, HOD CSE department, Jaypee University of Information Technology, Waknaghat, for providing excellent research environment and learning ambiance at the campus.

I am eternally grateful to my respected Mother, Father, Mother-in-law, Father-in-law and affectionate younger Brother and Sister-in law beyond words and for the most devoted and supportive Husband for his love and support he have given me all over these years. Without their sustained moral support, it would have been impossible to succeed in writing this wonderful thesis.

Shaily Mittal

# CANDIDATE'S DECLARATION

I hereby certify that the work, which is being presented here in the thesis entitled, MEMORY CUSTOMIZATION IN MULTIPROCESSOR SYSTEMS–ON–CHIP, submitted in partial fulfillment of the requirements for the award of the degree of Doctor of Philosophy in Computer Science & Engineering and submitted in Department of Computer Science & Engineering and Information Technology, Jaypee University of Technology (JUIT), Solan is an authentic record of my own work carried out (July 2010–April 2014) under the supervision and guidance of Dr. Nitin, Associate Professor, Department of Computer Science & Engineering and Information Technology Chancellor, JUIT, Solan.

I hereby declare that this Doctor of Philosophy thesis, my original investigation and achievement has not been submitted by me for the award of any other degree on this work in any other Institution/University.

DATE:

SHAILY MITTAL
(Enrollment No.: 106208)

I certify that I have read this thesis and that in my opinion; it is fully adequate in scope and quality as a thesis for the award of degree of the Doctor of Science in Computer Science and Engineering.

DATE:

DR. NITIN
(ADVISOR)

# ACKNOWLEDGEMENTS

I sincerely express my gratitude to my advisor, Dr. Nitin, for consistent guidance in this research work. I am fortunate to have blessings of almighty, brought closer to my advisor, is always interested in exploring new research directions and ideas with deep vision, encouragement and dimension. He gave freedom in performing research activities and ensured that my various contents are neatly tied together. He has personal touch and support to all his students. One can easily feel friend and philosopher in his disposition.

This research has benefited from the ideas and suggestions of many well wishers. Special thanks to Management, Jaypee Sewa Sansthan especially to Dr. Yajulu Medury, COO– Jaypee Education System & founder Vice Chancellor, Brig. Balbir Singh (Retd.), Director and Prof.(Brig.) S.P.Ghera, HOD CSE department, Jaypee University of Information Technology, Waknaghat, for providing excellent research environment and learning ambiance at the campus.

I am eternally grateful to my respected Mother, Father, Mother-in-law, Father-in-law and affectionate younger Brother and Sister-in law beyond words and for the most devoted and supportive Husband for his love and support he have given me all over these years. Without their sustained moral support, it would have been impossible to succeed in writing this wonderful thesis.

Shaily Mittal

# LIST OF FIGURES

# LIST OF TABLES

# CANDIDATE'S DECLARATION

I hereby certify that the work, which is being presented here in the thesis entitled, MEMORY CUSTOMIZATION IN MULTIPROCESSOR SYSTEMS–ON–CHIP, submitted in partial fulfillment of the requirements for the award of the degree of Doctor of Philosophy in Computer Science & Engineering and submitted in Department of Computer Science & Engineering and Information Technology, Jaypee University of Technology (JUIT), Solan is an authentic record of my own work carried out (July 2010–April 2014) under the supervision and guidance of Dr. Nitin, Associate Professor, Department of Computer Science & Engineering and Information Technology Chancellor, JUIT, Solan.

I hereby declare that this Doctor of Philosophy thesis, my original investigation and achievement has not been submitted by me for the award of any other degree on this work in any other Institution/University.

DATE:

SHAILY MITTAL
(Enrollment No.: 106208)

I certify that I have read this thesis and that in my opinion; it is fully adequate in scope and quality as a thesis for the award of degree of the Doctor of Science in Computer Science and Engineering.

DATE:

DR. NITIN
(ADVISOR)

# TABLE OF CONTENTS

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

An embedded system is an element of a complete device in itself which includes hardware as well as software. Embedded systems have major applications in lots of day today's appliances. Reduction in memory size is essential to achieve better performance. Embedded devices also often have different design constraints in comparison with general purpose computers. Some embedded system constraints are listed below:

1. Available system-memory
2. Available processor speed
3. Limited power dissipation

There are strict constraints on the embedded systems for small system size, low weight, low power consumption, good performance, high reliability and low cost. All these factors play a major role in the application of embedded systems which are going to be discussed below in detail [P. J. Koopman, 1989].

1. Size and weight: Applications such as aircrafts and portable equipments have a severe weight limitation. Some applications also have size limitation. A typical embedded system can have a maximum size in hardly any cubic inches and maximum weight in few pounds only.
2. Reliability: Military and automotive applications need extremely reliable operating conditions that can deal with vibration, shock, extreme heat and cold etc because in remote areas, system must be capable enough to survive without any repair service.
3. Cost: Applications like consumer electronics products have cost as a very important factor. In general, system complexity is directly proportional to the system cost. Hence, these systems must be designed with very less complexity.
4. Power and cooling: Power consumption of the system is also affected by the system complexity. High power consuming systems require heavy and huge power supply mechanism. Moreover, more the power consumed by the system,

more the heat produced and hence increase in the requirement of cooling mechanism.

5. Performance: There are a number of applications requiring higher performance, such as voice translation, moving picture recognition, and others. In many multimedia embedded applications performance constraints are formulated as soft or hard real-time time constraints for a periodic task, such as frame decoding in a video player. Unlike as in general purpose computers, where performance is an important optimization criteria, embedded systems may not always benefit from increased performance beyond given real-time constraints. For example, for a video player application, the user satisfaction with the device will not improve if the player can decode 100 instead of 30 frames per second, since the video clips are typically encoded with the rate of no higher than 30 fps due to the limited ability of a human eye to distinguish between fast changing frames.

## 1.2 Cache Memory Overview

Cache memory is like random access memory accessed by processor faster than accessing a RAM. When a processor requires some data for a process execution, it first checks the presence of data in the cache memory and if it finds the data there, it needs not to read data from larger memory i.e memory at next level in hierarchy which is more time-consuming.

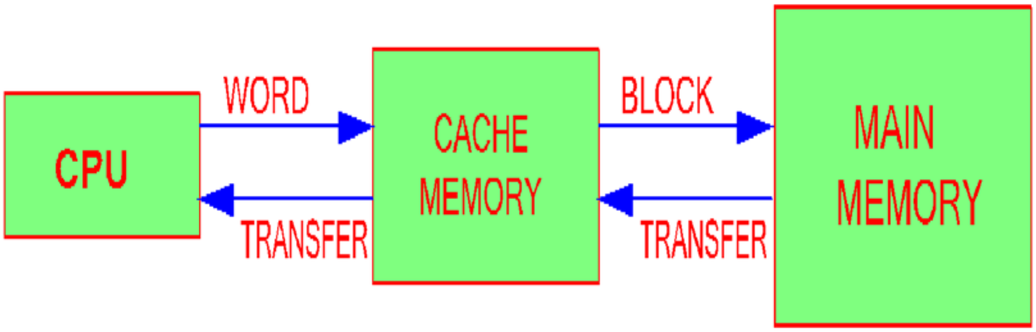


Figure 1.1: CPU organization

Cache is more close to processor as cache locates between normal main memory and CPU. The architecture of CPU is shown in fig. 1.1 [Rob Williams, 2006]. A word is transferred between CPU and cache memory while a block transfer takes place to and from main memory and cache memory.

### 1.2.1 Unit of Transfer

Figure 1.2 explains the concept of mapping a main memory address generated by the processor to cache memory [D. Tarnoff, 2006]. This mapping is used in case of reading and writing data from and to the cache memory. There are majorly three different types of mapping functions explained below.

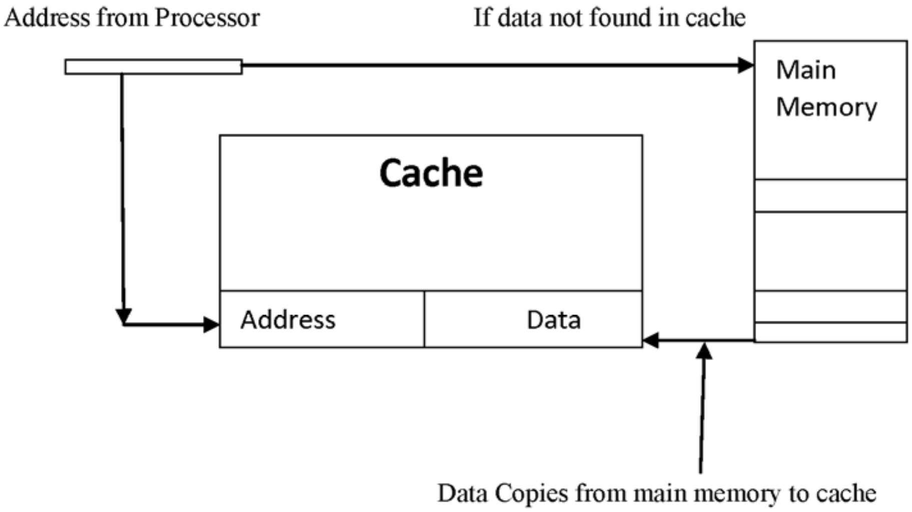

Address from Processor
If data not found in cache
Main Memory
Cache
Address
Data
Data Copies from main memory to cache

Figure 1.2: A basic main memory to cache memory mapping scheme

1. Direct mapping (fig. 1.3) : [N.P.Jouppi, 1990] In case of direct mapping, each main memory block maps to only one cache line, hence it will always be found in the same place in memory by using formula given below [D. Tarnoff, 2006]:

   l = b mod m

   Where

   l = Number of mapped cache line.

   b = Number of main memory block.

   m = Number of cache lines

Advantages and disadvantages of direct mapping are [R. Williams, 2006]:

It is simple to understand.

It is economical

It has a permanent location for a given block

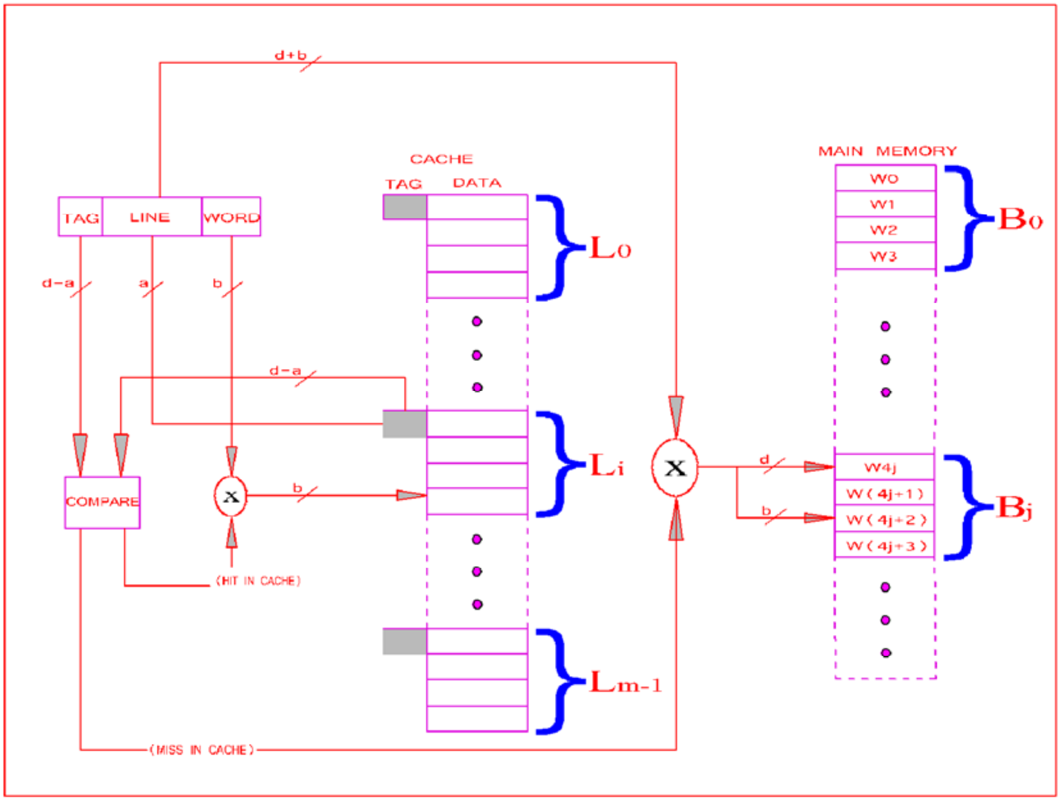


Figure 1.3: Direct mapping arrangement [M.D.Hill, 1988]

Ex. Main memory =1MB= $2^{20}$

Cache memory=128KB= $2^{17}$

Block size=16 B = $2^4$ Set size= 1block

No.of sets in cache= cache size / (block size * set size) = $2^{17}$ / ($2^4$* 1) = $2^{13}$

No. Of tag bits = 20 – 13 - 4 = 3

2. Fully associative mapping (fig. 1.4): In this mapping, a main memory block can be mapped to any cache block as whole memory is one set here [D. Tarnoff, 2006]. The size of tag is d bits and word size is b bits.

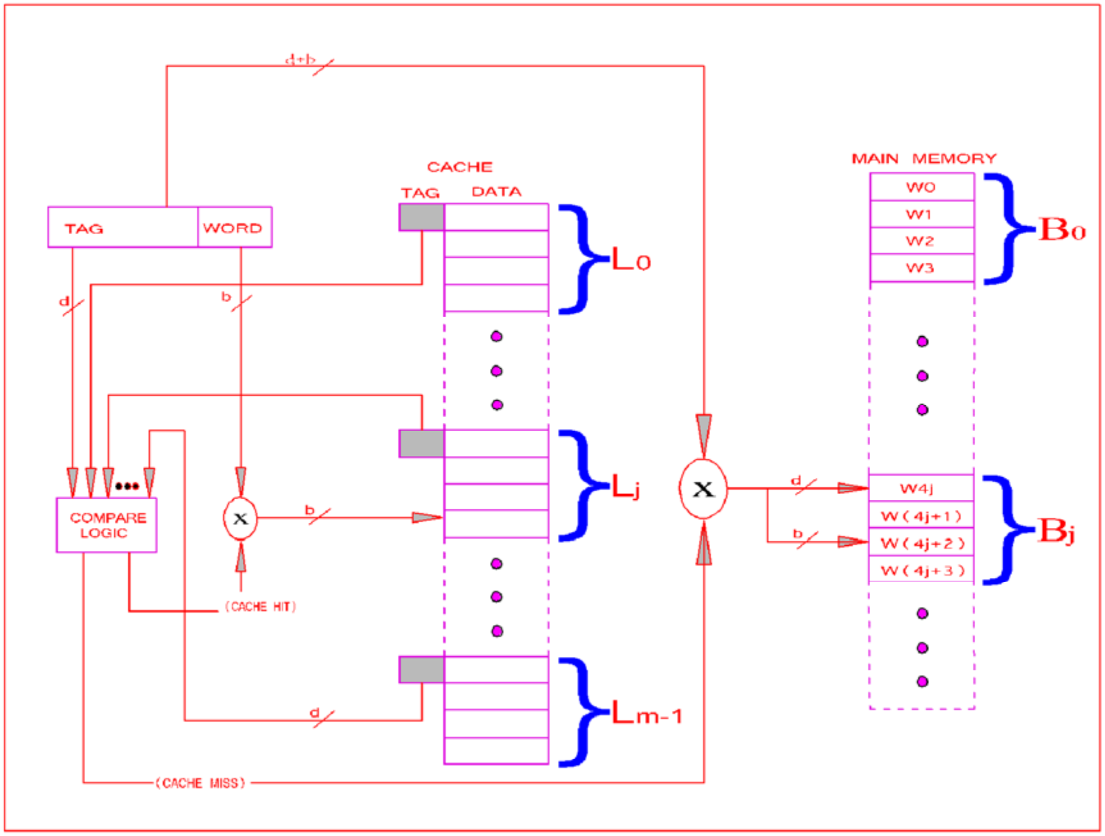


Figure 1.4: Fully associative mapping arrangement [M.D.Hill, 1988]

Ex. Main memory =256MB= $2^{28}$

Cache memory=64KB= $2^{6}$

Block size=32 B = $2^{5}$

No. Of tag bits = 28 - 5 = 23

3. Set associative mapping (fig. 1.5): Cache is divided into m sets. N blocks/lines can be contained within each set. Such a cache memory is called n-way set associative. It is hybrid of direct and associative mapping [D. Tarnoff, 2006].

Number of lines in a cache = m • n

Size of tag = (d-b) bits

when n = 1, direct mapping

when m = 1, fully associative mapping

Ex. Main memory =1MB= $2^{20}$

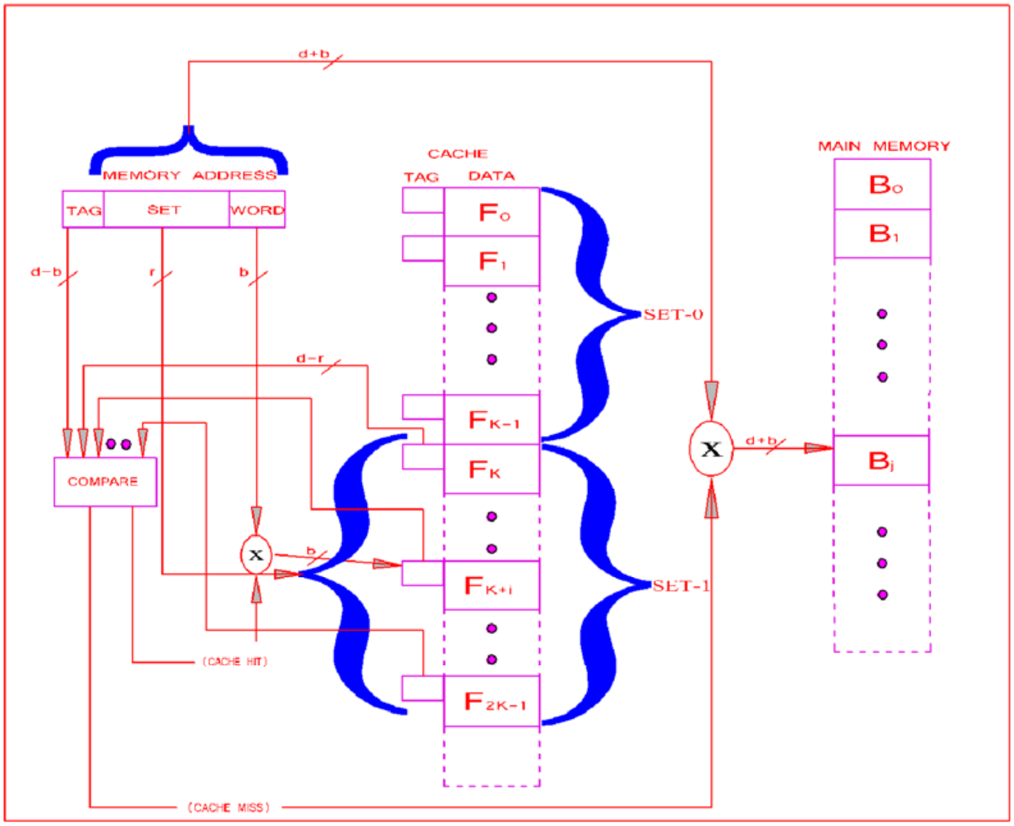
Cache memory=128KB= $2^{17}$

Block size=16 B = $2^4$ Set size= 8 blocks= $2^3$

No.of sets in cache= cache size / (set size * block size) = $2^{17}$ / ($2^3$ * $2^4$) = $2^{10}$

No. Of tag bits = 20 – 10- 4 = 6

A main memory block can map to any cache block in its specified set as shown in the fig. 1.5.



Figurc 1.5: Sct associativc mapping arrangcmcnt [M.D.Hill, 1988]

### 1.2.2 Locality of Reference [A.A.Jayya, 2005]

In programming, instructions and data are likely to cluster together (loops, subroutines etc.), this property is referred to locality of reference.

- Clusters will change in long duration.
- Clusters remain same in short duration [D. Tarnoff, 2006].

There are 3 types of locality of reference which can be exploited to keep more frequently referenced information into cache memory.

1. Spatial locality: The tendency of a process to retrieve items whose addresses are close to one another is called spatial locality [M. Mano, 1989]. For example, operations on tables or arrays involve accesses of a certain clustered area in the address space.
2. Temporal locality: Memory items that are recently referenced are more likely to be referenced soon than those which have not been referenced for a longer time. Memory items (instructions or data) in a few localized areas of the memory (program or data structure) are more frequently referenced than other areas. (Ex: loops, functions, subroutines, arrays etc.)
3. Sequential locality: Execution of instructions follows a sequential order unless branch instructions create out of order executions.

### 1.2.3 Replacement Policy

The main objective of replacement policy is to locate a space in cache memory for a required main memory block. The replacement policy tells the slot to be swapped out to make space free for the new block [M.J. Murdocca, 2007]. Some of the most widely used replacement policies are:

- Belady's algorithm: The most efficient caching algorithm rejects the unwanted information for the longest duration in the future reference. This optimal choice is referred to as Belady's optimal algorithm or sometimes the clairvoyant algorithm. Since it is generally impossible to guess how far in the future this information will be needed, hence this is generally not possible to implement in practice.
- Random: In this policy, a victim block is randomly chosen and then discarded to make space for new block. Due to its simplicity, it has been used in ARM processors. It admits efficient stochastic simulation.
- FIFO (First-In First-Out): The easiest page-replacement algorithm is FIFO algorithm. As the name suggests, it keeps track of all the pages in memory by implementing a queue, by keeping the latest arrived page at the back, and the earlier arrived pages in front. As deletion in queue is done from front, so as the

memory page from the front is replaced to vacant room for the new block to be inserted at the end of the queue [Shenoy, 2008]. Although, it does not perform good in practical application, still it is cheap.

- LRU (Least Recently Used): Least recently used discards the least recently used memory blocks first. When each block is used record is maintained by this algorithm which is an expensive job.

### 1.2.4 Cache Addressing Models [A. A. Jerraya, 2005]

There are two cache addressing models. One is physical address cache and another is virtual address cache. Both are discussed in detail below.

Physical address cache: When a cache is accessed with a physical memory address, it is called a physical address cache. Cache is indexed and tagged with physical address. Figure 1.6 describes the concept of physical addressing.

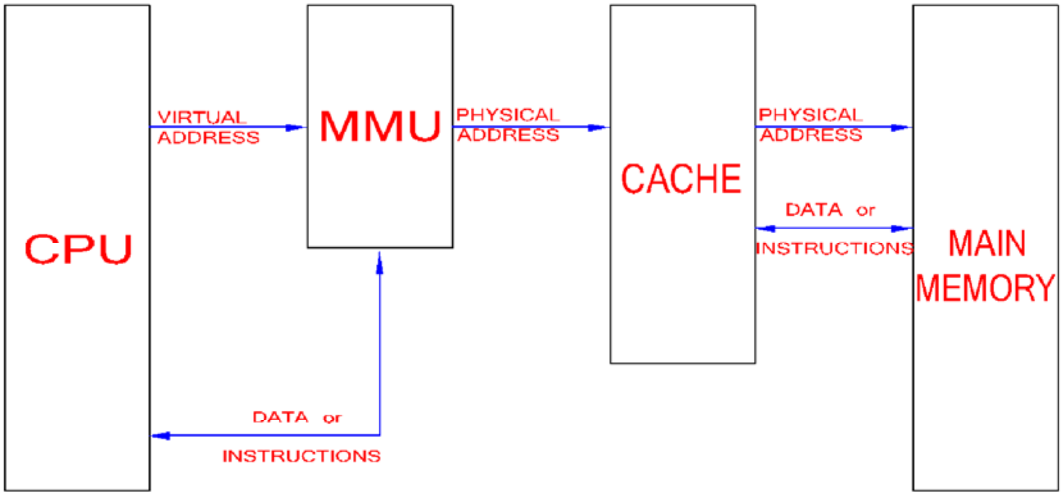


Figure 1.6: A unified cache accessed by physical address [Mark D. Hill, 1987]

Different pros and cons are:

- There is no need to perform cache flushing.
- There are no aliasing problem
- It slows down accessing cache until MMU finishes translating the address.

Virtual address cache: When a cache is indexed or tagged with a virtual address, it is called virtual address cache. In this model, both cache and MMU translation can be done in parallel. Virtual addressing is shown in fig. 1.7.

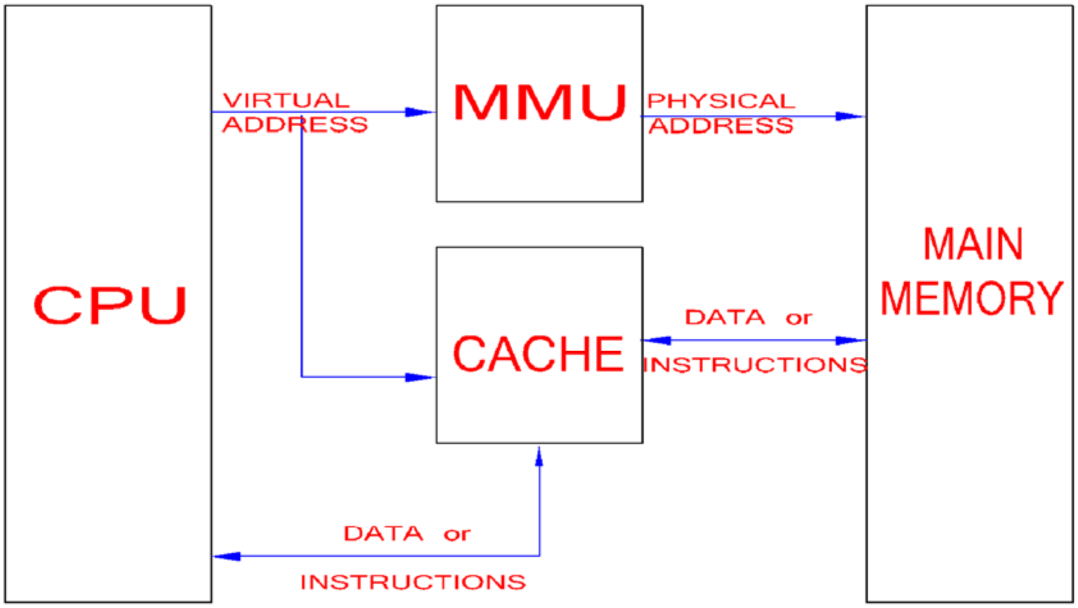


Figure 1.7: A unified cache accessed by virtual address [Mark D. Hill, 1987]

Different pros and cons are:

- Enhanced efficiency to access cache faster.
- Cache aliasing
- Frequent cache flushing

### 1.2.5 Memory Capacity Planning

As only a fraction of all main memory blocks can be placed in cache memory (CM) at a time, a word needed by the CPU may or may not be found in cache memory.

Hit ratio, H: Probability for the CPU to find the needed information in cache memory.

$$\text{Hit Ratio} = \frac{\text{No.of times referenced words are in cache}}{\text{Total Number of memory accesses}}$$

Miss ratio, M: Probability of not finding the needed information in cache memory and main memory has to be accessed.

$M = 1\text{-}H$

Access frequency to Mi:

$F_i = (1\text{-}H_1)(1\text{-}H_2)\ldots\ldots(1\text{-}H_{i\text{-}1})H_i$

Here $H_i$ corresponds to hit ratio at $i^{th}$ memory level.

Effective access time is defined as time taken to access data. [ K.Hwang, 1993]

$$\text{Effective access time} = \frac{(\text{No.of hits})(\text{Time per hit}) + (\text{No.of misses})(\text{Time per miss})}{\text{Total number of memory accesses}}$$

$T_{eff}$ = Sum ($F_i$ * $T_i$) for all memory levels

Cost optimization:

$C_{total}$ = Sum ($C_i$ * $S_i$) for all memory levels

Here $C_i$ represents cost of cache per bit and $S_i$ denotes the size of cache in bits at level i.

### 1.3 Systems on Chip

A system on a chip is an integration of computer components like processor, memories, communication network etc on a single chip. Embedded systems are characteristic applications of SoC. The SoC and microcontroller both are of same degree. Normally, microcontrollers have RAM of capacity 100 KB with single-chip-systems, while SoC has more powerful processors proficient of running desktop versions of Windows and Linux, which require external memory chips and are also used with a range of external peripherals. System on a chip indicates a technical direction more than reality and hence chip integration is increasing day by day to cut down manufacturing costs and to facilitate smaller systems.

## 1.4 Multiprocessor Systems on Chip

An MPSoC is a system on-chip, a VLSI system that incorporates most or all the components necessary for an application—that uses multiple programmable processors as system components. MPSoC's are widely used in networking, communications, signal processing and multimedia among other applications. MPSoC's embody an important and distinct branch of multiprocessors. They are not simply traditional multiprocessors shrunk to a single chip however they have been designed to satisfy the unique requirements of embedded applications. MPSoC's have been in production for much longer than multi core processors. In MPSoC's, each processor has its own cache memory with one or two levels, which is called local memory. In addition they have an independent private cache memory for each processor; there is a possibility that two or more cache memories may contain different versions of the same information at the same time. This is called cache-memory coherence problem discussed in detail later in this chapter.

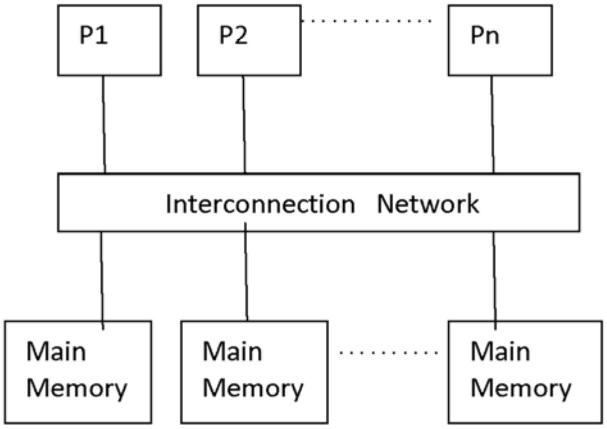


Figure 1.8: Architecture of multiprocessor system with shared memories [ V. Gandhi, 2013]

Figure 1.8 shows the traditional view of architecture of shared-memory multiprocessor i.e. a collection of processors and memory connected by an interconnection network. This architecture is less complicated and hence preferred than others.

The architecture of an MPSoC system is a combination of three things: PE's, memory elements and a communication infrastructure for communication between both. While keeping processing in mind, MPSoC's are classified into two classes:

- Homogeneous: In a homogeneous system, all PE's in the system have the same architecture. The fact that all PE's have the same architecture facilitates task migration, as a result of there's no would like for translating the code of a given application to alternative design.
- Heterogeneous: During this category the design of a minimum of one PE's is totally different from the others. The advantage of this category is to develop real time systems. As an example, an MPSoC system will have an ARM processor for handling system tasks and a DSP processor for handling 3G signals. Considering the communication infrastructure accustomed interconnect PE's and memory components, three infrastructures are ordinarily used in MPSoC: dedicated wires, shared buses and networks-on-chip (NoC's).

MPSoC styles are largely supported networks-on-chip (NoC's) as they supply quantifiability, high information measure, energy potency, dependability, parallel communication and climbable style exploration house. A NoC agent is associate on-chip network composed by switches that are connected among themselves by communication channels.

### 1.5 Memory Organization

According to [M. Kandemir, 2005], one in all the foremost essential elements that confirm the success of an MPSoC-based design is its memory system. This assertion is even by the actual fact that applications may spend many cycles expecting the conclusion of read/write memory operations.

The technology employed to develop processing elements advances quicker than that used in the development of memory components, which enables PE's to work at higher frequencies. This disparity allows PE's to consume data at rates not possibly achieved by DRAM memories and creating a performance gap. To decrease this gap, a solution commonly applied in high-end microprocessors is the use of static memories and a memory hierarchy. In a hierarchy, several levels of memories are used to decrease average memory access latency. The main idea is to place faster however smaller memories closer to processors and slower however larger memories in further levels. The smaller memories contain a subset, which consists usually of the most accessed data from the data stored in the next adjacent level. In general-purpose systems, there usually exist four levels of memory: level 1 cache, level 2 cache, main memory and secondary memory.

Cache memories can provide an acceptable data rate to feed the processor, maximizing the number of instructions that are executed in a certain period. Caches work as temporary, fast access memories that prevent the processors to stall while waiting for an instruction or data from main memory. Another interesting point in the use of caches is that they can reduce energy consumption, once memory accesses are local, avoiding transactions on a bus/network-on-chip that would be necessary to bring a block of data from main memory/secondary memory.

MPSoC systems tend to have hundreds of elements [ITR, 2011]. As the system size increases, there is a need to develop mechanisms that optimize these systems in several aspects such as energy consumption, latency and resource allocation. To decrease energy consumption, techniques such as DFVS may be applied to decrease the energy consumption of PE's when they are executing low priority tasks or are idle. Also, in multiple memory bank systems, there is the possibility of migrating data from a bank to another to approximate them to the processors that are mostly accessing those data. Accesses done to the data after migration tend to consume less energy in communication and take less time as the distance between the memory bank and the PE's decreases.

Shared memory is that memory which can be concurrently accessed by multiple programs for communication between them and to avoid redundant copies i.e cache incoherence. In such systems, depending on history programs may run on single processor or multiple processors [V. Gandhi, 2013].

Keeping shared memory near to processor increases the inter-processor communication hence, better except that it decreases non shared memory accesses and if shared memory is kept at lower level of hierarchy, it acts vice versa. Hence, the architectures sharing less layers of the memory hierarchy can scale better. The various available design alternatives are as follows:

- Shared L1 cache – It is used where all processors share a single pipeline and are using the concept of multithreading.
- Shared L2 cache – It is mainly used in chip multiprocessor systems (CMP). It is in level next to shared L1 cache. Access time of shared L2 cache is higher and hence private L2 caches are mainly used in a lot of existing CMP systems.
- Shared main memory - In this organization, every processor or core has its own private L1 and L2 caches as well as a single shared main memory.
- No physical sharing – [L.Ryzhyk, 2006] In this architecture, each processor has its own private main memory and also have an access to the memory located remotely to other nodes through interconnection network. As, the cost of local memory access is much lower than the remote memory access; thus this architecture is known as NUMA ( Non – uniform memory access) [ Burroughs Corporation, 1990].

After the design of memory hierarchy is decided, the next step is to devise the interconnection between different levels of the memory hierarchy. The interconnection is divided into two major types that are bus-based and network-based interconnects. The main difference between these two is that a bus consists of shared resource used by a single client at a time, and hence supports a single flow of data, whereas an interconnection network allows multiple concurrent flow of data and hence larger bandwidth.

Data memory architecture consists of three components: cache memory, scratch-pad memory (SPM) and main memory. The cache memory and the SPM are on-chip SRAMs and the main memory can be assumed to be an off-chip DRAM (with a higher access latency). As shown in fig. 1.9, the address space is divided between off-chip memory and on-chip SPM, the former of which is accessed through the on-chip cache.

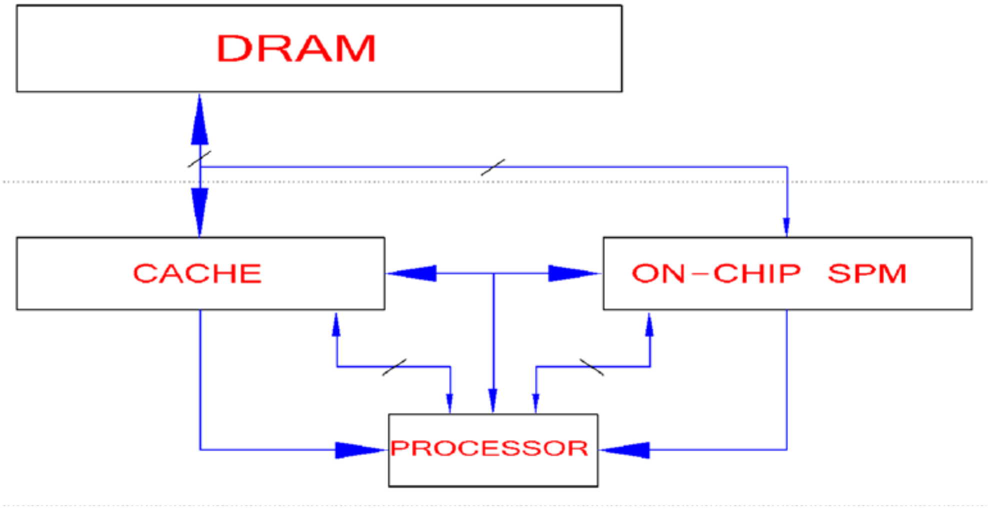


Figure 1.9: System architecture with on-chip and off-chip memory organization

## 1.6 Current Issues of Cache Memory Access

### 1.6.1 Multiple Processor Synchronization

Embedded platforms are almost invariantly deployed in tightly resource constrained contexts. Hence, the performance and power cost of synchronization is a serious concern that is further compounded by the issue of providing effective programming abstractions. Deadlocks are difficult to avoid, especially when systems have multiple resources. Moreover, even when the system is operating correctly, conditions like lock spinning may impose a significant overhead in power and performance, since they tend to flood the interconnect with useless transactions.

### 1.6.2 Processor-Memory Speed Gap

In multiprocessor systems on chip (MPSoC's), the effect of the increasing processor–memory speed gap is being more significant due to the heavier access contention on the network and the use of shared memory. Therefore, improvement in memory performance is critical to the successful use of MPSoC systems.

### 1.6.3 Cache Coherence

Shared memory is a common inter-processor communication paradigm for single-chip multiprocessor platforms. Presence of duplicate date in multiple caches is cache coherence as shown in fig. 1.10. These are 2 general techniques to avoid cache coherence issues and others in fig. 1.11:-

1. Write through: [D.Tarnoff, 2006] Every time any modification to any value is updated in main memory as well as in cache memory. It generates lots of traffic. It also delays write operation as it has to update two memories simultaneously.
2. Write back:  Updates is initially made in cache memory only and corresponding dirty flag bit is set. So, only that blocks whose dirty bit is set needs to be written to main memory and other caches remain same. Research shows that 15% of total memory references are write operations [D.Tarnoff, 2006].

Write through technique still not guaranteed the presence of invalid data in other processors private caches.

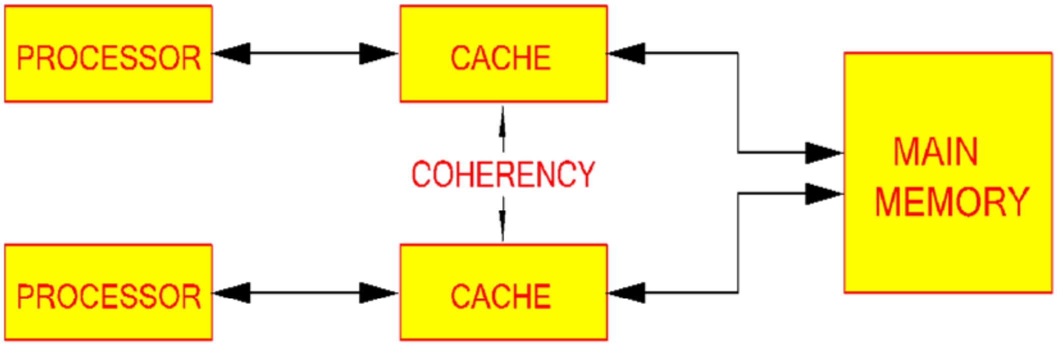


Figure 1.10: Cache coherency

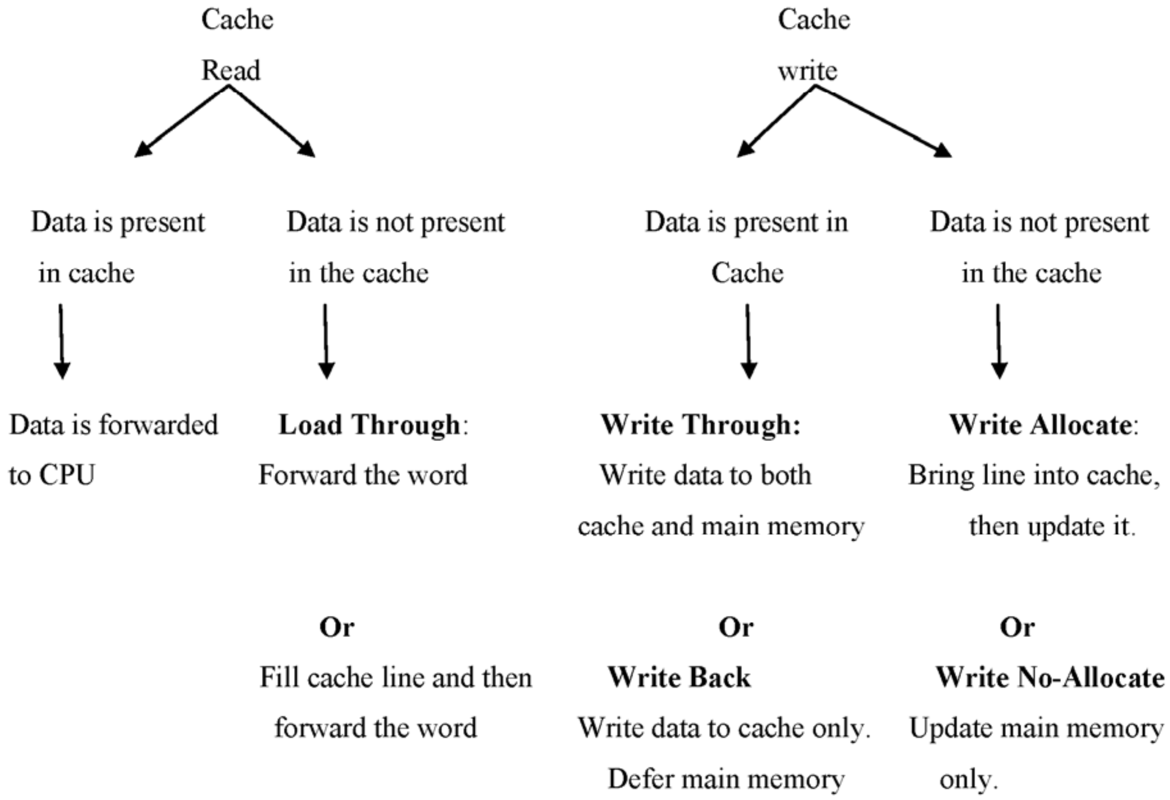

Cache Read
Cache write
Data is present in cache
Data is not present in the cache
Data is present in Cache
Data is not present in the cache
Data is forwarded to CPU
**Load Through**: Forward the word
**Write Through:** Write data to both cache and main memory
**Write Allocate**: Bring line into cache, then update it.
**Or**
Fill cache line and then forward the word
**Or**
**Write Back** Write data to cache only. Defer main memory
**Or**
**Write No-Allocate** Update main memory only.

Figure 1.11: Types of cache access and their methods

### 1.6.4 Power Consumption [M. Kandemir, 2009]

It is important for system availability to minimize power consumption by the system. Power consumption is one of the most significant factors that differentiate an MPSoC-based system from all other equivalent systems. Increasing the number of processors as in MPSoC means increase in the power consumption by processors with their local caches. Hence, this clearly leads to higher power consumption.

### 1.7 SPM

Scratchpad memory (SPM) is a small on chip memory with fast speed [R. Banakar, 2002]. Scratchpad is a memory mainly used to store small data items to increase to retrieval time and hence it is called as special high-speed memory.

SPM is located next closest to CPU apart from its internal registers and hence can be considered similar to the L1 cache and it uses DMA to transfer data between memory and CPU as shown in fig. 1.12. There is a difference between memory access latencies of main memory and scratch pad memory as a system with scratchpads is also called as a system with non-uniform memory access latencies. A scratch pad usually does not contain the data stored in main memory while cache memory always contains the replicate data.

Introduction of SPM simplifies the caching logic as it stores the intermediate results in it and only in case of multiple processors in MPSOC and these results are not always be taken to the main memory.

### 1.8 Dissertation Contribution and Outline

In this dissertation, the possible solutions to the cache coherence problem in multiprocessor systems have been proposed. The thesis has been organized into 11 chapters and out of that chapter 1 present's introduction comprises problem statement and contribution. Chapter 2 presents history of problem stated and their already given solutions.

In chapter 3, implementation of semaphore scheme to synchronize different shared cache memories in an MPSoC is proposed. Furthermore, proposal of using semaphores for processor synchronization is evaluated and proves better than locks and transactions.

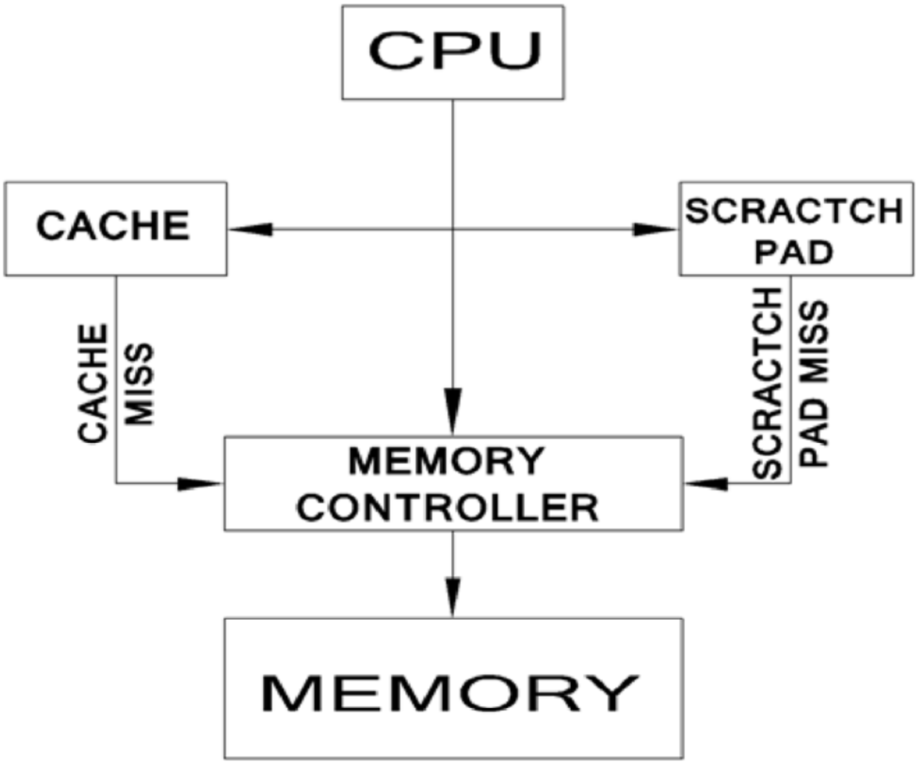


Figure 1.12: SPM and cache organization in SoC

Chapter 4 describes a simulation based performance evaluation of some memory design issues like associativity and replacement policy in cache and scratch-pad memory (SPM). Moreover, three common cache replacement policies FIFO, LRU and Random in SPM is implemented and evaluated the finest replacement policy in SPM for embedded systems. Moreover, a comparison study between use of SPM in the system and a system without SPM is also performed. LRU outperforms among three replacement policies in both the environments. An overall decrease of 61% in cache miss rate is achieved with the use of SPM in the system taking same environment for all evaluations.

Chapter 5 designs a multiprocessor shared memory simulator named Memory Map. This simulator allows user to specify its own runtime cache configuration and also can vary the number of processors within the application program. This simulator can find only cache miss and cache hit rate for any configuration of the system.

A new highly efficient cache replacement policy called tag based dual mapping replacement policy (TDMRP) is proposed in chapter 6. It has also been compared with two existing policies i.e. FIFO and LRU in terms of cache hit ratio and energy consumed taking 2-way associative cache using extended Simple Scalar and CACTI power model. Simulation results on SimpleScalar demonstrate an increase of 7% cache hit ratio than in LRU and 13% to FIFO. In the similar manner, TDMRP saves 48% of energy consumed in case of LRU and consumes 35% less energy as compared to FIFO. Hence, TDMRP proves out to be the outstanding option in case of replacement policy for 2-way and more associative caches.

With the implausible rise in microprocessor technology having high speed processors and enhanced the processor-memory speed gap, the design of on chip memory hierarchy is a momentous concern in embedded systems. Chapter 7 describes a simulation based performance evaluation of some design issues like associativity and replacement policies of cache memory vs. scratch-pad memory (SPM). In addition, the use of new cache replacement policy, called tag based dual cache replacement policy in SPM is implemented and access the best approach for replacement in SPM for embedded systems. A considerable decrease in cache miss rate has been evaluated by the use of tag based dual replacement policy in comparison to some other policies like LRU, FIFO and Random under same environment.

The major issue in integrating heterogeneous processors with their private memories on a single chip in a MPSoC is to keep the updated data in caches. In chapter 8, a record based cache coherence protocol is implemented to make caches coherent in heterogeneous MPSoC. Performance improvement up to 18.75% is achieved while using proposed approach in four different benchmarks on LIMES simulator as compared to three other snoopy cache coherence protocols Dragon, Berkeley and MESI.

Chapter 9 describes the design of Fraction associative cache, which minimizes the conflicts that arise in direct-mapped accesses by reserving fractional space for conflicting locations; however, it does not affect primary mapped address. Thus this scheme exploits temporal locality without disturbing the spatial locality and at the same time it does not result in under-utilization of memory reserved for resolving conflict misses.

In chapter 10, the effect of increasing network load on throughput and average packet delay in different NoC topologies with 10 bit and 32 bit flit length is described and evaluated. Moreover, this chapter shows the effect of network size (No. of IP nodes) on performance metrics throughput and average packet delay.

Chapter 11 concludes this dissertation with a summary and discussion of future directions.

# CHAPTER 2

## LITERATURE REVIEW

2.1 Cesare Ferri, R. Iris Bahar and Maurice Herlihy. Energy efficient synchronization techniques for embedded architectures. Proceedings of the 18th ACM Great Lakes symposium on VLSI, pp 435-440, GLVLSI, 2008.

Conclusion: In this paper, authors have discussed transactional memory and distribution semaphores which are two hardware implementation techniques for obtaining energy-efficient synchronization for embedded systems. Authors in this paper have proposed a modification to the earlier used transactional hardware scheme. The proposed scheme clears the contents of the transactional cache into the memory present next in the memory hierarchy. They also proved a 17% decrease in energy over a traditional transactional memory implementation. However, authors have also shown that shutting down the TC can be counter-productive in cases where it leads to high bus traffic; therefore it is necessary to enable this "aggressive shutdown" policy adaptively according to application characteristics. For implementation of distributed semaphores to support conditional synchronization, it has been found that in most cases workloads using these distributed semaphores (sometimes in conjunction with transactional memory) were substantially more energy efficient than the same workloads using locking. Much of this benefit is due to a reduced load on the system bus. Finally they propose to perform wider experiments on more range of benchmarks.

2.2. Tali Moreshet, R. Iris Bahar and Maurice Herlihy. Energy-aware microprocessor synchronization: Transactional memory vs. locks. Workshop on memory performance issues held in conjunction with the International Symposium on High-Performance Computer Architecture, Austin, TX, February 2006.

Conclusion: in this paper, authors compared transactional memory to locks. When rare conflicts, transactions have an advantage over locks in terms of performance as well as energy due to fewer accesses to main memory. As conflicts become more common, however, the cost of transaction re-executions becomes a significant drawback in terms of energy as compared to locks. By simulation of several SPLASH-2 benchmarks, authors were unable to produce high contention levels. Therefore, to investigate the behaviour of high-contention applications, they formulate a simple synthetic benchmark in which the level of contention can be easily "tuned" to different levels. Then they used this benchmark under various configurations to test the high contention mode for transactional memory. Their results open a range of further issues. They prove that neither transactional memory nor locking code was designed with energy consumption in mind. Moreover, they propose a promising energy-aware approach for handling synchronization in shared-memory multiprocessors: use speculative synchronization via transactions for the majority of low-contention program executions, switching to enforced serialization when contention is high. Forcing serialization may reduce overall system throughput, however is advantageous in terms of energy.

2.3. Ilya Issenin, Erik Brockmeyer, Bart Durinck and Nikhil D. Dutt. Data-reuse-driven energy-aware co synthesis of scratch pad memory and hierarchical bus-based communication architecture for multiprocessor streaming applications. IEEE transactions on computer-aided design of integrated circuits and systems, vol. 27, no. 8, august 2008.

Conclusion: The increasing use of MPSoC’s places a big burden on system designers to evaluate customized memory and communication architectures having different power, cost, and performance tradeoffs. In this paper, authors have presented a novel multiprocessor data reuse analysis technique for SPM that opens up a large space of unexplored options for memory customization. Moreover, they presented several techniques aimed at memory and communication system synthesis. Traditionally, the memory and communication subsystems were designed and sequentially optimized, potentially missing out on a large number of good design points. Hence, authors proposed a novel approach for MPSoC memory an energy-aware co synthesis based on data reuse information and an architecture communication template for architecture with hierarchical buses with TDMA arbitration. They proposed a template for a data-parallel partitioned application and suggested several ways to solve the co synthesis problem—optimally by an MILP solver or by a heuristic. Lastly, they compared these two approaches, as well as against a traditional two-step synthesis technique that first determines memory configuration and then performs communication synthesis. According to their results, an optimal MILP solution takes a reasonable amount of time for systems with up to 16 processors and provides results which are at max 50% better than the results from the other two approaches. Additionally, while providing 17%, on average, worse results than the optimal MILP technique, heuristic achieves near-optimal results on some of the benchmarks with much smaller execution times. If the MILP solution does not terminate in an allotted amount of time, the designer can choose to use the best of the results obtained by heuristic and the two-step technique.

2.4.Med Aymen Siala, Slim Ben Saoud. A Survey on Existing MPSOCs architectures. International Journal of Computer Applications (IJCA), Volume 19, issue 3, April 2011.

Conclusion: In this paper, authors have presented a generic study on different MPSOC's aspects. First of all, they discussed about topologies and communications inside the chip. In this first section the presented different types of interconnections, from the very old (the not communicating processors, the point to point communications or the bus), to the most recent ones (the NOC's). In the second part of this survey, a brief introduction to "Globally Asynchronous Locally Synchronous" system was done, as this type of systems is very used for MPSOC's researches and industrial world examples. Finally, a view on different memory organizations of MPSOC's has been presented, since memories represent a real challenge for futures MPSOC's. After this study, future works can be devoted to give some solutions to actual existing problems. Among these problems memory organization strategies which present a big challenge, face to the huge increasing of the computing capacities which hasn't been followed by an equivalent amelioration in memories latencies.

2.5.Dr. Gheith Abandah, Asma Abdelkarim. Performance evaluation of recently proposed cache replacement policies. CPE 731: Advanced computer architecture, University of Jordan Computer Engineering Department, January 19, 2010.

Conclusion**:** In this simulation experiment five SPEC SPU2000 benchmarks were simulated for three of the recently proposed replacement policies. The benchmarks are: ammp, art, bzip2, equake and parser. The replacement policies are: MLP-aware, DIP and Adaptive (LRU-LFU) insertion policy. The results showed that adaptive policies can significantly improve the performance of the L2 cache for memory intensive workloads for which LRU has bad performance. Each of the simulated replacement policies has its own way in improving performance for these workloads. What makes adaptive policies appealing is that they maintain approximately equivalent performance for LRU-friendly workloads while achieving this improvement. The MLP-aware replacement policy and DIP use distinct approaches in improving the performance of the caches; the MLP-aware replacement policy improves miss penalty by exploiting memory level parallelism while DIP improves the miss rate by preventing thrashing of the cache. Combining these two ideas may combine the improvements of these two replacement policies to achieve even more and more performance improvement.

2.6. Mohsen Soryani, Mohsen Sharifi, Mohammad Hossein Rezvani. Performance evaluation of cache memory organizations in embedded systems. International Conference on Information Technology (ITNG'07), Las Vegas, Nevada, USA April, 2004.

Conclusion: Authors in this paper have main concern on cache memory organization. So, they performs evaluation to find main cache design issues like cache size, associativity and replacement policy in embedded processors. They choose some application from SPEC CPU2000 benchmark suite based on eccentricity and clustering concept. The results show that the gain of increasing associativity in the case of data and unified caches is more than instruction caches. In L1 data cache and L1 unified cache the largest miss rate reduction occurs for transition from a direct mapped to a 2-way set associative. In floating point applications, for large caches associativity higher than two does not efficiently reduce the miss rate; however for small caches the amount of reduction in miss rate is noticeable.

Typically, for larger data cache sizes random policy dominates, while for smaller cache sizes FIFO dominates. However in general for L1 data cache it is hard to select a winner replacement policy between FIFO and Random policies, and the difference between them decreases as the cache size increases. For large caches the MRU policy is a good approximation of LRU policy. Compared to LRU policy, MRU has less complexity and according to results has negligible miss rate degradation. According to their experiment results, the performance of OPT policy is near the performance of MRU of cache twice as big. Hence, eliminating this gap will reduce the size of caches even to one half of their current sizes.

2.7.G. Chen, O. Ozturk, M. Kandemir and M. Karakoy. Dynamic scratch-pad memory management for irregular array access patterns. Proceedings in Design, Automation and Test in Europe, DATE '06, Volume 1, pp 168-175, March, 2006.

Conclusion: Scratch-pad memories (SPMs) are being increasing used in embedded systems and recent research has studied several compiler optimizations, designed specifically for these software-managed on-chip memories. Most of these techniques are able to handle only applications with regular data access patterns. Unfortunately, not all embedded applications exhibit only regular data access patterns (e.g., some codes from embedded multi-media) and requirement of compiler techniques to optimize their behaviour when they need to be executed in the SPM-based architectures. Authors address this problem by proposing a scheme that enlists both compiler's and runtime system's help. They perform experiments with seven embedded applications dominated by irregular data access patterns which show that this hybrid scheme is very effective in practice. They prove by results that this approach is very successful with the applications that have irregular patterns and improves their execution cycles by about 54% over a state-of-the-art SPM management technique and 23% over the conventional cache memories. Also, the additional code size overhead incurred by the approach is less than 5% for all the applications tested.

2.8. Chuanjun Zhang and Bing Xue. A tag-based cache replacement.Computer Design (ICCD), IEEE International Conference in Amsterdam, pp 92-97, Oct 2010.

Conclusion: Authors proposed the tag-based replacements that share the status bits of cache blocks that have the same tag and use both the frequency and access information of tags instead of cache blocks. The experimental results show that the tag based replacement significantly reduces average miss rate at cache capacities of 512KB, 1MB and 2MB. The processor performance improvement using the tag-based replacement is up to 40% with an average of 4.5% over the block-based LRU. The performance improvement is achieved with significantly less hardware compared to conventional LRU, LFU, and their derivatives. In future, they propose to include the implementation of other block based replacements by using tag-based techniques and the development of new tag-based replacements for multi-core processors. One important difference between the tag-based replacement and conventional block based replacement is that the tags that have been evicted from the cache are still tracked. The tag-based replacement reduces the average miss rate of the baseline 1MB L2 cache by 15% over conventional LRU with 95% status bits reduction over conventional LRU. The performance improvement of a processor using the tag based replacement is up to 40% with an average of 4.5% over LRU.

2.9. Bahman Hashemi. Simulation and evaluation snoopy cache coherence protocols with update strategy in shared memory multiprocessor systems. Proceedings of the 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications Workshops, ISPAW 2011, pp 259-265, IEEE Computer Society Washington, DC, USA, 2011.

Conclusion: According to their results, WTU protocol has low performance however Firefly and Dragon have high performance, because precision block sharing information in WTU protocol is low however in Firefly and Dragon is high, therefore precision block sharing information in multiprocessor systems is the major factor for different performance of update snoopy cache coherence protocols. Therefore if information amount is to be high then performance will be high; and hence implementation cost will be high as well. Therefore in multiprocessor systems, selection of cache coherence protocol is important decision that can make meaningful effect on cost and performance of shared memory multiprocessor systems.

2.10 Samaher Al-Hothali, Safeeullah Soomro, Khurram Tanvir and Ruchi Tuli. Snoopy and directory based cache coherence protocols: A critical analysis. In Journal of Information & Communication Technology Volume 4, No. 1, Spring, 2010.

Conclusion: If enough bus bandwidth is available, then all transactions are a either request or response messages observed by all processors and hence snoopy protocols perform faster than other protocols. Snoopy protocols have one disadvantage of non scalability. Each and every request must be broadcasted in a system, so that with the increase in system size, the logical or physical bus and the provided bandwidth should also grow. On the other hand, in directory based protocols messages are point to point and hence requires less bandwidth and have less tendency to have higher latencies. Protocols for cache coherence are critical to multiprocessor systems. In general, the directory based protocol is more used for larger systems to increase their performance; while snooping protocol is used for smaller systems.

Snoopy protocols have low average miss latency, especially for cache-to cache misses. However, Directory based protocols scale much better than snoopy protocols. They have the ability to exploit arbitrary point-to-point interconnects. The directory access and the extra interconnect traversal is on the critical path of cache to cache misses. It involves the storage and manipulation of directory state.

2.11. P. Ezhumalai, A. Chilambuchelvan and C. Arun. Novel NoC topology construction for high-performance communications. Journal of Computer Networks and Communications, Hindawi Publishing Corporation,Volume 2011, Article ID 405697, 6 pages, March, 2011.

Conclusion: In this paper authors proved EBFT as the best topology due to least packet latency and highest throughput. Moreover, authors have compared it with mesh, torus and BFT NoC topologies. BFT comes second in this comparison of topologies. Although BFT and EBFT are performing better than mesh and torus topologies, they suffer from backbone link breaking problem. Their proposed solution provides the decoupling of the evaluation cost function from the exploration function, thereby enabling different user objectives and constraints to be considered.

2.12. Taehwan Cho and Sangbang Choi. A multi-path hybrid routing algorithm in network routing. International Journal of Hybrid Information Technology Vol. 5, Issue 3, July, 2012.

Conclusion: In this paper, authors have presented a Multi-Path Hybrid Shortest Path Tree (MP-HSPT) algorithm and it offers an efficient shortest path decision that can be used to reduce the total execution time using the multi-path information. The total execution time declined and hence also leads to reductions in packet losses. The proposed MP-HSPT algorithm provides better performance when compared with the well known methods like Dijkstra, Dynamic Dijkstra, and HSPT in terms of computation time for the shortest path. Their comparison results with other routing algorithms prove that the multi-path hybrid routing algorithm provides better performance due to decrease in the execution time.

# CHAPTER 3

# A SEMAPHORE IMPLEMENTATION FOR PROCESSOR SYNCHRONIZATION WITH SHARED MEMORY IN MULTIPROCESSOR SYSTEM-ON-A-CHIP

## 3.1 Introduction

A System-on-a-chip is basically a chip which consists of some components like processors, memory etc communicated through a network called NoC. It can perform different functions like digital, analog and mixed-signal functions. Embedded system is one of the major application of SoC's. With the increasing demand of compact and fast system results in the development of integrating multiple processors on a single chip and it is called multiprocessor system on chip (MPSoC). The multiprocessor System-on-a-chip [S. Pasricha, 2007] is defined as a special system-on-a-chip (SoC) which has multiple processors with their private memories and a shared memory is present on a single chip. They have major application in the field of embedded systems which require a fast, efficient and compact system.

The various components present on an MPSoC chip are a number of homogeneous or heterogeneous processors, two or more levels of memory hierarchy and also peripheral components. An on-chip interconnects for example AMBA is used to provide connections between all these components. Each component has its own utility.

Shared cache is the mode of communication between multiple processors on a single chip [D. Cho, 2009]. Parallel execution of multiple cores is the main requirement for embedded applications [I. Issenin, 2008]. SoCs with tens of cores are ordinary [S. Borkar, 2007], [M.A.J. Jamali, 2009], [P. Ezhumalai, 2009], [M. Ali, 2005] and platforms with hundreds of cores have been already announced. There are numerous benefits of using systems with multiple cores: Low power consumption, easy to scale and also faster execution time.

One of the most important and strong problem is how to implement parallelism in systems with multi-core architects. After the evolution of threads, a way to parallelism in MPSoC seems clear [J.W. Chung, 2006] because of the possible parallel execution of multiple threads. The actual benefit of implementing multiple processors on a single chip is parallel execution [I. Issenin, 2006]. There are already a lot of solutions provided by some authors to solve this problem of multiple processor synchronization which are going to be discussed in next section.

### 3.2 Literature Survey

The general approach used for solving synchronization problem is the use of locks [C. Ferri, 1993]. A lock on a data item is acquired by a processor if it wants to access that data item. Locks are exclusive in nature i.e. no two processors can simultaneously lock same data item. However locks are used worldwide, still they have some disadvantages. The first disadvantage is that locks cause more thread delay and sometimes failure also. The reason of thread delay is context switching. Hence, with the hindrance in freeing the thread by a lock, other concurrently executing threads waiting for the data to be free may also be blocked. Secondly, locks also obstruct concept of concurrency because locks must be acquired conventionally means a thread must always acquire a lock even if there is no chance of conflicts.

Authors in [T. Moreshet, 2006] evaluated and compared the energy consumed by transactional memory and locks which are well known solutions to multiprocessor memory synchronization problem. According to their results, these two synchronization techniques consume a lot of energy as it depends upon the conflict rate. According to their results, both use of transactional memory and use of locks are consuming a lot of energy which is not good for embedded systems. Hence, there is a need to develop an energy efficient synchronization approach.

C. Ferri and other authors in [C. Ferri, 2007] also compared locking mechanism with transaction based synchronization approach on the basis of frequency, power numbers and architectural assumptions which are further based upon simple cores used for an embedded system with multiple processors. They prove the advantage of transactional memory over locks in case of performance by their simulation results and also they prove that energy constraints of such embedded systems can only be achieved through vigilant hardware design. The graphs drawn on the basis of results are the proof of benefits in terms of energy consumption by the system as well as the easiness in coding. However, they show a limitation in terms of their applicability to multi-core embedded systems.

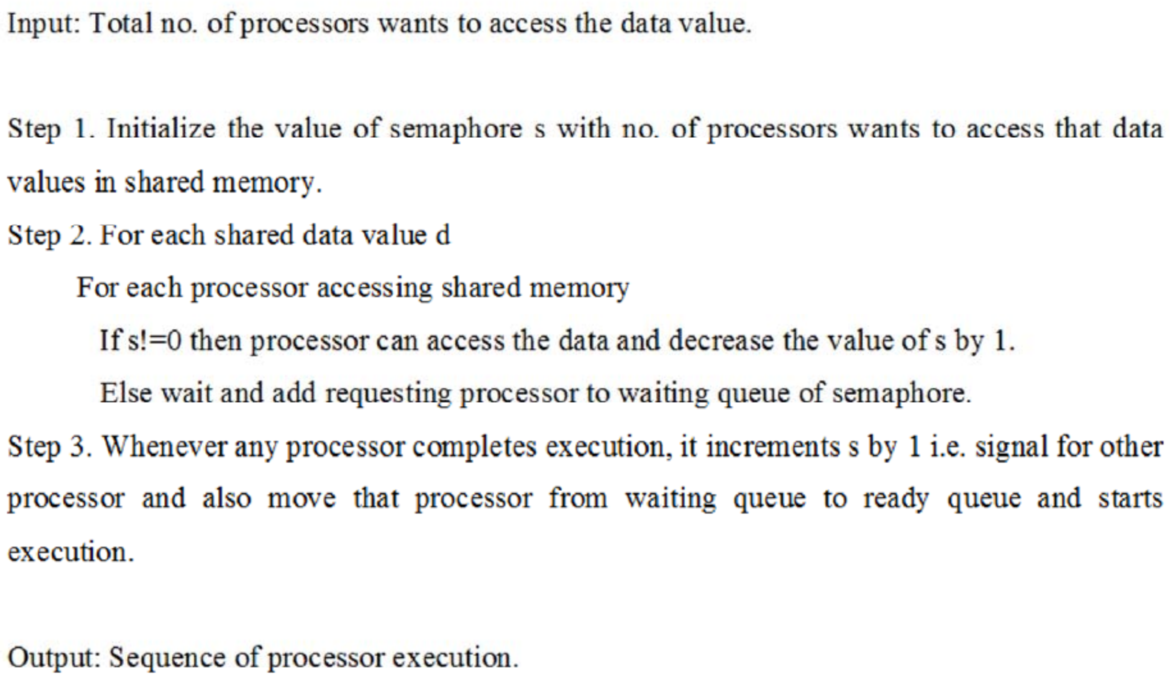
In [R. Bahar, 2008] authors implemented distributed semaphores and transactional memory approaches to evaluate their energy consumption. Their results clearly prove that these two approaches were not energy efficient due to some constraints on such systems as compared to general purpose systems. Hence, they proposed an enhancement to transactional hardware approach to solve this issue. In this approach, the contents of the transactional cache were flushed into the memory present next in the cache hierarchy. They prove a saving of 17% by using this modified approach over conventional transactional memory implementation. Although their mechanism proves an energy efficient solution in comparison to locking, still there does not exist any scheme which proves to be best in every situation.

Researchers have given many solutions to solve the problem of multiprocessor synchronization in an MPSoC like locking, transaction based synchronization [M. Loghi, 2006] and many more. Still there were a lot of drawbacks like some have high energy consumption; some have large cache miss rates and high CPU cycles etc. Hence, this problem still remains unsolved. In the next section, a new synchronization technique is proposed which tries to address these problems.

### 3.3 Semaphores

Most of the system energy gets wasted in the frequent transactions abort and restart operations which occurs due to the synchronization conflicts [E.W. Dijkstra, 1971]. Hence, motivating by this problem, authors proposed an energy efficient and high performance solution of implementing the concept of semaphores. A system used for sending signals by using two flags that are held in hands. This concept was first given in 1893 and used for signalling the trains. In computer science, a semaphore is a variable of integer data type. It is can only be accessed through wait and signal operations of semaphore in atomic manner. Moreover, the semaphore value can only be altered by wait and signal operations.  Signal operation incremented the value of semaphore by one while wait operation decrements the value by one. Unlike locks, semaphores are also exclusive in nature i.e. not more than one processor can simultaneously modify the same semaphore value. If a processor wants to access a data item which is currently accessed by another processor in shared memory, then the processor must wait for the other processor to complete the execution. This is called wait operation. A queue of waiting processors for the semaphore value is maintained in FIFO manner and the blocked processor is placed into that waiting queue. The status of the processor is set to waiting state.

A signal operation switches the processor from waiting to ready. The processor is now present in the ready queue i.e. ready to execute.  The algorithm explained above is listed below in steps in fig. 3.1. A counting semaphore is used. Its value has been initialized with maximum no. threads (processors) generated.



```
Input: Total no. of processors wants to access the data value.

Step 1. Initialize the value of semaphore s with no. of processors wants to access that data values in shared memory.
Step 2. For each shared data value d
        For each processor accessing shared memory
           If s!=0 then processor can access the data and decrease the value of s by 1.
           Else wait and add requesting processor to waiting queue of semaphore.
Step 3. Whenever any processor completes execution, it increments s by 1 i.e. signal for other processor and also move that processor from waiting queue to ready queue and starts execution.

Output: Sequence of processor execution.
```

Figure 3.1: Semaphore algorithm

A threads increments the semaphore value by one if a processor requests access to the resource and decrement the value of semaphore by one if one processor executes signal operation i.e. completes execution.

### 3.4 Simulation Environment

When the processor requests a data access, it is firstly get checked in cache memory and if data is present in cache memory then the processor reads data from cache or writes data to the cache instantly [D.H. Albonesi, 1999], [R.I. Bahar, 1998], which is much quicker than reading to and from main memory. Similar is the case with writing operation.

There are three different categories of cache memory : [H. Akkary, 2003] an instruction cache to increase the speed of fetch instruction, [R. Fromm, 1997] a data cache which speeds up fetching and storing of data and a translation look aside buffer (TLB) [L. Hammond, 2012] used for converting virtual address generated by processor to physical address of memory for both instructions and data. In this implementation, both data cache and TLB cache has been considered. In a MPSoC, there are processors own private memory and also common shared memories for all processors [M. Huang, 2000].

The evaluation environment consists of four processors on a chip with their own private memories and also two shared memories attached to each processor. ARM bus is used for communication between processors and memory as shown in fig. 3.2.

Threads are used for implementing multiple cores on a chip. The system consists of 4 cores, each having direct-mapped D1and T1 cache memory, a set of 64 KB private memories for each processor and also two 64KB shared memories, all connected through an interconnect [S. Mittal, 2011]. Linux operating system is used for this implementation.

Simple Scalar functional simulator [D. Burger, 1997] is used to implement the above organization. The Simple Scalar tool set is used for performance analysis of process in execution and opportunity to design modeling applications on it. This tool set includes a number of sample simulators like sim fast to evaluate energy consumption, sim-cache to implement memory hierarchy and evaluating a number of parameters like cache miss rate and many more. This tool provides the facility to develop your own simulation environment.

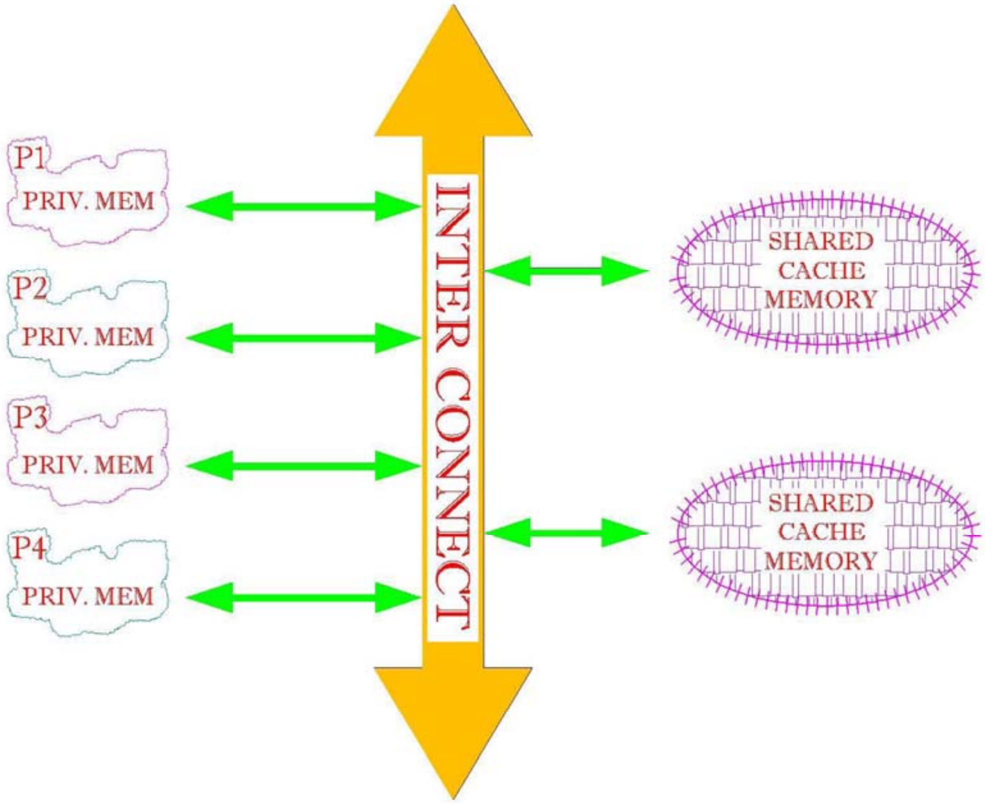


Figure 3.2: Architecture implemented on simulator for evaluation of locks, transactions and semaphores.

The original tool sim-fast needs to be customized to evaluate energy consumption [M. Herlihy, 2003] by the system and sim-cache tool is used to find cache miss rate for various benchmarks and synchronization techniques. The total energy consumption is calculated as $E= E_{cache}+ E_{Dcore}$ where,

$E_{cache}= E_{caccess}$ * No. of cache access.

$E_{Dcore}=E_{daccess}$ * No. of D core accesses + $E_{trans}$ * No. of transactions + $P_{idle}$* $T_{idle}$

No. of D core access= No. of cache access / block size

No. of transactions= 2* No. D core accesses

$T_{idle}$= Applications execution time – time spent accessing data from DRAM core and transitioning between power states.

The final value taken is the average of ten simulation runs. These benchmarks were chosen to run longer simulations so that better and real results can be taken.

### 3.5 Experimental Results

This work evaluates and compares three synchronization approaches in terms of energy consumption and cache miss rate using three benchmarks: FFT, Splash 2 and Red black trees. FFT is Fast fourier transform and it is an algorithm to compute discrete fourier transform and its inverse. A fourier transform converts time to frequency and vice versa. Splash 2 is a collection of 11 multithreaded workloads. The bars labeled with locks in fig. 3.3 show the energy consumption in µJ for each of the benchmarks taken in consideration. The bar graph displays the energy consumption for transactions as well as for semaphores using same benchmarks. As per the results, locks are consuming maximum energy because every time processor wants to access a data item, it has to lock it and hence number of locks increases, energy consumption also increases. Transactions consume less energy for these benchmarks as compared to using locks. Semaphores have approximately analogous energy consumption as that of transactions.

The cache miss rate for each of the technique is shown in the bar graph of fig. 3.4. For all benchmarks, transactions prove better than locks in case of cache miss rate. Transactions have lower cache miss rate as compared to locks. And semaphore comes out best among locks and transactions with lowest cache miss rate. For splash-2 micro-benchmark, running transactions without semaphore generates more re-executions, which finally resulted in more conflicts and hence high cache miss rate which eventually leads to consumption of more CPU cycles and hence more energy consumed. Table 3.1 and table 3.2 describe the tabular representation of experimental results.

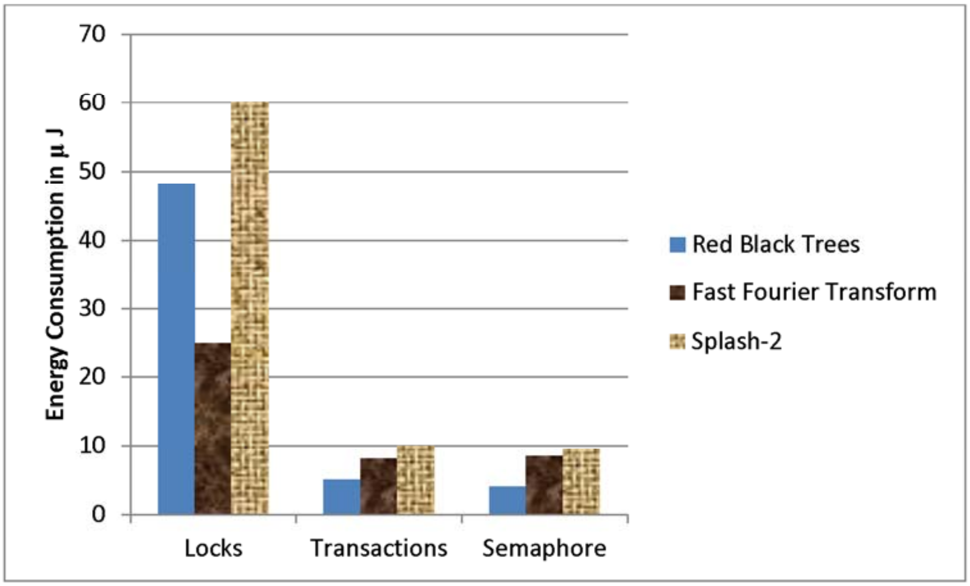


Figure 3.3: Energy consumption by the use of locks, transactions and semaphores

Table 3.1: Energy consumption in µJ by the use of locks, transactions and semaphore

| | Locks | Transactions | Semaphore |
|---|---|---|---|
| Red Black Trees | 48 | 5 | 4 |
| Fast Fourier Transform | 25 | 8.5 | 8.5 |
| Splash-2 | 60 | 10 | 9.5 |

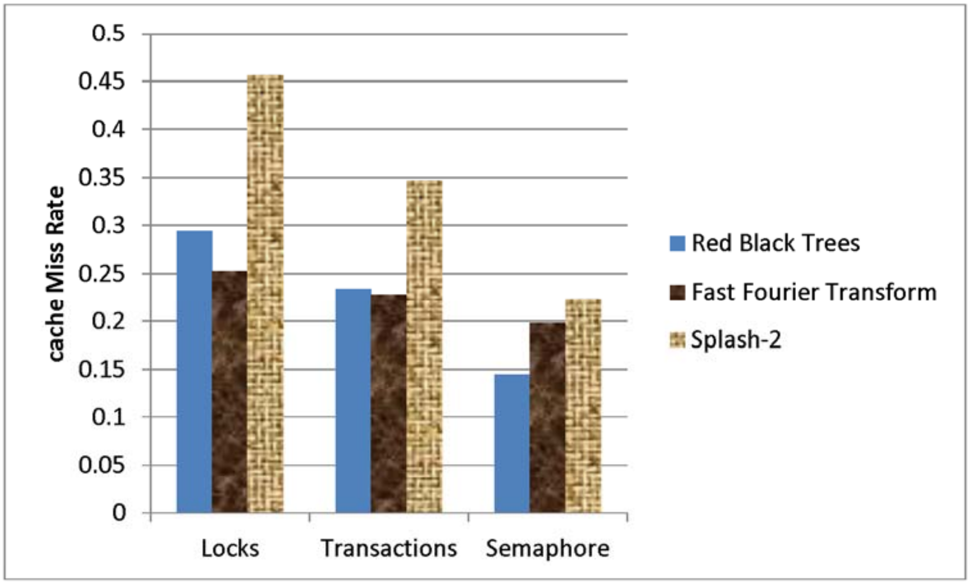


Figure 3.4: Cache miss rate by the use of  locks, transactions and semaphores

Table 3.2: Cache miss rate obtained using locks, transactions and semaphore

| | Locks | Transactions | Semaphore |
|---|---|---|---|
| Red Black Trees | 0.294 | 0.234 | 0.145 |
| Fast Fourier Transform | 0.253 | 0.228 | 0.198 |
| Splash-2 | 0.456 | 0.347 | 0.223 |

## 3.6 Conclusion

Two processor synchronization techniques transactions and locks are compared to semaphore. Semaphore outperforms locks and transactions. Number of locks attained by the processor is directly proportional to the cost of synchronization when using locks. Hence, more the number of locks acquired, less the performance. Similarly in case of transactions, the cost depends on the conflict rate. And in case of semaphore, the cost of synchronization is directly proportional to the processor waiting time. So, lesser the processor waiting time, better the performance. The energy consumption for locks and transactions depends upon a number of parameters like system configuration, conflict scenarios, types of locks and also transactional memory hardware.

The results prove that semaphore outperforms with lowest cache miss rate. And in case of energy consumption, semaphore is almost comparable to transaction however much better than locks because the owner for calls to lock and unlock is same thread, hence consuming more energy in switching. However calls to P and V in semaphores can be made by different threads and consumes less energy.

Future work will improvise these results with implementing some more constraints in semaphores so that it comes out to be best energy efficient technique.

The contribution towards this research is published and is as follows:

Shaily Mittal and Nitin, *A Resolution for Shared Memory Conflict in Multiprocessor System-on-a-Chip*, International Journal of Computer Science Issues, Volume 8, Issue 4, Number 1, ISSN: 1694-0814, July 2011, pp. 503-507. [IF: 0.242 (2011), Indexed in DBLP]

# CHAPTER 4

# REPLACEMENT POLICIES FOR SCRATCH PAD MEMORY IN EMBEDDED SYSTEMS

## 4.1 Introduction

An embedded system is a computer system designed to do one or a few dedicated functions with real-time constraints. Embedded system is part of a complete device often counting hardware and mechanical parts. The applications of such systems are in the fields of hard real time systems like automotive, aeronautics, electronics and industrial automation [J. Reineke, 2006]. The major constraint in designing an embedded system is to achieve high speed and small size. Compact systems perform better than huge and bulky systems. Hence, in order to achieve better performance, instructions and data caches are embedded on a single chip now a day's. Scratch-pad memory (SPM) is a small size on chip memory which is currently used in almost all embedded systems [R. Banakar, 2002] [P.R. Panda, 1997].

Cache performance has major impact of cache associativity on it. Now days, systems are designed with cache hierarchy and increased associativity. When there is no space for the new main memory block in the cache memory i.e. all locations are full, then one of the old block from the cache memory needs to be swapped out to make space for the new block. The same replacement procedure is implemented for scratch pad memory in this work. The model is designed with scratch pad memory. Currently, embedded systems make use of various replacement policies such as LRU (Least Recently Used) [A. Kalavade, 2000], Random [J.L. Hennessy, 2003], FIFO (First in First Out), PLRU (Pseudo LRU) [M. A. J. Jamali, 2009] and N-HMRU [S. Roy, 2009].

In [S. Udayakumaran, 2003, 2006] authors had proposed a compile time SPM allocation approach which was tailored at the entry point of each loop or function. As in these papers their focus was only on stack data and global variables in the allocation of SPM, therefore they proposed an additional approach taking into account of heap data into the SPM allocation in [A. Dominguez, 2005]. In general words, the most significant conclusion made by them is that the SPM allocation should be adapted at the start point of each loop or function.

Sheng-Wei Huang and others in [S.W. Huang, 2009] had effectively developed a new allocation approach for the page-based SPM in embedded systems. Efficiency of the proposed approach was evaluated by simulating and comparing the execution of a set of programs. Performance evaluation results proved the effective enhancement in hit rate by the use of proposed approach. Thus, the energy-delay-product (EDP) values of the test programs were significantly improved by 60% in average.

The simulation-based performance evaluation in [M. Soryani, 2007] discussed some of the cache design issues like unified and split cache, cache size, cache associativity and cache replacement policies in embedded processors. Their evaluated results using SPEC CPU 2000 benchmark prove the gain of increasing associativity more in the case of data and unified caches than that of instruction caches. In addition, they prove random policy performed practically better than LRU and MRU for instruction caches. They also compared LRU to MRU policy and according to results MRU comes out with insignificant miss rate deprivation. Experiment results illustrated that the performance of OPT policy is near the performance of MRU of double cache size. They evaluated and compared replacement policies in cache memory only and not in SPM. Detailed values of the energy consumption of both architectures are important for a comparison between scratchpad memory and cache memory. This was done by Wilton et al. in [S. J. E. Wilton, 1994, 1996] who proposed a cache model named CACTI. The algorithm in [D. Cho, 2009] places selected program parts and variables into scratchpad memory as part of a compiler. The ILP model presented in this thesis seems to be an optimal solution and as it saves about 22% of the electrical energy as compared to a cache memory.

The main goal is implementation and performance evaluation of the memory design issues in embedded processors like cache associativity and replacement policies (FIFO, LRU and Random) in SPM which is never made earlier.

### 4.2 Problem Description

Cache associativity specifies how the data block of main memory is mapped to the cache memory locations. It is classified into three categories: Fully associative cache, direct mapped cache and n-way set associative cache. If a data block can be mapped to every location in cache memory, then it is called as fully associative cache memory. It is flexible, costly and has slow cache reads and writes. Direct mapped cache is also called direct mapped cache associativity. Direct mapping is a cheap operation as a data block all the time mapped into the same place in cache. Single association between cache and main memory diminishes the cache hit rate and increases contention problem as additional addresses compete for free cache lines. SRAM is an example of direct mapped cache. On the other hand, if a main memory block can be placed into any of the n cache blocks of a set, where n is in powers of 2, is known as an n-way set associative cache. Figure 4.1 describes the associativity in detail.

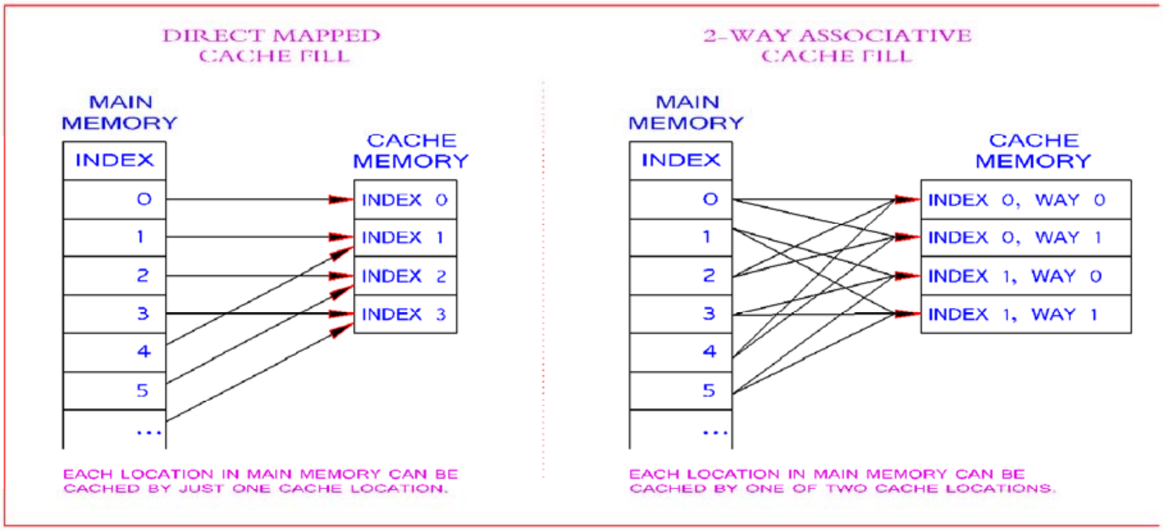


Figure 4.1: Cache associativity: Direct mapped and 2-way associative cache

Associativity is one of the issues which have been considered for work as it affects performance a lot [D. Cho, 2009]. The focus is to evaluate and compare influence of increase in associativity on cache miss rate. With the increase in degree of cache associativity, selection of an efficient replacement policy becomes more important.

Replacement policy is defined as the algorithm or way to decide the victim memory block needs to be replaced and save back to disk when a memory block needs to be allocated. This occurs on a cache miss and needs a free page to store the required page. If the memory block replaced earlier is referenced by the processor again then it has to be moved in the cache memory from disk increases waiting time of this I/O operation [A. Shaik, 2012]. This waiting time is the main factor for determining the quality of the page replacement algorithm. Less the waiting time, better is the algorithm. Traditionally, LRU policy is employed as replacement policy for most processors. LRU replacement policy works on the concept of FIFO in a different manner by maintaining a queue of length m, where m is the associativity of the SPM [J. Reinke, 2006]. In case of cache miss, the accessed data element is placed in the front of the queue and the last element from the queue is then removed as it was the least recently used element present in the queue. While in case of cache hit, the element which was earlier present anywhere in the queue is moved to the front in the queue. One weak point of this policy is it consumes a lot of time and power. Random policy is used to condense the cost of LRU, however its performance is also low [J.L. Hennessy, 2003]. In this policy, the sufferer line is chosen randomly from all the lines in the set. Another aspirant policy is FIFO (First In First Out), which can also be seen as a queue in which new elements are inserted at the front and expel element from the end of the queue.

Replacement policy is another important issue which affects the speed and performance of memory. Therefore, finding the best replacement policy for memories in embedded systems that is cache and SPM is the main focus. For that reasons, aim is to search suitable solutions for cache and on chip SPM memory problems.

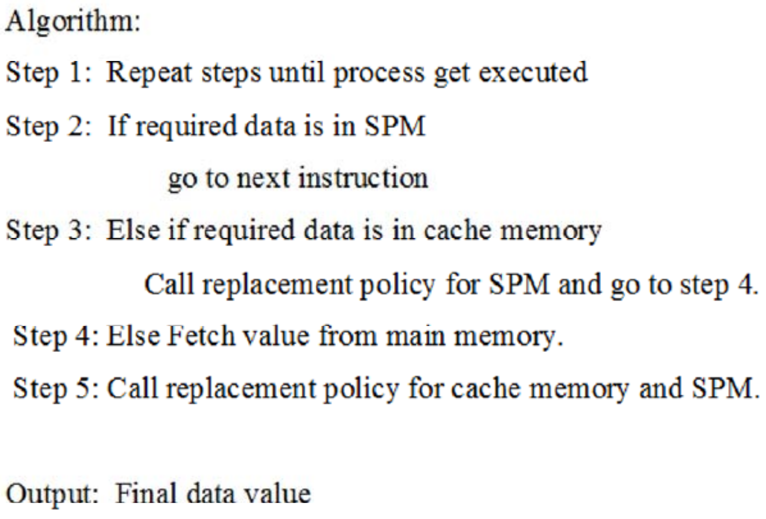
On chip memories are fast and energy efficient for data and instruction storage when compared to caches or external memories. Speed with low energy consumption is the recent and important need of applications to save time and resources [I. Issenin, 2008]. There is approximately 10 times extra consumption of memory cycles in case of cache miss rather than in case of cache hit which decreases the speed of the system. Hence cache miss rate have huge impact on speed and performance of the system. To keep this point in view, this chapter address the problem of reducing cache miss rate.

### 4.3 Proposed Solution

To address these problems an on chip scratch pad memory with three replacement policies in addition to cache memory is introduced. Size of SPM is taken very small as compared to cache memory. According to proposed algorithm, processor first checks the availability of required data in SPM and if the data is not present in SPM, it goes for cache memory. Existence of data in cache memory leads to cache hit otherwise a miss which further leads to retrieval of data from main memory along with replacement in cache memory as well as in SPM according to considered replacement policy. The applied algorithm is shown in fig. 4.2. This thesis has implemented replacement policies in SPM for the first time.

The work stream was used to compare different benchmarks such as sorting algorithms and some other applications stimulated from [S. Steinke, 2002]. These benchmarks require memory with high associativity or suffering from recurring conflict misses, and having low spatial vicinity or do good in case of extremely large caches.

Input: Address of data operand



```
Algorithm:
Step 1: Repeat steps until process get executed
Step 2: If required data is in SPM
             go to next instruction
Step 3: Else if required data is in cache memory
           Call replacement policy for SPM and go to step 4.
Step 4: Else Fetch value from main memory.
Step 5: Call replacement policy for cache memory and SPM.

Output: Final data value
```

Figure 4.2: Algorithm for proposed SPM replacement policy

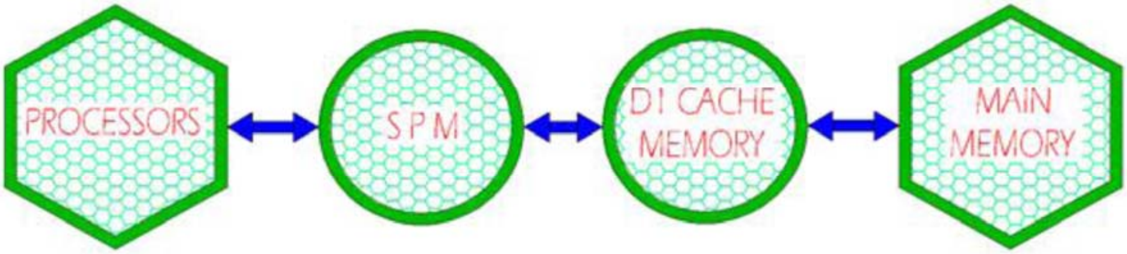


Figure 4.3: Two level architecture implemented for SPM replacement policy evaluation

### 4.4 Experimental Environment

The simulator used in this evaluation is SimpleScalar [D. Burger, 1997]. An infrastructure for simulation and architectural modelling can easily be provided by SimpleScalar. The toolset provide a mixture of platforms ranging from simple non pipelined processors to detail dynamically scheduled micro architectures with multiple-level memory hierarchies which have been used in study. Performance evaluation of FIFO, LRU and Random cache replacement policies has been done using alpha version of Sim-cache simulator of this toolset. For this work, two models were compared, one with a cache and the second with a scratchpad memory as well as cache. Sim-cache engine simulates associative caches with FIFO, Random, and LRU policies. The original simulator is modified to support hierarchical two level memory architecture as shown in fig. 4.3 as well as three replacement policies for newly employed SPM. The modification in simulator was to dump all memory accesses to a trace file. Then this trace file was analyzed to see how many memory accesses was there, what was the footprint, and how many accesses was into the statically declared array that represented scratch pad memory in the source code. Size of SPM was taken very small as match up to cache memory i.e. size of L1 data cache is taken as 4KB. For each benchmark, a number of simulations have been run for various D1data cache organizations with 1, 2, 4, 8 and 16 way associativity and replacement policies as random, FIFO and LRU.

In the similar manner, diverse simulations have been performed for SPM organizations of 1, 2, 4, 8 and 16 way associativity in conjunction with cache with different combination of replacement policy and associativity. A list of 1000 elements to sort in case of all sorting algorithms has been captured. In the analogous line of attack, Matrix multiplication act upon matrices of size 100 X 100 for large calculations. In signal processing, biquad is a second order recursive linear filter, containing two poles and zeros. In Z domain, its transfer function is the ratio of two quadratic equations.

### 4.5 Results

This work was used to compare cache with SPM using different benchmarks such as sorting algorithms. Cache miss rate values for matrix multiplication benchmark in table 4.1 show that LRU bear out to be the best replacement policy among three with minimum cache miss rate irrespective of using SPM.

To demonstrate and evaluate the best replacement policy graphs for associativity values 1 and 16 for both SPM and cache from the calculated values are shown in fig. 4.4 and fig. 4.5. It is well understood from the graph that least recently used (LRU) is the best policy among three as it has minimum cache miss rate.

The table 4.1 shows the average of cache miss rate using three replacement policies namely FIFO, LRU and Random in an embedded system using SPM and without using SPM. As it is clear from the results that cache miss rates decrease with the increase in associativity and moreover, effect of using scratch pad memory becomes clearer in a system of high value of memory associativity. Figure 4.6 displays the cache miss rate in biquad benchmark. As seen from the graph, fully associative mapped cache shows best results with lowest miss rate as compared to other associative caches. In comparison, cache miss rate with associativity value 2 using and without using SPM is drawn in fig. 4.7. In the similar approach, fig. 4.8 and fig. 4.9 shows the cache miss rate for associativity values 4 and 8 using and without using SPM. It is clear from these figures that cache miss rate decreases with the use of SPM as compared to without SPM.

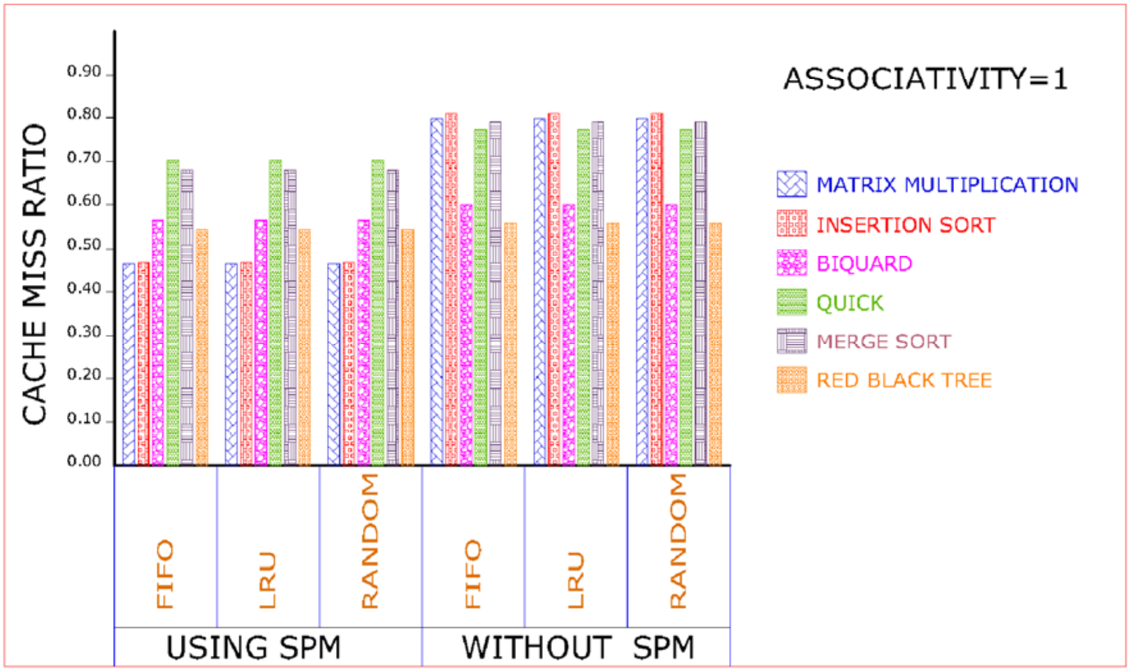


Figure 4.4: Direct mapped: cache miss rate with SPM and without SPM using 6 benchmarks

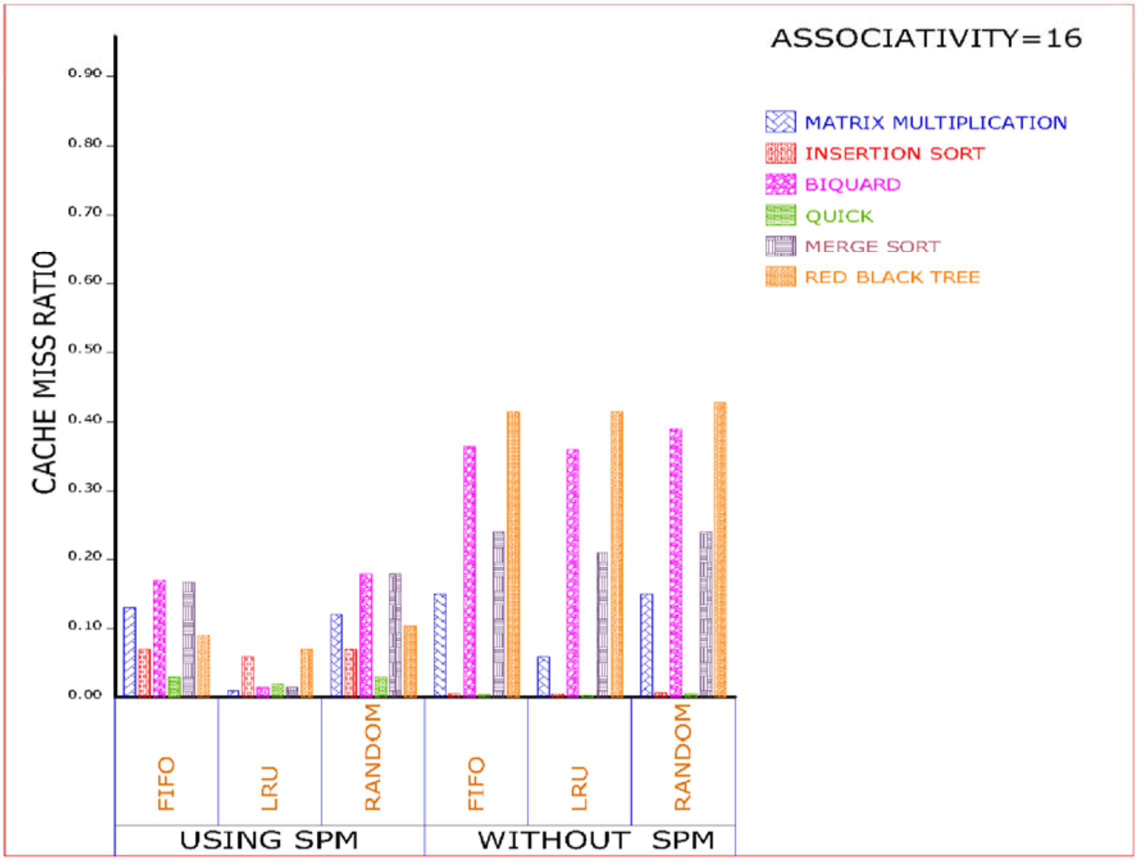


Figure 4.5: Fully associative: cache miss rate with SPM and without SPM using 6 benchmarks

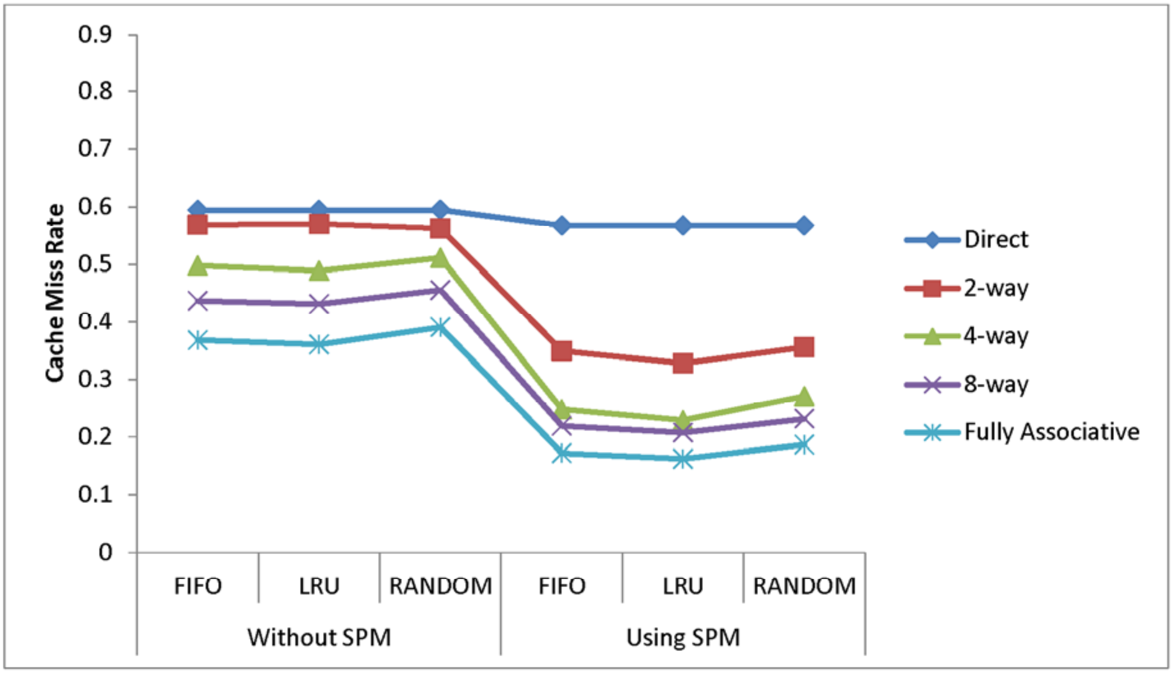


Figure 4.6: Cache miss rate for biquad benchmark for different associativity values

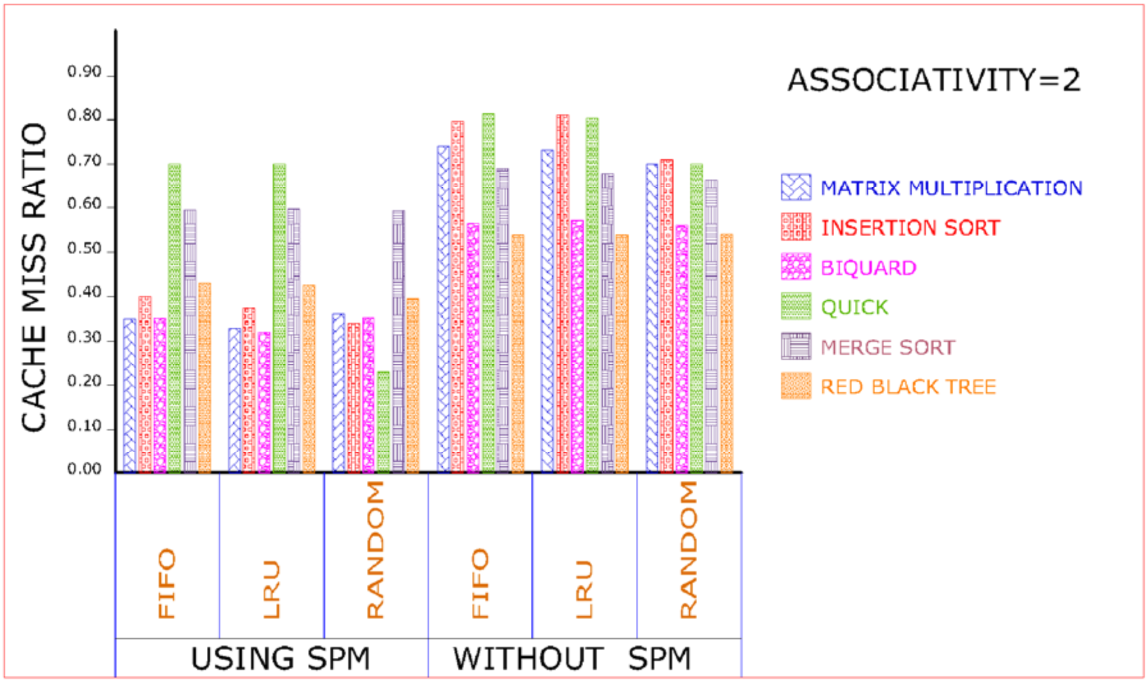


Figure 4.7: 2-way associative cache: cache miss rate using SPM and without using SPM using 6 benchmarks

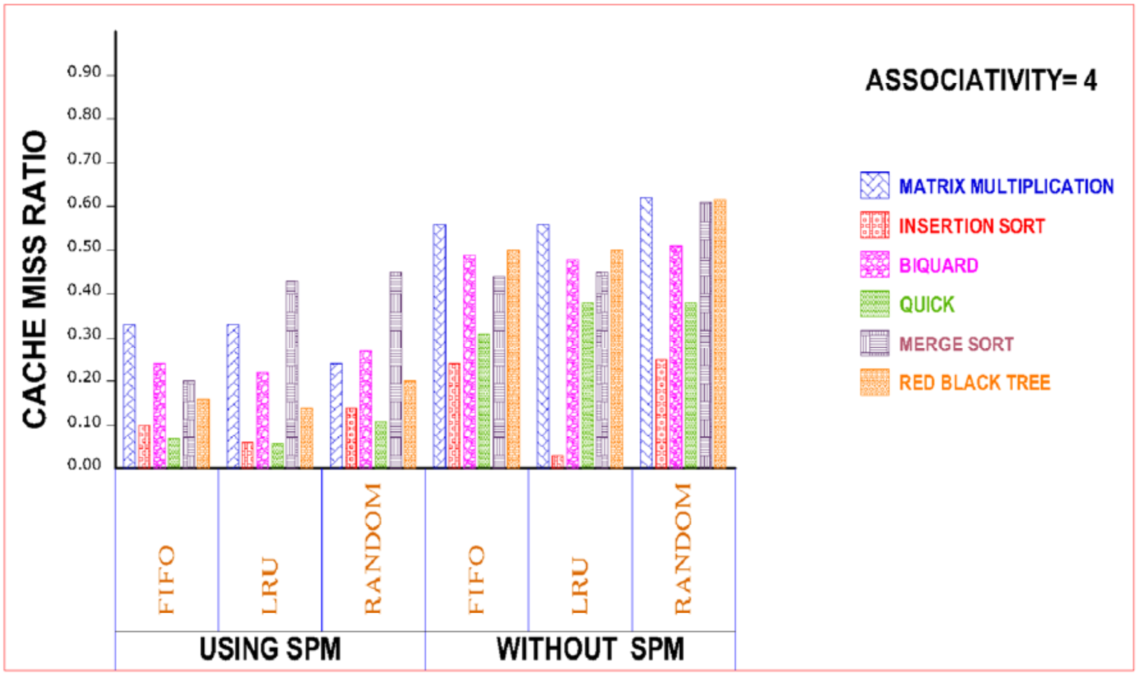


Figure 4.8: 4-way associative cache: cache miss rate using SPM and without using SPM using 6 benchmarks

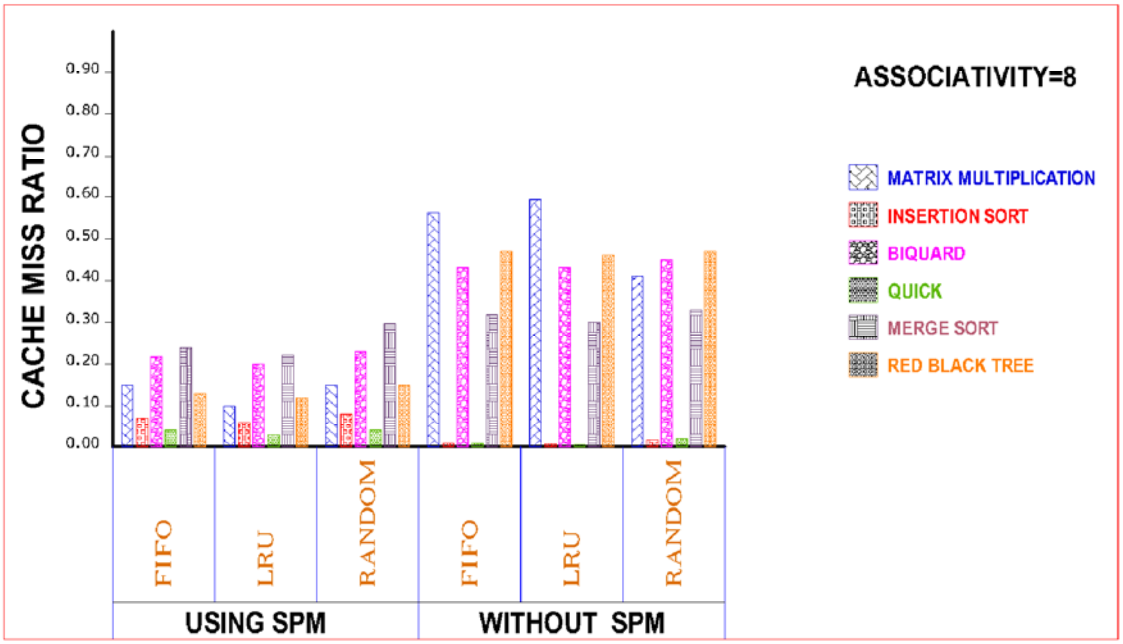


Figure 4.9: 8-way associative cache: cache miss rate using SPM and without using SPM using 6 benchmarks

Table 4.1: Average miss rate comparison cache vs SPM using LRU, FIFO, and random replacement policies

| Benchmark | Associativity | Av. Cache Miss Rate Using SPM | Av. Cache Miss Rate Without SPM | % Av. Cache Miss Rate Improvement |
|---|---|---|---|---|
| Matrix Multiplication | 1 | 0.4654 | 0.7993 | **18.82%** |
| Insertion sort | | 0.4688 | 0.81 | |
| Biquad | | 0.5675 | 0.595 | |
| quick | | 0.7047 | 0.7764 | |
| Merge sort | | 0.6832 | 0.7827 | |
| RB Tree | | 0.5452 | 0.5592 | |
| Matrix Multiplication | 2 | 0.3437 | 0.7281 | **34.68%** |
| Insertion sort | | 0.3746 | 0.7731 | |
| Biquad | | 0.3443 | 0.5673 | |
| quick | | 0.5477 | 0.7752 | |
| Merge sort | | 0.5947 | 0.6774 | |
| RB Tree | | 0.4172 | 0.5418 | |
| Matrix Multiplication | 4 | 0.3009 | 0.5825 | **51.14%** |
| Insertion sort | | 0.1026 | 0.1772 | |
| Biquad | | 0.249 | 0.4997 | |
| quick | | 0.0824 | 0.3623 | |
| Merge sort | | 0.3636 | 0.4708 | |
| RB Tree | | 0.1702 | 0.508 | |
| Matrix Multiplication | 8 | 0.1353 | 0.5225 | **61.283%** |
| Insertion sort | | 0.0138 | 0.0766 | |
| Biquad | | 0.2191 | 0.4414 | |
| quick | | 0.0128 | 0.0405 | |
| Merge sort | | 0.252 | 0.323 | |
| RB Tree | | 0.1349 | 0.4694 | |
| Matrix Multiplication | 16 | 0.1148 | 0.1231 | **57.019%** |
| Insertion sort | | 0.00665 | 0.0665 | |
| Biquad | | 0.1736 | 0.3729 | |
| quick | | 0.005 | 0.034 | |
| Merge sort | | 0.167 | 0.232 | |
| RB Tree | | 0.09 | 0.4208 | |

Figure 4.10, 4.11, 4.12, 4.13 and 4.14 shows the graphical results for evaluating bubble sort algorithm on a scale of increasing data set size (100, 500, 1000, 1500 and 2000). As it is clear from the graph that with the increase in number of elements, cache miss rate increases however with the increase in associativity, it decreases also.

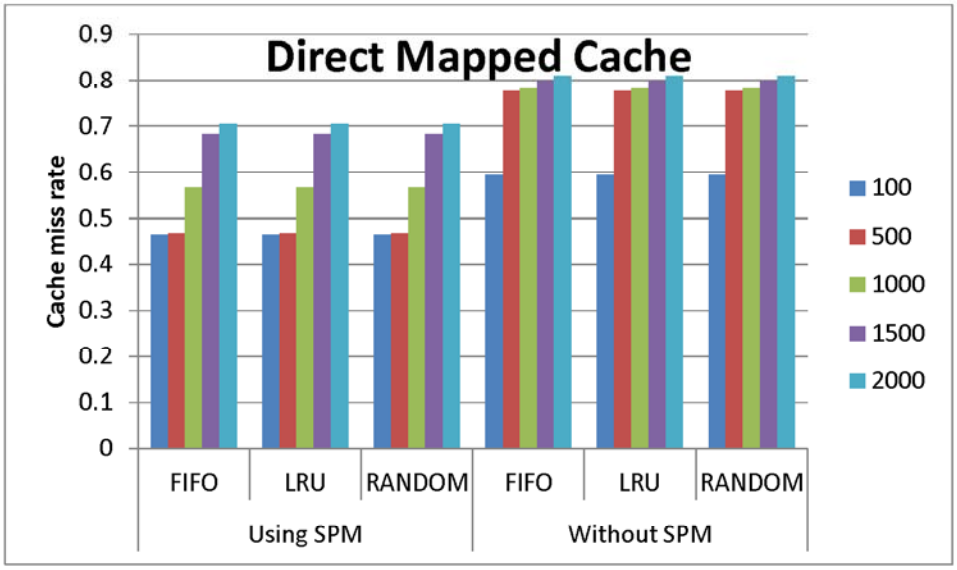


Figure 4.10: Cache miss rate for bubble sort with increasing number of elements in direct mapped cache.

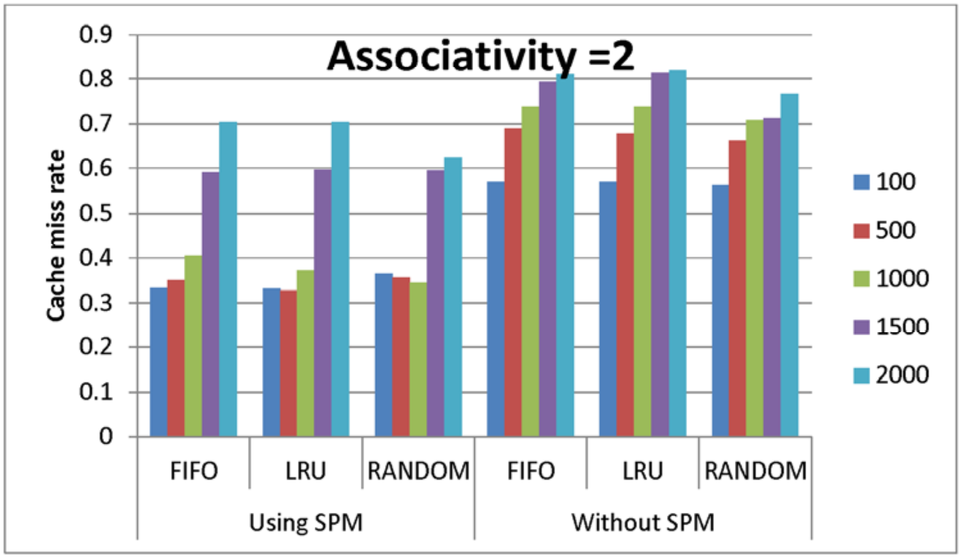


Figure 4.11: Cache miss rate for bubble sort with increasing number of elements in 2-way associative cache.

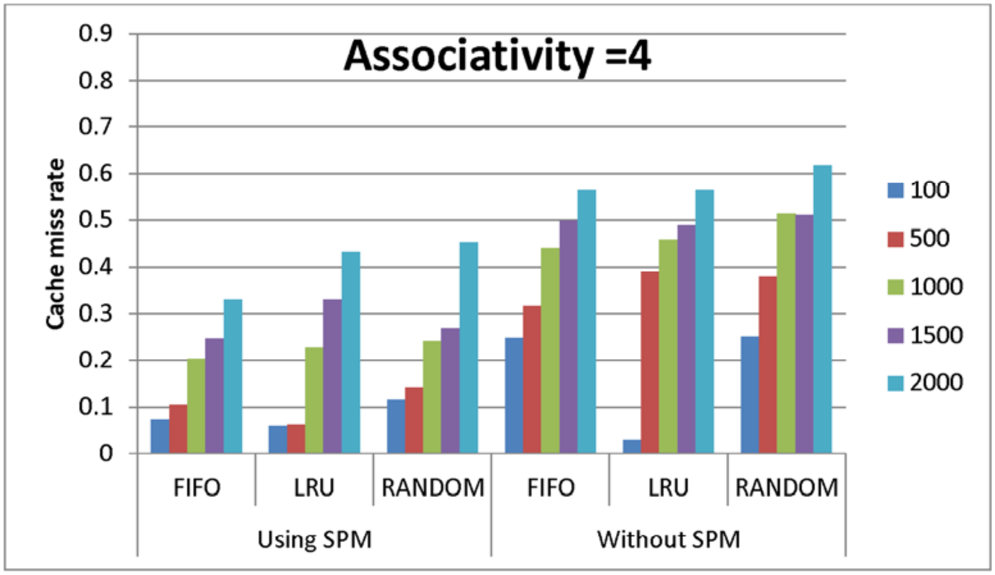


Figure 4.12: Cache miss rate for bubble sort with increasing number of elements in 4-way associative cache

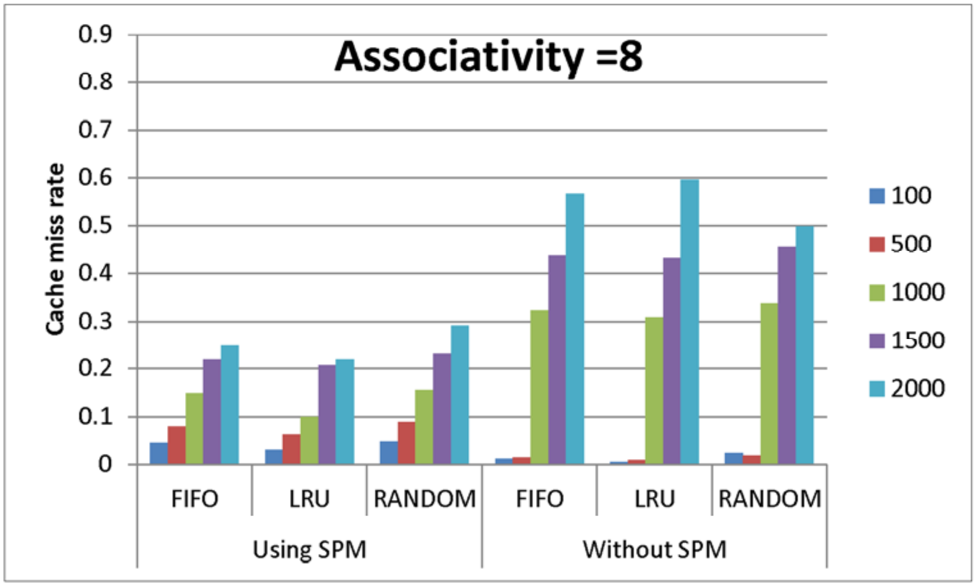


Figure 4.13: Cache miss rate for bubble sort with increasing number of elements in 8-way associative cache

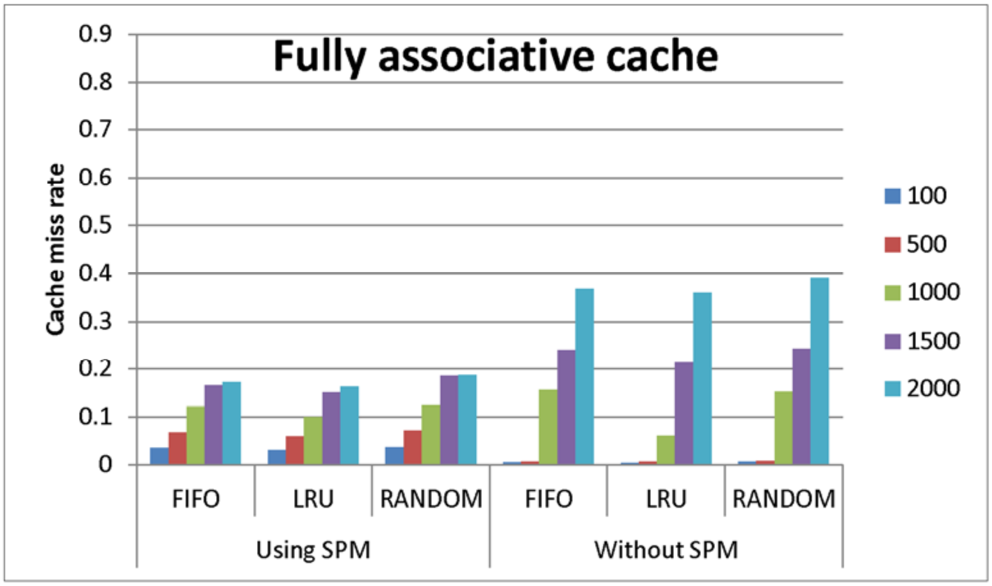


Figure 4.14: Cache miss rate for bubble sort with increasing number of elements in fully associative cache

### 4.6 Conclusion and Future Work

This chapter presented a simulation based evaluation of cache with scratch pad memory in terms of main memory design issues like replacement policies and associativity in embedded systems. The simulation results demonstrate the reduction in cache miss rate up to 61% by the use of SPM. Hence, the addition of on-chip memory SPM build the system more efficient with lesser no. of miss rates. Moreover, results maintain the status of LRU as the best replacement policy on FIFO and Random in SPM as well as in cache memory. With the increase in associativity, cache miss rate decreases as data can be easily get accessed in higher associative caches. Moreover, cache miss rate decreases when using SPM as it is smaller in size as compared to cache memory and hence fast availability of data.

Future work will improve these results by considering dynamic moving memory data in and out of the scratchpad that is to propose and implement a new and efficient replacement policy in embedded systems. In addition, research can be done for the extension of this approach to multiprocessor systems on chip (MPSoC) [I. Issenin, 2006].

The contribution towards this research is published and is as follows:

Shaily Mittal and Nitin, *Replacement Policies for Scratch Pad Memory in Embedded Systems*, Proceedings of the IEEE TENCON, Bali, INDONESIA, November 21-24, 2011, pp. 159-163. [Indexed in SCOPUS].

# CHAPTER 5

# MEMORY MAP: A MULTIPROCESSOR MULTILEVEL CACHE SIMULATOR

## 5.1 Introduction

In the memory hierarchy, cache is the first memory present next to the CPU. Hence, when CPU generates an address, it first checks in cache [L. Hennessy, 2006]. Cache memory is costly and rather small as compared to the other memories present in the memory hierarchy. Cache memory provides temporary storage and able to satisfy most of the data requests generated by the CPU as the most recent used data is present in the cache.

On-chip caches with increased size diminish the speed gap between memory and processor. According to a literature survey in [M.B. Kamble, 1997] cache consumes 25 to 50% of total chip energy, while covering only 15% to 40% of total chip area. That's why designers main focus is to improve the cache performance by reducing energy consumed and high cache hit rate rather than reducing chip area.

Automotive, aeronautics, electronics and automation of industries generally have hard real-time constraints [J. Reineke, 2006]. These are the major applications of embedded systems also. The main concern in the designing of memory hierarchy of embedded systems is small systems with fast speed. These two factors effect the performance of the system a lot. In case of cache hit, one to two CPU cycles are used, while tens of cycles are required in case of a cache miss treated as a consequence of miss handling. Hence, the speed decreases and becomes key factor in designing memory hierarchy in the system. To increase the speed of the system, generally instructions and data caches are taken on same chip. To implement such on chip designing, scratch-pad memory (SPM) has become a substitute for embedded systems memory hierarchy [R. Banakar, 2002] [P.R. Panda, 2005].

Shared caches generally act as a communication medium between multiple processors present on a single chip [S. Pasricha, 2007]. Simultaneous execution of multiple cores simultaneously is today's requirement for embedded applications [I. Issenin, 2008]. SoCs with tens of cores are commonplace [D. Cho, 2009] and platforms with hundreds of cores have been proclaimed. However, performance and power benefits can only be achieved by making use of concurrency. The main challenge faced by the programmers is to implement this core level parallelism.

Thread is the solution for this problem in MPSoC [T. Harris, 2003]. Just because of the ability of executing multiple threads simultaneously, implementation of multiple processors on a single chip is possible [J.W. Chung, 2006]. However, this result in a problem of simultaneous access of processors to same shared cache as it requires a synchronization mechanism [I. Issenin, 2006]. Memory Map provides framework for evaluating cache hit and miss ratio. It is a fast, flexible, open source and easy to learn and understand.

When the processor requests a data access, it is firstly get checked in cache memory and if data is present in cache memory then the processor reads data from cache or writes data to the cache instantly [D.H. Albonesi, 1999], [R.I. Bahar, 1998], which is much quicker than reading to and from main memory. Similar is the case with writing operation.

The cache memory used in systems can be divided into three types: [H. Akkary, 2003] an instruction cache to increase the speed of fetch instruction, [R. Fromm, 1997] a data cache which speeds up fetching and storing of data and a translation look aside buffer (TLB) [L. Hammond, 2012] maps virtual to physical address of instructions and data streams both. In this current study, both data and TLB cache has been implemented. In a multiprocessor system on chip, memory can be divided into two types: Processors own private memory and common shared memory for processors [M. Huang, 2000].

Cache stores the information in the form of cache blocks which is the smallest unit of data storage. Cache associativity majorly affects the performance of the cache. Now a days, systems with high associativity in multilevel caches are preferred.

## 5.2 Background

Various simulators are already developed by a number of authors to evaluate the features of multiprocessors with shared memory architecture. The advantages and disadvantages of some of them are going to be discussed next which results in the evolution of memory map multiprocessor simulator.

SMPCache [M.Á.V. Rodríguez, 2001] is a simulator for symmetric multiprocessors only. SMPCache has user friendly interface which can be installed on any windows operating system. One disadvantage of this simulator is that it requires some tool to generate memory traces.

OpenMP [W.C. Jeun, 2007] is a standard interface for the multiple processors with address space running simultaneously. It can be programmed in C or C++. OpenMP API implement parallelism by the use of threads. OpenMP also use compiler directives to implement parallel programming. The main disadvantage with OpenMP is that it can’t be able to get memory access statistics.

Simple Scalar [T.D.C. Burger, 1997] is C based set of simulation tools that provides the feature to design your own virtual computer system with processor and different levels of memory hierarchy. A collection of tools like sim-fast, sim-cache and many more are present in this toolset. Simple Scalar tool set also performs statistical analysis of various resources, performs debugging and verification of architecture designed. However, the major problem with Simple Scalar is that it can’t able to handle a system with multiple processors.

M-Sim [J. Loew, 2011] is an extension of Simple Scalar 3.0d toolset. It supports multiple processor environments. Multiple processors can be implemented by using concurrent thread execution.

For executing the program, [S. Mittal, 2012]

```
./sim-outorder num_cores 3 max_contexts_per_core 3 cache:dl1 dl1:1024:8:2:l -
cache:dl2 dl2:1024:32:2:l program.arg
```

where max_contexts means number of threads and in the above example, number of cores are taken as 3.

An executable file prog.out will be produced:

```
1000000000 # prog < prog.in > prog.out
```

M-Sim can be used to simulate multiprocessors. The problem is that it requires separate program per core and not a single program is divided for multiple cores. The other major disadvantage of this simulator is that it accepts only alpha-binary files which are created by DEC compiler which is not free of cost.

SystemC is a freely available simulator through SystemC portal [T. Rissa, 2005]. It is based on standard Linux C++. However, the major limitation is that the compiler of this software is debugging the development software only rather than debugging the code running on it. Hence, it lacks in finding compile time errors and results in run time errors which are difficult to rectify. Secondly, SystemC does not have any linker available till now.

## 5.3 Alternate Approach for Multiprocessor Synchronization

### 5.3.1 Memory Interleaving

To increase the speed of DRAM, authors have implemented the concept of memory interleaving [C.L. Chen, 1989]. In memory interleaving, processors access simultaneously alternate sections of the memory without caching. Memory is divided into memory banks. If there are m banks, $i^{th}$ map memory block would store in a bank no. i mod m and this is called m-way interleaved memory.

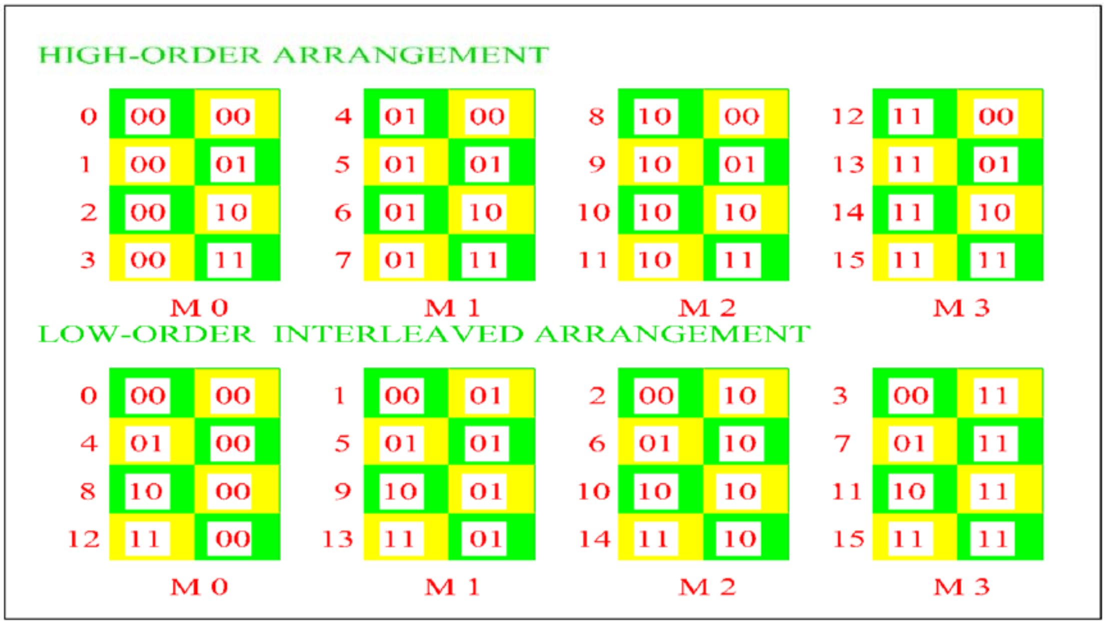


Figure 5.1: Interleaved structure [Norman Matloff, 2003]

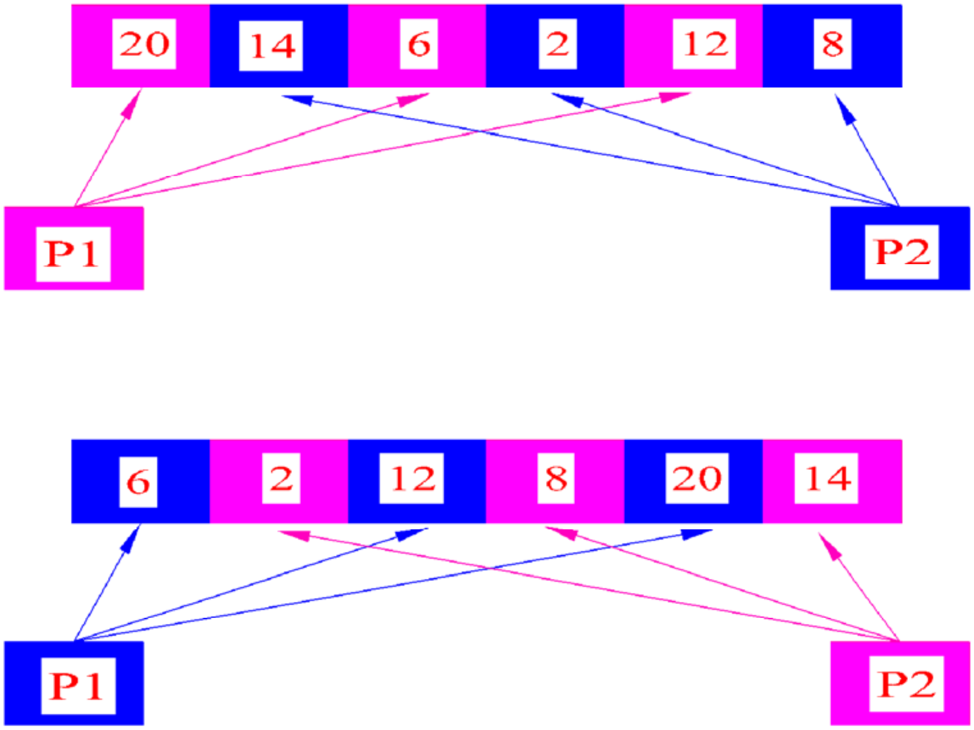


Figure 5.2: Example of merge sort using the concept of memory interleaving [W. Stallings, 2009]

The concept of memory interleaving makes possible the simultaneous access of memory through multiple processors. It can access more than one word concurrently in only one memory access cycle. This can only be possible because of dividing the complete memory into m distinct memory modules which enables the simultaneous access to m memory modules and hence increases the speed of DRAM.

To implement the concept of memory interleaving, merge sort algorithm is taken as benchmark by the authors. Generally, processors have a tendency to access continuous memory locations for executing a program rather than alternate as in case of memory interleaving. The low order and high order interleaved arrangement is shown in fig. 5.1. The low order arrangement have major advantage of parallel processing as continuous words are now in alternate memory modules and hence more preferred rather than in sequence.

In general, the list is divided into two equal parts in merge sort, however in interleaving instead of dividing the list; it has been accessed by m processors simultaneously providing a virtual division of memory into m blocks. Two processors are accessing alternate memory locations using low order memory interleaving. By implementing memory interleaving in merge sort, merging of elements now becomes simple as elements in the same memory block are compared also shown in example drawn in fig. 5.2. All elements will be at consecutive locations simultaneously during merging. Presence of all merging elements simultaneously during merging increases the cache performance and cache hit ratio. Original merge sort algorithm along with modified merge algorithm with interleaving is shown in fig. 5.3. In the continuation of fig. 5.3, modified insertion sort is represented using memory interleaving.

Input: Array A

Modified Merge sort Algorithm:

// A is array of size Maxsize elements, which need to be sorted from left to right position, M is the number // of processors, which sort elements of array in parallel, inter is degree of interleaving.

```
Step1: factor = (right – left + 1)/ Pow (M , inter).
Step 2: if ( factor > 3) then   goto step 3 else goto step 4.
Step 3:  For i = 1 to M                      // for loop partitions elements into M processor
        Sort individual elements in the partitioned array with starting position as I and end
        position till factor or factor + 1, each element placed with “inter” positions next to
        previous elements
        Merge ( A, left, right, M, inter)
Step 4: Else Insertion sort(A, left, n, right, Pow(M,inter-1)
```

Modified Merge algorithm:

```
Step1: pinter = Pow (M, inter)
Step2: for i = 0 to M
          Pointer[i] = left + r* pinter / M
         N= (right – left +1)

Step 3: for i = 0 to n
            For (j= 0, j<M;++j)
                    If ((pointer[j] != -1) && (tmparray[i] > A[pointer[j]))
                          Tmparray[i] = A[pointer[j]]
                          Indexj = j;
                    If (ponter[indexj] + pinter > n-1)
                          Pointer[indexj] = -1
                    Else pointer[indexj] + = pinter

Step 4: for ( i = 0; i <  M; ++ i)
             A[i] = tmparray[i]
```

Output: Sorted array A

2)

```
Input: Array A
Insertion sort Algorithm

Step1:  for( p= i+inter, count-1; count < n, p <= right; p= p+ inter, ++count)
        Tmp = A[p]

Step 2: for(j = p;  j >=1 && j – inter >= i && A[j – inter] > tmp; j= j – inter )
        A[j] = A[j – inter]

Step 3: A[j] = tmp;

Output: Sorted array A
```

Figure 5.3: Modified merge sort, merge and insertion sort algorithms for memory map simulator

The main memory consists of $2^n$ words which are further divided into $2^m$ independent memory modules and each memory module consists of $2^{n-m}$ words. Working of these M modules simultaneously or using pipelining increases the speed of the program execution by M times approximately. Hence, increases the speed of memory read and writes operations. The complete n bit address is divided into two parts: m-bit to specify the memory module and rest n-m bits are used to locate the word into the specified module.

In this evaluation, an array of 30 elements has been taken with least recently used replacement policy. As far as cache configuration is concerned, L1 cache is taken as 16B and a 64 B L2 cache. Both are 2-way set-associative.

### 5.3.2 Observations

Merge sort algorithm plain and with interleaved memory have been implemented on sim-cache and sim-fast tools of Simple Scalar functional simulator. It has been installed on linux operating system. Cache hit and miss rate have been evaluated for comparison between normal and interleaved memory. Hit ratio for L1 cache treated as SPM with and without using memory interleaving in merge sort benchmark is shown in fig. 5.4. the cache hit ratio reaches 99.87 from 99.85 after introducing the concept of memory interleaving in merge sort which is approximately reaches 100 means a very less cache miss. This is a good attainment. Similarly, fig.5.5 shows the L2 cache hit ratio. In the same manner, cache hit ratio reaches 99.74 approximately 100 when interleaved rather than 95.54 when non interleaved memory is considered.  This is the highest hit ratio achieved as acquiring hit ratio 100 is impractical.

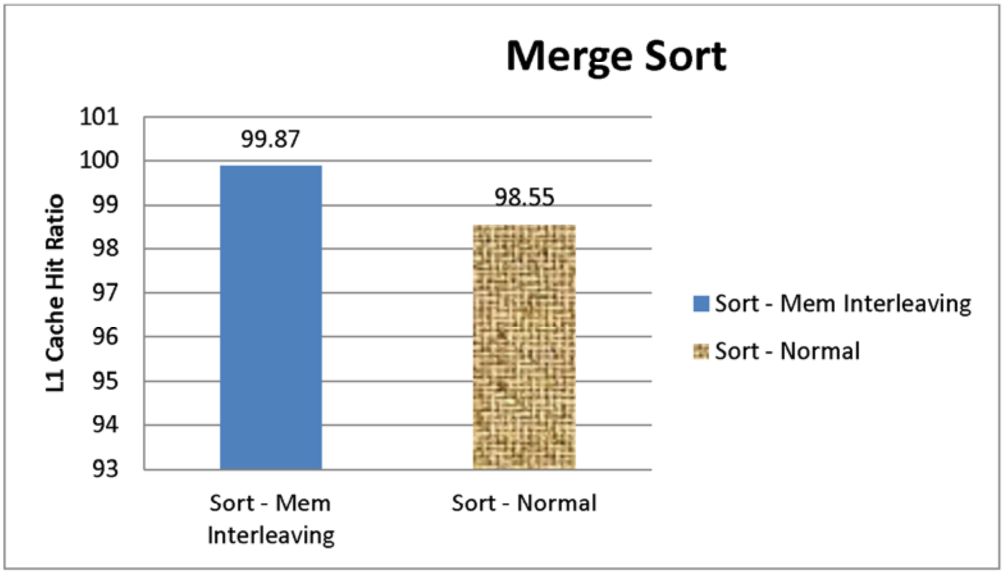


Figure 5.4: DL1 (SPM) hit ratio for normal and interleaved merge sort

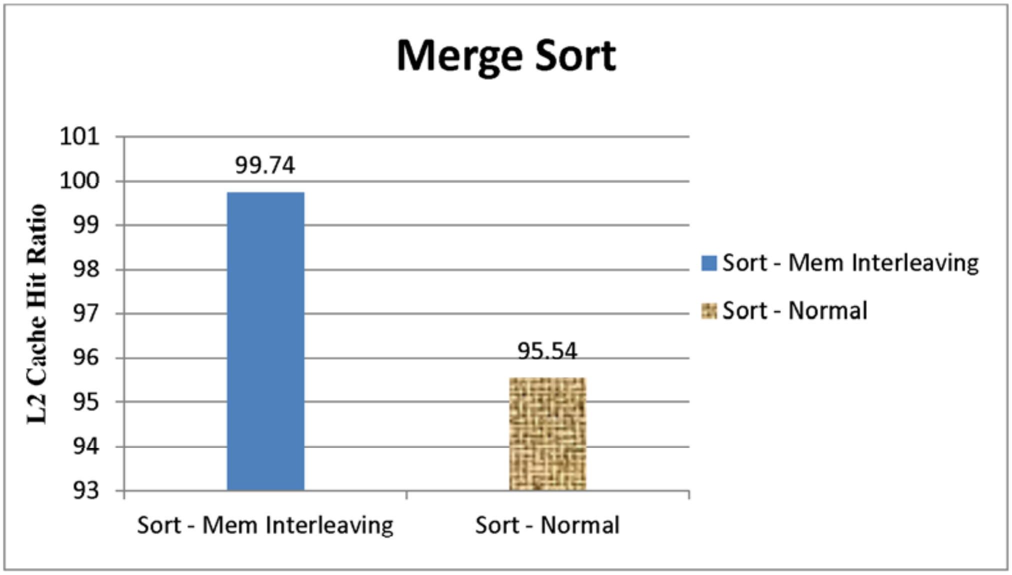


Figure 5.5: L2 cache hit ratio for normal and interleaved merge sort

100 - Hit Ratio is miss ratio and miss rate is miss ratio divided by 100. First two bars in fig. 5.6 shows the SPM miss rate with normal and interleaved execution. The bars in fig. 5.6 shows decrease in miss rate from 4.45 to 0.26 in merge sort with interleaved memory. Figure 5.7 is drawn for L2 cache miss rate which shows a decrease in value from 1.46 to 0.13 which is very close to 0 and also attaining 0 miss rate is impractical. DL1 cache has been considered. The replacement rate for DL1 using merge sort normal and also interleaved is shown by next two bars in the graph. The red colored bar is for normal execution while textured is for interleaved execution. The miss rate for L1 cache reaches 0.15 from 2.14 when non-interleaved and also miss rate for L2 cache decreases 1.27 when interleaved. As cache hit rate increases, replacement rate decreases which in turn leads to lesser memory access time and hence better performance. Cache write back rate for L1 and L2 cache is shown by last two bars in graphs of fig. 5.6 and fig. 5.7. There is also decrease in write back rate with the use of interleaved memory. For L1, it was 1.04 when not interleaved and later on it becomes 0.06 when using interleaving in merge sort. Similarly, write back rate for L2 cache also decreases with interleaving by 1.75. It is low again due to decrease in cache miss rate and hence requirement for write back to memory decreases.

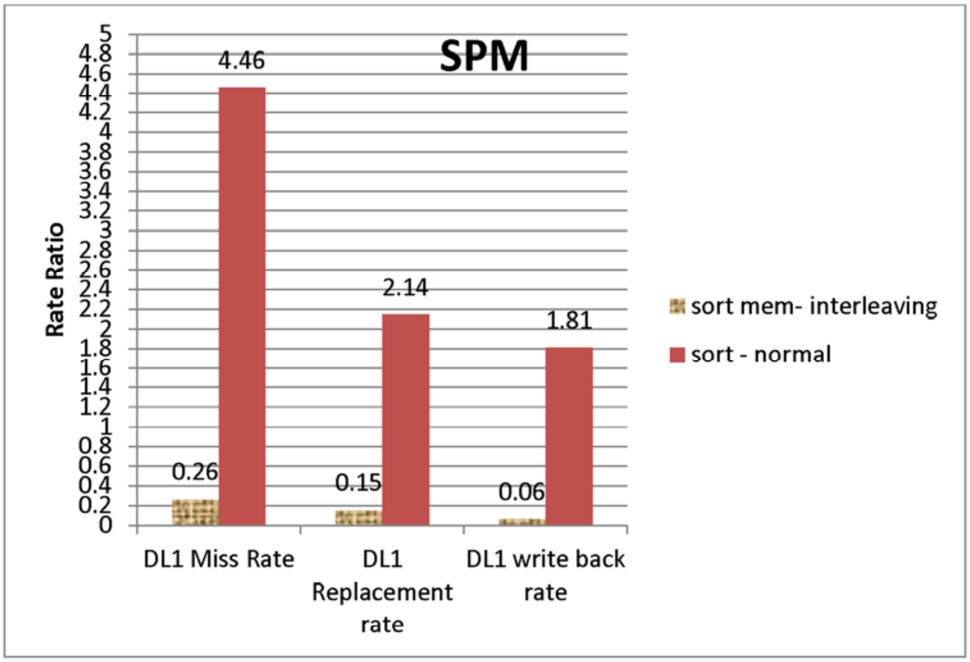


Figure 5.6: SPM cache miss rate, replacement rate and write back rate using merge sort plain and also interleaved

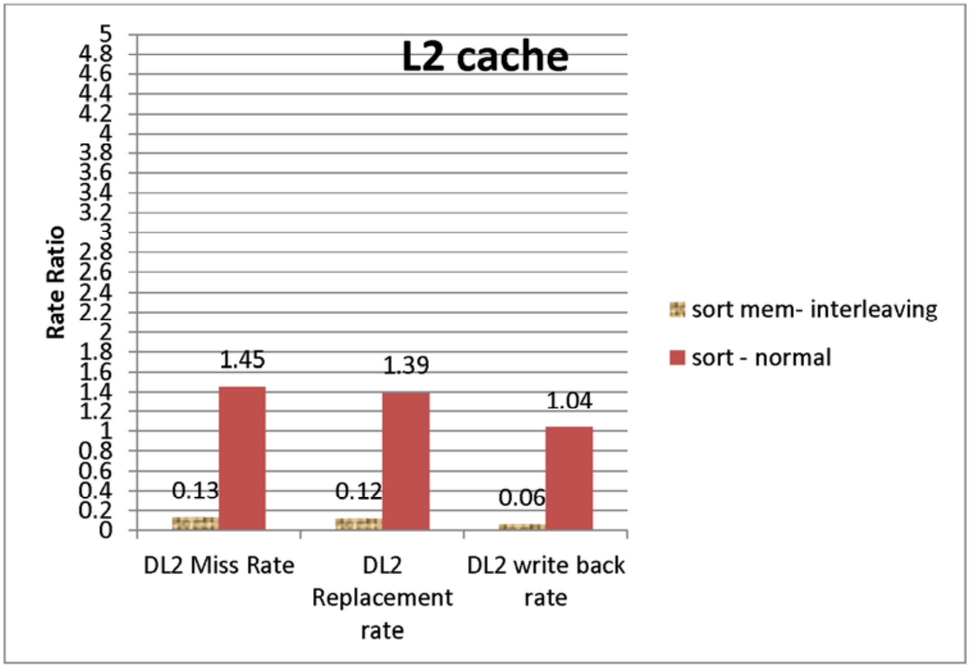


Figure 5.7 : L2 cache miss rate, replacement rate and write back rate using merge sort plain and also interleaved

### 5.3.3 Limitations

The results drawn in previous section have proved that use of memory interleaving has significantly improves the performance by decreasing the cache miss rate. However this approach has some limitations also. The major limitation of this concept is that the complexity of the modified algorithms are very high, which have not been taken into consideration till now. Hence, it increases the power consumption of the system with the decrease in cache miss rate and further memory access time. The second limitation is that this architecture has been implemented on Simple Scalar Simulator, which does not support multiple processors in the way as it is required by the given problem.

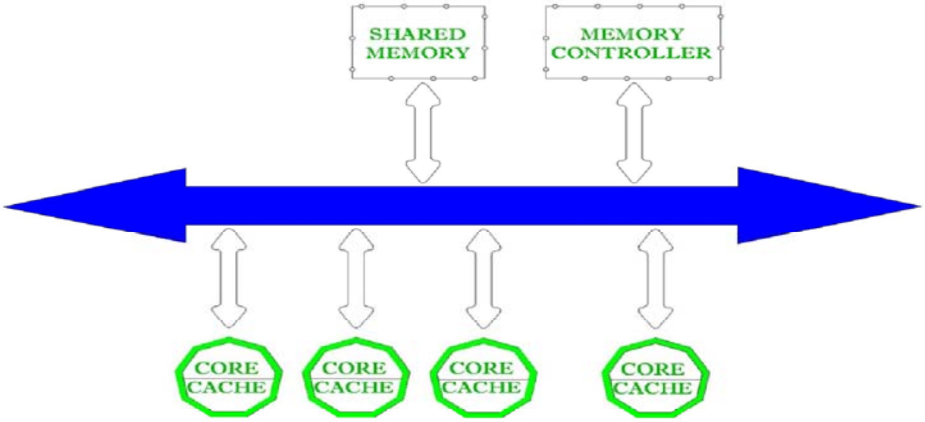


Figure 5.8: Architecture used in sorting algorithm evaluation on Simple Scalar

### 5.4 Architecture of Memory Map Simulator

The designed architecture consists of multiprocessors with shared memory as well as with private caches [K. Hwang, 1984]. This platform used shared memory for communication between multiple homogeneous processors. The architecture used in this evaluation is shown on fig. 5.8. The design consists of following components:

- A number of homogeneous processors
- Processors own private cache memory

- A shared-memory for inter processor communication
- A memory controller to manage the working of system

This memory map simulator has different features which going to be discussed next.

The physical address generated by the processor is used to map cache memory address. One main memory block is placed into only one cache memory block i.e. used direct mapping. This simulator enables simultaneous execution of a single program. As each processor has its own private memory and also shared memory, so to avoid cache coherence problem memory controller permits processors to access only to some parts of memory. Moreover, it uses write-back strategy to synchronize cache and main memory [D.A. Patterson, 1994].

It has been discussed in section 5.2 that although there are a lot of simulators available still they are not fulfilling the purpose of using multiple processors in the system. This simulator solves this issue by implementing multiple processors through threads which can run concurrently on a single program and these processors communicate to each other through common shared memory. Meta data is used to map variable to logical address and it increases the speed of cache memory access [M. Farrens, 1991].

Various operations performed in memory map simulator are:

1. Memory Allocation: Memory is considered as an array of data bits. For storing a variable, it first checks for first continuous free space in memory. It uses java API to determine the size of different primitive types and classes [S. Mittal, 2012]. It converts variable to bit. Different operations also modify the value of invalid flag.

2. Memory Query: Firstly physical to logical address mapping is required when any query is generated. Then processor first checks the presence of required block in its own cache and if it is there then it generates a cache hit with a cache read operation. If the required data is not present then it counts it as cache miss and hence it needs to be fetched from main memory and also to be loaded into cache memory using cache replacement policy.

3. Memory Update: In case of any updation in the memory, then also physical address is mapped to the logical address first to locate the updation point. Then processor first checks the presence of required block in its own cache and if it is there then it generates a cache hit and hence it updates the cache with setting dirty flag bit to 1. And if block is not present in the cache, it is counted as cache miss. In cache miss, it finds out a cache block that can be replaced to accommodate required data and also reset the dirty flag bit. Hence, this operation updates the data in main memory and also brings the same block to cache memory also for future reference.

Flowchart shown in fig. 5.10 describes the whole operation in memory map simulator in detail.

## 5.5 Test bed and Experimental Setup

Figure 5.9 describes the design of memory map simulator in detail. It consists of multiple processors with their own private caches [A. Samih, 2011] and one common main memory for inter processor communication. L2 cache in the designed system is treated as private caches to processors with varying block sizes of 32 B, 64 B and 128B. Each block is using direct mapping. In simulator, only LRU and FIFO replacement policies are considered. Benchmarks executing on simulator are merge sort, bubble sort and average calculation. Similar as earlier two ways of evaluation is taken: one with normal memory and one with interleaved memory. An average of 10 simulations is considered to find the correct value.

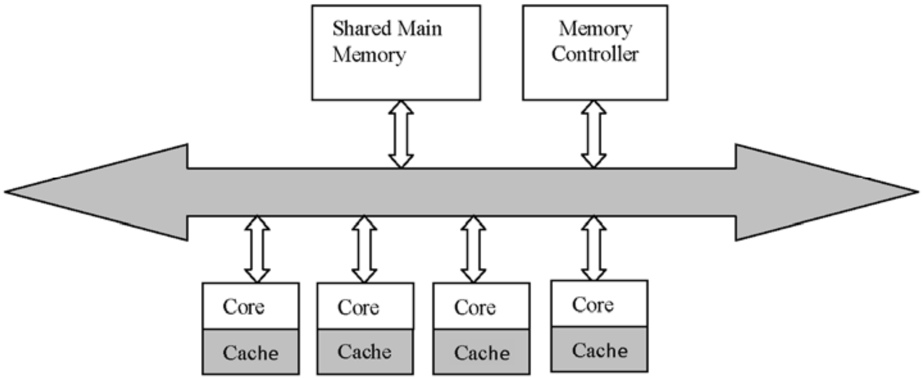


Figure 5.9: Memory map simulator design

Authors have used Java technology (i.e. JDK 1.6) to design simulator and corresponding graphs are drawn in Microsoft excel. The size of data type is determined by Java.lang.reflect class. Custom convert utility package of java is also used to modify data into bits and vice versa [S. Mittal, 2012]. FreePool class is used as memory controller which searches for any free space to allocate and also provides 2 level of hashing by implementing Hashmap. CMap java class is also used for direct address mapping.

## 5.6 Experimental Results

Figure 5.11 displays the execution of merge sort with normal and interleaved memory on memory map simulator. A random array of 30 elements is taken for evaluation. The simulator evaluated total number of accesses to the memory; number of cache hits and also the number of cache miss for normal as well as for interleaved memory in merge sort.

$$Hit\ Ratio = \frac{\text{No. of hits}}{\text{No. of memory accesses}}$$

$$Miss\ Ratio = \frac{\text{No. of miss}}{\text{No. of memory accesses}}$$

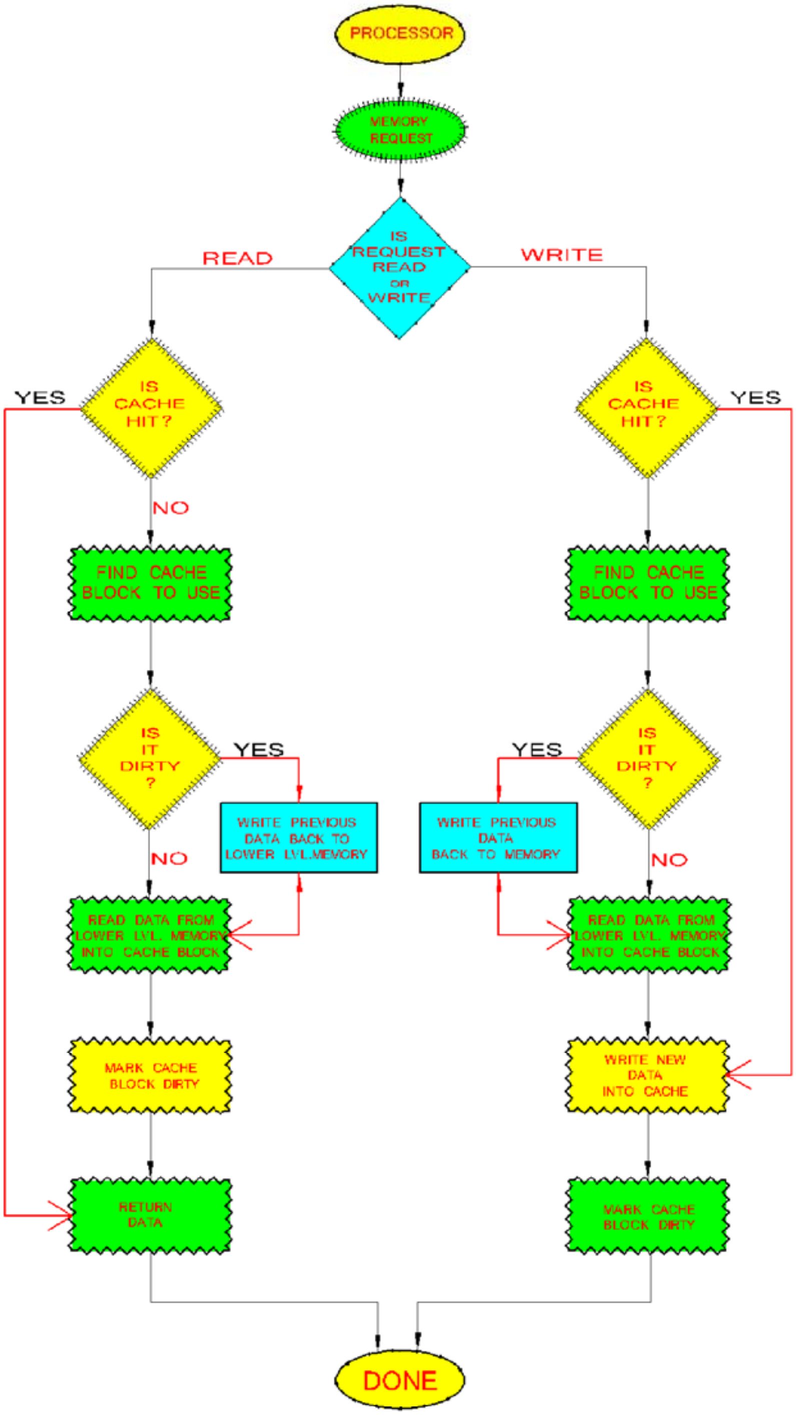
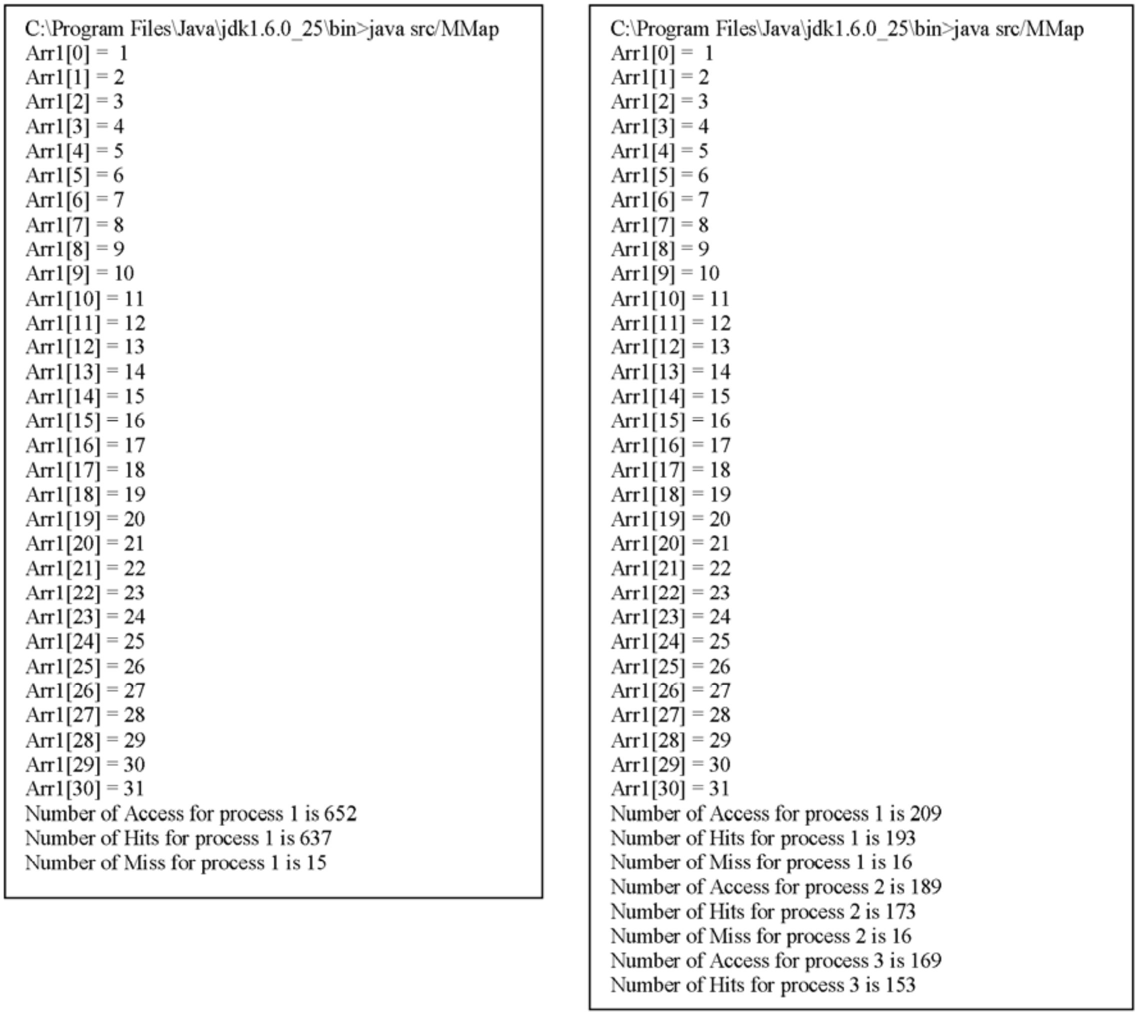


Figure 5.10: Data flow graph of Memory Map simulator



```
C:\Program Files\Java\jdk1.6.0_25\bin>java src/MMap
Arr1[0] =  1
Arr1[1] = 2
Arr1[2] = 3
Arr1[3] = 4
Arr1[4] = 5
Arr1[5] = 6
Arr1[6] = 7
Arr1[7] = 8
Arr1[8] = 9
Arr1[9] = 10
Arr1[10] = 11
Arr1[11] = 12
Arr1[12] = 13
Arr1[13] = 14
Arr1[14] = 15
Arr1[15] = 16
Arr1[16] = 17
Arr1[17] = 18
Arr1[18] = 19
Arr1[19] = 20
Arr1[20] = 21
Arr1[21] = 22
Arr1[22] = 23
Arr1[23] = 24
Arr1[24] = 25
Arr1[25] = 26
Arr1[26] = 27
Arr1[27] = 28
Arr1[28] = 29
Arr1[29] = 30
Arr1[30] = 31
Number of Access for process 1 is 652
Number of Hits for process 1 is 637
Number of Miss for process 1 is 15
```

```
C:\Program Files\Java\jdk1.6.0_25\bin>java src/MMap
Arr1[0] =  1
Arr1[1] = 2
Arr1[2] = 3
Arr1[3] = 4
Arr1[4] = 5
Arr1[5] = 6
Arr1[6] = 7
Arr1[7] = 8
Arr1[8] = 9
Arr1[9] = 10
Arr1[10] = 11
Arr1[11] = 12
Arr1[12] = 13
Arr1[13] = 14
Arr1[14] = 15
Arr1[15] = 16
Arr1[16] = 17
Arr1[17] = 18
Arr1[18] = 19
Arr1[19] = 20
Arr1[20] = 21
Arr1[21] = 22
Arr1[22] = 23
Arr1[23] = 24
Arr1[24] = 25
Arr1[25] = 26
Arr1[26] = 27
Arr1[27] = 28
Arr1[28] = 29
Arr1[29] = 30
Arr1[30] = 31
Number of Access for process 1 is 209
Number of Hits for process 1 is 193
Number of Miss for process 1 is 16
Number of Access for process 2 is 189
Number of Hits for process 2 is 173
Number of Miss for process 2 is 16
Number of Access for process 3 is 169
Number of Hits for process 3 is 153
```

Figure 5.11: Snapshots of merge sort execution (non interleaved and interleaved) with 128 Byte block size

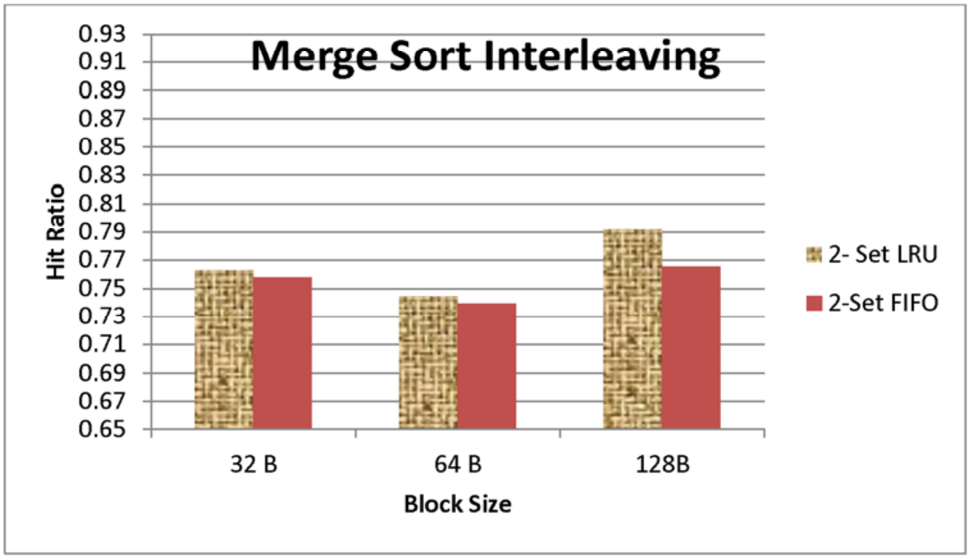


Figure 5.12: Hit ratio of 2-set LRU and FIFO replacement policies using merge sort interleaving

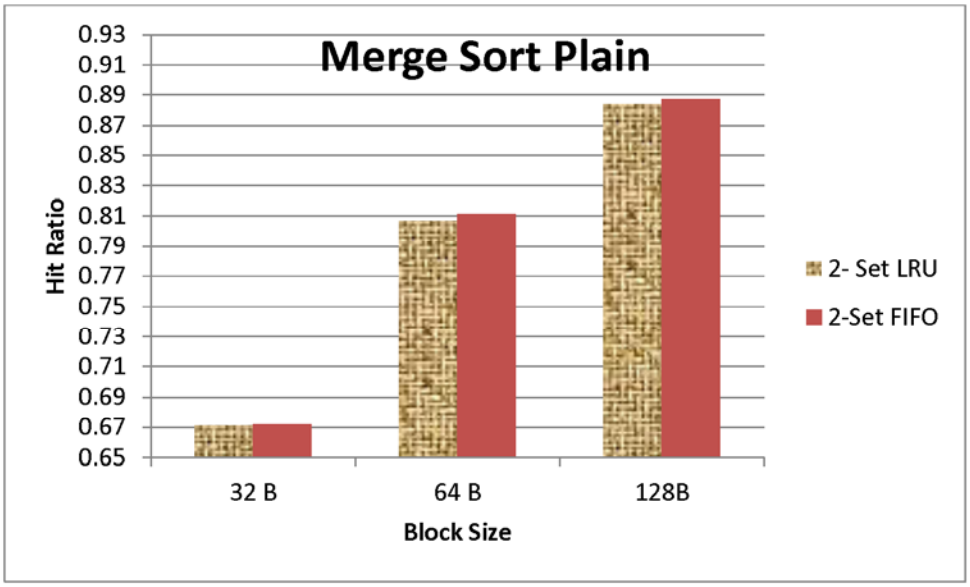
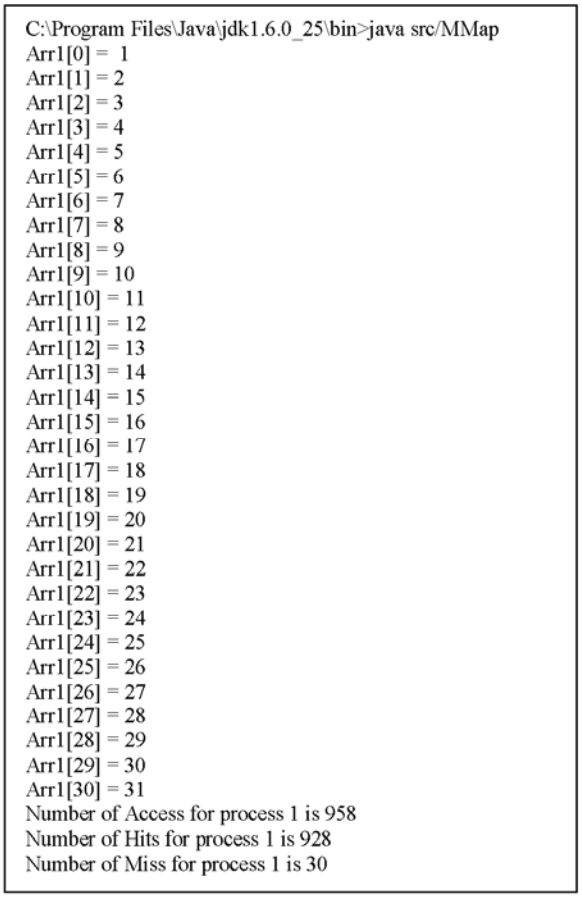


Figure 5.13: Hit ratio of 2-set LRU and FIFO replacement policies in merge sort without interleaved memory

LRU outperforms FIFO when interleaved memory is used as shown in fig. 5.12 while in fig. 5.13, LRU and FIFO shows almost similar behavior when memory is not interleaved and using the merge sort as benchmark. Merge sort with 128 B block size reaches a maximum of 0.79 hit ratio when interleaved in comparison to 0.88 in case of non interleaved memory structure. The results of all evaluations using three benchmarks with and without interleaving are tabulated in table 5.1.

Figure 5.14 displays the screen shot of memory map simulator executing bubble sorting of 30 elements. As seen in the figure, it evaluates number of memory accesses for a processor, no. of hits and no. of misses for each processor.

The graph in fig. 5.15 is drawn on the basis of results obtained in fig. 5.14. Three different block sizes 32B, 64B and 128B are considered. The first two bars in fig. 5.15 displays the hit ratio for 32B blocks size and proves a little better performance of LRU than FIFO. With 64 B blocks, cache hit rate increases 7% in comparison to 32 B blocks and finally touches 92.3 with 128B blocks.



```
C:\Program Files\Java\jdk1.6.0_25\bin>java src/MMap
Arr1[0] =  1
Arr1[1] = 2
Arr1[2] = 3
Arr1[3] = 4
Arr1[4] = 5
Arr1[5] = 6
Arr1[6] = 7
Arr1[7] = 8
Arr1[8] = 9
Arr1[9] = 10
Arr1[10] = 11
Arr1[11] = 12
Arr1[12] = 13
Arr1[13] = 14
Arr1[14] = 15
Arr1[15] = 16
Arr1[16] = 17
Arr1[17] = 18
Arr1[18] = 19
Arr1[19] = 20
Arr1[20] = 21
Arr1[21] = 22
Arr1[22] = 23
Arr1[23] = 24
Arr1[24] = 25
Arr1[25] = 26
Arr1[26] = 27
Arr1[27] = 28
Arr1[28] = 29
Arr1[29] = 30
Arr1[30] = 31
Number of Access for process 1 is 958
Number of Hits for process 1 is 928
Number of Miss for process 1 is 30
```

Figure 5.14: : Snapshots of bubble sort execution (non interleaved and interleaved) with 32 Byte block size.

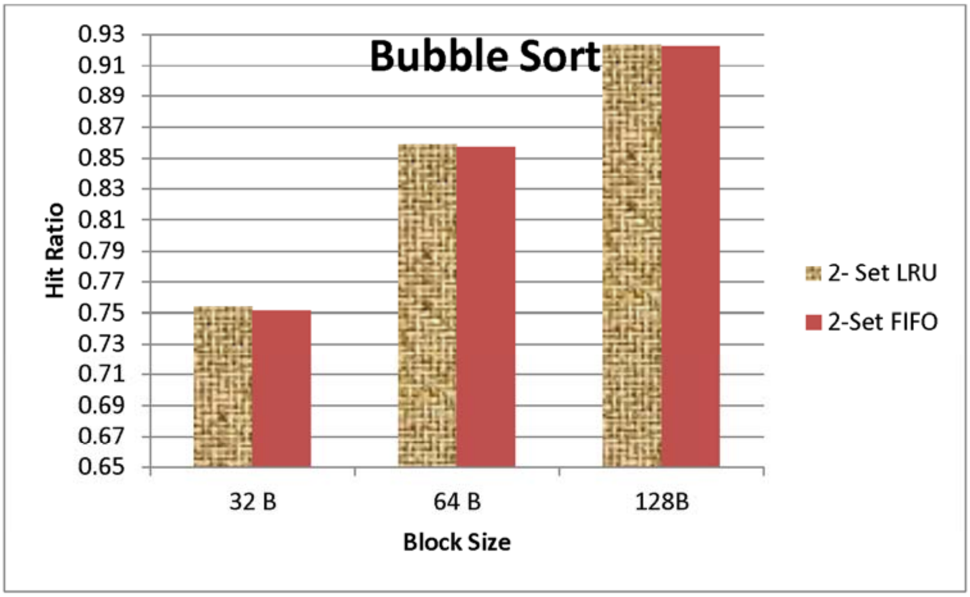
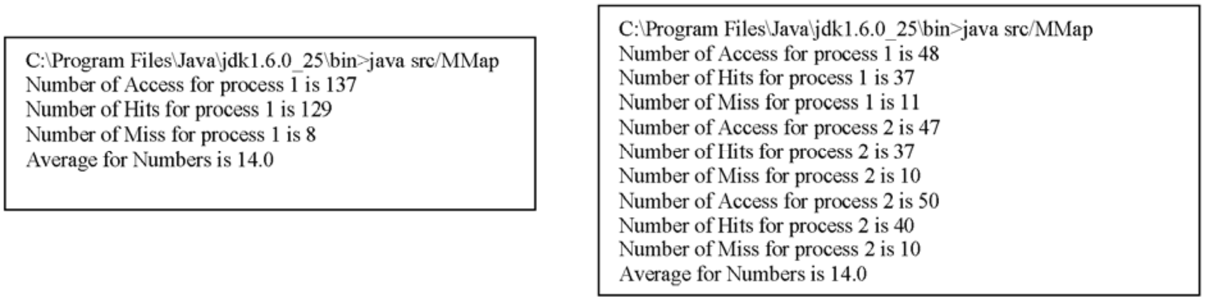


Figure 5.15: Hit ratio of 2-set LRU and FIFO replacement policies using bubble sort



```
C:\Program Files\Java\jdk1.6.0_25\bin>java src/MMap
Number of Access for process 1 is 137
Number of Hits for process 1 is 129
Number of Miss for process 1 is 8
Average for Numbers is 14.0
```

```
C:\Program Files\Java\jdk1.6.0_25\bin>java src/MMap
Number of Access for process 1 is 48
Number of Hits for process 1 is 37
Number of Miss for process 1 is 11
Number of Access for process 2 is 47
Number of Hits for process 2 is 37
Number of Miss for process 2 is 10
Number of Access for process 2 is 50
Number of Hits for process 2 is 40
Number of Miss for process 2 is 10
Average for Numbers is 14.0
```

Figure 5.16: : Snapshots of average calculation of 30 elements execution (non interleaved and interleaved) with 32 Byte block size

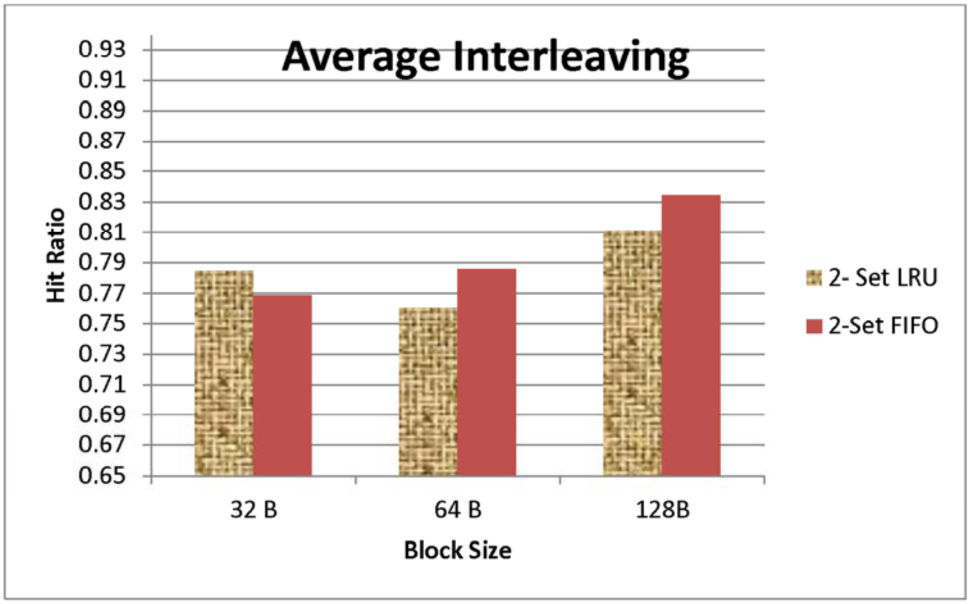


Figure 5.17: Hit ratio of 2-set LRU and FIFO replacement policies using interleaved average

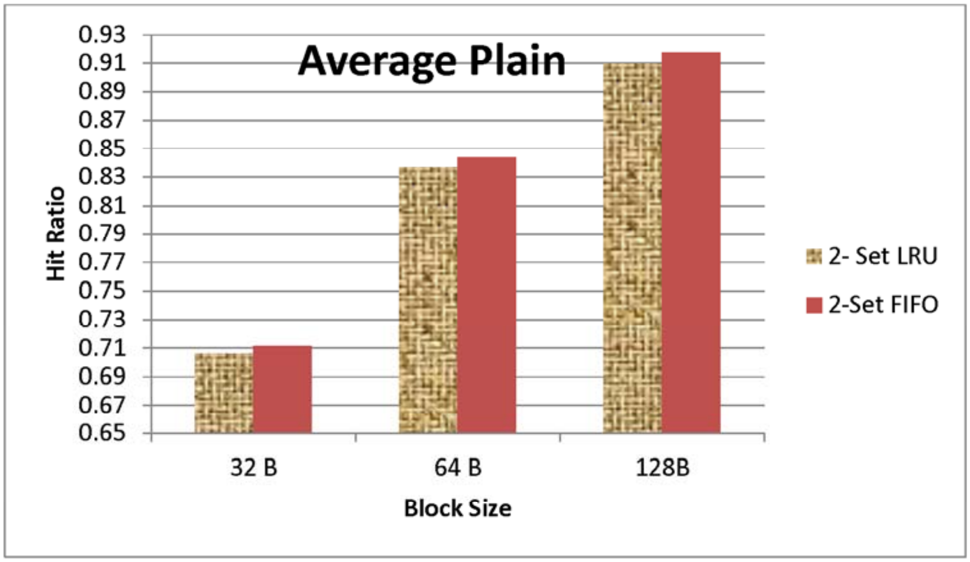


Figure 5.18: Hit ratio of 2-set LRU and FIFO replacement policies using average plain.

Screen shots for average calculation of 30 elements using interleaving and without interleaving are shown clearly in fig. 5.16. Figure 5.17 displays the graphical results of average calculation with interleaving and bars in fig. 5.18 are for non interleaved memory execution of average calculation using 32B, 64B and 128B block sizes. LRU outperforms FIFO when interleaved while FIFO shows little bit better performance than LRU when non interleaved due to locality of reference.

### 5.7 Conclusion and Future scope

Due to the presence of locality of reference, bubble sort outperforms than merge sort. As average calculation does not require any track of used pages, hence, LRU replacement policy does not performs better than FIFO as LRU has an extra cost of maintaining history of used pages. Hit ratio increases with the increase in block size. Moreover, memory map multiprocessor simulator evaluates comparable results as that of similar environment on Simple Scalar simulator which proves the correctness of memory map multiprocessor simulator.

The contribution towards this research is published and is as follows:

Shaily Mittal and Nitin, *Memory Map: A Multiprocessor Cache Simulator*, Journal of Electrical and Computer Engineering, Hindawi Publishing Corporation, Volume 2012, DOI:10.1155/2012/365091, September 2012, pp. 1-12. [Indexed in SCOPUS, DBLP]

Table 5.1: Experimental results calculating hit ratio for 2-set LRU and 2-set FIFO using block sizes 32, 64 and 128 B on different algorithms

| **Block Size** | **2- Set LRU** | **2-Set FIFO** | **Algorithm** |
|---|---|---|---|
| 32 B | 0.7846 | 0.7689 | Average-Interleaving |
| 64 B | 0.7603 | 0.7858 | Average-Interleaving |
| 128 B | 0.8111 | 0.8346 | Average-Interleaving |
| 32 B | 0.7059 | 0.7117 | Average-Plain |
| 64 B | 0.8367 | 0.8442 | Average-Plain |
| 128 B | 0.9092 | 0.9177 | Average-Plain |
| 32 B | 0.7539 | 0.7514 | Bubble Sort |
| 64 B | 0.859 | 0.8574 | Bubble Sort |
| 128 B | 0.9231 | 0.9222 | Bubble Sort |
| 32 B | 0.7628 | 0.7579 | Merge Sort-Interleaving |
| 64 B | 0.7444 | 0.7393 | Merge Sort-Interleaving |
| 128 B | 0.7916 | 0.765 | Merge Sort-Interleaving |
| 32 B | 0.6712 | 0.6717 | Merge sort - Plain |
| 64 B | 0.8067 | 0.8114 | Merge sort - Plain |
| 128 B | 0.884 | 0.8874 | Merge sort - Plain |

# CHAPTER 6
# AN EFFICIENT TAG-BASED DUAL MAPPING REPLACEMENT POLICY

## 6.1 Introduction

As embedded systems became more and more intricate, applications became very huge and caches became foreseeable [E.P. Markatos, 1994]. In the last decade, cache architectures have entered into the world of embedded processors. At present embedded processors, have cache architectures as intricate as general-purpose processors [T.L. Johnson, 1997]. The cache architecture of embedded processor, particularly on mobile devices, is convoluted, as all three metrics of performance, power and area have to be contented within the given constraints. Today in the epoch of multi-core, there is an inclination towards multi-processors embedded systems with low area and frequency connected together with high throughput interconnect fabric [K.S. McKinley, 1996]. Hence, there is a need to condense the complexity and power of individual processor components like caches and other memories with small reduction in system performance [A. Agarwal, 1993].

In computing, when the cache is full, we require cache replacement algorithms. The algorithm must decide the items to be abandoned to make space for the new ones. Cache replacement policies determine the data space to replace whenever there is a miss. In a set-associative cache, a miss occurs when the accessed cache set is full. The three widely used replacement policies are LRU (least recently used) [C. CaBcaval, 2003], FIFO (first in first out) [A. Kalavade, 2003] and RAND (random) [J.L. Hennessy, 2003]. Among these, LRU is the most preferred cache replacement policy. LRU replacement policy is most closely related to the concept of temporal locality. Temporal locality assumes that when a particular memory location is accessed by a processor for a process, then it has high probability of referencing the same location again in the near future. In this case, it is necessary to store a copy of the referenced data in special memory storage [Y. Yan, 2000], so that it can be accessed faster in future reference. This is called temporal locality.

In [Q.M. Jaleel, 2007], Qureshi and other coauthors proposed a DIP replacement policy that chooses the appropriate policy to be applied to the cache from either LRU or Bimodal

Insertion Policy. In [Q.M. Lynch, 2006], authors proposed utilizing Memory-Level-Parallelism (MLP) to reduce the miss penalty to the memory by producing the impression of the MLP-aware replacement policy. Their proposal was based on the truth that cache misses do not occur consistently across the workload that means some misses occur in parallel and others occur in isolation. This means that different misses of the cache blocks will differ in their exploitation of MLP. In [R. Subramanian, 2007], Subramanian proposed an adaptive policy that dynamically chooses from one of the four well-known policies LRU, LFU, FIFO and Random to be applied. In their simulation project, the adaptive policy is implemented for LRU and LFU only. In this proposal, Subramanian et al. used the Sampling Based Adaptive Replacement, which in turn employs auxiliary tag directories for one of the policies, and contribute sets from the cache for the other policy.

## 6.2 Proposed Replacement Policy

In the new proposed policy, access information of each cache block is recorded and further that information is used to select the victim block. Address of 32 bits is divided into 16 bits as tag bits and rest 16 bits for dual mapping. Hence, total number of tags is limited to 65536 tags. Figure 6.1 shows the implementation of the proposed dual mapping replacement policy for a 1M 16-way cache with a line size of 128B. The high order 16 bits of each address is read out as tag and stored in registers. The rest 16 bits are used to generate the effective address of the data. Each 8 higher order bits of this 16 bit field is fed to the 8×256 decoder to generate 256 new address blocks. Then the lower 8 bits are again fed to 8 x 256 decoder of their respective higher order bits combination. Cache blocks with the same 16 bits of tag and higher order 8 bits of address share the same status bits. Moreover, the information memory contains 256 5-bit counters to record the access frequency of each cache block and 1-bit counter to record the validity of each block ( data is present ). The proposed dual replacement thus uses both the frequency and validity information of the blocks.

On a cache hit, the low order 8-bit of the tag is used to decode the status bits for updating the access information of the tag. The valid bit is set to 1 and the frequency counter is incremented by one. If the frequency counter reaches to the value with all 1's then all the 256 frequency counters are divided by two through right shifting all the counters by one bit position to make them at equal level. On a cache miss, all the low order 8-bit of tags in the set is read from the tag register. These tags are used to access the status bits to determine the victim block using the following procedure. When there is a cache miss, first the proposed tag based dual replacement policy uses the validity information (valid bits). All the 256 valid bits of the 256 cache blocks in the missed set are read out. The victim block is chosen from the cache blocks whose valid bits are zero as the bit is set in case of a hit. If all the valid bits in the set are all ones, then the frequency counter with lowest value is chosen from the block as victim block.

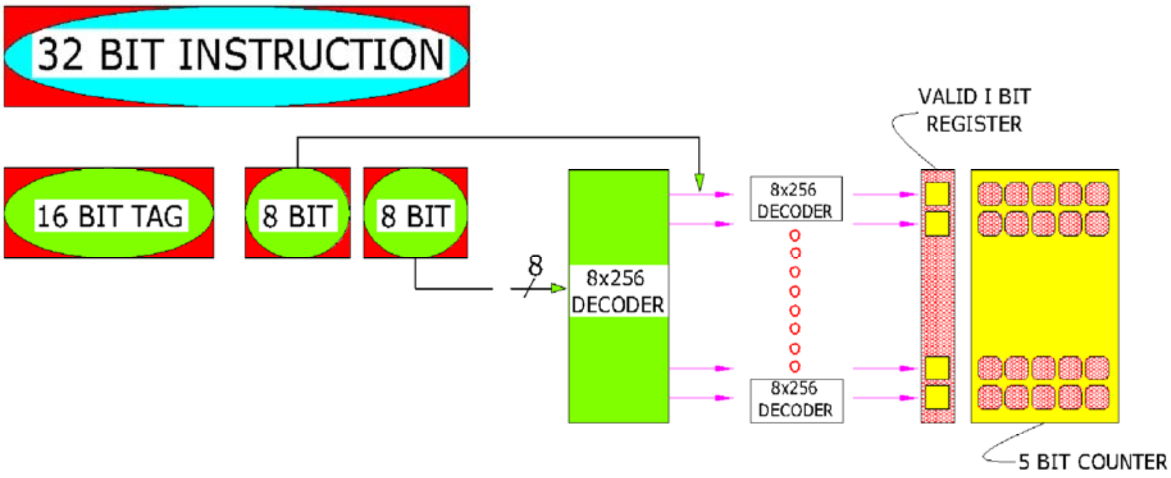


Figure 6.1: Architecture of proposed tag based dual replacement policy

### 6.3 Experimental Methodology and Results

### 6.3.1 Cache Configuration and Benchmarks

Simple Scalar tool [P.R. Panda, 2004] set is used to measure cache hit ratio and CACTI 5.3 tool [P. Shivakumar, 2001] to evaluate processor performance through power of tag-based dual mapping replacement policy. The sim-cache tool of Simple Scalar simulator is extended to implement new proposed replacement policy. The energy consumption for the cache memory has been obtained by isolating the data array subsystem out of the CACTI cache model 5.3. CACTI incorporated cache access time, cycle time and power model [I. Issenin, 2008]. CACTI is planned for use by computer architects to better understand the performance tradeoffs in different cache sizes and organizations. The number of simulated instructions in each benchmark is 250 M instruction. In addition, a fast forward interval of 50 M instructions is included to make caches stable and to obtain correct results. The cache hit ratio and power are the primary metrics. The L1 cache parameters were steady for all experiments. The L2 cache is 1MB 16-way set associative. All L1 and L2 cache in the baseline use a 64 B block. The extent of instruction window is 128 instructions. The parameters are summarized in table 6.1. The address bits are 32 bits. 5 benchmarks suite (Matrix multiplication, Heap Sort, Bubble sort, Biquad and Merge sort) are evaluated as they are parallel applications best suited for multiprocessor environment [I. Issenin, 2006]. All the benchmarks are precompiled for the Alpha binaries ISA.

Table 6.1: Cache configurations used in evaluating cache hit ratio and power for comparing tag based dual replacement policy to LRU and FIFO in Simple Scalar simulator

| **Parameter Name** | **Settings** |
|---|---|
| L1 Instruction Cache | 64KB; 64B line-size; 2-way with LRU replacement Policy. |
| L1 Data Cache | 64KB; 64B line-size; 2-way with LRU replacement Policy. |
| L2 Baseline | 1MB; 32B, 64B, 128 B block size; 16-way, LRU |
| Branch Predictor | Default predictor |
| Instruction fetch queue | 4 instruction/cycle |
| Decode width | 16 |
| Window Size | 128 |
| Memory Latency | 10 cycles |

### 6.3.2 Experimental Evaluation

Figure 6.2 shows the cache hit ratio of 5 benchmarks of block sizes 32 B, 64B and 128 B for LRU, FIFO and Tag based Dual replacement policy with 2- way associativity. Firstly, as it is shown from the bars, with the increase in block size, hit ratio increases. As in the case of Heap sort, its value rises from 69.5 to 89.8. Analogous performance is seen in case of other benchmarks also. Secondly, the hit rate of the proposed tag based dual mapping replacement policy is always greater than two other replacement policies namely LRU and FIFO for all benchmarks. For example, in case of matrix multiplication, hit ratio rises from 78.9(LRU) and 75.89(FIFO) to 84.11(TDMRP).On average, the hit ratio of dual replacement policy is ((highest$_{\text{dual}}$ –highest$_{\text{LRU/FIFO}}$) highest$_{\text{dual}}$)) i.e. (94.39- 88.21)/94.39 = 7% and (94.39 – 82.12)/94.39 = 13% higher than the LRU and FIFO respectively.

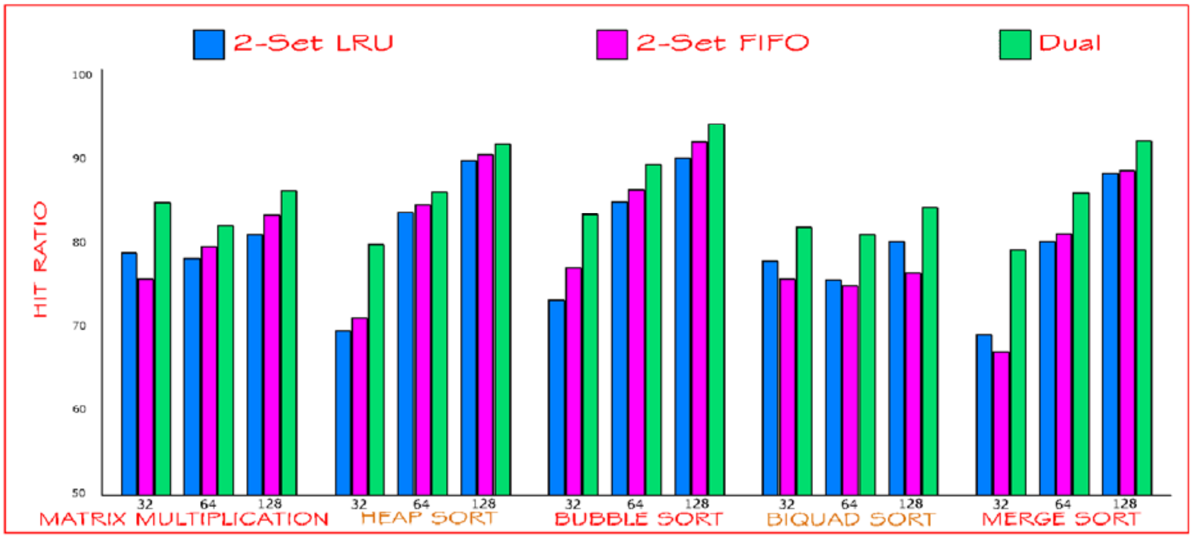


Figure 6.2: Hit ratio of 32B, 64B and 128 B block size of LRU, FIFO and dual replacement policies

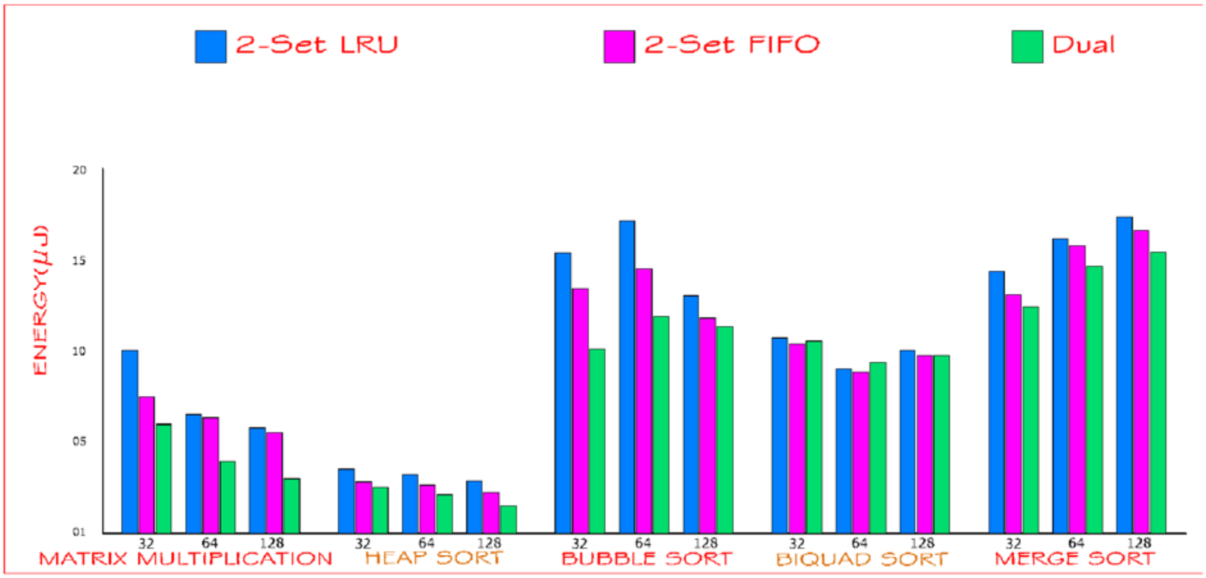


Figure 6.3: Energy estimation (µJ) of 32B, 64B and 128 B block size of LRU, FIFO and dual replacement policies

Figure 6.3 explains the energy estimation of same 5 benchmarks of block sizes 32 B, 64B and 128 B for LRU, FIFO and Tag based Dual replacement policy with 2- way associativity. Firstly, increase in block size results in decline of energy consumption. Similarly, for merge sort, energy consumption decreases from 3.2 to 2.9 as block size increases from 32B to 128 B. All benchmarks have the similar characteristics. Secondly, as clearly seen from the bars, the energy consumption of the proposed tag based dual mapping replacement policy is always lower for all benchmarks than two other replacement policies LRU and FIFO. For example as seen from the bars, energy value of LRU (10.2) and FIFO (7.5) to TDMRP (6) in matrix multiplication. On average, the power estimation of dual replacement policy is (2.9 - 1.5)/2.9 = 48% and (2.3 – 1.5)/2.3 = 35% lower than the LRU and FIFO respectively. Lowest values have been taken to calculate the average decrease in power.

### 6.4 Conclusion and Future Work

A new, efficient, high performance cache replacement policy called tag based dual mapping replacement policy (TDMRP) is proposed that raises hit ratio and diminishes the power consumption when compared to two well-known replacement policies LRU and FIFO. Also three replacement policies (one proposed and two existing policies) are compared on the basis of five well-known benchmarks namely matrix multiplication, heap sort, bubble sort, biquad and merge sort. The two parameters hit ratio and energy are measured using SimpleScalar and CACTI tool. Results have proved that new proposed TDMRP increases hit ratio by 7% and 13% with respect to LRU and FIFO. In the similar manner, it has proved reduction in energy consumption in case of TDMRP as compared to LRU and FIFO by 48% and 35% respectively showing an excellent performance. TDMRP cache replacement policy is explored in terms of hit ratio and energy first as these two is on the priority for embedded systems.

Hit ratio increases as this policy is taking the advantages of LRU and using the concept of status bits. Energy consumption is less as number of memory accesses is less as compared to the memory accesses in FIFO and LRU due to less frequent changes in data in memory and hence lesser main memory accesses.

The contribution towards this research is published and is as follows:

Shaily Mittal and Nitin, *An Efficient Tag-based Dual Mapping Replacement Policy*, Proceedings of the First International Conference on Information Science and Management (ICoCSIM), Toba Lake, North Sumatera, INDONESIA, Volume 1, pp 93-97, December 3-5, 2012.

# CHAPTER 7

# A NEW EFFICIENT REPLACEMENT POLICY FOR SCRATCH PAD MEMORY

## 7.1 Introduction and Motivation

With the advances in time, there is increase in processor memory speed gap. Scratch-pad memory (SPM) has been used mostly in the design of modern embedded system processors [R. Banakar, 2002] to reduce this memory speed gap in processors.

Cache associativity has direct impact on cache performance. Cache replacement policy is required when there is no space for the new block in cache memory. Hence, to accommodate the new block, some of the old blocks need to be replaced. Same procedure is implemented for Scratch pad memory. The authors have designed the proposed model with Scratch pad memory and implemented new replacement policy on SPM. The present processors employ various cache replacement policies such as LRU (Least Recently Used) [A. Kalavade, 2000], Random [J.L. Hennessy, 2003], FIFO (First in First Out), PLRU (Pseudo LRU) [S. Borkar, 2007] and N-HMRU [S. Roy, 2009].

Udayakumaran and his co-authors in [S. Udayakumaran, 2003, 2006] had proposed a compile time SPM allocation approach tailored at the entry point of each loop or function. In these papers author’s key focal point was only on stack data and global variables in the allocation of SPM, hence they proposed an additional approach of taking account of heap data into the SPM allocation in [A. Dominguez, 2005]. In broad-spectrum, the most momentous conclusion made by them is that the SPM allocation is supposed to be adapted at the start point of each loop or function.

In [M. Soryani, 2007] authors discussed the simulation-based performance evaluation in cache design issues such as cache size, associativity and replacement policies in embedded processors. They evaluate results using SPEC CPU 2000 benchmark showing the gain of increasing associativity more in the case of data and unified caches in contrast to instruction caches. Moreover, they verify through their results that random policy performed practically better than LRU and MRU for instruction caches. In their research, they also compared LRU to MRU policy and according to results MRU comes out with insignificant miss rate deprivation. They have evaluated and compared replacement policies in cache memory only and not in SPM.

Wilton et al. in [S. J. E. Wilton, 1994, 1996] proposed a cache model named CACTI to evaluate energy consumed in Joules. Detailed values of energy consumption of both memories are important for a comparison between scratchpad memory and cache memory. This was done by the algorithm proposed in [D. Cho, 2009] that places selected program parts and variables into scratchpad memory as part of a compiler. The ILP model presented in this paper seems to be an optimal solution as it saves about 22% of the electrical energy as compared to a cache memory.

In [S.W. Huang, 2009], Sheng-Wei Huang and others had effectively built up a new page-based SPM allocation approach in embedded systems. The efficiency of the proposed approach was evaluated by simulating a set of programs. Obtained performance evaluation results prove the effective enrichment in hit rate by the use of proposed approach however also reduce the cost of the SPM reallocation. Hence, as per their results, the energy-delay-product (EDP) values of the test programs were significantly improved by 60% in average.

In [M. Qureshi, 2006], Qureshi with his co-authors proposed a new scheme for utilizing Memory-Level-Parallelism (MLP) to reduce the miss penalty by producing the impression of the MLP-aware replacement policy. Their proposal was based on the truth that some cache misses occur in parallel and others occur in isolation means that cache misses do not occur consistently across the workload. This means that different misses of the cache blocks will differ in their exploitation of MLP. In [M. Qureshi, 2007] authors proposed a DIP replacement policy that chooses the best suited policy to be applied to the cache from either LRU or Bimodal Insertion Policy. In contrast, Subramanian in [R. Subramanian, 2002], proposed an adaptive policy that also dynamically chooses from one of the four well-known policies LRU, LFU, FIFO and Random to be applied. While in their simulation project, they implemented adaptive policy for LRU and LFU only. In this proposal, Subramanian used the Sampling Based Adaptive Replacement, which in turn employs auxiliary tag directories for one of the policies, and contribute sets from the cache for the other policy.

The goal of this chapter is to implement earlier proposed cache replacement policy [S. Mittal, 2012] named tag based dual replacement policy in SPM and compares it with architecture designed without using SPM.

### 7.2 Proposed Policy

In the new proposed policy, previously recorded access information of each cache block is used to select the victim block. Address bits of 32 is divided into two parts, in which16 bits are treated as tag bits and rest 16 bits for dual mapping. Hence, total number of tags is limited to 65536 tags. The implementation of the proposed dual mapping replacement policy for a 1M cache with a line size of 128B is shown in fig. 7.1. The high order 16 bits of each address is stored in registers. The rest lower order 16 bits are used to generate the effective address of the data. Further, each 8 higher order bits of this 16 bit lower order field is fed to the 8×256 decoder to generate 256 new address blocks. Then the remaining lower 8 bits are also fed to 8 x 256 decoder of their respective higher order bits combination.  Cache blocks having same tag bits and higher order address bits share the same status bits. Moreover, the information memory contains 5-bit counters numbered from 0 to 255 to keep record of the access frequency of each cache block and 1-bit counter to record the validity of each block ( data is present ). Thus, both the frequency and validity information of the blocks are used in dual replacement policy.

When there is a cache hit, the low order 8-bit of the tag is used to decode the status bits for updating the access information of the tag. At the same time, corresponding valid bit is set to 1 and the frequency counter is incremented by one. If the frequency counter reaches to the maximum value of 255 with all 1’s then all the 256 frequency counters are divided by two through right shifting all the counters by one bit position to make them at equal level. When there is a cache miss, all the low order 8-bit of tags in the set is read from the tag register. These tags are used to access the status bits in order to determine the victim block. On a cache miss, first the proposed tag based dual replacement policy uses the valid bits. All the 256 valid bits of the cache blocks in the missed set are read out. The victim block is elected from the cache blocks whose valid bits are zero as the bit is set in case of a hit. If all the valid bits in the set are all ones, then the frequency counter with lowest value is chosen from the blocks as victim block.

### 7.3 Experimental Setup and Evaluation

The simulator used is SimpleScalar [P.R. Panda, 1997]. SimpleScalar simulator provides an infrastructure for simulation and architectural modelling in an easier manner. The complete toolset provide a mixture of platforms ranging from simple processors to detail dynamically scheduled micro architectures with multiple-level memory hierarchies as used.

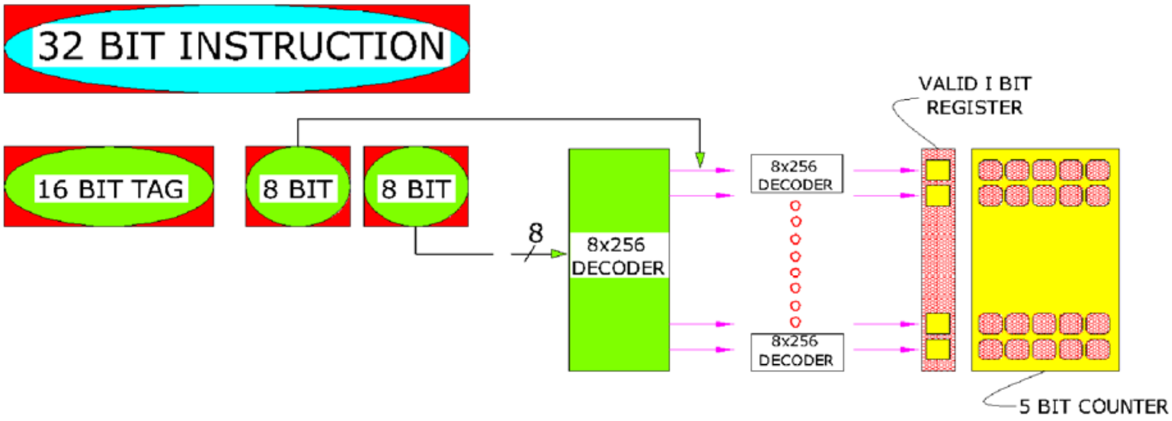


Figure 7.1: Architecture of proposed tag based dual replacement policy in SPM

Performance evaluation of different cache replacement policies FIFO, LRU and Random has been done using Sim-cache simulator from the alpha version of this toolset and later extended to implement proposed replacement policy. For this work, two models are compared, one with a cache and second with a scratchpad memory as well as cache. The original simulator is modified to support hierarchical two level memory architecture as well as four replacement policies for newly employed SPM including new dual replacement policy. All memory accesses were dump to a trace file and then this trace file was analyzed to get the value of memory accesses, the footprint, and accesses into the statically declared array that represented scratch pad memory in the source code. A very small size SPM has been taken to match up with cache memory. Size of L1 data cache is 64KB. For each 6 benchmarks considered, a number of simulations have been run for D1 data cache organizations with 1, 2, 4, 8 and 16 way associativity and replacement policies as random, FIFO, LRU and Dual. Table 7.1 describes the cache configuration in detail.

In the similar manner, diverse simulations have been performed for SPM organizations of 1, 2, 4, 8 and 16 way associativity in concurrence with cache having different combination of replacement policies and associativity. A list of 2000 elements is captured to sort in case of all sorting algorithms. In the similar fashion, matrix multiplication act upon matrices of size 200 X 200 for large calculations.

Table 7.1 lists the detailed cache configurations used in simulation environment. Authors run 6 benchmarks suite (Matrix multiplication, Insertion sort, Biquad, Quick sort, Merge sort and Red black tree) as they are parallel applications best suited for multiprocessor environment [I. Issenin, 2006]. All the benchmarks are precompiled for the alpha binaries ISA to be used in Simple Scalar simulator.

### 7.4 Observations

This chapter compares and evaluates newly proposed replacement policy in SPM and also without SPM. Figure 7.2 and fig. 7.3 show the cache miss rate value for 6 benchmarks using four different replacement policies FIFO, LRU, Random and dual replacement policy using SPM and without using SPM for direct mapped and 2-way associative cache. It is clearly visible from the bar graphs that newly proposed dual replacement policy have lower miss rate in comparison to FIFO, LRU and Random for both associativity values 1 and 2. Similarly, table 7.2 lists the values of miss rate of various benchmarks using different replacement policies for associativity value as 4. The values in the table clearly shown the best results i.e. lowest cache miss rate in case of dual replacement policy as compared to other three well known replacement policies while using SPM as well as without using SPM. Figure 7.4 and fig. 7.5 displays the cache miss rates values for 4-way and 8-way associative caches. The bar graph in fig. 7.6 describes the cache miss rate chart for fully associative mapped cache using 6 different parallel benchmarks and four replacement policies. Similar results are obtained in this case also with lowest cache miss rate for dual replacement policy rather than others. Likewise, results are also obtained for associativity value of 8 with best results shown by dual replacement policy.

Table 7.1: Cache configurations used in simple scalar tool for comparing tag based dual replacement policy to LRU, FIFO and random in SPM.

| **Parameter Name** | **Settings** |
|---|---|
| L1 Instruction Cache | 64KB; 64B line-size; 2-way with LRU replacement Policy. |
| L1 Data Cache | 64KB; 64B line-size; 2-way with LRU replacement Policy. |
| L2 Baseline | 1MB; 32B, 64B, 128 B block size; 16-way, LRU |
| Branch Predictor | Default predictor |
| Instruction fetch queue | 4 instruction/cycle |
| Decode width | 16 |
| Window Size | 128 |
| Memory Latency | 10 cycles |

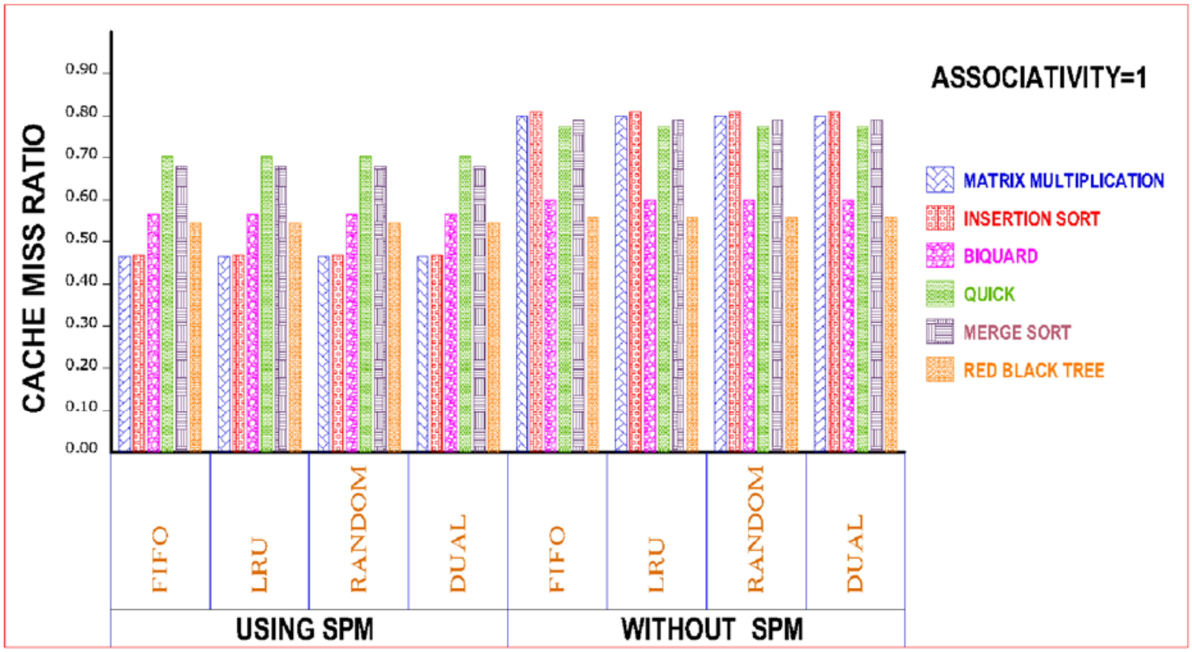


Figure 7.2: Cache miss rate for FIFO, LRU, random and dual replacement policies using and without using SPM implementing 6 benchmarks for direct mapped associative cache

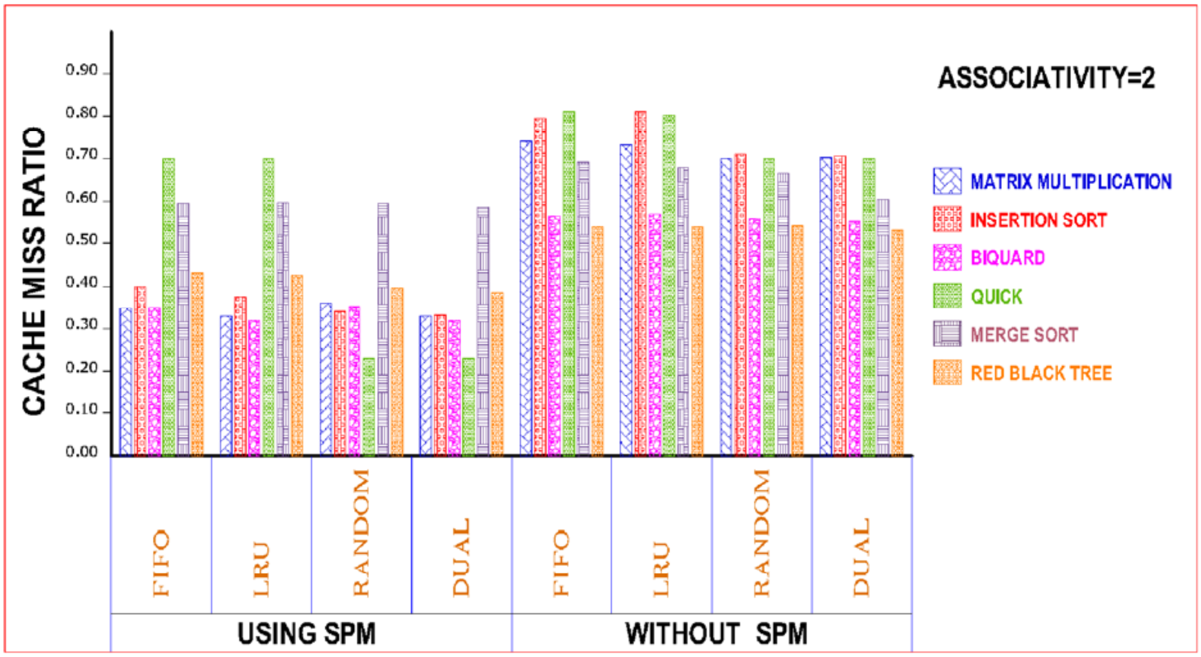


Figure 7.3: Cache miss rate for FIFO, LRU, random and dual replacement policies using and without using SPM implementing 6 benchmarks for 2-way associative cache.

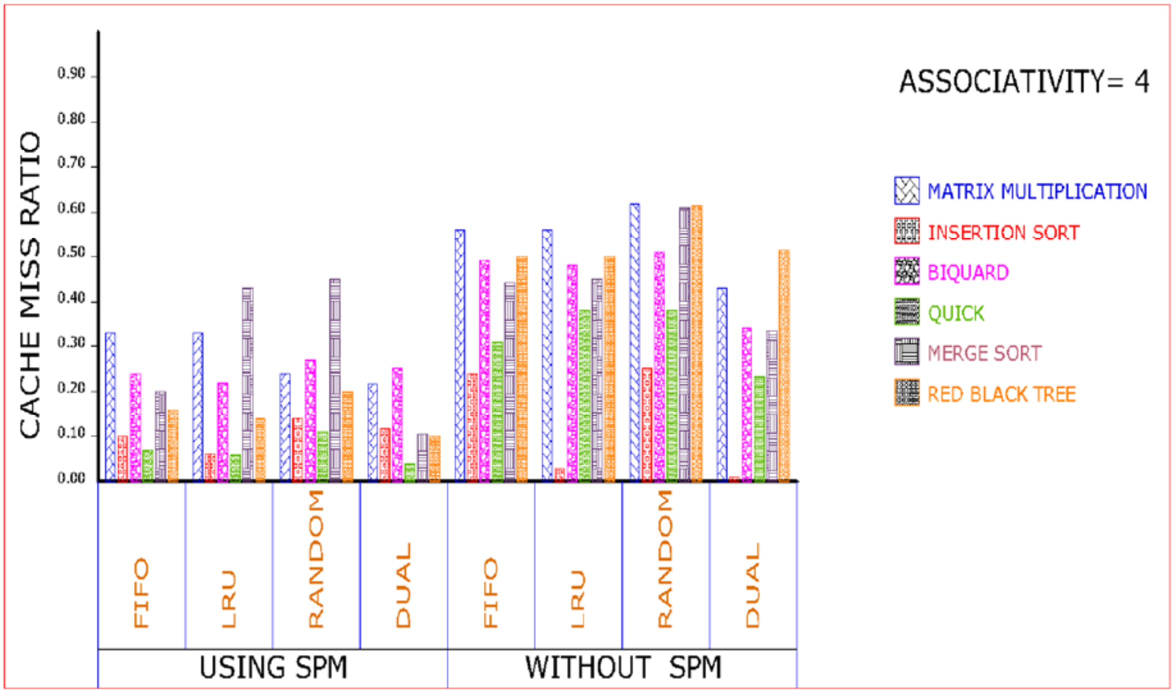


Figure 7.4: Cache miss rate for FIFO, LRU, random and dual replacement policies using and without using SPM implementing 6 benchmarks for 4-way associative cache

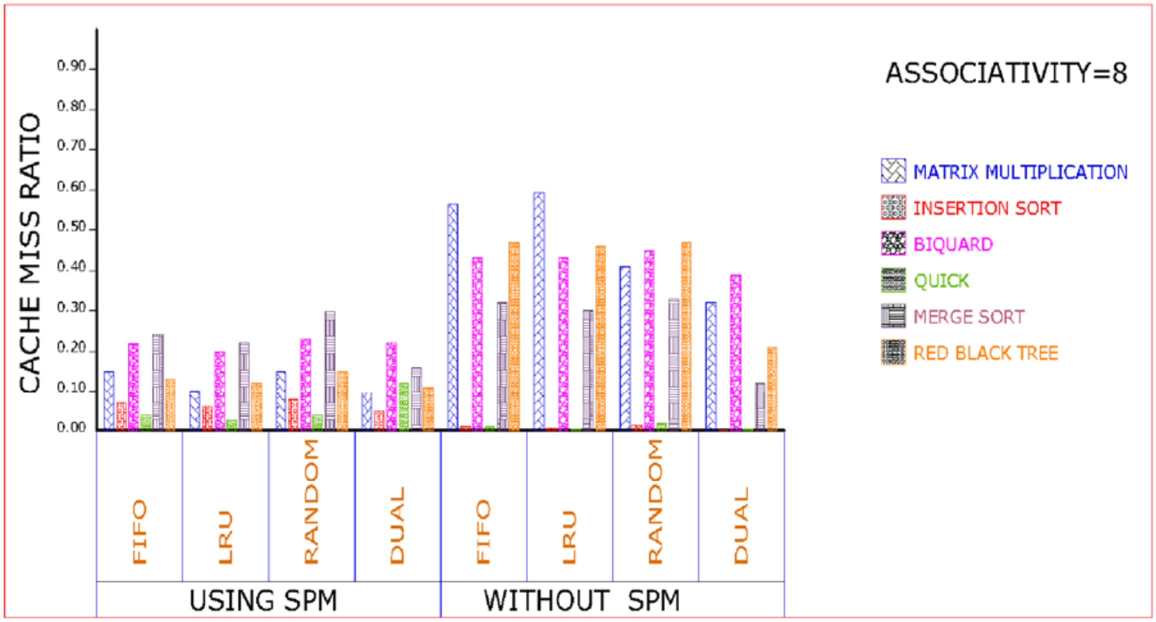


Figure 7.5: Cache miss rate for FIFO, LRU, random and dual replacement policies using and without using SPM implementing 6 benchmarks for 8-way associative cache.

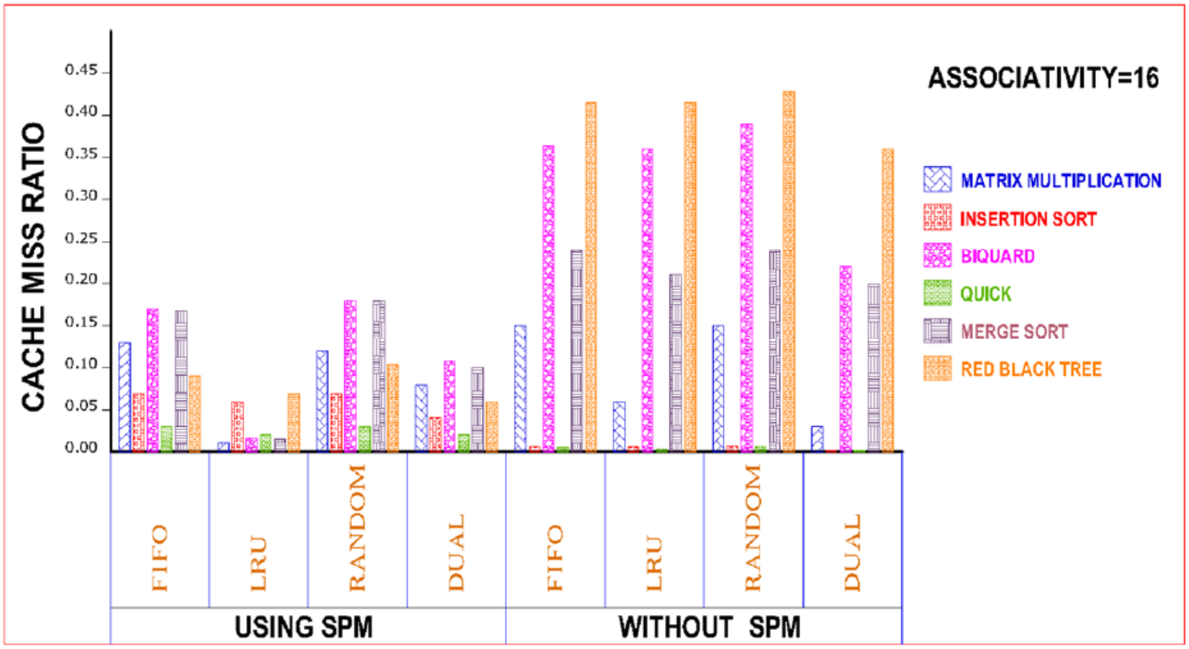


Figure 7.6: Cache miss rate for FIFO, LRU, random and dual replacement policies using and without using SPM implementing 6 benchmarks for fully associative cache.

Table 7.2: Cache miss rate values for FIFO, LRU, random and TDMRP in SPM and cache

| | | Using SPM | | | | Without SPM | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Benchmark** | **Associativity** | **FIFO** | **LRU** | **RANDOM** | **Dual** | **FIFO** | **LRU** | **RANDOM** | **Dual** |
| Matrix Multi. | 1 | 0.4654 | 0.4654 | 0.4654 | 0.4654 | 0.7993 | 0.7993 | 0.7993 | 0.7993 |
| Insertion sort | | 0.4688 | 0.4688 | 0.4688 | 0.4688 | 0.81 | 0.81 | 0.81 | 0.81 |
| Biquad | | 0.5675 | 0.5675 | 0.5675 | 0.5675 | 0.595 | 0.595 | 0.595 | 0.595 |
| Quick | | 0.7047 | 0.7047 | 0.7047 | 0.7047 | 0.7764 | 0.7764 | 0.7764 | 0.7764 |
| Merge sort | | 0.6832 | 0.6832 | 0.6832 | 0.6832 | 0.7827 | 0.7827 | 0.7827 | 0.7827 |
| Red Black Tree | | 0.5452 | 0.5452 | 0.5452 | 0.5452 | 0.5592 | 0.5592 | 0.5592 | 0.5592 |
| Matrix Multi. | 2 | 0.3331 | 0.3325 | 0.3655 | 0.3301 | 0.7381 | 0.7374 | 0.7089 | 0.7009 |
| Insertion sort | | 0.4058 | 0.3728 | 0.3452 | 0.3354 | 0.7931 | 0.8134 | 0.713 | 0.7069 |
| Biquad | | 0.3493 | 0.328 | 0.3557 | 0.3197 | 0.5689 | 0.5702 | 0.5629 | 0.5576 |
| Quick | | 0.7043 | 0.7043 | 0.2345 | 0.2341 | 0.8106 | 0.8086 | 0.7064 | 0.702 |
| Merge sort | | 0.5918 | 0.5973 | 0.5952 | 0.5876 | 0.6902 | 0.6787 | 0.6635 | 0.6565 |
| Red Black Tree | | 0.4318 | 0.4264 | 0.3936 | 0.3848 | 0.5418 | 0.5428 | 0.5451 | 0.5345 |
| Matrix Multi. | 4 | 0.33 | 0.3302 | 0.2427 | 0.2134 | 0.5651 | 0.5642 | 0.6184 | 0.4335 |
| Insertion sort | | 0.1044 | 0.0629 | 0.1406 | 0.1198 | 0.2496 | 0.0306 | 0.2516 | 0.0123 |
| Biquad | | 0.2477 | 0.2289 | 0.2706 | 0.2534 | 0.4983 | 0.4893 | 0.5117 | 0.3423 |
| Quick | | 0.0738 | 0.059 | 0.1145 | 0.045 | 0.3173 | 0.3897 | 0.3799 | 0.2349 |
| Merge sort | | 0.2036 | 0.4332 | 0.454 | 0.1034 | 0.441 | 0.4586 | 0.5128 | 0.3345 |
| Red Black Tree | | 0.1632 | 0.1422 | 0.2052 | 0.1023 | 0.5073 | 0.5022 | 0.5145 | 0.5132 |
| Matrix Multi. | 8 | 0.1494 | 0.1002 | 0.1564 | 0.0987 | 0.5656 | 0.5952 | 0.4096 | 0.321 |
| Insertion sort | | 0.0795 | 0.0619 | 0.0885 | 0.0519 | 0.0141 | 0.0088 | 0.0185 | 0.0012 |
| Biquad | | 0.2188 | 0.2073 | 0.2314 | 0.2221 | 0.4371 | 0.4316 | 0.4557 | 0.3956 |
| Quick | | 0.0444 | 0.0297 | 0.0474 | 0.1234 | 0.011 | 0.0044 | 0.0229 | 0.0021 |
| Merge sort | | 0.249 | 0.2187 | 0.291 | 0.165 | 0.323 | 0.3082 | 0.338 | 0.1289 |
| Red Black Tree | | 0.132 | 0.1227 | 0.1501 | 0.1101 | 0.4711 | 0.464 | 0.4732 | 0.2198 |
| Matrix Multi. | 16 | 0.1201 | 0.1002 | 0.1243 | 0.0789 | 0.1561 | 0.061 | 0.1522 | 0.0349 |
| Insertion sort | | 0.0679 | 0.0599 | 0.0718 | 0.0453 | 0.0062 | 0.0055 | 0.0079 | 0.0022 |
| Biquad | | 0.1719 | 0.1622 | 0.1867 | 0.1093 | 0.3677 | 0.3603 | 0.3904 | 0.2265 |
| Quick | | 0.0359 | 0.0297 | 0.0364 | 0.0234 | 0.0053 | 0.0039 | 0.0062 | 0.0019 |
| Merge sort | | 0.1649 | 0.151 | 0.1851 | 0.1003 | 0.24 | 0.2125 | 0.2436 | 0.1986 |
| Red Black Tree | | 0.0884 | 0.0791 | 0.1047 | 0.0678 | 0.4181 | 0.4151 | 0.4292 | 0.3675 |

### 7.5 Conclusion and Future Work

In this chapter, a simulation based evaluation of cache with and without scratch pad memory in terms of main memory design issues like replacement policies and associativity in embedded systems is presented. The simulation results demonstrate the reduction in cache miss rate up to 20% by the use of newly proposed dual replacement policy.

Hence, the addition of on-chip memory SPM and use of dual replacement policy in SPM build the system more efficient with lesser no. of miss rates. Future work will improve these results by considering dynamic moving memory data in and out of the scratchpad with evaluation of more parameters for comparison. In addition, research can be done for the extension of this approach to multiprocessor systems on chip (MPSoC).

The contribution towards this research is published and is as follows:

Shaily Mittal and Nitin, *A New Efficient Replacement Policy for Scratch Pad Memory,* Proceedings of the IEEE 5th International Conference on Computational Intelligence and Communication Networks CICN-2013, Mathura, INDIA, pp 432-435, September 27-29, 2013.

# CHAPTER 8

# A NOVEL DIRECTORY BASED SOLUTION TO CACHE COHERENCE PROBLEM

## 8.1 Introduction

MPSoC with shared memory have been considered notably for research purpose. [S. A. Hothali, 2010] The programming model used in MPSoC is not complicated and hence they can provide better results. There is one benefit of using multiprocessors on a single chip that an address generated by a processor enables the communication between all processors. One major drawback of using multiple caches is the cache inconsistency. The read process doesn't have any problem of having same cache block in multiple private caches of processors. The problem arises in the case of modifying the data in one cache as data in other caches becomes invalid [J. Gomez, 2009]. Cache coherence ensures the presence of valid data in all caches. Cache coherence occurs due to presence of multiple copies of same data in local caches of a shared resource and one copy in the main memory. [D. Patterson, 2009] When there is a change in one copy of the operand, the other copies of the operand present in other caches and main memory must be changed also.

Use of multiple processors on a single chip leads to one or two levels of cache associated with each processor as it directs to required high performance of the system. However, this kind of organization creates cache coherence problem. The cache coherence problem is the occurrence of multiple copies of the same data in different caches simultaneously, and if processors are allowed to update their own copies freely, results in an inconsistent view of memory. The two well known write policies for cache coherency are:

1. Write back: In this policy, updation is made only to the cache memory at time and propagated to the main memory only when the related cache line is empty for any further modification.
2. Write through: In this policy, all updations to data are made to cache memory and at the same also communicated to the main memory. It makes data in main memory valid always.

Write back policy generally leads to inconsistency. If two caches contain the same line, and the line is updated in only one cache, leads to the other cache with unknowingly an invalid value. Consequently, the reading of that invalid line produces invalid results. Data inconsistency can take place yet with the write through policy. The main objective of cache coherence protocol is to have the local variables used recently into the proper cache for numerous reads and write and to maintain consistency of shared variables that might be present in multiple caches at the same time.

## 8.2 Motivation and History

Solutions to cache coherence problem are divided into two parts: software and hardware solutions [J. Archibald, 1986]. Therefore, hardware does not cache those objects. In general terms, the main objective is to avoid caching for any shared data variables. However this is a conventional approach, as shared data may be read-only during some phases and read and write for some other phases. Because the problem is only dealt with when it actually arises, there is more effective use of caches, leading to improved performances over a software approach. Further, cache coherence protocols can be categorized as: snoopy protocols and directory based protocols [C. P. Thacker, 1987].

The schemes that have been or are being implemented most in multiprocessors are called snoopy cache protocols [S. J. Frank, 1984][M. S. Papamarcos, 1985] because all coherency transactions must be watched by system cache to determine when consistency-related actions should take place for shared data. Snoopy protocols can use [L. Rudolph, 1985] write invalidates and write- update approaches. In write invalidate protocol; cache line becomes exclusive when one of the caches wants to write to the cache by invalidating all data lines in the other caches. And in write update protocol, there can be multiple writers and readers concurrently. When there is an updation in a shared line, the data to be updated is distributed to all caches and the caches containing that particular line update it immediately.

Another classification of coherency protocols is directory-based protocols [R.H. Katz, 1985][C.K. Tang, 1976][L.M. Censier, 1978][J. Archibald, 1985]. [A. Agarwal, 1988] Directory-based protocols maintain a directory linked with main memory which stores the state of each block of main memory. Corresponding to each entry in centralized directory there exists a dirty bit, a bit indicating the cached status of block and also pointers to the caches storing the repective block.

Shared memory multiprocessor architectures make use of some other cache coherence protocols such as MSI [F. Baslett, 1988], MESI [M. Papamarcos, 1985], Dragon protocol [E. McCreight, 1984] and Berkeley protocol [J. Archibald, 1986], to guarantee data integrity and accuracy when data is shared and cached within each processor. The MOESI protocol have states Exclusive modified, Shared MOdified, Exclusive Clean, Shared Clean and Invalid and it is a variation of MESI protocol. In [B. Hashemi, 2003], author evaluated and compared three snoopy based protocols Dragon, Firefly and WTU using Limes simulator. According to their results, WTU protocol has low performance as compared to Firefly and Dragon as precision block sharing information in WTU protocol is low however high in Firefly and Dragon.

## 8.3 Proposed Solution

To address cache coherence problem, a new protocol named Valid Directory protocol is designed. In this scheme, a directory or table in main memory is storing data and status bit. The directory contains only the updated and invalid data from different private caches.

When a processor updates some data in its private cache, it becomes invalid in the cache and its updated value along with the status bit set entered in the main memory directory for future reference of any processor. Invalid data in private caches is directly and only updated in the directory entry in main memory.

When a processor tries to read a data present in its cache, its request has been granted immediately if valid data is present in the cache. In case of invalid data, it goes directly to the main memory to access the required data from the directory.

There is no concept of pointers as in case of earlier directory based protocol and hence its complexity and execution time is less. As the size of the entries associated with each block in the directory is directly proportional to the number of processors, the memory consumed in directory is proportional to the size of memory Θ (n), where n is the number of processors and multiplied by the size of each directory entry Θ (n). Hence, the total memory overhead complexity in this case is the square of the number of processors i.e. Θ ($n^2$).

## 8.4 Experimental Setup and Test bed

LIMES simulator is used for the purpose of evaluation and comparison of different protocols (Valid directory, Dragon, Berkeley and MESI). Limes is a tool which provides environment for simulation of multiprocessors. It runs on a computer with Linux operating system and hence the name Linux Memory Simulator. Limes contains N processors ( $N = 2^m$ ) connected to each other and giving an impression of concurrent execution on a unique processor machine. Results are evaluated using 2, 4, 6 and 8 processors. Actually processes are lightweight threads. A lime uses the execution-driven approach, which ensures maximum simulation speed. Existing applications do not require any kind of source code modifications to compile and run under Limes. Simulation time is expressed in processor cycles of the simulated target multiprocessor system, which accurately reflects the behaviour of a real multiprocessor system at the cost of time spent for simulating references and context switching. Table 8.1 explains complete experimental setup in detail. Simulator is extended to implement newly proposed protocol.

## 8.5 Comparative Analysis of Valid Directory, DRAGON, BERKELEY AND MESI

In this section, a simulation results are discussed. For the purpose of simulations, Limes simulator with protocols implemented in C++ is used. Figure 8.1 illustrates the simulation effect of number of processors on the execution time using FFT application with values tabulated in table 8.2 Valid directory protocol has the lowest execution time in comparison to Dragon, Berkeley and MESI and among them, Berkeley is the worst performer. There is a decrease of 23%, 23.5% and 20% in execution cycles for newly proposed protocol from dragon, Berkeley and MESI respectively. On the other hand, fig. 8.2 presents the execution time of different protocols with increasing number of processors with LU benchmark. Table 8.3 displays the execution time in µsec for LU benchmark. Valid directory protocol performs better than other protocols when using LU benchmark. Dragon and berkeley protocol shows almost similar behaviour with 14% increase in execution time. Figure 8.3 presents the execution time for Ocean application when four different cache coherence protocols are applied. Again the proposed protocol outperforms among all with 30%, 31% and 16%

decrease in execution time w.r.t dragon, berkeley and MESI. Dragon and berkeley perform almost in the similar manner. MESI also have same execution time as that of proposed when number of processors is 16. In the fig. 8.4, again proposed protocol shows best performance with lowest execution time as compared to other protocols. Thesis evaluated these results using radix benchmark. Here, dragon shows the worst result with highest execution time.

Table 8.1: Simulation environment for evaluating execution time for Dragon, Berkeley, MESI and Valid directory cache coherence protocol in LIMES simulator

| | |
|---|---|
| **Simulator** | LIMES |
| **Processors** | 2,4,8,16 |
| **Test Block Size** | 32 B |
| **Protocols** | Dragon, Berkeley, MESI, Valid Directory |
| **Applications** | FFT, LU, Ocean, Radix |

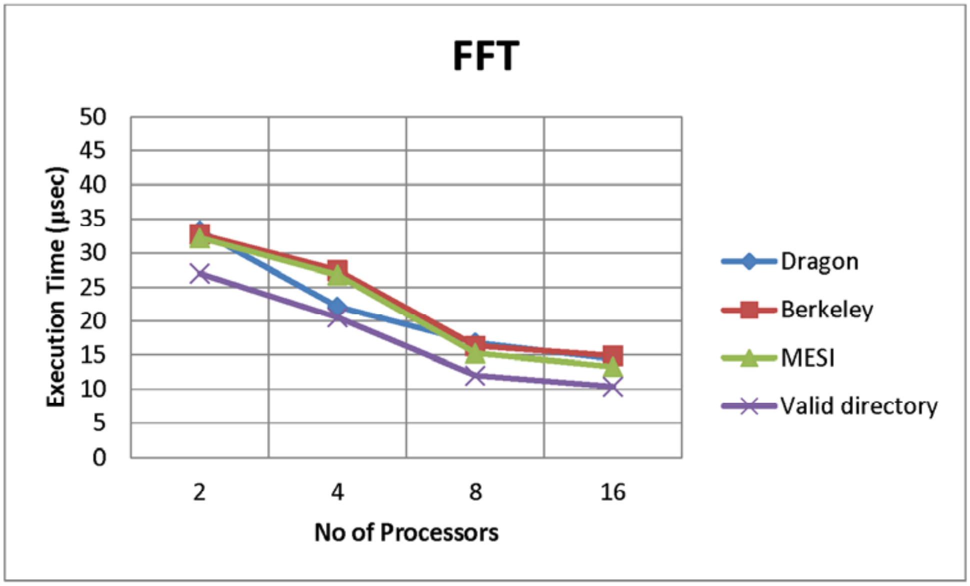


Figure 8.1: Execution cycles in FFT with Dragon, Berkeley, MESI and Valid directory cache coherence protocols

Table 8.2: Execution time (µsec) of Dragon, Berkeley, MESI and Valid directory cache coherence protocols using FFT benchmark in LIMES simulator

| **Benchmarks** | **No. of processors** | | | |
|---|---|---|---|---|
| | **2** | **4** | **8** | **16** |
| **Dragon** | 33.272412 | 22.134532 | 16.78543 | 14.424321 |
| **Berkeley** | 32.745612 | 27.453975 | 16.32342 | 14.873291 |
| **MESI** | 32.214567 | 26.789302 | 15.23901 | 13.26792 |
| **Valid directory** | 27.005672 | 20.534251 | 12.00234 | 10.379043 |

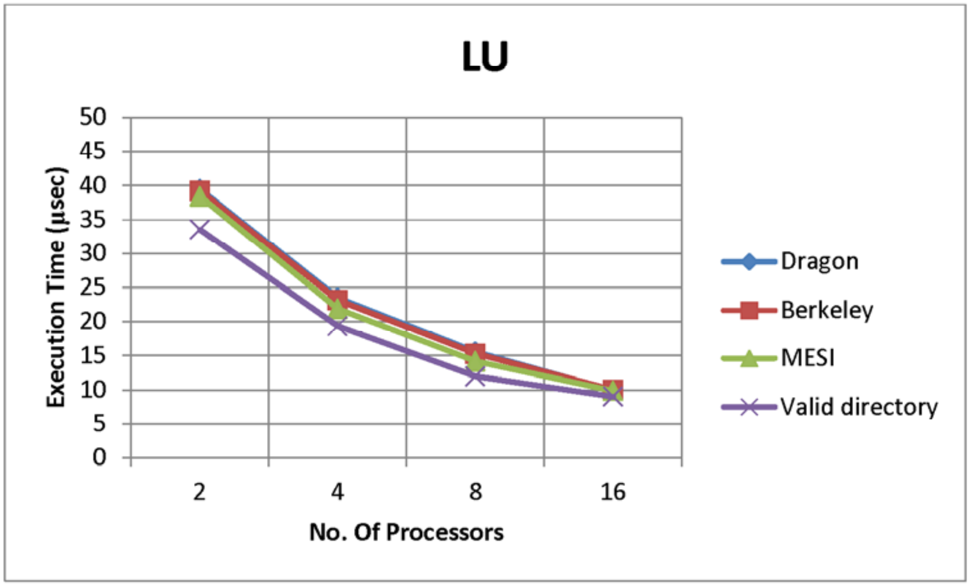


Figure 8.2: Execution cycles in LU with Dragon, Berkeley, MESI and Valid directory cache coherence protocols

Table 8.3: Execution time (µsec) of Dragon, Berkeley, MESI and Valid directory cache coherence protocols using LU benchmark in LIMES simulator

| **Benchmarks** | **No. of processors** | | | |
|---|---|---|---|---|
| | **2** | **4** | **8** | **16** |
| **Dragon** | 39.453211 | 23.456788 | 15.55341 | 9.907185 |
| **Berkeley** | 39.223876 | 23.129063 | 15.28349 | 9.987654 |
| **MESI** | 38.414164 | 21.879023 | 14.17654 | 9.856321 |
| **Valid directory** | 33.556532 | 19.325678 | 11.97653 | 9.023457 |

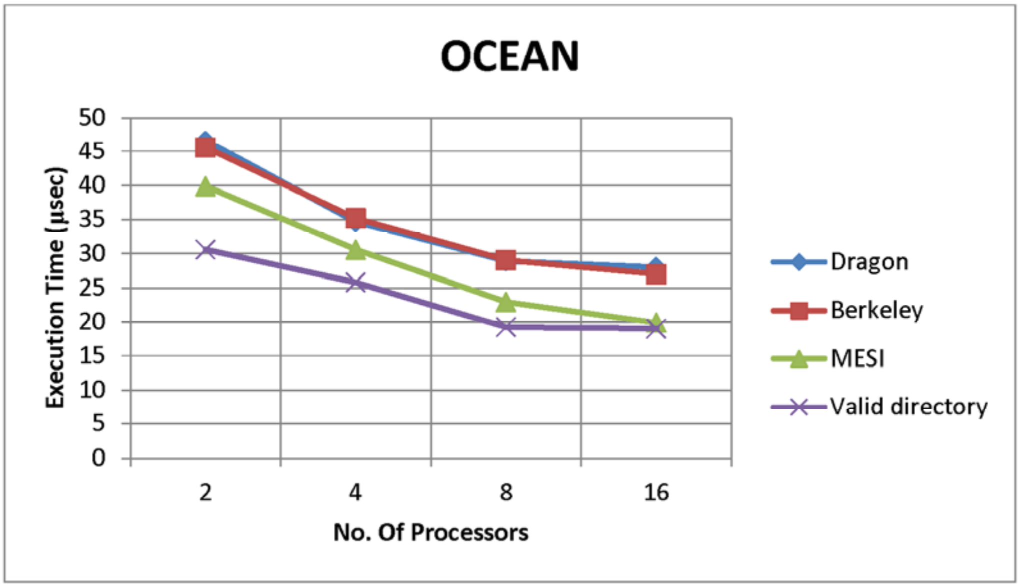


Figure 8.3: Execution cycles in OCEAN with Dragon, Berkeley, MESI and Valid directory cache coherence protocols

Table 8.4: Execution time (µsec) of Dragon, Berkeley, MESI and Valid directory cache coherence protocols using OCEAN benchmark in LIMES simulator

| **Benchmarks** | **No. of processors** | | | |
|---|---|---|---|---|
| | **2** | **4** | **8** | **16** |
| **Dragon** | 46.532987 | 34.659876 | 28.86544 | 27.986543 |
| **Berkeley** | 45.56321 | 35.190432 | 29.03462 | 26.984531 |
| **MESI** | 39.864521 | 30.563211 | 22.89899 | 19.876932 |
| **Valid directory** | 30.562109 | 25.789342 | 19.22212 | 18.989763 |

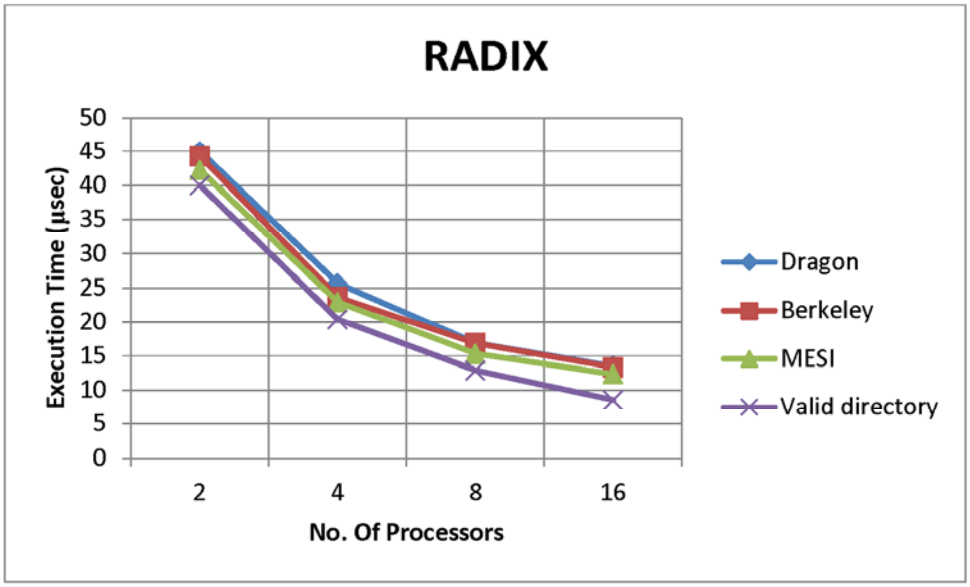


Figure 8.4: Execution cycles in RADIX with Dragon, Berkeley, MESI and Valid directory cache coherence protocols

Table 8.5: Execution time (µsec) of Dragon, Berkeley, MESI and Valid directory cache coherence protocols using RADIX benchmark in LIMES simulator

| **Benchmarks** | **No. of processors** | | | |
|---|---|---|---|---|
| | **2** | **4** | **8** | **16** |
| **Dragon** | 44.976321 | 25.643892 | 16.900972 | 13.5354 |
| **Berkeley** | 44.231768 | 23.632131 | 16.873421 | 13.35538 |
| **MESI** | 42.251621 | 22.897654 | 15.324716 | 12.34568 |
| **Valid directory** | 39.987651 | 20.358765 | 12.879764 | 8.567543 |

## 8.6 Experimental Setup

Simulations were performed with the micro bench programs using seamless CVE simulator.

Table 8.6: Simulation environment in seamless CVE to evaluate ratio of execution time

| | |
|---|---|
| **Simulator** | Seamless CVE |
| **Instruction Cache** | Enabled |
| **Data Cache** | • Private cache enabled.<br>• Shared Cache partly enabled |
| **Memory access time** | • 6 cycles for the 1st word<br>• 1 cycle for each subsequent word |

The micro bench program was implemented with each task by acquiring the lock alternatively. The simulation environment is summarized in table 8.6. If all the caches lines were become invalid in the critical section and after that processor exits the critical section, then it is called as software solution.

Figure 8.5 shows ratio of the execution time on x axis and the number of cached lines on y axis and the corresponding values are tabulated in table 8.7. The proposed solution shows a decrease in execution time of 67.66% than of software solution when execution time is 1. It also proves better than the software solution by 72.3% when the execution time is increases from 1 to 2.

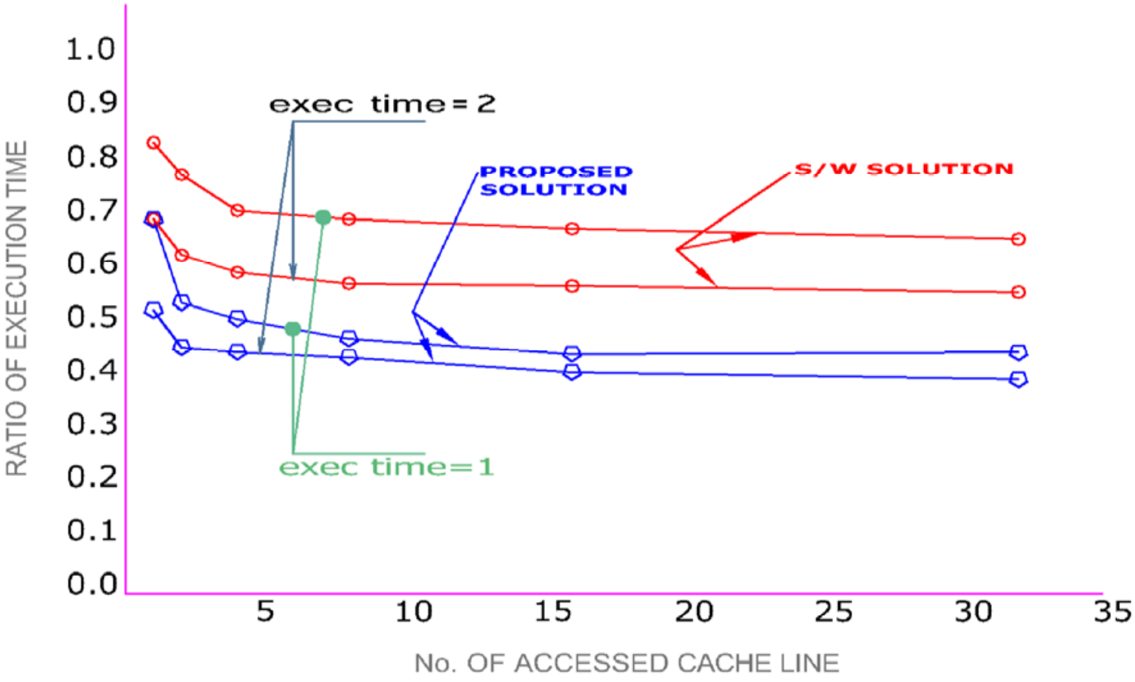


Figure 8.5: Ratio of execution times for proposed valid directory vs s/w based cache coherence solution

Table 8.7: Experimental values for ratio of execution time for directory protocol vs proposed solution

| | | No. of accessed cache lines | ratio of execution time |
|---|---|---|---|
| **Directory Protocol** | **exec time =1** | 1 | 0.84 |
| | | 2 | 0.78 |
| | | 4 | 0.71 |
| | | 8 | 0.7 |
| | | 16 | 0.68 |
| | | 32 | 0.66 |
| | **exec time =2** | 1 | 0.7 |
| | | 2 | 0.63 |
| | | 4 | 0.6 |
| | | 8 | 0.58 |
| | | 16 | 0.581 |
| | | 32 | 0.562 |
| **proposed Solution** | **exec time =1** | 1 | 0.7 |
| | | 2 | 0.54 |
| | | 4 | 0.51 |
| | | 8 | 0.48 |
| | | 16 | 0.46 |
| | | 32 | 0.43 |
| | **exec time =2** | 1 | 0.52 |
| | | 2 | 0.46 |
| | | 4 | 0.45 |
| | | 8 | 0.43 |
| | | 16 | 0.41 |
| | | 32 | 0.4 |

## 8.7 Conclusion

In this chapter, a directory based protocol to solve the cache coherence problem in heterogeneous systems is presented. The simulation results show an average of 22% speedup for execution time in FFT application and 12% speedup for LU in proposed solution at the expense of simple hardware, compared to a pure software solution. In the similar manner, valid directory protocol shows 25% and 16% faster than others while using Ocean and Radix benchmark. Moreover, simulation results showed a increase of 68% in ratio of execution time for execution time 1 and a 72% speedup for execution time 2 in proposed solution as compared to a software solution.

The contribution toward this research is published and is as follows:

1. Shaily Mittal and Nitin, *A Novel Directory Based Solution to Cache Coherence Problem*, in 7th international conference on Advanced Computing & Communication Technologies, ICACCT 2013, APIIT, Panipat, November 16, 2013 sponsored by IETE and Inderscience publishers.

The contribution toward this research is communicated and is as follows:

2. Shaily Mittal and Nitin, A New approach to Directory Based Solution for Cache Coherence Problem, Proceedings of IEEE sponsored 3rd International Conference on Computing of Power ,Energy & Communication, ICCPEIC -2014, Adhiparasakthi Engineering College, Melmaruvathur, kanchipuram dt, April 16 – 17, 2014.

# CHAPTER 9

# FRACTIONAL ASSOCIATIVE MEMORY: A SOLUTION TO CONFLICT MISSES

## 9.1 Introduction and Motivation

Cache memory is located nearest to the processor and used to decrease the average memory access time. The cache is comparatively small and fast in processing than main memory as it keeps only the frequently used data items. When processor generates an address, it first checks in the cache memory.

Presence of requested data in the cache memory results in cache hit while the non existence of data in cache is called cache miss. Cache hit results in the immediate read or write from or to the cache line. And in case of a cache miss, a new entry is allocated in the cache to copy the requested data from main memory so that the request is satisfied from the contents of the cache itself.

Cache miss results in delay in execution as it requires extra time to transfer data from memory to cache itself.

1. Replacement policies: The technique used for deciding the victim entry to expel in case of cache miss is called the replacement policy. The three broadly used replacement policies are LRU (least recently used) [C. Calin, 2003], FIFO (first in first out) [A. Kalavade, 2000] and RAND (random) [J.L. Hennessy, 2003]. LRU is the most preferred cache replacement policy among all. LRU replacement policy is most close to the concept of temporal locality. It is good practice to store a copy of the referenced data in special memory storage for future reference due to property of locality of reference [Y. Yan, 2000], so that it can be accessed faster.
2. Associativity: If the main memory block can be placed to any location in the cache memory, then cache is called fully associative [Q. Zhu, 2012]. While in direct mapped cache each main memory block can be mapped to only one location in the cache memory. If the cache memory is divided into sets and each set contains n blocks and each main memory block can be placed to any one of n blocks in corresponding set of the cache memory, it is called as N-way set associative. Authors have used direct mapped cache memory. Direct-mapped caches implementation have the advantage of

requiring significantly less chip space than fully associative caches because they only require one comparator to determine if a hit has occurred, while in fully associative caches, one comparator for each line in the cache is required.

3. Locality of reference: Generally, instructions and data are stored collectively; this is termed to as locality of reference. Locality of reference assumes that the same data value or the storage location is accessed regularly. There are two different kinds of locality of reference: Temporal locality and spatial locality. Temporal locality says the currently used data will be accessed again in near future with high probability. In contrast, spatial locality says that there is high probability of referencing the value stored nearby to a certain location that is referenced earlier.
4. Types of Cache miss: There are four different types of cache miss [M.D. Hill, 1987]:
    - Compulsory: When the cache block is first accessed; it is moved into the cache.
    - Capacity: When there is not a sufficient space to contain all the required blocks for the program execution in cache memory at the same time, then some blocks are being discarded from cache.
    - Conflict: When more than one blocks are mapped to the same set or block in set associative or direct mapped block placement strategies, this is called conflict miss [C.Zhang, 2006].
    - Coherence: Misses that occur as a result of invalidation to maintain multiprocessor cache consistency.

### 9.2 Related Work

Due to lack of associativity, direct-mapped caches have more conflict misses rather than caches with higher associativity. Inspite of higher conflict miss rate, access time of direct mapped is lower than set-associative caches. [A. Aggarwal, 1988], [J.R. Goodman, 1998], [S. Przybylski, 1990], these authors proposed a very simple method to decrease the cache misses for uni-processors by enlarging the cache line size. Because of the presence of unnecessary data into the cache due to large cache line sizes, the capacity misses increases. Torrellas et al. in [J. Torrellas, 1994] showed that increase in cache lines in multiprocessors do not have any impact on cache misses as proficiently it is in uni-processors due to the property of poor spatial locality in shared data. The proposed prefetching schemes are divided into software-based [F. Dahlgren, 1996], [T. Mowry, 1991], [T. Mowry, 1998] or hardware-based [J.L. Baer, 1991], [E.H. Gornish, 1995], [E. Hagersten, 1992], [D.M. Koppelman, 2000] and [M.K. Tcheun, 1997]. Software approaches introduced instruction overhead and increased memory traffic is due to hardware approaches. Therefore, in [T.F. Chen, 1994] Chen and Baer propose the use of a blend of hardware and software prefetching to utilize the advantages of both. Qureshi and other coauthors in [Q.M. Jaleel, 2007] proposed a DIP replacement policy which selects the appropriate policy to be applied to the cache among LRU and Bimodal Insertion Policy. Also, in [Q.M. Lynch, 2006], Qureshi proposed Memory-Level-Parallelism (MLP) to reduce the miss penalty to the memory by producing the impression of the MLP-aware replacement policy.

## 9.3 Fraction associative memory

The proposed approach is simple variation of direct mapping approach where a fraction of memory is kept reserved for resolving conflict misses [N. Jain, 2012]. The total number of cache blocks are divided into two parts, Blocks in primary cache memory represented by D and Blocks in conflict cache memory represented by C. Address mapping in primary cache memory is used as direct mapped memory using modulo D function as below:-

$$i = M \bmod D$$

Where

i = cache block no.

M = main memory block number

D = number of blocks in primary cache memory

When conflict occurs, primary mapped block i is mapped to conflict memory block j using farmula explained below:-

$$j = (i \text{ modulo } C) + D$$

Where

i = primary mapped cache block

j = primary mapped cache block

C = number of blocks in conflict cache memory

D = number of blocks in primary cache memory

Figure 9.1 shows how conflict miss corresponding to memory block 11 has been mapped to cache block 11. In this example, fraction size is considered as ¼. Hence, 3 cache conflict blocks are taken. One point to be taken care of number of cache blocks (C+D) should be divisible by fraction size f such that

$$C = (C+D) * f$$

$$D = (C+D) * (1- f)$$

Subjected to (C+D) modulo 1/f = 0

Fractional ratio can be adapted as per load conditions statically; however cache memory needs to be relocated as per new cache size available for direct mapping scheme.

A. Handling Conflict miss: In case of conflict miss, it first checks whether conflict cache memory block j mapped to primary cache memory block i is occupied or not by referring to hashing entries for conflicted blocks. If occupied, it checks for primary cache memory block k corresponding to it and identifies victim block as per the Least Recently Used (LRU) block out of two k and j block for replacement. It moves the MRU block in primary cache memory k block and old conflict block j (corresponding to primary cache memory k) is replaced by new conflict block j corresponding to primary cache memory M block. It clears the conflict flag for primary cache memory k block. Moreover, it sets a flag in primary cache memory block i to indicate conflict block mapped to it. A two way hashing entry is maintained for new conflict cache memory block j and memory block M. Thus, it only causes two memory accesses only at time of conflict. It avoids two level memory accesses by maintaining a hashing of all occupied conflict cache blocks.

B. Features of Fraction Associative Memory: Fraction Associative approach takes advantage of irregular memory access pattern and does not cause under-utilization of memory reserved for resolving conflict misses unlike set associative memory. At the same time, it does not cause spatial locality to be hindered due to temporal locality pattern. It is simple in implementation. It conserves space for resolving conflicts by reserving only fraction of cache. It achieves excellent hit access time as it does not require two level memory accesses for checking presence in conflict memory and this approach uses hashing to maintain memory blocks present in conflict memory. It is efficient as it does not use associative cache to store conflict references.

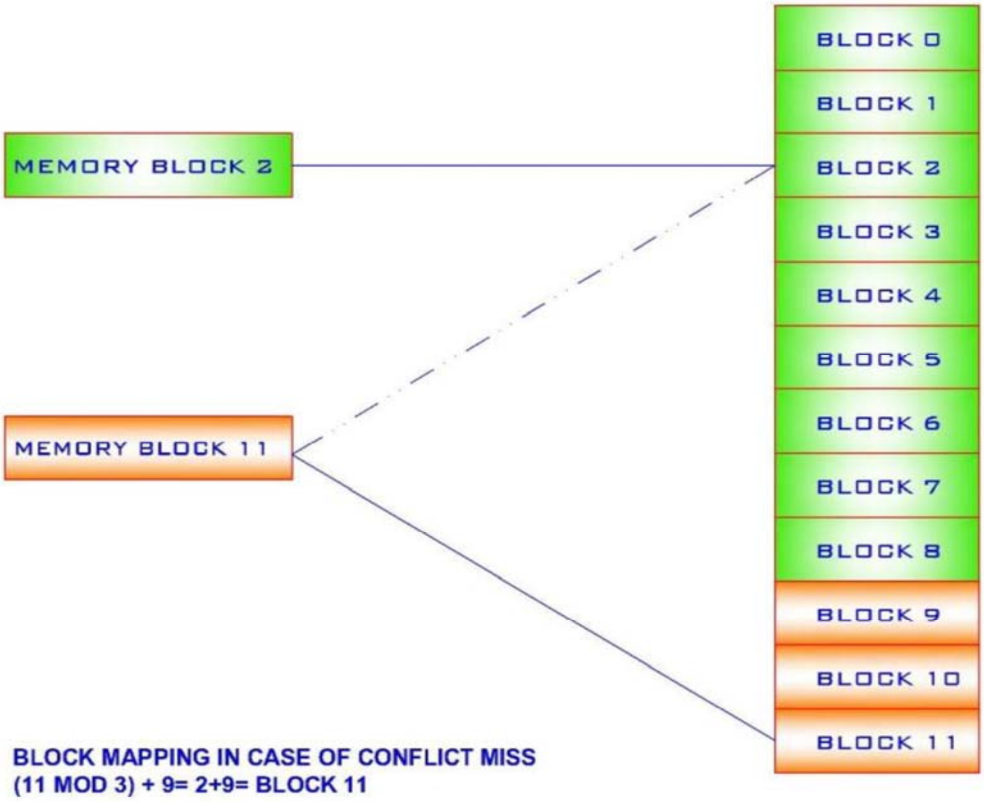


Figure 9.1: Fraction associative mapping explanation [N. Jain, 2012]

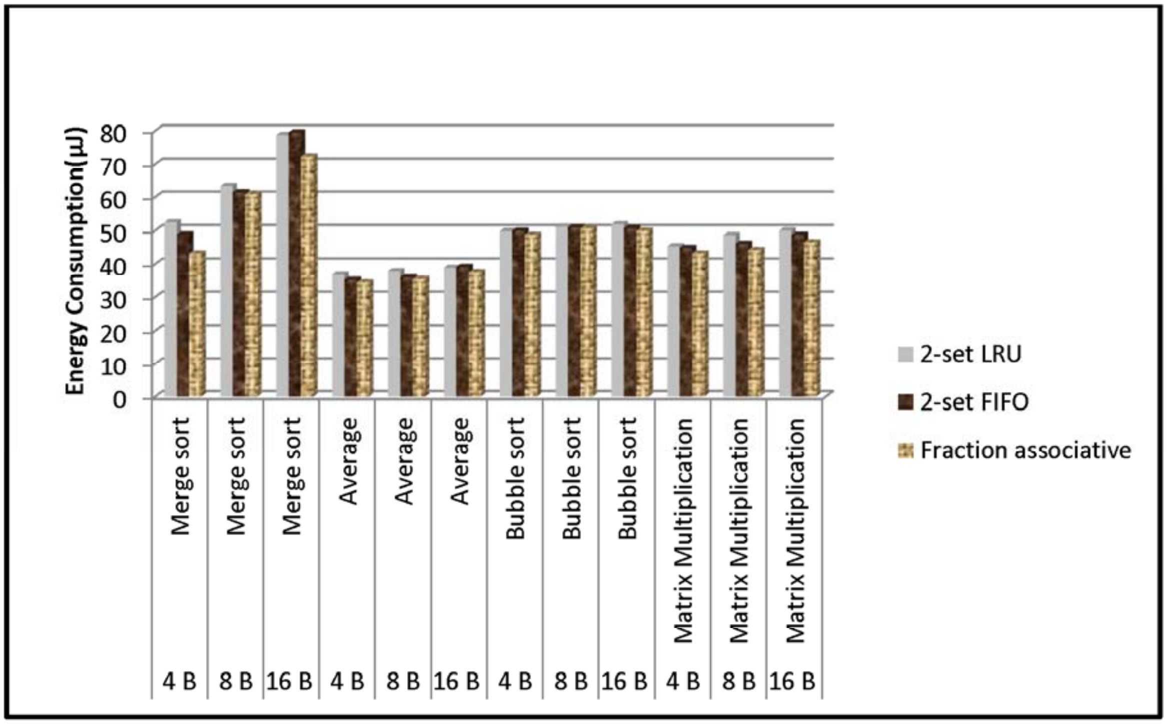


Figure 9.2: Analysis of energy conversion using different benchmarks for, 2-set FIFO, 2-set LRU and fraction associative mapping

Table 9.1: Experimental values for energy consumption using fraction associative mapping

| **Block Size** | **Algorithm** | **2-set LRU** | **2-set FIFO** | **Fraction associative** |
|---|---|---|---|---|
| **4 B** | Merge sort | 52.5 | 48.9 | 43.2 |
| **8 B** | Merge sort | 63.45 | 61.43 | 60.89 |
| **16 B** | Merge sort | 78.75 | 79.54 | 72.31 |
| **4 B** | Average | 36.87 | 35.34 | 34.53 |
| **8 B** | Average | 37.86 | 36.09 | 35.55 |
| **16 B** | Average | 38.89 | 39.12 | 37.53 |
| **4 B** | Bubble sort | 49.87 | 49.98 | 48.76 |
| **8 B** | Bubble sort | 51.23 | 51.01 | 50.72 |
| **16 B** | Bubble sort | 51.93 | 50.74 | 49.98 |
| **4 B** | Matrix Multiplication | 45.32 | 44.87 | 43.21 |
| **8 B** | Matrix Multiplication | 48.67 | 46.09 | 44.23 |
| **16 B** | Matrix Multiplication | 50.04 | 48.75 | 46.53 |

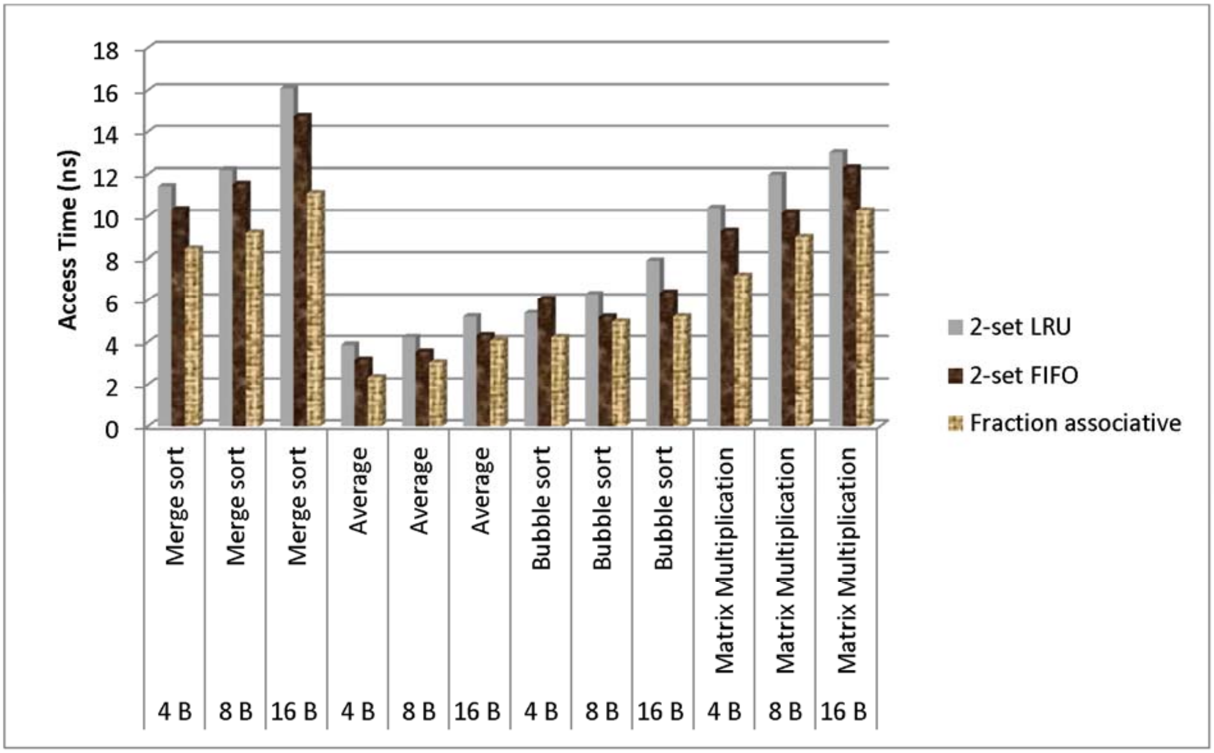


Figure 9.3: Analysis of access time using different benchmarks for, 2-set FIFO, 2-set LRU and fraction associative mapping

Table 9.2: Experimental values for access time using fraction associative mapping

| **Block Size** | **Algorithm** | **2-set LRU** | **2-set FIFO** | **Fraction associative** |
|---|---|---|---|---|
| **4 B** | Merge sort | 11.43 | 10.32 | 8.45 |
| **8 B** | Merge sort | 12.2 | 11.54 | 9.21 |
| **16 B** | Merge sort | 16.09 | 14.76 | 11.1 |
| **4 B** | Average | 3.87 | 3.18 | 2.34 |
| **8 B** | Average | 4.23 | 3.56 | 3.05 |
| **16 B** | Average | 5.22 | 4.32 | 4.06 |
| **4 B** | Bubble sort | 5.38 | 6.05 | 4.23 |
| **8 B** | Bubble sort | 6.29 | 5.21 | 4.97 |
| **16 B** | Bubble sort | 7.9 | 6.37 | 5.22 |
| **4 B** | Matrix Multiplication | 10.39 | 9.29 | 7.18 |
| **8 B** | Matrix Multiplication | 11.98 | 10.17 | 9 |
| **16 B** | Matrix Multiplication | 13.02 | 12.3 | 10.26 |

### 9.4 Test bed and Experimental Setup

This mapping scheme is implemented on shared memory and multiprocessor based architecture. Different processors may execute independently having mutually exclusive access to shared memory at a time. Array based computation is used to select algorithms for studying data locality. Cache size is taken as 64 bytes and utilizes block size of 4 bytes, 8 bytes and 16 bytes. For the purpose of implementation and simulation of whole architecture, CACTI 2.0 [P. Shivakumar, 2001/2] timing and power model is used. CACTI 2.0 model can evaluate three parameters: cache access time, cycle time, and power of the system. Cache access time and power consumption are the two parameters on the basis of which different replacement policies are compared. Different benchmarks used are: merge sort, average, bubble sort and matrix multiplication. Such benchmarks are chosen as they require parallel and long calculations temporally. Tables 9.1 and 9.2 show the experimental values.

### 9.5 Results and Discussions

From fig. 9.2, in case of energy consumption, the bar graph clearly shows the lowest energy consumed (μJ) in case of fraction associative mapping than 2-set LRU and FIFO policies. Merge sort shows highest energy consumption due to complexity. Because of simplicity in average calculation, it shows best results. Figure 9.3 illustrates the cache access time in ns for three different replacement policies namely 2-set LRU,2-set FIFO and fraction associative mapping using block sizes 4,8 and 16 B on four different benchmarks. Again, proposed policy outperforms LRU and FIFO with lowest cache access time.

### 9.6 Conclusion

Fraction associative cache conserves space and cost because only fraction of cache is reserved for resolving conflict misses. Moreover, run or compile time data transformation techniques used to exploit data locality involves additional runtime overhead, whereas fraction associative scheme does not include runtime overhead and does not involve any data transformations. Increased block size, increased energy consumption and cache access time; however fraction associative cache provides midway to achieve both factors in optimal way. Fraction associative mapping takes benefits of direct mapping of being cost optimal, simple and having less access time and power consumption.

In essence, fraction associative mapping is simple, elegant, cost effective and low-overhead high performing scheme which outperforms set associative mapping.

The contribution towards this research is communicated and is as follows:

Shaily Mittal and Nitin, *Fractional Associative Memory: A Solution to Conflict Misses*, Proceedings of 4th IEEE International Advance Computing Conference (IACC) February 21-22, 2014. IEEE sponsored 3rd  International Conference on Computing of Power ,Energy & Communication, ICCPEIC -2014, Adhiparasakthi Engineering College, Melmaruvathur, kanchipuram dt, April 16 – 17, 2014.

# CHAPTER 11

# CONCLUSION AND FUTURE DISCUSSIONS

This chapter sums up the entire study and list out further scope of works along with the open problems concern with multiple processor synchronization and shared memory conflicts.

## 11.1 Conclusion

The cost of processor synchronization when using locks is totally determined by the number of locks acquired in the application. The cost of synchronization for transactions depends on the conflict rate. The overall waiting time for a processor determines the performance of the system when using semaphores. Hence, semaphores outperforms with lowest cache miss rate and best performance. And in case of energy consumption, semaphore is almost comparable to transaction however much better than locks because the owner for calls to lock and unlock is same thread, hence consuming more energy in switching. However, Signal and wait calls can be made by different threads.

A simulation based evaluation of cache with scratch pad memory in terms of main memory design issues like replacement policies and associativity in embedded systems is presented. Moreover, results maintain the status of LRU as the best replacement policy on FIFO and Random in SPM as well as in cache memory. With the increase in associativity, cache miss rate decreases as data can be easily get accessed in higher associative caches. Moreover, cache miss rate decreases when using SPM as it is smaller in size as compared to cache memory and hence fast availability of data.

Bubble sort benefits more from the property of locality of reference and hence performs better than merge sort. Moreover, LRU comes out as the best replacement policy in merge sort with interleaved memory, while FIFO performs best in the case of finding average of 30 elements.  The reason of lacking LRU may be the overhead of maintaining the record of recently used pages which is not functional in average calculation. All simulations are performed on Simple Scalar as well as Memory map simulator and similar results prove authenticity of the simulator. This simulator was designed keeping in mind the architectures where more than one processor is not accessing the shared memory concurrently.

A new, efficient, high performance cache replacement policy called tag based dual mapping replacement policy (TDMRP) is proposed that raises hit ratio and diminishes the power consumption when compared to two well-known replacement policies LRU and FIFO. A simulation based evaluation of cache with and without scratch pad memory in terms of main memory design issues like replacement policies and associativity in embedded systems is presented. The simulation results demonstrate the reduction in cache miss rate up to 20% by the use of newly proposed dual replacement policy. Hit ratio increases as this policy is taking the advantages of LRU and use of status bits. Energy consumption is less as memory access is less as compared to the memory accesses in FIFO and LRU due to less frequent changes in data in memory and hence lesser main memory accesses.

Fraction associative cache conserves space and cost because only fraction of cache is reserved for resolving conflict misses. Fraction associative scheme does not include runtime overhead and does not involve any data transformations. Fraction associative mapping takes benefits of direct mapping of being cost optimal, simple and having less access time and power consumption. In essence, Fraction associative mapping is simple, elegant, cost effective and low-overhead high performing scheme which outperforms set associative mapping.

To address the coherence of data caches in heterogeneous processor platforms, a methodology is presented. Berkeley protocol has low execution time because exclusive ownership is required before the line can be modified. In Dragon protocol, exclusive ownership of the line is not required to modify the line however the protocol does require the special sharing line. In MESI, when a block is in the valid state and a store is issued to that block, a Bus Write, instead of a Bus Invalidate occurs, and a transfer from memory is initiated even though it is unnecessary. This difference will definitely hurt performance since the writing processor will have to wait for the entire cache block to be transferred before it can proceed. In addition, unnecessary bus traffic is created. All three protocols have high complexity due to the use of different states and state transitions. Valid directory based protocols have low complexity and hence better execution time.

Due to regular updation in routing tables based on changing topologies, dynamic routing algorithm have better performance than static routing algorithms. Throughput decreases and packet delay increases with the increase in traffic load. Due to the increase in traffic in the network, packet loss increases and hence throughput decreases and congestion leads to increased packet delay. Among four different network topologies, mesh topology shows best results with highest throughput and EBFT with lowest throughput as mesh is the simplest topology and EBFT is the complex among all. While EBFT shows best results with lowest latency because routes can easily be determined in this topology and BFT with highest latency. With the increase in network size, throughput declines and latency increases.

## 11.2 Future Work

Future work will consider some more issues in multiprocessor systems like designing an efficient routing algorithm for NoC, improved NoC topology.

In this dissertation, TDMRP cache replacement policy is explored in terms of hit ratio and energy only as these two is on the priority for embedded systems. In future, it can be evaluated and compared with some other replacement policies on the basis of some other parameters.

Future work will improve results obtained for scratch pad memory by considering dynamic moving memory data in and out of the scratchpad with evaluation of more parameters for comparison. In addition, research can be done for the extension of this approach to multiprocessor systems on chip (MPSoC).

As heterogeneous processor SoC's become more ubiquitous in future system design, directory based protocol for solving cache coherence will be very useful and effective for integrating heterogeneous coherence protocols in the same system processors.

# REFERENCES

[ K Srinivasan, 2005] K Srinivasan, K S Chatha, G Konjevod. An automated technique for topology and route generation of application specific on-chip interconnection networks. Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD ’05), 2005.

[A. Agarwal, 1988] A. Agarwal, J. Hennessy and M. Horowitz. Cache Performance of Operating System and Multiprogramming Workloads. ACM Transactions on Computer Systems (TOCS),6(4):393–431, 1988.

[A. Agarwal, 1993] A. Agarwal and S.D. Pudar. Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct-Mapped Caches. Proceeding of the 20th International Symposium on Computer Architecture, pp. 179-180, May 1993.

[A. Jantsch, 2003] A. Jantsch and H. Tenhunen. Networks on Chip. Kluwer Academic Publishers, 2003.

[A. Kalavade, 2000] A. Kalavade, J. Knoblock, E. Micca, M. Moturi, J.H. O’Neill, J. Othmer, E.Sackinger, K.J. Singh, J. Sweet, C.J. Terman, and J. Williams. A Single-Chip, 1.6 Billion, 16-b MAC/s Multiprocessor DSP, IEEE Journal of Solid-state circuits, 35(3), pp. 412-423, 2000.

[Abdulla shaik, 2012] Abdulla shaik. Dynamic Sort (DS) Page Replacement. The International Journal of Computer Science & Applications (TIJCSA), Volume 1, No. 3, May 2012.

[Ahmed Amine Jerraya, 2005] Ahmed Amine Jerraya, Wayne Wolf, Morgan Kaufmann. Multiprocessor systems-on-chip. Elsevier 2005.

[Ahmed Samih, 2011] Ahmed Samih, Yan Solihin and Anil Krishna. Evaluating placement policies for managing capacity sharing in CMP architectures with private caches. In ACM Transactions on Architecture and Code optimization volume 8 Issue 3, October 2011.

[Alokika Dash, 2006] Alokika Dash. Masters of Science thesis submitted to the Faculty of the Graduate School of the University of Maryland, College Park, 2006.

[AMD, 2009] AMD64 Technology [Online document] available http://www.amd.com/usen/assets/content_type/white_papers_and_tech_docs/24593.pdf, 2009.

[Anant Agarwal, 1988] Anant Agarwal, Richard Simoni, John Hennessy, and Mark Horowitz. An Evaluation of Directory Schemes for Cache Coherence. Proceedings of the 15th annual international symposium on computer architecture (ISCA), Honolulu, Hawaii, pp 280-289, June 1988.

[Angel Dominguez, 2005] Angel Dominguez, Sumesh Udayakumaran and Rajeev Barua. Heap Data Allocation to Scratch-Pad Memory in Embedded Systems. Journal of Embedded Computing, Volume 1, Issue 4, 2005.

[Bahman Hashemi, 2011] Bahman Hashemi. Simulation and Evaluation Snoopy cache coherence protocols with Update strategy in shared memory multiprocessor systems. In Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications Workshops, 2011.

[Behnam Golvardzadeh, 2012] Behnam Golvardzadeh and Pouya Derakhshan-Barjoei. A Novel Semi-Adaptive Routing Algorithm for Delay Reduction in Networks on Chip. Research Journal of Applied Sciences, Engineering and Technology 4(19), October 01, 2012.

[C. Ferri, 1993] C.Ferri. Lock-Free Data Structures. In International Symposium on Computer Architecture, 1993.

[C. Ferri, 2007] C. Ferri, T. Moreshet, R.I. Bahar, L. Benini and M. Herlihy. A Hardware/Software Framework for supporting Transactional Memory in a MPSoC Environment. ACM SIGARCH Computer Architecture News, 35(1), 2007.

[C. Ferri, 2008] C. Ferri, R.I. Bahar, T. Moreshet, A. Viescas and M. Herlihy. Energy Efficient Synchronization Techniques for Embedded Architectures. GLSVLSI'08, USA, 2008.

[C. K. Tang, 1976] C. K. Tang. Cache Design in the Tightly Coupled Multiprocessor System. In AFIPS Conference Proceedings, National Computer Conference, NY, pages 749-753, June 1976.

[C. Zhang, 2006] Balanced Cache: Reducing Conflict Misses of Direct-Mapped Caches through Programmable Decoders. Proceedings of the 33rd IEEE International Symposium on Computer Architecture (ISCA'06), 2006.

[C.L. Chen, 1989] C.L. Chen and C.K. Liao. Analysis of Vector Access Performance on Skewed Interleaved Memory. Proceedings of the 16th Annual International Symposium on Computer Architecture, pp. 387-394, 198.

[Cacti, 2009] http://www.hpl.hp.com/research/cacti/.

[Calin CaBcaval, 2003] Calin CaBcaval, David A. Padua. Estimating cache misses and locality using stack distances. Proceedings of the 17th annual international conference on Supercomputing, 2003.

[Charles P. Thacker, 1987] Charles P. Thacker and Lawrence C. Stewart. Firefly: a Multiprocessor Workstation. Proceedings of ASPLOS 11, pages 164-1'72, October 1987.

[D. Burger, 1997] D. Burger and T. Austin. The SimpleScalar tool set, CS Dept., Univ. Wisconsin, Madison, Rep. 1342, version 2.0.7, 1997.

[D. Cho, 2009] D. Cho, S. Pasricha, I. Issenin, N.D. Dutt, M. Ahn and Y. Paek. Adaptive Scratch Pad Memory Management for Dynamic Behavior of Multimedia Applications. IEEE Transactions on Computer Aided design of Integrated Circuits and Systems 28(4), 2009.

[D. M. Koppelman, 2000] D. M. Koppelman. Neighbourhood Prefetching on Multiprocessors Using Instruction History. In Proceedings of International Conference on Parallel Architectures and Compilation Techniques, pages 123–132, 2000.

[D.H. Albonesi, 1999] D.H. Albonesi. Selective cache ways: On-demand cache resource allocation. In 32nd International symposium on Microarchitecture, 1999.

[David Patterson, 2009] David Patterson, john Hennessy, .Computer Organization and Design (4th edition).Morgan Kaufmann, 2009.

[David Tarnoff, 2006] David Tarnoff. Computer organization and Design Fundamentals. Publisher: Lulu.com, pages 434, 13 January 2006.

[E. Dijkstra, 1959] E. Dijkstra. A note two problems in connection with graphs. Numerical Math, vol. 1, pp. 269-271, 1959.

[E. H. Gornish, 1995] E. H. Gornish. Adaptive and Integrated Data Cache Prefetching for Shared-Memory Multiprocessors. PhD thesis, University of Illinois at Urbana-Champaign, 1995.

[E. Hagersten, 1992] E. Hagersten. Toward Scalable Cache-Only Memory Architectures. PhD thesis, Royal Institute of Technology, Stockholm, 1992.

[E. McCreight, 1984] E. McCreight .The Dragon Computer System: An Early Overview. Tech. report, Xerox Corp, September 1984.

[E.P. Markatos, 1994] E.P. Markatos and T.J. LeBlanc. Using Processor Affinity in Loop Scheduling Scheme on Shared-Memory Multiprocessors. IEEE Trans. Parallel and Distributed Systems, 5(4), pp. 379-400, 1994.

[E.W. Dijkstra, 1971] E.W. Dijkstra, Hierarchical ordering of sequential processes, Acta Informatica, 1, 115-138, 1971.

[F. Baslett, 1988] F. Baslett, T. Jermoluk, and D. Solomon. The 4D-MP Graphics Superworkstataion: Computing+Graphics= 40MIPS+40MFLOPS and 100,000 Lighted Polygons per Second. Proceeding of 33rd IEEE Computer Society Int'l Conference – COMPCON`88, pp 468-471, February 1988.

[F. Dahlgren, 1996] F. Dahlgren and P. Stenstrom. Evaluation of HardwareBased Stride and Sequential Prefetching in Shared-Memory Multiprocessors. IEEE Transactions on Parallel and Distributed Systems, 7(4):385–398, 1996.

[Frank S. J, 1984] Frank S. J. Tightly Coupled Multiprocessor System Speeds Up Memory Access Times. Electronics, 57, 1, January 1984.

[G. S. Cho, 2006] G. S. Cho and J. K. Ryeu. An Efficient Method to Find a Shortest Path for a Car-Like Robot. International Journal of Multimedia and Ubiquitous Engineering, vol. 1, no. 1, pp. 1-6. 2006.

[H. Akkary, 2003] H. Akkary, R. Rajwar and S.T. Srinivasan. Checkpoint processing and recovery: Towards scalable large instruction window processors. In the 36th Intl. Symposium on Microarchitecture, 2003.

[H. Hossain, 2007] H. Hossain, M. Ahmed, A. Al-Nayeem, T.Z. Islam, M. Akbar. Gpnocsim - A General Purpose Simulator for Network-On-Chip. In proceeding of: Information and Communication Technology, ICICT '07 Univ. of Rochester, 2007.

[I. Issenin, 2006] I. Issenin, E. Brockmeyer, B. Durinck and N.D. Dutt. Multiprocessor system-on-chip data reuse analysis for exploring customized memory hierarchies. In Proceedings Des. Autom. Conference, pp 49-52, 2006.

[I. Issenin, 2008] I. Issenin, E. Brockmeyer, B. Durinck and N.D. Dutt. Data- Reuse-Driven Energy-Aware Cosynthesis of Scratch Pad memory and Hierarchical Bus-Based Communication Architecture for Multiprocessor Streaming Applications. IEEE Transactions on Computer Aided design of Integrated Circuits and Systems 27(8), 2008.

[Intel, 2002] Intel Pentium 4 and Intel Xeon Processor Optimization Reference Manual, http://developer.intel.com, 2002.

[ITR, 2011] International Tecnology Roadmap for Semiconductors. ITRS update 2010. Accessed in: http://www.itrs.net/reports.html. November 2011.

[J. Henkel, 2004] J. Henkel, W. Wolf, S. Chakradhar. On-chip networks: A scalable, communication-centric embedded system design paradigm. VLSID '04: Proceedings of the 17th International Conference on VLSI Design, IEEE Computer Society, Washington, DC, USA, p. 845, 2004.

[J. Hu, 2004] J. Hu , R. Marculescu. Application-Specific Buffer Space Allocation for NoCs Router Design. IEEE/ACM International Conference on Computer Aided Design, ICCAD-2004 , pp. 354 – 361, Nov. 2004.

[J. Reineke, 2006] J., Reineke, D., Grund, C., Berg, R., Wilhelm. Predictability of Cache Replacement Policies. AVACS Technical Report No. 9, SFB/TR 14 AVACS, 2006.

[J. Torrellas, 1994] J. Torrellas , M. S. Lam, and J. L. Hennessy. False Sharing and Spatial Locality in Multiprocessor Caches. IEEE Transactions on Computers, 43(6):651–663, 1994.

[J.L. Baer, 1991] J.-L. Baer and T.-F. Chen. An Effective On-Chip Preloading Scheme to Reduce Data Access Penalty. In Proceedings of the 1991 Conference on Supercomputing, pages 176–186, 1991.

[J.L.Hennessy, 2003] J.L., Hennessy, D., Patterson. Computer Architecture: A Quantitative Approach. Third Edition, Morgan Kaufmann Publishers, 2003.

[J.W. Chung, 2006] J.W. Chung, H. Chafi, C.C. Minh, A. McDonald, B.D. Carlstrom, C. Kozyrakis and K. Olukotun. The common case transactional behavior of multithreaded programs. In Proceedings of the 12th International Symposium on High- Performance Computer Architecture, 2006.

[James Archibald, 1985] James Archibald and Jean-Loup Baer. An Economical Solution to the Cache Coherence Problem. Proceedings of the 12th International Symposium on Computer Architecture, pages 355-362, June 1985.

[James Archibald, 1986] James Archibald and Jean-Lou Baer. Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model. ACM Transactions on Computer Systems, 4(4):273-298, November 1986.

[James R. Goodman, 1983]. James R. Goodman .Using Cache Memory to Reduce Processor-Memory Traffic. In Proceedings of the 10th Annual Symposium on Computer Architecture, pages 124-131, June 1983.

[Jan Reineke, 2006] Jan Reineke , Daniel Grund Christoph Berg, Reinhard Wilhelm. Predictability of Cache Replacement Policies. AVACS – Automatic Verification and Analysis of Complex Systems, Technical Report No. 9, September 2006.

[Jason Loew, 2011] Jason Loew, Jared Schmitz, Jesse Elwell, Dmitry Ponomarev and Nael Abu-Ghazaleh. TPM-SIM: A Framework for Performance Evaluation of Trusted Platform Modules. DAC 2011, San Diego, California, USA, June 5-10, 2011.

[Jei Chen, 2011] Jie Chen, Cheng Li and Paul Gillard. Network-on-Chip (NoC) Topologies and Performance: A Review. IEEE Newfoundland and Labrador Section, http://necec.engr.mun.ca/ocs2011, 18 Oct, 2011.

[Jie Yu, 2009] Jie Yu and Satish Narayanasamy. A Case for an Interleaving Constrained Shared-Memory Multi-Processor. ISCA'09, ACM, Austin, Texas, USA, June 20–24, 2009 .

[Juan Gomez, 2009] Juan Gomez ,Luna Herruzo and Jose Ignacio Benavides. MESI Cache Coherence Simulator for Teaching Purposes. CLEI ELECTRONIC journal pp 1-7, 2009.

[Jude Rivers, 1996] Jude Rivers, Edward S. Davidson. Reducing Conflicts In Direct-Mapped Caches with A Temporality-Based Design. Proceedings of the 1996 IEEE ICPP , 1996.

[K. Hwang, 1984] K. Hwang and F.A. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill, 1984.

[K. Hwang, 1993] K.Hwang. Advanced Computer Architecture. McGraw-Hill, 1993.

[K.S. McKinley, 1996] K.S. McKinley, S. Carr, and C.W. Tseng. Improving Data Locality with Loop Transformations. ACM Transactions Programming Languages and Systems, 18(4), pp. 424-453, 1996.

[L. Benini, 2002] L. Benini and G. De Michelli. Networks on chips: a new SoC paradigm. IEEE Computer, Vol. 35, 1, pp. 70-78, January 2002.

[L. Benini, 2005] L. Benini ,D. Bertozzi. Network-on-chip architectures and design methods. IEEE Proceedings - Computers and Digital Techniques, Vol. 152, 2, pp. 261-272, March 2005.

[L. Hammond, 2012 ] L. Hammond, V. Wong, M. Chen, B.D. Carlstorm, J.D. Davis, B. Hertzberg, M.K. Prabhu, H. Wijaya, C. Kozyrakis and K. Olukotun, Transactional memory vol. 23, no. 7, pp. 1312-1325, July 2012.

[L. Rudolph, 1985] L. Rudolph and Z. Segall. Dynamic Decentralized Cache Consistency Schemes for MIMD Parallel Processors. In Proceedings of the 12th International Symposium on Computer Architecture, pages 340-347, June 1985.

[L. Zhang, 2007] L. Zhang , H. Chen, B. Yao, K. Hamilton, and C. K. Cheng. Repeated on-chip interconnect analysis and evaluation of delay, power, and bandwidth metrics under different design goals. Proceedings of the 8th International Symposium on Quality Electronic Design (ISQED '07), pp. 251–256, March 2007.

[Leonid Ryzhyk, 2006] Leonid Ryzhyk. http://www.scribd.com/doc/122524637/shared-memory. April 21, 2006.

[Lucien M. Censier, 1978] Lucien M. Censier and Paul Feautrier. A New Solution to Coherence Problems in Multicache Systems. IEEE Transactions on Computers, pp 1112-1118, December 1978.

[M. A. J. Jamali, 2009] M. A. J. Jamali and A. Khademzadeh. MinRoot and CMesh: interconnection architectures for network-on-chip systems. World Academy of Science, Engineering and Technology, vol. 54, pp. 354–359, 2009.

[M. Farrens, 1991] M. Farrens and A. Park. Dynamic Base Register Caching: A Technique for Reducing Address Bus Width. Proceedings of the 18th International Symposium on Computer Architecture, pp.128-137, 1991.

[M. Garzaran, 2001] M. Garzaran J. Briz, P. Ibanez, and V. Vinals. Hardware Prefetching in Bus-Based Multiprocessors: Pattern Characterization and Cost-Effective Hardware. In Proceedings of Parallel and Distributed Processing 2001, pages 345–354, 2001.

[M. Herlihy, 2003] M. Herlihy, V. Luchangco, M. Moir and W. Scherer. Software transactional memory for dynamic-sized data structures. In Symposium on Principles of Distributed Computing, 2003.

[M. Huang, 2000] M. Huang, J. Renau, S.M. Yoo and J. Torrellas. A framework for dynamic energy efficiency and temperature management. In the 33rd International Symposium on Microarchitecture, 2000.

[M. K. Tcheun, 1997] M. K. Tcheun H. Yoon, and S. R. Maeng. An Effective OnChip Preloading Scheme to Reduce Data Access Penalty. In Proceedings of the International Conference on Parallel Processing, pages 306–313, 1997.

[M. Kandemir, 2005] Kandemir M, Dutt N. Memory Systems and Compiler Support for MPSoC Architectures. In Multiprocessor Systems-on-Chips, Kluwer Academic Publishers, pp. 251–281, 2005.

[M. Kistler, 2006] M. Perrone, F. Petrini. Cell multiprocessor communication network: Built for speed. IEEE Micro 26 (3) pp 10–23. doi:http://dx.doi.org/10.1109, 2006.

[M. Loghi, 2006] M. Loghi, M.P. and L. Benini. Cache Coherence Tradeoffs in Shared-Memory MpSoCs. ACM Transactions on Embedded Computing Systems, 5(2), pp. 383-407, 2006.

[M.Á.V.Rodríguez, 2001] M.Á.V. Rodríguez, M.S. Pérez and J.A.G. Pulido. An Educational Tool for Testing Caches on Symmetric Multiprocessors. Microprocessors and Microsystems, 24(5), pp. 187-194, 2001.

[M.B. Kamble, 1997] M.B. Kamble and K. Ghose, Analytical Energy Dissipation Models for Low-Power Caches, Proceedings of the International Symposium on Low-Power Electronic Devices, pp. 143-148, 1997.

[M.Morros Mano, 1982 ] M. Morros Mano. Computer system architecture. Prentice Hall of India, 1982.

[Maksat atagoziyev, 2007] Maksat Atagoziyev. Routing algorithms for on chip networks. Master of science in Electrical and electronics engineering Thesis submitted to the graduate school of natural and applied sciences of Middle east technical university. December, 2007.

[Maksat Atagoziyev, 2009] Maksat Atagoziyev. Networks on Chip: Topology, Switching, Routing. Publisher: VDM Verlag, pp 92, 8 May 2009.

[Mark D. Hill, 1987] Mark D. Hill. Aspects of Cache Memory and Instruction Buffer Performance. Ph.D. Th., University of Califomia, Berkeley, 1987.

[Mark D. Hill, 1988] Mark D. Hill. A Case for Direct-Mapped Caches. IEEE Computer, 21(12), pp. 25-40, 1988.

[Mark S. Papamarcos, 1985] Mark S. Papamarcos and Janak H. patel. A Low- Overhead Coherence Solution for Multiprocessors with Private Cache Memories. In Proceedings of the 12th International Symposium on Computer Architecture, pages 348-354, June 1985.

[Masoumeh Ebrahimi, 2009] Masoumeh Ebrahimi , Masoud Daneshtalab, Mohammad Hossein Neishaburi, Siamak Mohammadi. An Efficent Dynamic Multicast Routing Protocol for Distributing Traffic in NOC. In DATE 09, EDAA, 2009.

[Miles J. Murdocca, 2007] Miles J. Murdocca and Vincent Heuring. Computer Architecture and Organization: An integrated approach. Published by Wiley, 1st edition, pp 544, March, 2007.

[Mohapatra, P, 1998] Mohapatra, P . Wormhole routing techniques for directly connected multicomputer systems. ACM Computing Surveys, 30(3), pp. 374-410, Sep. 1998.

[Mohsen Soryani, 2007] Mohsen Sharifi Mohammad Hossein Rezvani. Performance Evaluation of Cache Memory Organizations in Embedded systems. International Conference on Information Technology(ITNG'07) Las Vegas, Nevada, USA, pp. 1045-1050, April 2007.

[Muhammad Ali, 2005] Muhammad Ali, Michael Welzl, Sybille Hellebrand. A Dynamic Routing Mechanism for Network on chip. In IEEE NORCHIP conference 2005.

[Nitika Jain, 2012] Nitika Jain , Shaily Mittal and Prachi. Reducing conflict misses using Fraction associative mapping. 2nd IEEE International Conference on Parallel, Distributed and Grid Computing 978-1-4673-2925-5/12, 2012.

[Nomadik, 2009] Nomadik platform. www.st.com, 2009.

[Norman Matloff, 2003] Norman Matloff. Memory Interleaving. Department of Computer Science University of California at Davis November 5, 2003

[Norman P. Jouppi, 1990] Norman P. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. Proceedings of the 17th Annual Symposium on Computer Architecture, pp. 364-373, 1990.

[OMAP, 2010] OMAP5910 platform. www.ti.com, 2010.

[OSCI, 2004] Open SystemC Iniative OSCI, SystemC documentation, http://www.systemc.org, 2004.

[P. C. Newton, 2009] P. C. Newton , L. Arockiam and T. H. Kim. An Efficient Hybrid Path Selection Algorithm for an Integrated Network Environment. International Journal of Database Theory and Application, vol. 2, no. 1, pp. 31-38, 2009.

[P. Ezhumalai, 2009] P. Ezhumalai ,S. Aravind, and D. Sridharan. A survey of architectural design and analysis of network on chip systems. Proceedings of the International conference on Signals, Systems and Communication, pp. 87–91, college of Engineering Guindy campus, Anna University Chennai, December 2009.

[P. Ezhumalai, 2011] P. Ezhumalai, A. Chilambuchelvan and C. Arun. Novel NoC Topology Construction for High-Performance Communications. Hindawi Publishing Corporation Journal of Computer Networks and Communications Volume 2011, Article ID 405697, 6 pages, 2011.

[P. P. Pandeet, 2005] P. P. Pandeet. Performance evaluation and design trade-offs for network-on chip interconnect architectures. IEEE Transactions on Computers, Vol. 54, 8, pp. 1025-1040, August 2005.

[PC, 2005] PC205 platform. www.picochip.com, 2005.

[Philip J. Koopman, 1989] Philip J. Koopman. Stack Computers: the new wave. Ellis Horwood 1989.

[Prashant Shenoy, 2008] Prashant Shenoy. Lass.cs.umass.edu, September 2008 http://lass.cs.umass.edu/~shenoy/courses/fall10/lectures/Lec16notes.pdf, 2008.

[Preeti Ranjan Panda, 1997] Preeti Ranjan Panda, Nikil D. Dutt, and Alexandru Nicolau. Efficient Utilization of Scratch-Pad Memory in Embedded Processor Applications. European Design Automation and Test Conference, 1997.

[Qi Zhu, 2012] Qi Zhu 1 and Ying Qiao. A Survey on Computer System Memory Management and Optimization Techniques. American Journal of Computer Architecture, volume 1 issue 3, pp 37-50, 2012.

[Qureshi M, 2006] Qureshi M., Lynch D., Mutlu O. & Patt Y. A Case for MLP-Aware Cache Replacement. Proceedings of the 33th annual international symposium on Computer architecture, pp. 167-178, 2006.

[Qureshi M, 2007] Qureshi M., Jaleel A., Patt Y., Jr. S. & Emer J. Adaptive Insertion Policies for High Performance Caching. Proceedings of the 34th annual international symposium on Computer architecture, pp. 381-391, 2007.

[R. Banakar, 2002] R. Banakar. Scratch-pad Memory: A Design Alternative for Cache On-chip Memory in Embedded Systems. Proceedings of the 10th International Workshop on Hardware/Software Codesign, pp.73-78, 2002.

[R. Fromm, 1997] R. Fromm, S. Perissakis, N. Cardwell, C. Kozyrakis, B. McGaughy, D. Patterson, T. Anderson and K. Yelick. The energy efficiency of IRAM architectures. 24th Intl. Symposium on Computer Architecture, 1997.

[R. H. Katz, 1985] R. H. Katz, S. J. Eggers, D. A. Wood, C. L. Perkins, and R. G. Sheldon. Implementing a Cache Consistency Protocol. In Proceedings of the 12th

International Symposium on Computer Architecture, pages 276-283, June 1985.

[R.I. Bahar, 1998] R.I. Bahar, G. Albera and S. Manne. Power and performance tradeoffs using various caching strategies. In Proceedings of the International Symposium on Low Power Electronics and Design, 1998.

[Rob Williams, 2006] Rob Williams. Computer systems architecture. 2$^{nd}$ edition, Prentice Hall of India, 2006.

[S. Borkar, 2007] S. Borkar .Thousand core chips: a technology perspective.DAC '07: Proceedings of the 44th annual Design Automation Conference, ACM, New York, NY, USA, 2007, pp. 746–749, 2007.

[S. J. E. Wilton, 1994] S. J. E. Wilton and N. P. Jouppi. An Enhanced Access and Cycle Time Model for On-Chip Caches. Technical Report 93/5, Western Research Laboratory, July, 1994.

[S. J. E. Wilton, 1996] S. J. E.Wilton and N. P. Jouppi. CACTI: An enhanced cache access and cycle time model. IEEE Journal of Solid-State Circuits, 31(5):677–688, May 1996.

[S. Kumar, 2002] S. Kumar, A. Jantsch, J. P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, A. Hemani. A network on chip architecture and design methodology. Proceedings of IEEE Computer Society Annual Symposium on VLSI, 2002, pp. 105-112, April 25- 26, 2002.

[S. Pasricha, 2007] S. Pasricha, N.D. Dutt and M.B. Romdhane. BMSYN: Bus Matrix Communication Architecture Synthesis for MPSoC. IEEE Transactions on Computer Aided design of Integrated Circuits and Systems 26(8), 2007.

[S. Przybylski, 1990] S. Przybylski. The Performance Impact of Block Sizes and Fetch Strategies. In Proceedings of the 17th Annual International Symposium on Computer Architecture, pages 160– 169, 1990.

[S. Roy, 2009] S. Roy. H-NMRU: A Low Area, High Performance Cache Replacement Policy for Embedded Processors. Proceedings of the 22nd International Conference on VLSI Design, pp. 553-558, 2009.

[S. Vangal, 2007] J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, N. Borkar. An 80-tile 1.28tflops network-on-chip in 65nm cmos, in: Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International, 2007, pp. 98–589, 2007.

[Samaher Al-Hothali, 2010] Samaher Al-Hothali, Safeeullah Soomro, Khurram Tanvir, Ruchi Tuli .Snoopy and Directory Based Cache Coherence Protocols: A Critical Analysis. Journal of Information & Communication Technology Vol. 4, No. 1, Spring 2010.

[Sheng-Wei Huang , 2009] Sheng-Wei Huang, Yung-Chang Chiu, Zhong-Ho Chen, Ce-Kuen Shieh Alvin Wen-Yu Su, Tyng-Yeu Liang. A Region-based Allocation Approach for Page-based Scratch-Pad Memory in Embedded Systems. IEEE International Conference on Computational Science and Engineering Vancouver, Canada August 29-August 31, 2009.

[Shivakumar P, 2002] Shivakumar P., Jouppi N. CACTI : An Integrated Cache Timing, Power, and Area Model. WRL Technical Report 2001/2.

[Simple, 2002] http://www.simplescalar.com/ 12 april 2002.

[Stefan Steinke, 2002] Stefan Steinke, Lars Wehmeyer, Bo-Sik Lee, Peter Marwedel. Assign in Program and Data Objects to Scratchpad for Energy Reduction. Proceeding DATE '02 Proceedings of the conference on Design, automation and test in Europe, 2002.

[Steven Przybylski, 1988] Steven Przybylski, Mark Hoowitz, & John Hennessy. Performance Tradeoffs in Cache Design. Proceedings of the 15th Annual Symposium on Computer Architecture, pp. 290-298, 1988.

[Sumesh Udayakumaran, 2003] Sumesh Udayakumaran and Rajeev Barua. Compiler-Decided Dynamic Memory Allocation for ScratchPad Based Embedded Systems. Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems, pp. 276-286, 2003.

[Sumesh Udayakumaran, 2006] Sumesh Udayakumaran, Angel Dominguez and Rajeev Barua. Dynamic Allocation for Scratch-Pad Memory using Compile-Time Decisions. ACM Transactions on Embedded Computing Systems (TECS), Volume 5, Issue 2, pp. 472-511, 2006.

[Sun Microsystems, 2003] Sun Microsystems. UltraSPARCTM User’s manual, [Online document], Available HTTP: http://www.sun.com/ processors/manuals/802-7220-02.pdf, 2003.

[T. Harris, 2003] T. Harris and K. Fraser. Language support for lightweight transactions. In Conference on Object-Oriented Programming Systems, Languages and Applications, 2003.

[T. Moreshet, 2006] T. Moreshet, R.I. Bahar and M. Herlihy. Energy-Aware Microprocessor Synchronization: Transactional Memory vs. Locks. In Workshop on Memory Performance Issues, 2006.

[T. Mowry, 1991] T. Mowry and A. Gupta. Tolerating Latency through Software-Controlled Prefetching in Shared-Memory Multiprocessors. Journal of Parallel and Distributed Computing,12(2):87–106, 1991.

[T. Mowry, 1998] T. C. Mowry. Tolerating Latency in Multiprocessors through Compiler-Inserted Prefetching. ACM Transactions on Computer Systems (TOCS), 16(1):55–92, 1998.

[T. Rissa, 2005] T. Rissa, A. Donlin and W. Luk, Evaluation of SystemC Modelling of Reconfigurable Embedded Systems, DATE, pp. 1530-1591, 2005.

[T.-F. Chen, 1994] T.-F. Chen and J.-L. Baer. A Performance Study of Software and Hardware Data Prefetching Schemes. In Proceedings of the 21st Annual International Symposium on Computer Architecture, pages 223–232, 1994

[T.L. Johnson, 1997] T.L. Johnson and W.W. Hwu, Run-time Adaptive Cache Hierarchy Management via Reference Analysis, In Proceedings of International Symposium on Computer Architecture, 1997.

[Taehwan Cho, 2012] Taehwan Cho and Sangbang Choi . A Multi-path Hybrid Routing Algorithm in Network Routing in International Journal of Hybrid Information Technology Vol. 5, No. 3, July, 2012.

[Taeweon Suh, 2003] Taeweon Suh, Douglas M. Blough, Hsien-hsin S. Lee . Supporting Cache Coherence in Heterogeneous Multiprocessor Systems . Proceedings of the Design Automation and Test in Europe DATE , 2003

[Tilera, 2009] First 100-core processor with the new tile-gx family, http://www.tilera.com/news & events/press release 091026.php, Oct, 2009.

[Varun Gandhi, 2013] Varun Gandhi, Vicky Ahuja, Varsha Yadav, Vivek Anand. Shared memory multiprocessor: a communication. Discovery Engineering, volume 2 number 8, pp 94-96, 2013.

[Vikash Kumar Singh, 2011] Vikash Kumar Singh, Hemant Makwana, Richa Gupta. Comparative Study of Cache Utilization for Matrix Multiplication Algorithms. IJCTA Volume 2 issue 4, pp 1078- 1081, August 2011.

[W. J. Dally, 2001] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection. Proceedings of the 38th Design Automation Conference. June 2001.

[W.C. Jeun, 2007] W.C. Jeun and S. Ha. Effective OpenMP Implementation and Translation for Multiprocessor System-on-Chip without Using OS. DAC, pp. 44-49, 2007.

[William Stallings, 2009] William Stallings. Computer Organization and Architecture. Eighth Edition Prentice Hall, 2009 .

[Yong Yan, 2000] Yong Yan, Member, Xiaodo Zhang. Cacheminer: A Runtime Approach to Exploit Cache Locality on SMP. IEEE Transactions on Parallel and Distributed Systems, 11(4), 2000.

[Z. Marrakchi, 2009] Z. Marrakchi, H. Mrabet, U. Farooq, and H. Mehrez. FPGA interconnect topologies exploration. International Journal of Reconfigurable Computing, vol. 2009, Article ID 259837, 13 pages, 2009.

[Zheng Xu, 2013] Zheng Xu. Control Techniques for uncore power management in chip multiprocessor designs. Master of Science thesis submitted to the Office of Graduate Studies of Texas A&M University, august 2013.