IMPROVING PERFORMANCE, POWER AND SECURITY OF MULTICORE SYSTEMS USING SMART CACHE ORGANIZATION

A Thesis by

Tania Jareen

Bachelor of Engineering and Technology, Khulna University of Engineering and Technology,

2010

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

May 2014

IMPROVING PERFORMANCE, POWER AND SECURITY OF MULTICORE SYSTEMS USING SMART CACHE ORGANIZATION

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Electrical Engineering.

_____
Abu Asaduzzaman, Committee Chair

_____
Ramazan Asmatulu., Committee Member

_____
Zheng Chen, Committee Member

DEDICATION

To the Almighty, my loving parents, and my sisters for their encouragement throughout my education, for their ultimate support and advice throughout my life. At last to my husband for his encouragement and support.

ACKNOWLEDGEMENTS

I am thankful to my thesis advisor Dr. Abu Asaduzzaman for his continuous encouragement and support throughout my research work. His timely supervision of my work and guidance allowed this research work to be completed on time. He always had time and patience to guide me in accomplishing this work in spite of his busy schedule and offered me assistance from time to time.

I express my deep gratitude toward Dr. Ramazan Asmatulu (WSU ME Department) and Dr. Zheng Chen for taking time from their busy schedules and serving in my thesis committee.

I take pleasure in recognizing, the efforts of all those who encouraged and assisted me both directly and indirectly with my experimental research. Finally, I acknowledge the WSU CAPPLab research group and facilities for providing me with all the necessary resources to prepare my research work.

ABSTRACT

The need of multicore/manycore systems for today's world is significantly increasing. But the multicore system is considered to be power-hungry as well as high latency system. Different researches show that it is possible to increase the performance to power ratio by wisely locking the memory blocks inside the cache memory. But this method introduces cache underutilization problem which reduces the effective cache size and also it is hard to configure. Also depending on the processor type, some processor may not have the option of cache locking. Also cache side channel attack and cache interference become a security threat for the cache design. In this paper, a smart cache technique is proposed which decreases the memory access latency and cache power consumption, as well as increases the overall system security. Propose smart victim cache (SVC) between level-1 cache (CL1) and leve-2 cache (CL2) eliminates the cache locking. SVC holds the higher missing memory blocks and also supports stream buffering. For security improvement for the cache, we randomize the cache mapping between main memory and CL1. The randomized cache mapping makes the attacker fool by showing the false positions of the memory blocks in the cache. In the experiment, a quad-core Intel-type system is used, where CL1 is private and CL2 is shared among the cores. A tree based analyzer HEPTANE (Hades Embedded Processor Timing Analyzer) and a system level simulator VisualSim are used on diverse applications (including MPEG-4 and H.264/AVC). From the simulation results, it is seen that 17% of memory access latency and 21% of total power consumption is reduced using SVC comparing with cache locking without using SVC. For 16-block CL1, it is estimated that the probability of cache side channel attack reduces from 40K to 1.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In this chapter we will discuss about the detail overview of cache organization, cache mapping, cache replacement policy, cache update policy and cache locking. Also we will discuss the performance, power and security issues for cache.

1.1 Cache Organization

Cache is a small essential buffer which is used to store recent information. It is used to mitigate the speed gap between the processor and the main memory. The memory speed of the recent modern computer has increased significantly than before, but though it is much less compared to the processor speed. One of the big obstacles to increase the memory speed is the increase of total cost [1]. This speed gap between the processor and memory introduces serious performance issue for the computer system. Two important parameters to characterize the performance of processor - memory are – the bandwidth and the latency [2]. Bandwidth can be defined with the rate at which any information is transferred from or to the computer memory system. On the other hand, the latency can be defined with the time difference between a request initiates and the request completes by the processor. Zero latency and infinite bandwidth can ensure the maximum performance of a computer system. The speed gap problem of the processor – memory creates the latency issue which affects the overall performance of the computer system. Cache is introduced to solve the speed gap problem and increase the overall performance of the computer system.

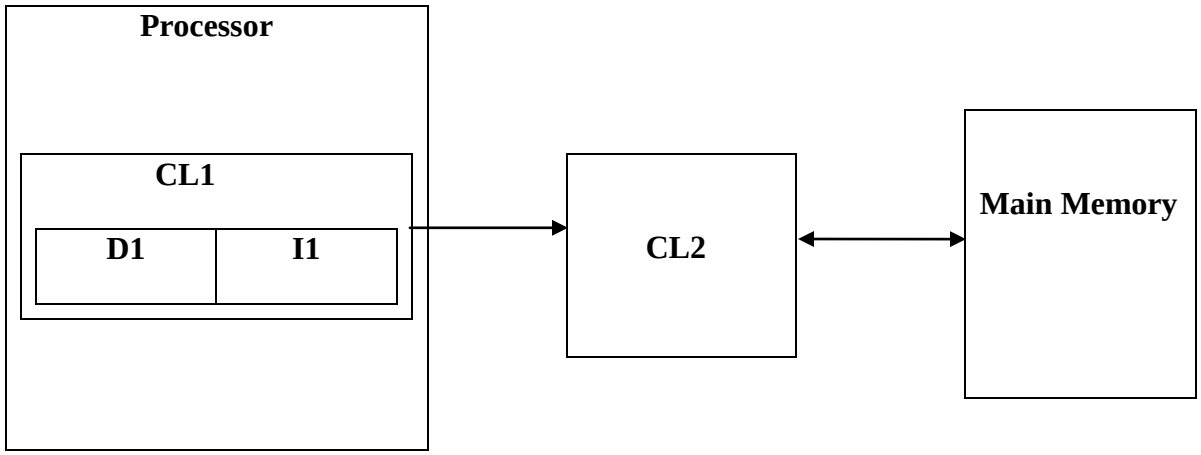

Figure 1.1 Cache Organization

Cache increases the performance significantly as it stores the most frequently accessed important data which saves time by not searching the information every time in the main memory. At first, only CL1 cache was introduced, but with the increasing processor speed multiple levels cache hierarchies has been introduced now. Figure 1.1 shows 2 level of simple cache organization. CL1 is the unique cache which is divided into data caches and instruction caches. When CPU needs data, first it searches the data from CL1 cache. If the CPU gets the data in CL1, which is called the cache hit, the CPU can responds to the instruction at a very high speed. But if the CPU does not get the data in CL1, which is called cache miss, the CPU searches the data into CL2 cache. If the CPU gets the data in CL2, the data will be propagated every single level of the cache which had a cache miss for the data.

Cache is designed in a way so that the requested instruction can be easily found in the cache. Logically cache is used between the CPU and main memory. But physically cache can be placed in several different places. Three design concerns for the cache are – cache access, cache replacement and write policy. Cache access represents where a block can be placed in the cache

as cache is dived into block and line. Cache replacement indicates which block from the cache should be replaced as the cache memory is limited. Write policy defines whether the information should be written only on the cache or both on the cache and main memory. The performance of the cache depends on these design concerns and so the overall performance of the computer system. Three popular cache mapping methods – direct mapped, set associative mapped, fully associative mapped, are used to map the memory lines to the cache lines. We will discuss more about cache mapping on the coming chapters.

Cache locking and victim cache are two popular methods of improving cache performance. Cache locking means lock some of the cache lines or blocks with predicted information, which increase the cache hit and overall performance [3]. Victim Cache is a small cache memory, used between the main memory and the cache, to store the information which is evicted from the cache [4]. It works as a secondary cache and also helps to improve the performance by reducing the time to search the information in the main memory. We will give more in depth idea on the cache locking and the victim cache on the coming chapters.

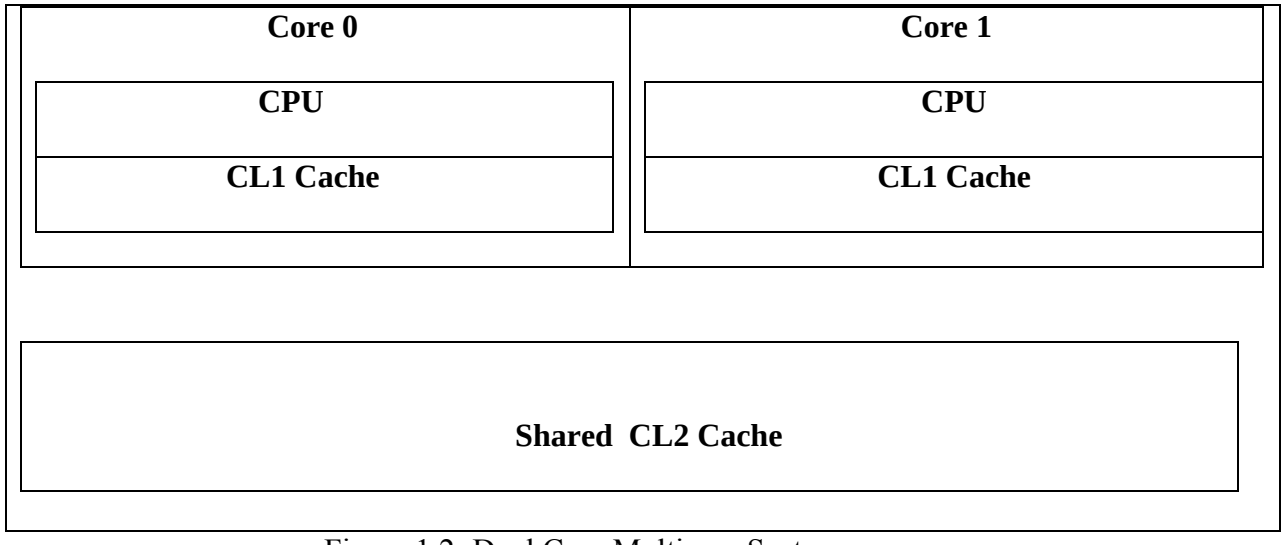

Figure 1.2: Dual Core Multicore System

The demand for multicore system is rising day by day for both desktop and embedded systems [25-27]. Multicore system is a collection of parallel or concurrent processing units which solves the large complex problem fast. Fundamentally, the multicore systems are designed to improve the power/performance ratio with the help of dividing the large problem into many small tasks and then running the small tasks in parallel on multiple cores. Figure 1.2 shows a simple dual core multicore architecture, where core 1 and core 2 work parallel with each other to process a complex problem faster. Different approaches of parallel techniques like pipelining and caching have been used to improve the power and performance of the overall system. Most of the cases, the researchers are able to improve the performance by decreasing the average access time to memory [21, 22]. But due to use of numerous multilevel caches in the multicore system, the power consumption and the average memory latency become very high. So it becomes a great challenge to achieve low latency and low power requirements for the multicore system [23, 24].

Different types of cache organization and memory optimization have been introduced to decrease the average memory latency and power consumption for the multicore system. Researchers used victim cache to keep the victim blocks to decrease the latency and also stream buffers which needs stream buffer, though is a better technique of prefetching has been introduced. Most of these researches are done only to improve power consumption and overall performance of the multicore system. But for today's world security has become also a great concern with power and performance. In case of multicore also, as many cores in the multicore system share the same cache, it becomes a vulnerable place to deploy attack. Some of security researches have been done, but they were concentrating on software level security. But recent researches have shown that it is easy to break software level security. So it is important to

implement hardware level security along with software level security for better result. We will discuss in details about the performance, power consumption and hardware level security on the coming sections.

1.1.1 Cache Mapping

Cache mapping is an important concept for cache performance. There are three well known cache mapping techniques – Direct mapped, Set Associative mapped and Fully Associative mapped.

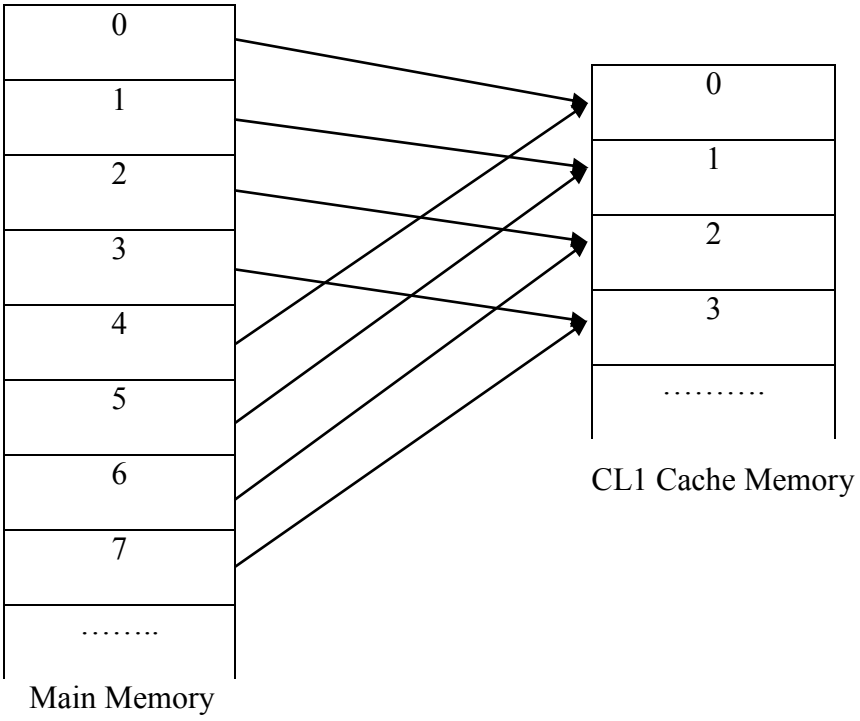

Figure 1.3: Direct Mapped Cache

Direct mapped cache is where each memory block is mapped to a specific place in the cache. In Figure 1.3 Each block of the main memory has a specific place in the cache memory. As each word has a specific location in the cache, there is no need of cache replacement policy.

Direct mapped is cheap but the addresses compete with each other for getting into the cache line. To overcome the access problem of Direct mapped cache, Set Associative mapped and Fully Associative mapped caches have been introduced.

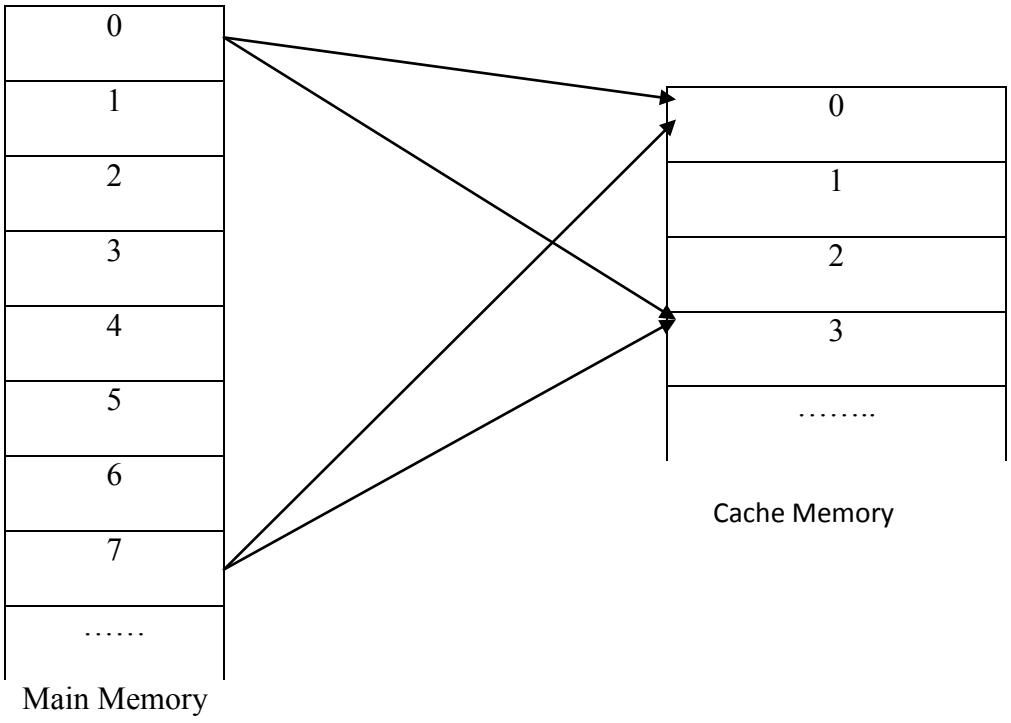

Figure 1.4: Fully Associative Mapped Cache

Set Associative mapping and Fully Associative mapping solve the problem of contention during direct mapped caching. In Fully Associative mapped cache, a memory block can be placed anywhere in the cache. In Figure 1.4, the memory block of the main memory can mapped to any place of the cache memory. Full Associative mapping needs replacement policy during loading new blocks in the cache as the blocks are not fixed in the cache. As the specific block can be anywhere in the cache, it is needed to be searched in the whole cache, which takes more time compared to other mapping policies.

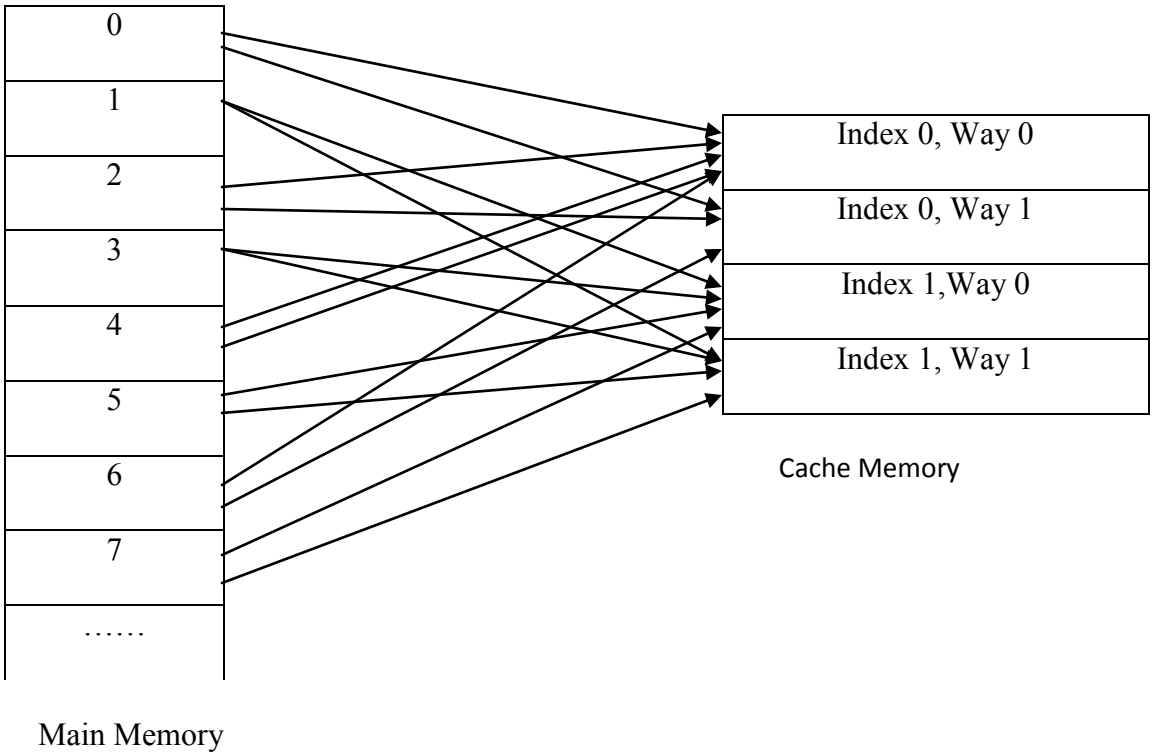

Figure 1.5: 2-Way Set Associative Mapped Cache

Set associative cache mapping is the combination of Direct mapped and Fully Associative mapped cache. It maps the memory block into cache memory in a grouped fashioned way. The cache is divided into multiple groups of blocks and the memory blocks are mapped as a direct mapped fashion into those blocks. In Figure 1.5, a 2-way Set Associative mapped cache mapping has been shown, where a block from the main memory can be placed either way 0 or way 1 of the cache memory. Set associative cache mapping saves the time of searching as no need to search the entire cache like Fully Associative mapping. Also, it does not need to use the replacement policy frequently as when a set is full, only then it needs to be replaced. It shows better performance than Direct mapped and Fully Associative mapped.

1.1.2 Cache Replacement Policy

Cache replacement policy is an important principle for cache design. Cache is a small size buffer, whose size is fixed. So when it become full, some blocks need to be replaced so that new memory block can be loaded. The replacement should be in a manner that miss ration will be low and hit ration will be high. Cache replacement policy decides which block should be replaced from the cache so that hit ratio will always be high. For direct mapped cache there is no replacement policy. Only for Fully Associative and Set Associative cache mapping have the cache replacement policy.

Different strategies are used to replace the block from the cache. Two of the most popular strategies are – Lease Recent Used (LRU) and Random. LRU is the cache replacement strategy where the blocks are replaced which are unused for a long time. This strategy reduces the chance to through the blocks from the cache which will be needed soon. But it is difficult to implement, expensive and also the performance is slow. Random strategy replaces blocks randomly.

There are also other cache replacement policies – Most Recent Used, First In First Out, Least Frequent Used, Most Frequent Used etc.

1.1.3 Memory Update Policy

Memory update policy is the combination of read policies and write policies. Read policy indicates how a word is to be read. Two methods are used for read policy – Read Through and No Read Through. Read Through method of read policy is used to show how a word from main memory to the CPU will be read and No Read Through method shows how a block will be read from the main memory to the cache memory and then from the cache memory to the CPU.

The write policy indicates how the write of a memory will be handled. There are two popular methods of write policy. One is Write – Through and the another is Write – Back. Write – Through method of write policy updates both the main memory and the cache memory where Write – Back method updates the cache memory only. Only when the cache blocks are replaced, they are written to the main memory. For Write – Through method the main memory is always consistent with the cache as it has the same copy of memory as cache. It is also easy to implement, but it is slower as every write needs to write on the main memory and so needs more bandwidth. Write – Back is faster, uses less bandwidth compared to Write – Through, but harder to implement.

1.1.4 Cache Locking

Cache locking is an effective feature to secure the most usable cache memory and also to increase the hit ratio which improves the overall cache performance. Cache locking can be defines as the ability to lock the most usable data or instruction cache lines. During replacement policy, these locked cache line will never be replaced, only the unlocked cache lines will be replaced. As these locked cache lines are the most usable cache line, so this feature also increase the hit ratio and make the cache performance better.

Cache memory is very hard to predict. Cache locking is an effective feature to reduce cache miss. It also improves the predictability, which reduce the access time to the blocks for future and lower the power consumption. Cache can be locked for some cache line or for the entire cache. Cache hits for a cache locking system work similarly as a cache unlocking system. The invalid cache for a cache locking system remains invalid as long as the cache becomes

unlocked. Though the cache locking feature increases the efficiency of the cache behavior, it become inefficient if the locked cache size is smaller compared to the size of the whole cache.

1.1.5 Victim Cache

Victim cache is one of the most popular and oldest techniques to improve the cache performance. Average memory latency can be reduced significantly by using victim cache with CL1 and also CL2. Victim cache implementation is similar to the implementation of CL1, but the main difference between these two cache type is the data input and output techniques [31]. The victim cache is implemented between CL1 and CL2. As cache has a definite size, when it becomes full, cache replacement policy is used to replace the dead blocks. Victim cache holds these dead blocks and reduces execution time and latency by not searching those dead blocked in the main memory for future references. When there is a miss, first it is searched in CL1, if the block is not found in CL1, it is searched in the victim cache. If the desired block is found in the victim cache, the block is brought to CL1. If the block is not found in victim cache, then only it is searched in the main memory. This is how, the victim cache not only reduces the access time and latency but also improves the overall performance of the system. Victim cache is also used to remove conflict misses [20].

1.1.6 Stream Buffer

Stream buffer is an efficient technique of prefetching cache lines. Stream buffer supports stream buffering. It is used to remove capacity and compulsory miss. Also it can reduce cache conflict miss. Stream buffer prefetches successive cache lines when there is a miss, starting from the cache miss address. During a cache miss, not only the required blocks but also some additional blocks are prefetched from the main memory to CL2 and then copy to CL1.

Prefetching these additional blocks are called stream buffering and these additional blocks are kept in the stream buffer for future use as it is desired that, the additional blocks will need soon. This helps to reduce the memory latency and total power consumption and also improves the overall efficiency of the system.

1.2 Performance

Cache is introduced to increase the performance of the system. Cache performance can be defined as the average memory access time.

Average memory access time = hit time + miss ratio * miss penalty

To understand the cache performance, three important terms are – hit ratio and miss ratio and miss penalty. Cache hit represents that the reference data is found in the cache and cache miss represents that the reference data is not in the cache. Cache hit ratio is defined by the percentage of how much data from the requested memory is found in the cache. Cache miss ratio is defined by the percentage of how much data from the requested memory is not found in the main memory. Miss penalty is the time to load data from main memory to cache when there is a cache miss.

$$\text{Cache Hit Ratio} = \frac{\text{Total No.of cache hit}}{\text{Total No.of requests}}$$

$$\text{Cache Miss Ratio} = \frac{\text{Total No.of cache miss}}{\text{Total No.of requests}}$$

Cache hit ratio and cache miss ratio can be represented by this two equations. For improve cache performance, the time to heat the cache, miss rate and the miss penalty should be reduced.

1.3 Power Consumption

Energy efficiency is a very important issue for today's world. Cache increases the performance of the CPU, but at the same time it consumes a significant amount of energy. To improve the performance a huge amount of transistors are used, which increased the power consumption significantly. Research showed that, cache can consumes up to 70% of the total power [5,6]. More power consumption not only costs more to the users but also generates a large amount of heat which creates an extreme condition for the user to use the device. Also the portable devices are depended on the battery which becomes difficult to provide services for long time for high power consumption. Most of the previous researches were only to improve the performance. But now-a-days the design goal of the cache is to increase the performance with low power consumption. Many researches and techniques (like victim cache, selective pre-fetching) are proposed to reduce the power consumption. But these techniques may increase the power consumption if the victim blocks are not predicted correctly or aggressive pre-fetching happens. So we want to propose a smart cache organization which reduces the total power consumption significantly without any prediction.

1.4 Security

Security is the biggest issue for today's world with all sorts of communication and computing devices [18]. It is the most recent concern for designing cache. As cache is the most shared resources in the computer system, they are more vulnerable to attack [7]. Previous researches for security are mostly for software security. But cache level attack is enough to break software level isolation mechanism [34]. The most popular lower level programming languages like C as well as many higher level programming languages like Java, has many implementation bugs which

insecure the software level security significantly [9,10]. A good example of this type of problem can be virtual machine which runs on same processor and shares same cache though they are logically separated. So it is very easy to break the software level security if there is no processor level security.

In the modern era, billions of transistors are integrated into same chip to enhance the performance which makes more difficult to identify information leakage and security attack. Researches have been shown that the recent sensitive attacks like leakage the cryptographic keys are demonstrated from shared caches. Side-Channel attack is one of the most dangerous processor level security attack and when it attacks the cache it is called Cache based Side-Channel attack [8]. It can be deployed very quickly without any extra hardware or software. Not only that, the victim even cannot detect this attack on the machine. So this attack is called the silent cache attack. Cache based Side-Channel attack is the most concerning issue for cloud computing security. So it is very necessary to implement processor level security along with software level security.

Three important components to ensure security are – confidentiality, integrity and availability. Confidentiality means the concealment of the data so that unauthorized access can be denied. Integrity can be defined as denial of unauthorized data modification. Availability ensures the reliable access to the data.

Hardware level security (cache level security) along with software level security is very important to implement to ensure the overall system security. We want to implement a hardware security by organizing the cache effectively to secure the system from the possible cache level attack.

1.4.1 Cache Side Channel Attack

Cache Side Channel attack is the hardware level attack to exploit important information by passively monitoring the physical properties like timing variation, power consumption, sound variation, heat produce etc [11, 12, 13, 14]. Though theoretically Cache Side Channel attack can attack any kind of system, normally it is implements on cryptosystem including simple crypto-graphic devices like smart card. The goal of Cache side channel attack is similar to classical crypto-analysis attack, but in the classical crypto-analysis attack, attackers attack the weak mathematical composition of the object where in cache based side channel attack, attackers target the physical object and attack physically. This makes the cache based side channel the most powerful and dangerous hardware level attack. Security can be provided to the system by using strong mathematical cipher in case of classical crypto-analysis attacks, but it is impossible to provide security against Cache Side Channel attack without any strong hardware level security.

The side channel attack for cache behavior was first introduced by Kocher [15]. He showed that by observing the access time to perform any private key operation, the attackers can able to extract the Diffie-Hellman private keys.

1.4.2 Types of Cache Side Channel Attack

Cache side channel attack is one of the most dangerous hardware levels security attacks. Cache side channel attack can be classified into different categories depending on computational process, accessing module and analysis process. The cache side channel attack can be classified into three levels [18]:

- Classification on action over computational process
- Classification on the way of accessing the module

- Classification on analysis process

1.4.2.1 Classification on Action over Computational Process

Side channel attacks are two types depending on the action over the computational process – Passive Attack and Active Attack. Passive attacks are those attacks which do not interfere the normal behavior of the system. It just observes properties of the system like access time, power consumption and gains the necessary information of the pattern of operation. The target system cannot be informed about the passive attack. On the other hand, the active attack is the attack where the attacker tries to interfere the system's properties and so the system's normal behavior like by changing useful information of the system or introducing some computational error. The target system can easily notice some difference in the operation during active attack.

1.4.2.2 Classification on the Way of Accessing the Module

Depending on the way the cache access the module side channel attack can be classified into three categories – Invasive attack, Semi-Invasive attack and Non-Invasive attack. Invasive attacks are done to get information of the internal parts of the target system via deconstruction like depackaging chip to access internal data bus to observe data transfer. Non-Invasive attack exploits only the external information of the target system like access time or power supply by monitoring or manipulating the operation of the system. This type of attack is totally undetectable and low cost to deploy. On the other hand, Semi-Invasive attack was first introduced by Skorobogatov et al. where attacks are done by depackaging the device to get internal access but without making any damage with the internal components [17].

1.4.2.3 Classification on Analysis Process

Based on analysis process cache side channel attack can be classified into two categories – Simple Side Channel Attacks and Differential Side Channel Attacks. Simple side channel attacks exploits information based on performed operation. It keeps a single trace only and so easy to extract secret key directly from the side channel trace. The differential side channel attack exploits information based on performed data. For differential side channel attack, many traces are gathered and so statistical method is used to extract secret key. Differential side channel attack is more powerful than simple side channel attack.

1.5 Problem Statement

High performance, low power consumption and better security are the ever growing demands for computing systems. As the multicore systems use multiple levels of caches, it consumes high power and introduces unpredictable memory latency. Also the traditional cache becomes a vulnerable place to attack the system. Currently there is no suitable technique to enhance performance, power and security.

1.6 Contribution

In this work, a smart victim cache is introduced to improve performance to power ratio. Also, a new memory to level-1 cache mapping technique is introduced to improve security. Major contribution of this work includes:

- A smart victim cache for multicore systems to reduce the power consumption and the average memory latency.
- A hardware based technique to improve security.

1.7 Thesis Organization

The rest of the thesis is organized as follows:

Chapter-2 describes the literature review of current techniques of cache organization to improve power, latency and security.

In the Chapter-3, represents the proposed architecture to improve power, latency and security.

Chaper-4 represents the analysis the simulation for the proposed cache architecture.

Chapeter-5 describes the results and discussion on the proposed design of cache implementation according to the simulation results.

Chapter-6 describes the conclusion and future extension of the work.

CHAPTER 2
LITERATURE SURVEY

For multicore architecture, cache is a very sensitive place which introduces low performance, high power consumption and security issues. But recent many techniques or methods have been proposed to solve the cache design issues. This chapter provides a brief overview of the recent researches on different methods of cache system design.

2.1 Performance Improvement by Selective Victim Cache and Prefetch Buffers

A new hardware - victim cache was first introduced by P. Jouppi between CL1 and main memory. Also he introduced stream buffer along with fully associative victim cache to improve the cache performance and power consumption [20]. Victim cache was used to hold the victim blocks and stream buffer to improve the pre-fetching by bringing multiple blocks from the main memory. He showed that for using victim cache and stream buffer the performance improved orthogonally and also power consumption decreases significantly.

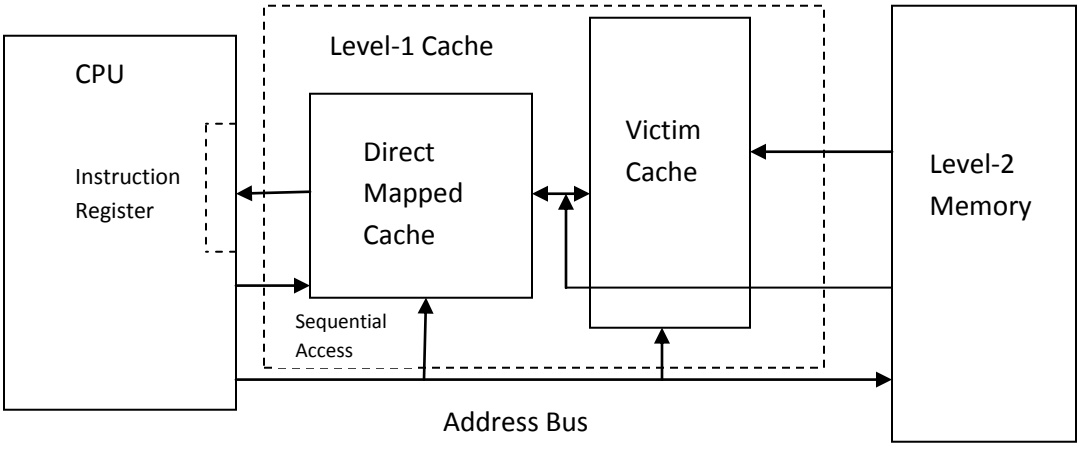

Figure 2.1: Selective Victim Cache Memory [35]

Stiliadis and Varma improved the victim cache architecture by adding prediction algorithm called selective victim caching [35]. Figure 2.1 shows cache organization with fully associative victim cache. In this cache architecture, victim cache is used to hold the victim blocks and also some incoming blocks. A prediction algorithm is used to predict whether the incoming block should be placed in the victim cache or in the main memory during a new block fetches to the CL1. This method shows that the miss rate decreases sharply compared to using only the victim cache in Jouppi's cache architecture.

Though this cache architecture improves the cache performance and reduces the power consumption significantly, it has a drawback that it does not guarantee to lock the blocks with maximum misses. Also there is a possibility to pollute the cache by unnecessary pre-fetches. This can increase the miss penalty along with serious performance and power issue.

2.2 Selective Pre-fetching When Required

As the previous selective pre-fetching technique may introduce great drawback on performance and power issues of the system, Pendse and Katta proposed selective pre-fetching technique based on the history of references which is used to decide the blocks to pre-fetch on a miss [36]. This technique is able to cut down the cache miss by 25% to 50%.

2.3 Partitioned Cache Architecture

Partitioned cache was first introduced by Page and was a popular method to solve cache side channel attack [11]. The partitioned cache can be defined as direct mapped cache which is partitioned into protected regions with partitioned identifier. The partition is done via software based cache partitioned method [16]. Either a register or an operand, depending on cache

architecture, is used as cache partitioned identified. Embedded and media processor use partitioned cache architecture because of their characteristics like power, performance and size.

Page added three new features on cache system design to implement cache partitioning successfully. At first, he included stride cache line which is actually the distance between two sub words in the cache to prevent interference. This can be configured per partitioned basis. Secondly Page defined a fixed size of physical line to span the virtual line into physical lines. Lastly, he added mask to the original address for each cache partition to allow randomized address translation by virtually moving the address according to the cache operation.

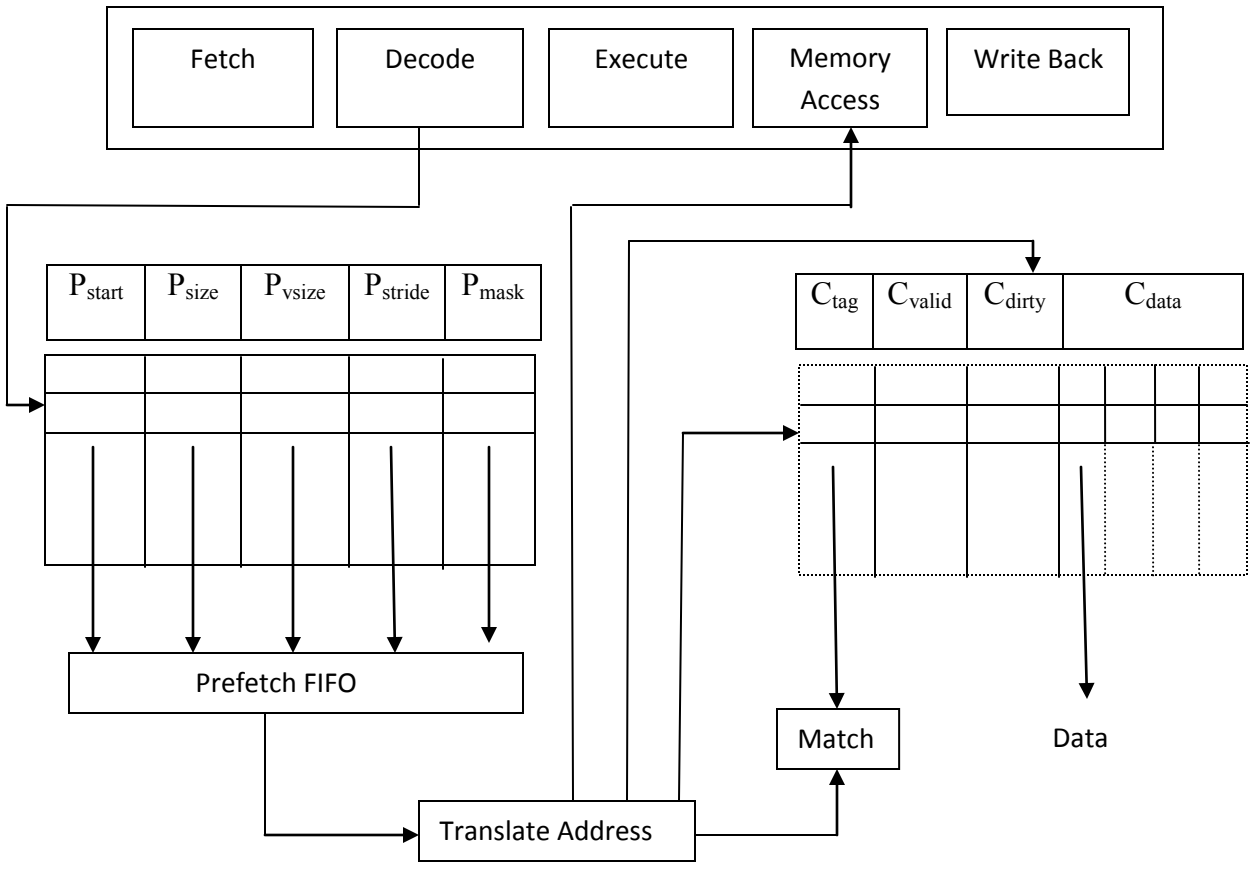

Figure 2.2: The Partitioned Cache Architecture [11]

Figure 2.2 describes the operation of a partitioned RISC processor architecture. Here, P_{start} is the line from where the cache partitioning beings, P_{psize} is the number of physical lines which are composed of sub words, P_{vsize} is the number of physical lines per virtual line, P_{stride} is the strides between the sub words and P_{mask} is the address perturbation mask.

The cache partitioned can prevent inter process attack as access from one partition to another is totally invalid. Also it can prevent from flushing the cache which is a technique to attack on cache. In this hardware design, the virtual cache line size allows to design larger fetch size which can make the attack difficult to implement. Cache partitioning also prevent unwanted cache eviction and cache interference as objects are kept into private partition.

2.4 Dynamic Memory-to-Cache Remapping

Cache partitioning the way to put the cache address into private or locked partitioned cache and prevent unnecessary cache eviction. But this has a significant drawback and that is cache underutilization. Cache partitioning uses private or locked space which cannot be used by other process even if it is empty or unused. This reduces cache performance significantly. Also for partitioning the cache, it needs to trust the software, which can be a weak point to attack.

Wang and Lee proposed a novel cache architecture named Newcache with dynamic memory-to-cache remapping technique to prevent cache side channel attack [17]. It solves cache underutilization problem of cache partitioning and cache line locking as the memory blocks can be mapped to dynamic cache line.

This proposed Newcache architecture is much similar to direct mapped cache architecture. They modified the address decoder of the Newcache by integrating LNregs to implement dynamic mapping of memory to cache. Some extra index bits, n+k are accessed to

the $s < 2^{n+k}$ sized cache. Various process can have various memory to cache mapping as each process uses separate RMT ID to identify the remapping table of the process.

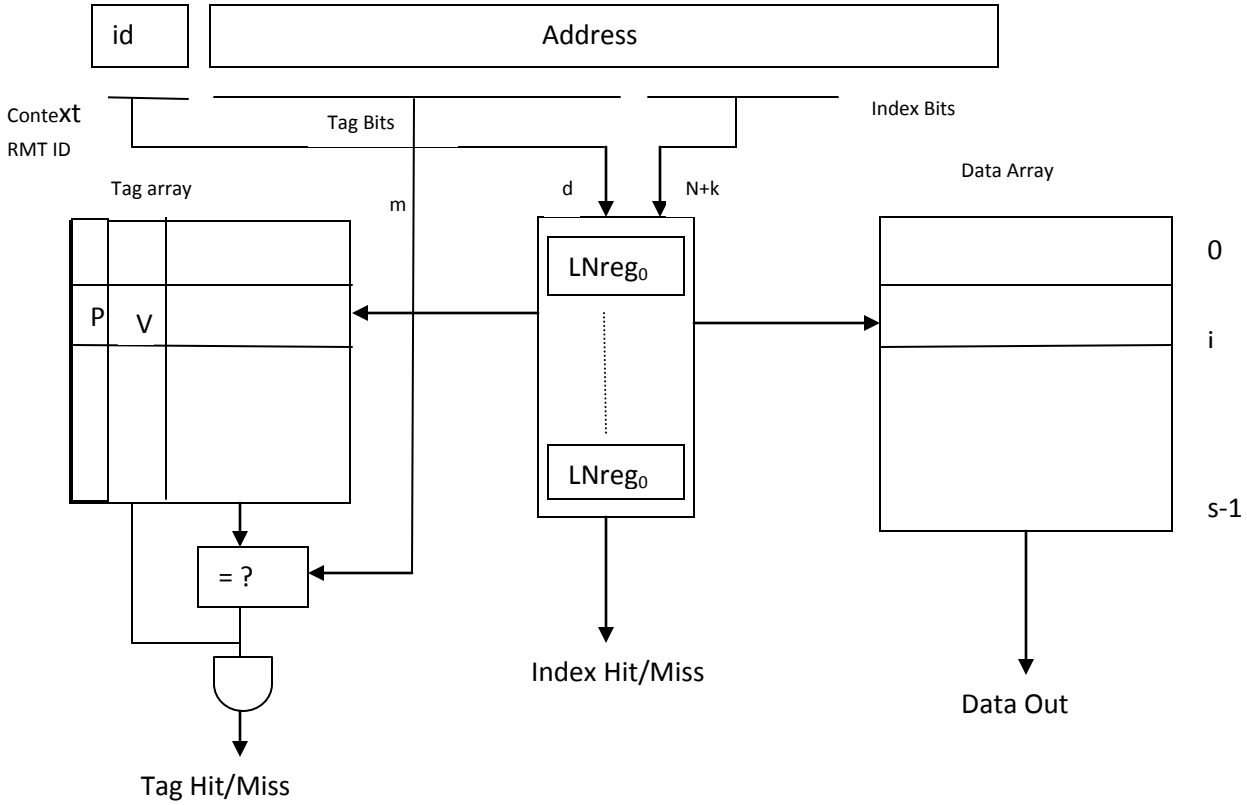

Figure 2.3: Dynamic Memory to Cache Remapping [17]

The research showed the Newcache architecture provides dynamic cache partitioning which prevents the underutilization problem and also provides fast, low power consumption and more secure cache architecture. Comparable with cache partitioning and cache line locking, the Newcache also provides some additional benefits like fault tolerance and hot-spot mitigation.

CHAPTER 3

PROPOSED CACHE MECHANISM

Multicore is a way to split a large complex problem into various concurrently running small tasks to make faster the whole process. As a lot of cores runs simultaneously to fast the process in the multicore system, it needs huge power to operate. As a result the overall multicore system becomes a multi-level power hungry architecture. Various techniques have been proposed to solve the problem. In this research, a Smart Victim Cache (SVC) is proposed to decrease the power consumption and memory access latency. But victim cache is not a new idea. Many researches have been done on victim cache along with level-1 cache (CL1) and level-2 cache (CL2) to improve average memory latency. But using cache locking can decrease the average memory latency by locking some or all blocks inside the cache for entire execution time. So we propose a smart victim cache architecture which totally eliminates the locking mechanism and improves the multicore system performance and lower the power consumption. Also the proposed SVC architecture stores victim blocks, streaming blocks along with block addresses and cache miss information (BACMI) entries [28] which helps to improve overall performance and power ratio of the multicore system.

3.1 Proposed Solution to Improve Performance and Reduce Power Consumption

An Intel like quad-core architecture is used for the proposed cache architecture. The cache architecture is selected where each core has own private CL1/ PFLC and this is divided into two partitioned cache – Instruction cache (I1) and Data cache (D1). The CL2 or shared last level cache (SLLC) of the proposed architecture is unified and shared by the all the four cores of

the system. The main memory is connected to SLLC. In the proposed design, the main objective is to reduce the communication latency and power consumption among the cores by decreasing the number of hops in traversing from the source core to the destination core. In multicasting architecture of the mesh topology flooding is among all the cores of a multicore architecture, if a source core requests the data which is present in some unknown cache of a destination core then it sends a multicast message to all the cores requesting for the data which makes the network congested and moreover, a lot of power is consumed and more heat dissipation takes place which is not preferred in the designing of a good architecture.

3.1.1 Design of SVC

The SVC of the proposed architecture is placed between CL1 and CL2, same like regular other victim cache. As shown in the figure 3.1, the CL1 is private to each core and it is divided into two parts – data cache (D1) and instruction cache (I1). SVC is connected to all the I1 s and D1of each core of the proposed multicore architecture. SVC is connected to the shared, unified CL2/ shared last level cache (SLLC). The main memory of the system is connected to SLLC.

Logically SVC is also divided into two parts – SVC1 and SVC2. SVC1 holds miss cache blocks, so in other word, it can be called missed cache block (MCB). SVC2 again is divided into two parts – one is victim cache blocks (VCB) and another is stream buffering blocks (SBB). Top few blocks of the VCB is considered as SVC1/MCB and it holds block address and cache miss information (BACMI). During the run time the SVC1/MCB cannot be changed. The bottom few blocks are considered as SVC2. VCB are top few blocks of SVC2 and holds the victim cache blocks. Lower few blocks of SBC2 are called stream buffering blocks (SBB) which is used to temporary hold the extra blocks during stream buffering. SVC2 is sorted after each operation of

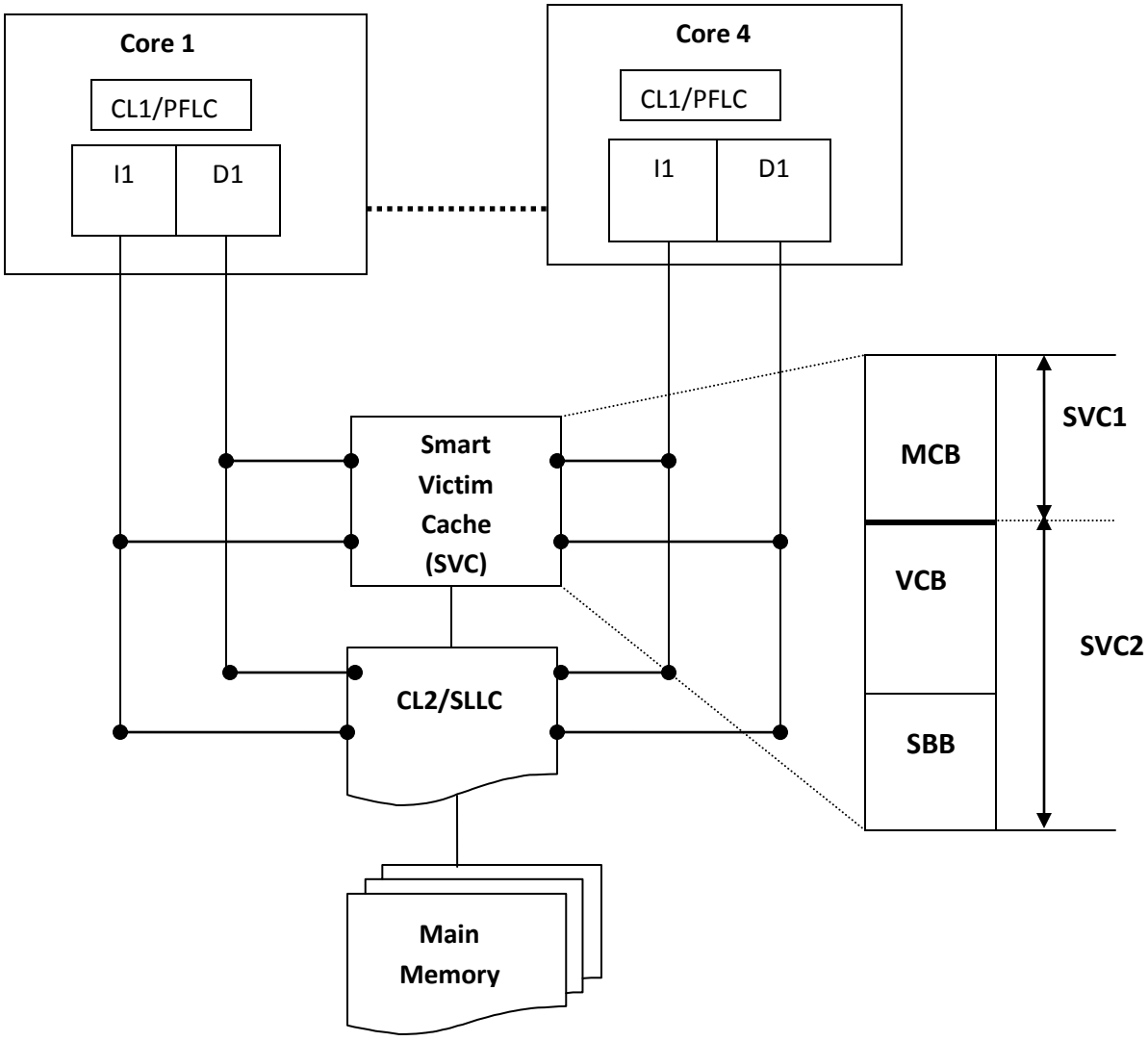

Figure 3.1: Cache Organization with Smart Victim Cache (SVC)

the blocks (like swapping between SVC1 and CL1). This helps to move the blocks with the lowest cache missed to the bottom.

Below are the different cases that show how to divide the SVC into MCB, VCB and SBB. We consider the each block size of 128 Bytes and 4 GB (= 2^32 bits) main memory; 64 bits BACMI entry where 32 bits among the 64 bits of the BACMI entry are considered for address location and rest of the 32 bits are for the number of misses. That means, 16 BACMI entries (= 128*8/64) can be hold by each block.

Case 1: If we consider the bottom 8 blocks for SVC2 where 5 blocks are for VCB and 3 blocks are for SBB, and top 8 blocks are for MCB to store BACMIs, then total the total number of BACMI s that can be stored is 128 (=16*8 = per block number of entry * number of MCB blocks).

Cast 2: If the bottom 11 blocks is considered for SVC2 where 8 blocks are for VCB and remaining 3 blocks are for SBB, holds victim cache blocks (VCB) and top 5 blocks are for MCB to store BACMIs, then the total number of BACMI is 80 (= 16*5 = per block number of entry * number of MCB blocks).

To fit 16 blocks in SVC, the size of the SVC should be 2 KB (= 16*128). The number of blocks required for various SVC, MCB, VCB and SBB are shown on Table I.

Table I: Maximum number of BACMI entries for a given SVC with various MCB size (SSB size = 3 blocks)

Block size = 128 Bytes; main memory = 4 GB				
SVC Size (KB)	Num. of Blocks	SVC1: MCB (Block)	SVC2: VCB+SBB (Block)	Max. Num. of BACMIs (MCB*16)
2	16	8	5 + 3	128
		5	8 + 3	80
4	32	8	21 + 3	128
		16	13 + 3	256
8	64	8	53 + 3	128
		48	13 + 3	768
16	128	8	117 + 3	128
		112	13 + 3	1792
32	256	8	245 + 3	128
		240	13 + 3	3840

From the table I, it is cleared that 8 blocks VCB can be used between MCB and SBB for a 2 KB SVC, where 3 bottom blocks can support stream buffering and 80 BACMI entries can be

stored by 5 MCB blocks. For same size of 2 KB SVC, 5 blocks of VCB can be used between MCB and SBB, where same 3 bottom blocks are for stream buffering, can hold 128 BACMI entries by 8 MCB blocks. From the table I, it is cleared that the number of BACMI entries increases with the number of MCB block. That means, a bigger MCB can hold more BACMIs. So it is important to determine the effective size of the SVC1/MCB depending on the application. A bigger MCB is needed to store all or the most of the miss blocks if the application has more blocks of cache misses. The maximum number of BACMI entries can be stored is 3840 for a 32 KB SVC with 13 blocks of VCB and 3 blocks of stream buffer.

3.1.2 Work Flow Diagram

In the proposed architecture, the SVC is designed in a manner that it can be disabled or enabled anytime. If the SVC is enabled, it should work with both CL1 and CL2 synchronously at the correct time. But at the same time the proposed SVC should start functioning similar to any regular system without any victim cache.

If there is any CL1 miss, first the system checks SVC2, not the CL2, for the missed block. If the missed block is found in SVC2, the block is swapped with SVC1. If the required block is not found in the SVC2, it checks CL2 for the required block. If the block is found in the CL2, it copies in CL1. If the CL1 is totally full, the victim blocks goes to the very bottom blocks of SVC2. The whole process is showed on figure 3.2. Then the SVC2 is sorted in parallel with CL1 during normal operation. If the required block is not in CL2, a check on the main memory with a stream buffering is done. During the stream buffering the required block with the additional blocks come from the main memory to CL2 and copies to CL1. Additional blocks are sent to the bottom of the SVC (SBB) for future instruction. During the copy to CL1, if the CL1 is

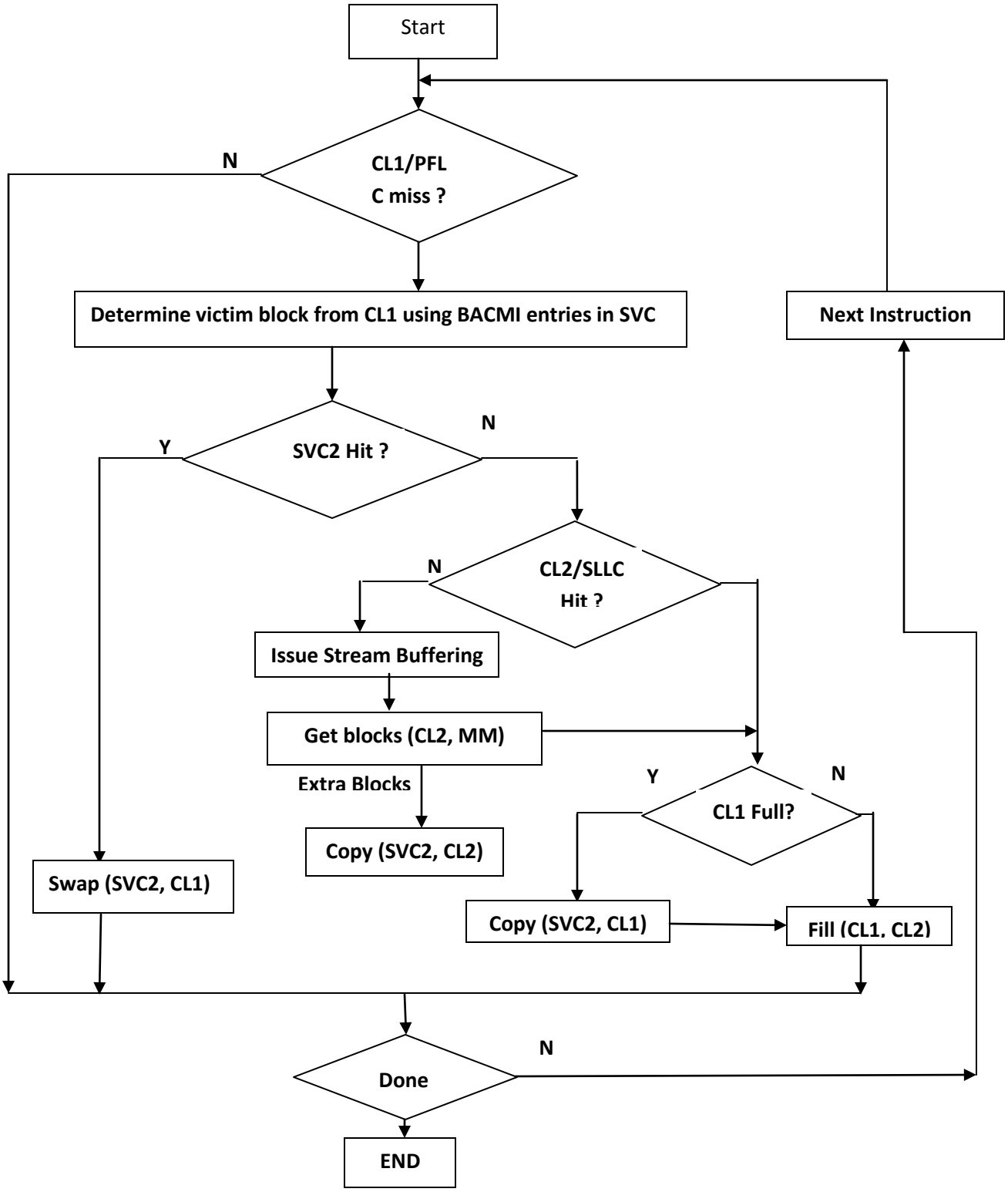

Figure 3.2: Flow diagram of proposed SVC

full, the victim blocks are sent to bottom of SVC2 and the SVC2 is sorted in parallel with CL1. This process of searching repeats until the process is finished.

3.2 Proposed Solution for Security Improvement

Cache side channel attack and cache interference are the big concern for cache design. Cache side channel attack is where important information is exploited through cache properties like power, sound vibration, time variation, heat produce etc. Cache interference is where a process interfere other process and evict cache lines form the victim process. Both types of cache attacks can exploits full secret key and needs less time and minimum power consumption. Also these types of attacks are very easy to implement, even without having any special types of equipment any remote user can launch these types of attacks on the cache system.

Many hardware methods have been proposed so far to prevent these types of cache attacks. Partitioned cache and Cache line locking are the most popular hardware methods invented to prevent cache attacks. But both of the methods cause cache underutilization problem. To prevent cache underutilization problem, cache randomization mapping has been introduces, which solves the cache underutilization problem.

In this paper, we want to show, a complete cache architecture which consumes low power consumption, provides greater security as well as high performance. We also want to implement dynamic memory-to-cache mapping as previous method, where memory blocks are mapped to any random cache lines during run time.

First we want to see how the connection established between a sender and a receiver and how the attacker gets the private key. Receiver generates a pair of public and private key for the connection. Public key is easily accessible and known to everyone including the attacker. But the

private key is secret and it is only accessible by the receiver, even the sender does not know about the private key. After generating the both public and private keys, the receiver shares the only public key with the sender and request for data. The sender getting the request, first check the authentication whether the request is from the correct receiver or not. For the authentication, the sender checks the certificate which is attached with the public key of the receiver. After checking the authentication the sender encrypts the data with the public key and sends the data to the receiver. Receiver getting the encrypted data from the sender, decrypts the data with its own private key. The attacker keeps looking at the receiver's machine for the private key. The private key is kept in the receiver's cache. So the attacker tries to attack the receiver's cache to get the private key.

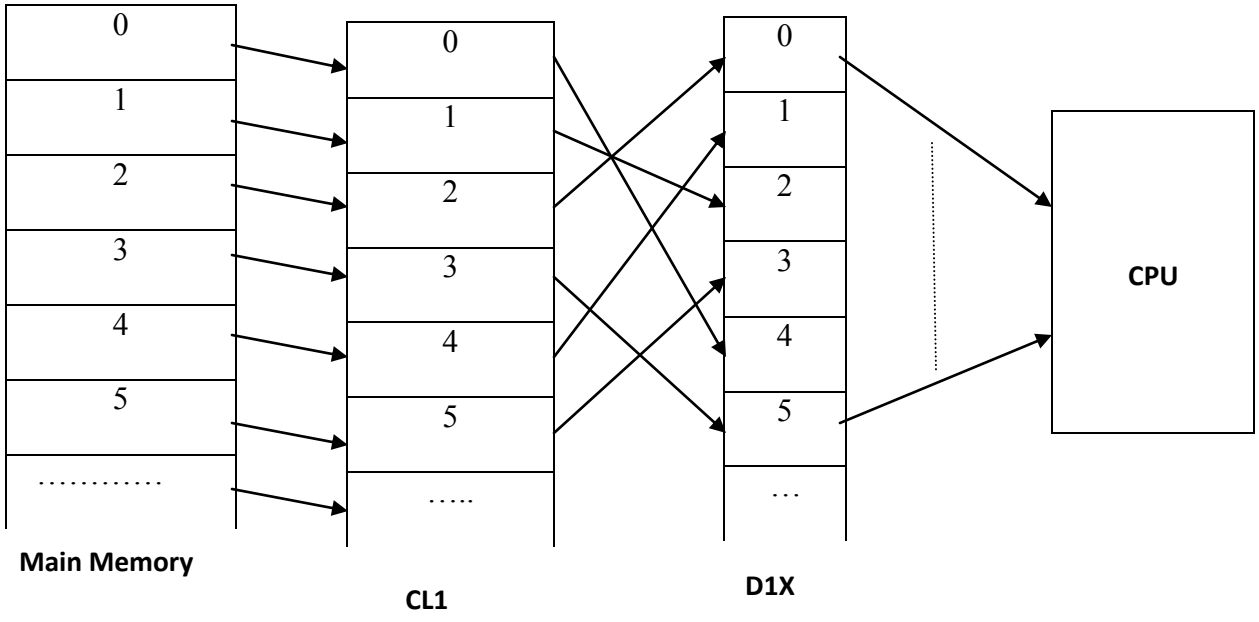

Figure 3.3: Randomized cache mapping between CL1 and D1X

As we know that CL1 is combination of data cache D1 and instruction cache I1, for our proposed solution for security improvement, we only concentrate on data cache D1 of CL1. I1

only holds the instruction and D1 hold the data from where important information can be exploited, so D1 is the place to concentrate to prevent information leaked from the cache. In our proposed solution we physically implement a new extra cache between CL1 and CPU, called D1X and it is used to hold the D1 of CL1 only. The mapping between CL1 and D1X will be totally randomized so that the attacker cannot understand or determine the actual place of any block in CL1. The figure 3.3 shows our proposed architecture to improve hardware level security. In the figure D1X is the extra new cache added between CL1 and CPU to make the attacker fool. CL1 is divided into two part- instruction cache, I1 and data cache, D1. D1X holds the data cache, D1 only from CL1. The mapping between CL1 and D1X is randomized. For example, block 0 of CL1 is mapped to block 4 of D1X, block 1 of CL1 is mapped to block 2 of D1X. The attacker can only see D1X blocks during a hardware attack. This configuration of cache makes the attacker fool as the attacker cannot assume the position of the blocks of CL1 from D1X.

The physical implementing of additional cache, D1X, improves the security. But at the same time, it consumes more power to start and run an extra cache. Also cache size becomes bigger and need more time to access and randomize the data. So cache performance and power consumption become the big concerns for implementing new extra hardware, D1X, though it improves hardware security. Approximately it consumes 17% more power than not using D1X and also it needs more time to process data from CL1 to D1X and randomize those data to make the attacker fool. So a new modification is done on the proposed solution of hardware security improvement.

In the new modified proposed solution, the extra new cache hardware is totally eliminated from the cache organization as it consumes more power and takes more access time.

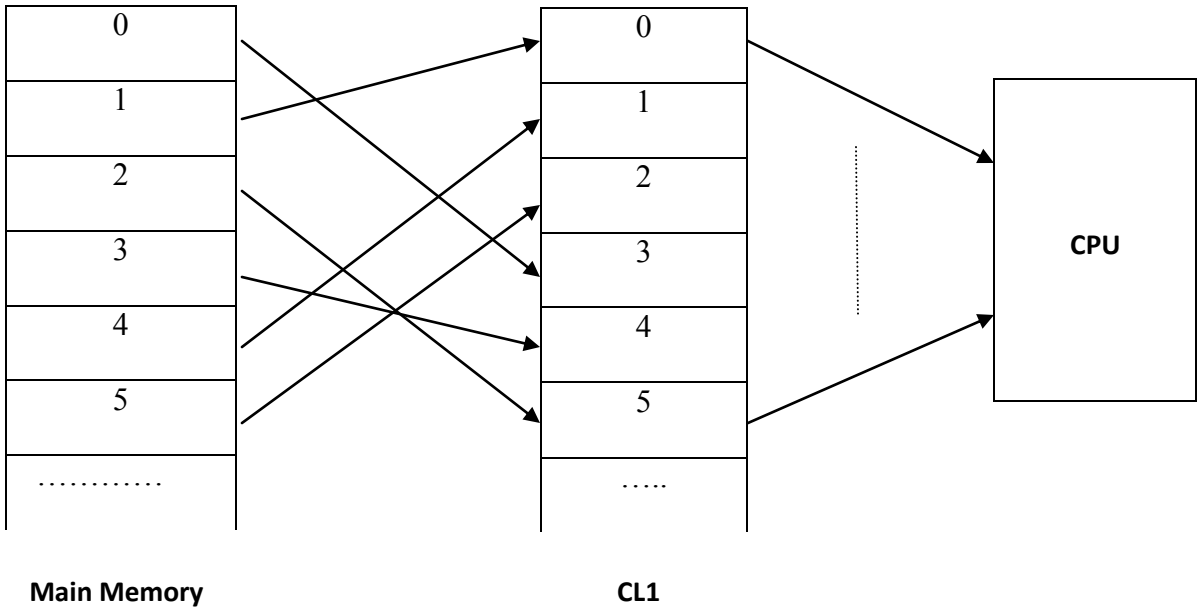

Figure 3.4: Randomized cache mapping between main memory and CL1

The cache organization of the modified solution is same as normal cache organization in the hardware system. But the cache mapping between the main memory and CL1 becomes randomized as opposite to the direct cache mapping in the simple cache architecture. For example, block 0 of main memory is mapped to block 2 of CL1. So the attacker when try to exploit the important information from CL1, the attacker can only see the randomized position of the block of CL1, but cannot assume the actual position of the real block in the main memory.

Figure 3.4 shows the modified solution for improving hardware level security. The mapping between main memory and CL1 is totally randomized. Block 0 of main memory is randomly mapped to the block 3 of CL1, then block 1 of main memory is randomly mapped to block 0 of CL1 and so on. So when the attacker deploys attack on CL1, the attacker can only see the block position in the CL1, but cannot assume the block position on the main memory because of the randomized mapping between main memory and CL1.

This modified solution of hardware security improvement solves the performance and power issue of the first proposed solution as the extra cache hardware, CL1X, is totally eliminated from the cache architecture. The architecture of the modified solution is same the architecture of normal cache. But the randomized mapping between main memory and CL1 makes the attacker hard to predict the actual position of the block in the main memory. So it consumes less power and takes less access time compared to previous proposed solution as well as it improves the hardware security.

CHAPTER 4

Simulation Analysis

This chapter describes the assumptions and simulation architecture for the proposed architecture to evaluate the result of power consumption, performance of proposed multicore architecture. The result of the proposed multicore architecture is plotted as graphs and analyzed efficiently. The quad-core system is simulated with and without the proposed SVC. Simulation is run with or without CL2/SLLC cache locking.

4.1 Assumptions

For the proposed architecture we follow some important assumptions:

- SVC can be enabled and disabled based on the need of operation. If the SVC is disabled, the system operates like regular quad core system which has two levels of caches (CL1 and CL2).
- CL2 cache locking is implemented when SVC is disabled. This result of CL2 cache locking while SVC is disabled, is compared with the result without cache locking while SVC is enabled.
- All the cores of the system equally share the SVC.
- To select the victim cache for replacement, well known Least Recent Unit (LRU) replacement algorithm is used.
- For update policy, Write-Back is used.
- CL2 is connected to the main memory by a bus where the latency of the bus is 12 times longer than the latency of any bus that connects a core to the CL2.

- Due to the change of cache size, line size and associativity, the change of cache hit or miss times are considered negligible.
- Each core of the proposed quad core architecture operates its task with the help of its own private CL1 which is divided into I1 and D1 and shared unified CL2. The simulation program that we want to run is single threaded. Only one single application which is running on only one single core is simulated by the simulator at a time. During this time, the cores keep idle. Most of the cache effects disappear if the threads have very less similarity and if the data and the code are placed properly in the cache memory as this prevents cache conflict miss [28]. Also we assume that multiple cache effects are negligible and should not effect on our results of simulation with proper code allocation to I1 and CL2.

4.2 Simulation Architecture

We took Intel like quad core system where SVC is placed between CL1 and CL2. CL1 is private to each core, but CL2 is unified and equally shared to all four cores of the system. We also assumed that the SVC can be enabled and disabled based on the need of the operation. But if the SVC is enabled it should synchronize with the correct timing with both CL1 and CL2. If SVC is disabled, still the system should act same as regular quad core system. Figure 4.1 shows the simulated quad core system architecture where the dashed lines indicate the disabled or enabled characteristic of the SVC. CL1 is divided into two parts – instruction cache (I1) and data cache (D1) and SVC is connected to each I1 and D1 of the four cores of the system. SVC is connected to CL2 or shared last level cache (SLLC). CL2 or SLLC is connected to the main memory.

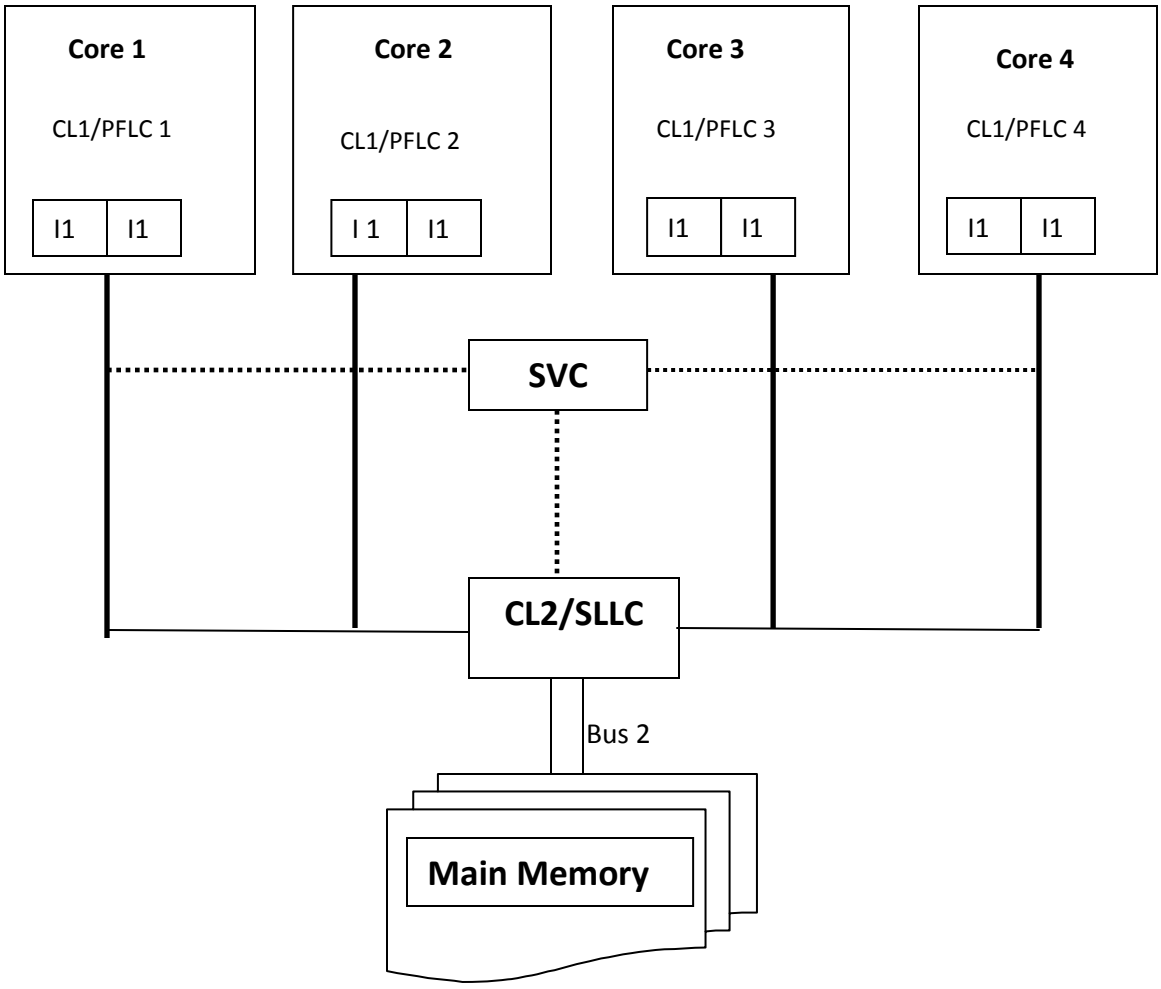

Figure 4.1: Quad-core system with SVC between CL1/PFLC and CL2/SLLC

4.3 Workload for the Proposed Architecture

Different well known workloads have been used for this work. Some of those are Moving Picture Experts Group's MPEG-4 decoding, Advanced Video Coding which is mostly known by H.264/AVC decoding, Matrix Inversion (MI) and Fast Fourier Transform (FFT) etc. Important parameters of the workload are shown in Table II. The code size for MPEG-4, H.264/AVC, MI and FFT are 93.04 KB, 78.85 KB, 1.47 KB and 2.34 KB respectively.

Table II. Important characteristics of MPEG-4 decoding, H.264/AVC decoding, FFT, MI, and DFT Codes

Applications	Code Size (KB)	Number of Instructions
MPEG-4	210	52,071,180
H.264/AVC	186	40,922,256
FFT	2	365,184
MI	1	227,518

4.4 Input and Output Parameters

Different input and output parameters are considered for the proposed quad core architecture with SVC.

Input parameters of the proposed architecture are considered as follows:

Number of cores of the system = 4

Size of SVC = 2, 4, 8, 16, 32 KB

Size of I1/D1 of CL1 = 8/8, 16/16, 32/32, 64/64, 128/128 KB

Size of CL2 = 256, 512, 1024, 2048, 4096 KB

Size of line = 16, 32, 64, 128, 256 Byte

Level of associativity = 1-, 2-, 4-, 8-, 16- way

The output parameters of the proposed architecture are average latency per task and the total power consumption of the system. Latency can be defined as the time difference between

the starting of any task and ending of the particular task. Latency can be considered as delay in another word.

The delay penalty for cache misses and other parameters are considered as follows for the proposed architecture:

Number of cycle needs to satisfy any instruction inside the CPU (at ALU) = 1

Number of cycle needs to satisfy any instruction inside the CPU (at private CL1) = 3

Number of cycle needs to satisfy instruction outside the CPU (at shared CL2) = 10

Number of cycle needs to load and store any operation = 100

Number of cycle needs to branch operation = 150

Power consumption is also considered as an output parameter for the proposed quad core system. Power analysis is very important for today's world. Low power consumption is desired from every device. Normally there are three states for power analysis for any device – active, idle and sleep. Active state is considered when the system is operating and consuming the maximum amount of power to turn on and operate in a regular manner. Sleep is the state when the system is considered as turn off and consumes no power. Idle is the state in between active and sleep. This is the state when the system is just turn on but not operating regularly and so consumes less amount of power than active state.

Some of the components of the proposed quad core processor are considered as follows for power consumption [29, 30]:

Power consumption by CPU = 3.6 mWatts/operation

Power consumption by I1 = 2.7 mWatts/operation

Power consumption by other components = 2.1 mWatts/operation

The power consumption of CL2 and main memory of the proposed quad core system is assumed to be proportionate to the size and cache hits.

CHAPTER 5

Results and Discussion

This chapter describes important simulation results of our proposed quad core architecture. We also showed our result of the proposed architecture by some graph to show how the victim cache reduces the average memory latency and total power consumption of the system. Our proposed system totally eliminates the locked mechanism for the multicore system. Results for H.264/AVC and MI are not shown during showing the result as the result for H.264/AVC behaves similar as MPEG-4 and MI shows similar result as FFT.

5.1 Impact of SVC Size

SVC size is an important factor to consider. Average memory latency and total power consumption of the system greatly depend on the size of the proposed SVC of the multicore system. From the table I, it is clear that as bigger as the SVC size, more BACMI entries and more victim blocks can be hold by the multicore architecture. So, the average memory latency and total power consumption of the multicore system also decreases with the increased size of SVC. Figure 5.1 and Figure 5.2 shows the impact of SVC size on average memory latency and total power consumption of the multicore system.

Figure 5.1 shows clearly how the size of SVC (2 KB to 32 KB) effect on the average memory latency per task for MPEG-4 and FFT. Also the task is done with both locking and without locking condition to see the difference in average latency. For MPEG-4, it shows improvement of average memory latency with cache locking even if not using any SVC. But the great improvement of the average memory latency can be seen for MPEG-4 with SVC and no

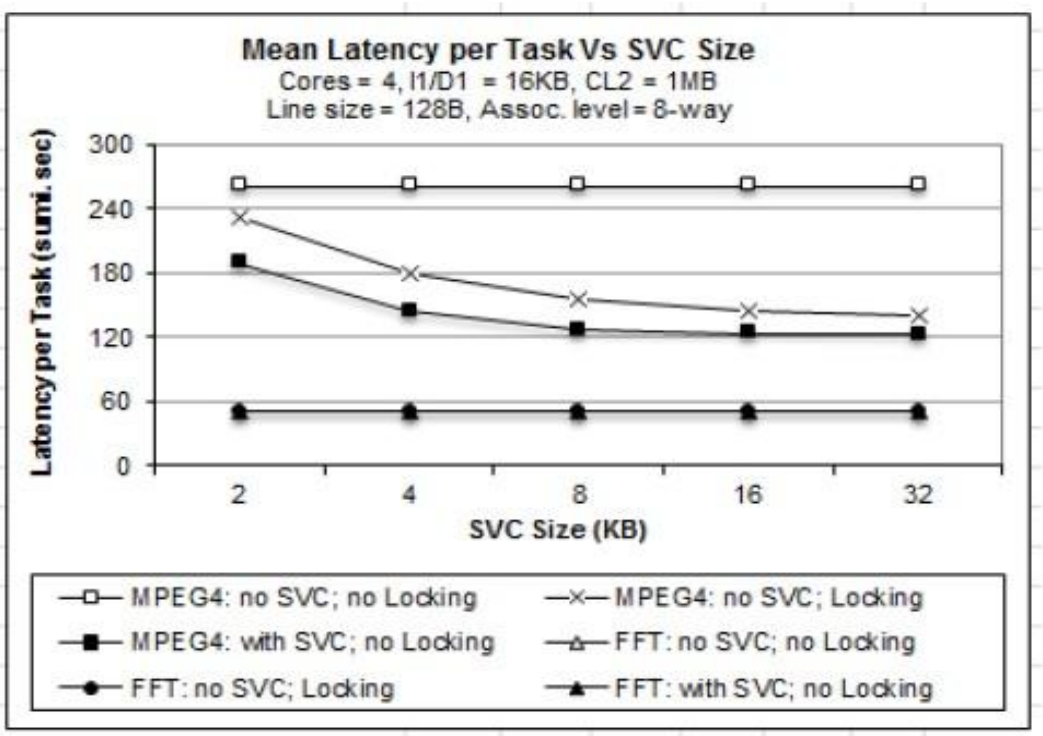

Figure 5.1: Impact of SVC size on memory latency

cache locking. Also it is observed that with the increase of SVC size (from 2 KB to 4 KB), the average memory latency per task decreases. But with the size larger than 8 KB, the decrease in average memory latency per task is not so significant for MEPG-4. For FFT, not any significant impact on average memory latency per task has been observed for SVC and no cache locking.

Figure 5.2 shows the effect of SVC size on total power consumption of the muticore system. For MPEG-4, it is observed that the total power decreases with the increase of SVC size (from 2 KB to 4 KB). But the total power decrease is not very significant after increasing the SVC size larger than 8 KB. Also from the figure it is observed that, for MPEG-4, the total power consumption of the multicore system is the highest during no locking and no SVC. The total power consumption decreases when using cache locking (without SVC). But a significant

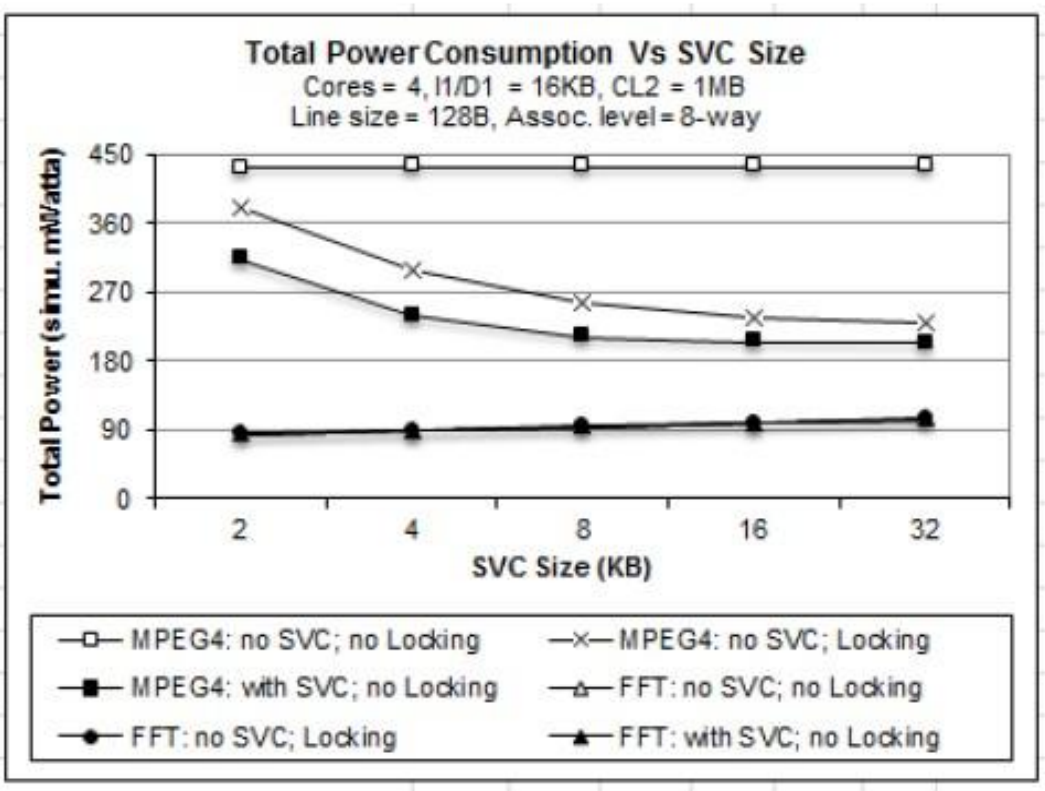

Figure 5.2: Impact of SVC on total power consumption

decrease in total power consumption is seen using SVC and no cache locking. For FFT, the impact of increasing SVC size on total power consumption is negligible.

5.2 Impact of SVC and CL1/PFLC Size

The impact of SVC and CL1/PFLC size on average memory latency and total power consumption of the multicore system is shown in figure 5.3 and figure 5.4. I1 and D1 are both considered for CL1/PFLC size. These two figure also reveals the impact with cache locking and without cache locking. Figure 5.3 shows how with the increase of CL1/PFLC cache size from 8KB to 128 KB effects on average memory latency for MPEG-4 and FFT. For MPEG-4, the impact of CL1 size from 8 KB to 16 KB is very significant, where from 16 KB to 32 KB is somewhat significant and from larger than 32 KB the impact is negligible. Also from the figure

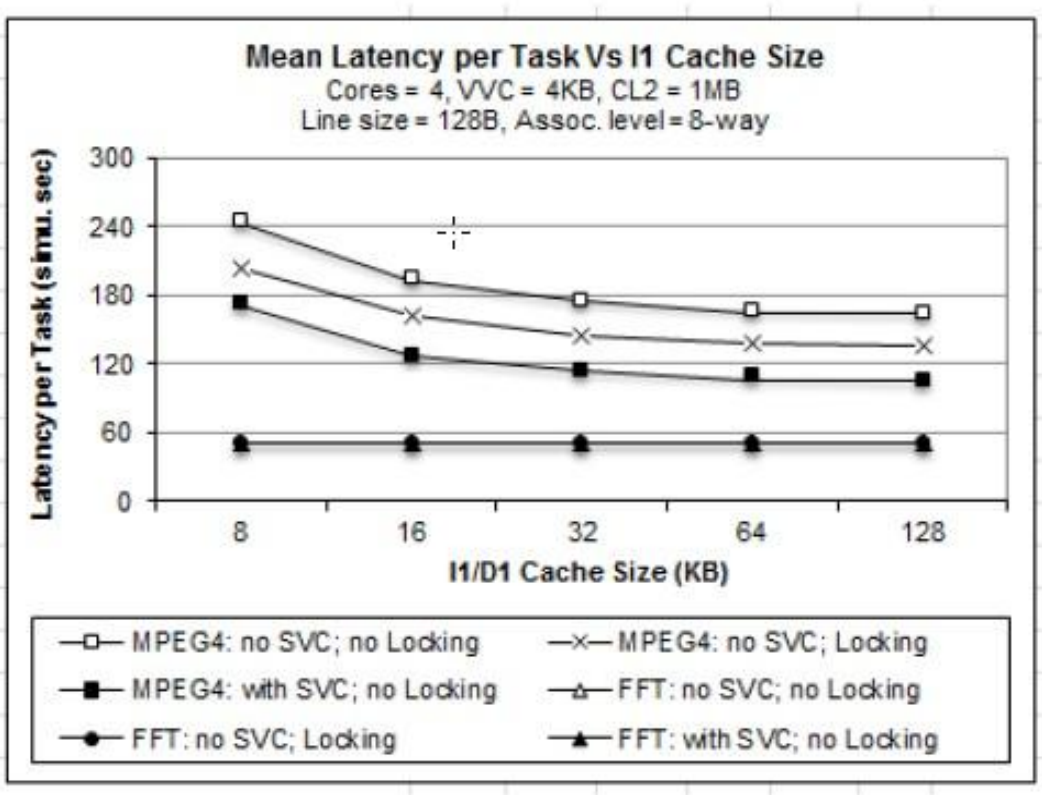

Figure 5.3: Impact of SVC and CL1/PFLC on memory latency

5.3, it reveals that, the average memory latency per task is the lowest for using SVC and no cache locking. But the average memory latency per task is the highest during using no SVC, and no locking. For FFT, the CL1/PFLC cache size increase does not effect on average memory latency per task.

Figure 5.4 illustrates the impact of CL1/PFLC cache size increase on total power consumption of the multicore system. From the figure, we can easily understand that the total power consumption decreases with the increase of CL1 cache size (from 8 KB to 128 KB) for MPEG-4. Also it is clearly understand from the figure that the total power consumption is less for while using SVC with no locking than without using SVC. For FFT, total power consumption increases with the increases the size of CL1/PFLC. For FFT, as the size of cache CL1 increases,

it starts consuming some power. So for FFT, the power consumption slightly increases for increasing the size of CL1 cache.

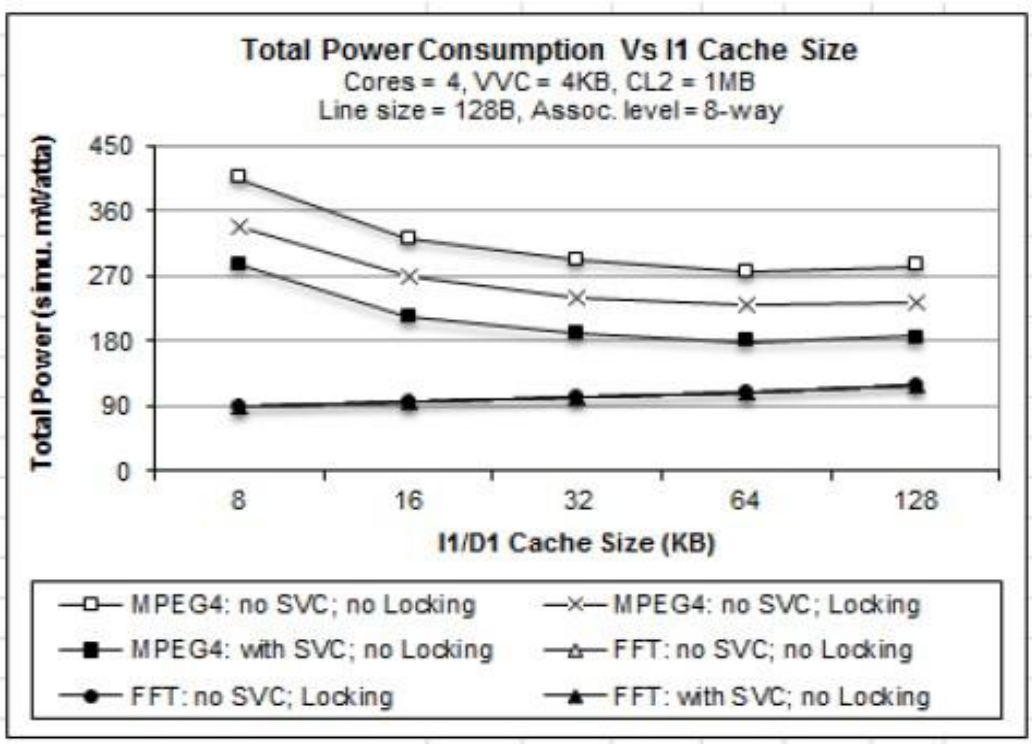

Figure 5.4: Impact of SVC and CL1/PFLC size on total power consumption

5.3 Impact of SVC and Line Size

For this experiment, same line size for CL1/PFLC, SVC and CL2/SLLC is used for each of the test. Figure 5.5 and figure 5.6 shows the impact of line size from 16 B to 256 B on average memory latency and total power consumption of the multicore system. For MPEG-4, with the increase of line size from 16 B to 64 B, the average memory latency decreases. But for the line size larger than 64 B, the average memory latency difference is negligible for MPEG-4. Because of the cache pollution effect the average memory latency difference is negligible for the size

greater than 64 B. Also from the figure, it is clear that, the average memory latency is highest during no SVC and no cache locking. The average memory latency becomes less during cache locking. But the average memory latency become minimum compared with the previous two

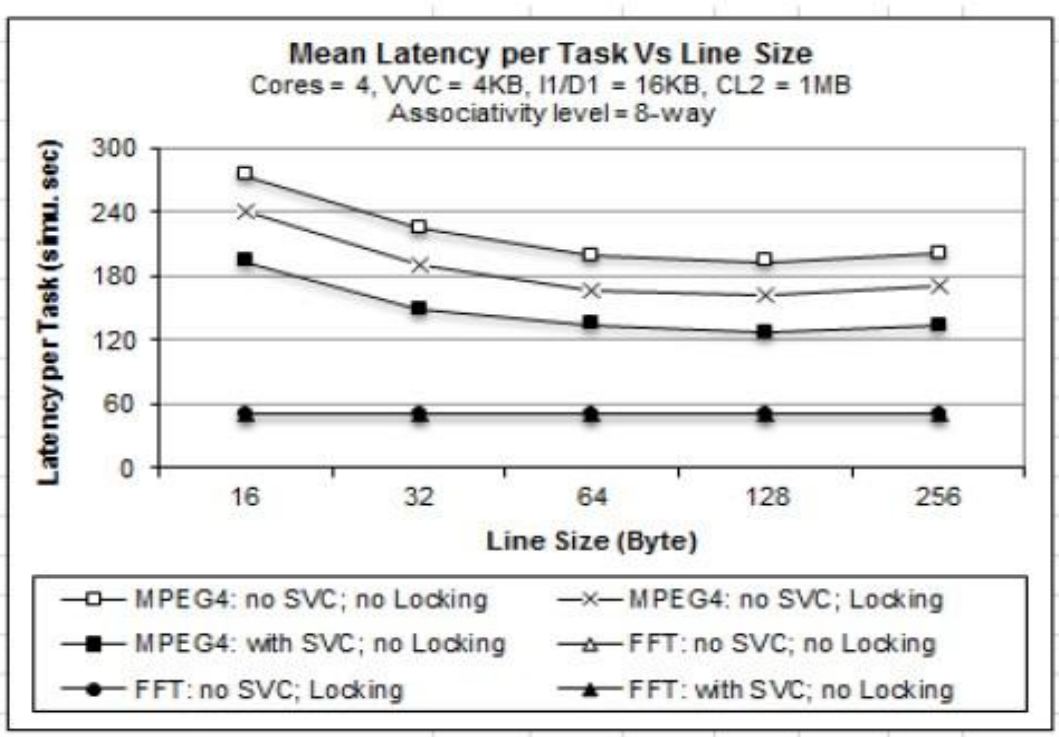

Figure 5.5: Impact of SVC and line size on memory latency

cases, while using SVC but no cache locking. So, it is clear that the average memory latency decreases with using SVC than CL2 cache locking. For FFT, using SVC or using greater line size do not affect any difference for average memory latency per task.

From figure 5.6, the total power consumption with using SVC and without using SVC and cache locking is shown for MPEG-4 and FFT. Total power consumption of the multicore system decreases with the line size increases for MPEG-4. The decrease of the total power consumption is more effective for the cache size from 16 B to 64 B. For cache size larger than 64

B, the total power consumption difference is negligible for MPEG-4. For FFT, the total power consumption remains stable for both cases whether using SVC or without using SVC, but cache

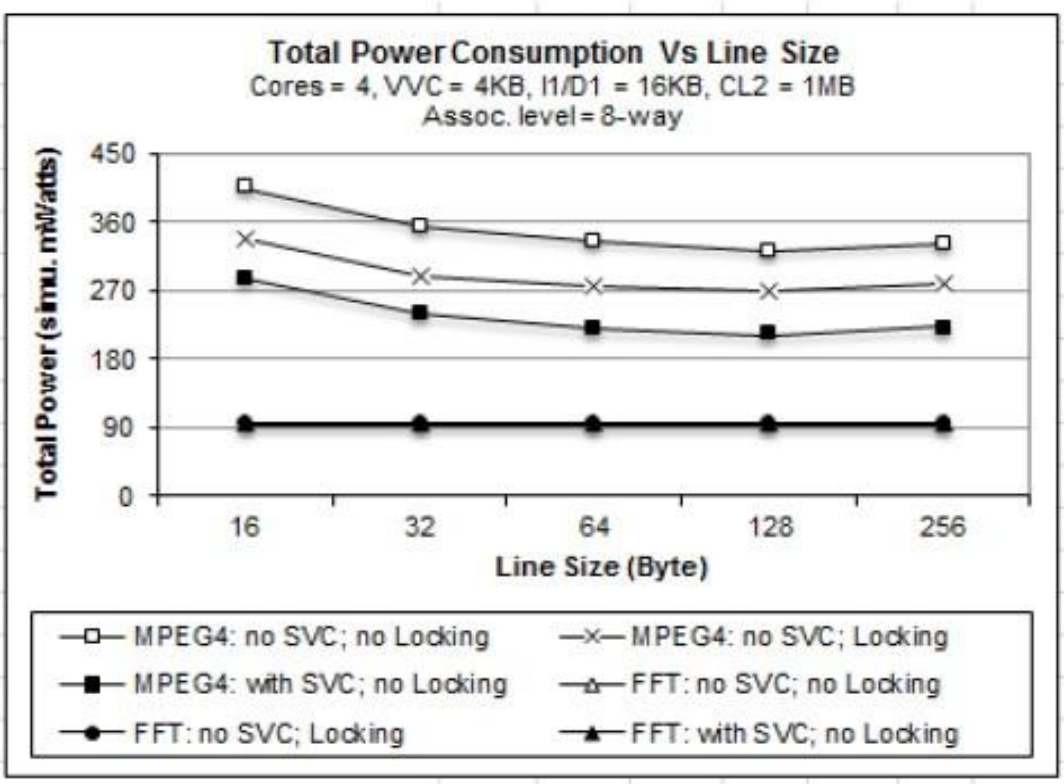

Figure 5.6: Impact of SVC and line size on total power consumption

locking.

5.4 Impact of SVC and Associativity Level

For this experiment, same associativity level for CL1/PFLC, SVC nad CL2/SLLC has been used for each of the test. Figure 5.7 and figure 5.8 shows the impact of SVC and associativity level on average memory latency and total power consumption of the multicore system. In figure 5.7, the impact of associativity, from 1-way direct mapped to 16-way set associative mapped, using SVC and without using SVC for MPEG-4 and FFT on average

memory latency per task is shown. From the figure, it is clear that, the average memory latency per task for MPEG-4, decreases with increases of associativity from 1-way direct mapped to 4-

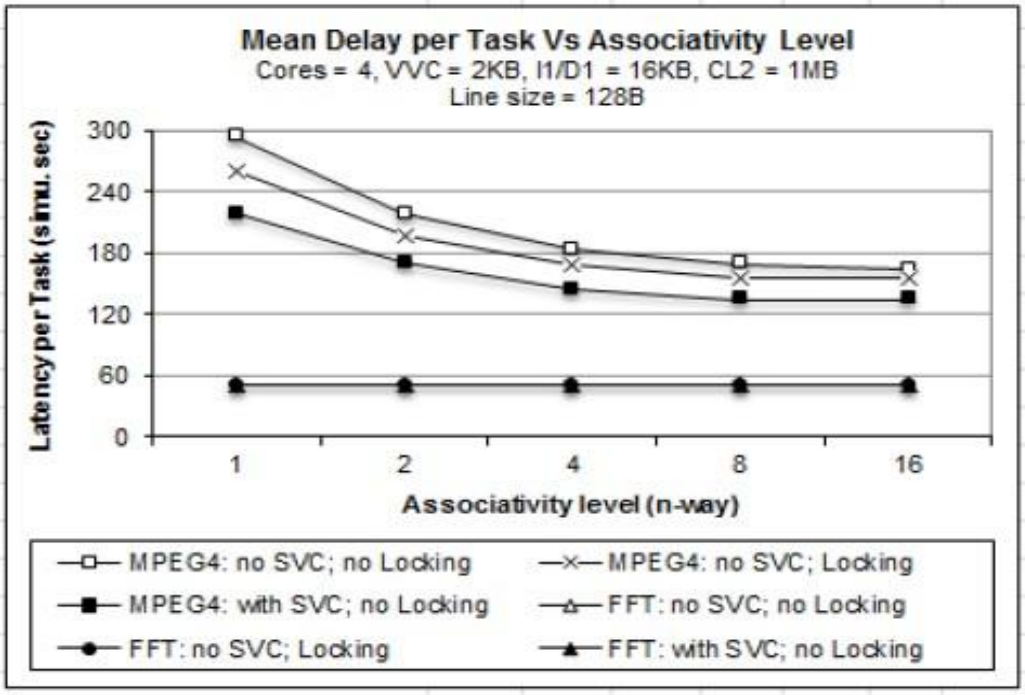

Figure 5.7: Impact of SVC and associativity level on memory latency

way set associative. After 4-way set associative mapped, the impact on average memory latency per task for MPEG-4 is not so significant. Also the figure shows that, average memory latency is highest during no SVC and no cache locking. But the average memory latency is lowest with using SVC, but no cache locking. For FFT, associativity level incensement or using SVC does not effect on the average memory latency per task for the multicore system.

Figure 5.8, illustrate the effect of associativity level and SVC on total power consumption for the multicore system architecture. For MPEG-4, the total power consumption decreases with the increasing associativity level from 1-way direct mapped to 4-way set

associative mapped. After 4-way set associative mapped, with the increasing level of associativity, the impact on total

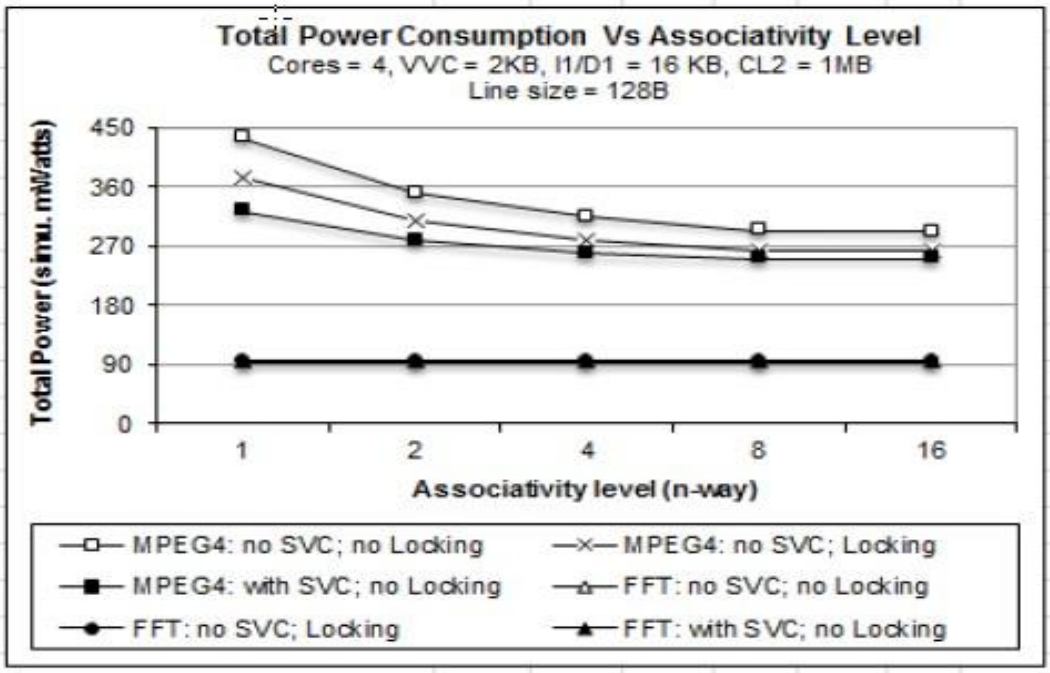

Figure 5.8: Impact of SVC and associativity level on total power consumption

power consumption is not so significant. Also, it is clear that, the total power consumption is much less during using SVC for MPEG-4, than without using SVC or only cache locking. For FFT, the impact on total power consumption is stable even increasing the associativty level or using SVC.

5.5 Impact of SVC and CL2/SLLC Size

The size of the CL2/SLLC of the multicore system can effect on average memory latency or total power consumption of the system, based on the size of the application and its characteristics. Figure 5.9 and figure 5.10 shows the impact of CL2/SLLC size from 256 KB to 4 MB on average memory latency and total power consumption for MPEG-4 and FFT. Form the

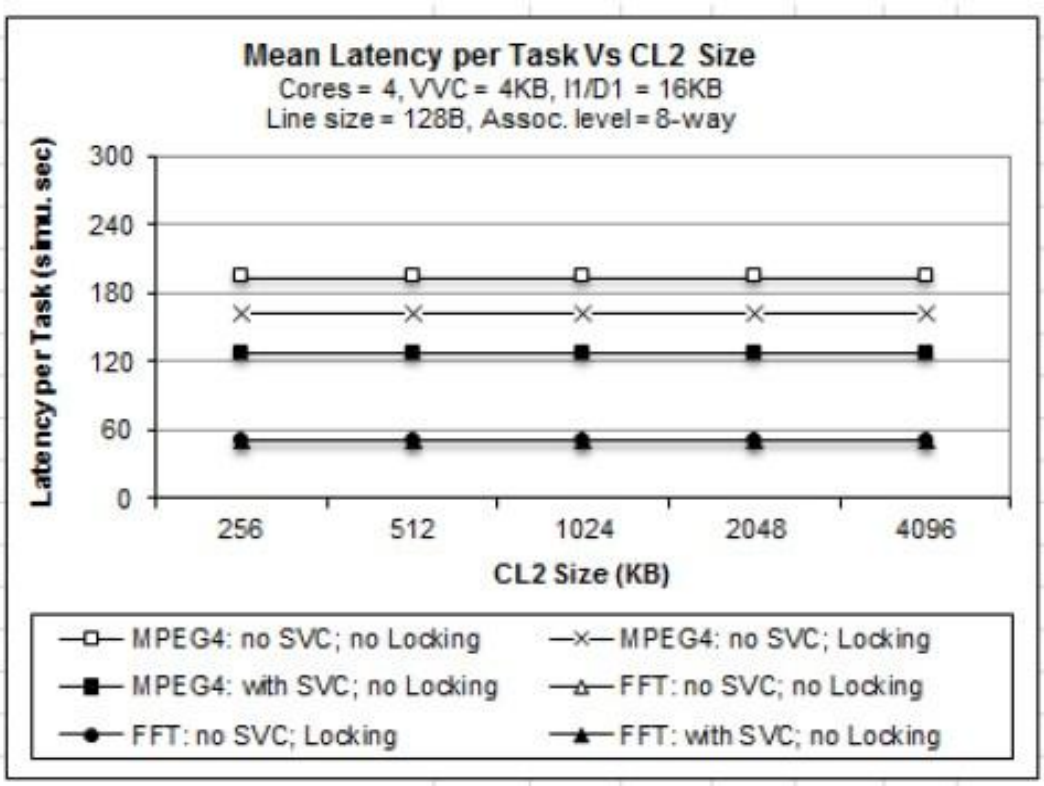

Figure 5.9: Impact of SVC and CL2/SLLC Size on memory latency

figure 5.9, it is clear that, with the there is no impact on average memory latency for MPEG-4 while increasing the size of CL2. But the figure also illustrates that the average memory latency decreases using SVC than without using SVC or using only cache locking. For FFT, the average memory latency has no effect by using SVC or without using SVC. Also for FFT, there is no difference on average memory latency for increasing CL2 size.

Figure 5.10 illustrates the impact on total power consumption for different CL2 size and using SVC. Total power consumption for MPEG-4 decreases significantly while increasing the size of CL2 larger than 2 MB. The decrease in total power consumption is less or negligible for the increasing CL2 size from 256 B to 2 MB. Also we can see that, using SVC, the total power consumption becomes significantly less compared to without using SVC or using only cache

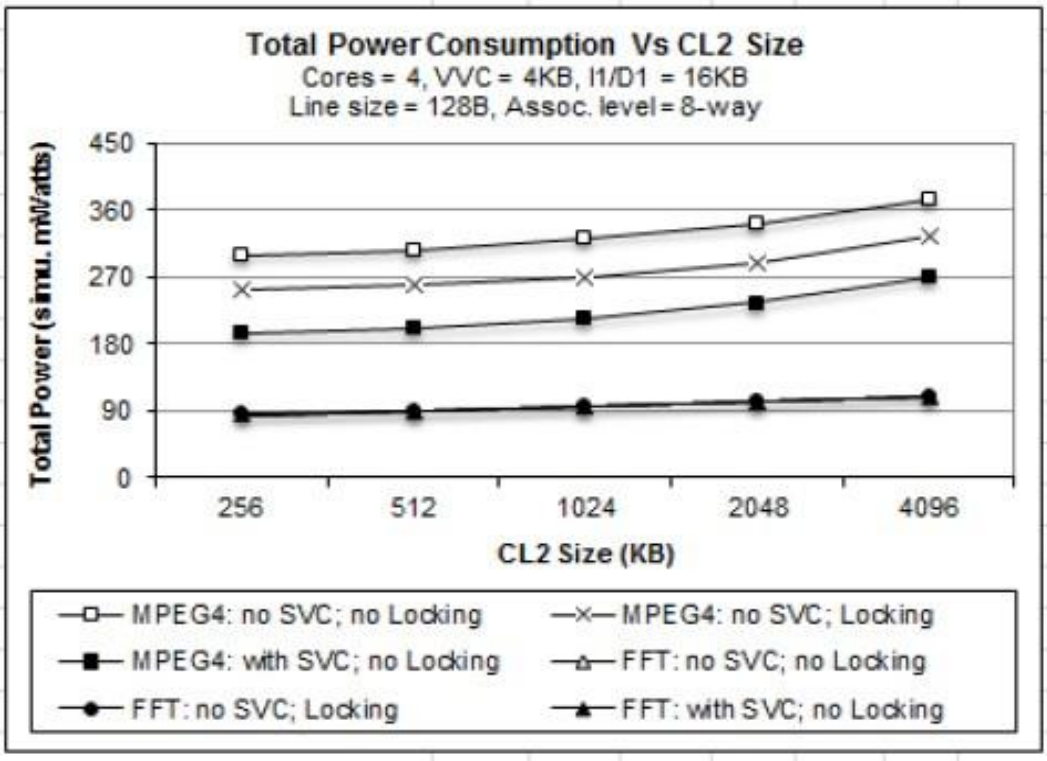

Figure 5.10: Impact of SVC and CL2/SLLC size on total power consumption

locking. For FFT, the total power consumption for the system slightly increases with the increasing cache size from 2 MB. But the total power consumption remains stable while using SVC or without SVC.

5.6 Comparison between SVC and Cache Locking

From all the result we can understand the difference between using SVC and not using SVC with cache locking. The average memory latency and total power consumption both decreases significantly with using SVC and no cache locking than implement only cache locking without SVC. Figure 5.1 to figure 5.10, each figure illustrates the difference of using SVC on average memory latency and total power consumption for the system.

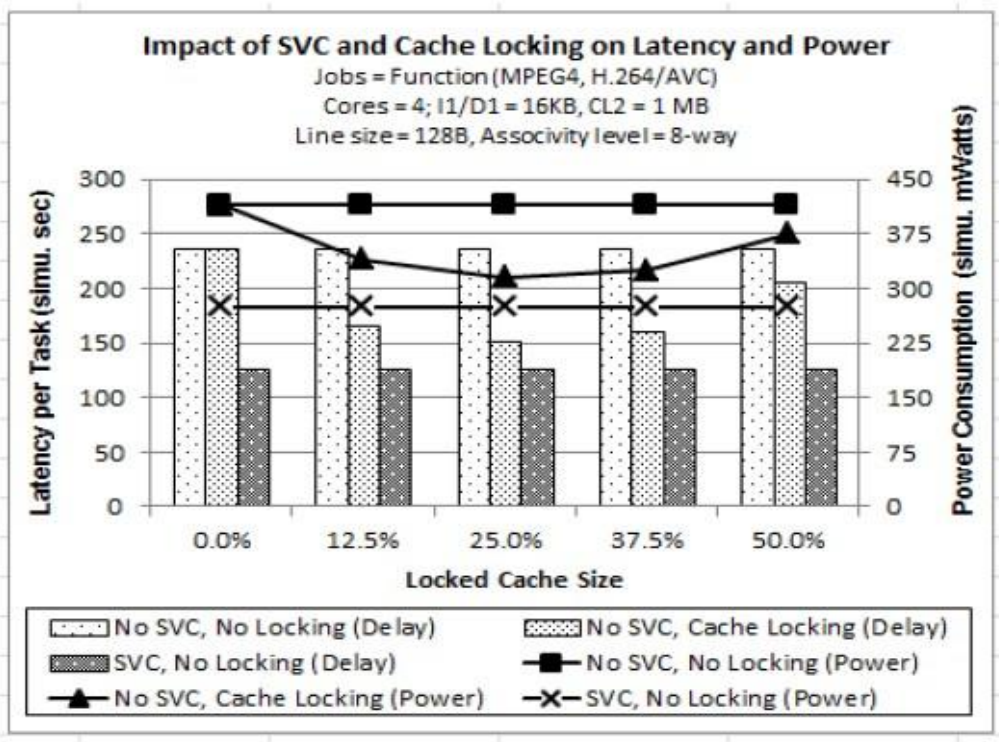

Figure 5.11: Impact of SVC and cache locking on memory latency and total power consumption

Figure 5.11 illustrates a closer look and explains the performance of using SVC for multicore system. From the figure it is clear that average memory latency and total power consumption of the system decreases while the CL2 cahe size increases from 0% for no locking to 25 % for locking. After that, when the size of the locked CL2 increases, both the average memory latency and total power consumption of the system increases. This is obvious as during higher degree of cache locking, like when the cache size of locked CL2 is more than 25%, the size of the effective cache decreases and so the capacity miss increases. The figure also reveals that, both the average memory latency and total power consumption of the system is smaller compared to the best case of cache locking (25% CL2). SVC supports stream buffering and also holds the block with maximum number of misses which makes the average memory latency and total power consumption less for the system.

5.7 Enhancement of Security

Cache side channel attack is one of the most dangerous hardware level security attacks. This attack is done through physical parameters like power, time variation, sound variation, heat production etc of the system. The first proposed solution for security enhancement is using an extra hardware, D1X, and randomly cache mapped between CL1 and D1X. But as the extra hardware consumes some power, which increases the power consumption as well as it is hard to implement the extra hardware. Not only that, the processing time also increases for implementing the extra hardware. So a new approach is proposed which totally eliminates the implementation of the extra hardware, D1X. The cache mapping between main memory and CL1 becomes totally randomized which increases the security. It also decreases the power consumption and memory access time compared to the first proposed mechanism. It is estimated that the probability of hardware level attack is reduced from 40K to 1 with the randomized cache mapping technique.

CHAPTER 6

Conclusion and Future Extensions

This chapter concludes this work and lists some future extensions to decrease average memory latency and overall power consumption and to improve system security.

6.1 Conclusion

Cache is used to solve the speed gap problem between the CPU and the main memory of the system. Several cache-memory organizations are introduced and being used to adjust the speed gap problem between the high speed CPU and low speed main memory. But using these several levels of cache introduces some serious issues on overall performance as memory latency and power consumption increases. For multicore system, it is difficult to hide the last level latency as the number of cores increases. Also the total power consumption increases as each core consumes some power in the multicore architecture. Different approaches (line locking, dynamic cache-to-memory mapping) have been taken to solve this problem. But there are some severe issues with these methods – aggressive cache locking can reduce the effective cache size and also it is difficult to select right cache lines to lock. Also security attack on cache has been a great concern now-a-days. Important information exploits from the cache easily by security attacks like cache side channel attack or cache interference. Many software approaches has been introduced to solve the security attack problem on cache. But software protection is not enough to solve the issue. Hardware level security along with software level security is important to solve the problem. So, we propose a new approach of cache design which improves performance to power ratio and system security.

We introduce a SVC (smart victim cache) between CL1 and CL2 which supports stream buffering and holds the miss blocks. The simulations run on various applications including MPEG-4 and H.264/AVC. Experimental results indicates that using SVC, the average memory latency and total power consumption of the system decreases by 17 % and 21% respectively. Also SVC totally replaces the cache locking.

For security improvement we propose a technique of totally randomizing the cache mapping between main memory and CL1. This makes the attacker fool of actual position of the block in the main memory. In the same time, as no extra hardware is used, it does not consume extra power or take extra time to process. So, randomized cache mapping between main memory and CL1 improves the hardware security significantly. It is estimated that the probability of cache side channel attack for a system with 16-block CL1 reduces from 40K to 1.

6.2 Future Extensions

Proposed smart SVC and randomized cache mapping approaches can be extended and applied into many modern computing systems including the following:

- Embedded Systems – Embedded systems suffer due to limited resources. Proposed smart SVC can be customized for embedded systems to decrease average memory latency and power consumption.
- Handheld Computers – Due to the convenience of using handheld computers, its popularity is skyrocketing. Most usages of these devices now-a-days include internet and social-networks, which makes them easy-to-attack targets. Proposed hardware approach can be used to enhance the security of these devices.

REFERENCES

LIST OF REFERENCES

[1] Alted, F. “Why Modern CPUs Are Starving and What Can Be Done about It” Computing in Science & Engineering, Vol. 12, Issue. 2. pp. 68-71, 2010. DOI: 10.1109/MCSE.2010.51

[2] Carvalho C. “The Gap between Processor and Memory Speeds” Proc. Of IEEE International Conference on Control and Automation, 2002

[3] Anand K. and Barua R. “Instruction cache locking inside a binary rewriter” CASES ’09 Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems. pp. 185-194. DOI: 10.1145/1629395.1629422

[4] Khan M.S., Jim`enez A.D., Burger D., Falsafi B. “Using dead blocks as a virtual victim cache” PACT’10 Proceedings of the 19th international conference on Parallel architectures and compilation techniques. pp. 489-500, 2010 DOI: 10.1145/1854273.1854333

[5] Wong F. W., Koh K.C, Chen Y., Li H. “VOSCH: Voltage scaled cache hierarchies” 25th International Conference on Computer Design, 2007. pp. 496-503. DOI: 10.1109/ICCD.2007.4601944

[6] Zhang C., Vahid F., Najjar W. “A highly configurable cache for low energy embedded systems” ACM Transactions on Embedded Computing System (TECS). Vol. 4, Issue 2. pp. 363-387, 2005. DOI: 10.1145/1067915.1067921

[7] Z. Wang and R.B. Lee, “Covert and Side Channels due to Processor Architecture” Proceedings of the 22nd Annual Computer Applications Conference (ACSAC’06). Vol. 23, Issue 6. pp. 473-482, 2006. DOI: 10.1109/MM.2003.1261386

[8] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in Proceedings of the 34th annual international symposium on Computer Architecture, San Diego, California, USA, 2007, pp. 494-505

[9] A. One, "Smashing the stack for fun and profit," in Phrack vol. 7, ed, 1996

[10] C. Cowan, et al., "Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade," in DARPA Information Survivability Conference and Exposition, 2000

[11] D. Page, “Partitioned Cache Architecture as a Side-Channel Defense Mechanism” Cryptology ePrint Archive, Report 2005/280, 2005

[12] Aciicmez O. “Yet another Micro Architectural Attack: exploiting I-Cache” CSAW ’07 Proceedings of the 2007 ACM workshop on Computer security architecture, pp. 11-18. DOI: 10.1145/1314466.1314469

[13] Kocher C. P. “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems” Springer Berlin Heidelberg. pp. 104-113, 1996. DOI: 10.1007/3-540-68697-5_9

[14] P. Kocher, et al., "Differential Power Analysis," in Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, 1999

[15] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," presented at the Advances in Cryptology - CRYPTO '96, 1996

[16] Mueller F. “Compiler support for software-based cache partitioning” LCTES ’95 Proceedings of the ACM SIGPLAN 1995 workshop on Languages, compilers, & tools for real-time systems. pp. 125-133. DOI: 10.1145/216636.216677

[17] Z. Wang and R. B. Lee, "A novel cache architecture with enhanced performance and security," in Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture, 2008, pp. 83-93

[18] Zhou Y. and Feng D. “Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing” IACR Cryptology ePrint Archive 2005

[19] S. P. Skorobogatov and R. J. Anderson, "Optical Fault Induction Attacks," presented at the Cryptographic Hardware and Embedded Systems - CHES 2002, 2003

[20] Jouppi NP. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. Western Research Laboratory (WRL), Digital Equipment Corporation, URL: https://www.cis.upenn.edu/~cis501/papers/jouppi-victim.pdf. 1990

[21] Kozyrakis C and Patterson D. Vector Vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks. In the Proceedings of the 35th International Symposium on Microarchitecture, Instabul, Turkey, 2002

[22] Flynn MJ and Rudd KW. Parallel Architectures. In ACM Computing Surveys, Vol. 28, No. 1, pp. 67-70, 1996

[23] Asaduzzaman A, Rani M, and Sibai FN. On the Design of Low-Power Cache Memories for Homogeneous Multi-Core Processors. In IEEE Int’l Conference on Microelectronics (ICM'10), Cairo, Egypt, 2009

[24] Chakraborty K and Roy S. Topologically homogeneous power-performance heterogeneous multicore systems. Design, Automation & Test in Europe Conference & Exhibition (DATE'11), pp. 1-6, 2011

[25] Fruehe J. Planning Considerations for Multicore Processor Technology. Dell Power Solutions. URL: http://web.cefriel.it/~brandole/pc/14-dell-multicore.pdf. 2005

[26] Knapp E. Growing Popularity of Tablets Hits PC Market. Wall St. Cheat Sheet. URL: http://wallstcheatsheet.com/breaking-news/growing-popularity-of-tablets-hits-pc-market.html/. 2011

[27] NVIDIA Corporation. The Benefits of Multiple CPU Cores in Mobile Devices. Whitepaper. URL: http://www.nvidia.com/content/PDF/ tegra_white_papers/Benefits-of-Multi-core-CPUs-in-Mobile-Devices_Ver1.2.pdf. 2010

[28] Mezetti E, Holsti H, Colin A, Bernat G, and Vardanega T. Attacking the Sources of Unpredictability in the Instruction Cache Behavior. In Proc. 16th Int'l Conference on Real-Time and Network Systems (RTNS'08), pp. 151–160, 2008

[29] Tang W, Gupta R, and Nicolau A. Power Savings in Embedded Processors through Decode Filter Cache. In Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE'02), pp. 1-6, 2002

[30] Balfour J, Dally WJ, et al. "An Energy-Efficient Processor Architecture for Embedded Systems". In IEEE Computer Architecture Letters, Vol. 7, No. 1. 2008

[31] Fang S, Lin K, Tsai J, and Yu Q. Final Project. "Super-Scalar. Stream Buffer. Victim Cache". UCB, URL: www.cs.berkeley.edu/~kubitron/courses/cs152-01/projects/submit/project7_report2.doc. 2006

[32] Hades Embedded Processor Timing Analyzer (Heptane), A Tree-based WCET Analysis Tool, 2009. URL: http://ralyx.inria.fr/2004/Raweb/aces/uid43.html

[33] Short for VisualSim Architecture (VisualSim), A System-level Simulator from Mirabilis Design, Inc, 2009. URL: http://www.mirabilisdesign.com/>

[34] Z. Wang. PhD Thesis. "Information Leakage Due to Cache and Processor Architecture" PALMS, 2012. URL: http://palms.ee.princeton.edu/node/407

[35] D. Stiliadia and A. Varma "Selective Victim Caching: A Method to Improve the Performance of Direct-Mapped Caches". IEEE Transactions on Computers, 2002. Vol 46, Issue 5. pp 603-610, DOI: 10.1109/12.589235

[36] R. Pendse and H. Katta "Selective Prefetching: Prefetching when only required". 42nd Midwest Symposium on Circuits and Systems, 1999. Vol 2. pp 866-869.DOI: 10.1109/MWSCAS.1999.867772

[37] A. Asaduzzaman, F.N.Sibai, M.Rani "Improving cache locking performance of modern embedded systems via the addition of a miss table at the L2 cache level". The EUROMICRO Journal of Systems Architecture, 2010, Vol 56, Issue 4-6. pp 151-162.