Performance of different Cache Coherence Protocols on different Multi-core Architecture Systems

PROJECT REPORT

Submitted by

V. KALAVATHI

in partial fulfillment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

Department Of Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

JUNE 2022

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

2022

CERTIFICATE

This is to certify that this thesis (or project report) entitled ***"Performance of different Cache Coherence Protocols on different Multi-core Architecture Systems"*** submitted by **V. KALVATHI** to the Indian Institute of Technology Madras, for the award of the degree of **Masters of Technology** is a bonafide record of the research work done by him under my supervision. The contents of this thesis (or project report), in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma..

Dr. T. G. Venkatesh
Project Guide
Associate Professor
Department of Electrical Engineering
IIT Madras 600036

Acknowledgment

First and foremost, I would like to express my deepest gratitude to my guide, **Dr. T G Venkatesh**, Associate Professor, Department of Electrical Engineering, IIT Madras, for providing me an opportunity to work under him. I would like to express my deepest appreciation for his patience, valuable feed backs, suggestions and motivations.

I convey my sincere gratitude to Shubhang Pandey, MS Scholar, IIT Madras, for all his suggestions and support during the entire course of the project. Throughout the course of the project he offered immense help and provided valuable suggestions which helped me in completing this project.

I would like to extend my appreciation to all my friends and for their help and support in completing my project successfully.

Abstract

The gap between the performance of the processor and the performance of the main memory is often known as the Memory Wall. To mitigate this gap, caches have been introduced as the on-chip memory, which holds the most relevant data closer to the processor for faster access. The processors evolved from one powerful core to multiple smaller cores, and the concept of multithreading came into existence. Multiple levels of caches were introduced, which could be kept as either private to the cores, partially shared among a few cores, or completely shared. Therefore, it became essential to keep the cores coherent. For some time now, numerous protocols have evolved to address the cache coherence problem.

This report discusses three important cache coherence protocols: MSI, MESI, and MESIF. We model these Cache Coherence Protocols as a Markov Chain Model and measure the optimality of these protocols with respect to a standard measure called Average Memory Access Time (AMAT). Out of various methods of finding the AMAT, we choose the Markov Model to make the analysis and compare the value obtained with the conventional model of AMAT calculation. The simulations are done using the Sniper simulator, compatible with x86 ISA, and the PARSEC and SPLASH2 benchmarks suite, namely Blackscholes, Bodytrack, Cholesky, FFT, Radix and Vips are used. The report also performs a detailed study of the performance of cache coherence protocol with varying associativity and block sizes.

Contents

List of Figures

List of Tables

Abbreviations

CPU	Central Processing Unit
DRAM	Dynamic Random Access Memory
FPGA	Field Programmable Gate Array
I/O	Input Output
AMAT	Average Memory Access Time
MSI	Modified Shared Invalidate
MESI	Modified Exclusive Shared Invalidate
MESIF	Modified Exclusive Shared Invalidate Forward

Chapter 1

Introduction

Due to the recent advancements in high accuracy and high speed computing, multi core processors have found an increased importance in recent years. Coupling this with the fact that performance scaling is difficult in single core processor further enforces the importance of their multi-core counterparts. Parallelism obtained due to multi threading allows to use the full potential of multi core processor in terms of performance improvement with respect to faster execution, better utilization of the system and lower power consumption. Multi-threading simply refers to using multiple threads present in a core all at once to run several smaller computations parallelly that may sum up to a larger calculation being done in a very small amount of time.

The optimal performance of multi-core processor can be levied by utilizing the cache hierarchy. Although caches provide a faster way of accessing rudimentary data, the use of shared memory presents a number of challenges. Multiple caches sharing a single memory may cause huge data traffic, resulting in a performance bottleneck. Many solutions have been proposed to this problem, hierarchical clustered cache design being one possible way out. Grouping cores and their caches in clusters allows the reduction of network congestion by localising traffic among different hierarchy levels, enabling more scope for scalability.

Cache coherence is important for consistent view of memory across all cores in a shared memory system. For a more diverse set of cores, more sophisticated coherence management system is required. In hierarchical clustered design, each cluster may function according to a different coherence protocol. The design becomes more and more complex with the increase in such protocols working in tandem.

Private processor caches in a shared memory environment may include copies of data that are irrelevant in relation to main memory. In such instances, cache coherence takes care of data management and dissemination. The consistency model of architecture may be violated if there is no coherence. When a private processor cache encounters a store that isn't visible in other processors' local caches, the consistency of the copies of data stored in the private processor caches is broken. This has an impact on the way processors with one another in a shared memory environment. Every processor should be aware of the load and stores performed by every other processor for functional correctness and this behaviour is ensured by cache coherence.

AMAT is a well-known measure for assessing system performance of a computer memory.It uses hit time, miss penalty and the miss rate to measure the memory performance.To evaluate memory performance, the AMAT employs hit time, miss penalty, and miss rate. It explains why hits and misses have differing effects on the memory system's performance. AMAT can also be used to several layers of the memory hierarchy in a recursive fashion. It focuses on how cache misses and locality effect overall performance, and it allows for a fast comparison of various cache design strategies.

There are conventional methods to find the AMAT but newer approaches have found importance in recent days. One such approach is the AMAT calculation using the Markovian Model. Markovian model is a stochastic model that assumes the future state of a randomly changing system does not depend on the past state. This analysis is commonly used for the purpose of forecasting the value of a variable whose predicted value gets influenced only by the present value and not any other value or state that has occured prior to the current state. This modeling finds itself of use in calculating and analyzing the AMAT of multi core processor systems.

1.1 Motivation

The area of chip designing for computer systems has seen an immense growth in recent years. The tradeoff between performance of the processor and the main memory is a possible hindrance for this growth. The Cache Coherence Problem is the problem of keeping multiple local caches synchronized when any one of the processors update its local copy of data that is shared among multiple caches. Cache coherence protocols have a tremendous impact on the performance of distributed and centralized shared memory of a processor and are required to maintain data consistency in a chip-multiprocessor system (CMP). Thus, these cache coherence protocols play a major role in the improvement of the performance of multiprocessor systems. An efficient cache coherent protocol particularly ensures the updating of processor data, broadcasting valid data and prevent main memory or other processors from loading invalid values.

1.2 Aim of the project

In this thesis, we study about the performance of various Cache Coherence Protocols on various Multi-core Architecture Systems. We discuss about the miss rates, hit rates of MSI, MESI, MESIF protocols using Sniper simulator. We then find the read hit, read miss, write hit, write miss probabilities of these protocols using two benchmark suites : PARSEC Suite and SPLASH-2 Suite. After all these calculations, the Average Memory Access Time (AMAT) is obtained using conventional method. This is followed by the analytical modeling that is done using the Markov model. From the state diagram of the MESI protocol, equations such as probabilities of each state are derived using Markov model. The state transition matrices of MESI protocol are then derived and the AMAT is calculated using the same Markov model equations and matrices. The AMAT values obtained from both the conventional and Markov models are then compared and the difference in their values are expected to be minimum.

1.3 Outline of Report

Chapter 1 gives a brief introduction into the project along with the motivation behind doing this work and the aim of the project. Chapter 2 presents a theoretical background on various terms involved in the project across various domains ranging from basics of Cache to Cache Coherence protocols and various mapping techniques. Chapter 3 presents a review of literature encompassing a plethora of papers across different topics similar to this work. It discusses mainly about the papers related to cache coherence, different benchmarks used to measure the processor performance etc. In Chapter 4, we discuss the simulation framework. Here, we talk about the simulator used, the procedure followed for simulation, different simulation results pertaining to different benchmarks across different number of cores for all the three levels of caches.

Chapter 5 presents the analytical modeling where we discuss various cache coherence protocols with respect to their state diagrams. We calculate different probabilistic values and a very crucial quantity called the Average Memory Access Time using two different approaches, the conventional approach and the Markovian approach. We present a hoard of results of analytical modeling spread across various benchmarks (Blackscholes, Bodytrack, Cholesky, FFT, Radix, Vips) and varying number of cores (4,8,12,16). The AMAT is then compared for different benchmarks taking into consideration the conventional model as well as the novel Markov Model. Chapter 6 culminates with a conclusion on the different results obtained from the previous chapters.

Chapter 2

Background

The performance of the CPU registers have been improved exponentially from the past 30 years whereas the growth of the memory is improving still not up to the same growth as that of the CPU. This led to the variation in the speed of the processor operation. To eliminate that, memory hierarchy structure was proposed.

- **Role of Memory hierarchy:** It eliminates the gap between CPU and the main memory. It has multiple levels of memory and each level of memory has different sizes and speeds. In memory hierarchy, the fastest and smallest memories are kept closer to the CPU and the slowest and largest memory is kept near to the main memory. In between the main memory and the CPU, we have different level of caches with different size and speeds and then we have the disk storage next to the main memory. By the bridging the gap between the main memory and the CPU, it improves the performance of the processor and decreases the dependency of the main memory which is slower.

2.1 Introduction of Cache memory

In Computer Architecture, a cache is a component which stores the data temporarily so that the data can be served faster on a further request. The data which is stored in the cache is either the result of prior computation or a copy of data which is stored somewhere else. A cache is small, faster and act as buffer between the processor and memory. Cache memory improves the processor overall performance by decreasing the number of time to access data from the main memory.

- **Cache hit:** When the data requested by the processor is available in the cache, then it is called Cache hit.
- **Cache miss:**When the data requested by the processor is not available in the cache, then it is called Cache miss.
- **Hit rate:** It is the total no. of the cache hits to the total no. of the memory request made to the cache in the given time.
- **Miss rate:**It is the total no. of cache misses to the total no. of the memory request made to the cache in the given time.
- **Miss penalty:** The extra time taken to bring the data required into the cache from the memory when the miss occurs.
- **Line/block:** The minimum unit of data which can be either present or not present in the cache level.
- **Cache size:** The cache size is the quantity of data that main memory can only hold . It is the no. of the data i.e stored in each of the block to the total no. of blocks stored in the cache.

2.1.1 Types of Cache Misses

- **Compulsory Misses:** It occurs when we miss an address for the first time and it may not be present in the cache.
- **Capacity Misses:** When the cache is completely full and we need to replace the new cache.
- **Conflict Misses:** This miss happens due to the conflict of the set index. Suppose when 'x' data is thrown out from the cache and when we need the same data again, this miss occurs.

2.2 Principle of locality

Locality of reference is the event in which the processor tends to access the same set of memory location repeated number of times for a specific period of time. Also it can be defined as the event in which the processor tends to access the data whose addresses are near by. They are 2 types:

- **Temporal locality:** Temporal locality is the present data i.e being fetched may be required very shortly. Hence we need to store that data in the cache so that we can save time instead of searching the same old data in the main memory.
- **Spatial locality:** Spatial locality is the data which close to the present location of the i.e being fetched which can be required very shortly. Temporal locality is all about the actual memory location which is being fetched whereas Spatial locality is about near by located memory locations.

2.3 Cache coherence

Cache coherence is the consistency of shared data which is stored in multiple local caches. The Cache Coherence Problem arises because we need to keep the multiple local caches in synchronization so that when anyone of the processors updates its copy of data which is shared among the multiple local caches. This problem occurs due to the following causes:

- Sharing of resourced writable data
- Input-Output activity
- Migration of process

2.4 Cache Write Policies

Caches are used because it stores the data temporarily and data can be accessed faster when compared to the main memory. But caches have limited size, so we need to have a proper management polices to access the data so that we can write, read the data. There are 2 cache write policies:

- **Write through:** Here all the write operation are updated simultaneously to the cache and the main memory. This process is used when the write operations are less.This policy is more simpler but it has more delay as we have to update/write the data to both the locations i.e cache and main memory. It helps in finding the recovery of data in case of system failure.

- **Write back:** Here in this method the data is updated in the cache every time when change occurs and memory at later time. The data ishttps://www.overleaf.com/project/6242f9a31fc332993e07568b written or updated only in the cache and the cache block which is modified is updated to the main memory only when it's replaced.

Write back is more preferred because it has better performance. The main reason to use the cache is to avoid multiple times accessing to the main memory but whereas in write through policy, it is updated every time to the main memory. In write back policy, the data is only updated or written to the main memory when the cache line or cache block is ready to be replaced or invalidated. Write back saves a lot of bandwidth when compared to the write through caches.

2.5 Cache mapping techniques

There are different types cache mapping techniques to load the data from the main memory which are as follows:

- 1. Direct mapping technique

- 2. Set associative mapping technique

- 3. Fully associative mapping technique

- **Direct mapping:** It maps all block of the main memory only to a single line of the cache. Say if the line has a data which is taken up by the main memory and if the new block arrives, the data which is currently stored in the block will be replaced by the new data. This is the simplest technique as it requires less hardware. It has 2 fields: Tag and the Index field. The tag field is saved in the cache and the index is saved in the main memory. This direct mapping technique behaviour is proportionate to the hit ratio. As each block of the main memory maps only to a single line of the cache, so if the new block arrives, the old block is trashed and if both the blocks are continuously referenced, the 2 block will be swapped in and out continuously and it leads to the conflict miss.
- **Set associative mapping:**This technique is introduced which decreases the drawbacks of the previous technique that is direct mapping. Here instead of a single line, we have set of lines which maps to the main memory block. Here the main memory block can be mapped to one of the lines of a particular set. The tag bit is to identify which of the particular set is used and the main memory block data is stored in the index address.
- **Fully associative mapping:**In this technique, any block can be mapped to any line of the cache. It means that a single cache set with multiple cache lines. Here the index bit is used to identify which data in the block is required and the tag is the rest bits. It is much flexible and has better hit rate than the other mapping techniques. But it takes more time as we need to search all the cache lines for a particular block in the cache, as it can be anywhere in the cache.

There are 3 main Cache Coherence Protocols:

2.6 MSI PROTOCOL

MSI protocol is primary cache coherence protocol which is used in the multi processor system. It has 3 states in which cache can be present. In MSI, each of the block present in the cache can have one of the 3 following states:

- **Modified:** Here the block inside the cache is modified. When the data in the cache is evicted, it has to write the block to the main memory.

- **Shared:**This state is used for read only in atleast one cache and the block is unmodified here. Here the cache can remove the data without writing it to the main memory.
- **Invalidate:** Here the block is either invalidated or present in the current cache and it must be retrieve the data either from main memory or the other cache if the block has to be saved in the cache.

2.7 MESI PROTOCOL

MESI protocol is the more widely used cache coherence protocol. It is an invalidate based cache coherence protocol and supports write back. This MESI protocol decreases the number of times the main memory access compared to the MSI protocol. It has a dirty state present which shows that the data present in the cache is unlike from the main memory. It has 4 states:

- **Modified:** The cache block here is exclusively available just in the current cache and it has been modified. Hence the copy present in the main memory becomes old.
- **Exclusive:** Here the data in the block is present only in this cache and not present in any other caches of the core. So, therefore the copy present in the main memory is same as the cache block.
- **Shared:** Here the copy of the cache block/line may be present in other caches. This state is clean and it has the same copy as the main memory.
- **Invalidate:**It means the data present in the cache block/line is invalid. It may be due to the copy of the data in that local cache is not the most new one, ans so it is invalid to use it.

2.8 MESIF PROTOCOL

The MESIF protocol has 5 states and the first 4 states(M, E, S, I) are same as of the MESI protocol. It has 1 special state called forward state, which is the more special form of the shared state. If anyone of the cache holds a line in the shared state, then other cache holds the line in the forward state.

A cache line request received by multiple caches holding a line in the S state will be operated inefficiently in a system of caches using the MESI protocol. It could be satisfied from slow main memory, or it could be satisfied from all sharing caches, deluging the requestor with redundant responses. A cache line request will only be decided to respond to by the cache holding the line in the F state in a system of caches using the MESIF protocol.

The major difference from the previous protocol is that whenever the read request for a copy of the line occurs, it always goes the cache in the F state. When it satisfy the read request from the memory, then only the cache enters in the shared state.

Chapter 3

Literature survey

This section provides a quick overview of the literature on Cache coherence protocols.The following are of some of the literature that discusses about the need of the cache coherence protocols and their advantages, disadvantages.

In today's computing world, multicore processors are gaining traction. Cache coherent memory is implemented in all multicore processors. To avoid cache incoherence, a variety of techniques can be used. Joshi discussed the Cache miss rate, traffic, execution time, energy, and average memory access time are all common parameters that affect cache coherency. And also discovered a need for a scalable cache coherence technique that was less complex and had better values for common parameters [1].John L. Hennesy and David explained about the Memory hierarchy design and level parallelism of Computer Architecture with a quantitative approach [2].

Cache coherence problem cannot be solved by just using only write-through or write back without cache coherence protocol, so to keep the cache consistency, protocols must be used [3]. Al-Waisi explained about the comparision of Invalidation-based protocol (MSI, MESI protocol). and Update-based protocol (Dragon, Firefly protocol), a)The states available in each protocol, b)Energy consumption of each protocol and their misses, c)Bandwidth and d)how the processors works for each protocol [4]. The literature explains the performance analysis of cache replacement policies using simulations on a variety of benchmarks, as well as some of the recent advances in cache design to improve memory management unit access time, latency, and energy consumption. According to the results of this survey, processor speeds are increasing faster than memory latency is decreasing, so eliminating cache misses is critical for overall processor performance [5].Vakil-Ghahani et al. proposed a new replacement policy which utilises of the relationship between the reciprocal of hit counts and the block's reuse distance [6]. The snoopy bus cache coherence protocols using 3-state, 4-state and 5-state are designed and implemented using the write-invalidate approach in a shared memory dual processor system [7].Jang proposed a cache design for larger last-level caches, so that even with high granularity, good performance can be achieved [8].Sun [9] presents a survey to estimate the impact of the shared data and a directory based MESI protocol using write-update approach. Tiwari focused on multi-core processor coherence techniques. The results of the Snoopy coherence method simulation are examined in terms of block size, cache size, and associativity. Also, as we move from MI, MESI, and MOESI in Directory coherence technique, performance improves [10]. The paper gives an insight on the variations of associativity affecting the miss ratio. The paper discusses using simulations of alternative caches to remove the causes of miss. [11]

Michael et. al. have explained with detail about the Nehalem Processor and Nehalem Platform Architecture [12]. Mithil explains about the performance of cache coherence protocols with 4 different benchmarks [13].The paper describes both the hardware and software in detail, allowing for a variety of accuracy and simulation performance trade-offs, as well as interval simulation, and explains why the Sniper is an important tool in the architect's toolset for modelling high performance multi-core and many-core systems [14]. There are 2 types of benchmark suites. They are PARSEC benchmark suite and the other is SPLASH-2 benchmark suite [15]. PARSEC (Princeton Application Repository for Shared-Memory Computers) is a benchmark suite for Chip-Multiprocessor studies. The benchmark suite is varied of working set, locality, data sharing, synchronisation, off-chip traffic, parallelization, as well as their communication-to-computation ratio are illustrated in this paper by Christian Bienia, Jaswinder, Sanjeev and Kai Li [6]. Steven has detaily explained about the SPLASH-2 suite and the need of the benchmark suite.The paper describs how the characteristics scale with key application and machine parameters and defined the programmes along several based on behavioral axes [16]. In this study,Christian Bienia, Jaswinder, Sanjeev and Kai Li compared the SPLASH-2 and PARSEC benchmark suites seeing what differences and similarities exist between the two pairs of programmes. On Chip-Multiprocessors, we assess the suites for redundancy and overlap using standard statistical methods and machine learning (CMPs). PARSEC workloads are fundamentally different from SPLASH-2 benchmarks, according to our research [17].

The solution of linear time varying fractional order systems is discussed in this paper by Zhang, Yuanwei. The state transformation matrix of linear time varying fractional order systems is developed [18]. Using state transition matrix inequalities, sufficient conditions for stochastic admissibility are proposed [19].The model was tested against simulations using the multi-core simulator Sniper with the PARSEC and SPLASH benchmark suites, and the study presents an analytical method to find the miss rate of L2cache for various configurations with regard to L1 cache [20]. Balakrishnan gives the idea about the least recently used bits and optimize the cache block using invalidation policy [21]. The paper gives the details about the analytical modeling of the locality and caches and also explained about the probability of the cache hit of each stack [22]. Harper gives the details on the number of array references in the programs, rather than the actual number of memory accesses, and explained the affect of the prediction rate set associative cache of the analytical modeling [23].

Chapter 4

Simulation Framework and Results

In this chapter, we will be discussing about the simulator used and the benchmarks used to configure MSI, MESI and MESIF protocols. We will then discuss about the results obtained of different benchmarks for MSI, MESI and MESIF protocols.

4.1 Simulator used

Sniper is an x86 simulator which provides next generation, parallel, high speed and accuracy. This simulator is build on the graphite infrastructure and the interval core model which provides fast and accurate simulation. This multi core simulator also permits a range of flexible simulation options for different heterogeneous and homogeneous multi core architectures. It also provides to perform the timing simulation for both the multi threaded and multi programmed workloads ans shared memory applications which runs 10 to 100 cores with high speed when compared with other simulators. This simulator is core model which is build on the interval simulation which provides faster simulation and evaluation duration. Sniper is a multi-socket Intel core2, Nehalem system which provides the average prediction errors which are less than 25% with a speed upto several Million instruction per second(MIPS). The simulator is used for uncore and system level studies which provides more detail rather than the one IPC models. This interval core models allows the CPI stacks generation, that gives the number of cycles lost because of the different characteristics of the system. It provides characteristics such as branch predictor, cache hierarchy which leads for a better understanding of how each component effect on the system performance.

In addition to these features, it has few different benchmarks suits in the study. The quantitative foundation of computer architecture research is benchmarking. The only way to accurately measure performance is to look at how long it takes for a programm to run.They are SPLASH-2 and PARSEC. The SPLASH-2 has Cholesky, FFT and Radix benchmarks whereas PARSEC has Blackscholes, Bodytrack and Vips benchmarks.

4.1.1 SPLASH-2 benchmark:

SPLASH-2(Stanford ParalleL Applications for SHared memory) is a suite of parallel programs written for cache coherent shared address spaced machines [16]. Their work is based on two main goals - quantitative characterization of SPLASH-2 programs in terms of fundamental properties and assist people using the set of programs to make out a meaningful and informed conclusion out of it.

Limitations: It consists of a small number of programs and does not provide broad enough covering of science or engineering domain. It is not implemented for optimal interaction with modern memory system characteristics like long cache lines, high latency, physically distributed memory etc.

Expansions to overcome limitations: It contains wide range of computational and engineering problems, uses better algorithms and implementations and have a more architecturally aware structure.

4 axes to consider while choosing experimental parameters to understand shared address space programs:

- speedup and load balancing
- working sets
- communication to computation ratios and traffic requirements
- spatial locality issues

Despite being a flexible and fairly useful benchmark suite, it still has a some limitations. Its program collection has an affinity towards High-Performance

Computing and graphics programs. This limits it from including parallelization models such as the pipeline model which are widely used in other areas. This led to exploration of new benchmark suites, PARSEC being one of them.

4.1.2 PARSEC benchmark:

Bienia et. al described about PARSEC (Princeton Application Repository for Shared Memory Computers) [24]. PARSEC is a benchmark package for analysing Chip-Multiprocessors (CMPs). Previous multiprocessor benchmarks focused on high-performance computing applications and used a restricted variety of synchronisation methods.It encompasses new applications in recognition, mining, and synthesis (RMS), as well as system applications that emulate large-scale multi-threaded commercial programs. The requirements of a benchmark suite according to this paper are as follows:

- Multi-threaded Applications
- Emerging Workloads
- Diversity
- Employ State-of-the-Art Techniques
- Support Research

4.2 Simulation Procedure

The detailed procedure for the simulations are given as follows for different benchmarks and different core setups (core 4, 8, 12 and 16).

- We first open the Sniper simulator and open the configurations Nehalem and Gainestown.
- We then select the desired protocol (MSI, MESI, MESIF) in Nehalem and then select the required cache size and associativity of L1, L2 and L3 caches.
- We launch the UBUNTU terminal through which we consequently launch the Sniper simulator and go into the benchmarks and run it. For example to run a benchmark named blackscholes for core 4 using the following command:

  - ./run-sniper -p parsec-blackscholes -i small -n 4 -c gainestown

- The output is obtained in sim.out in the benchmarks file which is present in the Sniper folder.
- The required results are skimmed out of a large number of other values for eg. miss rates of caches.
- We then run the following command to get the detailed information like store misses, total stores, load misses, total loads etc. of all the caches.

  - ˜/sniper/tools/dumpstats.py > output.txt

After obtaining the values following the above procedure, we can calculate the required probabilities that are needed to find the probabilities of each state of MESI protocol and AMAT. We can make the graphs that are required are shown in the upcoming sections, with a detailed inference of the same.

4.3 Simulation Results

We ran the simulations of different benchmarks such as Blackscholes, Bodytrack, Cholesky, Vips, Radix and FFT and plotted the miss rates for all these benchmarks for core 4, core 8, core 12 and core 16 setups.

4.3.1 Average Missrate of all benchmarks for core 4

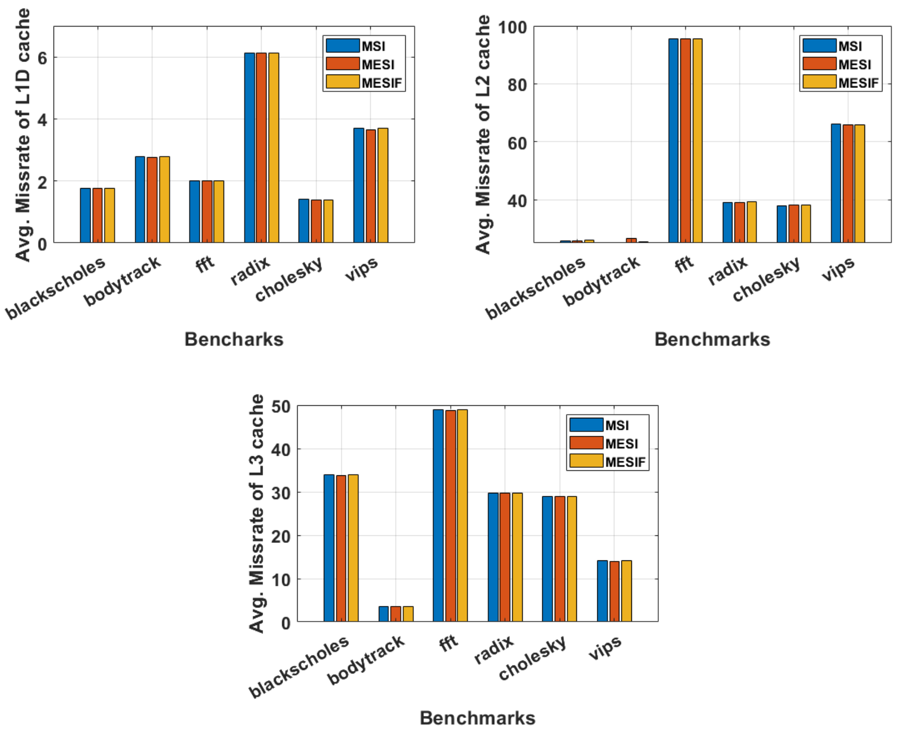

Figure 4.1: Average Missrate for core4

Radix has a higher miss rate than other benchmarks for L1D cache while FFT has higher miss rate for L2 and L3 caches [24]. Bodytrack has an overall lower miss rate as compared to the other benchmarks.

4.3.2 Average Missrate of all benchmarks for core 8

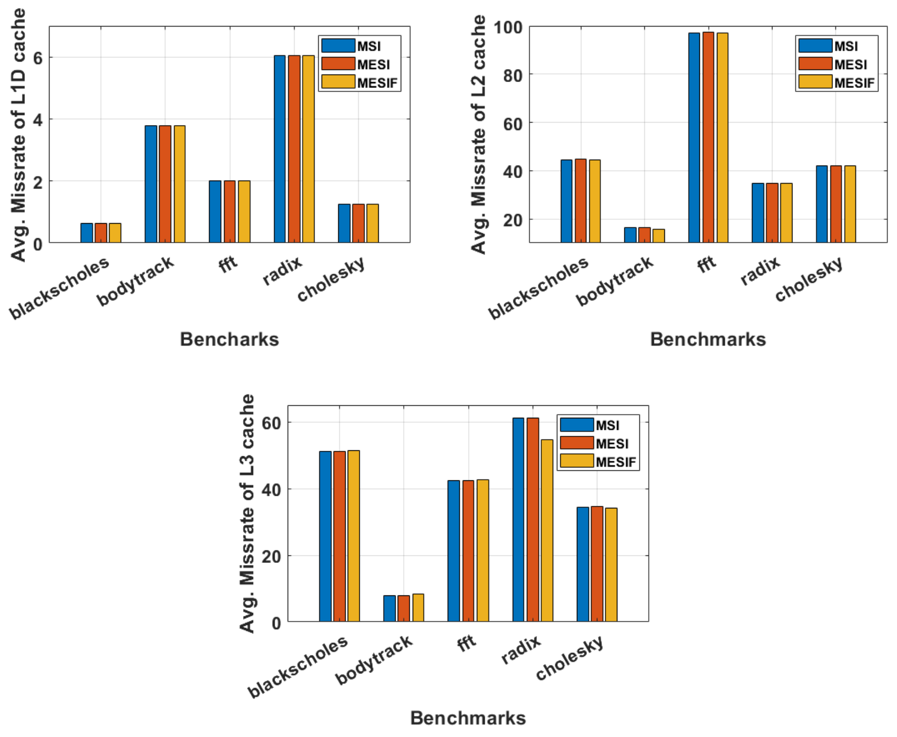

Figure 4.2: Average Missrate for core8

As we progress from core 4 to core 8, there is a very minute change in the miss rate. This can allow us to conclude that using 8 core is more beneficial than its 4 core counterpart.

4.3.3 Average Missrate of all benchmarks for core 12

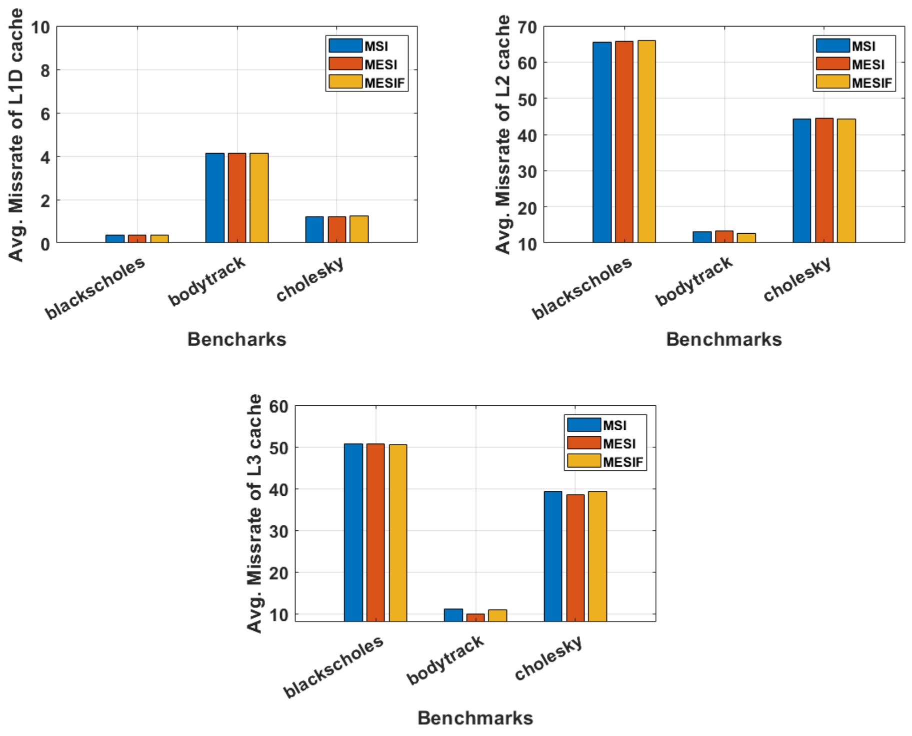

Figure 4.3: Average Missrate for core12

As we go from core 8 to core 12, the miss rates of L2 cache in Blackscholes benchmark increases drastically whereas the other benchmarks still maintain more or less the same miss rate values. So to calculate the miss rates for core 12, it is recommended to use benchmarks other than blackscholes.

4.3.4 Average Missrate of all benchmarks for core 16

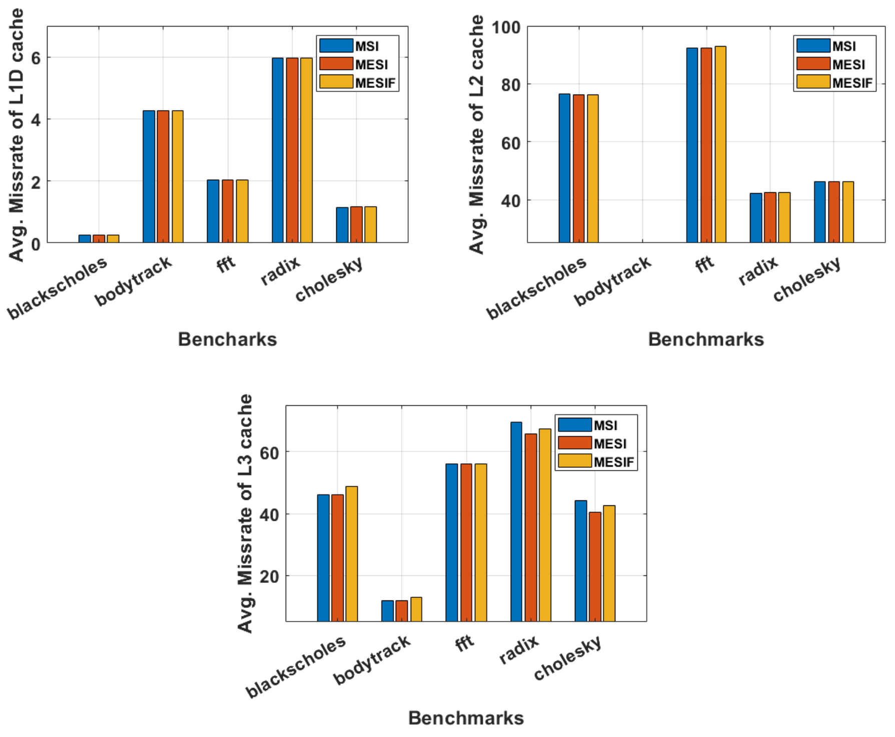

Figure 4.4: Average Missrate for core16

When we go from core 12 to core 16, the miss rates of all the benchmarks remain more or less the same. In a day to day operation where heavy computation is desired, we can use the higher core processor that gives the same miss rate as its low core counterpart.

4.3.5 Average Missrate for variation of L2,L3 associativity keeping L1 associativity as 4

The indices for A, B and C are 4, 8 and 16 respectively. This means AAA represents 4, 4, 4 associativity of L1D, L2 and L3. By this logic ABA will be 4, 8, 4 associativity of L1D, L2 and L3 and ABC will be 4, 8, 16 associativity of L1D, L2 and L3 and so on.

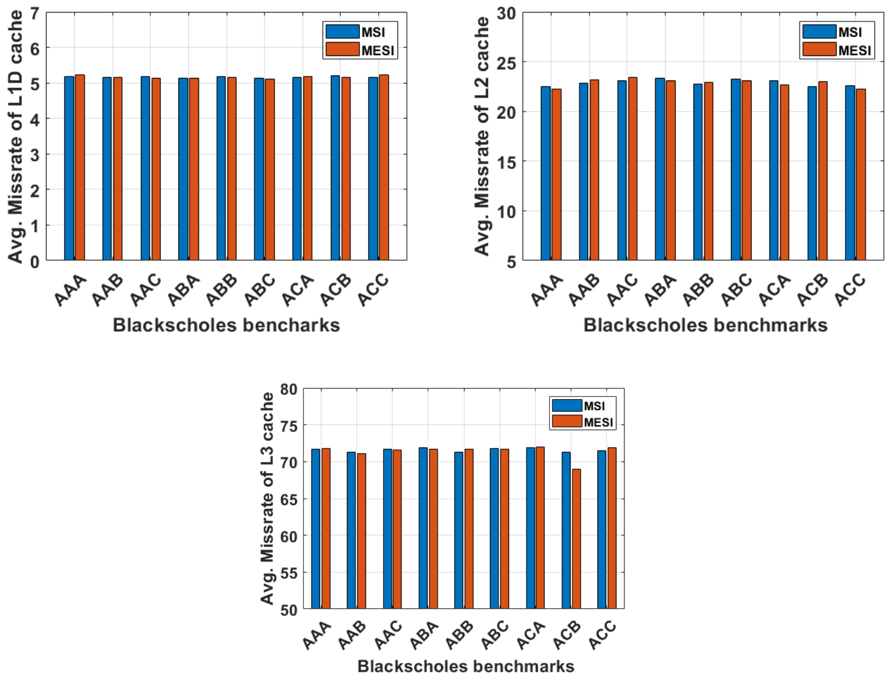

Figure 4.5: Average Missrate for variation of L2,L3 associativity keeping L1 associativity as 4

Associativity used in the caches to store the data and address the memory word. The MESI protocol has an advantage over the MSI protocol in that if the current cache is in the Exclusive state, it can silently drop the cache block without having to perform the costly writeback operation. The MESI protocol ensures that the processor always has the most current value. So when the miss rate of MSI and MESI is less or more the same, we go for the MESI protocol as it has few advantages over MSI.

4.3.6 Average Missrate for variation of L2,L3 associativity keeping L1 associativity as 8

The indices for A, B and C are 4, 8 and 16 respectively. This means BAA represents 8, 4, 4 associativity of L1D, L2 and L3. By this logic BAC will be 8, 4, 16 associativity of L1D, L2 and L3 and BAB will be 8, 4, 8 associativity of L1D, L2 and L3 and so on.

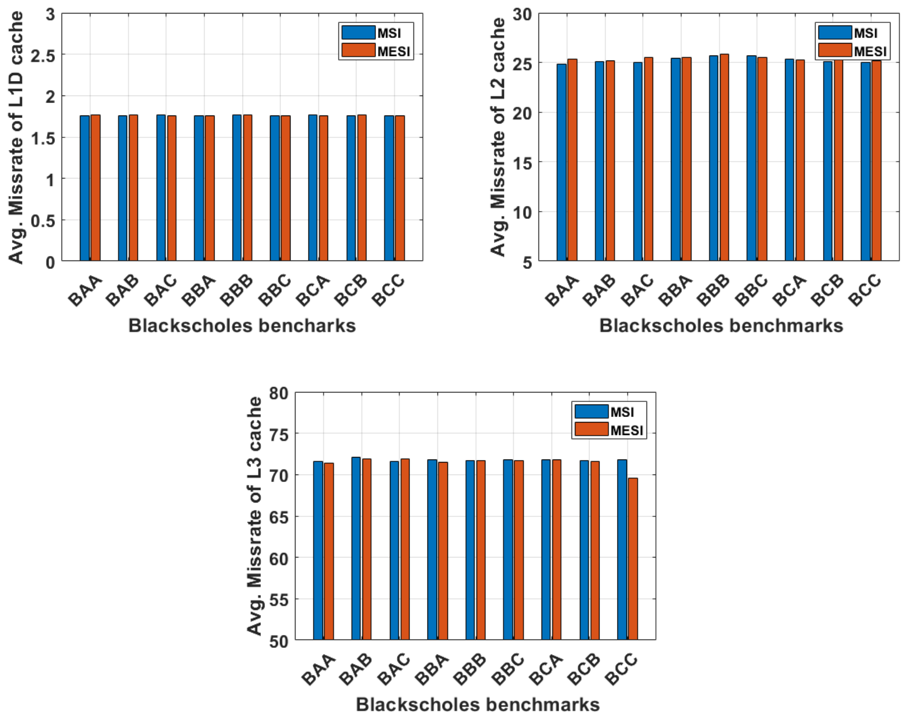

Figure 4.6: Average Missrate for variation of L2,L3 associativity keeping L1 associativity as 8

When we go from associativity of L1 from 4 to 8, the miss rate of L1 decreases to almost half and the L2 and L3 miss rate remain almost the same. Based on the requirements, we can use the required associativity.

4.3.7 Average Missrate for variation of L2,L3 associativity keeping L1 associativity as 16

The indices for A, B and C are 4, 8 and 16 respectively. This means CAA represents 16, 4, 4 associativity of L1D, L2 and L3. By this logic CAC will be 16, 4, 16 associativity of L1D, L2 and L3 and CAB will be 16, 4, 8 associativity of L1D, L2 and L3 and so on.

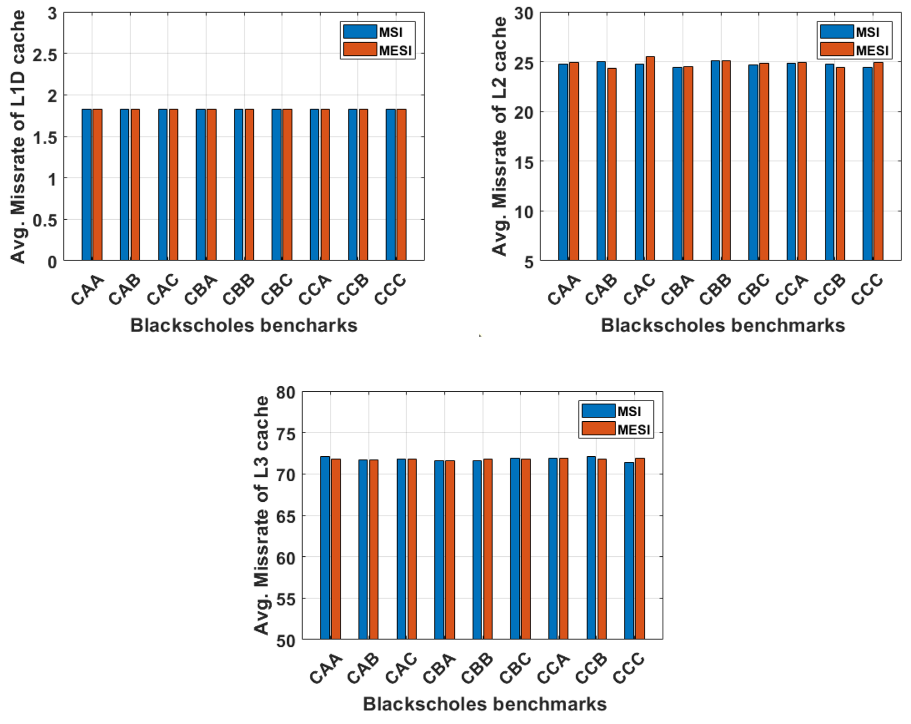

Figure 4.7: Average Missrate for variation of L2,L3 associativity keeping L1 associativity as 16

When we go from associativity of L1 from 8 to 16 and varying L2 and L3, the miss rates almost remains the same. So we can use the more associativity of caches as the miss rates are same for lower associativity Upon moving from 8 to 16, the miss rates remain the same.

4.3.8 Average Missrate for variation of L2,L3 cache size keeping the L1 cache size as 32

The following table gives the size of the L1, L2 and L3 cache size for the following graph.

Cache size	**L1**	**L2**	**L3**
1	32	64	1024
2	32	64	2048
3	32	64	4096
4	32	64	8192
5	32	128	1024
6	32	128	2048
7	32	128	4096
8	32	128	8192

Table 4.1: Cache size of L1, L2 and L3

When the cache size of L2 and L3 changes keeping L1 as same size, the miss rate of L1 of MSI and MESI protocol are almost same whereas the L2 and L3 missrates of MSI and MESI protocols varies. The miss rate of MESI decreases when compared to the MSI protocol. So as the cache size increases, it is better to go for MESI protocol over MSI.

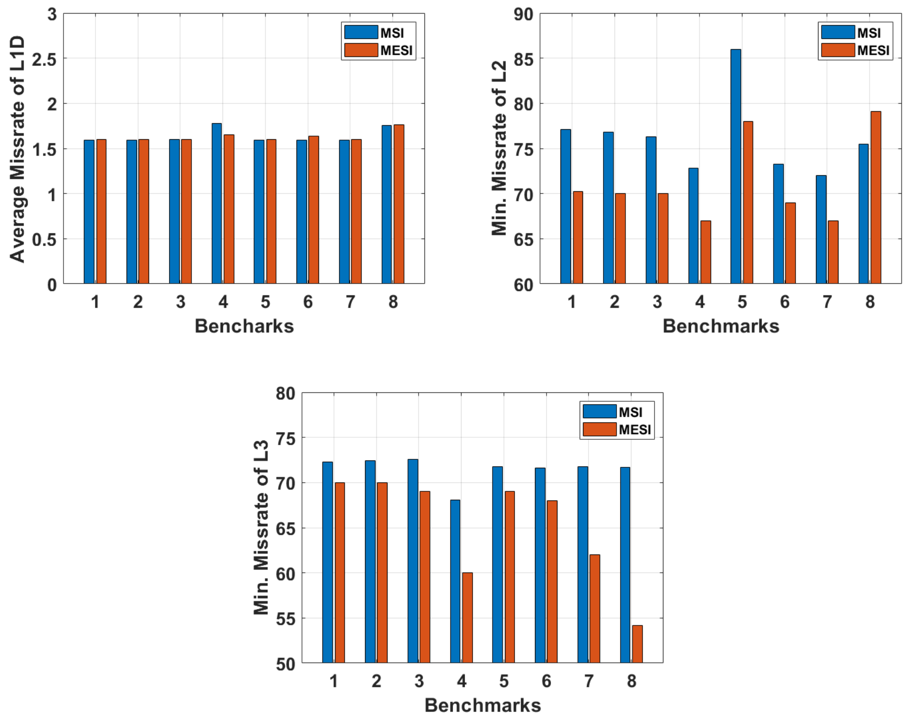

Figure 4.8: Average Missrate for variation of L2,L3 cache size keeping the L1 cache size as 32

4.3.9 Average Missrate for variation of L2,L3 cache size keeping the L1 cache size as 32

The following table gives the size of the L1, L2 and L3 cache size for the following graph.

When the cache size of L2 and L3 changes keeping L1 as same size, the miss rate of L1 of MSI and MESI protocol are almost same whereas the L2 and L3 missrates of MSI and MESI protocols varies a lot. The miss rate of MESI decreases drastically when compared to the MSI protocol. So as the cache size increases, MESI protocol is best compared to MSI.

Cache size	L1	L2	L3
9	32	256	4096
10	32	256	8192
11	32	512	4096
12	32	512	8192
13	32	1024	4096
14	32	1024	8192

Table 4.2: Cache size of L1, L2 and L3

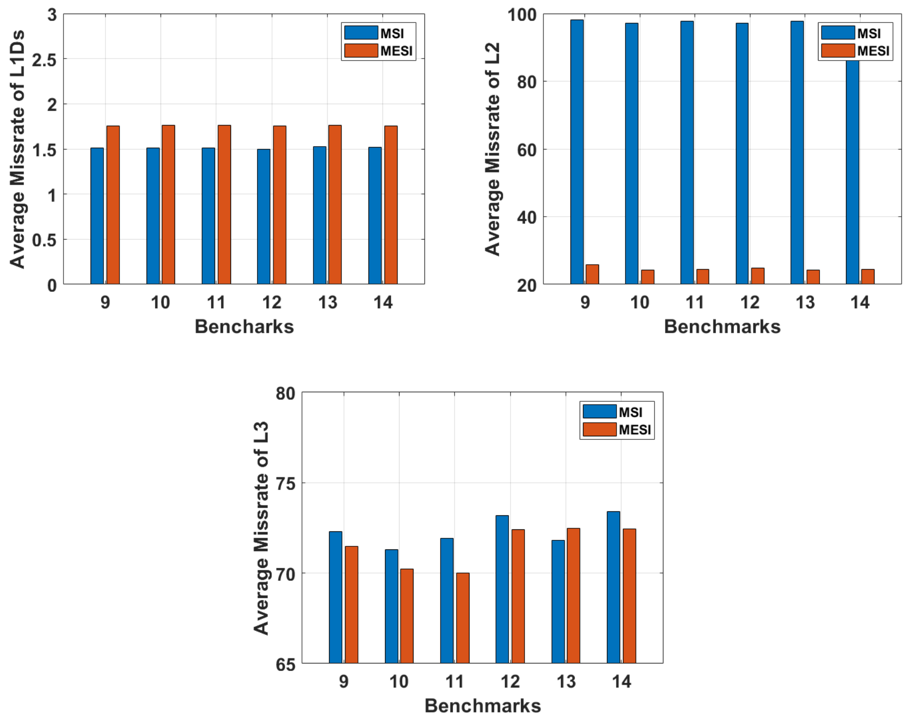

Figure 4.9: Average Missrate for variation of L2,L3 cache size keeping the L1 cache size as 32

4.3.10 Average Missrate for variation of L2,L3 cache size keeping the L1 cache size as 64

The following table gives the size of the L1, L2 and L3 cache size for the following graph. When the cache size of L2 and L3 changes keeping L1 as

Cache size	**L1**	**L2**	**L3**
15	64	128	4096
16	64	128	8192
17	64	256	4096
18	64	256	8192
19	64	512	4096
20	64	512	8192
21	64	1024	4096
22	64	1024	8192

Table 4.3: Cache size of L1, L2 and L3

same size, the miss rate of L1 of MSI and MESI protocol are almost same as the previous case whereas the L2 and L3 missrates of MSI and MESI protocols varies. The miss rate of MESI protocol decreases as the cache size increases.

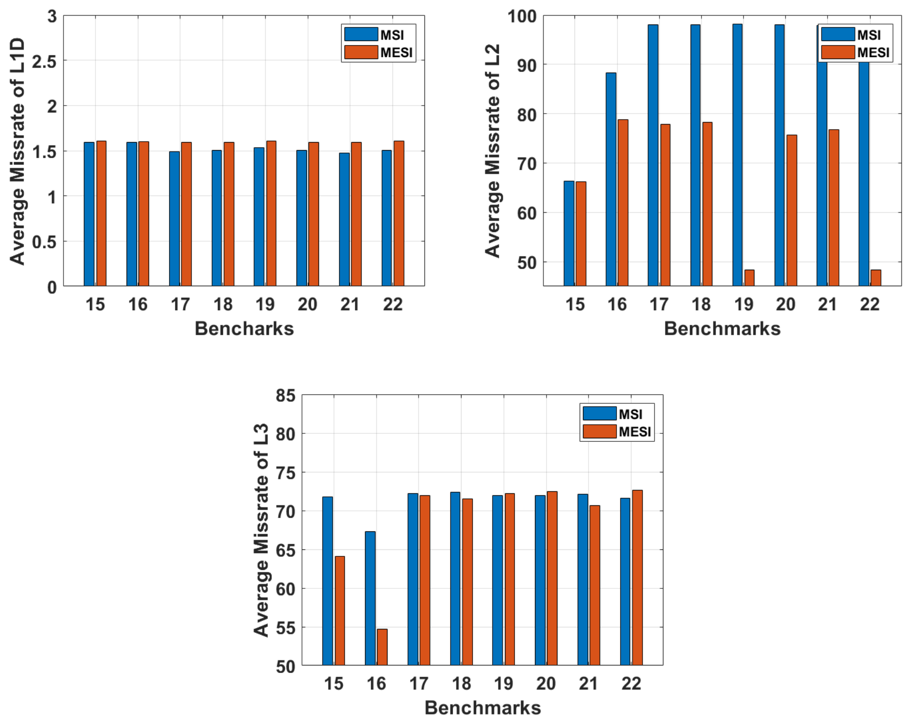

Figure 4.10: Average Missrate for variation of L2,L3 cache size keeping the L1 cache size as 64

Chapter 5

Analytical Modeling

In this chapter, we will be solving the probabilities of cache hit, cache miss, read and write ratio. Using the Markov model, we will be solving the probabilities of each state of MESI protocol.

5.1 State diagram of MESI protocol

For the model, let us assume that RH be the Read hit of the cache and the RH occurs when the cache block is either in modified, shared or exclusive state. When the read operation of the cache occurs, it's stays in the same state as shown below in 5.4. It stays in the same state as long as the block is recent and so it is valid to read the same. If the block is in the shared state, then all the shared blocks are constant and memory read will not modify the content of the block. so therefore, it's not necessary to modify the state of the shared blocks that are present in other cores [25].
When the block is not present in the cache, then it means the state as invalid. Let RM be the Read miss, and When the block is in the invalid state, it is same as the cache miss of the read operation. To fulfil a read, the invalid block must be fetched from main memory and placed in the shared or exclusive states. Before doing so, the processor sends a message to the snoop bus called SHR(Snoop hit on a read). The other caches snoop the message and, depending on the state of the cache block, perform one of the following activities.

- **Exclusive:** One of the local caches has an exclusive cache block (multiple cache blocks cannot be in exclusive state). When the processor (which has that block in exclusive state) hears SHR, it replies by indicating that it shares the block and changes the state of its block from exclusive to shared. The starting processor will then perform a block read operation, which will alter the block's state from invalid to shared.

- **Shared:** The block is in the shared state in one or more caches. Hearing SHR, all of those processors send a signal to the originating processor which indicates that they are sharing the block. The initiating processor reads the block and changes the state from invalid to shared because the main memory copy is valid.
- **Modified:** In one of the local caches, one cache block has been modified (multiple cache blocks cannot be in modified state). The responding CPU sends a signal to the initiating processor when it hears the SHR, instructing it to try the read operation again later. The initiating processor stays in an invalid state and tries to read again later. Meanwhile, the responding processor changes the block's block status from modified to shared and updates the main memory copy of the block.

Because no other processor has a copy of this block, the SHR receives no signal. So the processor reads the block and updates the state to exclusive state.

Let us consider WH be the write hit of the cache and it occurs when the block is present in the cache. Write hit occurs either in Modified, Exclusive or Shared state. The block enters to the invalid state when write miss(WH) occurs.

- **Exclusive:** The block is only available to the beginning processor. The initiating processor just writes the block and switches the state from exclusive to modified.
- **Modified:** The block is exclusive to the initiating processor, but it is modified. The initiating processor simply writes the block and saves the state as the modified state.
- **Shared:** The block exists in one or more shared copies. A signal called invalidate transaction is sent by the processor. The processor with the shared block understand that the processor's plan to change the data when it receives this signal and those processors invalidate their blocks. And then the block is modified by the initiating processor, and the status is changed to modified.

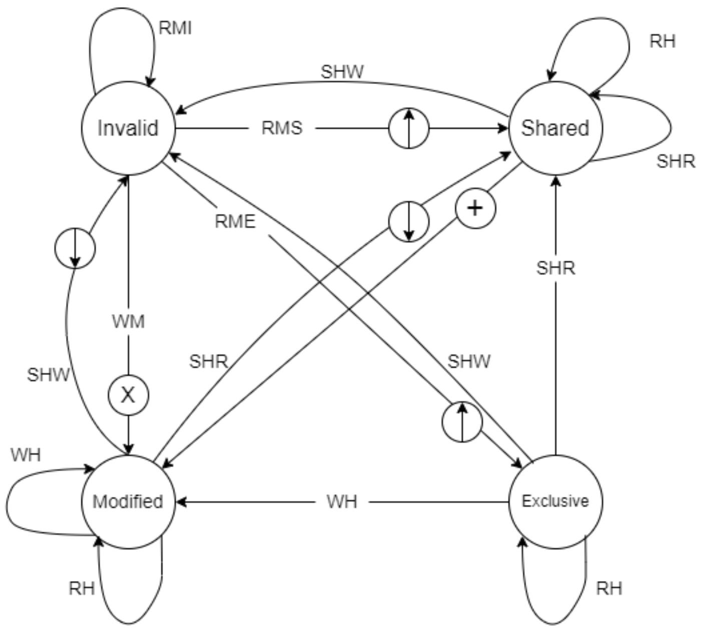

Figure 5.1: State diagram of MESI protocol

If the block is not available in the local cache, it is called write miss and is denoted as WM. When the block isn't in the local cache, it must be read from main memory and modified from there.

- **Modified:** After invalidating the block, the responding processor writes it back into main memory. After that, the initiating processor reads the block, writes it, and update it as modified state.

- **Shared or Exclusive:** No response will be send back. The other caches, on the other hand, invalidate their blocks since the initiating processor will modify them.
  The equations that follow in the next section are derived from the above state diagram 5.4 .

5.2 Probabilities

In the following sections, the probabilities of cache hit, cache miss, MESI protocols are derived.

5.2.1 Cache hit and miss probabilities

Let N be the number of cores in a multiprocessor system. P_H and P_M are the hit ratio and miss ratio probability respectively.

$$P_H + P_M = 1 \tag{5.1}$$

$$P_H = \frac{\text{Total no. of cache hits}}{\text{Total no. of cache hits+ Total no. of cache misses}} \tag{5.2}$$

$$P_M = \frac{\text{Total no. of cache misses}}{\text{Total no. of cache hits+ Total no. of cache misses}} \tag{5.3}$$

The read hit, write hit probabilities can be denoted by P_{RH} and P_{WH}.

$$P_{RH} = \frac{\text{Read hits}}{\text{Total number of Reads}} \tag{5.4}$$

$$P_{WH} = \frac{\text{Write hits}}{\text{Total number of Writes}} \tag{5.5}$$

$$P_{RM} = \frac{\text{Read misses}}{\text{Total number of Reads}} \tag{5.6}$$

$$P_{WM} = \frac{\text{Write misses}}{\text{Total number of Writes}} \tag{5.7}$$

$$\text{Total number of Reads} = \text{Read hits + Read misses} \tag{5.8}$$

$$\text{Total number of Writes} = \text{Write hits + Write misses} \tag{5.9}$$

$$P_{RH} + P_{RM} = 1 \tag{5.10}$$

$$P_{WH} + P_{WM} = 1 \tag{5.11}$$

5.2.2 Probabilities of MESI protocol using Markov model

If the probability of being in modified state, exclusive state, shared state and invalid state are denoted by π_M, π_E, π_S and π_I respectively, their individual probabilities can be calculated by the equations given below by deriving it from the above state diagram 5.4:

$$\pi_M = \frac{P_{RH}}{N}\pi_M + \frac{P_{WH}}{N}\pi_M + \frac{P_{WH}}{N}\pi_E + \frac{P_{WH}}{N}\pi_S + \frac{P_{WH}}{N}(N-1)\pi_I \quad (5.12)$$

$$\pi_E = 0 + \frac{P_{RH}}{N}\pi_E + 0 + \frac{P_{RM}}{N}(N-1)\pi_I \quad (5.13)$$

$$\pi_S = \frac{P_{RH}}{N}\pi_M + \frac{P_{RH}}{N}\pi_E + \frac{P_{RH}}{N}\pi_S + \frac{P_{RM}}{N}\pi_I \quad (5.14)$$

$$\pi_I = \frac{P_{WH}}{N}\pi_M + \frac{P_{WH}}{N}\pi_E + \frac{P_{WH}}{N}\pi_M + \frac{P_{RM}}{N}\pi_I \quad (5.15)$$

State transition probability matrix:
The probabilities of changing from one state to another in a single time unit are given by the state transition probability matrix of a Markov chain.
We know that the transition probability matrix P can be defined from the above equations as:

$$\text{P}=\begin{bmatrix} \frac{(P_{RH}+P_{WH})}{N} & 0 & \frac{P_{RH}}{N} & \frac{P_{WH}}{N} \\ \frac{P_{WH}}{N} & \frac{P_H}{N} & \frac{P_{RH}}{N} & \frac{P_{WH}}{N} \\ \frac{P_{RH}}{N} & 0 & \frac{P_{RH}}{N} & \frac{P_{WH}}{N} \\ \frac{P_{WH}}{N}(N-1) & \frac{P_{RM}}{N}(N-1) & \frac{P_{RM}}{N}N-1) & \frac{P_{RM}}{N} \end{bmatrix}$$

With the notion that sum of all the probabilities is equal to 1:

$$\sum_i \pi_i = 1$$

$$i.e \quad \pi_M + \pi_E + \pi_S + \pi_I = 1 \quad (5.16)$$

Adding the second, third and fourth terms in equation 5.1:

$$\pi_M = \frac{P_{RH}}{N}\pi_M + \frac{P_{WH}}{N}(\pi_M + \pi_E + \pi_S) + \frac{P_{WH}}{N}(N-1)\pi_I \tag{5.17}$$

Putting the value obtained eq. 5.5 we get

$$\pi_M = \frac{P_{RH}}{N}\pi_M + \frac{P_{WH}}{N}(1-\pi_I) + \frac{P_{WH}}{N}(N-1)\pi_I \tag{5.18}$$

Similarly the other probabilities can be solved in similar manner and as a result, we get the following equations:

$$\pi_E = \frac{P_{RH}}{N}\pi_E + \frac{P_{RM}}{N}(N-1)\pi_I \tag{5.19}$$

$$\pi_S = \frac{P_{RH}}{N}(1-\pi_I) + \frac{P_{RM}}{N}\pi_I \tag{5.20}$$

$$\pi_I = \frac{P_{WH}}{N}(1-\pi_I) + \frac{P_{RM}}{N}\pi_I \tag{5.21}$$

5.3 Average memory access time:

The average time a processor needs to wait for memory every load or store instruction is known as average memory access time (AMAT). Every access is either a success or a failure. If the cache misses, the processor searches main memory for the data.
Let h_{L1} be the cache hit of Level1 cache(L1), and m_{L1} be the cache miss of L1. We define T_{c1} as Level 1 cache access time upon a cache hit and T_{m1} as the miss penalty of L1 cache i.e after a cache miss, it's time to access the value. As a result, the average memory access time (AMAT) is as follows:

$$\begin{aligned}\text{AMAT of L1 cache} &= (\text{cache hit of L1} \times \text{L1 access time}) \\ &\quad + (\text{cache miss of L1} \times \text{miss penalty of L1}) \\ &= (h_{L1} \times T_{c1}) + (m_{L1} \times T_{m1})\end{aligned} \tag{5.22}$$

Let h_{L2} be the cache hit of Level2 cache(L2), and m_{L2} be the cache miss of L2. We define T_{c2} as L2 cache access time upon a cache hit and T_{m2} as the miss penalty of L2 cache.

$$\begin{aligned}\text{AMAT of L2 cache} &= (\text{cache hit of L2} \times \text{L2 access time}) \\ &+ (\text{cache miss of L2} \times \text{miss penalty of L2}) \\ &= (h_{L2} \times T_{c2}) + (m_{L2} \times T_{m2})\end{aligned} \tag{5.23}$$

$$\begin{aligned}\text{Miss penalty of L1 cache} &= (\text{cache hit of L2} \times \text{L2 access time}) \\ &+ (\text{cache miss of L2} \times \text{miss penalty of L2}) \\ &= (h_{L2} \times T_{c2}) + (m_{L2} \times T_{m2})\end{aligned} \tag{5.24}$$

Similarly let us define h_{L3} as the cache hit of Level3 cache(L3), and m_{L3} be the cache miss of L3. We define T_{c3} as L3 cache access time upon a cache hit and T_{m3} as the miss penalty of L3 cache.

$$\begin{aligned}\text{Miss penalty of L2 cache} &= (\text{cache hit of L3} \times \text{L3 access time}) \\ &+ (\text{cache miss of L3} \times \text{miss penalty of L3}) \\ &= (h_{L3} \times T_{c3}) + (m_{L3} \times T_{m3})\end{aligned} \tag{5.25}$$

$$\text{Miss penalty of L3 cache} = \text{Main memory access time} \tag{5.26}$$

The following equations are derived from MESI protocol and Markov based model as shown below:

$$\text{Cache hit} = \pi_M + \pi_E + \pi_S \tag{5.27}$$

$$\text{Cache miss} = \pi_I \tag{5.28}$$

As said before, AMAT is defined as the hit time + (miss ratio * miss penalty).From the state diagram of MESI protocol as explained above 2.8, we can define AMAT as:

$$\begin{aligned}\text{AMAT of L1 cache} &= ((\pi_M + \pi_E + \pi_S) \times \text{L2 access time}) \\ &+ ((1 - \pi_I) \times (\text{L2 access time} \\ &+ \text{miss penalty of L2 cache}) \times (1 + \pi_M)) \\ &= ((\pi_M + \pi_E + \pi_S) \times T_{c2}) \\ &+((1 - \pi_I) \times (T_{c2} + T_{m2}) \times (1 + \pi_M))\end{aligned} \tag{5.29}$$

$$\begin{aligned}\text{AMAT of L2 cache} &= (\text{cache hit} \times \text{L2 access time}) \\ &+ (\text{cache miss} \times \text{miss penalty of L2} \times (1 + \pi_M)) \\ &= (h_{L2} \times T_{c2}) + (m_{L2} \times T_{m2} \times (1 + \pi_M))\end{aligned} \tag{5.30}$$

5.4 Results of Analytical Modeling

The following results are obtained using the analytical modeling formulas and values obtained using the Sniper simulator.

5.4.1 Blackscholes benchmark with 4 cores:

Probability	**Values**
P_{H1}	0.9824
P_{H2}	0.74355
P_{M}	0.25645
P_{H3}	0.6616
P_{RH}	0.96966
P_{RM}	0.03033
P_{WH}	0.94207
P_{WM}	0.0579

Table 5.1: Probabilities of Blackscholes benchmark with 4 cores

We can reconfigure the state diagram of MESI protocol to arrive at the following Blackscholes benchmark specfic state diagram.

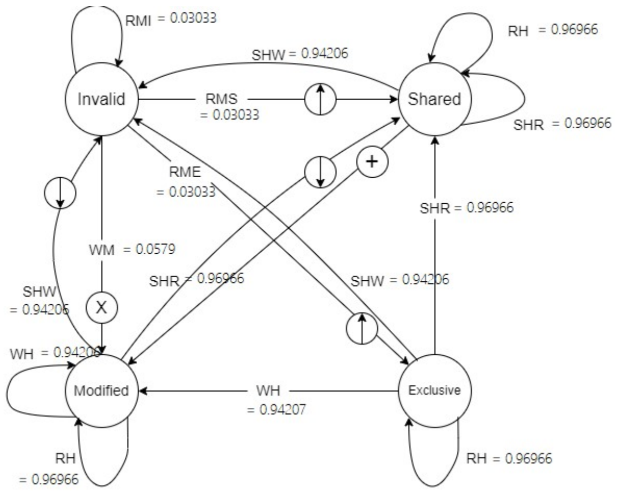

Figure 5.2: State diagram of MESI protocol

And from the values obtained from the above table, we can define State transition probability matrix as:

$$P=\begin{bmatrix} 0.04779 & 0 & 0.24241 & 0.2355175 \\ 0.2355175 & 0.242415 & 0.242415 & 0.2355175 \\ 0.2355175 & 0 & 0.242415 & 0.2355175 \\ 0.7065525 & 0.0227475 & 0.0227475 & 0.0075825 \end{bmatrix}$$

$\pi_M = 0.4962795, \pi_E = 0.05758804, \pi_S = 0.22244855, \pi_I = 0.21864666$
And by solving we get
AMAT of L1 cache using tradtional method = 4.1926
AMAT of L2 cache using tradtional method = 14.9436
AMAT of L2 cache using Markov model = 17.68587

5.4.2 Bodytrack benchmark with 4 cores:

Probability	**values**
P_{H1}	0.7899
P_{H2}	0.6469
P_{M2}	0.35303
P_{H3}	0.5126
P_{RH}	0.9305
P_{RM}	0.069497
P_{WH}	0.5756
P_{WM}	0.4243

Table 5.2: Probabilities of Bodytrack benchmark with 4 cores

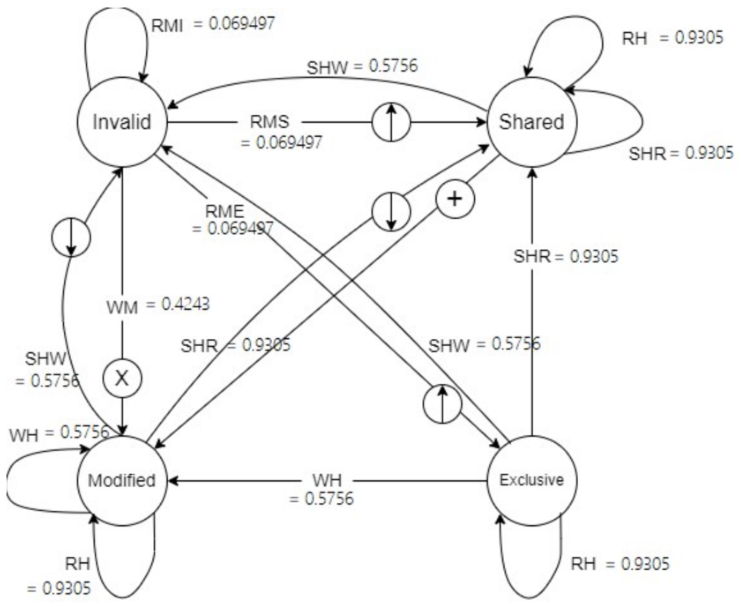

Figure 5.3: State diagram of MESI protocol

State transition probability matrix as:

$$P=\begin{bmatrix} 0.376525 & 0 & 0.232625 & 0.1439 \\ 0.1439 & 0.232625 & 0.232625 & 0.1439 \\ 0.1439 & 0 & 0.232625 & 0.1439 \\ 0.4317 & 0.05212275 & 0.05212275 & 0.01737425 \end{bmatrix}$$

$\pi_M = 0.37713057, \pi_E = 0.13524958, \pi_S = 0.26219889, \pi_I = 0.19535402$
And by solving we get
AMAT of L1 cache using tradtional method = 6.88572
AMAT of L2 cache using tradtional method = 18.349
AMAT of L2 cache using Markov model = 16.234

5.4.3 Cholesky benchmark with 4 cores:

Probability	values
P_{H1}	0.985925
P_{H2}	0.710425
P_{M2}	0,38035
P_{H3}	0.61965
P_{RH}	0.62456
P_{RM}	0.375
P_{WH}	0.16907
P_{WM}	0.8309

Table 5.3: Probabilities of Cholesky benchmark with 4 cores

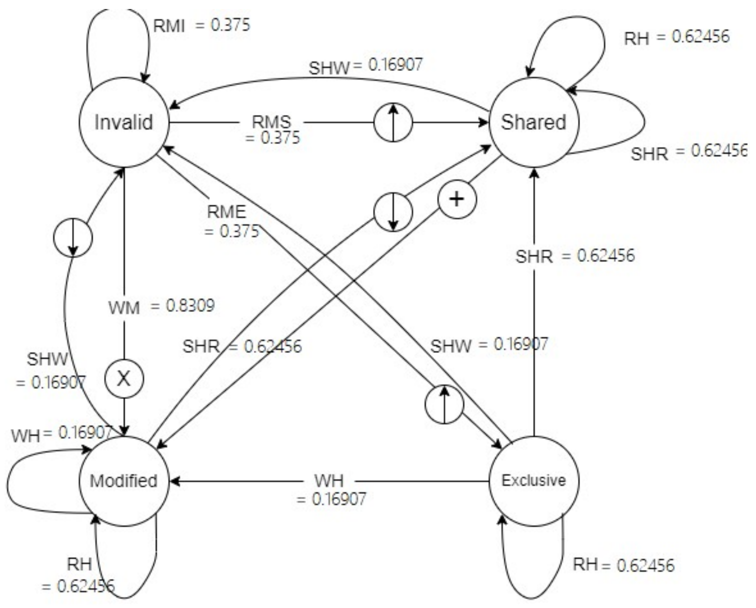

Figure 5.4: State diagram of MESI protocol

State transition probability matrix as:

$$P=\begin{bmatrix} 0.1984075 & 0. & 0.15614 & 0.0422675 \\ 0.0422675 & 0.15614 & 0.15614 & 0.0422675 \\ 0.0422675 & 0. & 0.15614 & 0.0422675 \\ 0.1268025 & 0.28158 & 0.28158 & 0.09386 \end{bmatrix}$$

$\pi_M = 0.212912, \pi_E = 0.217856, \pi_S = 0.274991, \pi_I = 0.21968$
And by solving we get
AMAT of L1 cache using tradtional method = 4.1973
AMAT of L2 cache using tradtional method = 18.0197
AMAT of L2 cache using Markov model = 15.436

5.4.4 FFT benchmark with 4 cores:

Probability	**Values**
P_{H1}	0.7989
P_{H2}	0.04605
P_M	0.95395
P_{H3}	0.5126
P_{RH}	0.01878
P_{RM}	0.98121
P_{WH}	0.000005
P_{WM}	0.9999

Table 5.4: Probabilities of FFT benchmark with 4 cores

State transition probability matrix as:

$$P=\begin{bmatrix} 4.968755e-03 & 0 & 4.967500e-03 & 1.255000e-06 \\ 1.255000e-06 & 4.967500e-03 & 4.967500e-03 & 1.255000e-06 \\ 1.255000e-06 & 0 & 4.967500e-03 & 1.255000e-06 \\ 3.765000e-06 & 7.350975e-01 & 7.350975e-01 & 2.450325e-01 \end{bmatrix}$$

$\pi_M = 0.06834831, \pi_E = 0.28489013, \pi_S = 0.28631409, \pi_I = 0.29311493$

And by solving we get
AMAT of L1 cache using tradtional method = 10.427403
AMAT of L2 cache using tradtional method =35.9612
AMAT of L2 cache using Markov model =16.800316

5.4.5 Radix benchmark with 4 cores:

Probability	**Values**
P_{H1}	0.93875
P_{H2}	0.608725
P_M	0.391275
P_{H3}	0.532
P_{RH}	0.2053
P_{RM}	0.7946
P_{WH}	0.7347
P_{WM}	0.26527

Table 5.5: Probabilities of Radix benchmark with 4 cores

State transition probability matrix as:

$$P= \begin{bmatrix} 0.235 & 0 & 0.0513 & 0.183675 \\ 0.183675 & 0.0513 & 0.0513 & 0.183675 \\ 0.183675 & 0 & 0.0513 & 0.183675 \\ 0.551025 & 0.596025 & 0.596025 & 0.198675 \end{bmatrix}$$

$\pi_M = 0.322678, \pi_E = 0.192718, \pi_S = 0.219027, \pi_I = 0.257908$
And by solving we get
AMAT of L1 cache using tradtional method = 6.1059
AMAT of L2 cache using tradtional method = 18.3567
AMAT of L2 cache using Markov model = 17.631192

5.4.6 Vips benchmark with 4 cores:

Probability	**Values**
P_{H1}	0.963
P_{H2}	0.34155
P_M	0.65845
P_{H3}	0.8598
P_{RH}	0.65028
P_{RM}	0.3497
P_{WH}	0.09711
P_{WM}	0.9022

Table 5.6: Probabilities of Vips benchmark with 4 cores

State transition probability matrix as:

$$P=\begin{bmatrix} 0.1868475 & 0 & 0.16257 & 0.0242775 \\ 0.0242775 & 0.16257 & 0.16257 & 0.024277 \\ 0.0242775 & 0 & 0.16257 & 0.024277 \\ 0.0728325 & 0.26229 & 0.26229 & 0.08743 \end{bmatrix}$$

$\pi_M = 0.197661, \pi_E = 0.224887, \pi_S = 0.28107, \pi_I = 0.210506$
And by solving we get
AMAT of L1 cache using tradtional method = 4.7352
AMAT of L2 cache using tradtional method = 23.8706
AMAT of L2 cache using Markov model = 20.72265

5.4.7 Blackscholes benchmark with 8 cores:

Probability	**Values**
P_{H1}	0.99355
P_{M1}	0.0064
P_{H2}	0.553775
P_{M2}	0.446225
P_{H3}	0.48965
P_{M3}	0.5103
P_{RH}	0.68107
P_{RM}	0.318922
P_{WH}	0.4157
P_{WM}	0.5842

Table 5.7: Probabilities of benchmark with 4 cores

State transition probability matrix as:

$$P = \begin{bmatrix} 0.137096250 & 0.08513375 & 0.0519625 & \\ 0.0519625 & 0.08513375 & 0.08513375 & 0.0519625 \\ 0.0519625 & 0 & 0.08513375 & 0.0519625 \\ 0.3637375 & 0.27906375 & 0.27906375 & 0.03986625 \end{bmatrix}$$

$\pi_M = 0.257944, \pi_E = 0.1972, \pi_S = 0.2289, \pi_I = 0.2356$
And by solving we get
AMAT of L1 cache using tradtional method = 4.111
AMAT of L2 cache using tradtional method = 21.23291
AMAT of L2 cache using Markov model = 16.6379

5.4.8 Bodytrack benchmark with 8 cores:

Probability	**Values**
P_{H1}	0.9620875
P_{M1}	0.4457
P_{H2}	0.83605
P_{M2}	0.16395
P_{H3}	0.920999
P_{M3}	0.0790
P_{RH}	0.91779
P_{RM}	0.08220
P_{WH}	0.5466
P_{WM}	0.4534

Table 5.8: Probabilities of Bodytrack benchmark with 8 cores

State transition probability matrix as:

$$\text{P}=\begin{bmatrix} 0.18304875 & 0 & 0.11472375 & 0.068325 \\ 0.068325 & 0.11472375 & 0.11472375 & 0.068325 \\ 0.068325 & 0 & 0.11472375 & 0.068325 \\ 0.478275 & 0.07193375 & 0.07193375 & 0.01027625 \end{bmatrix}$$

$\pi_M = 0.32035794, \pi_E = 0.16702118, \pi_S = 0.21303608, \pi_I = 0.21629485$
And by solving we get
AMAT of L1 cache using tradtional method $= 4.29576$
AMAT of L2 cache using tradtional method $= 12.80120$
AMAT of L2 cache using Markov model $= 14.50938$

5.4.9 Cholesky benchmark with 8 cores:

Probability	**Values**
P_{H1}	0.987375
P_{M1}	0.01262
P_{H2}	0.578875
P_{M2}	0.421124
P_{H3}	0.6535
P_{M3}	0.3465
P_{RH}	0.5843
P_{RM}	0.4157
P_{WH}	0.1851
P_{WM}	0.8149

Table 5.9: Probabilities of Cholesky benchmark with 8 cores

State transition probability matrix as:

$$P=\begin{bmatrix} 0.096175 & 0 & 0.0730375 & 0.0231375 \\ 0.0231375 & 0.0730375 & 0.0730375 & 0.0231375 \\ 0.0231375 & 0 & 0.0730375 & 0.0231375 \\ 0.1619625 & 0.3637375 & 0.3637375 & 0.0519625 \end{bmatrix}$$

$\pi_M = 0.18951681, \pi_E = 0.22525884, \pi_S = 0.24837293, \pi_I = 0.24024348$
And by solving we get
AMAT of L1 cache using tradtional method = 4.19510
AMAT of L2 cache using tradtional method = 19.4535
AMAT of L2 cache using Markov model = 16.36370

5.4.10 FFT benchmark with 8 cores:

Probability	Values
P_{H1}	0.98
P_{M1}	0.02
P_{H2}	0.02707
P_{M2}	0.97293
P_{H3}	0.5759875
P_{M3}	0.420125
P_{RH}	0.039170
P_{RM}	0.96083
P_{WH}	9.99e-05
P_{WM}	0.9999

Table 5.10: Probabilities of FFT benchmark with 8 cores

State transition probability matrix as:

$$P=\begin{bmatrix} 4.9087375e-03 & 0 & 4.8962500e-03 & 1.2487500e-05 \\ 1.2487500e-05 & 4.8962500e-03 & 4.8962500e-03 & 1.2487500e-05 \\ 1.2487500e-05 & 0 & 4.8962500e-03 & 1.2487500e-05 \\ 8.7412500e-05 & 8.4072625e-01 & 8.4072625e-01 & 1.2010375e-01 \end{bmatrix}$$

$\pi_M = 0.07563127, \pi_E = 0.2946039, \pi_S = 0.29605542, \pi_I = 0.259218$
And by solving we get
AMAT of L1 cache using tradtional method = 4.6318
AMAT of L2 cache using tradtional method = 35.592
AMAT of L2 cache using Markov model = 15.468

5.4.11 Radix benchmark with 8 cores:

Probability	**Values**
P_{H1}	0.9395125
P_{M1}	0.060487
P_{H2}	0.6526375
P_{M2}	0.3473625
P_{H3}	0.3884
P_{M3}	0.611525
P_{RH}	0.5035
P_{RM}	0.4965
P_{WH}	0.6976
P_{WM}	0.3024

Table 5.11: Probabilities of Radix benchmark with 8 cores

State transition probability matrix as:

$$\text{P}=\begin{bmatrix} 0.1501375 & 0 & 0.0629375 & 0.0872 \\ 0.0872 & 0.0629375 & 0.0629375 & 0.0872 \\ 0.0872 & 0 & 0.0629375 & 0.0872 \\ 0.6104 & 0.4344375 & 0.4344375 & 0.0620625 \end{bmatrix}$$

$\pi_M = 0.303157, \pi_E = 0.19146, \pi_S = 0.21966, \pi_I = 0.240$
And by solving we get
AMAT of L1 cache using tradtional method = 4.8969
AMAT of L2 cache using tradtional method = 18.8289
AMAT of L2 cache using Markov model = 18.009

5.4.12 Blackscholes benchmark with 12 cores:

Probability	**Values**
P_{H1}	0.99625
P_{M1}	0.375
P_{H2}	0.34193
P_{M2}	0.6580
P_{H3}	0.493467
P_{M3}	0.506533
P_{RH}	0.5405
P_{RM}	0.4595
P_{WH}	0.267
P_{WM}	0.733

Table 5.12: Probabilities 0f Blackscholes benchmark with 12 cores

State transition probability matrix as:

$$\text{P=}\begin{bmatrix} 0.06729167 & 0 & 0.04504167 & 0.02225 & \\ 0.02225 & 0.04504167 & 0.04504167 & 0.02225 & \\ 0.02225 & 0 & & 0.04504167 & 0.02225 \\ 0.24475 & 0.42120833 & 0.42120833 & 0.03829167 & \end{bmatrix}$$

$\pi_M = 0.195087, \pi_E = 0.2251, \pi_S = 0.2397, \pi_I = 0.24585$
And by solving we get
AMAT of L1 cache using tradtional method = 4.08804
AMAT of L2 cache using tradtional method = 27.477
AMAT of L2 cache using Markov model = 16.3262

5.4.13 Bodytrack benchmark with 12 cores:

Probability	**Values**
P_{H1}	0.9586
P_{M1}	0.041341
P_{H2}	0.86572
P_{M2}	0.134275
P_{H3}	0.9009
P_{M3}	0.0990
P_{RH}	0.903
P_{RM}	0.097
P_{WH}	0.536
P_{WM}	0.464

Table 5.13: Probabilities of Bodytrack benchmark with 12 cores

State transition probability matrix as:

$$P=\begin{bmatrix} 0.11991667 & 0 & 0.07525 & 0.04466667 \\ 0.04466667 & 0.07525 & 0.07525 & 0.04466667 \\ 0.04466667 & 0 & 0.07525 & 0.04466667 \\ 0.49133333 & 0.08891667 & 0.08891667 & 0.00808333 \end{bmatrix}$$

$\pi_M = 0.301768, \pi_E = 0.17054, \pi_S = 0.19778, \pi_I = 0.22717$
And by solving we get
AMAT of L1 cache using tradtional method = 4.2961
AMAT of L2 cache using tradtional method = 11.153
AMAT of L2 cache using Markov model = 14.672

5.4.14 Cholesky benchmark with 12 cores:

Probability	**Values**
P_{H1}	0.987709
P_{M1}	0.012291
P_{H2}	0.554325
P_{M2}	0.445675
P_{H3}	0.6158584
P_{M3}	0.3841416
P_{RH}	0.5637
P_{RM}	0.4363
P_{WH}	0.1922
P_{WM}	0.8078

Table 5.14: Probabilities of Cholesky benchmark with 12 cores

State transition probability matrix as:

$$P=\begin{bmatrix} 0.06299167 & 0 & 0.046975 & 0.01601667 \\ 0.01601667 & 0.046975 & 0.046975 & 0.01601667 \\ 0.01601667 & 0 & 0.046975 & 0.01601667 \\ 0.17618333 & 0.39994167 & 0.39994167 & 0.03635833 \end{bmatrix}$$

$\pi_M = 0.1815, \pi_E = 0.2281, \pi_S = 0.2424, \pi_I = 0.2448$
And by solving we get
AMAT of L1 cache using tradtional method = 4.20124
AMAT of L2 cache using tradtional method = 20.3732
AMAT of L2 cache using Markov model = 15.565

5.4.15 Blackscholes benchmark with 16 cores:

Probability	**Values**
P_{H1}	0.99735
P_{M1}	0.00265
P_{H2}	0.2368
P_{M2}	0.7631
P_{H3}	0.5396
P_{M3}	0.460368
P_{RH}	0.454
P_{RM}	0.545
P_{WH}	0.153
P_{WM}	0.847

Table 5.15: Probabilities of Blackscholes benchmark with 16 cores

State transition probability matrix as:

$$P=\begin{bmatrix} 0.0379375 & 0 & 0.028375 & 0.0095625 \\ 0.0095625 & 0.028375 & 0.028375 & 0.0095625 \\ 0.0095625 & 0 & 0.028375 & 0.0095625 \\ 0.1434375 & 0.511875 & 0.511875 & 0.034125 \end{bmatrix}$$

$\pi_M = 0.155682, \pi_E = 0.24358, \pi_S = 0.25205, \pi_I = 0.248718$
And by solving we get
AMAT of L1 cache using tradtional method = 4.069
AMAT of L2 cache using tradtional method = 30.0610
AMAT of L2 cache using Markov model = 15.818

5.4.16 Bodytrack benchmark with 16 cores:

Probability	**Values**
P_{H1}	0.9572375
P_{M1}	0.04276
P_{H2}	0.8815
P_{M2}	0.1185
P_{H3}	0.8822
P_{M3}	0.1178
P_{RH}	0.9261
P_{RM}	0.0739
P_{WH}	0.5378
P_{WM}	0.4622

Table 5.16: Probabilities of Bodytrack benchmark with 16 cores

State transition probability matrix as:

$$P=\begin{bmatrix} 0.09149375 & 0 & 0.05788125 & 0.0336125 \\ 0.0336125 & 0.05788125 & 0.05788125 & 0.0336125 \\ 0.0336125 & 0 & 0.05788125 & 0.0336125 \\ 0.5041875 & 0.06928125 & 0.06928125 & 0.00461875 \end{bmatrix}$$

$\pi_M = 0.29807, \pi_E = 0.16845, \pi_S = 0.1883, \pi_I = 0.23017$
And by solving we get
AMAT of L1 cache using tradtional method = 4.29174
AMAT of L2 cache using tradtional method = 10.8163
AMAT of L2 cache using Markov model = 14.341

5.4.17 Cholesky benchmark with 16 cores:

Probability	**Values**
P_{H1}	0.98834
P_{M1}	0.1166
P_{H2}	0.5378
P_{M2}	0.4622
P_{H3}	0.5961
P_{M3}	0.4039
P_{RH}	0.5462
P_{RM}	0.4538
P_{WH}	0.2029
P_{WM}	0.7974

Table 5.17: Probabilities of Cholesky benchmark with 16 cores

State transition probability matrix as:

$$P=\begin{bmatrix} 0.04681875 & 0. & 0.0341375 & 0.01268125 \\ 0.01268125 & 0.0341375 & 0.0341375 & 0.01268125 \\ 0.01268125 & 0. & 0.0341375 & 0.01268125 \\ 0.19021875 & 0.4254375 & 0.4254375 & 0.0283625 \end{bmatrix}$$

$\pi_M = 0.178325, \pi_E = 0.2299, \pi_S = 0.2401, \pi_I = 0.2467$
And by solving we get
AMAT of L1 cache using tradtional method = 4.1978
AMAT of L2 cache using tradtional method = 20.9686
AMAT of L2 cache using Markov model = 17.672

5.4.18 FFT benchmark with 16 cores:

Probability	**Values**
P_{H1}	0.9797
P_{M1}	0.0203
P_{H2}	0.07598
P_{M2}	0.92401
P_{H3}	0.43951
P_{M3}	0.56049
P_{RH}	0.1091
P_{RM}	0.8909
P_{WH}	0.0001
P_{WM}	0.999

Table 5.18: Probabilities of FFT benchmark with 16 cores

State transition probability matrix as:

$$P= \begin{bmatrix} 6.8250000e-03 & 0 & 6.8187500e-03 & 6.2500000e-06 \\ 6.2500000e-06 & 6.8187500e-03 & 6.8187500e-03 & 6.2500000e-06 \\ 6.2500000e-06 & 06.8187500e-03 & 6.2500000e-06 & \\ 9.3750000e-05 & 8.3521875e-01 & 8.3521875e-01 & 5.5681250e-02 \end{bmatrix}$$

$\pi_M = 0.083897, \pi_E = 0.292136, \pi_S = 0.29414, \pi_I = 0.2476$
And by solving we get
AMAT of L1 cache using tradtional method = 4.65162
AMAT of L2 cache using tradtional method = 36.0995
AMAT of L2 cache using Markov model = 15.6713

5.4.19 Radix benchmark with 16 cores:

Probability	**Values**
P_{H1}	0.9402625
P_{M1}	0.0597375
P_{H2}	0.574975
P_{M2}	0.425025
P_{H3}	0.3433
P_{M3}	0.6567
P_{RH}	0.5835
P_{RM}	0.4165
P_{WH}	0.5568
P_{WM}	0.4432

Table 5.19: Probabilities of Radix benchmark with 16 cores

State transition probability matrix as:

$$P= \begin{bmatrix} 0.07126875 & 0. & 0.03646875 & 0.0348 \\ 0.0348 & 0.03646875 & 0.03646875 & 0.0348 \\ 0.0348 & 0. & 0.03646875 & 0.0348 \\ 0.522 & 0.39046875 & 0.39046875 & 0.02603125 \end{bmatrix}$$

$\pi_M = 0.259015, \pi_E = 0.20135, \pi_S = 0.21503, \pi_I = 0.243700$
And by solving we get
AMAT of L1 cache using tradtional method = 5.04781
AMAT of L2 cache using tradtional method = 21.5396
AMAT of L2 cache using Markov model = 17.6302

5.5 Comparison of AMAT for different benchmarks:

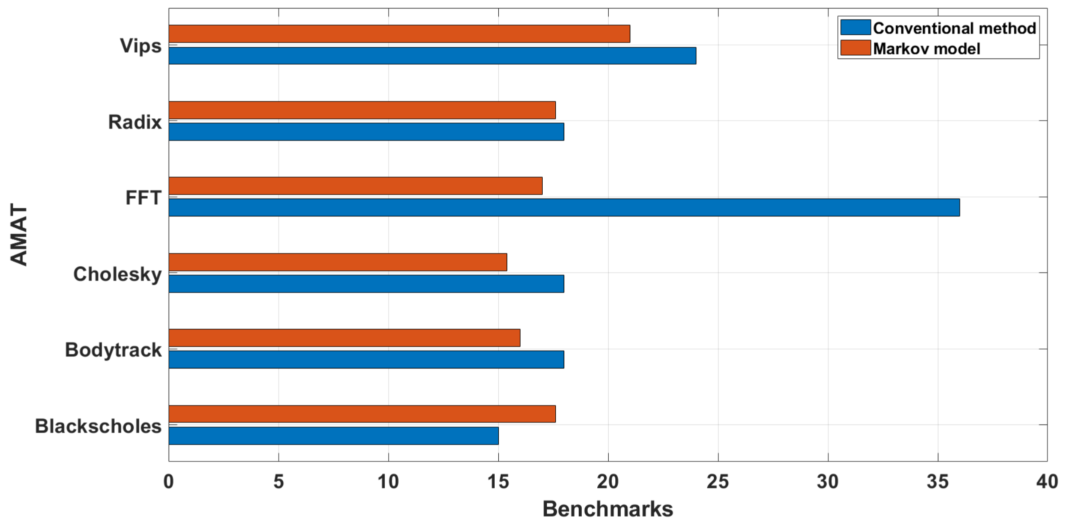

Figure 5.5: AMAT of different benchmarks of core 4

Markov Model is a new approach used to calculate AMAT. The above plot very closely compares the Markov Model with the conventional model. From the previous chapter, we have observed that the FFT benchmark has higher L2 miss rate while it is less in Markov model. So Markov model is applicable for all the benchmarks but FFT has deviated a little.

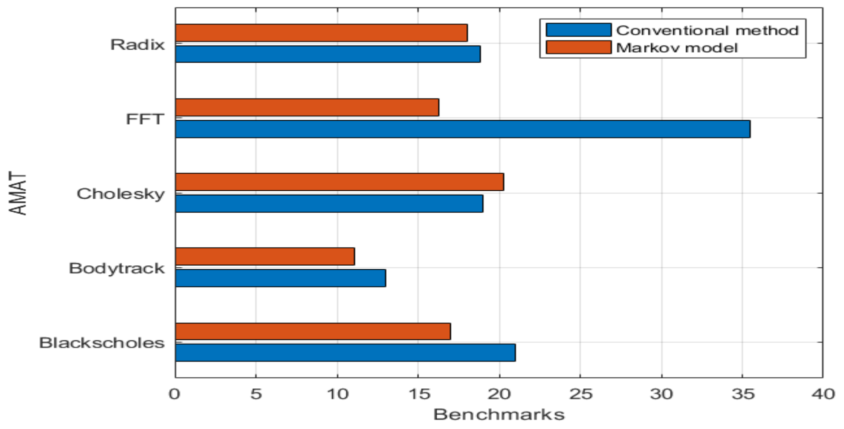

Figure 5.6: AMAT of different benchmarks of core 8

When we go from core 4 to core 8, the Markov Model still justifies except for the FFT benchmark. This can be seen from the 5.6.

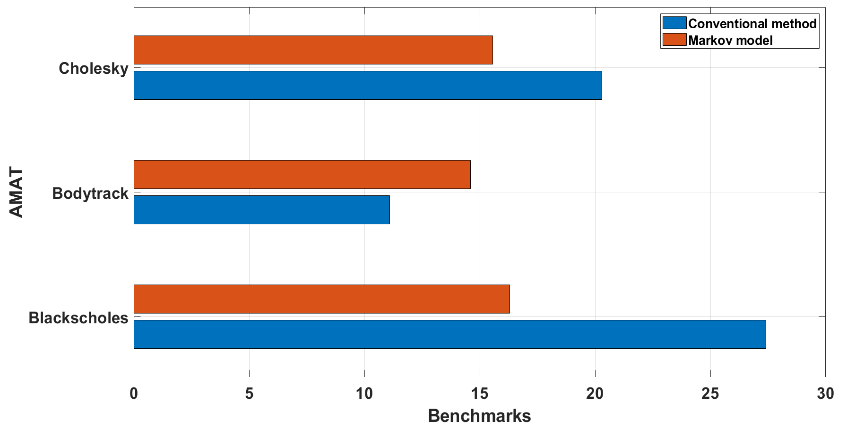

Figure 5.7: AMAT of different benchmarks of core 12

As we have seen in the previous simulation results, as core increases, the miss rate of blackscholes increases. So that's the reason there is a difference between the AMAT obtained using Conventional method and Markov model whereas the Markov model is justified for other benchmarks.

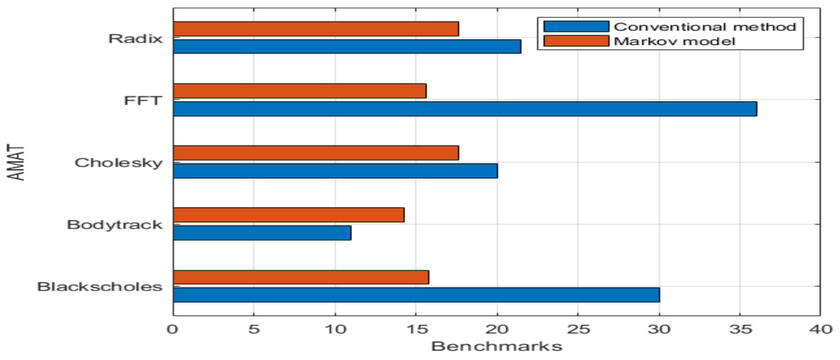

Figure 5.8: AMAT of different benchmarks of core 16

When the core increases from core 8 to core 16, the blackscholes miss rate increases and FFT has higher benchmark for all the cores of L2 cache. So the markov model is deviated a little here for blackscholes and FFT benchmarks whereas it is justified for all other benchmarks.

Chapter 6

Conclusion

A number of conclusions can be drawn from the host of simulations that we have discussed in the report. We have checked the hit and miss rates of various benchmarks (Blackscholes, Bodytrack, Cholesky, FFT, Radix, Vips) of different cache coherence protocols (MSI, MESI, MESIF) of different cores sizes (4, 8, 12, 16), associativity and cache sizes. This led us to the conclusion that FFT has the highest miss rate for L2 and L3 caches compared to the other benchmarks. As we move higher from core 4 to core 16, the miss rates of all benchmarks remain more or less the same, giving us the freedom to use higher core processors in the day to day operations when available.

When we increase the associativity of L1 from 4 to 16, the miss rates of MSI and MESI decrease. The remaining two caches (L2 and L3) exhibit no considerable change. So, we can go for MESI protocol over MSI as it has various other notable advantages. When we increase the cache size, the miss rate of MESI decreases marginally compared to MSI. This shows that MESI is better than MSI as we increase the cache size.

Our main aim was to calculate the AMAT using Markov Model and compare it with the conventional method. The Markov Model is felt appropriate for all the benchmarks except for FFT, as it has deviated a little. So the results shows that for calculating AMAT, Markov model can be used.

Bibliography

[1] A. D. Joshi and N. Ramasubramanian, "Comparison of significant issues in multicore cache coherence," in *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, 2015, pp. 108–112.

[2] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[3] H. Shukur, S. Zeebaree, R. Zebari, O. Ahmed, L. Haji, and D. Abdulqader, "Cache coherence protocols in distributed systems," *Journal of Applied Science and Technology Trends*, vol. 1, no. 3, pp. 92–97, 2020.

[4] Z. Al-Waisi and M. O. Agyeman, "An overview of on-chip cache coherence protocols," in *2017 Intelligent Systems Conference (IntelliSys)*. IEEE, 2017, pp. 304–309.

[5] S. Kumar and P. K. Singh, "An overview of modern cache memory and performance analysis of replacement policies," in *2016 IEEE International Conference on Engineering and Technology (ICETECH)*, 2016, pp. 210–214.

[6] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008, pp. 72–81.

[7] D. Sam and M. O. Agyeman, "An overview of design space exploration of cache memory," in *Proceedings of the 2nd International Symposium on Computer Science and Intelligent Control*, 2018, pp. 1–6.

[8] G. Jang and J.-L. Gaudiot, “Data shepherding: A last level cache design for large scale chips,” in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019, pp. 1920–1927.

[9] S. Sun, H. An, and J. Chen, “Cache coherence method for improving multi-threaded applications on multicore systems,” in *2014 6th International Conference on Multimedia, Computer Graphics and Broadcasting.* IEEE, 2014, pp. 47–50.

[10] A. Tiwari, “Performance Comparison of Cache Coherence Protocol on Multi-Core Architecture,” NIT Rourkela, Dissertation 1, 2014, Project Report.

[11] M. Hill and A. Smith, “Evaluating associativity in cpu caches,” *IEEE Transactions on Computers*, vol. 38, no. 12, pp. 1612–1630, 1989.

[12] M. E. Thomadakis, “The architecture of the nehalem processor and nehalem-ep smp platforms,” *Resource*, vol. 3, no. 2, pp. 30–32, 2011.

[13] S. Mittal and Nitin, “A new approach to directory based solution for cache coherence problem,” in *2014 3rd International Conference on Eco-friendly Computing and Communication Systems*, 2014, pp. 9–13.

[14] T. E. Carlson, W. Heirman, and L. Eeckhout, “Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation,” in *SC ’11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–12.

[15] C. Bienia, *Benchmarking modern multiprocessors.* Princeton University, 2011.

[16] E. T. J. P. S. Steven Cameron Woo, Moriyoshi Oharat, “The SPLASH-2 Programs: Characterization and Methodological Considerations,” Princeton University, Dissertation 1, 2008, Technical Report.

[17] S. Christian Bienia, Jaswinder and K. Li, “PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites

on Chip-Multiprocessors," Princeton University, Dissertation 1, 2008, Technical Report.

[18] X. Zhang, H. Wang, and Y. Lv, "State transition matrix of linear time varying fractional order systems," in *2017 29th Chinese Control And Decision Conference (CCDC)*, 2017, pp. 7857–7861.

[19] M. Sun, Y. Wang, G. Li, and W. Zhang, "Stability and stabilization of discrete-time markov jump singular systems with general uncertain transition rates," in *2016 Chinese Control and Decision Conference (CCDC)*, 2016, pp. 1819–1823.

[20] J. M. Sabarimuthu and T. Venkatesh, "Analytical miss rate calculation of l2 cache from the rd profile of l1 cache," *IEEE Transactions on Computers*, vol. 67, no. 1, pp. 9–15, 2017.

[21] G. Balakrishnan and A. Krishna, "Optimizing a cache back invalidation policy," Jan. 29 2013, uS Patent 8,364,898.

[22] M. Brehob and R. Enbody, "An analytical model of locality and caching," *Michigan State University, Department of Computer Science and Engineering MSU-CSE-99-31*, 1999.

[23] J. Harper, D. Kerbyson, and G. Nudd, "Analytical modeling of set-associative cache behavior," *IEEE Transactions on Computers*, vol. 48, no. 10, pp. 1009–1024, 1999.

[24] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.

[25] T. V. Vincent P.Heuring, Harry F.Jordan, *Computer Systems Design and Architecture*, 2nd ed. USA: Pearson, 2008.