

**Design of Energy Efficient and Deterministic
Memory Subsystem for Uniprocessor and
Multicore Systems**

THESIS

Submitted in partial fulfillment of the requirements for the
degree of

DOCTOR OF PHILOSOPHY

by

PATIL GEETA NAYAKAPPA

Under the Supervision of

Dr. BIJU K. RAVEENDRAN

and Co-supervision of

Dr. LUCY J. GUDINO



BITS Pilani
Pilani|Dubai|Goa|Hyderabad

**BIRLA INSTITUTE OF TECHNOLOGY AND
SCIENCE, PILANI**

2018

Declaration

I, Geeta Patil, hereby declare that the work presented here in the thesis titled, 'Design of Energy Efficient and Deterministic Memory Subsystem for Uniprocessor and Multicore Systems' is a bonafide research work done by me and it has not been submitted previously in part or in full to this University or any other University or Institution for award of any degree. Any literature work cited within this thesis has given due acknowledgement and is listed in bibliography.

Signature of the student :



Name of the student : **PATIL GEETA NAYAKAPPA**

ID number of the student : **2011PHXF0409G**


Date


: **08/12/2018**

**BIRLA INSTITUTE OF TECHNOLOGY AND
SCIENCE, PILANI**

Certificate

This is to certify that the thesis entitled 'Design of Energy Efficient and Deterministic Memory Subsystem for Uniprocessor and Multicore Systems' and submitted by **PATIL GEETA NAYAKAPPA**, ID.No. **2011PHXF0409G** for award of Ph.D. of the Institute embodies original work done by her under our supervision.

Signature of the Supervisor : 
Name in capital letters : **Dr. BIJU K. RAVEENDRAN**
Designation : **ASSISTANT PROFESSOR**
Date :

Signature of the Co-supervisor : 
Name in capital letters : **Dr. LUCY J. GUDINO**
Designation : **ASSISTANT PROFESSOR**
Date : **8/12/2018**

*Dedicated to
God,
My wonderful huge family
&
My best Supervisors.*

Acknowledgement

I would like to acknowledge my sincere thanks to each and everyone who have made this thesis an unforgettable experience and who have influenced this work in some way or the other. Let me begin by thanking the Almighty God for the innumerable blessings he has showered upon me.

Next, I would like to express my deepest gratitude to my supervisor, Dr. Biju K. Raveendran, for giving me an opportunity to work under his guidance. His patience, especially during the initial stages of my PhD, his vast knowledge, excellent supervision, prompt attention, dedicated help, advice, inspiration, encouragement and continuous support have made a deep impression on me.

I have been extremely lucky to have Dr. Lucy J. Gudino as my co-supervisor. I have learnt the basics of computer architecture from her motivational classes. She has provided her valuable guidance and consistent support throughout my research work. Her suggestions in completing my thesis are highly appreciated.

I would like to thank the members of Doctoral Advisory Committee, Prof. Bharat Deshpande and Prof. Neena Goveas, for their valuable time, guidance, critical suggestions and comments for overall improvement of research work.

I am grateful to Prof. Souvik Bhattacharyya, Vice-Chancellor, BITS Pilani, Prof. G. Raghurama, Director, BITS-Pilani, K. K. Birla Goa Campus, Late. Prof. S. K. Agarwal, former Director, BITS-Pilani, K. K. Birla Goa Campus, Prof. S. K. Verma, Dean, Academic Research Division, BITS-Pilani, Prof. P. K. Das, former Associate Dean, Academic Research Division, BITS-Pilani K. K. Birla Goa Campus, Prof. Bharat Deshpande, Associate Dean, Academic Research Division, BITS-Pilani K. K. Birla Goa Campus and the members of Doctoral Research Committee of Department of CS&IS, BITS-Pilani, K. K. Birla Goa Campus for providing administrative support, a conducive atmosphere and adequate facilities to carry out my research efficiently.

I am thankful to any time help of Dr. Ramprasad Joshi, and his initiation in introducing Latex software. A special mention of thanks to my friends in BITS-Goa Aruna Govada, Shubhangi Gawali, Sreejith V, Rajendra Kumar Roul, Mahadev Gawas, Ashu Sharma, Shamanth N. and many more for their timely help, constant support and cooperation.

I acknowledge my students Vijay, Neethu, Manali, Divya, Alen, Kajal, Samriti and Ayushi who made this journey so wonderful and who were ever willing to assist me. On this occasion I cannot forget my past teachers whose teaching at different stages of education has made it possible for me to reach a stage where I could write this thesis.

I owe my deepest gratitude towards my loving husband, Amol, for his eternal support and aspirations. His love and support has always been my strength. His patience and sacrifice will remain my inspiration throughout my life. Without his help, I would not have been able to complete much of what I have done and become who I am. I thank my son Pranil and daughters Sara, Meenali and Manushri for their understanding and support throughout this work. I am thankful to my mother Sukanti, father Nayakappa, mother in law Jayashree, father in law Jayawant, brother Amol, brother in law Sachin, co-sisters Siya, Sangeeta, and my entire family for believing in me and giving me space to explore the world.

Geeta

Abstract

Energy efficiency is one of the major design considerations of the modern day processor design. Memory subsystem consumes major portion of the on-chip energy. This motivates the designers to come up with cache memory subsystem design with least possible energy consumption without much performance degradation. System performance can be improved by increasing operating frequency of the system. However, increase in operating frequency leads to increase in energy consumption which in turn leads to increase in heat dissipation and leakage current. One of the possible solutions to this problem is to go for multicore systems with reduced frequency. Maintaining data consistency becomes a major challenge in multicore systems. Energy efficient and performance centric protocols are required to maintain data consistently in these system.

In hard real-time embedded systems, along with energy efficiency, deterministic tighter upper bound on the worst case execution time of the task is also a requirement. Deterministic tighter upper bound on worst case execution time can only be ensured by making the entire process of accessing memory system deterministic. The memory access model can be made deterministic by providing a hard upper bound on the number of misses in TLB, L1 Cache, L2 cache and main memory. This thesis addresses static energy consumption and dynamic energy consumption of uncore and multicore system. The thesis also addresses mechanisms to provide tighter upper bound on worst case execution time on memory sub-system performance in order to achieve deterministic memory performance.

To reduce energy consumption and response time of set associative caches, the thesis proposes a novel cache architecture - Way Halted Prediction. This is

achieved with the help of halt tag array and prediction circuit. Experimental evaluation of various SPLASH benchmark programs on SESC simulator reveal that way halted prediction architecture offers better energy efficiency over the other architectures analyzed. Way halted prediction offers 46.64%, 6.45% and 4.15% dynamic energy saving and 1.04%, 2.92% and -0.05% saving in response time over the CC, WP and WH respectively.

To reduce energy consumption and response time of multicore systems, the thesis proposes a novel cache coherence protocol **Modified Owned Exclusive Shared Invalid Forward** - MOESIF - to improve the off chip and on chip bandwidth usage for multicore systems. This is achieved by reducing the number of write backs to next level memory and by reducing the number of responders to a cache miss when multiple copies of data exists in private caches. Reduction in the number of write backs and the number of responders results in reducing time, energy and bandwidth usage. Experimental evaluation of various SPLASH benchmark programs on SESC simulators reveal that the MOESIF protocol outperforms all other hardware based coherence protocols in terms of energy consumption and response time. The energy savings of MOESIF protocol over MESI, MOESI and MESIF protocol is 88.58%, 4.33% and 88.52% respectively. The per response time saving of MOESIF protocol over MESI, MOESI and MESIF protocol is 91.37%, 6.17% and 91.32% respectively.

The thesis proposes a novel TLB architecture - **Deterministic Translation Lookaside Buffer (DTLB)**- to offer tighter upper bound on the worst case execution time . DTLB offers deterministic performance for low priority real-time tasks. DTLB achieves a tighter upper bound on the worst case execution time of real-time tasks by maintaining a copy of the current TLB in PCB of the task before preemption and transferring the contents back to TLB

while resumption of the task. DTLB reduces TLB response time, dynamic energy consumption and effective per response time by increasing TLB hit rate. TLB hit rate is increased by 9.46% as compared to conventional TLB for 4KB page size, with 16 preemptions and 32 TLB entries. DTLB offers on an average 6.74% of dynamic energy savings over conventional TLB. Effective per response time of DTLB reduced by 2.97% as compared to conventional TLB.

To have a tighter upper bound on the worst case execution time of real time task, the thesis presents a Deterministic Energy efficient process Aware Real-time Cache (DEARCACHE). DEARCACHE ensures deterministic tighter upper bound by eliminating cache related intertask interference. It allocates at least statically identified minimum ways to each job. It obtains tighter upper bound on number of cache misses. DEARCACHE reduces dynamic energy consumption by 38.10% for 4-way set associative cache configuration over CC with 4.39% overhead of static energy. Per response time of DEARCACHE is improved 3.87 times over NO CACHE model and with an additional requirement of 4.37% of per response time as compared to CC.

To get deterministic performance of L2 cache, this thesis proposes a Deterministic Energy Efficient Process aware (DEEP) design. DEEP cache is shared among all tasks running on different cores of the processor. It allocates minimum number of ways to each task which is identified as a result of static analysis. The performance of DEEP cache can be improved by using a shared way - DEEPS. On an average per response time, per access dynamic energy and per access static energy of DEEP is higher than conventional cache by 10.48%, 2.13%, 15.78% respectively. The per response time, per access dynamic energy and per access static energy of CC is higher than DEEPS by 38.50%, 71.69% and 50.01% respectively.

The thesis proposes an integrated design of deterministic memory named as Deterministic REAL-time Memory system (DREAM). DREAM achieves deterministic performance at TLB and L1 cache by incorporating DTLB and DEARCACHE with DEEP as L2 cache. The per response time, per access dynamic energy and per access static energy of conventional cache is higher than DREAM by 27.85%, 71.40% and 46.75% respectively.

Contents

List of Figures	xvi
List of Tables	xxiii
List of Abbreviations	xxiv
1 Introduction	1
1.1 Background	1
1.2 Motivation	4
1.3 Problem Statement	8
1.4 Research Goals	10
1.5 Contributions	10
1.6 Publications	12
1.7 Thesis Outline	13
2 Literature Survey	15
2.1 Introduction	15
2.2 Uniprocessor / Unicore energy optimisations	15
2.3 Cache Coherency Protocols	22
2.4 Deterministic Memory	27
2.4.1 Deterministic TLB	27
2.4.2 Deterministic Cache	30
3 WHP:Way Halted Prediction Cache	33

3.1	Introduction	33
3.2	WHP cache architecture	34
3.3	Energy Model	40
3.3.1	Conventional Cache	40
3.3.2	Way Predicting Cache	43
3.3.3	Way Halting Cache	45
3.3.4	Way Halted Prediction Cache	47
3.4	Time Model	52
3.4.1	Conventional Cache	53
3.4.2	Way Predicting Cache	54
3.4.3	Way Halting Cache	54
3.4.4	Way Halted Prediction Cache	55
3.5	Experimental Setup	56
3.6	Experimental Analysis	57
3.6.1	Prediction Hit Accuracy	58
3.6.2	Dynamic Energy per Access	58
3.6.3	Response Time	62
3.6.4	Static Energy Per Access	71
3.6.5	Time and Area Overhead	73
3.7	Conclusion	73
4	MOESIF : Cache Coherency Protocol	74
4.1	Introduction	74
4.2	Widely Used Cache Coherence Protocols	75
4.2.1	MESI Protocol	76
4.2.2	MOESI Protocol	78
4.2.3	MESIF Protocol	78

4.3	MOESIF Architecture	81
4.4	Energy and Time Model	87
4.5	Experimental Evaluation	90
4.5.1	Experimental Setup	90
4.5.2	Experimental Analysis of Protocols	90
4.5.3	Experimental Evaluation	92
4.5.3.1	Hit rate and Data transfers	92
4.5.3.2	Energy Consumption	95
4.5.3.3	Response Time	98
4.6	Conclusion	100
5	DTLB: Deterministic TLB for Real-time System	101
5.1	Introduction	101
5.2	DTLB Architecture	102
5.3	Energy and Time Modeling	105
5.3.1	Energy Modeling of TLB	105
5.3.2	Time Modeling of TLB	107
5.4	Experimental Setup And Evaluation	108
5.4.1	Experimental Setup	108
5.4.2	Experimental Evaluation	110
5.4.2.1	TLB Miss Rate	110
5.4.2.2	Dynamic Energy	113
5.4.2.3	Access Time	116
5.5	Conclusion	119
6	DEARCACHE - Deterministic Energy Efficient Process Aware Real-time Cache	121
6.1	Introduction	121

6.2	DEARCACHE Architecture	122
6.2.1	DEARCACHE Energy Modeling	125
6.2.2	DEARCACHE Time Modeling	126
6.3	Experimental Analysis	127
6.3.1	Tighter upper bound on WCET	127
6.3.2	Energy per Access	128
6.3.3	Response Time	132
6.3.4	Energy and Time comparison with energy efficient caches	134
6.4	Conclusion	137
7	DREAM - Deterministic Memory Subsystem	138
7.1	Introduction	138
7.2	DREAM Architecture	139
7.3	Time, Power and Energy Modeling	142
7.3.1	Energy Modeling of DEEP Cache with Shared way - DEEPS	145
7.3.2	Energy Modeling of DREAM system with Shared cache - DREAMS	146
7.3.3	Time Modeling of DEEPS	146
7.3.4	Time Modeling of DREAMS	146
7.4	Performance Evaluation	147
7.4.1	Experimental Analysis - L2 as DEEP Cache	147
7.4.1.1	Dynamic Energy per Access	147
7.4.1.2	Static Energy per Access	150
7.4.1.3	Effective per Access Time	152
7.4.1.4	Energy and Time comparison with energy efficient caches	156

7.4.2	Experimental Analysis - Complete Memory Model . . .	157
7.4.2.1	Dynamic Energy per Access	157
7.4.2.2	Static Energy per Access	158
7.4.2.3	Per Access Time	159
7.5	Conclusion	162
8	Conclusion and Future Directions	163
	Publications	166
	Biographies	167
	Bibliography	170

List of Figures

1.1	Cache hit time performance of various associativities with varying cache size	5
1.2	Cache hit rate performance of various associativities with varying cache size	6
3.1	Way Halted Prediction Cache Architecture	36
3.2	Prediction Circuit	37
3.3	Prediction Rate for a 4 way, 8B line size with varying Data Cache size	57
3.4	Prediction Rate for a 4 way, 8B line size with varying Instruction Cache size	58
3.5	Per access dynamic energy consumption for a 4 way, 8B, 32KB Data Cache with varying benchmark programs	59
3.6	Per access dynamic energy consumption for a 2 way, 8B, 32KB Instruction Cache with varying benchmark programs	60
3.7	Per access dynamic energy consumption for a 4 way, 8B, 32KB cache with varying benchmark programs	61
3.8	Per access dynamic energy savings for a 4 way, 8B, 32KB cache with varying benchmark programs over CC	63
3.9	Per access dynamic energy consumption for a 4 way, 8B line size with varying cache size	64

3.10	Per access dynamic energy consumption for a 16B, 8KB Data Cache with varying associativities	64
3.11	Per access dynamic energy consumption for a 4 way, 8KB cache with varying line Size	65
3.12	Response time for a 4 way, 8B, 32KB Data Cache with varying benchmark programs	66
3.13	Response time for a 2 way, 8B, 32KB Instruction Cache with varying benchmark programs	67
3.14	Response time for a 4 way, 8B, 32KB cache with varying benchmark programs	68
3.15	Response time saving over CC for a 4 way, 8B, 32KB cache with varying benchmark programs	69
3.16	Response time for a 4 way, 8B line size with varying cache size	70
3.17	Response time for a 4 way, 8KB cache with varying line Size .	70
3.18	Response time for a 16B, 8KB Data Cache with varying associativity	71
3.19	Per access static energy consumption for a 4 way, 8B, 32KB cache with varying benchmark programs	72
4.1	MESI Access and Snoop State Transitions	77
4.2	MOESI Access and Snoop State Transitions	79
4.3	MESIF Access and Snoop State Transitions	80
4.4	Quad-core Architecture	82
4.5	MOESIF cache access	83
4.6	MOESIF cache snoop	84
4.7	Design of random generator used for Quad-core Architecture .	85

4.8	Per access hit rate for varying cache sizes with 32B line size and associativity as 4 way	93
4.9	Per access write backs for varying cache sizes with 32B line size and associativity as 4 way	94
4.10	Per access data from L2 for varying cache sizes with 32B line size and associativity as 4 way	94
4.11	Per access data from other L1 for varying cache sizes with 32B line size and associativity as 4 way	95
4.12	Per access energy for varying cache sizes with 32B line size and associativity as 4 way	96
4.13	Per access energy for varying cache line sizes with 8KB cache size and associativity as 4 way	96
4.14	Per access energy for varying number of cores with 8KB cache, 16B line size and associativity as 4 way	97
4.15	Per access time for varying cache sizes with 32B line size and associativity as 4 way	98
4.16	Per access time for varying cache line sizes with 8KB cache size and associativity as 4 way	99
4.17	Per access time for varying number of cores with 8KB cache, 16B line size and associativity as 4 way	99
5.1	Deterministic TLB Architecture	103
5.2	Miss rate performance of DTLB and ASID-TLB with respect to conventional TLB for varying page size with 16 preemptions and 32 TLB entries	110

5.3	Miss rate performance of DTLB and ASID-TLB with respect to conventional TLB for varying preemptions with 4KB page size and 32 TLB entries	111
5.4	Miss rate performance of DTLB and ASID-TLB with respect to conventional TLB for varying TLB entries with 16 preemptions and 4KB page size	113
5.5	Per access dynamic energy with respect to conventional TLB saving for varying page size with 16 preemptions and 32 TLB entries	114
5.6	Per access dynamic energy with respect to conventional TLB saving for varying preemptions with 4KB page size and 32 TLB entries	115
5.7	Per access dynamic energy with respect to conventional TLB saving for varying TLB entries with 16 preemptions and 4KB page size	115
5.8	Per access dynamic energy for varying Splash benchmark programs	117
5.9	Per access time saving with respect to conventional TLB for varying page size with 16 preemptions and 32 TLB entries	117
5.10	Per access time saving with respect to conventional TLB for varying preemptions with 4KB page size and 32 TLB entries	118
5.11	Effective access time of 32 entry, 64bits TLB for varying Splash benchmarks	119
6.1	Deterministic process aware partitioned real-time cache	124
6.2	Miss rate of CC and DEARCache by varying number of preemptions	128

6.3	Per access energy for varying preemption with 8KB cache size, 32B line size and associativity as 4 way	129
6.4	Per access energy for varying cache size with 10 preemptions, 32B line size and associativity as 4 way	130
6.5	Per access energy for varying line size with 10 preemptions, 8KB cache and associativity as 4 way	130
6.6	Per access energy for varying associativity with 10 preemptions, 8KB cache and 32B line size	131
6.7	response time for varying preemption with 8KB cache size, 32B line size and associativity as 4 way	132
6.8	response time for varying cache size with 10 preemptions, 32B line size and associativity as 4 way	133
6.9	response time for varying line size with 10 preemptions, 8KB cache and associativity as 4 way	133
6.10	response time for varying associativity with 10 preemptions, 8KB cache and 32B line size	134
6.11	Per access dynamic energy for varying line size with 10 pre- emptions, 8KB cache size and associativity as 4 way	135
6.12	Per access Static energy for varying line size with 10 preemp- tions, 8KB cache size and associativity as 4 way	135
7.1	DREAM Memory Subsystem	140
7.2	Deterministic Energy Efficient Process aware(DEEP) Cache .	141
7.3	DREAM Memory Subsystem with shared way	143
7.4	Per access dynamic energy for a 4 way, 32B, 8KB cache with varying preemptions	148

7.5	Per access dynamic energy for a 4 way, 32B line size with varying cache size [#preemptions = 10]	148
7.6	Per access dynamic energy for a 4 way, 8KB cache with varying line size [#preemptions = 10]	149
7.7	Per access dynamic energy for a 32B, 8KB cache with varying associativity [#preemptions = 10]	149
7.8	Per access static energy for a 4 way, 32B, 8KB cache with varying preemptions	150
7.9	Per access static energy for a 4 way, 32B line size with varying cache size [#preemptions = 10]	151
7.10	Per access static energy for a 4 way, 8KB cache with varying line size [#preemptions = 10]	151
7.11	Per access static energy for a 32B, 8KB cache with varying associativity [#preemptions = 10]	152
7.12	Per access time for a 4 way, 32B, 8KB cache with varying preemptions	153
7.13	Per access time for a 4 way, 32B line size with varying cache size [#preemptions = 10]	153
7.14	Per access time for a 4 way, 8KB cache with varying line size [#preemptions = 10]	154
7.15	Per access time for a 32B, 8KB cache with varying associativity [#preemptions = 10]	154
7.16	Per access dynamic energy for varying line size with 10 preemptions, 8KB cache size and associativity as 4 way	155
7.17	Per access Static energy for varying line size with 10 preemptions, 8KB cache size and associativity as 4 way	155

7.18	Per access dynamic energy for a 64KB L2 cache with varying L1 cache size	158
7.19	Per access dynamic energy for a 8KB L1 cache with varying L2 cache size	159
7.20	Per access static energy for a 64KB L2 cache with varying L1 cache size	160
7.21	Per access static energy for a 8KB L1 cache with varying L2 cache size	160
7.22	Per access time for a 64KB L2 cache with varying L1 cache size	161
7.23	Per access time for a 8KB L1 cache with varying L2 cache size	161

List of Tables

2.1	Comparison of Replacement Strategies	17
2.2	Cache Coherence States and Descriptions	25
3.1	SESC Components for Energy and Power Modeling of Cache .	41
3.2	WHP variables used	51
3.3	SESC Components for Time Modeling of Cache	53
4.1	Read, Write and Snoop operations in MOESIF Protocol	81
4.2	Energy and time modeling for cache operations	89
5.1	TLB Components for Energy and Time Modeling	106
5.2	Task Set Execution Schedule Format	109

List of Abbreviations

ASID-TLB Address Space Identifier based Translation Lookaside Buffer

CC Conventional Cache

DEARCache Deterministic Energy efficient process Aware Real-time Cache

DEEP Deterministic Energy Efficient Process aware

DREAM Deterministic REAL-time Memory system

DREAMS Deterministic REAL-time Memory system with Shared way

DTLB Deterministic Translation Lookaside Buffer Buffer

FIFO First In First Out

LFU Least Frequently Used

LFUDA Least Frequently Used Dynamic Aging

LRU Least Recently Used

MC Multi-core

MESI Modified Exclusive Shared Invalid

MESIF Modified Exclusive Shared Invalid Forward

MI Modified Invalid

MOESI Modified Owned Exclusive Shared Invalid

MOESIF Modified Owned Exclusive Shared Invalid Forward

MOSI Modified Owned Shared Invalid

MRU Most Recently Used

MSB Most Significant Bit

MSI Modified Shared Invalid

PCB Process Control Block

PLRU Pseudo Least Recently Used

RAND RANDom

RT Response Time

SA Set Associative

SESC Super EScalar Simulator

TLB Translation Lookaside Buffer

WCET Worst Case Execution Time

WH Way Halting Cache

WHP Way Halted Prediction

WP Way Predicting Cache

Chapter 1

Introduction

1.1 Background

Technology advancement has helped in increasing the processing speed for almost all the architecture components. Processors speed increases at higher rate than memory speed. Speed mismatch between these components is one of the major performance bottlenecks in modern processors which upto a limit is alleviated by using hierarchical arrangement of cache memories. Memory subsystems improve system performance by taking advantage of locality of reference - both temporal and spatial. Translation lookaside buffer (TLB) helps in converting logical address into physical address and is accessed at least once per instruction cycle. Hence, the TLB and cache plays a major role in determining system performance.

Irrespective of the processor in use, energy efficiency is one of the major design considerations of the modern day processor design. Shrinking size of transistors because of the advancement in fabrication technology increases the transistor density on chip. This improves processing power of the system at the cost of energy consumption. Narrowing of channel width in transistors result in reduced switching current and increased leakage current. This leads to reduction in dynamic energy at the cost of increased static energy.

The Dynamic energy consumption of a CMOS circuit is given as :

$$E_{dynamic} = A * V^2 * f * C \quad (1.1)$$

where A , V , f and C are cache activity factor, operating voltage, operating frequency and effective load capacitance respectively. The Static energy consumption of a CMOS circuit is as :

$$E_{static} = V * I_{leak} * N * K_{design} \quad (1.2)$$

where I_{leak} , N and K_{design} are the leakage current, number of transistors in the circuit and design dependent parameters respectively. Dynamic energy can be reduced by reducing the voltage or frequency of operation or by reducing the activity factor. Static energy can be reduced by reducing total number of transistors or by shutting down some part of the system.

Memory subsystem consumes major portion of the on-chip energy. This forces the designers to come up with memory subsystem design with least possible energy consumption without much performance degradation.

System performance can be improved by increasing operating frequency of the system. However, increase in operating frequency leads to increase in energy consumption which in turn leads to increase in heat dissipation thus leakage current. One of the possible solution to this problem is to go for Multicore(MC) systems [1]. A shared memory MC system has more than one core where L1 and L2 caches are local to the core. The L3 cache and main memory are shared across all the cores. This results in possibility of having multiple copies of data in different locations. Cores access data from

local caches as the data transfer is much faster from/to it in comparison with shared memory. It is possible that the cached data is modified in one of the cores and these modifications are not reflected in other cores, leading to data inconsistency among cores. Thus, maintaining data consistency becomes a major challenge in MC systems. Energy efficient and performance centric protocols are required to maintain data consistently in MC systems.

Deadline misses in hard real-time embedded system results in catastrophic failure. In hard real-time embedded systems, along with energy efficiency, deterministic tighter upper bound on the Worst Case Execution Time (WCET) of the task is also required. Deterministic tighter upper bound on WCET can only be ensured by making the entire process of accessing memory subsystem deterministic. This helps in improving offline and online analysis to incorporate more real-time tasks without deadline misses. One of the major components which make the system non-deterministic is memory. The unpredictability of memory subsystem is mainly because of the global replacement policy where the memory entries can be replaced by other tasks. Global replacement results in increase in memory misses on preemption. The memory access model can be made deterministic by providing a hard upper bound on the number of misses in TLB, all levels of caches and main memory.

1.2 Motivation

The cache memory access time is calculated as equation 1.3.

$$\begin{aligned} \text{Cache memory access time} &= \text{Cache memory hit time} \\ &+ \text{Miss rate} * \text{Miss penalty} \end{aligned} \quad (1.3)$$

The cache performance can be optimised by reducing cache hit time, miss rate and miss penalty. The cache misses are categorised as compulsory, capacity, conflict and coherence misses [2]. Figures 1.1 and 1.2 show hit time and hit rate performance with varying cache size and associativity, while executing FFT - Splash benchmark program [3].

As shown in figures 1.1 and 1.2, the cache hit time and hit rate of the direct map cache is the least as compared to the same-sized associative cache organisations. This is because the requested data word is available in output bus before hit/miss decision. Though the hit time of direct-map cache is the least, its average access time is highest due to highest number of conflict misses.

The energy consumption - both static and dynamic - of a cache is proportional to hit energy, block transfer energy and hit rate. Dynamic energy consumption can be reduced by reducing switching activity in cache and static energy consumption can be reduced by shutting down unused part of the cache. Architects have to find the right cache configuration to reduce energy consumption without much performance degradation. Though direct-map cache offers least per access energy consumption and hardware complexity, most of the embedded architectures prefer set associative (SA) cache because of the moderate hit energy and the hit rate.

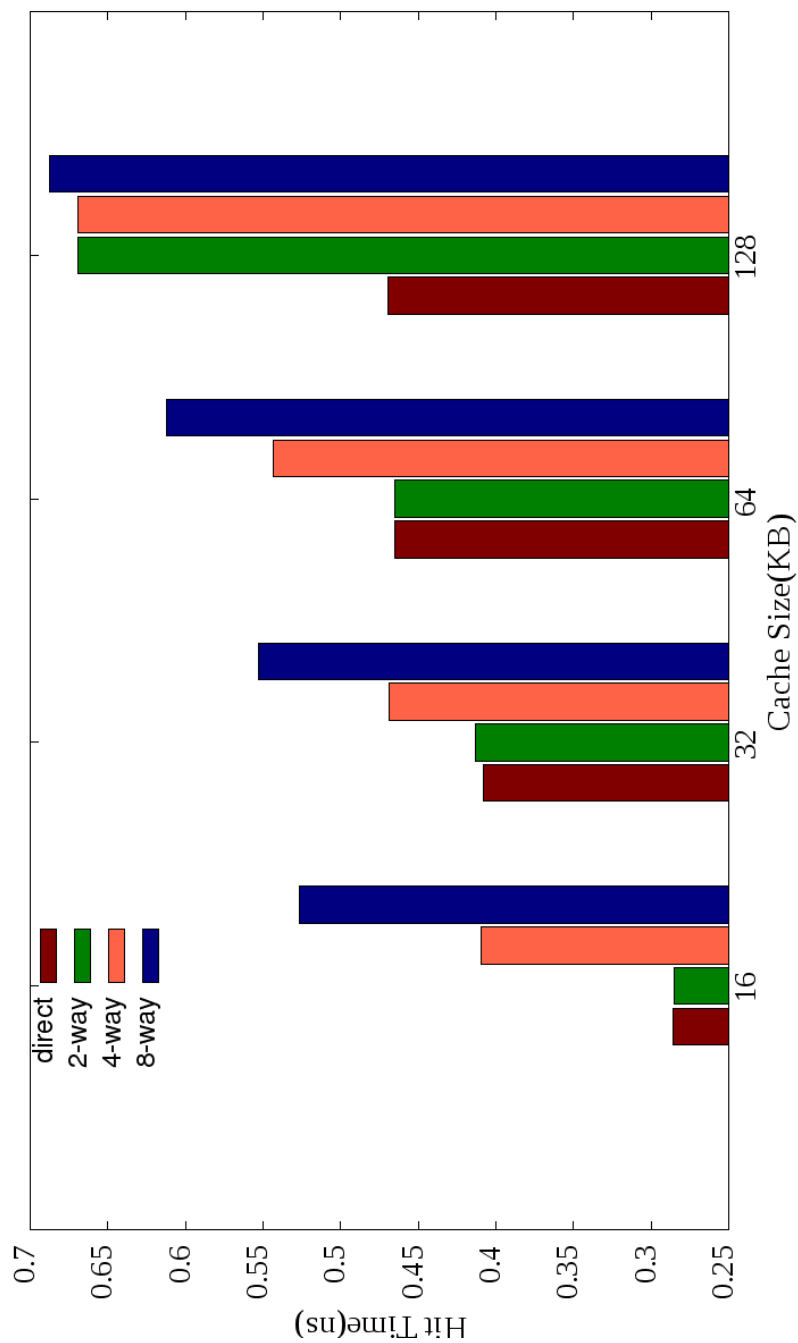


Figure 1.1: Cache hit time performance of various associativities with varying cache size

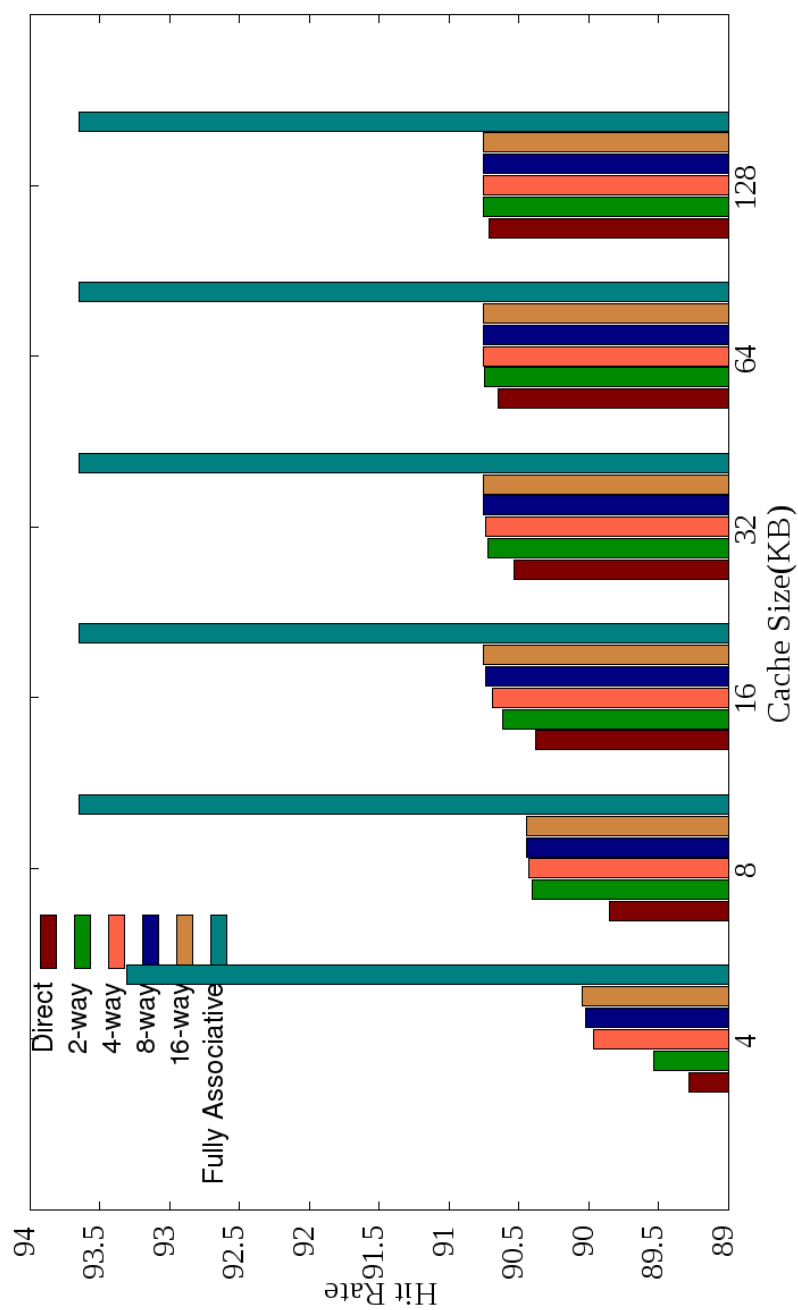


Figure 1.2: Cache hit rate performance of various associativities with varying cache size

For a SA cache, an ideal cache hit scenario is, compare one tag and access one data and an ideal cache miss scenario is to find a miss without accessing tag array and data array. Various cache optimisation strategies like way prediction [4], way halting [5] are proposed to achieve these objectives.

The growing computational demands is satisfied by adding more processors/cores to the system [6]. A coherent view of memory is crucial for these systems. A coherent data in cache is maintained by using cache coherency protocols. Cache coherency circuit, coherency misses and data/signal transfers across the network contributes towards the energy consumption in MC systems. Various coherency protocols are available to maintain data coherency but there exist a requirement of reducing coherency related energy consumption further [7].

In real-time systems, apart from the correct result of computation, the time at which the result is produced is critical [8]. This is true specially when it is used for critical applications like vehicular, aeronautical, military and industrial. Deadline miss of some of these applications lead to system failure and catastrophic consequences. Ensuring deadlines of critical applications is one of the design goals of hard real-time systems. As most of these systems are battery powered, energy efficiency is equally a design consideration along with deadline. Though cache memories are used widely to bridge the speed mismatch between processor and memory, they make the system non-deterministic. This is due to inter-task conflicts during execution. The preempted job's cache lines might get replaced by the cache lines of currently running job. When the preempted job resumes back its execution, many of its cache lines may not be present which increases its execution time. This

may result in deadline misses, especially for high priority tasks with memory operations. Design of energy efficient and performance centric hard real-time system requires a tighter upper bound on WCET. To provide tighter upper bound on WCET, memory subsystem needs to be designed with tighter upper bound on misses.

1.3 Problem Statement

This thesis addresses architecture level energy and performance optimisation of uniprocessor / uncore and MC system. With advancement in technology, the static energy consumption became an equally important component in total energy consumption along with the dynamic energy consumption. Static energy can be reduced by increasing cache hit rate and thus reducing operational time or by shutting down unused part of cache memory. Dynamic energy consumption can be reduced by reducing switching activities during cache access. Dynamic energy consumption is mainly due to charging and discharging of wordlines, bitlines, sense amplifiers, precharge circuits and decoders. To reduce dynamic energy consumption, this thesis aims at reducing switching activities of cache memory. This thesis also ensures deterministic tighter upper bound on memory access time in hard real-time systems. Detailed objective of the thesis are as follows:

Objective 1: Design of energy efficient uncore cache

The objective is to design an energy efficient cache for uncore system with reduced response time. The optimal cache hit energy can be achieved by accessing a tag array and a data array. The ideal cache miss scenario is

achieved by early detection of cache misses. Early detection of cache miss improves system performance. This objective is to achieve performance closer to ideal cache hit and cache miss scenarios. This objective is discussed in chapter 3 of this thesis.

Objective 2: Design of energy efficient cache coherency protocol for MC systems

The objective is to improve the off-chip and on-chip bandwidth usage. This work aims at designing an energy efficient cache coherency protocol by reducing coherency misses, write backs to next level memory and responders to any request. This objective is discussed in chapter 4 of this thesis.

Objective 3: Design of deterministic TLB

The objective is to design a TLB which offers tighter upper bound on number of misses and thus achieves a deterministic TLB performance. Details of this objective is discussed in chapter 5.

Objective 4: Design of deterministic L1 cache

The objective is to provide tighter upper bound on WCET of L1 cache by redesigning L1 instruction and L1 data caches. Inter-task interference makes the system non-deterministic. This issue needs to be addressed to achieve tighter upper bound on cache misses. Deterministic L1 cache design is discussed in chapter 6.

Objective 5: Design of deterministic memory subsystem

The objective is to provide tighter upper bound on WCET of memory subsystem by redesigning L2 cache and combining it with redesigned TLB and L1 cache. The deterministic memory subsystem is discussed in chapter 7.

1.4 Research Goals

This research focuses on optimising performance in terms of energy consumption and time for uncore and MC systems. Identified research goals are:

Research Goal 1: Identify time and energy impact of major components in memory subsystem.

Research Goal 2: Optimise uncore cache architecture with minimum response time and energy consumption.

Research Goal 3: Optimise MC cache architecture by minimising coherency misses and network traffic to improve performance in terms of time and energy.

Research Goal 4: Eliminate time uncertainty in TLB design to offer a tighter worst case upper bound on TLB accesses.

Research Goal 5: Eliminate time uncertainty in L1 private cache design and offer a tighter worst case upper bound on L1 accesses.

Research Goal 6: Eliminate time uncertainty in memory subsystem used for hard real-time systems by redesigning TLB, private L1, shared L2 and main memory to offer a tighter worst case upper bound.

1.5 Contributions

The major contributions of this thesis are as follows :

Contribution 1: Energy efficient uncore cache

Time and energy impact of major components in memory subsystem is identified. A part of this contribution is published in Paper A. An energy efficient

cache for uncore system with reduced per access time is designed and implemented. This contribution is discussed in chapter 3 of this thesis and is published in Paper B.

Contribution 2: Energy efficient cache coherency protocol for MC systems

Identification of various cache coherency related components contributing towards energy consumption is done. A part of this work is published in Paper C. The off-chip and on-chip bandwidth usage is improved. An energy efficient cache coherency protocol is designed and implemented. This work is published in Paper D and is discussed in chapter 4 of this thesis.

Contribution 3: Deterministic TLB

This work designed and evaluated a deterministic TLB which offers tighter upper bound on number of misses. This contribution is published in Paper E and is discussed in chapter 5 of this thesis.

Contribution 4: Deterministic L1 cache

The tighter upper bound on WCET of L1 cache is obtained by designing and evaluating deterministic L1 cache. This contribution forms chapter 6 of this thesis.

Contribution 5: Deterministic memory subsystem

The contribution provides tighter upper bound on WCET of memory subsystem. L2 cache is redesigned and combined with redesigned TLB and L1 cache. The deterministic memory subsystem is discussed in chapter 7.

1.6 Publications

Paper A: Geeta Patil, Parag Panda and Biju Raveendran, “A Survey on Replacement Strategies in Cache Memory for Embedded Systems,” IEEE Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), Mangalore, 2016, pp. 12-17.

Paper B: Geeta Patil, Neethu Bal Mallya and Biju Raveendran, “Way Halted Prediction Cache: An Energy Efficient Cache Architecture for Embedded Processors,” 28th International Conference on VLSI Design, Bangalore, 2015, pp. 65-70

Paper C: Geeta Patil, Neethu Bal Mallya and Biju Raveendran, “Simulation based Performance Study of Cache Coherence Protocols,” IEEE International Symposium on Nanoelectronic and Information Systems, Indore, 2015, pp. 125-130.

Paper D: Geeta Patil, Neethu Bal Mallya and Biju Raveendran, “MOESIF: A MC/MP Cache Coherence Protocol with Improved Bandwidth Utilization,” International Journal of Embedded Systems (In Press).

Paper E: Geeta Patil, Kajal Varma and Biju Raveendran, “DTLB: Deterministic TLB for Tightly Bound Hard Real-Time Systems,” 30th International Conference on VLSI Design and 16th International Conference on Embedded Systems, Hyderabad, 2017, pp. 207-212.

1.7 Thesis Outline

The outline of rest of the dissertation is as follows:

Chapter 2 - Literature Survey - This chapter presents a survey of state-of-the-art techniques used for energy optimisation in uncore and multicore cache architectures. The survey also includes various techniques used to obtain tighter upper bound on WCET of real-time tasks.

Chapter 3 - WHP : Way Halted Prediction Cache - This chapter presents an energy efficient set associative cache architecture named Way Halted Prediction (WHP) cache. WHP is designed to obtain reduced energy consumption and response time(RT).

Chapter 4 - MOESIF: Cache Coherency Protocol - This chapter concentrates on improving cache performance by redesigning coherency protocols. This chapter proposes an energy efficient cache coherency protocol - **Modified Owned Exclusive Shared Invalid Forward** - MOESIF. The redundant responses are concisely narrowed down in MOESIF protocol.

Chapter 5 - DTLB : Deterministic TLB for Real-time System - To have a tighter upper bound on WCET of real-time task, this chapter presents a TLB architecture - **Deterministic Translation Lookaside Buffer (DTLB)** which offers deterministic miss rate.

Chapter 6 - DEARCACHE: Deterministic Energy Efficient Process Aware Real-time Cache - To have a tighter upper bound on the WCET of real-time task, this chapter presents a **Deterministic Energy efficient process Aware Real-time cache (DEARCACHE)**. Tighter upper bound on the WCET is achieved by eliminating L1 cache related intertask interference. It allocates

at least statically identified minimum ways to each job.

Chapter 7 DREAM: Deterministic Memory Subsystem - This chapter presents an integrated design of deterministic memory named as Deterministic REAL-time Memory system (DREAM). DREAM achieves deterministic performance at TLB and L1 cache by incorporating DTLB and DEARCACHE along with deterministic energy efficient process aware L2 cache.

Chapter 8 Conclusion - This chapter concludes the thesis by summarising the results and future directions of the work.

Chapter 2

Literature Survey

2.1 Introduction

This chapter presents an exhaustive literature survey, analysis and comparison of the state-of-the-art techniques used for energy optimisation in uniprocessor/unicore and multicore memory subsystem architectures. The survey also includes various techniques used to obtain tighter upper bound on WCET of hard real-time tasks. The remainder of the chapter is organised as follows: Section 2.2 describes work done in energy efficient cache architecture. Related work in the field of cache coherency protocols and deterministic cache in real-time system are discussed in section 2.3 and 2.4 respectively.

2.2 Uniprocessor / Unicore energy optimisations

The most common approach to improve the cache hit rate with minimum energy consumption is to find the correct configuration of cache size, cache line size and associativity for the applications. Along with the configuration parameters, cache replacement strategy also plays a vital role in optimising cache performance [9]. It helps in reducing the number of cache misses and

hence, the energy consumption and effective cache access time. An optimal cache replacement strategy identifies a cache line which will not be accessed in near future as victim cache line for replacement. This is impractical as the future references are unknown [10]. The performance of the cache replacement strategy mainly depends on how accurately system can predict the future reference pattern based on the past references. The selection of a programmable replacement strategy for associative caches can have significant impact on the overall system performance. The choice of a replacement strategy is one of the most critical design issues.

Replacement strategies are classified based on time of the future reference (Optimal), time at which the cache line has arrived in cache (Arrival), time of the past reference (Recency), number of past references (Frequency), combination of recency and frequency (Recency + Frequency) or a random pick (Random).

Random (RAND) [11], First In First Out (FIFO) [9], Least Recently Used (LRU) [12], Most Recently Used (MRU) [12], Least Frequently Used (LFU) [13], [14], Least Frequently Used with Dynamic Ageing (LFUDA) [13], [15] and Pseudo-LRU (PLRU) [16] are some of the widely used hardware controlled cache replacement strategies . In order to analyse and compare hardware controlled cache replacement strategies, they are implemented on Xilinx ISE design 12.2. Memory traces of CPU2006 benchmark programs [17] were extracted by using SimpleScalar [18] simulator. The memory traces obtained are given as input to the implemented cache replacement strategies.

Table 2.1: Comparison of Replacement Strategies

Replacement Algorithm	Cache miss rate w.r.t. LRU	Speedup	Action on cache hit	Action on cache miss	Hardware Components
LRU	REFERENCE	REFERENCE	Update LRU counters	Update LRU counters	$N \log_2 N$ bits per set and associated circuitry
OPT	70%	MIN	NA	NA	NA
RAND	-22%	Average 22% slower	NONE	Update a register	$\log_2 N$ bits and Pseudo Random generator
FIFO	-20%	Average 20% slower	NONE	Update FIFO counter	$N \log_2 N$ bits per set and associated circuitry
LFU	-18%	Average 18% slower	Update LFU counter	Update LFU counter	$N \log_2 X$ bits per set and associated circuitry
LFUDA	-15%	Average 15% slower	Update LFU counters + shift LFU counters if reference counter = MAX	Update LFU counter + shift LFU counters if reference counter = MAX	$N \log_2 X$ bits per set, $\log_2 X$ bits reference counter and shifting operation after every MAX references
MRU	At least -100%	At least -100%	Update MRU counters	Update MRU counters	$N \log_2 N$ bits per set and associated circuitry
PLRU	2.5%	Best case 2.5% faster	Update MRU bits	Update MRU bits	N bits per set and associated circuitry

The recency and frequency based replacement strategies updates the state in every access where as other strategies update the states only on cache miss. This leads to increase in critical path delay for recency and frequency based strategies. The most widely used protocols at this point is LRU and its variants. There exist various implementations like counter, square matrix, skewed matrix, link list and variants like PLRU in market. This work uses LRU as replacement strategy for most of its implementation mainly because of its high predictability and high performance. The detailed comparison of replacement strategies are given in Table 2.1.

The per access energy consumption of cache can be minimized by partitioning the cache vertically or horizontally [19], [20]. Horizontal cache partitioning schemes reduce per access cache hit energy by introducing a small but powerful cache between the core and L1 cache. The cache architectures like loop cache, filter cache [21], instruction buffer [22],[23] and buffer cache [24] falls in this category. Though cache hit energy consumption is low, the hit rate of these caches are low. Vertical partitioning schemes [19] try to achieve the ideal cache access scenario, i.e., accessing only one tag and one data array for a cache hit and accessing no tag and data array for a cache miss. This is achieved by dividing the cache into banks and providing bankwise access control.

Compressed tag architecture proposed by Jong Wook Kwak and Young Tae Jeon [25] optimises energy consumption of embedded cache. Compressed tag architecture takes advantage of locality of reference. The optimisation is based on the understanding that majority of consecutive references will have same most significant bits (MSB). These distinct higher order bits are maintained

in shared registers called locality buffer register. Tag bits of desired address are divided into 2 parts - lower order tag bits, tl and higher order tag bits, th . tl bits of tag are stored in cache line along with the data part and th bits of tag are stored in separate energy efficient locations. tl bits of the desired tag are compared with lower order tag bits stored in cache line. Only on match of tl bits, th bits are compared with higher order entry stored in locality buffer register. On mismatch of tl or th bits, data is fetched from next level memory. This method requires additional hardware to maintain locality buffer, replacement circuit for locality buffer, circuit to maintain index to locality buffer.

In [26], Jones et al. proposed way placement cache to save dynamic energy consumption using static profiling to determine the most frequently used instructions. This approach uses way hint bit to decide whether to use the profiling information or not. Based on the hint bit, operating system sets additional way-placement access bits in TLB. When way-placement access bit is set, only one way is accessed otherwise all ways are accessed which results in dynamic energy saving.

Zhang et al. [27] proposed reconfigurable cache architectures - way concatenation and way shutdown caches - which can configure the cache parameters dynamically. Way concatenation cache uses two bit configurable register to select associativity between direct map, 2-way and 4-way. Way concatenation cache configures cache line size as 16, 32 and 64 bytes. Way shutdown cache saves static and dynamic energy consumption by shutting down unused ways based on application's memory usage.

Bournoutian et al. [28] used statically calculated miss rate to decide on

associativities. The reconfigurable cache architectures reduce the static and dynamic energy consumption at the cost of additional hardware complexity and reconfiguration time.

Tag less cache (TLC) proposed by Sembrant et al. [29] aimed at reducing the dynamic energy in virtually indexed physically tagged caches. TLC eliminates tag comparison in data cache by using extended TLB (eTLB), which stores the way location and valid bit of each cache line of the page. eTLB eliminates tag comparisons and detects early cache misses. TLC accesses only one data array during cache hit and does not access data array during cache miss. The cache line location in eTLB needs to be accessed to find out way of the accessed data which degrades performance in terms of energy and time due to additional data storage in TLB.

Megalingam et al. [30] proposed phased cache which accesses cache in two phases. During first phase, it compares all the tags of the given set index. If there exist a match, it accesses the data in next phase. Phased cache achieves energy saving at the cost of additional time and performance.

Inoue et al. [4] proposed way predicting (WP) cache architecture. WP enables the predicted way's tag and data array in first cycle. It compares the input tag bits with predicted way's tag bits in indexed set. If the tag matches, it accesses the data in the same cycle. If the tag is not matching in predicted way, then it enables all the other (N-1) tag and data arrays. If any of the (N-1) tags matches with input tag, the data is accessed, otherwise cache miss is executed. The energy saving of this architecture is around 60% for a 4-way set associative cache with a slight performance degradation because of prediction miss penalty. WP cache achieves ideal scenario during prediction

hit and its energy saving depends on prediction hit rate.

Access mode prediction proposed by Zhichun Zhu and Xiaodong Zhang [31] combines merits of phased cache and WP by using access mode prediction. It decides upon whether to access cache by predicting way or to access cache in phases. Performance of this is based on the accuracy of access mode prediction.

Batson et al. [32] proposed reactive associative (R-A) cache. R-A cache provides flexible associativity by placing most blocks in direct-mapped positions and reactively displacing only conflicting blocks to set-associative positions. To achieve direct-mapped hit times, R-A cache uses an asymmetric organization in which the data array is organized like a direct-mapped cache and the tag array like a set associativity cache. The data is accessed in the same cycle if the tag matches in direct-map way. R-A cache do not incur additional overhead during cache miss as it detects cache miss in first cycle itself. It needs an additional cycle if data is found in cache line other than direct mapped line.

To improve the energy efficiency by avoiding unnecessary tag comparison, Zhang [5] proposed way halting (WH) cache architecture. WH cache uses a small fully associative halt tag array per way to store the least significant 4 tag bits of each set in that way. The halt tag array is compared with 4 LSB's of address to reduce unnecessary accesses. As the halt tag comparison is happening in parallel with decoding, there is no performance degradation. All the ways where halt tag is hit, is compared with remaining tag bits to find whether there exists a match.

Though WP cache saves dynamic energy during prediction hit, it consumes

extra energy and time during prediction miss. WH cache saves dynamic energy during halt miss but does not offer optimal energy during halt hit as it may enable more than one way for comparison and access. To improve the energy efficiency and access time further, way halted prediction (WHP) cache architecture is designed and implemented as a part of this thesis.

2.3 Cache Coherency Protocols

Cache coherence is a well addressed problem in literature. The solutions to cache coherency problem is either hardware based [33] [7] [34], software based [35] [36] or hybrid [37].

Per Stenstrom surveyed various hardware and software based cache coherence protocols for shared memory MP systems [38]. Hardware based cache coherence protocols are categorized as snoopy based and directory based protocols depending on the strategy employed for maintaining data consistency [38] [39]. In directory based protocols, a shared global directory is used for maintaining state of each memory block. The snoopy based protocols maintain cache consistency with the help of local cache line states which updates itself by snooping the signals over the network. Software based cache coherence protocols maintain cache consistency by using compiler analyzed data. If data can be copied as private by multiple cores then compiler marks the data as cacheable. When data is read only or it is read / written by a single core, the data is cacheable. If any core writes data, it has to update the next level memory before some other core caches it. Data read/written by multiple cores is non-cacheable. Since performance and hardware requirements of

protocols are different, choice of cache coherence protocol become a major design decision [39].

Performance analysis of snoopy based, OS based and non-coherent protocols were explored in [40]. Results obtained imply that OS based cache coherence is very much inefficient in terms of energy consumption and performance. Energy consumption of snoopy based protocols is the least among the three categories of protocols explored in [40]. In [37], Chaves et al. discussed combination of software and hardware based protocols to maintain cache coherence. A part of this hybrid cache coherence protocol is implemented in hardware cache controller and remaining part is handled in software by using microkernel. In this scheme, the bandwidth utilization is improved by using multicasting of messages over unicasting. Latency of read request is reduced by using transition state.

In hardware controlled caches, the cache controller has to maintain states of the cache lines based on read and write operations. The widely used read mechanisms are look through and look aside. The widely used writing mechanisms are write back, write through and write buffer [41]. In case of write through mechanism, every write operation leads to update the next level memory whereas write back mechanism updates the next level memory on cache line replacement. Write buffer mechanism is an extension of write through with the help of a buffer to improve write performance further. In [42], Garo et al. combined benefits of write back and write through cache update policies. The write back reduces bus contention and dynamic energy consumption when there are frequent updates to a given cache line. Write through is beneficial when there are few writing operations. The cache

follows write through policy when cache line is updated rarely and write back policy when cache line is updated frequently. Garo et al. [42], used frequency shift register and write back bit for each cache line along with global countdown register to keep track of frequency of write back operations. The cache coherence protocols use either write update or write invalidate mechanisms to update other cores' [7]. The write update based protocols write the data and updates the data in other shared copies. When cache data is updated very frequently, write update mechanism floods the network. The write invalidate based protocols invalidate shared data in other cores before writing to the shared copy of data. Write invalidate based protocols offer less network traffic but suffers from high coherence miss latency [43]. In [43], Kayi et al. proposed an adaptive data forwarding protocol on top of write invalidate protocol for Producer-Consumer sharing pattern. This protocol uses additional Producer-Consumer Predictor Cache (PCPC) to track the sharing pattern. Write update mechanism is widely used for applications that follows Producer-Consumer sharing pattern whereas write invalidate mechanism is used for all the other applications.

In a generic cache coherence protocol, the coherence policy is defined by state transition diagram [7]. To maintain cache coherency, each cache line has a state associated with it along with other entries like data bits, tag bits and valid/invalid bits. The cache coherence state transits from one state to another according to the state transition protocol upon core read/write operation. The general description of cache states in snoopy write-invalidate protocols is given in Table 2.2.

Modified-Invalid (MI) is the simplest protocol in use for maintaining cache

Table 2.2: Cache Coherence States and Descriptions

State	Clean/ Dirty?	Access Type	Description
M	Dirty	R/W	The Cache line data has been modified by the core
O	Dirty	R	The Cache line data is modified. Other cores may have the cache line in S state
E	Clean	R/W	No other cache holds a copy of the cache line data
S	Clean	R	The Cache line data is present in more than 1 core
F	Clean	R	The Cache line which will act as a responder for any requests for the line
I	-	-	Cache line is either not present in cache or is invalid

coherency in MC systems. It uses two states: Modified (M) and Invalid (I). The hardware complexity of MI protocol is the least among all the coherence protocols. Read/write in a core results in invalidating the data in other core if the same data exists. This floods the network with large number of invalidation signals which is reduced by using S state in Modified-Shared-Invalid (MSI) protocol.

In MSI protocol, one or more cores can have valid copy of data. Data sharing results in reducing the number of coherence misses as compared to MI. In case of write on a shared copy of data, the invalidation signal is sent to all other cores irrespective of whether there exist a shared copy or not. This results in sending unwanted invalidation signals if modifying core has the exclusive valid copy of data., which is addressed by using E state in Modified-Exclusive-Shared-Invalid(MESI) protocol.

The E state in Modified-Exclusive-Shared-Invalid(MESI) protocol eliminates

unwanted invalidation signal. In MESI, for every state change from M needs data to be written back. This may result in frequent data writes between the core and next level when read and write alternate. This can be eliminated by using O state in Modified-Owned-Shared-Invalid (MOSI) protocol.

In MOSI, O state allows sharing of dirty copy between different cores. But the absence of E state results in sending invalidation signals to other cores while attempting write to an exclusive copy. This is eliminated by combining the states of MESI and MOSI protocols in Modified-Owned-Exclusive-Shared-Invalid(MOESI) protocol.

In MOESI protocol, by combining E and O state reduces number of invalidation signals and frequency of write-back operations. However, on a cache miss, all the sharers of requested data respond and the requestor is serviced by redundant responses. This is addressed by introducing F state in Modified-Exclusive-Shared-Invalid-Forward (MESIF) protocol.

MESIF protocol ensures that only the cache line in F state responds to the read/write request of other cores. Although redundant responses are eliminated by using F state, sharing of dirty data is not allowed in MESIF.

To improve energy consumption and response time further, MOESIF protocol is designed and implemented as a part of this thesis. The redundant responses in MOESI and number of write backs in MESIF are reduced by combining O and E state in MOESIF.

2.4 Deterministic Memory

Deterministic memory subsystem is a critical part of hard real-time system to improve number of executing job with no catastrophic failure. The tighter upper bound on WCET of memory subsystem requires tighter upper bound on TLB, various levels of caches and main memory.

2.4.1 Deterministic TLB

In recent past, researchers proposed various techniques to improve TLB performance. These techniques mainly focus on reducing TLB access time and/or energy consumption. This can be achieved by employing techniques which are implemented in hardware [44],[45], software [46] or combination of these [47]. The access time reduction can be achieved by reducing per TLB access time or increasing TLB hit rate. Hardware mechanism to improve TLB hit rate is to increase the number of TLB entries. This leads to increase in per access time and energy consumption.

Srilatha et al. [44] proposed a Banked Associative (BA) TLB design over a Fully Associative (FA) TLB design which increases the number of TLB entries with negligibly small impact on per access energy consumption. In BA-TLB design, only half of the entries in TLB are looked up on each TLB access. This reduces per access TLB energy consumption by half. BA-TLB offers almost the same TLB hit rate as compared to FA-TLB.

Jung-Hoon Lee et al. proposed Filter TLB [45] which uses a small TLB called filter TLB in addition to conventional TLB. Filter TLB stores most recently accessed TLB entries. For each virtual to physical address translation, the

filter TLB is accessed first. Conventional TLB is accessed only on filter TLB miss. Use of filter TLB improves TLB performance when it is filter TLB hit. But filter TLB needs additional time when it is filter TLB miss.

A recency based TLB preloading [48] is based on the principle of temporal locality of reference. Group of pages referred around the same time follow similar reference pattern. The past references can be used for predicting the future. In this model, each page table entry maintains two pointers. One points to page evicted from TLB just before this page and the other points to the page evicted after this page. On TLB miss, prefetch buffer is accessed for the required entry. If the entry is present in prefetch buffer, then it is copied to TLB. Otherwise the entry is updated from page table. The prefetch mechanism prefetches the entries and stores them in prefetch buffer. This approach requires additional hardware for storing prefetch entries and it increases size of each page table entry.

Software based approach proposed by M. Talluri and M. Hill [46] uses super pages. Size of a super page is multiple of base page size. Large page sizes can be used for storing big data like kernel data and large arrays. Use of large super pages increases TLB reach without affecting TLB access time. This model gives significant reduction in TLB misses. However, when TLB stores frame numbers with different frame sizes, number of offset bits calculation becomes dynamic. This dynamic decision makes virtual address to physical address translation difficult. Secondly, time to handle page table miss is larger for super pages with large sizes. Moreover to support the concept of super pages, substantial OS support is required.

Kandemir et al. [49] and Ilya et al. [47] follow combination of hardware and

software based techniques for TLB. Translation Registers (TRs) are used by Kandemir et al. [49] to reduce TLB access time. TRs store frequently used virtual to physical address translation. Kandemir et al. uses compiler based strategy for effectively using TRs. Compiler provides hints for address translation in TRs whenever it finds that the translation is going to be used heavily in near future. In all these approaches TLB is flushed on preemption in multitasking environment. TLB performance is affected by flushing while concurrently executing tasks. TLB misses increases due to interference caused by other tasks. Thus WCET of task in hard real-time systems becomes unpredictable. Moreover with increase in number of task in multitasking environment, the frequency of preemption increases. Ilya et al. [47] proposes Context-aware TLB Preloading (CTP) approach for multitasking environment. CTP uses static information offered by the compiler and runtime information offered by OS to identify page reference which will be made in future. CTP stores those entries from TLB which will be used in future time slot on task preemption. When task resumes, stored entries are loaded back in TLB and also TLB is preloaded with some entries which are not used yet but will be used in near future. For the smooth functioning of TRs and CTP, compiler and OS support is required. This increases complexity of compiler and OS. System performance degradation due to TLB flushing during preemptions and subsequent TLB misses is addressed by Girish et al. [50]. Girish et al. added process identifier (PID) to the TLB entry in order to associate TLB entry with the specific task. Instead of flushing TLB during preemption, TLB is accessed by comparing valid / invalid bit, virtual address and PID stored in each TLB entry. This approach increases storage overhead by 25% for

storing PID in TLB [51]. Most recently used n TLB entries are reserved on task preemption. Preempted task can have TLB entry reserved only if total number of TLB entries reserved is less than 50%. This approach reduces the number of TLB entries for executing task which increases TLB miss rate and hence adversely affects system performance. The DTLB approach proposed in this thesis offers least TLB misses as compared to TLB flushing and TLB reservation model without complicating compiler and OS.

2.4.2 Deterministic Cache

Energy efficiency, timeliness and size are the conflicting requirements of a real-time system. Energy efficiency can be attained using platform independent optimizations like reducing the number of preemptions and cache impacts [52],[53] and platform dependent optimizations like DVFS [54],[55] and DPM [56]. Timing performance is very critical for hard and firm real-time systems. Performance of a system can be analysed if tighter upper bound on WCET is available. The tighter upper bound on WCET of a real-time job is possible only if accurate values are obtainable from various levels of design hierarchies like architectural level, operating system level and application level.

Davis et al.[57] compared various scheduling algorithms using parameters like optimality, feasibility, comparability, predictability, sustainability and anomalies. Zhang et al. [58] reported Quick convergence Processor-demand Analysis(QPA) algorithm which reduces the schedulability analysis time for schedulable/unschedulable task sets. The scheduling algorithms with arbitrary preemptions induce additional cache flushes and reloads. Bril et al. [52] performed WCET analysis based on critical instant and busy period for

fixed priority preemptive scheduling.

Task preemption followed by fetching of cache lines after resumption leads to additional delay in WCET. Ju et al. [59] presented WCET analysis with this additional delay. Further, conflicting jobs may replace a dirty cache line. This results in swapping of blocks between cache and memory. It introduces two block transfers, thus twice the amount of delay and energy consumption. Dirty cache lines of low priority jobs may impact the response time of higher priority job and vice versa. Davis et al.[60] considered the impact of write back caches in WCET and analyzed the schedulability of fixed priority task with preemptive & non-preemptive schedules by incorporating the same. Altmeyer et al.[53] reduced preemption cost in real-time system by selecting a preemption point with lower preemption impact.

The program level memory access patterns have huge impact in architecture level energy consumption. Minor change in the program code or program input may lead to dramatic changes in memory behaviour. The WCET calculation is done with unrealistic assumption that all memory references lead to cache misses. This results in the execution time being overestimated by several hundred percent [61]. To offer a tighter bound on architecture, the memory system should be deterministic. Cache partitioning is one of the most widely adopted strategies to make real-time cache deterministic. Whitham et al. [62] described a method to reduce the cache-related preemption delay in hard real-time systems using explicit reservation of cache memory. Chang et al.[63] presented Cooperative Cache Partitioning scheme which makes use of multiple time-sharing partitions that allows greater speedup and fairness.

Falk et al. [64] proposed a static compiler based cache locking mechanism.

It generates execution flow graph using context-specific flow graph. It finds the longest execution path and locks cache entries of the same. Puaut et al. [65] proposed an algorithm which partitions the task into a set of regions. Dedicated cache line is allotted to each region. These methods partitions the cache memory and determines WCET by considering intra-task execution. This thesis proposes deterministic cache memory which offers tighter upper bound on the execution time of each task by extending process aware L1 cache and deterministic TLB along with partitioned L2 cache.

Chapter 3

WHP:Way Halted Prediction Cache

3.1 Introduction

This chapter presents an energy efficient set associative cache architecture named Way Halted Prediction cache (WHP). WHP aims at reducing energy consumption and Response Time (RT). Way Predicting cache (WP) proposed by Inoue et al.[4] offers ideal cache hit scenario in case of prediction hit, but needs additional cycle in case of prediction miss. Way Halting cache (WH) proposed by Zhang et al. [5] offers early detection of cache miss with the help of additional halt tag array. WH offers ideal cache miss scenario by not accessing any of the tag and data arrays when halt tag comparison in all ways is miss. It also offers ideal cache hit scenario when halt tag hit happens exactly in one way. In all other cases, WH offers higher energy consumption than WP with prediction hit. Better performance both in terms of energy and time can be achieved with the help of combining the merits of WP and WH. This is achieved by combining halt tag array of WH and way prediction circuit of WP. Halt tag array helps in early detection of prediction misses, which saves time and energy. Prediction hit reduces the number of enabled ways

from $k - ways$ to $1 - way$, which reduces the dynamic energy consumption further. In this chapter, WHP is compared with conventional cache (CC), WH and WP on the basis of energy consumption and RT.

3.2 WHP cache architecture

WHP combines the advantages of WH and WP architectures. WP uses MRU way for prediction. WP enables the predicted way's tag array and data array in first cycle. It compares the input tag bits with predicted way's tag bits in indexed set. If the tag matches, it accesses the data in same cycle. If the tag is not matching in predicted way, then it enables all the other $(k - 1)$ tag and data arrays where k is the number of ways with Halt tag hit. If any of the $(k - 1)$ tags matches with the input tag, the data corresponding to that tag is accessed, otherwise cache miss is executed. WHP compares no tag array and accesses no data array in case of halt miss which is an ideal cache miss scenario. It compares one tag array and accesses one data array in two scenarios (a) halt hit in one way (b) halt tag hit in more than one way and prediction hit, which are the ideal cache hit scenarios.

WP saves dynamic energy during prediction hit, but it results in increasing the RT during prediction miss. Prediction accuracy of WP is improved by using halt tags in WHP. WHP uses halt tags for the early detection of probable prediction misses in decode cycle of cache access. The early detection of prediction miss is achieved when predicted way is a halt tag miss. This results in WHP offering better performance as compared to WP.

In comparison with CC, WH saves dynamic energy when there exist (a) halt

tag miss (b) cache hit with halt tag hit in less than k ways. When the halt tag hits are more than one, WHP uses prediction circuit to predict the most recently accessed way. When predicted way is a hit, the energy consumption in WHP will be much lesser than WH. The WHP cache architecture is shown in Figure 3.1. The $\text{tag}(t)$, $\text{index}(i)$ and $\text{offset}(o)$, for cache is calculated from physical address. The cache access in WHP takes place as follows:

Step1: The derived index is decoded and in parallel the low-order 4 bits of the derived tag are compared against all the halt tags stored in fully-associative halt-tag array. The parallel comparison of halt tags will determine the ways with mis-matching tags to be halted. The output line of the decoder is ANDed with the results of the halt tag comparison for that row. Hence, the cache access would be continued only if the low-order 4 bits of the tag in the decoded row are matching with the low-order 4 bits of the derived tag. If all the halt tags are mis-match, its a cache miss. A processor is stalled till it gets the data.

The number of ways halted will determine the activation of prediction circuit. The prediction circuit is activated when halt-tag hit takes place in more than one way. The prediction circuit predicts a way based on history.

Step2: If halt tag is hit in only 1 way, the respective tag and data are accessed. If halt tag hits in k -ways, prediction circuit is activated. The prediction circuit used is shown in Figure 3.2. MRU way is the one selected as the predicted way. It is checked if the predicted way is in the halt tag hit ways. If yes, the tag and data array of the predicted way are accessed. Otherwise, the tag and data arrays of the k halt tag hit ways

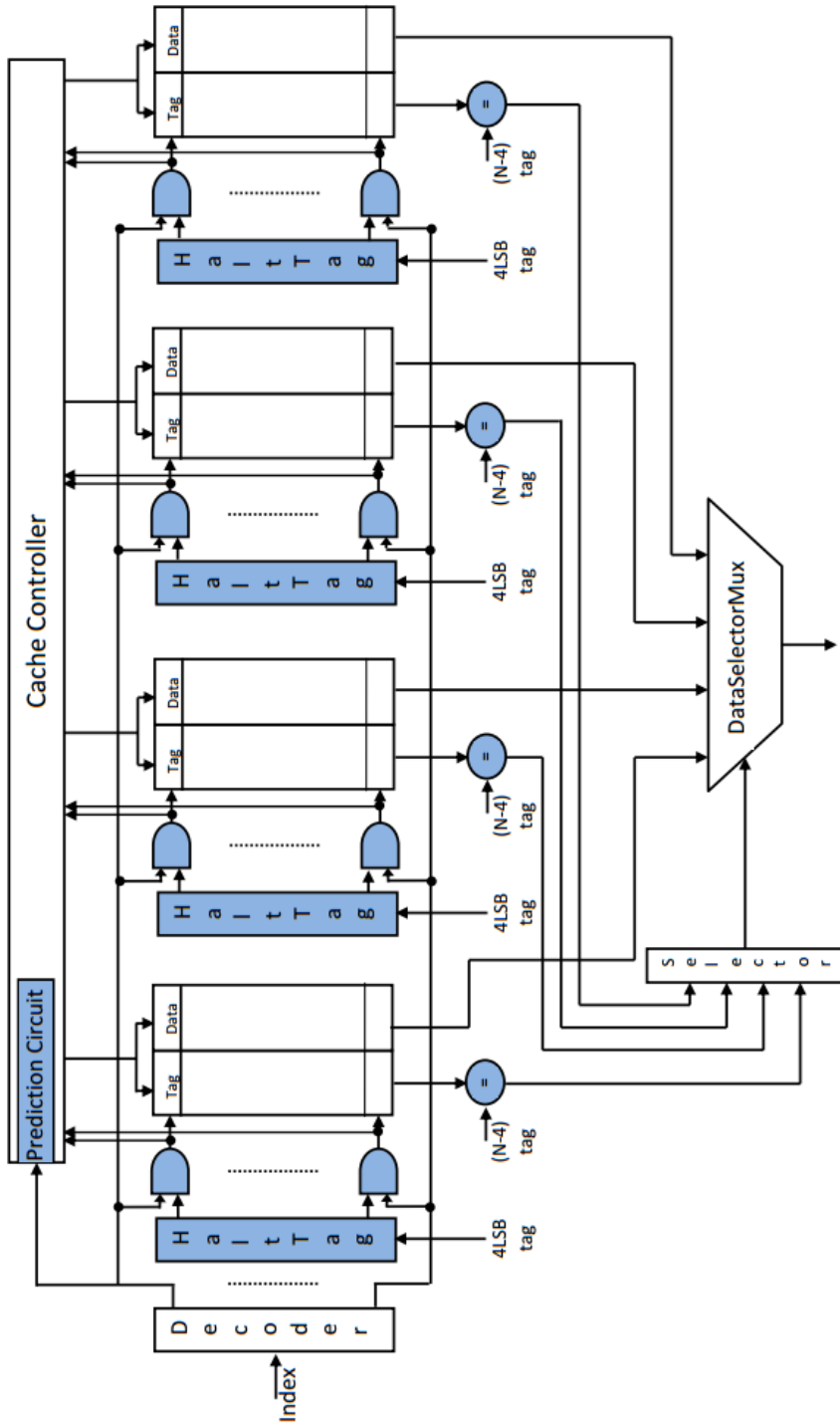


Figure 3.1: Way Halted Prediction Cache Architecture

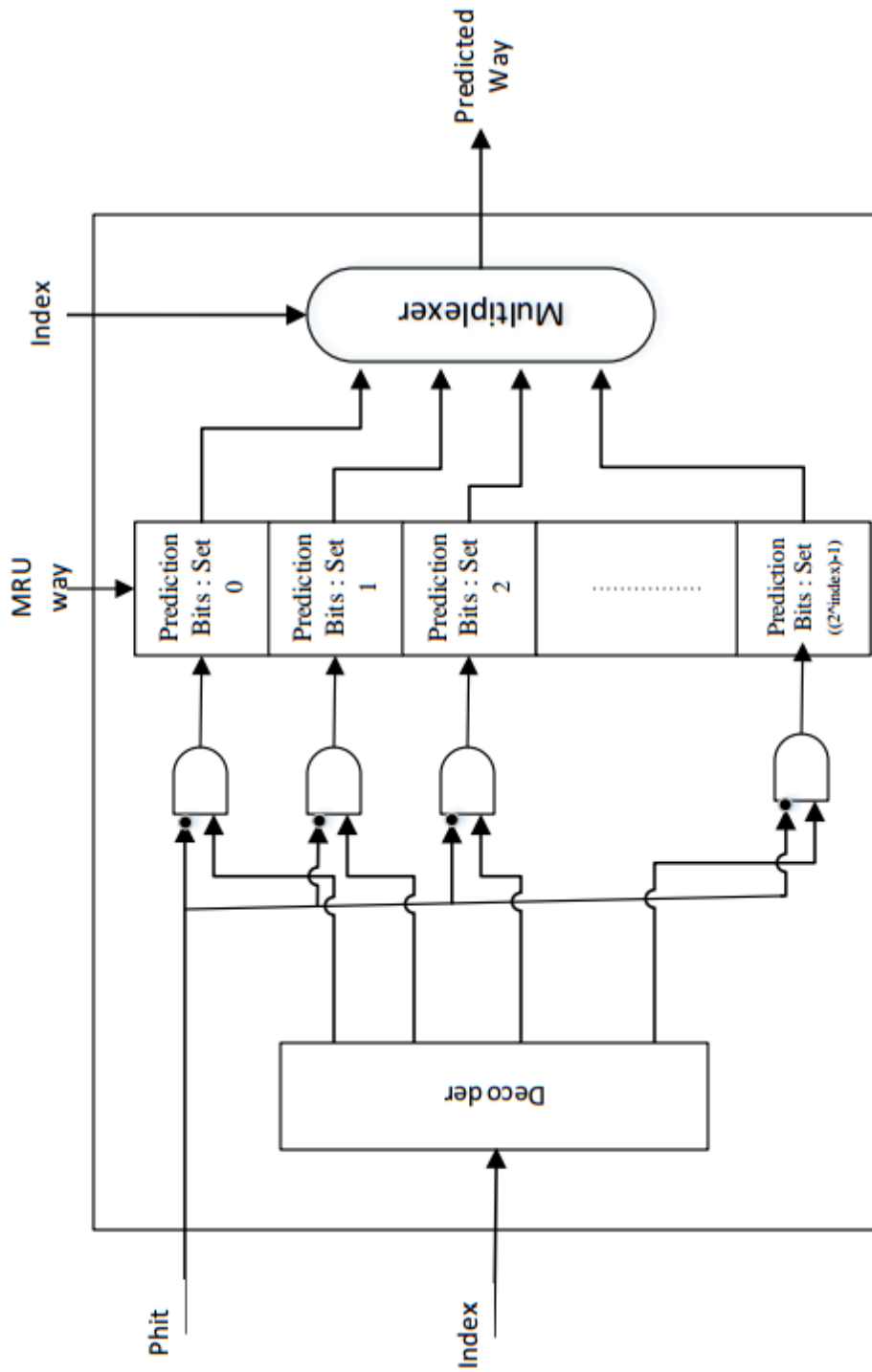


Figure 3.2: Prediction Circuit

are accessed.

The output driver of the tag array comparison indicates whether the tag is matching with the desired address tag. If the tag comparison is unsuccessful and there is only 1 way with the halt tag hit, its a cache miss. A processor is stalled till it gets the data. If the prediction is a miss, the remaining $k - 1$ tag ways are compared for a possible tag match which requires an additional cycle. If there exist no match in k ways, its a cache miss. The processor stalls till it gets the data.

Step3: This step accesses the data array of the hit way and gets the requested word using offset bits of the desired address.

The detailed algorithm which illustrates the working of WHP is given in Algorithm 1.

Algorithm WHP : The WHP algorithm takes physical address (P) as input and obtains tag, index and offset bits of P . The decoding takes place in parallel with halt tag comparison. Number of halt-tag array hits , C , in desired set is obtained.

Halt-Miss with C as 0 is the ideal scenario of cache miss with zero tag comparisons and zero data access. The early detection of miss in the first cycle of cache access reduces the dynamic energy spent in tag and data array comparison. The processor is stalled till it gets the data from the next level of memory resulting in additional cycles (miss penalty) to obtain the requested data.

WHP directly enables tag and data array of Halt-tag hit way if C is 1. It compares the desired tag with enabled tag bits. If tag matches, which is ideal cache hit scenario with one tag comparison and one data access.

Algorithm 1: Way Halted Prediction

Input: Physical address P
Output: Cache *hit/miss* and the requested data

```

1 begin
2   Decode  $P$ 's index bits to locate the desired set;
3   Compare Halt-tag array with low order 4 bits of  $P$ 's tag;
4   Count the number of Halt-tag hit ways,  $C$ , in the desired set;
5   if  $C == 0$  then
6     goto 37;
7   else
8     if  $C == 1$  then
9       Enable the tag and data array of Halt-tag hit way;
10      if tag matches then
11        Transfer requested data from / to Processor;
12        return;
13      else
14        goto 37;
15    else
16      Find the predicted way,  $W_p$ ;
17      if Halt-tag array hit ways contain  $W_p$  then
18        Enable the tag and data array of  $W_p$ ;
19        Compare remaining tag bits of  $P$  with enabled tag array
20        bits;
21        if tag matches then
22          Transfer requested data from / to Processor;
23          return;
24        else
25           $PHIT = 1$ ;
26        end
27      else
28         $PHIT = 0$ ;
29      end
30    end
31  Enable the tag and data array of remaining  $C - PHIT$  halt-tag hit
32  ways;
33  Compare remaining tag bits of  $P$  with enabled tag array bits;
34  if no tag matches then goto 37 ;
35  Transfer requested data from / to Processor;
36  return;
37  Activate Cache Miss Routine;
38  Transfer data from Lower level memory to Cache and transfer data
39  from / to Processor;
40  return;
41 end

```

The prediction circuit is enabled if Halt-tag is hit in more than 1 way, $C > 1$. If predicted way is in Halt-tag hit way, the remaining tag bits of predicted way is compared with remaining $t - 4$, tag bits of the desired address. If it is prediction hit, WHP achieves the ideal cache hit scenario. Remaining $C - 1$ tag and data arrays are enabled in case of prediction miss. $t - 4$ tag bits of P is compared with tag bits of enabled ways. The mis-prediction results in an additional cycle to determine cache hit/miss.

If predicted way is not in Halt-tag hit way then C tag and data arrays are enabled. $t - 4$ tag bits are compared with tag bits of enabled ways.

3.3 Energy Model

For energy and time modeling Super EScalar Simulator (SESC) is used. [66] simulator. SESC is a microprocessor architectural simulator which models a out-of-order pipeline with branch prediction, caches, buses, and processing component of a modern processor. SESC models different cache sizes, hit & miss latencies, replacement policies, cache-line sizes and associativities. The components used for modeling energy and power are given in Table 3.1.

3.3.1 Conventional Cache

Cache Read Hit Energy:

$$E_{dyn_CC_CRHit} = E_{dyn_dec} + N * E_{dyn_x} + E_{dyn_op_drvr} \quad (3.1)$$

Table 3.1: SESC Components for Energy and Power Modeling of Cache

Cache Components	Dynamic Energy Components		Total Dynamic Energy	Leakage Power Components		Total Leakage Power
	Data Side	Tag Side		Data Side	Tag Side	
Decoder	$E_{dyn_ds_dec}$	$E_{dyn_ts_dec}$	$E_{dyn_dec} = E_{dyn_ds_dec} + E_{dyn_ts_dec}$	$P_{leak_ds_dec}$	$P_{leak_ts_dec}$	$P_{leakage} = P_{leak_ds_dec} + P_{leak_ts_dec}$
Word Lines	$E_{dyn_ds_wline}$	$E_{dyn_ts_wline}$	$E_{dyn_w} = (E_{dyn_ds_wline} + E_{dyn_ds_bline} + E_{dyn_ds_sa} + E_{dyn_ds_opDrvr}) + E_{dyn_ts_wline}$	$P_{leak_ds_wline}$	$P_{leak_ts_wline}$	$P_{leak_ds_wline} + P_{leak_ds_sa} + P_{leak_ds_opDrvr} + P_{leak_ts_wline}$
Bit Lines	$E_{dyn_ds_bline}$	$E_{dyn_ts_bline}$	$E_{dyn_ds_sa} + E_{dyn_ts_sa}$	$P_{leak_ds_bline}$	$P_{leak_ts_bline}$	$P_{leak_ts_bline} + P_{leak_ts_sa}$
Sense Amplifier	$E_{dyn_ds_sa}$	$E_{dyn_ts_sa}$	$E_{dyn_ts_bline} + E_{dyn_ts_sa} + E_{dyn_ts_cmp} / N$	$P_{leak_ds_sa}$	$P_{leak_ts_sa}$	$P_{leak_ts_bline} + P_{leak_ts_sa} + P_{leak_ts_cmp}$
Data Output	$E_{dyn_ds_opDrvr}$	—	Where N is the associativity	$P_{leak_ds_opDrvr}$	—	$P_{leak_ds_mux} + P_{leak_ds_sel}$
Comparator	—	$E_{dyn_ts_cmp}$		—	$P_{leak_ts_cmp}$	
Multiplexer	$E_{dyn_ds_mux}$	—	$E_{dyn_opDrvr} + E_{dyn_ds_mux} + E_{dyn_ds_sel} + E_{dyn_ds_opDrvr}$	$P_{leak_ds_mux}$	—	
Select Signal	$E_{dyn_ds_sel}$	—		$P_{leak_ds_sel}$	—	

where $E_{dyn.x}$ is the dynamic access energy per way and N is the associativity. L1 Cache uses write through as the writing mechanism and write energy is twice the read energy.

Cache Write Hit Energy:

$$E_{dyn.CC.CWHit} = 2 * E_{dyn.CC.CRHit} \quad (3.2)$$

Every cache miss results in cache hit.

Total cache read dynamic energy is:

$$E_{dyn.CCR} = E_{dyn.CC.CRHit} + C_{RMiss} * E_{dyn.Tx} \quad (3.3)$$

Total Cache write dynamic energy is:

$$E_{dyn.CCW} = E_{dyn.CC.CWHit} + C_{WMiss} * E_{dyn.Tx} \quad (3.4)$$

where C_{RMiss} , C_{WMiss} and $E_{dyn.Tx}$ is the cache read miss rate, write miss rate and $E_{dyn.Tx}$ is the data transfer energy from next level cache. Total Cache dynamic energy is:

$$E_{dyn.CC} = E_{dyn.CCR} + E_{dyn.CCW} \quad (3.5)$$

Total Cache static energy is:

$$E_{static.CC} = P_{leakage} * RT_{CC} \quad (3.6)$$

Where RT_{CC} is the response time and is given by :

$$RT_{CC} = \text{total cycles required for completion of program} * \text{cycle time} \quad (3.7)$$

3.3.2 Way Predicting Cache

In way predicting cache, different components consuming energy are decoder, prediction circuit, way access energy and output driver energy.

Prediction Hit, Cache Read Hit Energy:

$$E_{dyn_PHit_CRHit} = (E_{dyn_dec} + E_{dyn_pred}) + E_{dyn_x} + E_{dyn_op_drvr} \quad (3.8)$$

Prediction Hit, Cache Write Hit Energy:

$$E_{dyn_PHit_CWHit} = 2 * E_{dyn_PHit_CRHit} \quad (3.9)$$

Prediction Hit, Cache Hit Energy:

$$E_{dyn_PHit_CHit} = E_{dyn_PHit_CRHit} + E_{dyn_PHit_CWHit} \quad (3.10)$$

Prediction Miss, Cache Read Hit Energy:

$$E_{dyn_PMiss_CRHit} = (E_{dyn_dec} + E_{dyn_pred}) + N * E_{dyn_x} + E_{dyn_op_drvr} \quad (3.11)$$

Here additional energy is required to access remaining $(N - 1)$ ways.

Prediction Miss, Cache Write Hit Energy:

$$E_{dyn_PMiss_CWHit} = 2 * E_{dyn_PMiss_CRHit} \quad (3.12)$$

Prediction Miss, Cache Hit Energy:

$$E_{dyn_PMiss_CHit} = E_{dyn_PMiss_CRHit} + E_{dyn_PMiss_CWHit} \quad (3.13)$$

Prediction Miss, Cache Read Miss Energy:

$$E_{dyn_PMiss_CRMiss} = E_{dyn_PMiss_CRHit} + E_{dyn_Tx} \quad (3.14)$$

Prediction Miss, Cache Write Miss Energy:

$$E_{dyn_PMiss_CWMiss} = 2 * E_{dyn_PMiss_CRHit} + E_{dyn_Tx} \quad (3.15)$$

Prediction Miss, Cache Miss Energy:

$$E_{dyn_PMiss_CRMiss} = E_{dyn_PMiss_CRMiss} + E_{dyn_PMiss_CWMiss} \quad (3.16)$$

Total Read Dynamic Energy:

$$\begin{aligned} E_{dyn_WPR} = & C_{PRHit} * E_{dyn_PHit_CRHit} \\ & + ((1 - C_{RMiss}) - C_{PRHit}) * E_{dyn_PMiss_CRHit} \\ & + C_{RMiss} * E_{dyn_PMiss_CRMiss} \end{aligned} \quad (3.17)$$

Total Write Dynamic Energy:

$$\begin{aligned}
E_{dyn_WPW} = & C_{PWHit} * E_{dyn_PHit_CWHit} \\
& + ((1 - C_{WMiss}) - C_{PWHit}) * E_{dyn_PMiss_CWHit} \quad (3.18) \\
& + C_{WMiss} * E_{dyn_PMiss_CWMiss}
\end{aligned}$$

where C_{PRHit} , C_{RMiss} , C_{PWHit} , C_{WMiss} is the prediction read hit rate, the cache read miss rate, prediction write hit rate, the cache write miss rate respectively.

Total WP Cache dynamic energy is:

$$E_{dyn_WP} = E_{dyn_WPR} + E_{dyn_WPW} \quad (3.19)$$

Total WP Cache static energy is:

$$E_{static_WP} = (P_{leakage} + P_{pred_ckt}) * RT_{WP} \quad (3.20)$$

Where RT_{WP} is the response time with WP and is given by :

$$RT_{WP} = \text{total cycles required for completion of program with WP} * \text{cycle time} \quad (3.21)$$

3.3.3 Way Halting Cache

Cache Read Hit Energy:

$$E_{dyn_HHit_CRHit} = (E_{dyn_dec} + E_{dyn_halt}) + k * E_{dyn_x} + E_{dyn_op_drvr} \quad (3.22)$$

where E_{dyn_halt} is the energy required to access halt tag array and k is number of halt tag hit ways.

Cache Write Hit Energy:

$$E_{dyn_HHit_CWHit} = 2 * E_{dyn_HHit_CRHit} \quad (3.23)$$

Halt Hit Cache Read Miss Energy:

$$E_{dyn_HHit_CRMiss} = E_{dyn_HHit_CRHit} + E_{dyn_Tx} \quad (3.24)$$

Halt Hit Cache Write Miss Energy:

$$E_{dyn_HHit_CWMiss} = E_{dyn_HHit_CWHit} + E_{dyn_Tx} \quad (3.25)$$

Halt Miss, Cache Read Miss Energy:

$$E_{dyn_HMiss_CRMiss} = (E_{dyn_dec} + E_{dyn_halt}) + E_{dyn_Tx} + E_{dyn_op_drv} \quad (3.26)$$

Halt Miss, Cache Write Miss Energy:

$$E_{dyn_HMiss_CWMiss} = 2 * (E_{dyn_dec} + E_{dyn_halt} + E_{dyn_op_drv}) + E_{dyn_Tx} \quad (3.27)$$

Total Read Dynamic Energy:

$$\begin{aligned} E_{dyn_WHR} = & C_{RHit} * E_{dyn_HHit_CRHit} + (C_{HRRit} - C_{RHit}) * E_{dyn_HHit_CRMiss} \\ & + (1 - C_{HRRit}) * E_{dyn_HMiss_CRMiss} \end{aligned} \quad (3.28)$$

where C_{RHit} and C_{HRHit} is cache read hit rate and halt tag read hit rate respectively.

Total Write Dynamic Energy:

$$E_{dyn_WHW} = C_{WHit} * E_{dyn_HHit_CWHit} + (C_{HWHit} - C_{WHit}) * E_{dyn_HHit_CWMiss} + (1 - C_{HWHit}) * E_{dyn_HMiss_CWMiss} \quad (3.29)$$

where C_{WHit} and C_{HWHit} is cache write hit rate and halt tag write hit rate respectively.

Total WH Cache dynamic energy is:

$$E_{dyn_WH} = E_{dyn_WHR} + E_{dyn_WHW} \quad (3.30)$$

Total WH Cache static energy is:

$$E_{static_WH} = (P_{leakage} + P_{halt}) * RT_{WH} \quad (3.31)$$

Where RT_{WH} is the response time with WH and is given by :

$$RT_{WH} = \text{total cycles required for completion of program with WH} * \text{cycle time} \quad (3.32)$$

3.3.4 Way Halted Prediction Cache

Halt Hit in 1 way, Cache Read Hit Energy:

$$E_{dyn_HHit1_CRHit} = (E_{dyn_dec} + E_{dyn_halt}) + E_{dyn_x} + E_{dyn_op_drvr} \quad (3.33)$$

Halt Hit in 1 way, Cache Write Hit Energy:

$$E_{dyn_HHit1_CWHit} = 2 * E_{dyn_HHit1_CRHit} \quad (3.34)$$

Halt Hit in 1 way, Cache Read Miss Energy:

$$E_{dyn_HHit1_CRMiss} = E_{dyn_HHit1_CRHit} + E_{dyn_Tx} \quad (3.35)$$

Halt Hit in 1 way, Cache Write Miss Energy:

$$E_{dyn_HHit1_CWMiss} = E_{dyn_HHit1_CWHit} + E_{dyn_Tx} \quad (3.36)$$

Halt Hit in k ways, W_p is a halt hit way, prediction hit, Cache Read Hit Energy:

$$E_{dyn_HHitk_Wp_PHit_CRHit} = E_{dyn_HHit1_CRHit} + E_{dyn_pred} \quad (3.37)$$

Halt Hit in k ways, W_p is a halt hit way, prediction hit, Cache Write Hit Energy:

$$E_{dyn_HHitk_Wp_PHit_CWHit} = 2 * E_{dyn_HHitk_Wp_PHit_CRHit} \quad (3.38)$$

Halt Hit in k ways, W_p is a halt hit way, prediction miss, Cache Read Hit Energy:

$$E_{dyn_HHitk_Wp_PMiss_CRHit} = E_{dyn_HHitk_Wp_PHit_CRHit} + (k-1) * E_{dyn_x} \quad (3.39)$$

Halt Hit in k ways, W_p is a halt hit way, prediction miss, Cache Write Hit Energy:

$$E_{dyn_HHitk_Wp_PMiss_CWHit} = 2 * E_{dyn_HHitk_Wp_PMiss_CRHit} \quad (3.40)$$

Halt Hit in k ways, W_p is a halt hit way, prediction miss, Cache Read Miss Energy:

$$E_{dyn_HHitk_Wp_PMiss_CRMiss} = E_{dyn_HHitk_Wp_PMiss_CRHit} + E_{dyn_Tx} \quad (3.41)$$

Halt Hit in k ways, W_p is a halt hit way, prediction miss, Cache Write Miss Energy:

$$E_{dyn_HHitk_Wp_PMiss_CWMiss} = 2 * E_{dyn_HHitk_Wp_PMiss_CRHit} + E_{dyn_Tx} \quad (3.42)$$

Halt Hit in k ways, W_p is a in halt miss, Cache Read Hit Energy:

$$E_{dyn_HHitk_CRHit} = (E_{dyn_dec} + E_{dyn_halt} + E_{dyn_pred}) + k * E_{dyn_x} + E_{dyn_op_drv} \quad (3.43)$$

Halt Hit in k ways, W_p is a in halt miss, Cache Write Hit Energy:

$$E_{dyn_HHitk_CWHit} = 2 * E_{dyn_HHitk_CRHit} \quad (3.44)$$

Halt Hit in k ways, W_p is a halt miss, Cache Read Miss Energy:

$$E_{dyn_HHitk_CRMiss} = E_{dyn_HHitk_CRHit} + E_{dyn_Tx} \quad (3.45)$$

Halt Hit in k ways, W_p is a halt miss, Cache Write Miss Energy:

$$E_{dyn_HHitk_CWMiss} = 2 * E_{dyn_HHitk_CRHit} + E_{dyn_Tx} \quad (3.46)$$

Halt Miss, Cache Read Miss Energy:

$$E_{dyn_HMiss_CRMiss} = (E_{dyn_dec} + E_{dyn_halt}) + E_{dyn_Tx} \quad (3.47)$$

Halt Miss, Cache Write Miss Energy:

$$E_{dyn_HMiss_CWMiss} = E_{dyn_HHit1_CWHit} + E_{dyn_Tx} \quad (3.48)$$

Description of various variables used to calculated total dynamic energy is given in table 3.2.

Total Dynamic Read Energy:

$$\begin{aligned} E_{dyn_WHPR} = & C_{HHit1_CRHit} * E_{dyn_HHit1_CRHit} + C_{HHit1_CRMiss} * E_{dyn_HHit1_CRMiss} + \\ & C_{HHitk_Wp_PHit_CRHit} * E_{dyn_HHitk_Wp_PHit_CRHit} + \\ & C_{HHitk_Wp_PMiss_CRHit} * E_{dyn_HHitk_Wp_PMiss_CRHit} + \\ & C_{HHitk_Wp_PMiss_CRMiss} * E_{dyn_HHitk_Wp_PMiss_CRMiss} + \\ & C_{HHitk_CRHit} * E_{dyn_HHitk_CRHit} + C_{HHitk_CRMiss} * E_{dyn_HHitk_CRMiss} + \\ & C_{HMiss_CRMiss} * E_{dyn_HMiss_CRMiss} \end{aligned} \quad (3.49)$$

Table 3.2: WHP variables used

Variable	Description
C_{HHit1_CRHit}	Rate of Halt tag hit in only 1 way and Cache read hit
C_{HHit1_CRMiss}	Rate of Halt tag hit in only 1 way and Cache read miss
$C_{HHitk_Wp_PHit_CRHit}$	Rate of Halt tag hit in k-ways, Predicted Way in Halt tag hit ways, Prediction Hit, Cache read hit
$C_{HHitk_Wp_PMiss_CRHit}$	Rate of Halt tag hit in k-ways, Predicted Way in Halt tag hit ways, Prediction Miss, Cache read hit
$C_{HHitk_Wp_PMiss_CRMiss}$	Rate of Halt tag hit in k-ways, Predicted Way in Halt tag hit ways, Prediction Miss, Cache read miss
C_{HHitk_CRHit}	Rate of Halt tag hit in k-ways, Predicted Way not in Halt tag hit ways, Cache read hit
C_{HHitk_CRMiss}	Rate of Halt tag hit in k-ways, Predicted Way not in Halt tag hit ways, Cache Miss
C_{HMiss_CRMiss}	Rate of Halt miss and Cache read miss
C_{HHit1_CWHit}	Rate of Halt tag hit in only 1 way and Cache write hit
C_{HHit1_CWMiss}	Rate of Halt tag hit in only 1 way and Cache write miss
$C_{HHitk_Wp_PHit_CWHit}$	Rate of Halt tag hit in k-ways, Predicted Way in Halt tag hit ways, Prediction Hit, Cache write hit
$C_{HHitk_Wp_PMiss_CWHit}$	Rate of Halt tag hit in k-ways, Predicted Way in Halt tag hit ways, Prediction Miss, Cache write hit
$C_{HHitk_Wp_PMiss_CWMiss}$	Rate of Halt tag hit in k-ways, Predicted Way in Halt tag hit ways, Prediction Miss, Cache write miss
C_{HHitk_CWHit}	Rate of Halt tag hit in k-ways, Predicted Way not in Halt tag hit ways, Cache write hit
C_{HHitk_CWMiss}	Rate of Halt tag hit in k-ways, Predicted Way not in Halt tag hit ways, Cache Miss
C_{HMiss_CWMiss}	Rate of Halt miss and Cache write miss

Total Dynamic Write Energy:

$$\begin{aligned}
E_{dyn_WHPW} = & C_{HHit1_CWHit} * E_{dyn_HHit1_CWHit} + C_{HHit1_CWMiss} * E_{dyn_HHit1_CWMiss} + \\
& C_{HHitk_Wp_PHit_CWHit} * E_{dyn_HHitk_Wp_PHit_CWHit} + \\
& C_{HHitk_Wp_PMiss_CWHit} * E_{dyn_HHitk_Wp_PMiss_CWHit} + \\
& C_{HHitk_Wp_PMiss_CWMiss} * E_{dyn_HHitk_Wp_PMiss_CWMiss} + \\
& C_{HHitk_CWHit} * E_{dyn_HHitk_CWHit} + C_{HHitk_CWMiss} * E_{dyn_HHitk_CWMiss} + \\
& C_{HMiss_CWMiss} * E_{dyn_HMiss_CWMiss}
\end{aligned} \tag{3.50}$$

Total WHP Cache dynamic energy is:

$$E_{dyn_WHP} = E_{dyn_WHPR} + E_{dyn_WHPW} \tag{3.51}$$

Total WHP Cache static energy is:

$$E_{static_WHP} = (P_{leakage} + P_{pred_ckt} + P_{halt}) * RT_{WHP} \tag{3.52}$$

Where RT_{WHP} is the response time with WHP and is given by :

$$RT_{WHP} = \text{total cycles required for completion of program with WHP} * \text{cycle time} \tag{3.53}$$

3.4 Time Model

Table 3.3 shows the access time components used for evaluating performance parameters. Calculations for total access time for various cache architectures

Table 3.3: SESC Components for Time Modeling of Cache

Cache Components	Time Components		Access Time (T_{access})
	Data Side	Tag Side	
Decoder	T_{ds_dec}	T_{ts_dec}	$T_{access} = \max(T_{ds_dec}, T_{ts_dec}, T_{Halt}^*) + T_{Pred}^*$ $+ \max((T_{ds_wline} + T_{ds_bline} + T_{ds_sa}), (T_{ts_wline} + T_{ts_bline} + T_{ts_sa})) + T_{ts_cmp}$ $+ T_{ts_vi} + T_{ds_opDrvr}$
Wordline	T_{ds_wline}	T_{ts_wline}	
Bitline	T_{ds_bline}	T_{ts_bline}	
Sense Amplifier	T_{ds_sa}	T_{ts_sa}	
Comparator	—	T_{ts_cmp}	
Valid Signal Driver	—	T_{ts_vi}	
Output Driver	T_{ds_opDrvr}	—	

*Where T_{halt} and T_{Pred} is the time required for accessing halt tag array and prediction circuit respectively. T_{Halt} and T_{Pred} is 0 for CC, T_{Halt} is 0 for WP, T_{Pred} is 0 for WH

is given in following subsections.

3.4.1 Conventional Cache

CC Read Time :

$$T_{CCR} = T_{access} + C_{RMiss} * T_{Tx} \quad (3.54)$$

Where T_{Tx} is the transfer time from/to processor.

CC Write Time :

$$T_{CCW} = T_{access} * 2 + C_{WMiss} * T_{Tx} \quad (3.55)$$

CC Time :

$$T_{CC} = T_{CCR} + T_{CCW} \quad (3.56)$$

3.4.2 Way Predicting Cache

WP Read Time :

$$\begin{aligned}
T_{WPR} = & T_{access} * C_{PRHit} \\
& + (T_{access} + T_{predRMiss}) * ((1 - C_{RMiss}) - C_{PRHit}) \\
& + T_{Tx} * C_{PMiss_CRMiss}
\end{aligned} \tag{3.57}$$

Where C_{PRHit} , C_{RMiss} and $T_{predRMiss}$ is the prediction read hit rate, cache read miss rate and the additional access time required in case of prediction miss.

WP Write Time :

$$\begin{aligned}
T_{WPW} = & 2 * T_{access} * C_{PWHit} \\
& + (2 * T_{access} + T_{predMiss}) * ((1 - C_{WMiss}) - C_{PWHit}) \\
& + T_{Tx} * C_{PWMiss_CWMiss}
\end{aligned} \tag{3.58}$$

Where C_{PWHit} , C_{WMiss} and $T_{predWMiss}$ is the prediction write hit rate, cache write miss rate and the additional access time required in case of prediction miss. WP Time :

$$T_{WP} = T_{WPR} + T_{WPW} \tag{3.59}$$

3.4.3 Way Halting Cache

WH Read Time :

$$\begin{aligned}
T_{WHR} = & T_{access} * C_{CRHit} + T_{Tx} * C_{HHit_CRMiss} \\
& + T_{HMiss_CRMiss} * C_{HMiss_CRMiss}
\end{aligned} \tag{3.60}$$

Where C_{CRHit} , C_{HHit_CRMiss} , C_{HMiss_CRMiss} and T_{HMiss_CRMiss} is cache read hit rate, halt hit - cache read miss rate, halt read miss and transfer time in case of halt read miss respectively. (where $T_{HMiss_CRMiss} < T_{Tx}$)

WH Write Time :

$$T_{WHW} = 2 * T_{access} * C_{CWHit} + T_{Tx} * C_{HHit_CWMiss} + T_{HMiss_CWMiss} * C_{HMiss_CWMiss} \quad (3.61)$$

Where C_{CWHit} , C_{HHit_CWMiss} , C_{HMiss_CWMiss} and T_{HMiss_CWMiss} is cache write hit rate, halt hit - cache write miss rate, halt write miss and transfer time in case of halt write miss respectively. (where $T_{HMiss_CRMiss} < T_{Tx}$)

WH Time :

$$T_{WH} = T_{WHR} + T_{WHW} \quad (3.62)$$

3.4.4 Way Halted Prediction Cache

WHP Read Time :

$$\begin{aligned} T_{WHPR} = & T_{access} * (C_{HHit1_CRHit} + C_{HHitk_Wp_PHit_CRHit} + C_{HHitk_CRHit}) \\ & + (T_{access} + T_{predRMiss}) * C_{HHitk_Wp_PMiss_CRHit} \\ & + T_{Tx} * (C_{HHit1_CRMiss} + C_{HHitk_CRMiss}) \\ & + (T_{Tx} + T_{predRMiss}) * C_{HHitk_Wp_PMiss_CRMiss} \\ & + T_{HMiss_CRMiss} * C_{HMiss_CRMiss} \end{aligned} \quad (3.63)$$

WHP Write Time :

$$\begin{aligned}
T_{WHPW} = & 2 * T_{access} * (C_{HHit1.CWHit} + C_{HHitk-Wp_PHit.CWHit} + C_{HHitk.CWHit}) \\
& + (2 * T_{access} + T_{predWMiss}) * C_{HHitk-Wp_PMiss.CWHit} \\
& + T_{Tx} * (C_{HHit1.CWMiss} + C_{HHitk.CWMiss}) \\
& + (T_{Tx} + T_{predWMiss}) * C_{HHitk-Wp_PMiss.CWMiss} \\
& + T_{HMiss.CWMiss} * C_{HMiss.CWMiss}
\end{aligned} \tag{3.64}$$

WHP Time :

$$T_{WHP} = T_{WHPR} + T_{WHPW} \tag{3.65}$$

3.5 Experimental Setup

This work uses SESC, simulation framework to model L1 cache of different set-associative architectures. Simulation framework configures SESC simulator using smp.conf file. In smp.conf, SESC simulator accepts various configuration parameters for Data TLB, Instruction TLB, L1 Data Cache, L1 Instruction cache, L2 cache, main memory and processor configuration. The simulation framework implemented in C++ estimates processing time, power/energy components for fetch, issue, memory access, execution and clock. It also estimates statistic of read/write access for TLB, L1 Cache, L2 Cache, main memory and branch prediction circuit. Various configurations of set-associative caches like CC, WP, WH and WHP are implemented by using SESC simulator. The framework uses 32-bit address and LRU replacement policy. As all the experimenting cache architectures use same cache configu-

ration parameters and replacement policy, their cache hit rate remains the same. The prediction circuit stores the most recently used (MRU) cache line for future prediction.

SESC simulator uses Splash benchmark programs to evaluate the performance of various cache architectures. SESC uses CACTI [67] to estimate time and energy parameters of cache.

3.6 Experimental Analysis

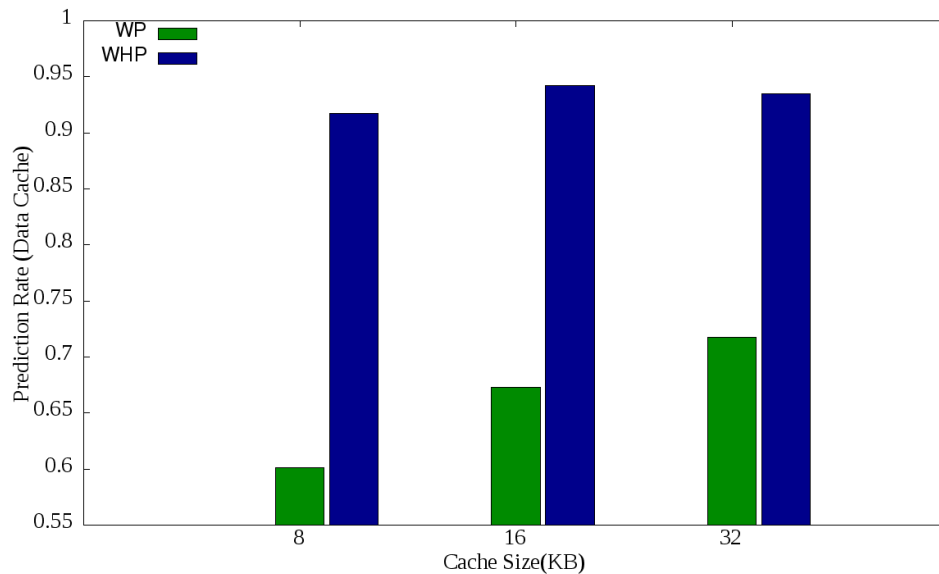


Figure 3.3: Prediction Rate for a 4 way, 8B line size with varying Data Cache size

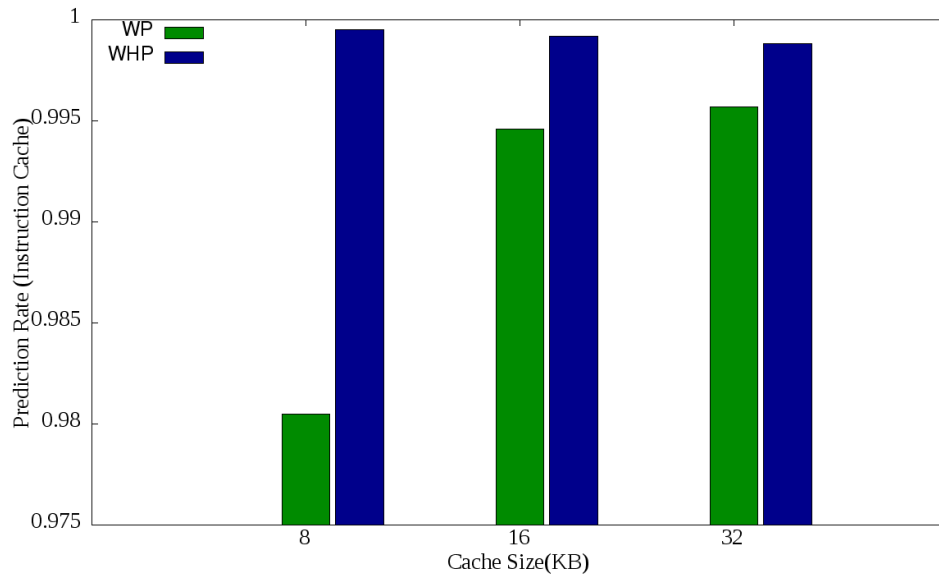


Figure 3.4: Prediction Rate for a 4 way, 8B line size with varying Instruction Cache size

3.6.1 Prediction Hit Accuracy

Figures 3.3 and 3.4 show the data cache and instruction cache prediction hit rate of WP and WHP for varying cache sizes. Irrespective of the configurations, WHP offers higher prediction rate than WP because of early cache miss detection and corner case elimination. An average prediction hit rate of data cache and instruction cache for WP is 66.39% and 99.03% whereas for WHP it is 93.17% and 99.92% respectively.

3.6.2 Dynamic Energy per Access

Figures 3.5, 3.6, and 3.7 show the data cache, instruction cache and combined dynamic energy consumption for various SPLASH benchmark programs respectively. It is observed that prediction accuracy is higher for instruction

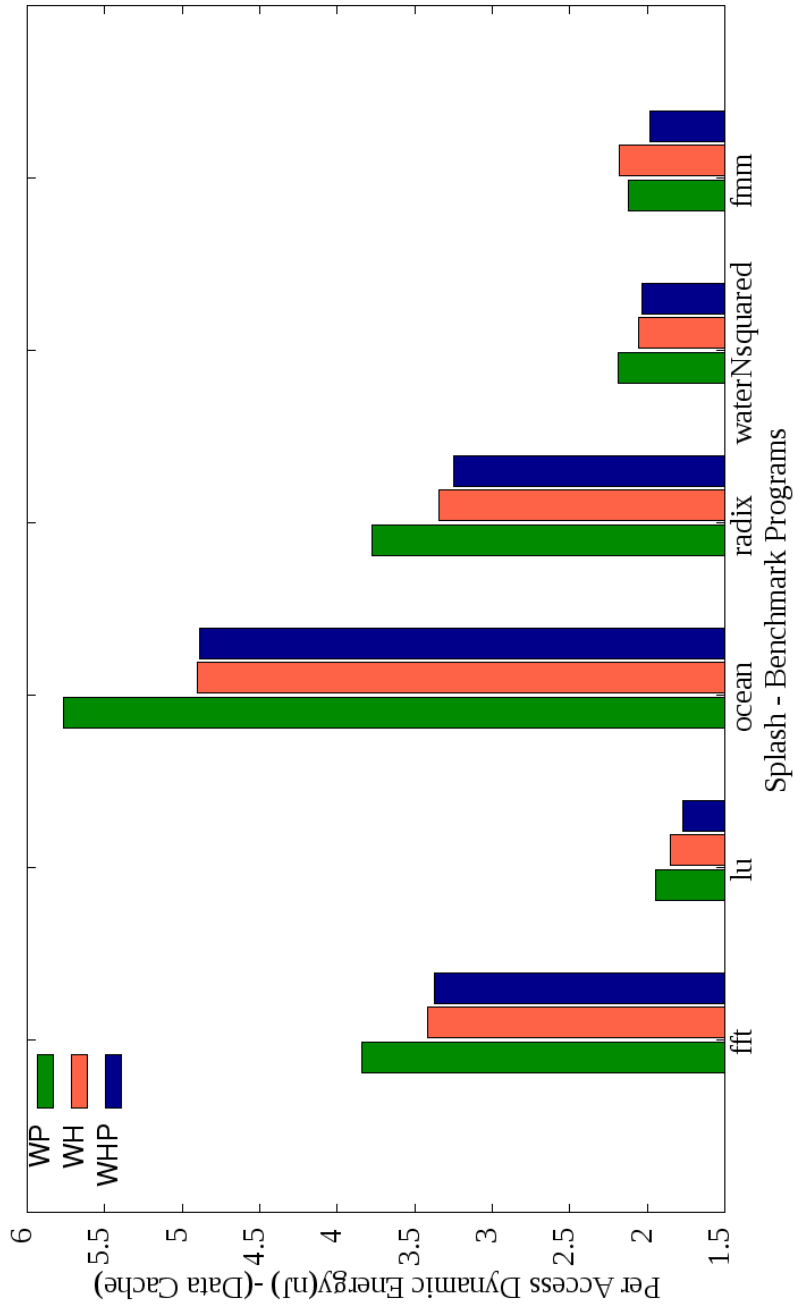


Figure 3.5: Per access dynamic energy consumption for a 4 way, 8B, 32KB Data Cache with varying benchmark programs

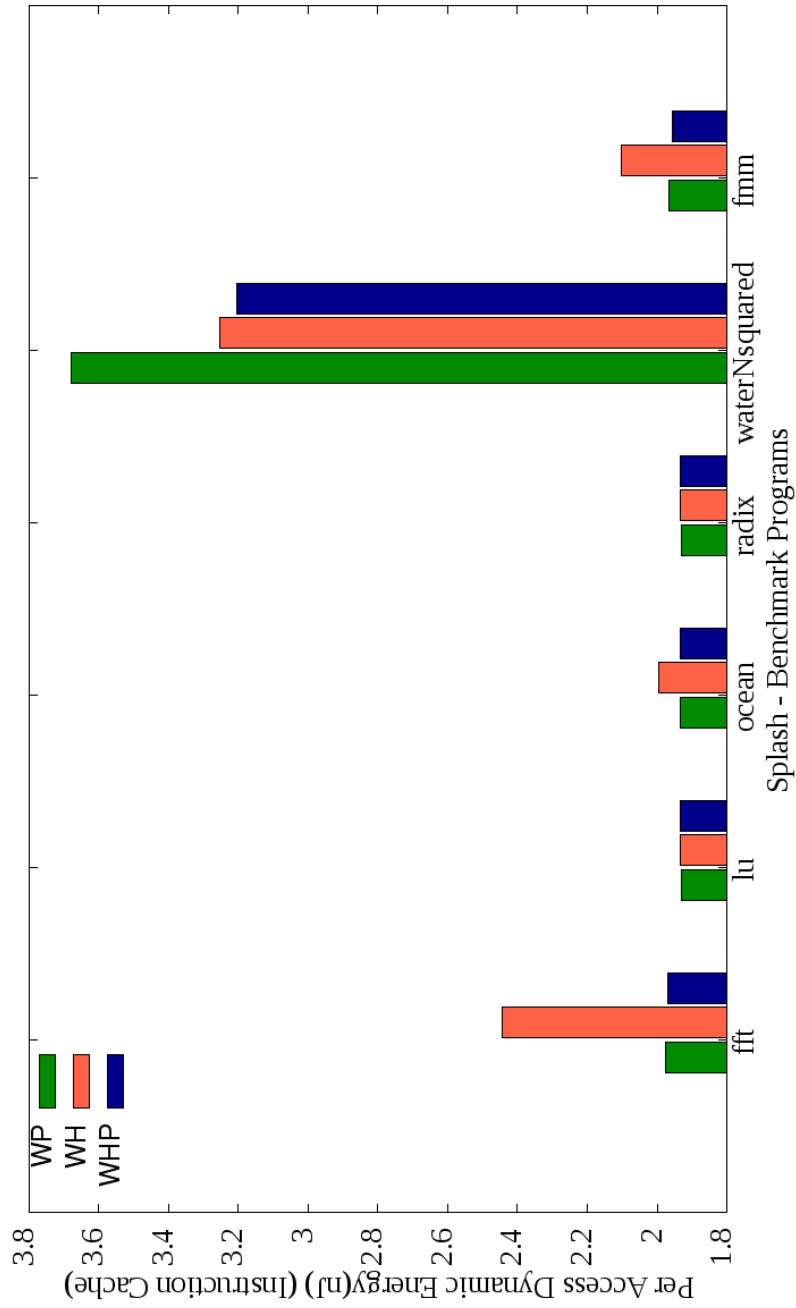


Figure 3.6: Per access dynamic energy consumption for a 2 way, 8B, 32KB Instruction Cache with varying benchmark programs

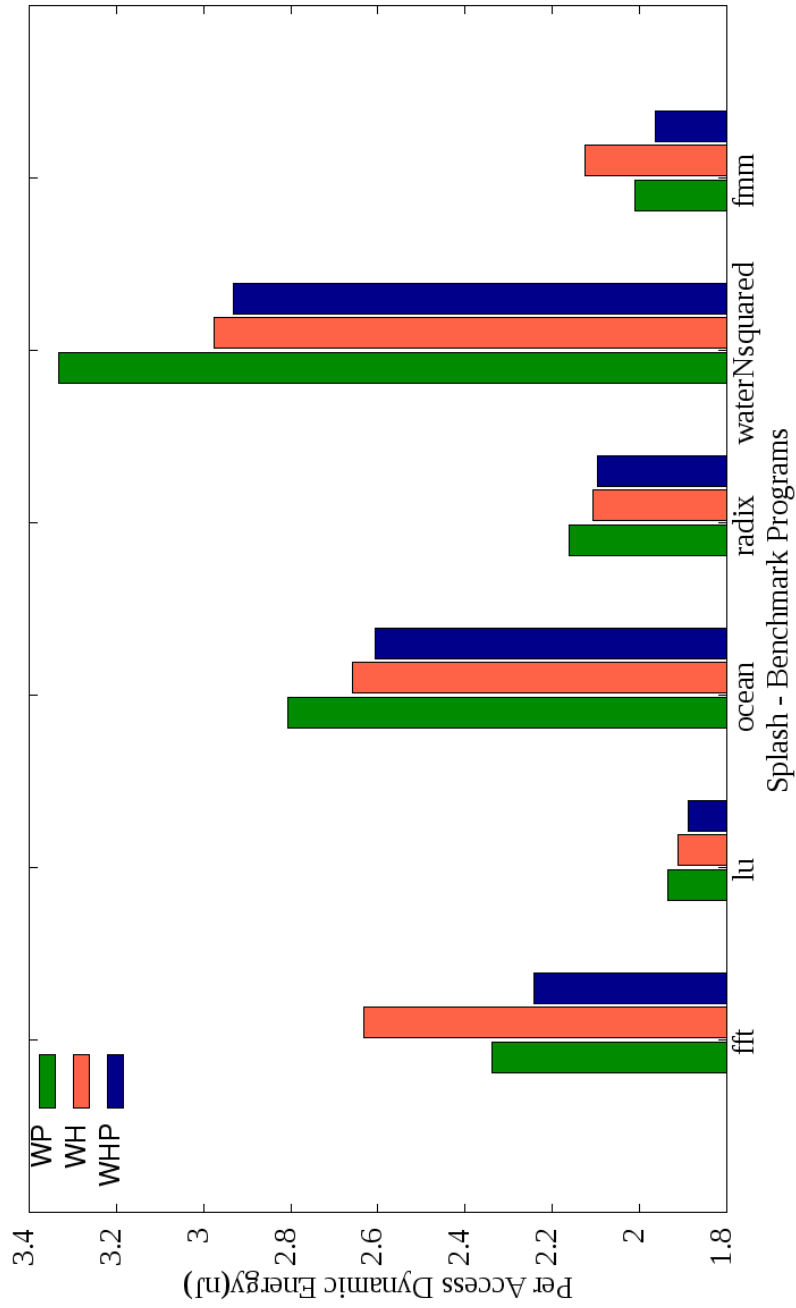


Figure 3.7: Per access dynamic energy consumption for a 4 way, 8B, 32KB cache with varying benchmark programs

cache over data cache. It is also observed that for all benchmark programs under consideration WHP's energy consumption is the least. Figure 3.8 shows dynamic energy savings over CC for various benchmark programs. Results show that all energy efficient caches consume less dynamic energy as compared to CC. The average dynamic energy saving of WP, WH and WHP over CC are 43.06%, 44.06%, 46.64% respectively. Experimental evaluation reveals that WHP achieves on an average 6.45% and 4.15% of dynamic energy over WP and WH respectively. The per access dynamic energy consumption for all the architectures increase with increase in cache size as shown in figure 3.9. Hit rate saturates with increase in cache size this results in increase dynamic energy consumption.

Irrespective of the cache architecture in use, the per access dynamic energy consumption decreases when associativity is changed from 4-way to 8-way. This is due to reduction in conflict misses. However, with further increase in associativity to 16-way output driver dynamic energy consumption out-trades hit rate improvement. Figure 3.10 shows per access dynamic energy consumption for various cache architecture by varying cache associativity. Increasing line size reduces number of compulsory misses. As shown in Figure 3.11 with increase in cache line size for all cache architecture per access dynamic energy consumption reduces.

3.6.3 Response Time

Figures 3.12, 3.13, and 3.14 show the data cache, instruction cache and combined response time for various SPLASH benchmark programs respectively. Figure 3.15 shows response time saving of WP, WH and WHP over CC. Figures

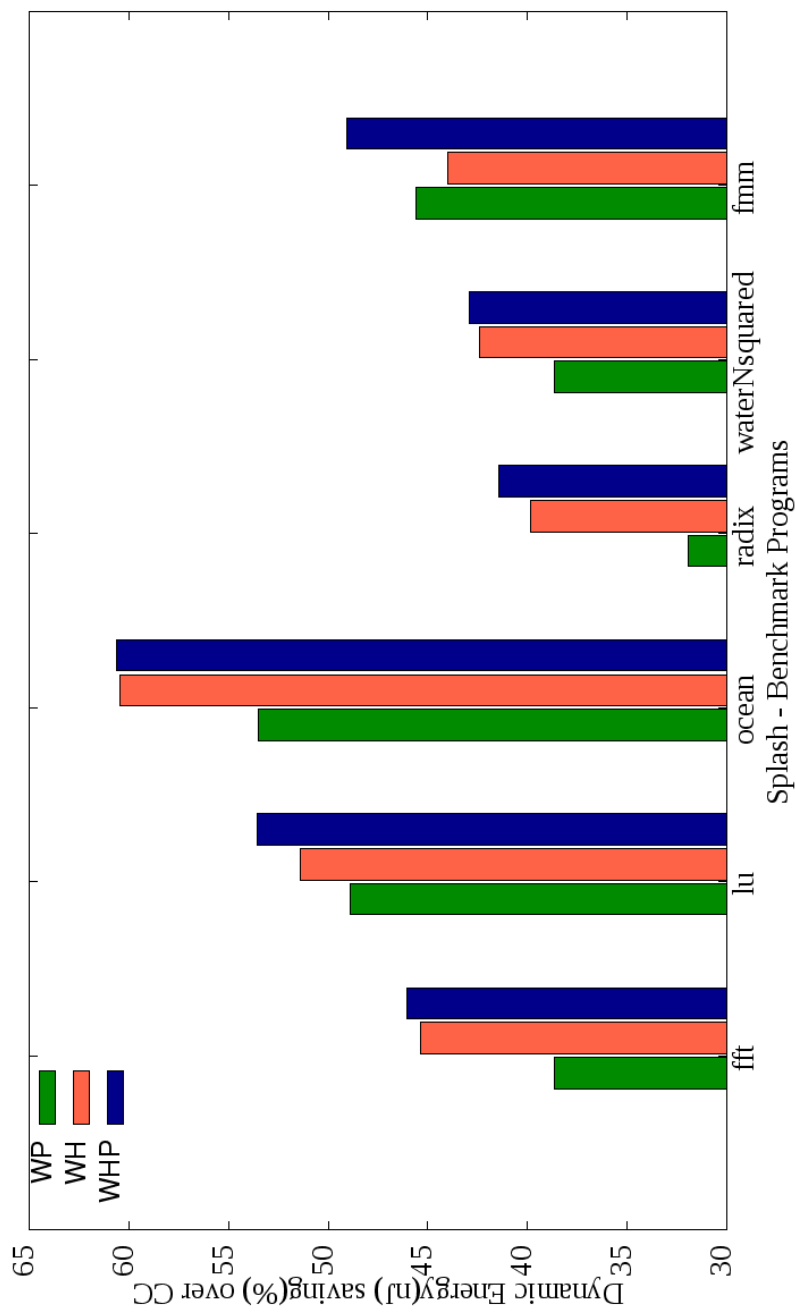


Figure 3.8: Per access dynamic energy savings for a 4 way, 8B, 32KB cache with varying benchmark programs over CC

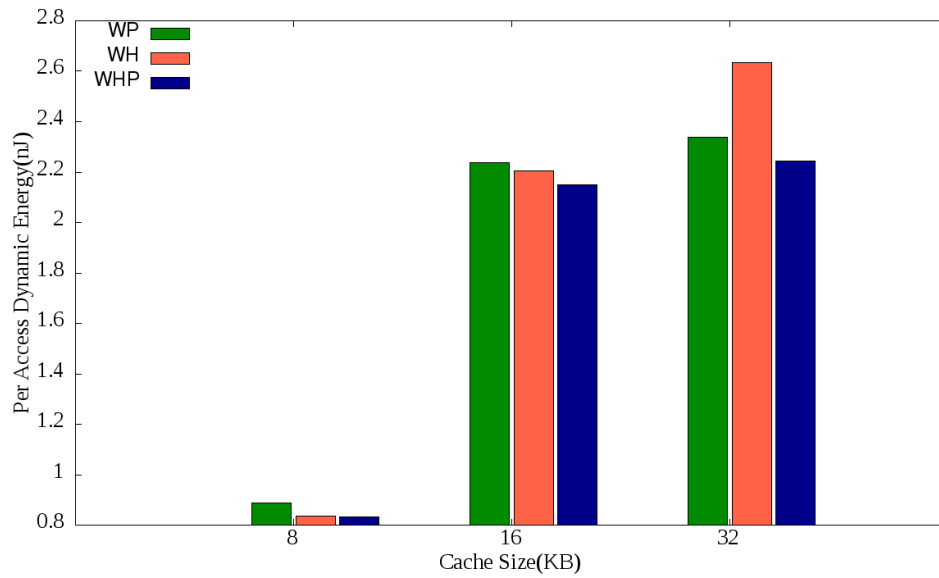


Figure 3.9: Per access dynamic energy consumption for a 4 way, 8B line size with varying cache size

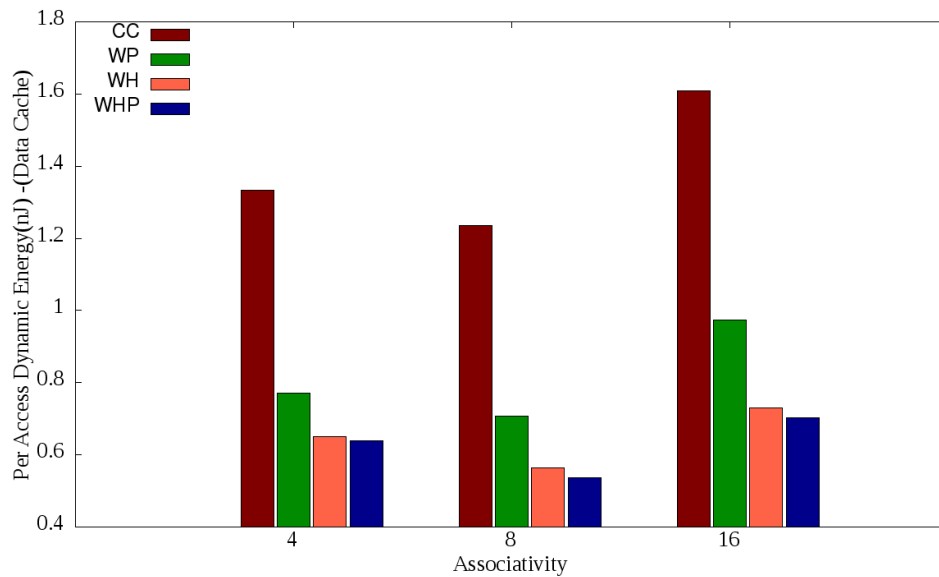


Figure 3.10: Per access dynamic energy consumption for a 16B, 8KB Data Cache with varying associativities

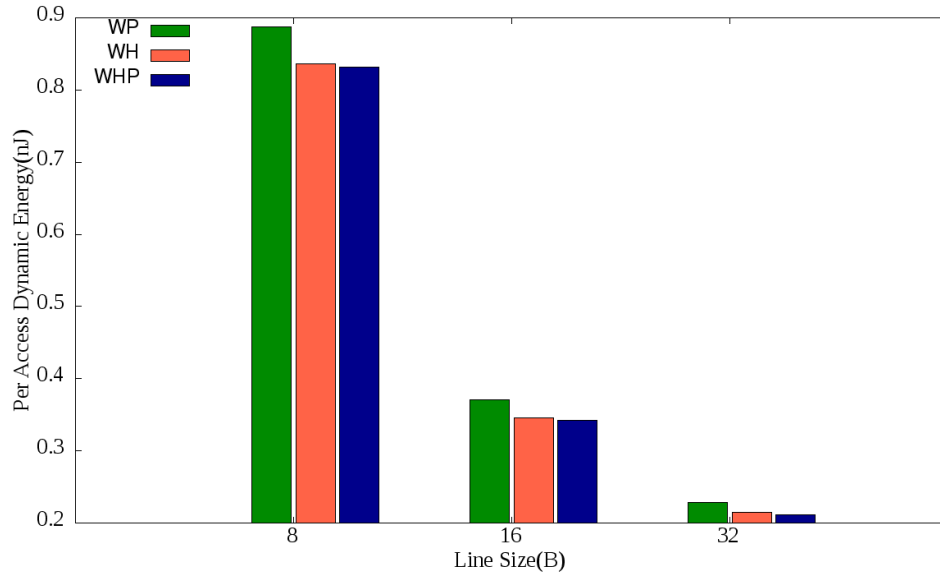


Figure 3.11: Per access dynamic energy consumption for a 4 way, 8KB cache with varying line Size

3.16, 3.17 and 3.18 show response time by varying cache size, line size and data cache associativity respectively. Irrespective of the configuration in use, the response time of WP is the highest among all the cache architectures and the response time of WH is the least among all the cache architectures.

WP needs additional cycles to access remaining ways in case of prediction miss hence its response time increases. The response time of WH cache is equivalent to CC in case of halt hit and it saves way access time in case of halt miss. Hence WH cache gives the least response time. The response time of WHP is better than CC due to halt tag miss scenarios. The response time of WHP is marginally higher than WH due to additional prediction miss time. The average response time saving of WP, WH and WHP over CC are -2.00%, 1.09% and 1.04% respectively. Experimental evaluation reveals that WHP achieves on an average response time savings of 2.92% and -0.05% over WP

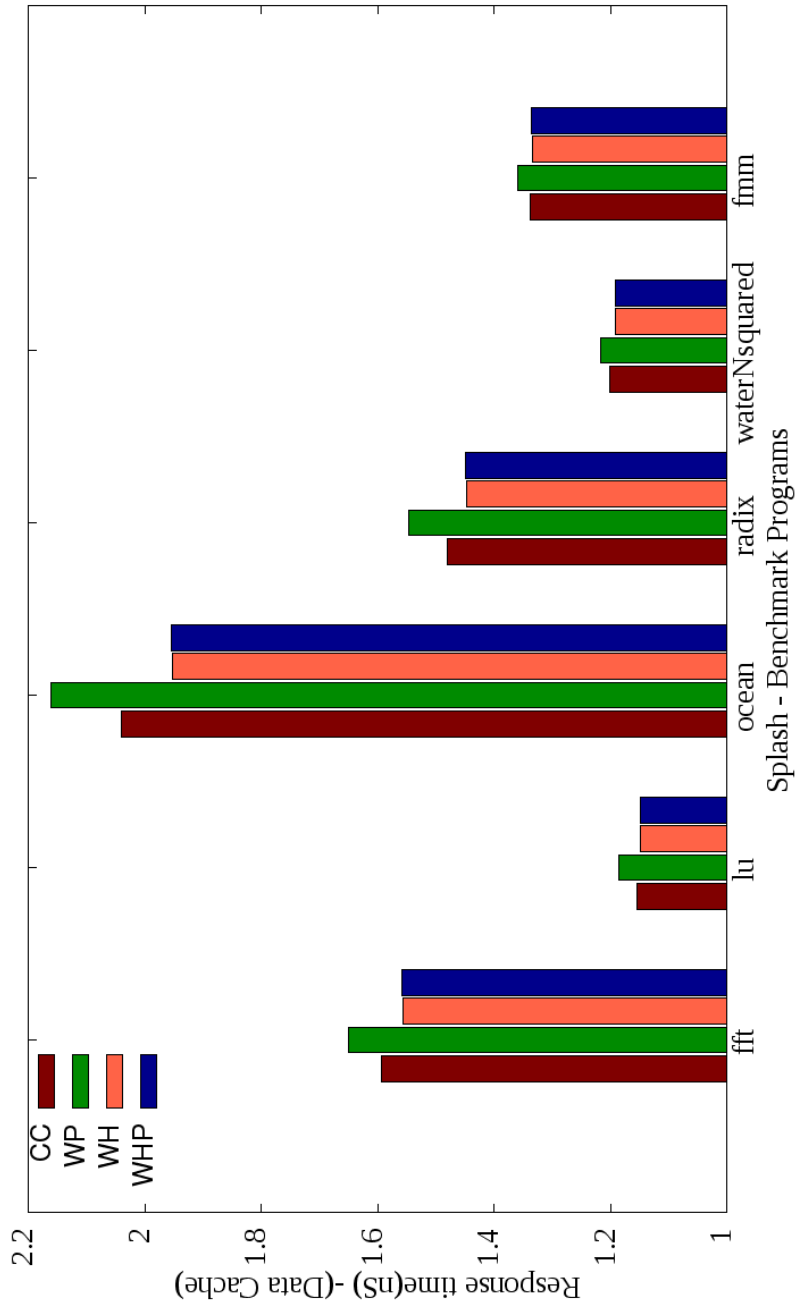


Figure 3.12: Response time for a 4 way, 8B, 32KB Data Cache with varying benchmark programs

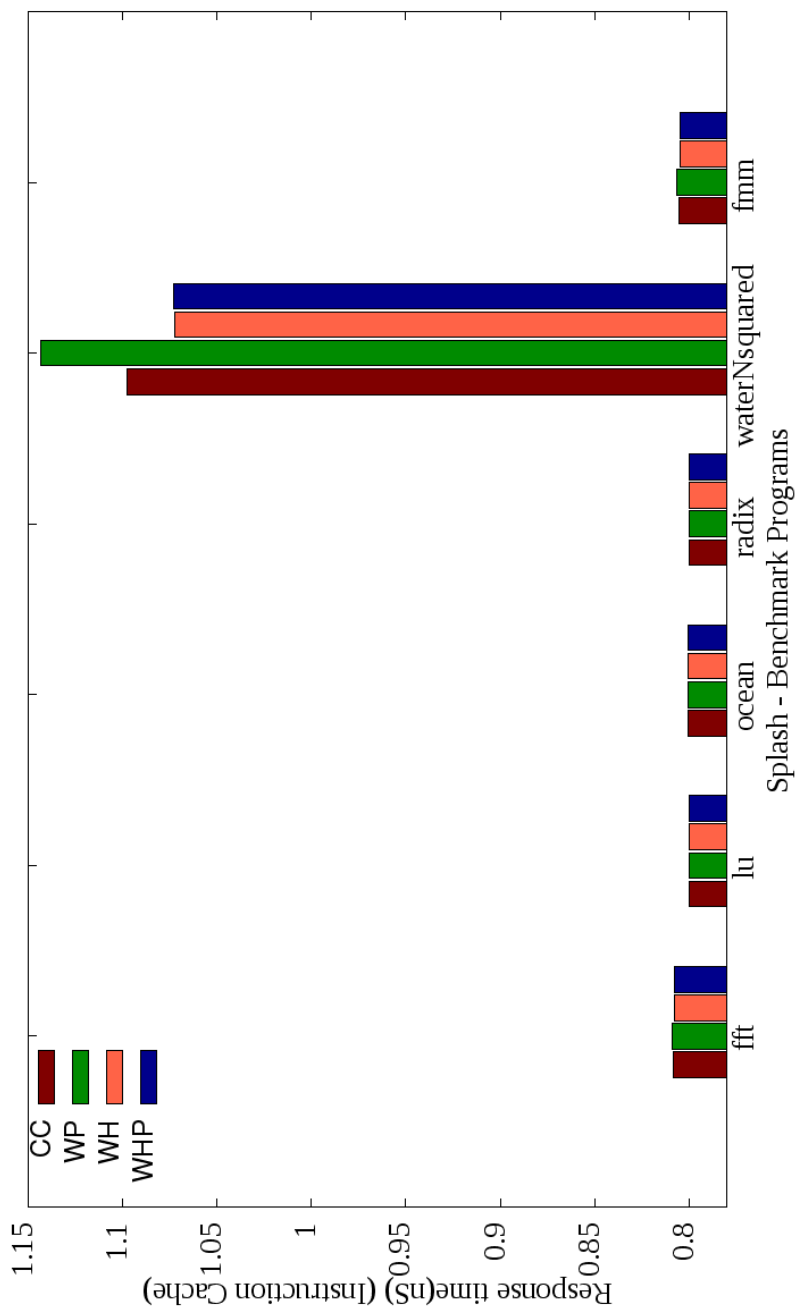


Figure 3.13: Response time for a 2 way, 8B, 32KB Instruction Cache with varying benchmark programs

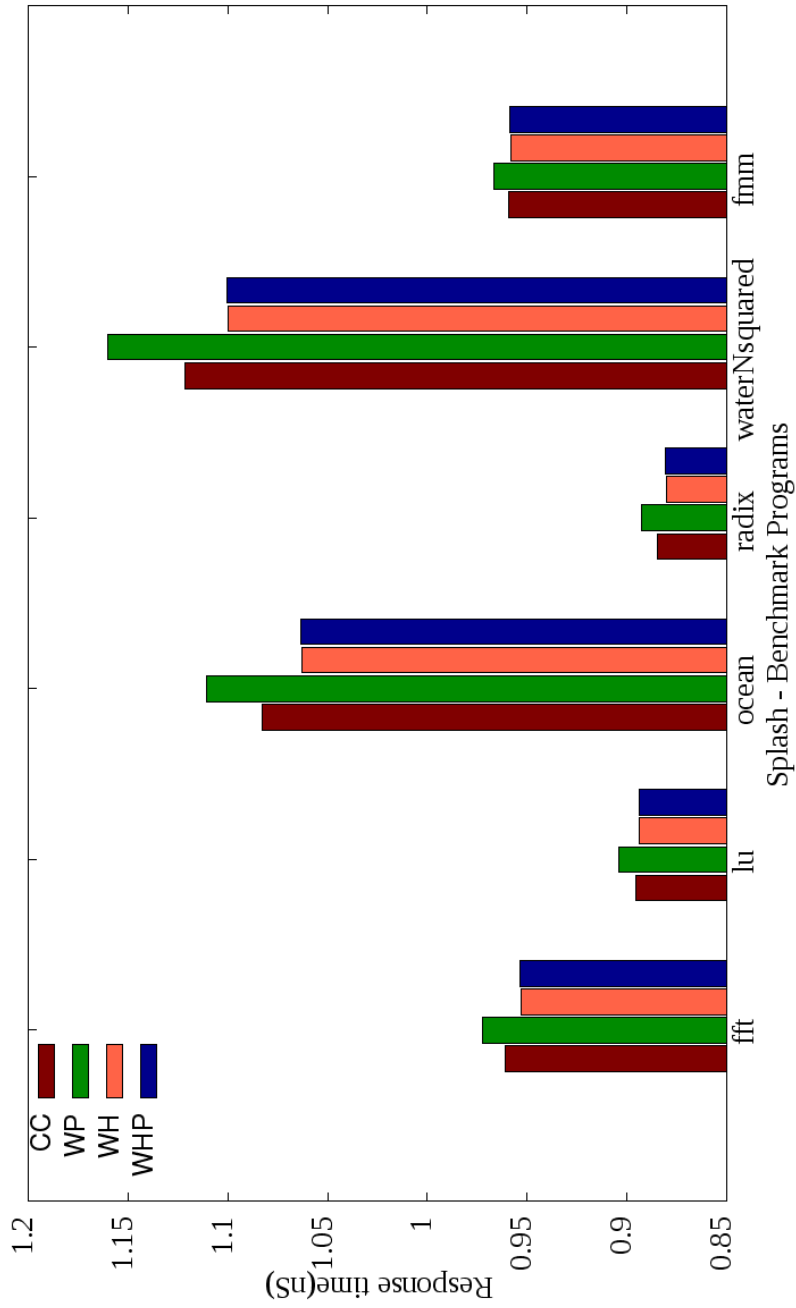


Figure 3.14: Response time for a 4 way, 8B, 32KB cache with varying benchmark programs

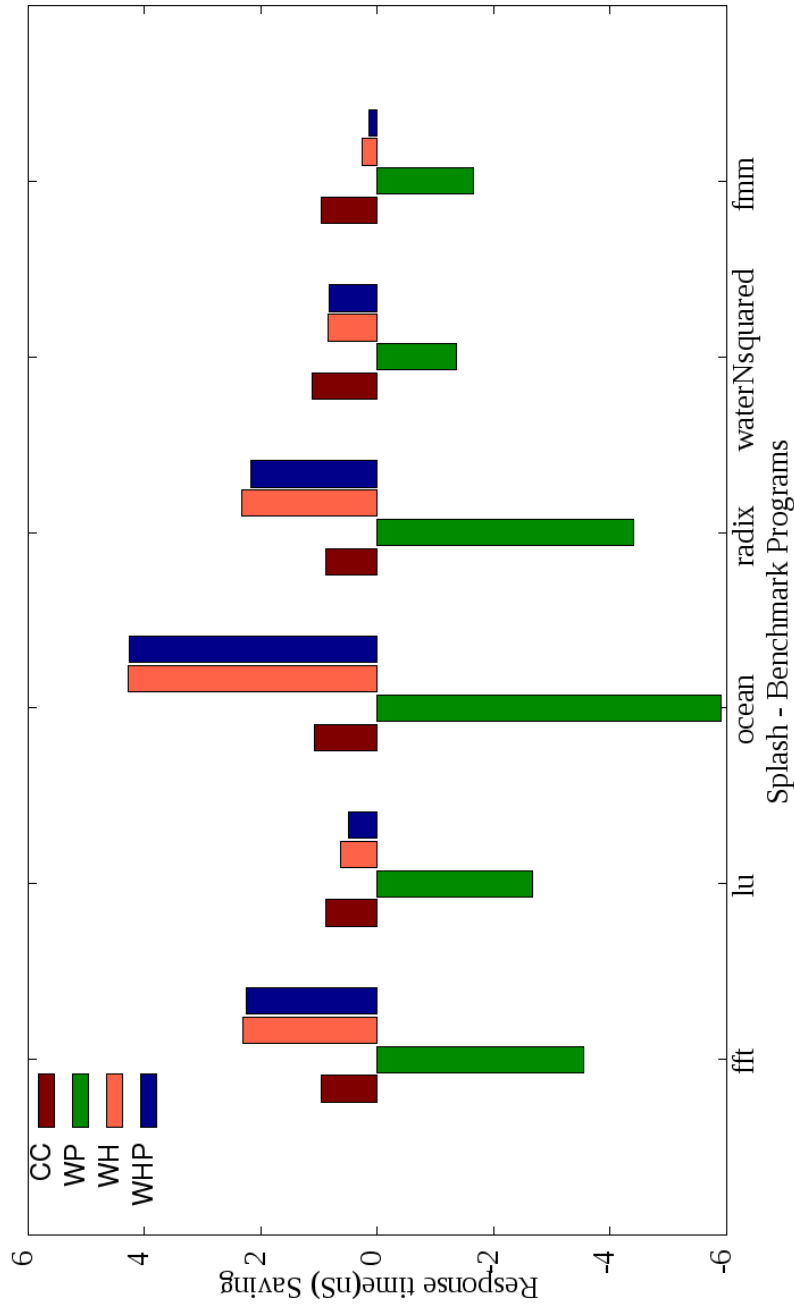


Figure 3.15: Response time saving over CC for a 4 way, 8B, 32KB cache with varying benchmark programs

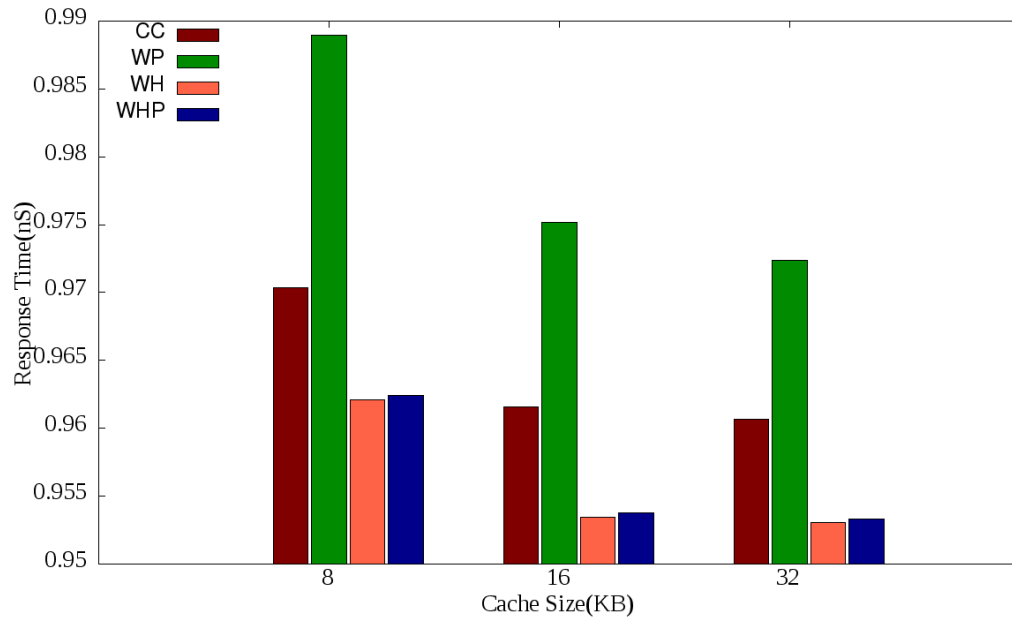


Figure 3.16: Response time for a 4 way, 8B line size with varying cache size

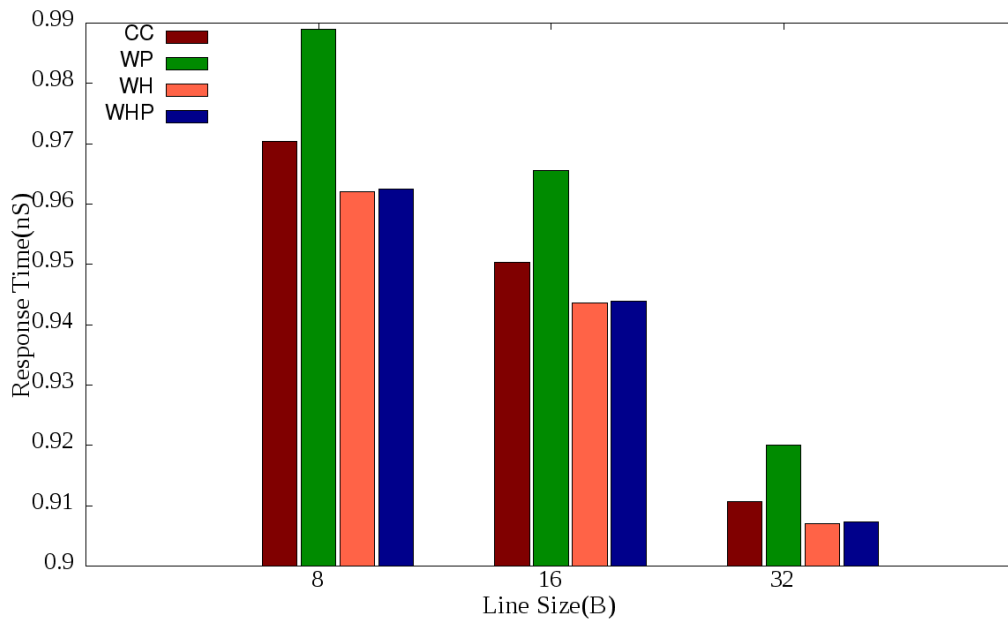


Figure 3.17: Response time for a 4 way, 8KB cache with varying line Size

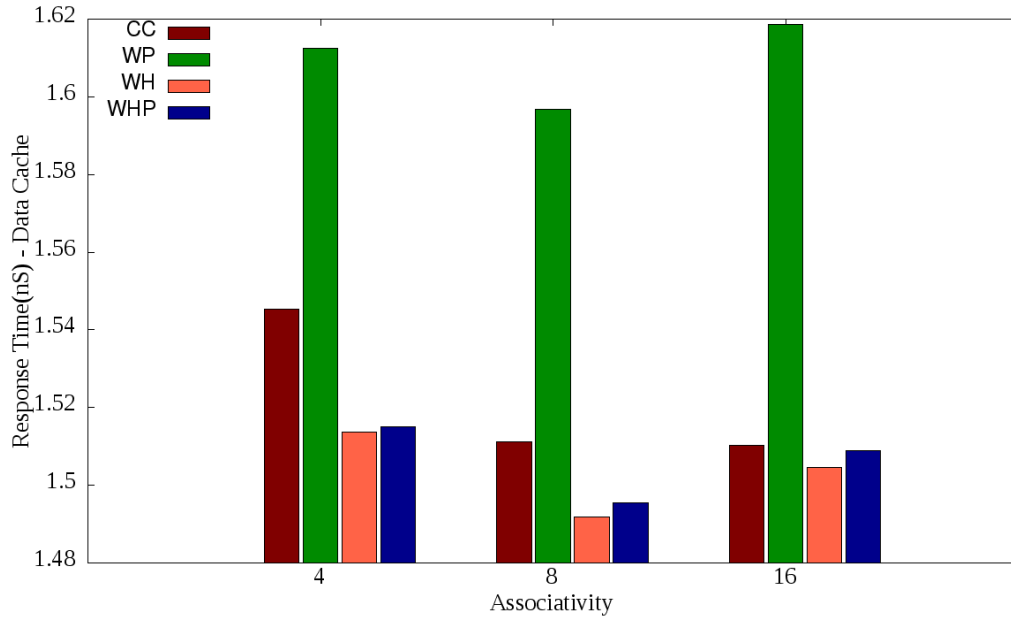


Figure 3.18: Response time for a 16B, 8KB Data Cache with varying associativity

and WH respectively.

3.6.4 Static Energy Per Access

Figure 3.19 shows the static energy consumption of various cache architectures for SPLASH benchmark programs. Prediction circuit and halting circuit adds on to the leakage power of WP and WH respectively. WHP circuit has an overhead of prediction and halting circuit. The response time outrades these additional overheads. The static energy per access follows the response time pattern for all the architectures.

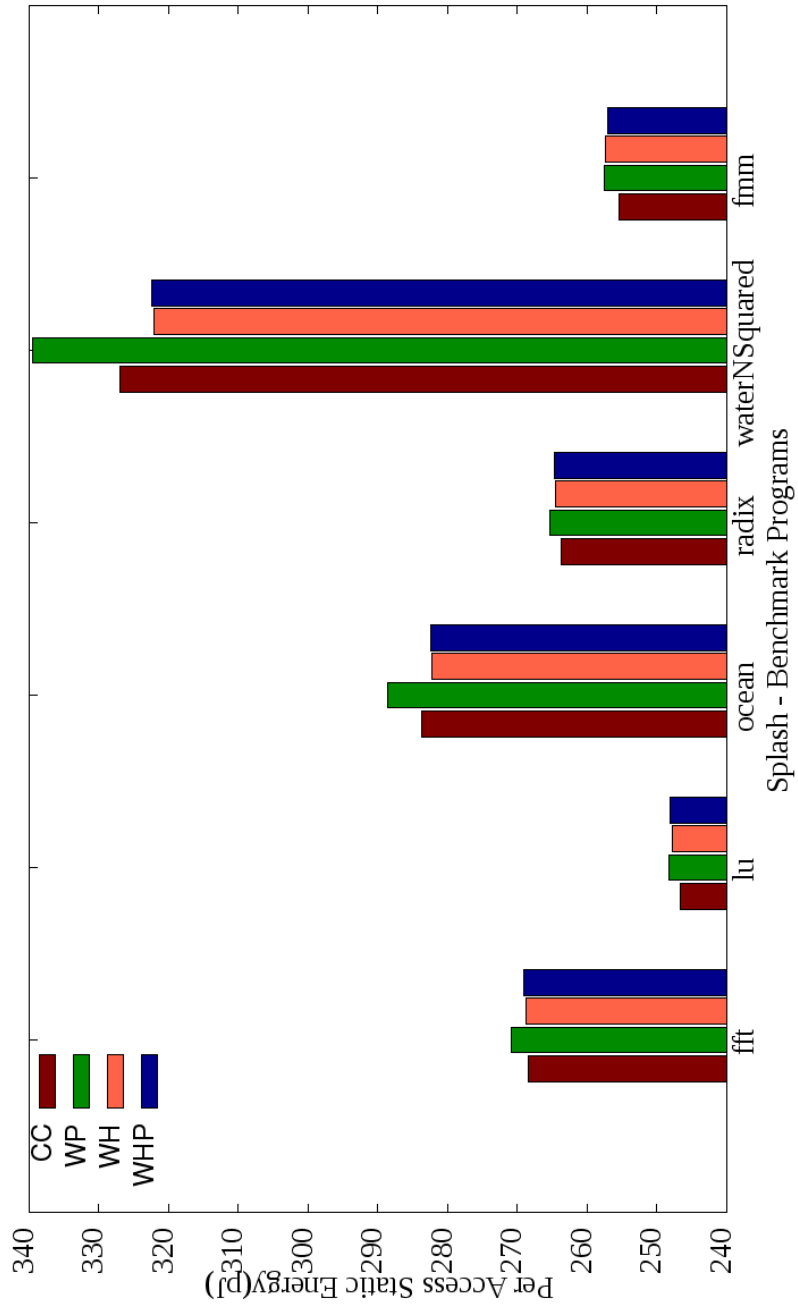


Figure 3.19: Per access static energy consumption for a 4 way, 8B, 32KB cache with varying benchmark programs

3.6.5 Time and Area Overhead

Access time of prediction circuit is measured as 0.16% of the total access time for 8KB, 8B, 4-way WP and WHP. Inoue [4] proposed a mode to bury this time with the previous pipeline stages. Though this work did not bury it with previous pipeline stages, this additional time did not increase number of cycles required to access cache and hence did not affect critical path delay. The area required for CC of 8KB, 8B, 4-way measured as 0.179mm² using SESC. The WH and WHP imposes 2% additional area overhead for the halt tag circuit. WP and WHP imposes 0.2% of area overhead for MRU and associated circuits. The overall area overhead of WHP is measured as 2.24% over CC.

3.7 Conclusion

WHP uses halt tag array and prediction circuit to achieve reduced energy consumption and response time. The combination of halt tag and prediction circuit reduces the number of ways to be activated for cache access. In WHP, the number of active ways are reduced from k ways to one way in most of the accesses with the prediction circuit. As the prediction circuit is enabled only when $k > 1$, the performance of WHP cache is improved with respect to energy and time. The results show that WHP offers better energy efficiency over the other architectures analyzed. WHP offers 46.64%, 6.45% and 4.15% dynamic energy saving and 1.04%, 2.92% and -0.05% saving in response time over the CC, WP and WH respectively. The overall area overhead of WHP is measured as 2.24% over CC.

Chapter 4

MOESIF : Cache Coherency Protocol

4.1 Introduction

The memory subsystem of energy efficient multicore embedded processors usually contain private split L1 caches, unified L2 caches and unified shared L3 cache. This chapter concentrates on improving cache performance by redesigning the most widely used cache coherency protocols, MOESI and MESIF. Cache controller of all cores containing valid data responds to read or write miss request in MOESI protocol. This generates redundant responses which flood the network. This results in increased data traffic and thus response time. The cache coherency traffic has to be be optimised by eliminating the redundant responses. In MESIF protocol, if any core sends read or write request for the modified data, the data is first written back to L3 cache and then the requesting core receives it from L3 cache. The time and energy required for transferring the data from L2 cache of some core to L3 cache and then from L3 cache to the requesting core's L2 cache can be optimised by sharing dirty copy of data.

This chapter proposes an energy efficient cache coherency protocol - **Modified**

Owned **E**xclusive **S**hared **I**nvalid **F**orward - MOESIF. The redundant responses are concisely narrowed down in MOESIF protocol. In MOESIF protocol, only the cache controller of the core in M, O or F state responds with the requested data. This results in reduction of total number of responses thus traffic and response time. Response time is further reduced in MOESIF protocol by sharing dirty copy of data. The core having modified data forwards dirty copy of data to the requesting core without updating L3 cache. This L2 to L2 transfer reduces the number of write backs to the next level memory. MOESIF achieves energy efficiency and high performance by optimizing data transfers between caches and with next level memory. It improves off-chip and on-chip bandwidth usage. MOESIF reduces the number of write backs to next level memory and the number of responders to a cache miss when multiple copies of data exists in private caches.

4.2 Widely Used Cache Coherence Protocols

Invalidation based cache coherence protocols maintain a state for each cache line along with the tag and data. The coherence policy is defined by the finite state machine in each node which changes the state of cache line based on read/write operation. Following subsections briefs some of the widely used cache coherence protocols namely MESI, MOESI and MESIF.

4.2.1 MESI Protocol

Modified-Exclusive-Shared-Invalid (MESI) is the four state protocol in use for maintaining cache coherency in MC/MP systems. The cache line in invalid state implies that the data present in the cache line is not valid. The shared state allows multiple nodes to have the same data block in consistent state with the next level memory. The cache line in E state holds an exclusive copy of data. Data from cache line in shared / exclusive state can only be read. If the cache line is in modified state, then it is the most up-to-date and the only valid copy available.

In case of write hit, the invalidation signal is sent to all other cores only when the cache line is in S state. Read/write misses are satisfied by transferring data to the cache either from other nodes or from the next level memory. If any node has the requested data in E or S state, the requestor is serviced by the node upon broadcast. If a node contains data with M state then it updates the next level memory with that data. All the cache lines containing the requested data will be in S state for read miss and I state for write miss. The state of requestor's cache line is set to E or S state and M state for read miss and write miss respectively. In read miss the requestor's copy will be in E state only if the data is transferred from next level memory. Presence of multiple shared copies reduces the number of coherence misses in MESI. However, for every state change from M, the data needs to be written to the next level. This result in frequent data writes between node and next level when read and write alternate. The state transition diagram of MESI protocol is shown in figure 4.1.

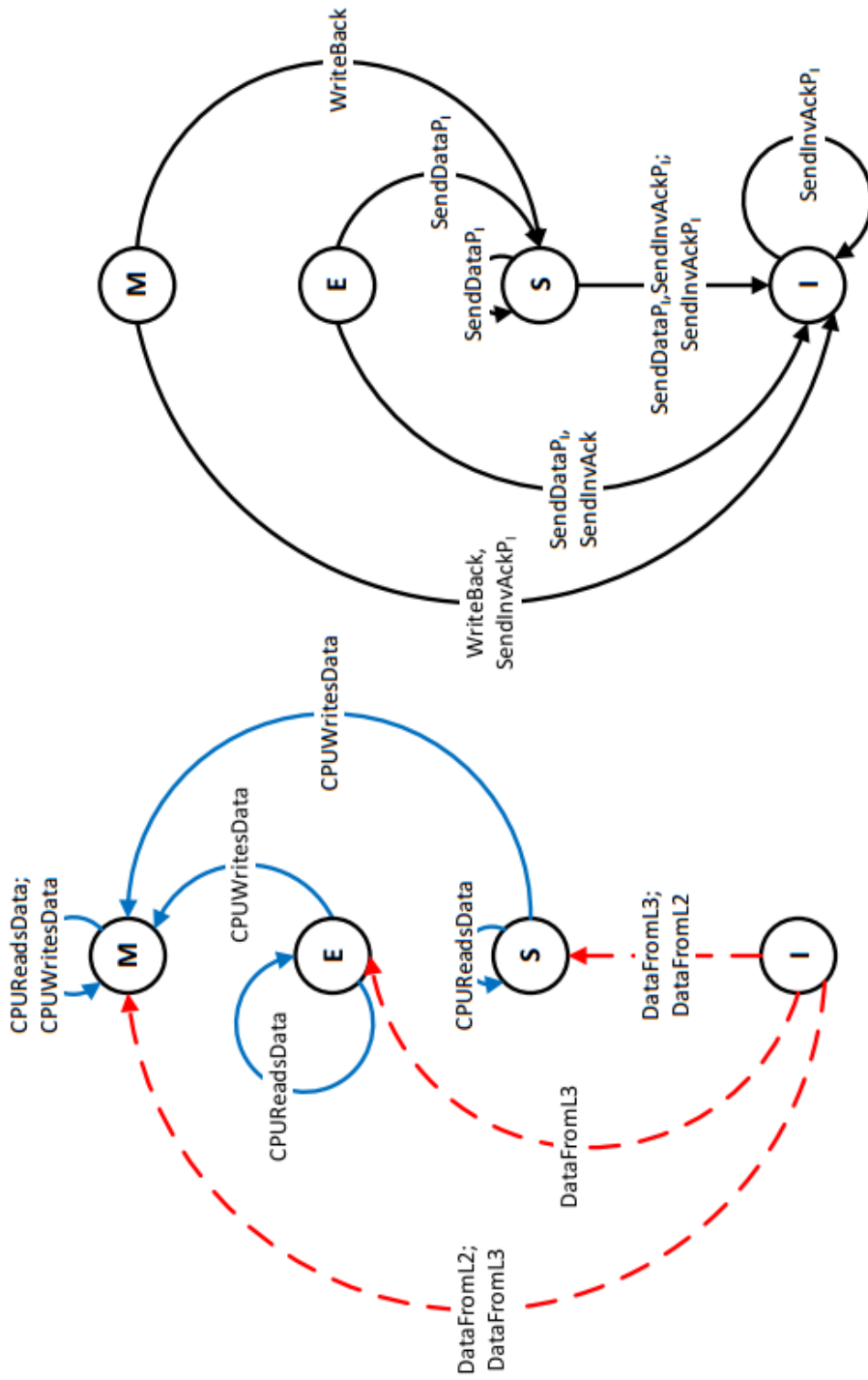


Figure 4.1: MESI Access and Snoop State Transitions

4.2.2 MOESI Protocol

The M state in MOESI protocol, allows transfer of data to a read requestor without writing back to the next level memory. The dirty copy sharing between different nodes using O state helps to reduce the frequency of write-backs. Write back in MOESI happens only when the cache line in M or O state is replaced. However, during a cache miss, all the sharers of requested data will respond and the requestor will be serviced by redundant responses. These redundant responses increase the traffic across the network as data transfer takes more bandwidth than signals. The cache state transitions for a cache access and snoop is shown in figure 4.2.

4.2.3 MESIF Protocol

The issue of redundant responses from all the sharers in MESI and MOESI is addressed by the F state. MESIF protocol ensures that only the cache line in F state responds to read/write request of other nodes. Among the sharers, the last recipient of data is assigned with F state. During read miss, the cache line in F state transfers the data and updates its state to S. Although redundant responses are eliminated using F state, sharing of dirty data is not allowed in MESIF. Write back happens for all the state change from M state. The cache state transitions for a cache access and snoop is shown in figure 4.3.

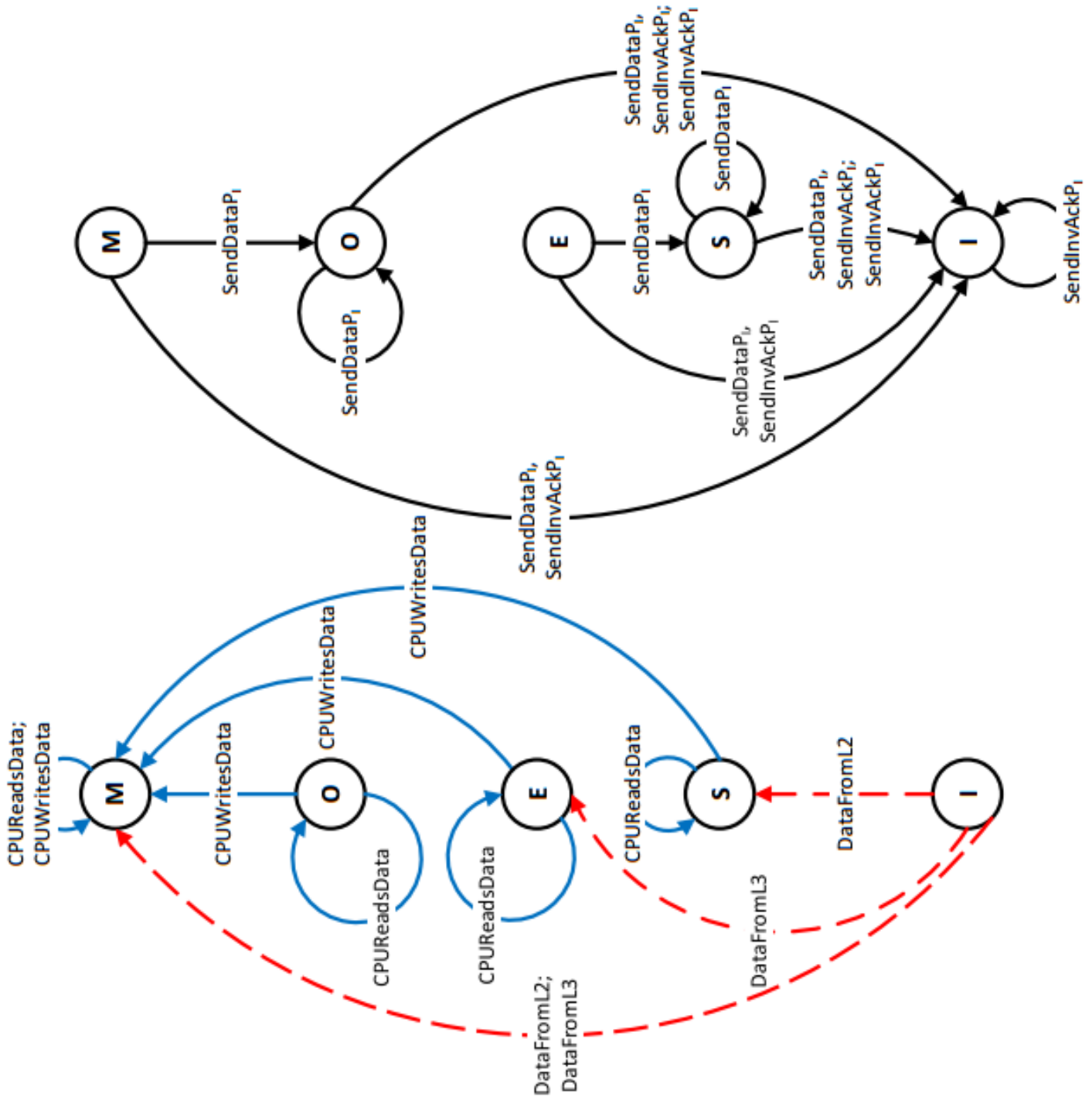


Figure 4.2: MOESI Access and Snoop State Transitions

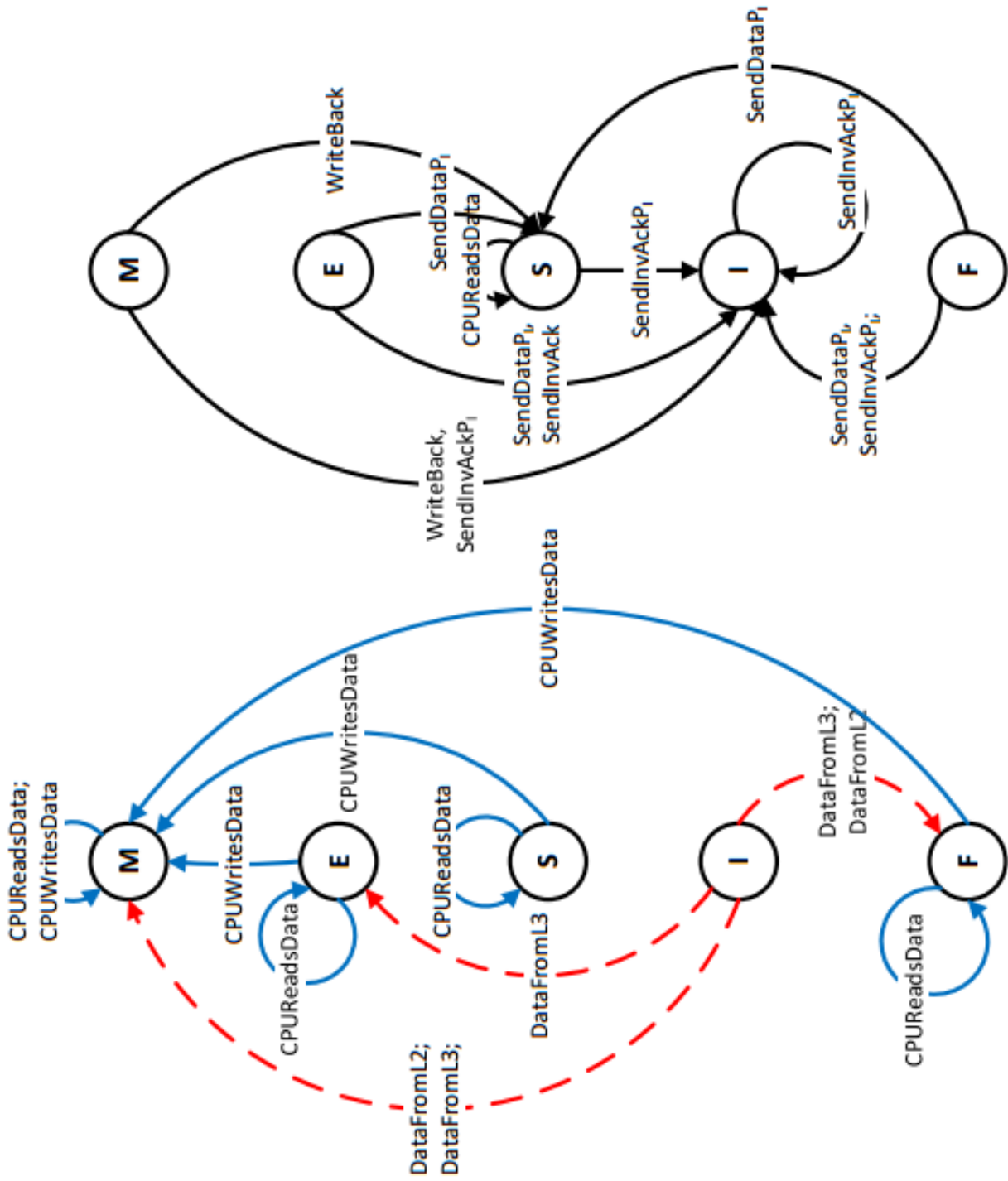


Figure 4.3: MESIF Access and Snoop State Transitions

4.3 MOESIF Architecture

MOESIF achieves energy efficiency and high performance by optimizing data transfers between caches and with next level memory. This is achieved by reducing the number of write backs to the next level memory and making sure only one responder sends data to the requestor. Combining O state and F state with MESI states helps in achieving this. This results in reduced traffic which in turn improves response time of the system. The quad-core processor architecture this work uses is shown in Figure 4.4. Each core has private split L1 cache, private unified L2 cache, shared L3 cache and main memory. Cores are connected to L3 cache using bus interconnect. Cores are connected to other cores using point to point interconnect. Figure 4.5 and

Table 4.1: Read, Write and Snoop operations in MOESIF Protocol

		Read $P_I \rightarrow$						Write $P_I \rightarrow$					
		M	O	E	S	I	F	M	O	E	S	I	F
SNOOP $\leftarrow P_j$	M	-	-	-	-	O/S	-	-	-	-	-	M/I	-
	O	-	-	-	S/O	O/S	-	-	-	-	M/I	M/I	-
	E	-	-	-	-	F/S	-	-	-	-	-	M/I	-
	S	-	O/S	-	S/S	F/S	F/S	-	M/I	-	M/I	M/I	M/I
	I	M/I	O/I	E/I	S/I	E/I	F/I	M/I	M/I	M/I	M/I	M/I	M/I
	F	-	-	-	S/F	F/S	-	-	-	-	M/I	M/I	-

4.6 show the access and snoop function in MOESIF. Corresponding state transitions for a pair of caches (P_I and P_j) for read and write operations in P_I is shown in Table 4.1.

In MOESIF protocol, one of the sharers will be an owner (O state) or a forwarder (F state). If multiple copies of shared data exist then the broadcasted cache miss signal is satisfied either by the owner or by the forwarder. If the cache line is in O state, all the copies in S state and itself

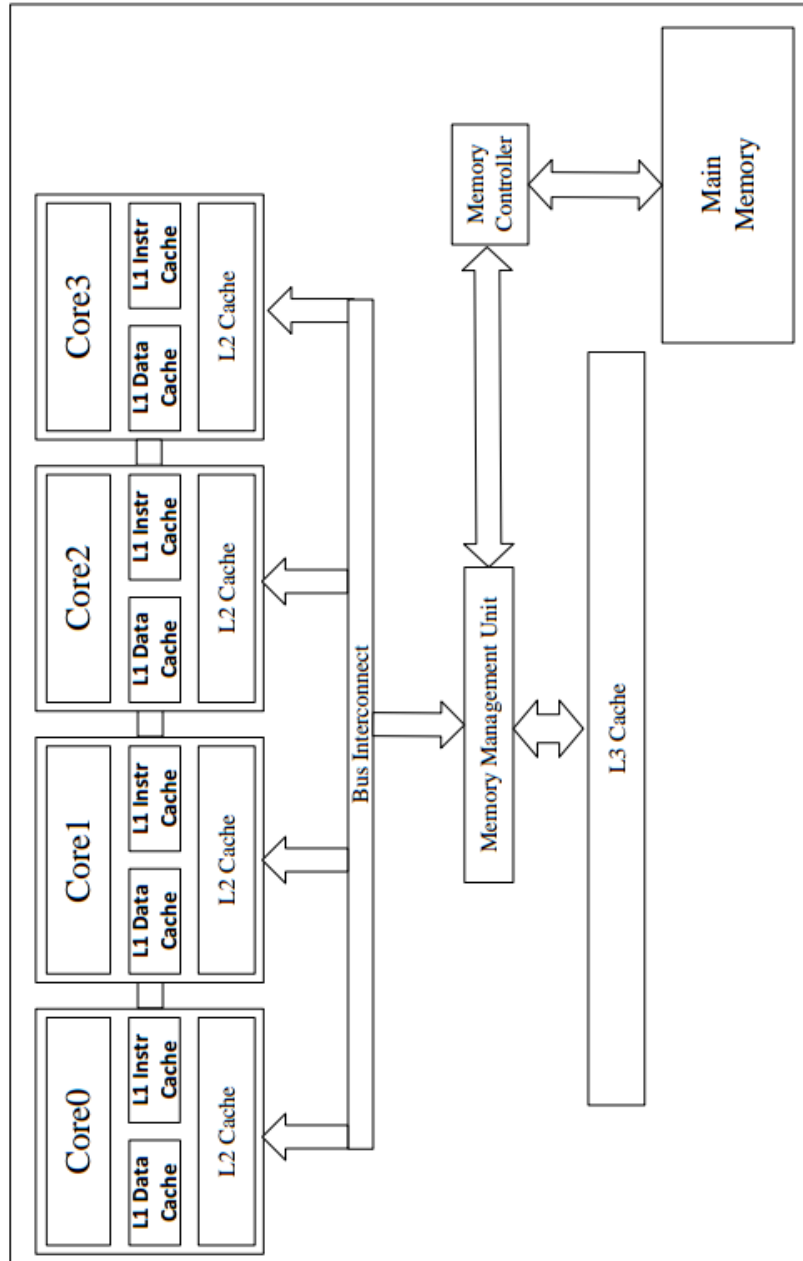


Figure 4.4: Quad-core Architecture

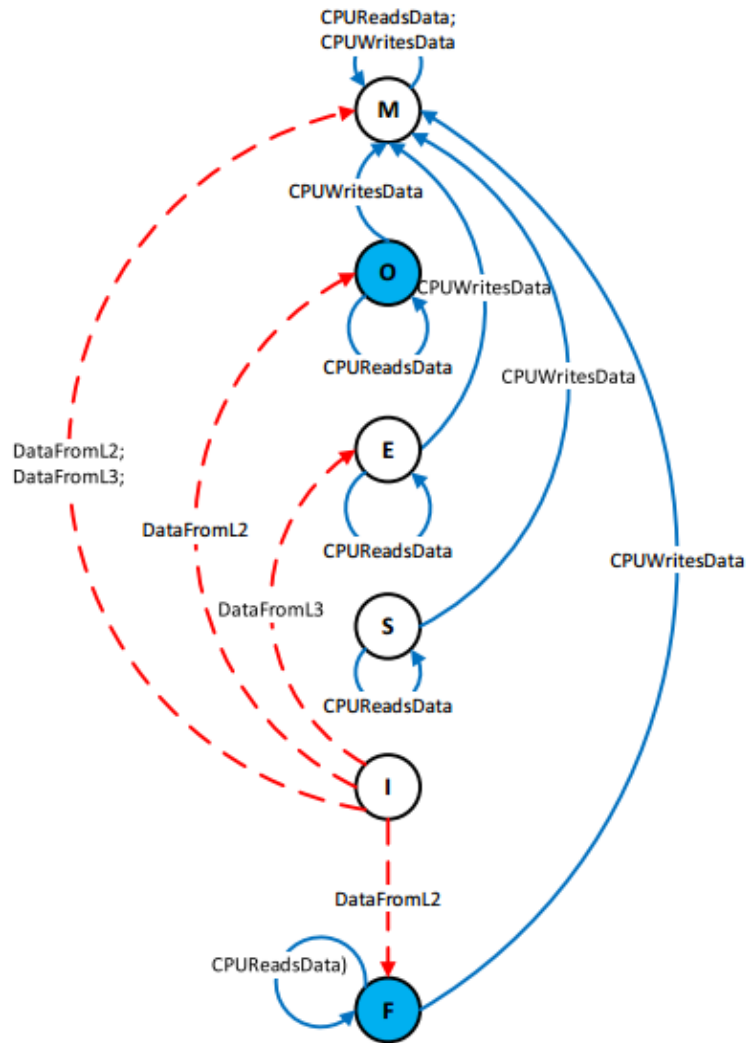


Figure 4.5: MOESIF cache access

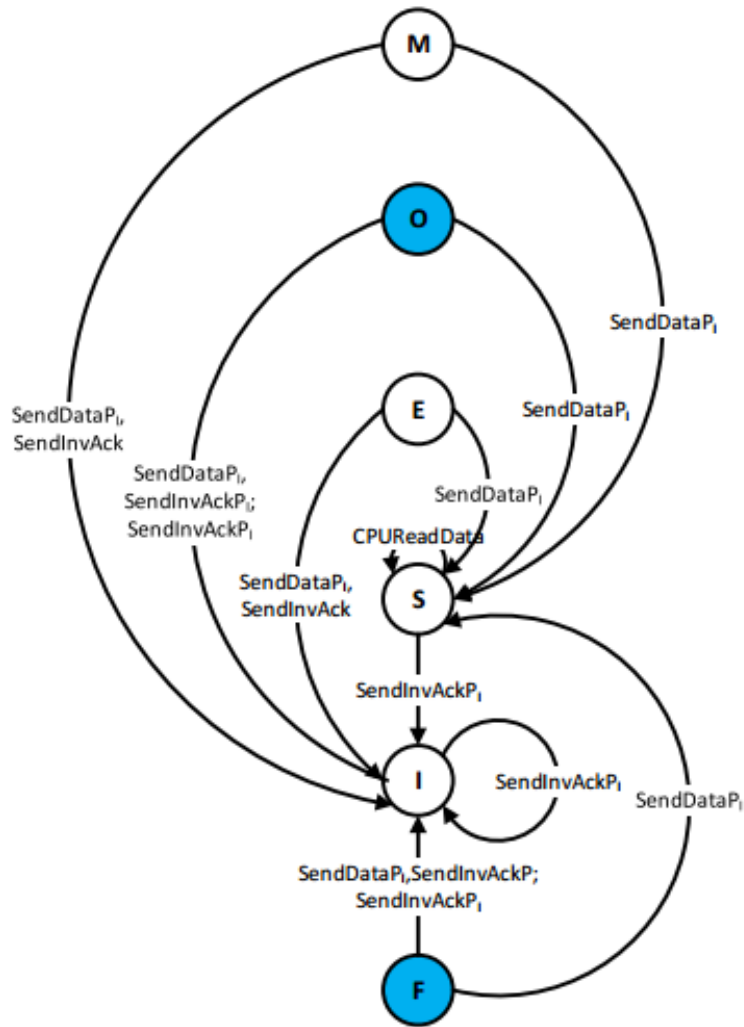


Figure 4.6: MOESIF cache snoop

are the most up to date copies and the next level memory contains stale data. The copy in O state only will act as forwarder which reduces the number of write backs and responders.

The replacement of owner cache line in MOESI and MOESIF protocols writes the dirty cache line back to the next level memory. The number of write backs due to replacement of cache line in O state is reduced in MOESIF protocol as the ownership of shared dirty data is transferred to the read requestor serviced by the owner. If forwarder gets replaced, the MOESIF protocol randomly picks one of the sharers as forwarder. If owner gets replaced, it writes the data to the next level memory and randomly picks one of the sharer as forwarder. This guarantees existence of a single responder in the system. Design of random number generator used for selecting forwarder is shown

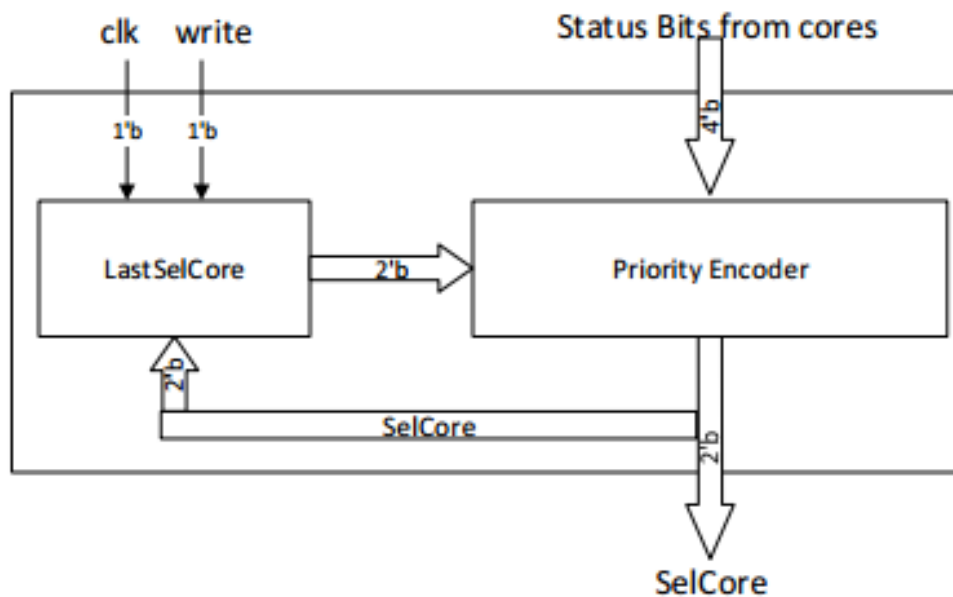


Figure 4.7: Design of random generator used for Quad-core Architecture

in Figure 4.7. The circuit is activated only when forwarder or owner gets replaced. Based on the previous selection and current status of data block in cores, the new forwarder is selected. The cache line state remains unchanged for read hits and write hits in M state. The cache line state updates from O, E, S and F state to M state for write hits. The cache read misses, write misses and write hits in O, S or F state broadcasts the request. When write request is broadcasted, the cache controllers containing shared data copy invalidates it and sends acknowledgement to the requestor. Upon receiving acknowledgements from $N - 1$ cores, requestor updates its cache line state to M. If any other core has the requested data in M state, that core transfers the data to the requestor and invalidates the copy for write snoop whereas it changes its state to S for read snoop. If any core has the requested data in cache line with E, O or F state, that core performs L2 to L2 transfer of data. For read snoop, the responder changes its state to S. If the core with O state forwards the data, the requestor changes its state to O. If the core forwarding the data is in F state or in E state, the requestor changes its state to F and the forwarder changes its state to S. For write snoop, the responder invalidates itself and sends invalidation acknowledgement to the requestor. The requestor changes its state to M after receiving $N - 1$ invalidation acknowledgements. When no core has the requested data, the requestor will receive data from L3 and sets the cache line state to E and M for read and write operations respectively.

The major modification implemented in MOESIF protocol in comparison with MOESI and MESIF protocols are read miss cases, where the data is received from L2 caches. In MOESIF protocol when read misses receive the

modified data, the ownership of the data is transferred to the requestor core. The requestor cache line state is changed to F when it receives a clean copy of data. In MOESI protocol, the requestor core will be in S state. Transfer of ownership to the latest requestor reduces the number of write backs. Read miss for a dirty copy of data of MOESIF differs from read miss for a dirty copy of data of MESIF protocol. The read miss request in MOESIF protocol is served by sharing a dirty copy of data which is L2 to L2 data transfer. The read miss for dirty copy of data in MESIF results in write back from L2 cache of the core which is having modified copy to L3 cache and then to L2 cache of requesting core.

To maintain cache in coherent state, MI and MESI requires 1 and 2 bits per cache line respectively, whereas MOESI, MESIF and MOESIF require 3 bits per cache line. Although MOESIF protocol has an overhead of additional bits per cache line to maintain cache coherency state over MI and MESI protocols, it reduces the network traffic by sharing of dirty data among cores. MOESIF protocol achieves energy saving and performance improvement over MESIF and MOESI protocol without additional hardware overhead and communication signals.

4.4 Energy and Time Model

The cache components used in the evaluation of energy consumption and access time based on SESC simulator [66] are shown in Table 3.1. The read access energy, write access energy and access time of L2 cache is calculated as

shown in equation 4.1, 4.2 and 4.3 respectively: Cache Read Access Energy:

$$E_{read_access} = E_{dyn_dec} + associativity * E_{dyn_x} + E_{dyn_op_drvr} \quad (4.1)$$

where E_{dyn_x} is the dynamic access energy per way.

Cache Write Access Energy:

$$E_{write_access} = 2 * (E_{read_access}) \quad (4.2)$$

Cache Access Time:

$$T_{access} = \max(T_{ds_dec}, T_{ts_dec}) + \max((T_{ds_wline} + T_{ds_bline} + T_{ds_sa}), (T_{ts_wline} + T_{ts_bline} + T_{ts_sa} + T_{ts_cmp}), (T_{ts_vi})) + T_{ds_opDrvr} \quad (4.3)$$

The energy and time model for cache read and write operations in an N -core system is shown in Table 4.2. Where E_{Inv} , $E_{Tx-L2toL2}$, $E_{Tx-L2toL3}$ and $E_{Tx-L3toL2}$ represent the energy consumed for cache line invalidation and sending acknowledgements, L2 to L2 transfer, L2 to L3 transfer and L3 to L2 transfer respectively. T_B , T_{AckAll} , T_{L2Read} , $T_{L2Write}$, T_{L3Read} , $T_{L3Write}$, $T_{Tx-L2toL2}$, $T_{Tx-L2toL3}$ and $T_{Tx-L3toL2}$ represent the time for broadcasting address, receiving all the acknowledgements, reading L2, writing L2, reading L3, writing L3, L2 to L2 transfer, L2 to L3 transfer and L3 to L2 transfer respectively. The transfer time among L2 and L3 caches is calculated as

Table 4.2: Energy and time modeling for cache operations

Cache Operation		Energy	Time
Read Hit	-	E_{read_access}	T_{access}
Read Miss	When no L2 copies exist	$N * E_{read_access} + E_{Tx-L3toL2}$	$T_B + T_{x32} + T_{access}$
	When k-L2 copies exist ($1 \leq k \leq N - 1$)	$N * E_{read_access} + k * E_{Tx-L2toL2}$	$T_B + k * T_{x22} + T_{access}$
	When a single modified copy exist in a L2	$N * E_{read_access} + E_{Tx-L2toL3} + E_{Tx-L3toL2}$	$T_B + T_{x23} + T_{x32} + T_{access}$
Write Hit	When no other L2 copies exist	$E_{access} + E_{PWrite}$	T_{PWrite}
	When other L2 copies exist	$N * E_{access} + E_{Inv} + E_{PWrite}$	$T_B + T_{AckAll} + T_{PWrite}$
Write Miss	When no L2 copies exist	$N * E_{access} + E_{Tx-L3toL2} + E_{Inv} + E_{PWrite}$	$T_B + \max(T_{x21}, T_{AckAll}) + T_{PWrite}$
	When k-L2 copies exist ($1 \leq k \leq N - 1$)	$N * E_{access} + k * E_{Tx-L2toL2} + E_{Inv} + E_{PWrite}$	$T_B + \max(k * T_{x11}, T_{AckAll}) + T_{PWrite}$
	When a single modified copy exist in a L2	$N * E_{access} + E_{Tx-L2toL3} + E_{Tx-L3toL2} + E_{Inv} + E_{PWrite}$	$T_B + \max(T_{x12} + T_{x21}, T_{AckAll}) + T_{PWrite}$

shown in equation 4.4

$$\begin{aligned}
 T_{x22} &= T_{L2Read} + T_{Tx-L2toL2} + T_{L2Write} \\
 T_{x32} &= T_{L3Read} + T_{Tx-L3toL2} + T_{L2Write} \\
 T_{x23} &= T_{L2Read} + T_{Tx-L2toL3} + T_{L3Write}
 \end{aligned}
 \tag{4.4}$$

When replacement results in write back of dirty data, the additional energy and access time is added up in the energy and time calculations respectively.

4.5 Experimental Evaluation

4.5.1 Experimental Setup

The time and energy estimation is done by using SESC simulator. The simulator estimates the energy consumption, access time, cache miss rate, L2 to L2 transfers, write backs, L3 to L2 transfers, invalidations and invalidation acknowledgements. The experimentation uses cache size, cache line size and associativity as 4KB to 32KB, 8B to 32B and direct mapped to 16-way set-associative respectively.

4.5.2 Experimental Analysis of Protocols

Shared copy of data does not exist in MI protocol. The cache read and write misses broadcast the request through the bus. If other core has the requested data, it performs write back operation. Upon receiving write back acknowledgement from L3, it invalidates its copy. For all cache misses, the requestor receives the data from L3.

MESI protocol satisfies read and write misses by transferring data to the cache either from the next level memory or from other cores. If any core has the requested data in exclusive (E) or shared S state, the requestor is serviced by the core upon broadcast. If a core contains data with modified (M) state then it updates the next level memory with that data. All the cache lines containing the requested data will be in S state for read miss and I state for write miss. Presence of multiple shared copies reduces the number of coherence misses in MESI over MI protocol. However, for every state change from M, the data needs to be written to the next level. This result in frequent data writes between core and next level when read and write alternate.

The M state in MOESI protocol allows transfer of data to a read requestor without writing back to the next level memory. The dirty copy sharing between different cores using Owned (O) state helps to reduce the frequency of write-backs. Write back in MOESI happens only when the cache line in M or O state is replaced. However, during a cache miss, all the sharers of requested data responds and the requestor is serviced by redundant responses. These redundant responses increase the traffic across the network as data transfer takes more bandwidth than transfer of signals through the network. The issue of redundant responses from all the sharers in MESI and MOESI is addressed by the Forward (F) state. MESIF protocol ensures that only the cache line in F state responds to the read / write request of other cores. Among the sharers, the last recipient of data is assigned with F state. During read miss, the cache line in F state transfers the data and updates its state to S. Although redundant responses are eliminated using F state, sharing of dirty data is not allowed in MESIF. Write back happens for all the state

changes from M state.

MOESIF protocol achieves energy efficiency and high performance by optimizing data transfers between caches and with next level memory. This is achieved by reducing the number of write backs to the next level memory and making sure only one responder sends data to the requestor. Combining O state and F state with MESI states help in achieving this. This results in improved hit rate and reduced traffic which in turn improves response time of the system.

4.5.3 Experimental Evaluation

4.5.3.1 Hit rate and Data transfers

Figures 4.8, 4.9, 4.10 and 4.11 show hit rate, per access writebacks, per access data received from L3 and per access data received from other L2 respectively for MI, MESI, MOESI, MESIF and MOESIF protocols for varying cache sizes. Number of Conflict misses reduces with increase in cache size. As shown in figure 4.8, for all cache coherency protocols, hit rate increases with increasing cache size. In MI, shared copy of data does not exist where as MESI and MESIF shares clean copy of data. The MOESIF and MOESI shares dirty copy along with clean copy of data. Hit rate of MI is the least among all the protocols.

Figure 4.9 shows per access write backs. The number of write backs in MI is the highest and it reduced by 53.87% in MESI and MESIF protocol. MOESI and MOESIF protocols reduce the number of write backs further and results in least number of write backs.

Figure 4.10 and 4.11 show per access data transfer rate between L2 and

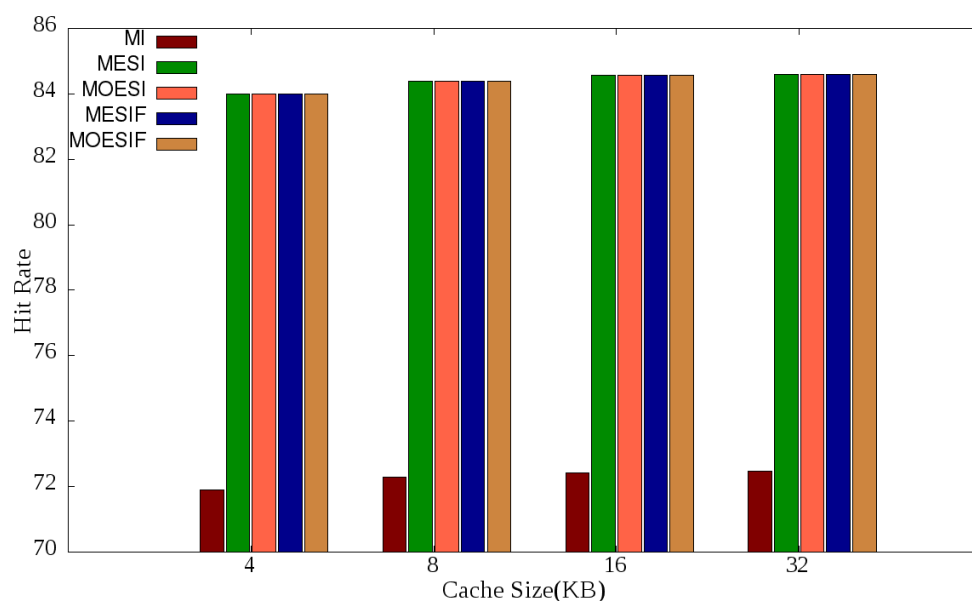


Figure 4.8: Per access hit rate for varying cache sizes with 32B line size and associativity as 4 way

L3 caches and among L2 caches of different cores respectively. In MI, data transferred to L2 cache is only from L3 cache and data is not transferred among L2 caches of different cores. In MI, the highest number of data transfers occur between L3 cache and L2 cache and no data transfer occurs among L2 caches of different cores .

In MESI and MESIF protocols, read and write miss requests for modified data is satisfied by performing write back operation and then transferring the requested data from L3 cache to the requestor. Data transfer rate between L2 and L3 caches is higher as compared to data transfer rate among L2 caches of different cores for MESI and MESIF protocols.

MOESI and MOESIF protocols share a dirty copy of data without writing

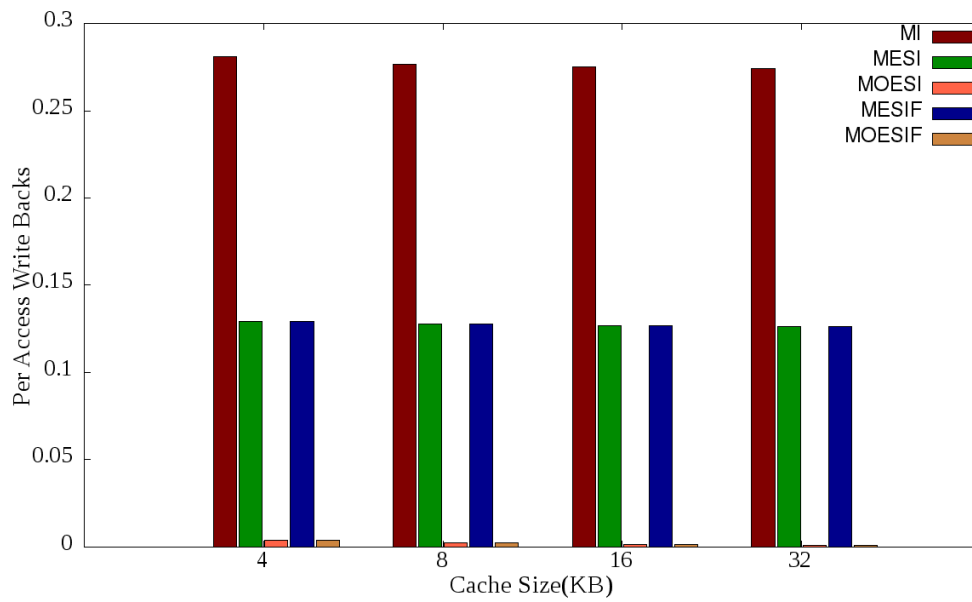


Figure 4.9: Per access write backs for varying cache sizes with 32B line size and associativity as 4 way

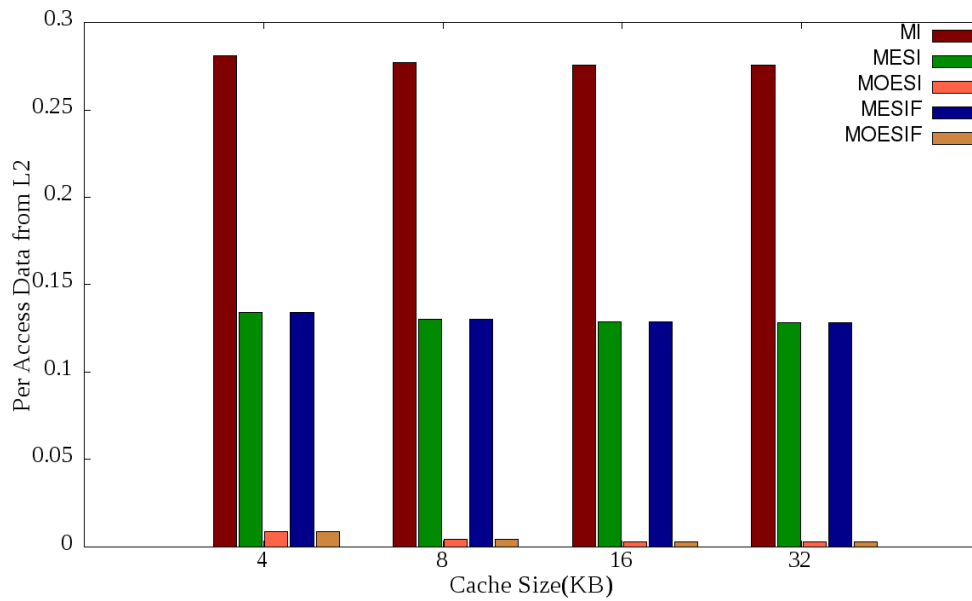


Figure 4.10: Per access data from L2 for varying cache sizes with 32B line size and associativity as 4 way

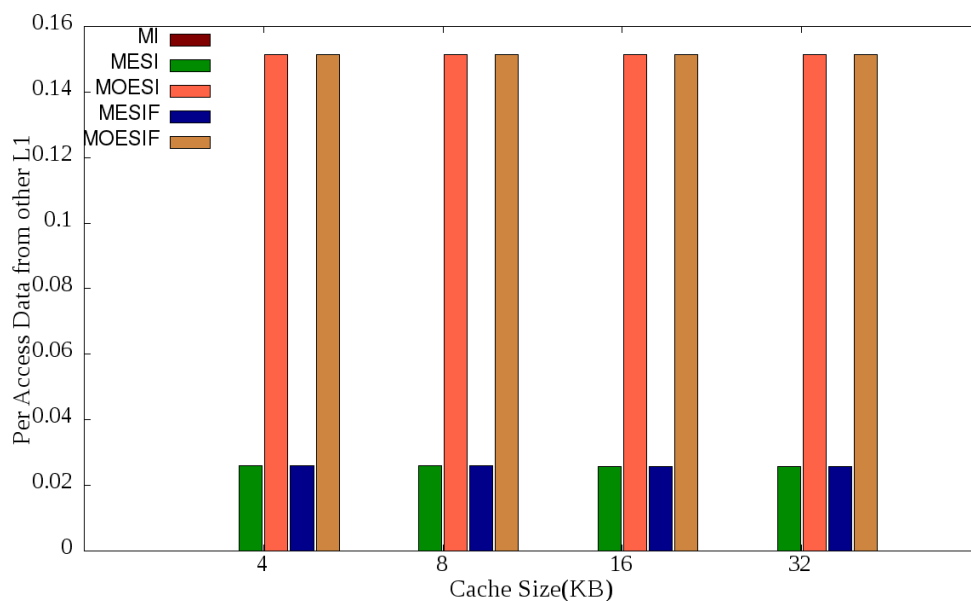


Figure 4.11: Per access data from other L1 for varying cache sizes with 32B line size and associativity as 4 way

back the data to L3 cache. Data transfer rate among L2 caches of different cores is higher as compared to data transfer rate between L2 and L3 caches for MOESI and MOESIF protocol. Similar trends are observed while varying cache line size and associativity as well.

4.5.3.2 Energy Consumption

Figures 4.12, 4.13, and 4.14 show energy consumption with varying cache size, cache line size and number of cores respectively. From the results, it is observed that energy consumption of MI is the highest due to high miss rate and highest number of write backs. Irrespective of the configuration parameters, MOESIF protocol outperforms other protocols in energy consumption.

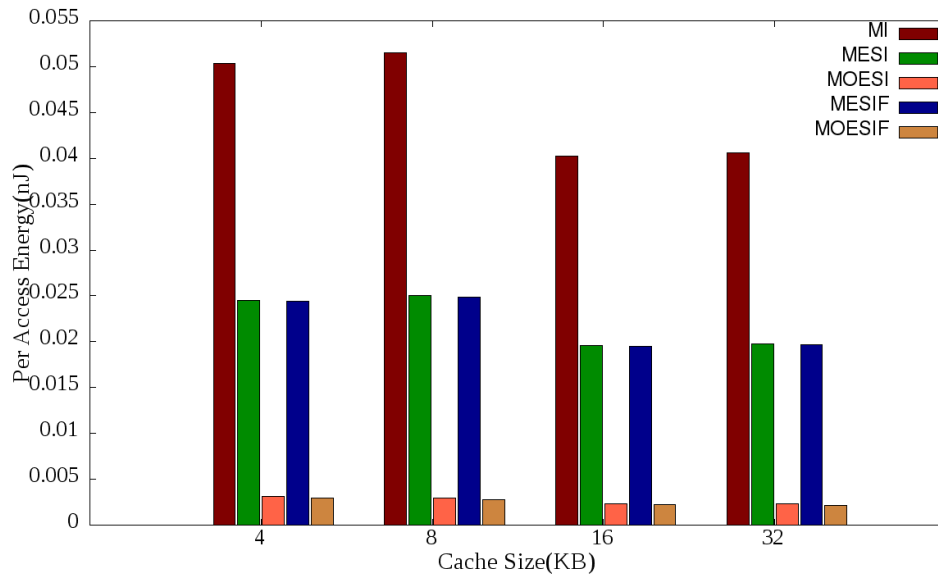


Figure 4.12: Per access energy for varying cache sizes with 32B line size and associativity as 4 way

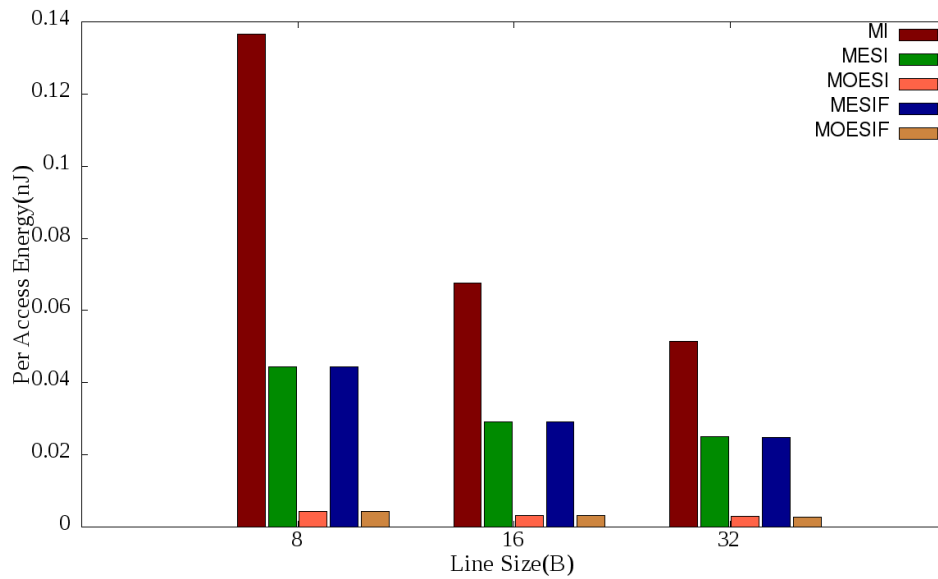


Figure 4.13: Per access energy for varying cache line sizes with 8KB cache size and associativity as 4 way

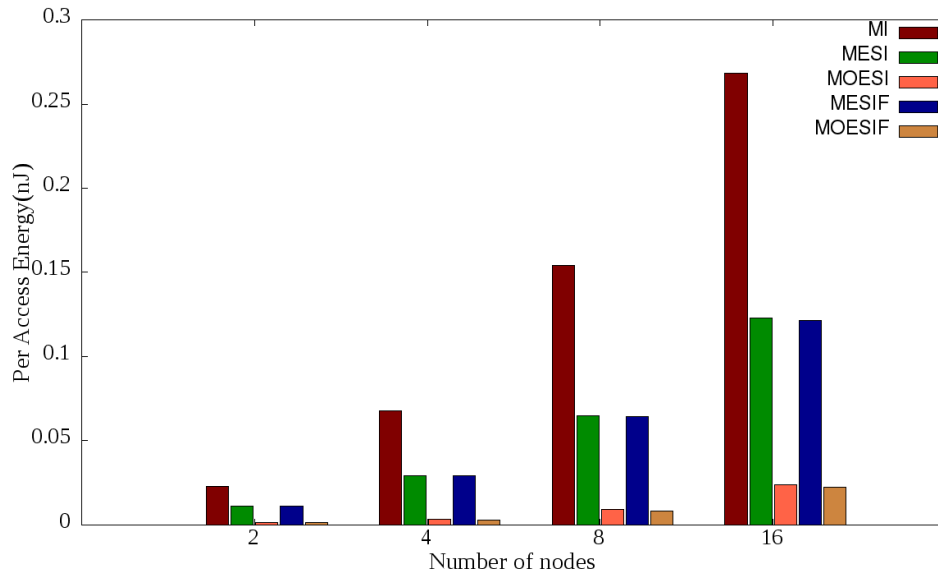


Figure 4.14: Per access energy for varying number of cores with 8KB cache, 16B line size and associativity as 4 way

On an average, for varying cache sizes MESI, MOESI, MESIF and MOESIF reduces 51.41%, 94.20%, 51.66% and 94.49% of the total energy over MI. The energy savings of MOESIF protocol over MESI, MOESI and MESIF protocols is 88.58%, 4.33% and 88.52% respectively.

It is evident from figure 4.13 that with increase in cache line size, the total energy consumption reduces due to reduction in compulsory misses.

MESI and MESIF have comparable energy when number of cores are less (say 2). For a larger number of cores, MESIF consumes less energy than MESI with a single responder for forwarded request. It is also evident that energy consumption increases exponentially with increase in number of cores. However, the increase is small for MOESI and MOESIF protocols over other protocols because of dirty sharing and reduced number of write backs.

4.5.3.3 Response Time

Figures 4.15, 4.16, and 4.17 show per access time with varying cache size, cache line size and number of cores respectively. It is observed that per access time of cache increases with increase in cache size and number of cores. Increase in cache size results in increasing cache cycle time and number of cores which results in increasing coherency misses. Increase in cache line size results in reducing access time because of the increase in hit rate.

For varying cache size, per access time of MESI, MOESI, MESIF and

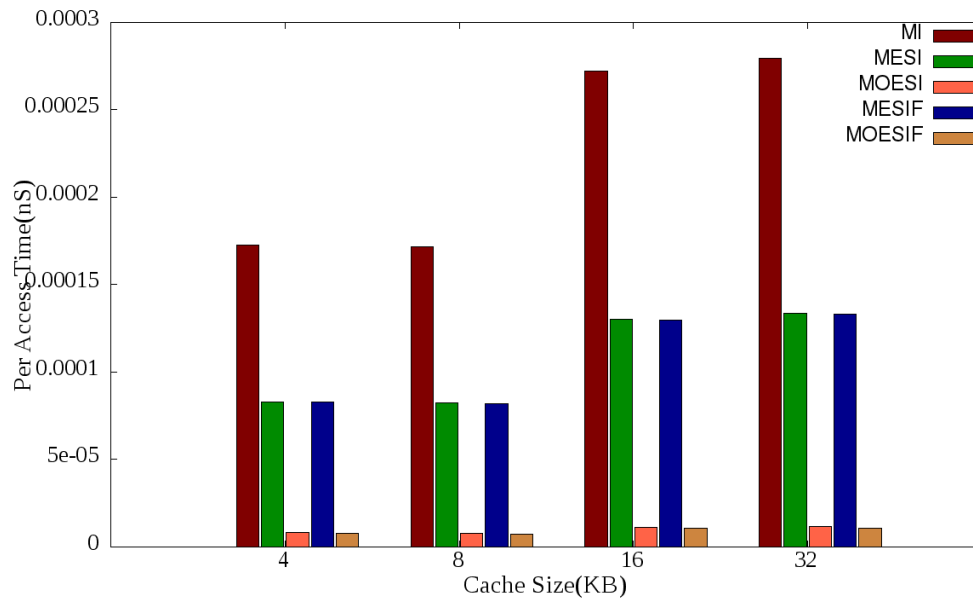


Figure 4.15: Per access time for varying cache sizes with 32B line size and associativity as 4 way

MOESIF protocols reduce by 52.04%, 95.59%, 52.31% and 95.86% respectively over MI. The per access time saving of MOESIF protocol over MESI, MOESI and MESIF protocol is 91.37%, 6.17% and 91.32% respectively.

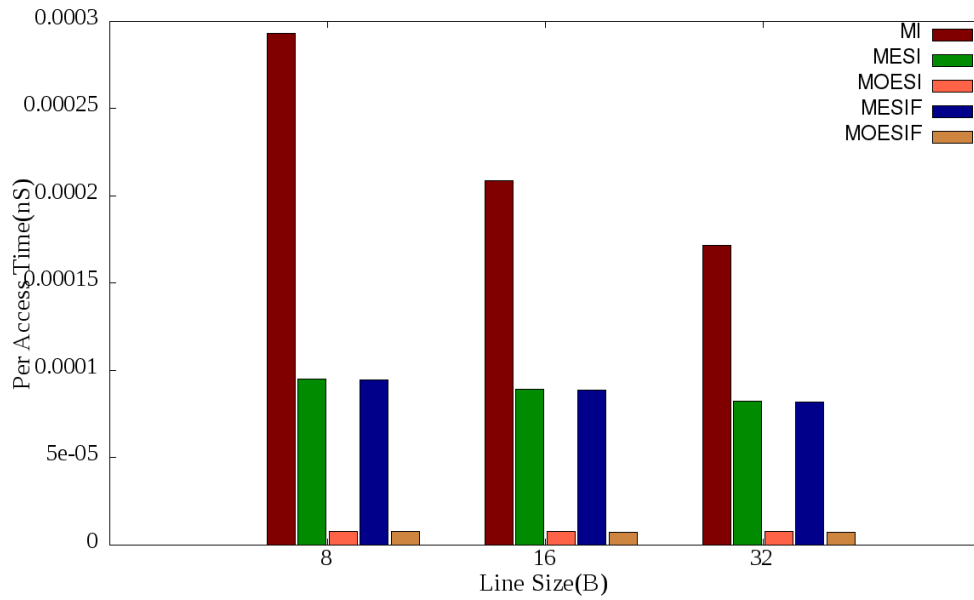


Figure 4.16: Per access time for varying cache line sizes with 8KB cache size and associativity as 4 way

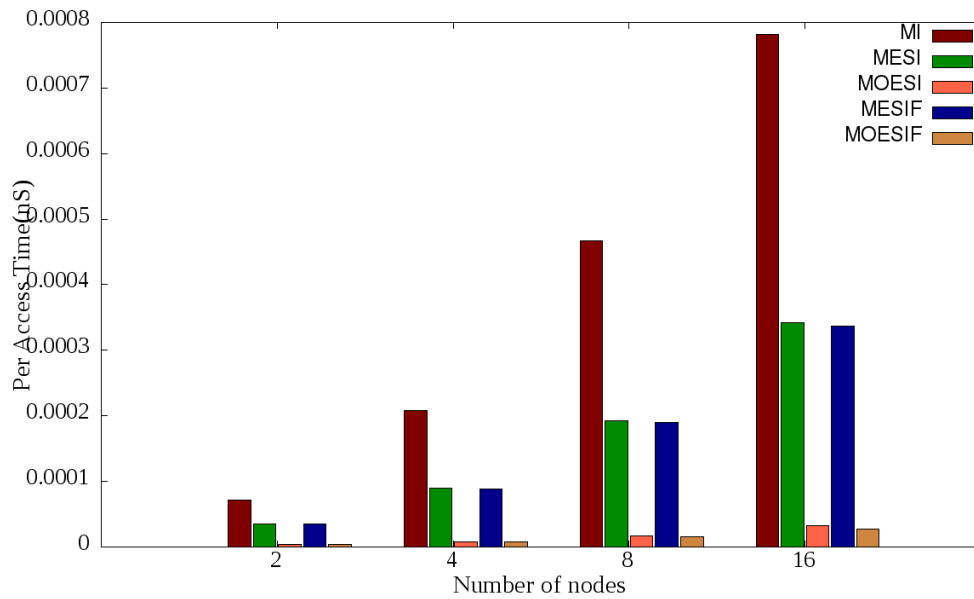


Figure 4.17: Per access time for varying number of cores with 8KB cache, 16B line size and associativity as 4 way

4.6 Conclusion

Cache coherence protocols achieve data consistency and coherency at the cost of performance degradation with respect to time and energy. The additional overhead can be minimized by optimizing the usage of interconnection bandwidth. This chapter discussed MOESIF protocol which improves the off-chip bandwidth by reducing write backs to next level memory and the on-chip bandwidth by reducing the number of responders to a cache miss when multiple copies of data exists in private L2 caches of various cores. For varying cache sizes, energy consumption in MESI, MOESI, MESIF and MOESIF protocols is reduced by 51.41%, 94.20%, 51.66% and 94.49% respectively over MI protocol. The energy savings of MOESIF protocol over MESI, MOESI and MESIF protocol is 88.58%, 4.33% and 88.52% respectively. For varying cache sizes, per access time of MESI, MOESI, MESIF and MOESIF protocols is reduced by 52.04%, 95.59%, 52.31% and 95.86% respectively over MI. The per access time saving of MOESIF protocol over MESI, MOESI and MESIF protocol is 91.37%, 6.17% and 91.32% respectively.

Chapter 5

DTLB: Deterministic TLB for Real-time System

5.1 Introduction

TLB plays a crucial role in speeding up the virtual to physical address translation. Each memory access results in accessing TLB. TLB is in critical path of memory access. TLB misses lead to accessing main memory multiple times which results in performance degradation both in terms of time and energy. Performance of the system can degrade upto 50% due to TLB's miss penalties [68]. TLB misses result in additional energy consumption because of page table walk through. TLB contributes upto 17% of the total on-chip energy due to its high access frequency [69].

In hard real-time systems, TLB misses affect the deadlines as WCET of the job increases with TLB misses. Deterministic worst case upper bound must be guaranteed for hard real-time tasks in order to ensure a tighter upper bound [70]. This helps in executing more tasks without deadline misses. Predictability in hard real-time system can be ensured by making the entire process of accessing memory subsystem deterministic, so that it can be used without deadline misses. The unpredictable nature of TLB is one of the major

factors which makes the memory subsystem non-deterministic, along with unpredictability of cache memories and main memory. The unpredictability in case of conventional TLB is because of TLB flushing on preemption. This can be addressed by reservation based TLBs like ASID-TLB[50]. In ASID-TLB, the number of lockable entries for a specific process is calculated statically. These entries are not available for replacement for other tasks. This results in reducing the size of TLB per process. This mechanism suffers with additional TLB misses because of reserved entries. Flushing/global replacement results in increase in TLB misses on preemption.

To have a tighter upper bound on the WCET of real-time task, this chapter presents a TLB architecture - Deterministic Translation Lookaside Buffer (DTLB). DTLB offers deterministic miss rate which is the least possible miss rate and is equal to the number of misses when that task is running as the only task in system. DTLB offers the least number of misses ever possible in a system when all tasks are available in main memory [71] [72]. It is achieved by storing all the TLB entries in process control block (PCB) during task preemption. TLB entries are loaded back from the PCB into TLB when task resumes back its execution. DTLB offers up to 24.77% reduction in TLB miss rate as compared to conventional TLB which flushes TLB entries during preemption.

5.2 DTLB Architecture

DTLB is designed to obtain deterministic WCET of real-time task. It improves memory performance by reducing the TLB misses for low priority

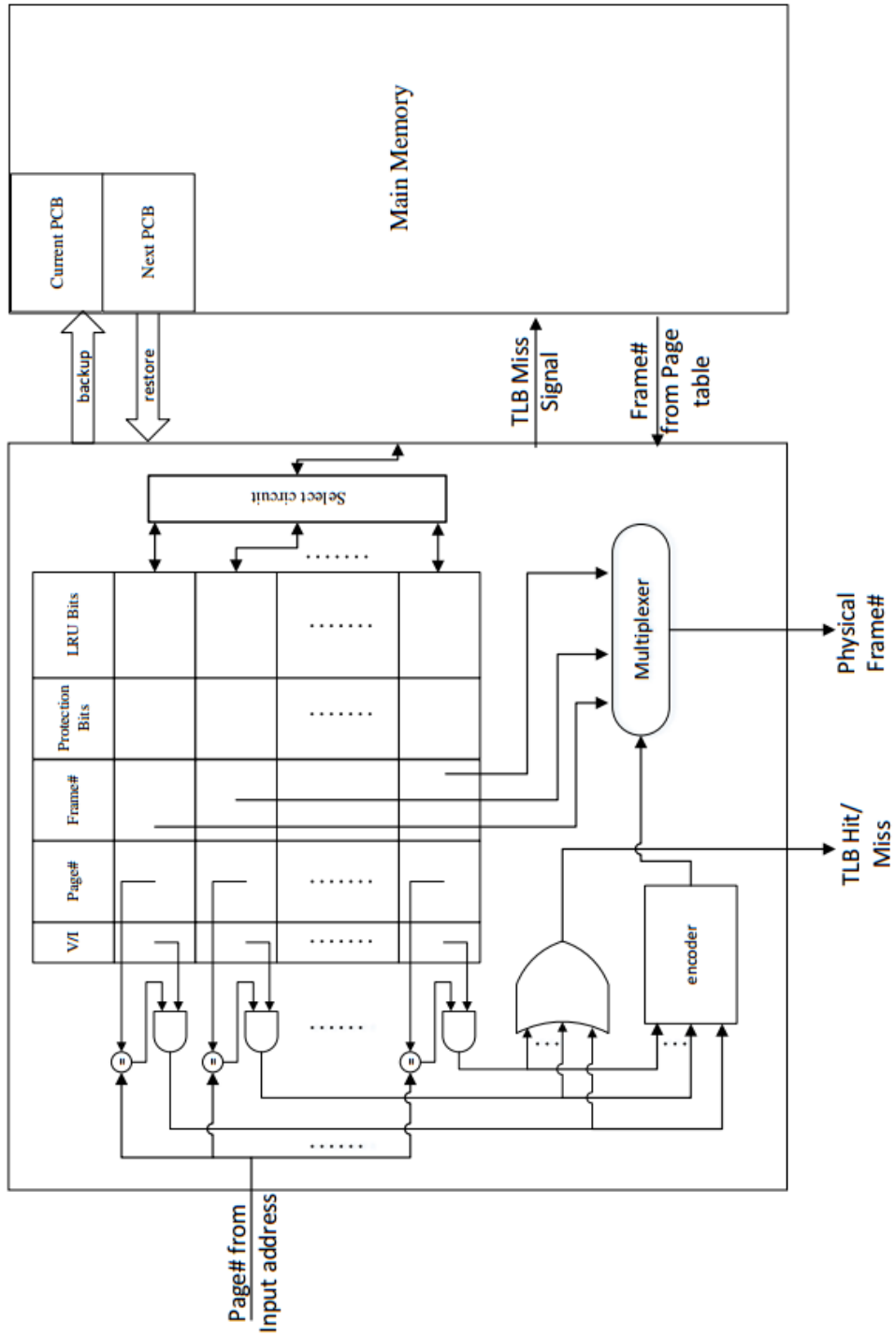


Figure 5.1: Deterministic TLB Architecture

real-time tasks. Detailed architecture of DTLB is shown in figure 5.1. Each entry in DTLB contains valid/invalid bit, page number, frame number of that page in memory, protection/access bits and LRU bits. In DTLB model, PCB contains space required for backing up all TLB entries. On preemption, a special instruction(FC000000) is executed as a part of preemption routine. Multiplexer is used for selecting TLB entry to be transferred through bus. This instruction initiates the transfer of all the TLB entries to PCB of that process. The TLB is flushed/filled with next process's PCB contents based on whether the process is new/already executed. The next dispatched job can use the entire TLB without being affected by the previous job. This repopulates the TLB with the same entries that were present at the time of preemption and the job resumes its execution from that point without additional TLB misses. The repopulation of TLB ensures no additional TLB misses across preemptions. The execution times of highest priority jobs are not affected by DTLB as it is dispatched without preemptions. Only the lower priority job executions suffer a constant time overhead at the time of preemption which helps in offering tighter WCET when number of preemptions are known.

DTLB guarantees that TLB entries of running job is isolated from other ready to run jobs in the system. For deterministic real-time performance, this work assumes that the main memory is big enough to hold the required pages of all ready to run jobs[71] [72]. The local replacement policy ensures a predictable upper bound on the number of TLB misses.

Experimental results prove that the DTLB offers the least number of TLB misses. It also offers deterministic performance. However there exist overhead associated with this model, i.e., the time taken to transfer TLB entries to

PCB and back. The number of transfers is proportional to the number of preemptions. The preemption depends on the number of higher priority job arrivals when lower priority job is executing. The theoretical upper bound of the same can be found using the task model analysis. If the total number of slots in TLB is N , all these entries need to be copied to its PCB. The transfer of N TLB entries to the PCB and vice versa takes approximately a cache block transfer time. The total overhead on the system depends on the number of preemptions that are being made and is equal to the product of the number of preemptions and the constant transfer time. The time consumed for this operation is deterministic. Hence the DTLB operation becomes deterministic.

5.3 Energy and Time Modeling

5.3.1 Energy Modeling of TLB

The components used in the evaluation of energy and time modeling of TLB are shown in Table 5.1.

TLB hit energy consists of energy required for page number comparisons, valid/invalid bit access, protection verification, replacement circuit and output driver and is given by:

$$E_{TLB_Hit} = E_{comp} + E_{v/i} + E_{protection} + E_{rep_ckt} + E_{access} + E_{op_drv} \quad (5.1)$$

TLB miss energy consists of additional energy required for page number comparisons, valid/invalid bit access, page table walk energy and TLB update

Table 5.1: TLB Components for Energy and Time Modeling

TLB Component	Energy Components	Time Components
Comparator	E_{comp}	T_{comp}
Valid-Invalid	$E_{v/i}$	$T_{v/i}$
Protection Bit	$E_{protection}$	$T_{protection}$
Replacement Circuit	E_{rep_ckt}	T_{rep_ckt}
Access	E_{access}	T_{access}
Output Driver	E_{op_drv}	T_{op_drv}
Main Memory Access	E_{mm_access}	T_{mm_access}
TLB Update	E_{tlb_update}	T_{tlb_update}
Transfer	$E_{Transfer}$	$T_{Transfer}$

energy apart TLB hit energy. TLB miss energy is:

$$E_{TLB_Miss} = E_{comp} + E_{v/i} + E_{mm_access} + E_{tlb_update} + E_{TLB_Hit} \quad (5.2)$$

The total energy of conventional TLB, ASID-TLB and DTLB is calculated as follows:

Conventional TLB:

$$E_{TLB_CONV} = E_{TLB_Hit} * N_{Hit_Conv} + E_{TLB_Miss} * N_{Miss_Conv} \quad (5.3)$$

Reservation TLB:

$$E_{TLB_ASID} = E_{TLB_Hit} * N_{Hit_ASID} + E_{TLB_Miss} * N_{Miss_ASID} \quad (5.4)$$

DTLB TLB:

$$E_{TLB_DTLB} = E_{TLB_Hit} * N_{Hit_DTLB} + E_{TLB_Miss} * N_{Miss_DTLB} + N_{Prem} * E_{Transfer} * 2 \quad (5.5)$$

where N_{Hit_Conv} , N_{Hit_ASID} , N_{Hit_DTLB} is the number of TLB hits and N_{Miss_Conv} , N_{Miss_ASID} , N_{Miss_DTLB} is the number of TLB misses of conventional, ASID and DTLB model respectively. N_{Prem} is the number of preemptions. Hit rate of DTLB is better than the hit rate of conventional and ASID model. DTLB requires additional energy for transferring TLB entries to PCB and back.

5.3.2 Time Modeling of TLB

The access time representations of the TLB components are shown in Table 5.1.

TLB hit time is given by :

$$T_{TLB_Hit} = T_{page_no_comp} + T_{op_drv} \quad (5.6)$$

where $T_{page_no_comp}$ is the time required for comparing all the page numbers in parallel and finding the matching page number from it. T_{op_drv} is the time required to output the frame number from the selected TLB entry.

TLB miss time is given by :

$$T_{TLB_Miss} = T_{mm_access} + T_{TLB_Hit} \quad (5.7)$$

The total time of conventional, ASID-TLB and DTLB is computed as follows:

Conventional TLB:

$$T_{TLB_CONV} = T_{TLB_Hit} * N_{Hit_Conv} + T_{TLB_Miss} * N_{Miss_Conv} \quad (5.8)$$

Reservation TLB:

$$T_{TLB_ASID} = T_{TLB_Hit} * N_{Hit_ASID} + T_{TLB_Miss} * N_{Miss_ASID} \quad (5.9)$$

DTLB TLB:

$$T_{TLB_DTLB} = T_{TLB_Hit} * N_{Hit_DTLB} + T_{TLB_Miss} * N_{Miss_DTLB} + N_{Prem} * T_{Transfer} * 2 \quad (5.10)$$

DTLB requires additional time for transferring TLB entries to PCB and back.

5.4 Experimental Setup And Evaluation

5.4.1 Experimental Setup

The evaluation of DTLB in comparison with seminal works were carried out with the help of SESC framework. Time and energy estimation from SESC is used for dynamic energy and time analysis [67]. Each trace file entry consists of the virtual memory address and the operation i.e., read or write. For each task in the task set, a different trace file from a different program is used. Execution schedule of the task set is specified as per the format in TABLE 5.2,

where IO/Preempt/Complete is a flag to specify whether the job goes for I/O, preemption or completion at the end of the execution period. The memory

Table 5.2: Task Set Execution Schedule Format

Start time of job	End time of job	Job ID	Stack memory usage	IO / Preempt / Complete
-------------------	-----------------	--------	--------------------	-------------------------

components in the simulator can be configured as needed. The TLB variants used for evaluation are DTLB, ASID-TLB and conventional TLB. At the beginning all TLB entries, the cache memory and the stack memory is free. The simulator reads the input schedule line by line. When a new job is to be executed, the required stack space is allocated to it. The simulator selects the appropriate trace file based on the job ID. Each time unit of the execution is mapped to 'x' memory accesses in the setup. The number of traces in the trace file is proportional to the execution time of the task. The traces offer virtual address along with read/write operation. The page number extracted from the virtual address is given to the TLB. A TLB hit offers the frame number corresponding to the page number, which is in concatenation with the offset, is given to the cache memory for instruction/data access. The mapping function with the help of the page table finds the frame number corresponding to the page number in case of TLB miss. This work assumes architecture with segmentation with two level hierarchical paging. The standard page size i.e., 4KB is used as the default page size. Counters are maintained to keep track of the number of TLB hits and cache hits. Irrespective of the TLB variants in use, parallel search in all TLB entries is used for finding TLB hit. At each preemption point, the DTLB is flushed after backing up the entries in PCB of the running task. When a preempted task resumes its execution in

CPU, the TLB entries are reinitialized with the backed up PCB content. As these activities are happening with preemption, it never impact the critical path delay. The number of such preemption points in real-time systems is very less as only the higher priority jobs can preempt the executing job. The simulator stops its execution when there exist no more jobs in the schedule.

5.4.2 Experimental Evaluation

This work compares the performance of DTLB with Conventional and ASID-TLBs. The parameters used for comparison are TLB miss rate, dynamic energy consumption and access time.

5.4.2.1 TLB Miss Rate

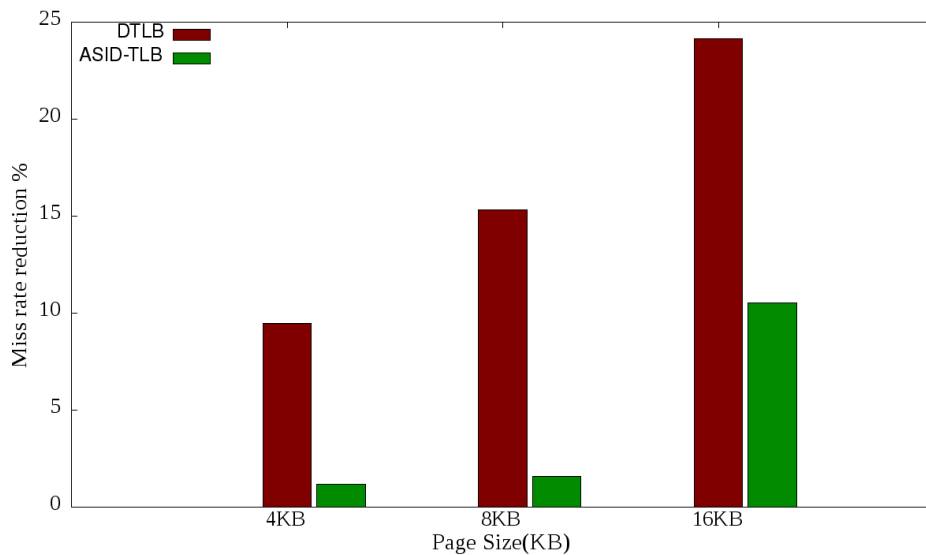


Figure 5.2: Miss rate performance of DTLB and ASID-TLB with respect to conventional TLB for varying page size with 16 preemptions and 32 TLB entries

Figure 5.2 shows TLB miss rate performance of DTLB and ASID-TLB over conventional TLB. FFT benchmark suite with 16 preemptions and 32 TLB entries for varying page sizes is used for the performance analysis. As reachability of the TLB increases with increase in page size, irrespective of the TLB model in use, the miss rate decreases. The number of misses decreases on an average by 34.67%, 32.03% and 28.63% respectively for DTLB, ASID-TLB and conventional model with every doubling of page size. DTLB and ASID-TLB reduces the miss rate on an average by 16.32% and 4.43% respectively over conventional TLB.

Figure 5.3 shows the miss rate reduction of DTLB and ASID-TLB over

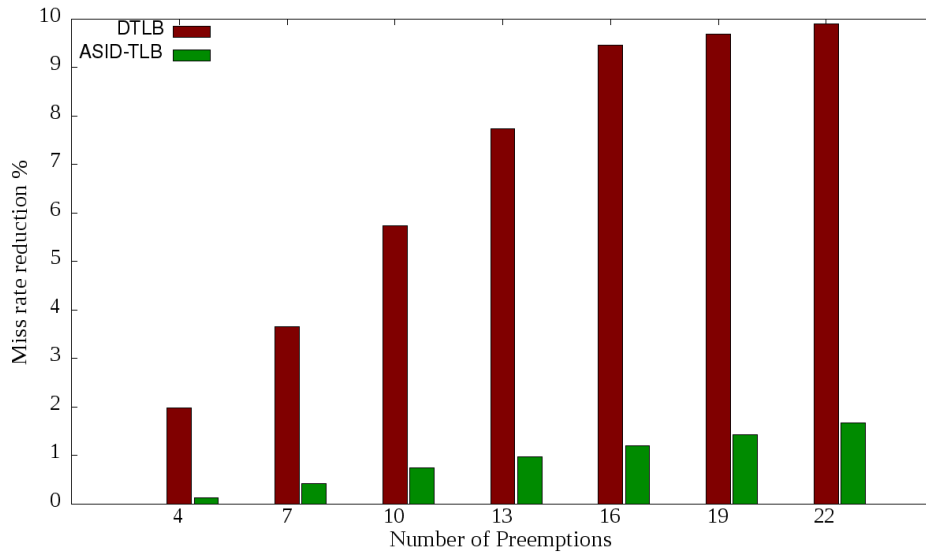


Figure 5.3: Miss rate performance of DTLB and ASID-TLB with respect to conventional TLB for varying preemptions with 4KB page size and 32 TLB entries

conventional TLB for varying number of preemptions. FFT benchmark suite with 4KB page size and 32 TLB entries is used for the performance analysis.

With increase in preemptions, the miss rate of DTLB remains constant whereas the miss rate in conventional and ASID-TLB increases. The increase in miss rate is exponential in conventional TLB because of invalidation at each preemption point whereas the increase is marginal in ASID-TLB. The DTLB provides deterministic miss rate which is equal to the miss rate when it runs without preemption. Reduction in DTLB miss rate over ASID-TLB varies from 1.86% to 8.37% when number of preemptions varies from 4 to 22. Reduction in DTLB miss rate over conventional varies from 1.98% to 9.90% for number of preemptions varying from 4 to 22. Results obtained by varying the number of tasks show similar trend as shown in Figure 5.3 as in real-time systems, the increase in number of tasks result in increasing number of preemptions.

As reachability of the TLB increase with increase in TLB entries, irrespective of the TLB models in use, the miss rate decreases. Figure 5.4 shows the reduction in miss rate of DTLB and ASID-TLB over conventional TLB for various TLB entries. FFT benchmark suite with 4KB page size and 16 preemptions is used for the performance analysis. Reduction in TLB misses is due to reduction in capacity misses and conflict misses. For lesser number of TLB entries, the ASID-TLB performs poor than conventional model because of the capacity reduction. The number of misses decreases on an average by 17.81%, 15.85% and 9.14% respectively for DTLB, ASID-TLB and conventional model with every doubling of TLB entries.

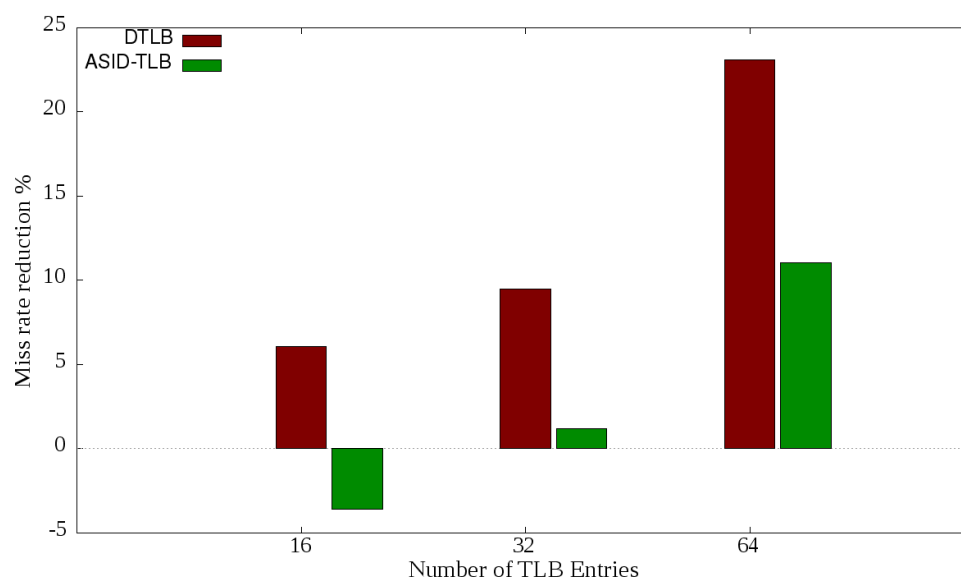


Figure 5.4: Miss rate performance of DTLB and ASID-TLB with respect to conventional TLB for varying TLB entries with 16 preemptions and 4KB page size

5.4.2.2 Dynamic Energy

Figure 5.5 shows the per access dynamic energy saving of DTLB and ASID-TLB over conventional TLB. FFT benchmark suite with 16 preemptions and 32 TLB entries for various page sizes is used for the performance analysis. With increase in page size, irrespective of the TLB models in use, the miss rate decreases. However, with increase in page size TLB miss penalty increases because of page table walk through. Increased miss penalty leads to increase in per access dynamic energy. The dynamic energy consumption is least for DTLB and is highest for conventional TLB. Access energy increases on an average by 1.65%, 3.53% and 5.66% for DTLB, ASID-TLB and conventional model respectively for every doubling of page size. Average energy savings for

various page sizes of DTLB and ASID-TLB is 6.74% and 1.95% respectively.

Figure 5.6 shows the dynamic energy saving of DTLB and ASID-TLB

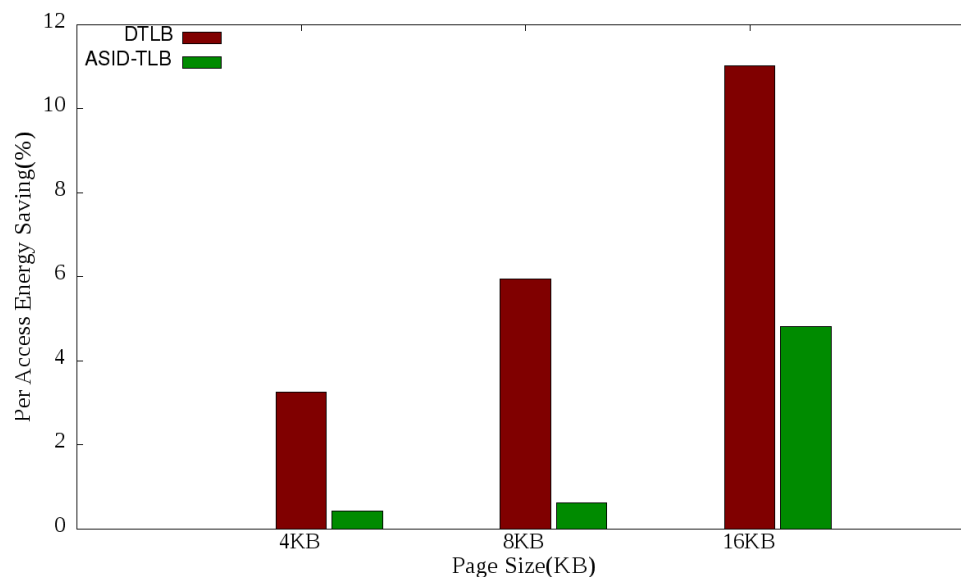


Figure 5.5: Per access dynamic energy with respect to conventional TLB saving for varying page size with 16 preemptions and 32 TLB entries

over conventional TLB for varying number of preemptions. FFT benchmark suite with 4KB page size and 32 TLB entries is used for the performance analysis. The dynamic energy consumption increases marginally by 0.01% with increase in preemptions. This increase is due to the transfer of TLB entries to PCB and vice versa. For ASID-TLB, the access time and energy remains same when task uses reserved entries when it resumes. If not, the access time and energy increases. For conventional model, the access energy increases with increase in preemptions due to invalidation of existing entries. Average per access energy savings of DTLB and ASID-TLB over conventional TLB is 2.34% and 0.33% respectively.

Figure 5.7 shows the per access dynamic energy saving of DTLB and ASID-

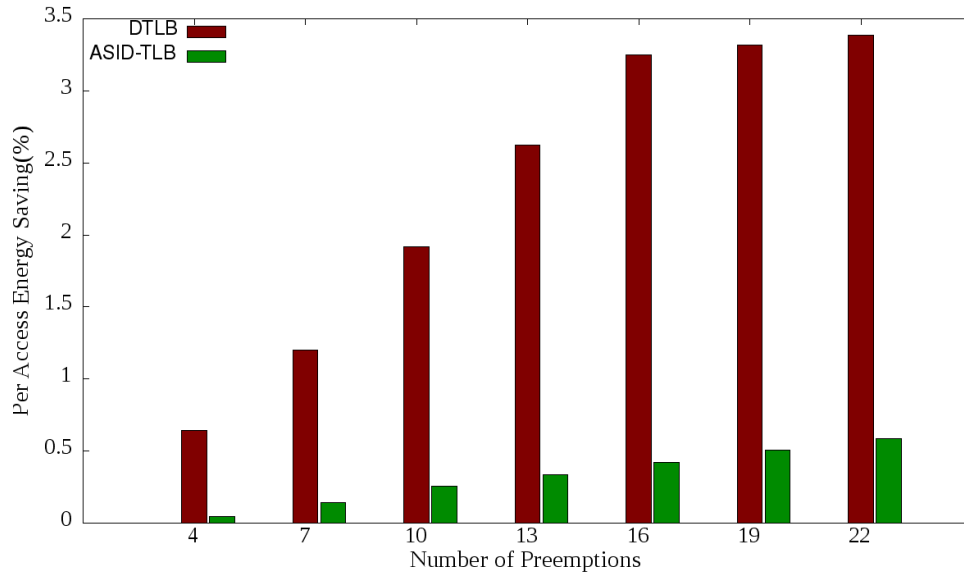


Figure 5.6: Per access dynamic energy with respect to conventional TLB saving for varying preemptions with 4KB page size and 32 TLB entries

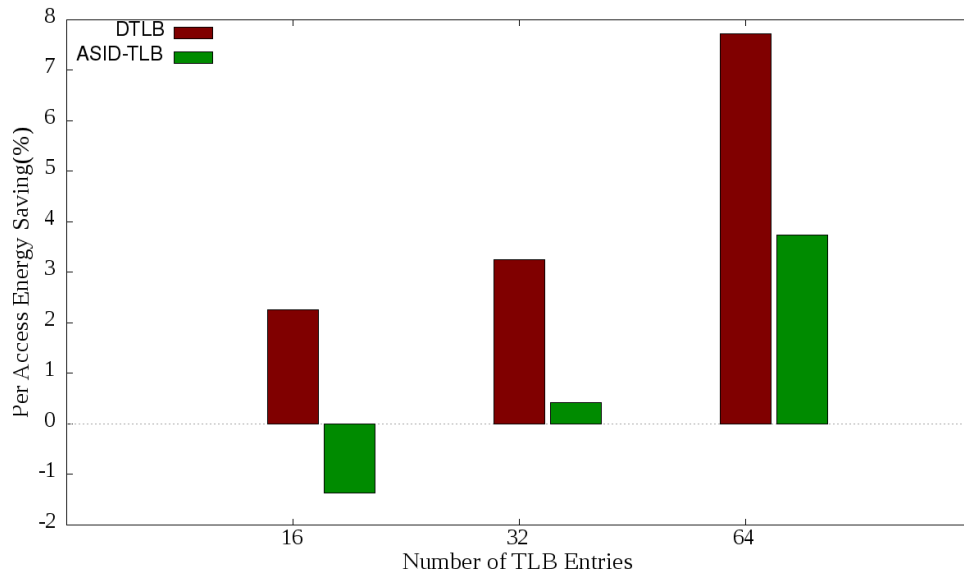


Figure 5.7: Per access dynamic energy with respect to conventional TLB saving for varying TLB entries with 16 preemptions and 4KB page size

TLB over conventional TLB. FFT benchmark suite with 16 preemptions and 4KB page size with varying TLB entries is used for the performance analysis. Access energy increases on an average by 46.75%, 46.90% and 48.25% with doubling of entries for DTLB, ASID-TLB based TLB and conventional TLB respectively. As shown in Figure 5.7, the conventional TLB outperforms ASID-TLB for small TLB entries as reservation in ASID-TLB further reduces number of per task available entries. The dynamic energy consumption of DTLB is the least among all the three models for all page sizes. Overall energy saving of DTLB and ASID-TLB over conventional TLB is 4.41% and 0.93% respectively. On an average DTLB energy saving over ASID-TLB is 31.52%. Figure 5.8 shows per access dynamic energy for various splash benchmark programs. In all cases the dynamic energy consumption of conventional TLB is highest and DTLB consumes the least dynamic energy. Energy consumption is directly proportional to hit rate.

5.4.2.3 Access Time

Figure 5.9 shows the access time reduction of DTLB and ASID-TLB over conventional TLB. FFT benchmark suite with 16 preemptions and 32 TLB entries for various page sizes is used for the performance analysis. On an average access time increases by 2.43%, 3.35% and 4.36% with every doubling of page size for DTLB, ASID-TLB and conventional TLB respectively. The access time of DTLB is the least among these models for various page sizes. Average per access time saving of DTLB over conventional and ASID-TLB is 2.97% and 2.09% respectively.

With increase in number of preemptions, access time of DTLB, ASID-TLB

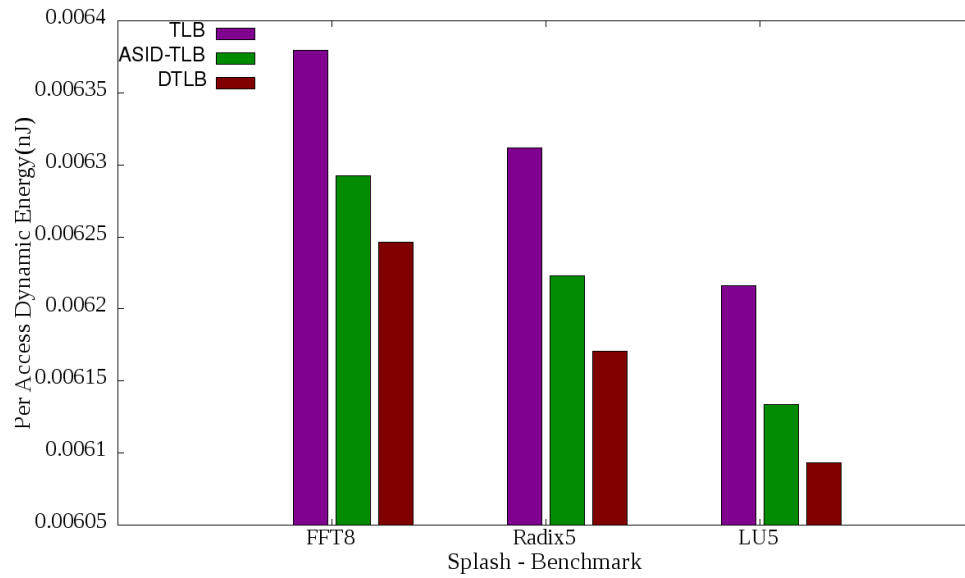


Figure 5.8: Per access dynamic energy for varying Splash benchmark programs

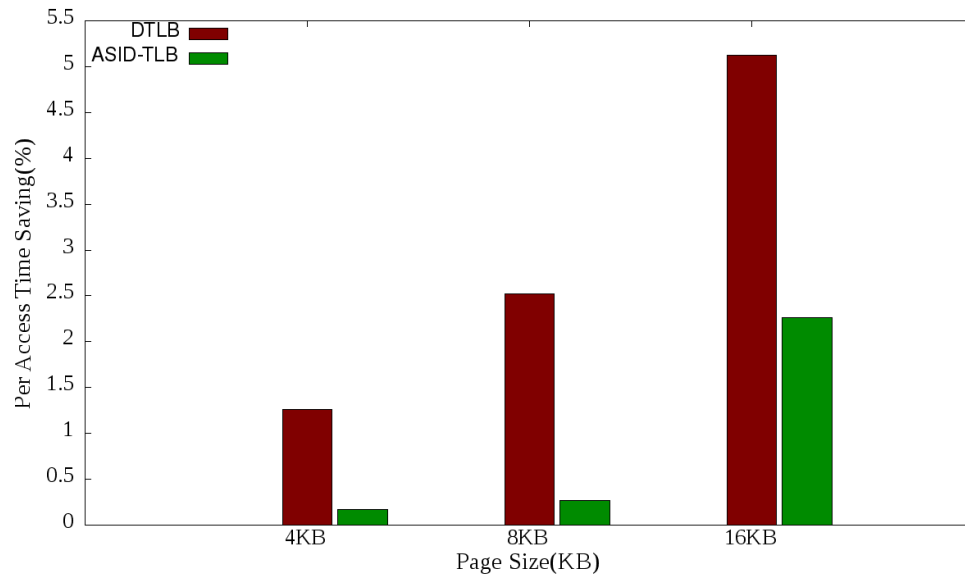


Figure 5.9: Per access time saving with respect to conventional TLB for varying page size with 16 preemptions and 32 TLB entries

and conventional TLB increases on an average by 0.02%, 0.18% and 0.23% respectively. With increase in number of preemptions, the increase in access time is the least in DTLB and the highest in conventional TLB. For DTLB, ASID-TLB and conventional TLB, per access time increases by 0.10%, 0.94% and 1.15% respectively when number of preemptions are increasing from 4 to 22 as shown in figure 5.10

Effective per access time decreases with increase in number of TLB entries for all the models as shown in Figure 5.11. Per access time of DTLB is the least for varying number of TLB entries. Conventional TLB outperforms ASID-TLB when number of DTLB entries are 16. However with increase in number of TLB entries ASID-TLB performs better than conventional model as shown in Figure 5.11.

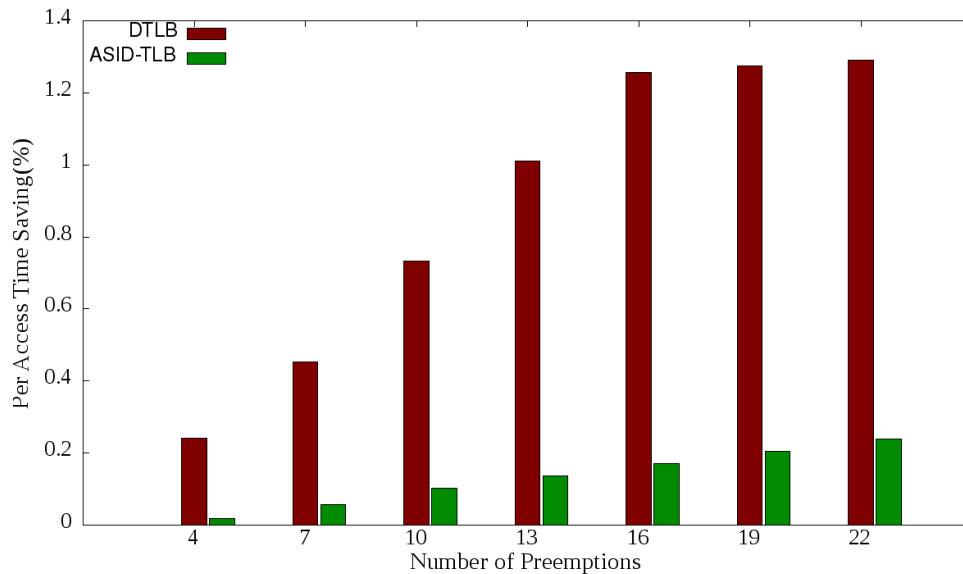


Figure 5.10: Per access time saving with respect to conventional TLB for varying preemptions with 4KB page size and 32 TLB entries

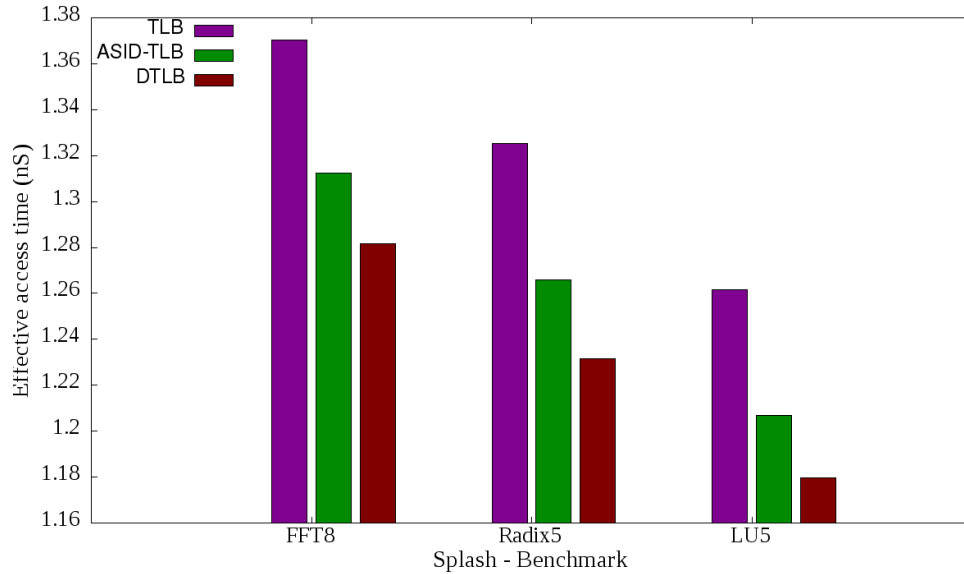


Figure 5.11: Effective access time of 32 entry, 64bits TLB for varying Splash benchmarks

5.5 Conclusion

This chapter proposes a novel TLB architecture Deterministic Translation Lookaside Buffer (DTLB). DTLB offers deterministic performance for low priority real-time tasks. DTLB achieves a tighter upper bound on the WCET of real-time tasks by maintaining a copy of the current TLB in PCB of the task before preemption and transferring the contents back to TLB while resumption of the task. DTLB reduces TLB access time, dynamic energy consumption and effective per access time by increasing TLB hit rate. TLB hit rate is increased by 9.46% as compared to conventional TLB for 4KB page size, with 16 preemptions and 32 TLB entries. DTLB offers on an average 6.74% and 4.91% of dynamic energy savings over conventional TLB

and ASID-TLB respectively. Effective per access time of DTLB reduced by 2.97% and 2.09% as compared to conventional TLB and ASID-TLB.

Chapter 6

DEARCACHE - Deterministic Energy Efficient Process Aware Real-time Cache

6.1 Introduction

The WCET of a task depends on program flow such as loop iterations, decision statements, function calls etc. and on architectural factors such as cache, memory, system resources etc [59]. From an architectural point of view, the pessimistic WCET can be obtained by considering a NO CACHE model as cache is highly non-deterministic. The non-deterministic nature of cache is because of the global replacement and its transparency to operating system and applications. However, this analysis gives an extremely loose WCET which is impractical to accommodate [61].

The RISC systems like MIPS, ARM and RISC V follows load-store architecture. All instructions except load and store variants in these architecture use only one memory access as instruction fetch which is a compulsory memory operation for all instructions. The load and store instructions result in multiple memory accesses. The average number of memory accesses per

instruction will be more than one because of this. The memory subsystem access contributes to a large portion of WCET. Design of energy efficient and performance centric hard real-time system requires a tighter upper bound on WCET. To find a tighter bound on WCET, the system should impose a tighter upper bound on cache misses which depends on number of worst case preemptions. To have a tighter upper bound on the WCET of real-time task, this chapter presents a Deterministic Energy efficient process Aware Real-time Cache (DEARCACHE). In process aware cache design, at each preemption point, the OS transfers job identification number(PID/TID) to the cache controller. Using this information, DEARCache provides tighter upper bound on WCET by eliminating cache related intertask interferences. It guarantees allocation of statically identified minimum ways to each job. DEARCACHE partitions the cache dynamically based on job requirements. It backs up allocated way(s) of job in backup storage, if the running job is in need of more ways and the WCET calculation incorporates the backup time.

6.2 DEARCACHE Architecture

The focus of DEARCACHE is to provide a tighter upper bound on WCET with cache memory and memory subsystem. DEARCACHE consists of dynamically partitioned set-associative private L1 instruction cache and data cache backed up by shared L2 cache and main memory. A new partition is created at job arrival and is dynamically expanded/shrunk during its stay in cache. To achieve deterministic performance, each job is guaranteed to have lower limit on the number of allocated ways at any point in time and upper

limit on the number of times the ways are backed up to memory. The lower limit on number of ways that can be allocated to the job are fixed by offline profiling. The minimum number of ways that can be allocated to the job are called as default ways. A special instruction in instruction set architecture is used at preemption to allocate new ways if the job is new or backed up. At run time, the system finds the need of additional ways with the help of miss counters and then calls the instruction to communicate with cache controller. The cache controller identifies the over allocated jobs or it backs up jobs to free ways so as to allocate it to the executing job. When a job requires a new way, if free cache way is available then it is allocated. If a free way is not available but non-default ways of other jobs are available for use, then find and release the least recently used non-default way from other job and allocate it to the running job. While releasing a way from a job, the non LRU entries in that way are shifted to the remaining allocated ways. If a job has K ways allocated to it and each way has M sets in it, the maximum number of cache line shifting required to keep MRU lines of the victim way is M/K . The time taken to move M/K cache lines to other ways determines the upper bound of transfer time. If non-default ways are not available for allocation, the ways of least recently executed job is backed up into the storage area. When the job resumes its execution next time, the stored backup is loaded back into the cache. Thus, deterministic tighter upper bound on the WCET of job is achieved.

Detailed architecture of DEARCACHE is shown in Figure 6.1. Cache obtains the physical address P and job identifier J of the requested data. After deriving the desired tag, index and offset, the derived index is decoded and

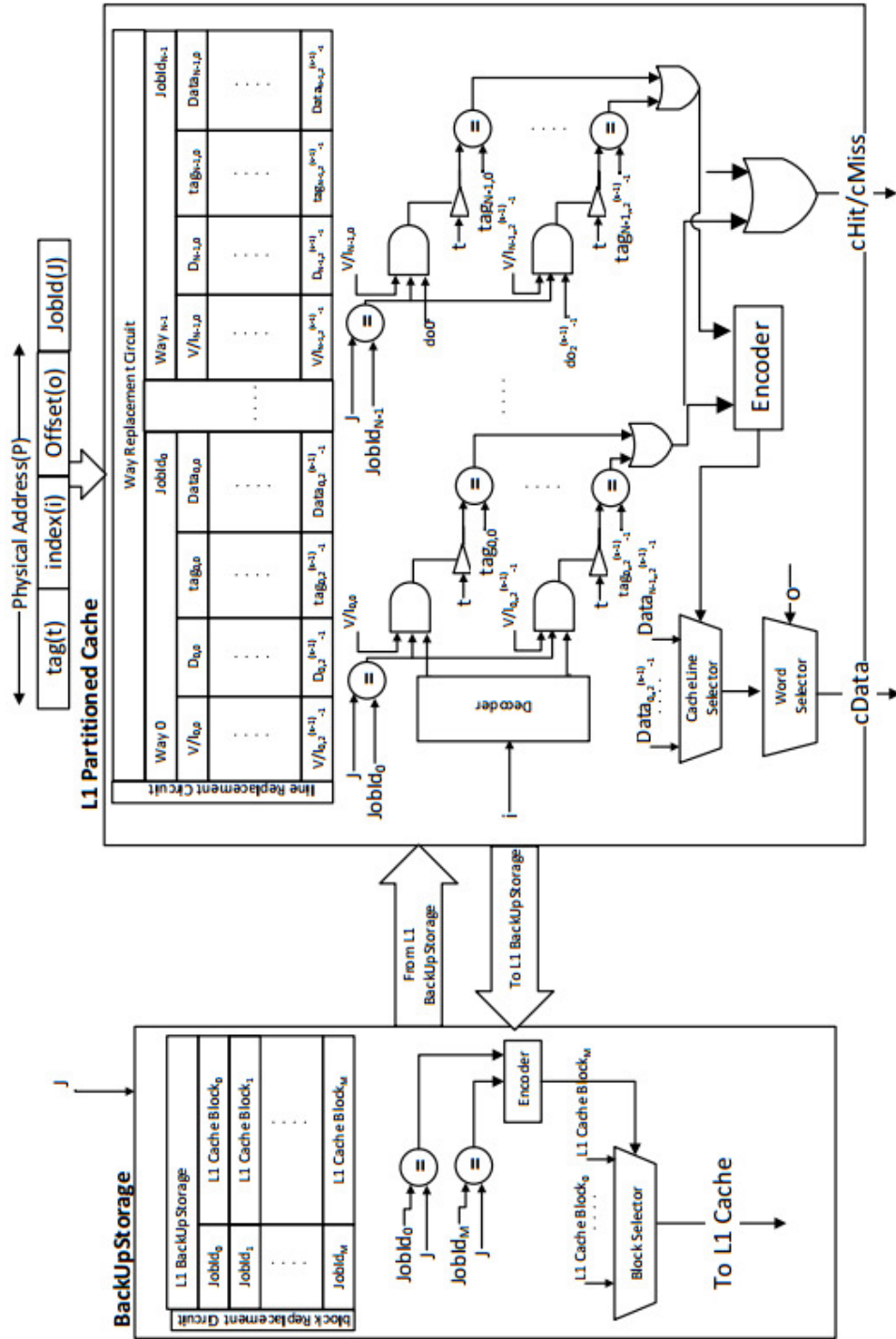


Figure 6.1: Deterministic process aware partitioned real-time cache

the corresponding set is activated. In parallel, J is compared with job identifier $JobId_i$ stored for each way of the cache. Only ways with $JobId_i$ matching is activated. This reduces the total number of active ways and results in saving cache access dynamic energy consumption. The cache components used for modeling energy consumption evaluation of DEARCACHE are given in Table 3.1. SESC [66] is used for finding the parameters in Table 3.1.

6.2.1 DEARCACHE Energy Modeling

Total energy consumption of DEARCACHE, $E_{DEARCACHE}$, is summation of dynamic energy consumption and static energy consumption.

$$E_{DEARCACHE} = DE_{DEARCACHE} + SE_{DEARCACHE} \quad (6.1)$$

where $DE_{DEARCACHE}$ and $SE_{DEARCACHE}$ is dynamic and static energy of DEARCache respectively. The dynamic energy consumption calculation is shown in equation 6.2

$$\begin{aligned} DE_{DEARCACHE} = & N_{accesses} * (E_{dyn_dec} + E_{dyn_op_drvr}) \\ & + E_{dyn_x} * w + N_{misses} * DE_{Tx} \\ & + DE_{DEARCACHE_Overhead} \end{aligned} \quad (6.2)$$

where w , $N_{accesses}$ and N_{misses} represent the number of enabled ways, accesses and misses respectively. The DE_{access} and DE_{Tx} represent the dynamic energy per way access and dynamic energy required to transfer cache block from next level memory respectively. $DE_{DEARCACHE_Overhead}$ is dynamic energy overhead required to access job identifier, most recently used entry transfers

and recency data.

$$DE_{\text{DEARCACHE_Overhead}} = E_{\text{dyn_pid}} + E_{\text{dyn_mruTransfers}} + E_{\text{dyn_recencyData}} \quad (6.3)$$

The static energy is given by,

$$SE_{\text{DEARCACHE}} = (P_{\text{leakage}} + P_{\text{DEARCACHE_Overhead}}) * RT_{\text{DEARCache}} \quad (6.4)$$

Where $P_{\text{DEARCACHE_Overhead}}$ is leakage power to maintain associated circuitry to maintain DEARCACHE and $RT_{\text{DEARCache}}$ is the response time of DEARCache. $RT_{\text{DEARCache}}$ is given by :

$$RT_{\text{DEARCache}} = \text{total cycles required for completion of program} * \text{cycle time} \quad (6.5)$$

6.2.2 DEARCACHE Time Modeling

The DEARCACHE cycle time, $T_{1\text{Cycle}}$, is obtained using SESC. Time calculations of DEARCACHE is as shown in equation 6.6.

$$T_{\text{DEARCACHE}} = T_{\text{access}} + N_{\text{misses_DEARCACHE}} * T_{Tx} \quad (6.6)$$

where T_{access} and $N_{\text{misses_DEARCACHE}}$ represents response time and and miss rate of DEARCACHE respectively.

6.3 Experimental Analysis

The evaluation of DEARCache is carried out by using SESC simulator. Simulator selects appropriate trace file based on job ID. The number of traces in the trace file is proportional to the execution time of the job. The traces offer virtual address along with read/write operation. The page number extracted from the virtual address is given to the TLB. TLB hit offers the frame number corresponding to the page number. The mapping function with the help of page table finds the corresponding frame number in case of TLB miss. Frame number is concatenated with the offset, and is given to the cache memory for instruction/data access. It simulates the cache model for a given configuration and stops its execution when there exist no more jobs in the schedule.

6.3.1 Tighter upper bound on WCET

Figure 6.2 shows the miss rate comparison of CC and DEARCache with varying number of preemptions. The miss rate increases with number of preemptions for CC due to intertask interference. When task resumes after preemption, its data may get replaced by preempting task. This leads to additional intertask conflict misses which increases the cache miss rate and execution time. In DEARCache, dedicated ways are allocated to the task. Cache lines do not get replaced by other executing tasks. Deterministic miss rate is obtained by using DEARCache. This deterministic miss rate gives tighter upper bound on WCET. The WCET of CC is 1.11ns and WCET of DEARCache is 1.09ns. The WCET of DEARCache is reduced by 1.12%. The

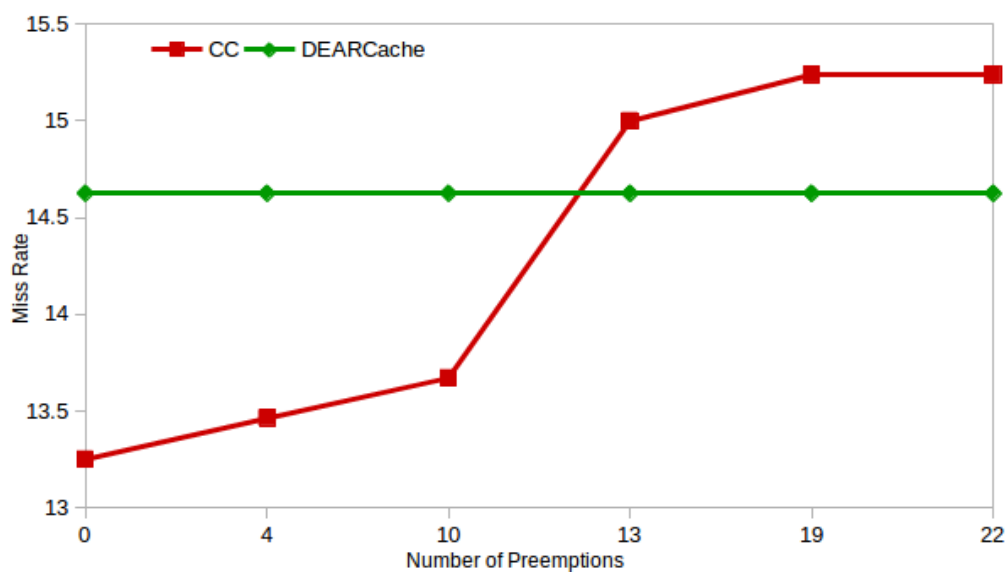


Figure 6.2: Miss rate of CC and DEARCache by varying number of preemptions

reduced WCET can be used to schedule more number of task.

6.3.2 Energy per Access

Figures 6.3 to 6.6 shows the per access energy consumption over varying preemptions, cache size, line size and associativity respectively for CC and DEARCACHE. Access energy consists of two major components - static energy and dynamic energy. The static energy consumption of cache is proportional to its size and operational time. The DEARCACHE consumes additional static energy because of task identifier, and way replacement circuitry. When the number of preemptions are less hit rate of CC is higher than DEARCACHE. When the preemptions increase DEARCACHE offers better hit rate. DEARCACHE takes higher operational time initially because

of lower hit rate. This results in DEARCACHE consuming higher static energy in comparison with CC over varying preemptions, cache size, line size and associativity. On an average, static energy consumption of DEARCACHE

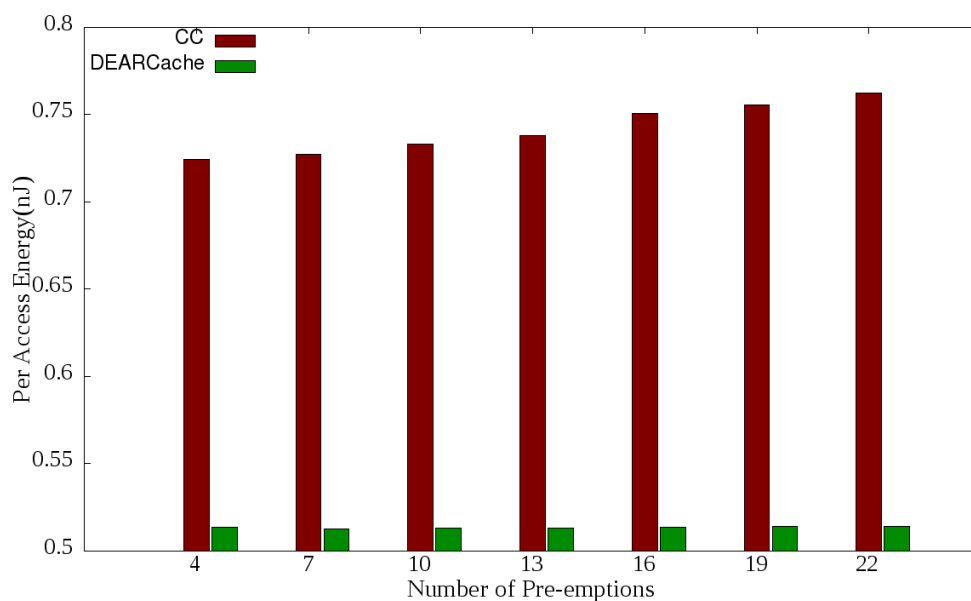


Figure 6.3: Per access energy for varying preemption with 8KB cache size, 32B line size and associativity as 4 way

is 4.39% higher than CC for varying number of preemptions. The static energy consumption increases with increase in the number of preemptions and cache size because of the reduction in hit rate and increase in cache size respectively. The static energy consumption reduces with increase in cache line size because of increase in hit rate and it follows the similar hit pattern with varying associativity.

Dynamic energy consumption of the cache depends on the number of active components and hit rate. Irrespective of the parameters used, dynamic energy consumption of DEARCACHE is lesser than CC. This is mainly because of

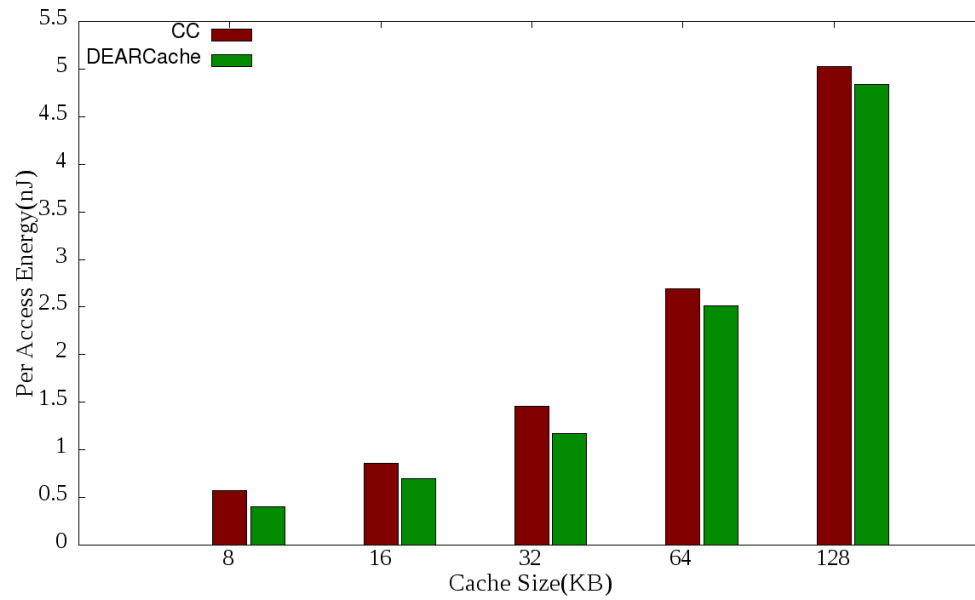


Figure 6.4: Per access energy for varying cache size with 10 preemptions, 32B line size and associativity as 4 way

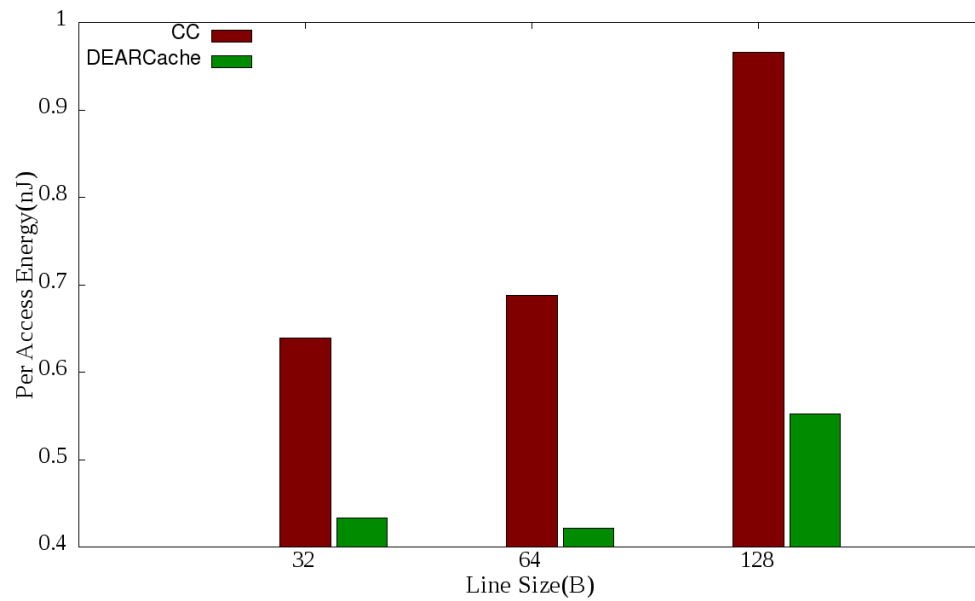


Figure 6.5: Per access energy for varying line size with 10 preemptions, 8KB cache and associativity as 4 way

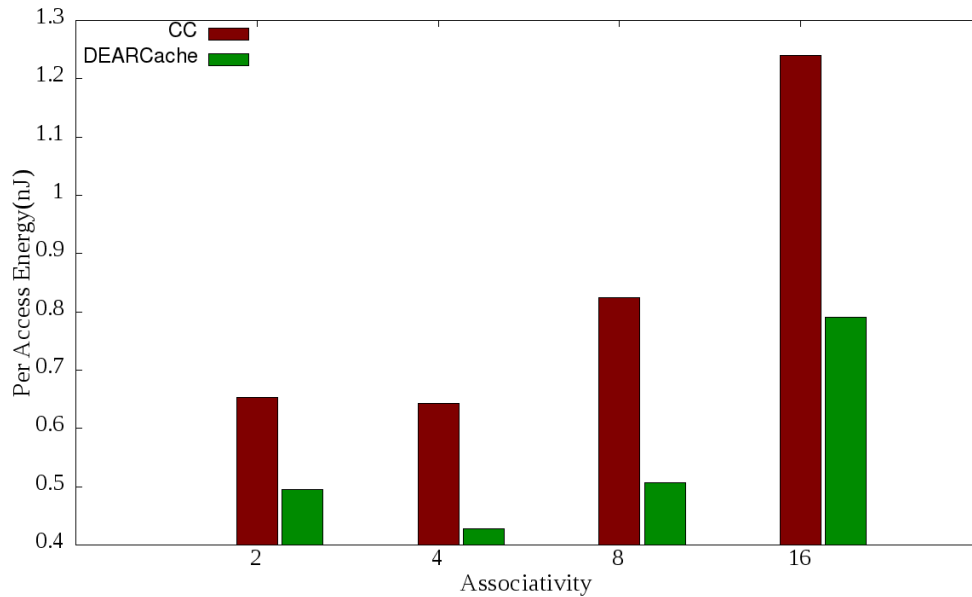


Figure 6.6: Per access energy for varying associativity with 10 preemptions, 8KB cache and 32B line size

the reduction in the number of active ways in use. DEARCACHE enables on an average 1.55 ways against 4 ways in CC for a 4-way set associative cache. Though the active components of DEARCACHE is reduced by 61.25%, the dynamic energy saving is restricted to 38.10% mainly because of the reduction in hit rate. For a 4-way set associative cache, CC offers 1.52% more hit rate than DEARCACHE. As dynamic energy dominates over static energy in overall energy consumption, DEARCACHE offers better energy saving than CC with a deterministic upper bound on WCET. On an average DEARCACHE offers 34.49% per access energy saving over CC.

6.3.3 Response Time

The response time depends on hit rate and hit time. The plot of response time verses varying preemptions, cache size, cache line size and associativity are in figure 6.7, 6.8, 6.9 and 6.10 respectively for CC and DEARCache using FFT benchmark.

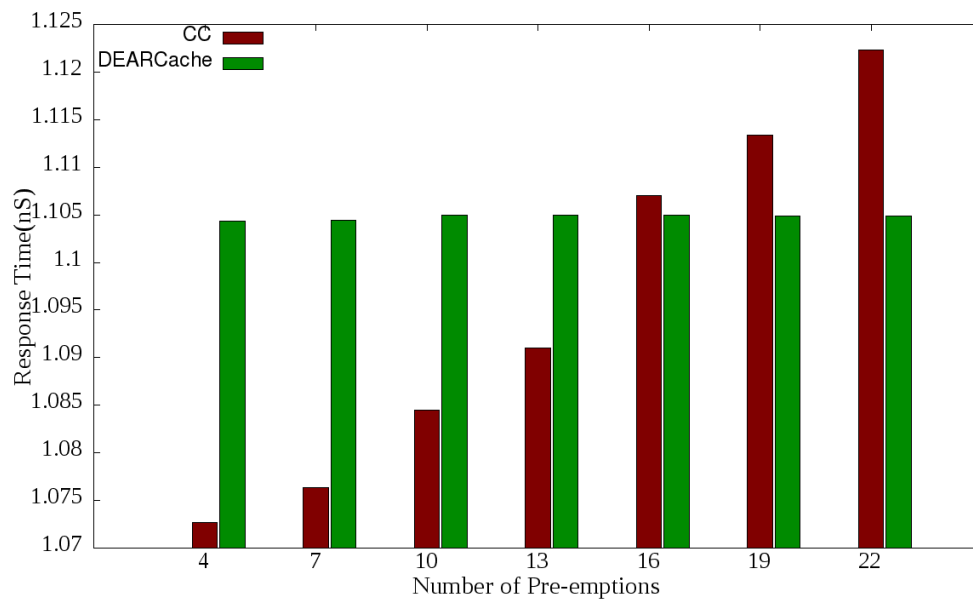


Figure 6.7: response time for varying preemption with 8KB cache size, 32B line size and associativity as 4 way

As shown in figure 6.7, the hit rate decreases with increase in preemptions contributing to increase in the response time. The response time also increases with increase in associativity due to increased cycle time. The increase in cache size and cache line size results in decreasing response time. This is because of the increase in cache hit rate. Experimental evaluation shows that response time of CC, DEARCache and NO CACHE model is 0.72ns, 0.75ns

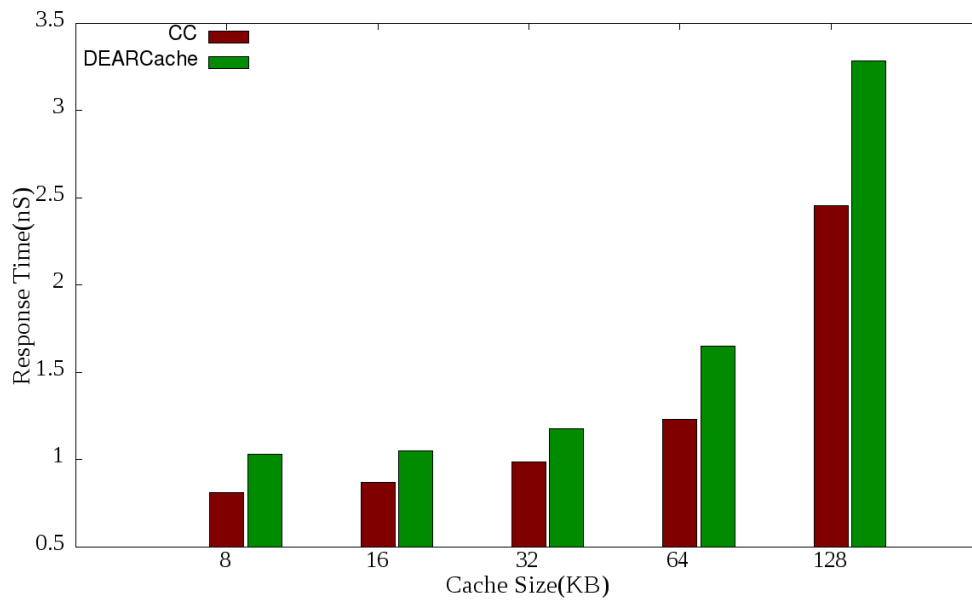


Figure 6.8: response time for varying cache size with 10 preemptions, 32B line size and associativity as 4 way

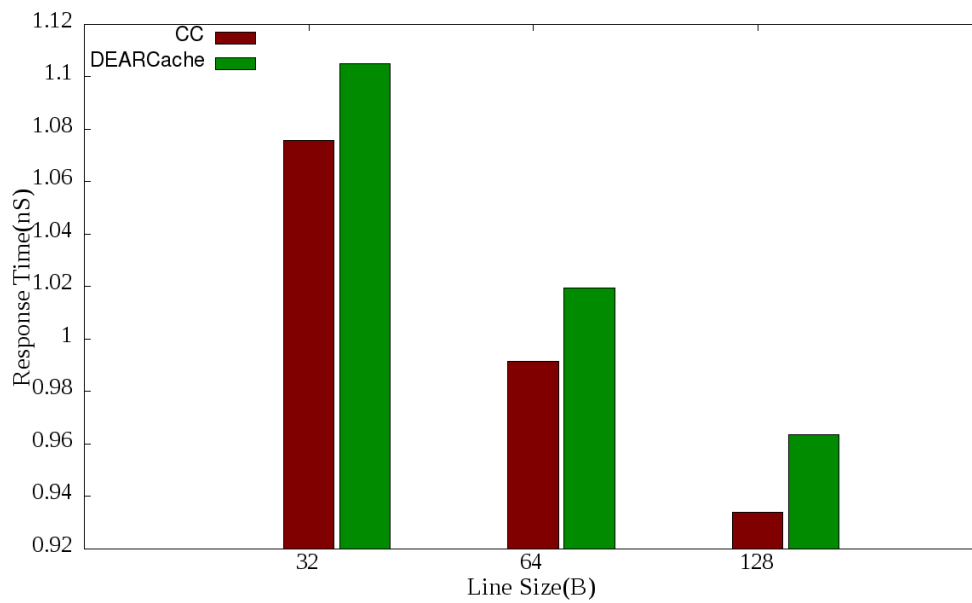


Figure 6.9: response time for varying line size with 10 preemptions, 8KB cache and associativity as 4 way

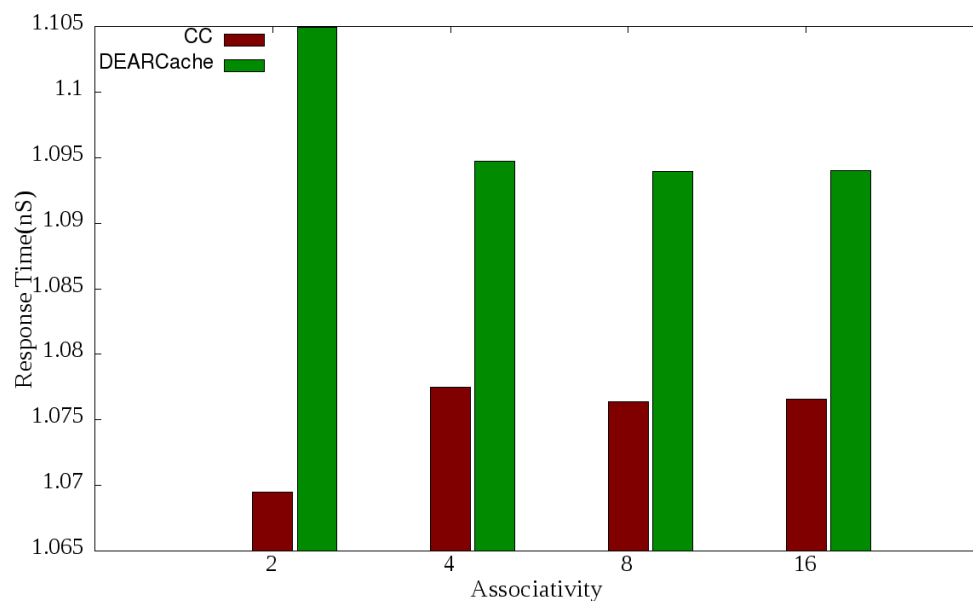


Figure 6.10: response time for varying associativity with 10 preemptions, 8KB cache and 32B line size

and 2.91ns respectively. DEARCache requires additional 4.37% of response time as compared to CC. The response time of DEARCache is improved by 287.09% over NO CACHE model.

6.3.4 Energy and Time comparison with energy efficient caches

The comparison of dynamic energy consumption and static energy consumption of DEARCache with respect to CC, WP and WH is as shown in Figures 6.11 and 6.12 respectively. Hit rate of CC, WP and WH is the same and is higher than DEARCache when number of preemptions . Irrespective of the measuring parameters used dynamic energy consumption of CC is the highest. CC accesses all the ways for every access. The number of ways

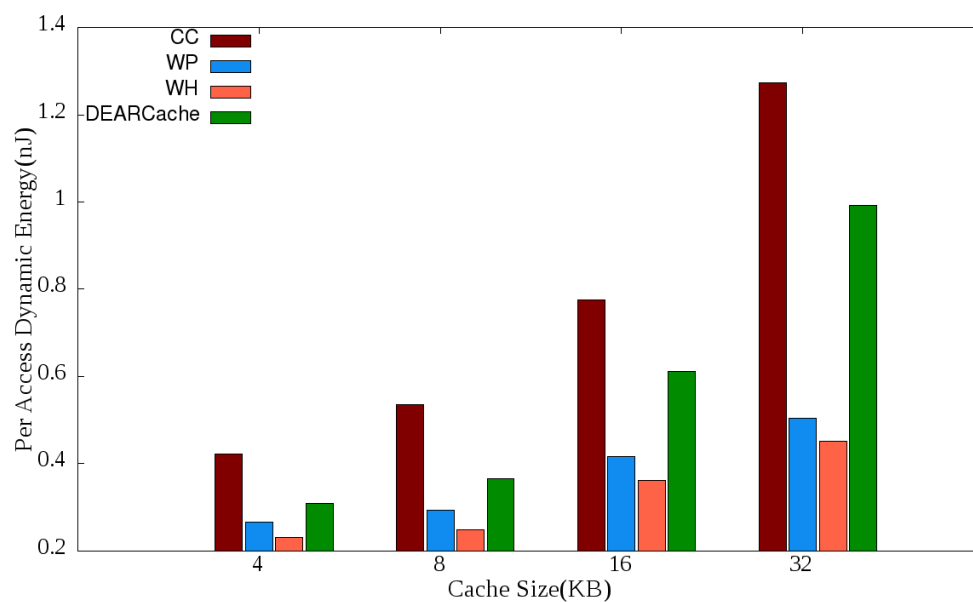


Figure 6.11: Per access dynamic energy for varying line size with 10 preemptions, 8KB cache size and associativity as 4 way

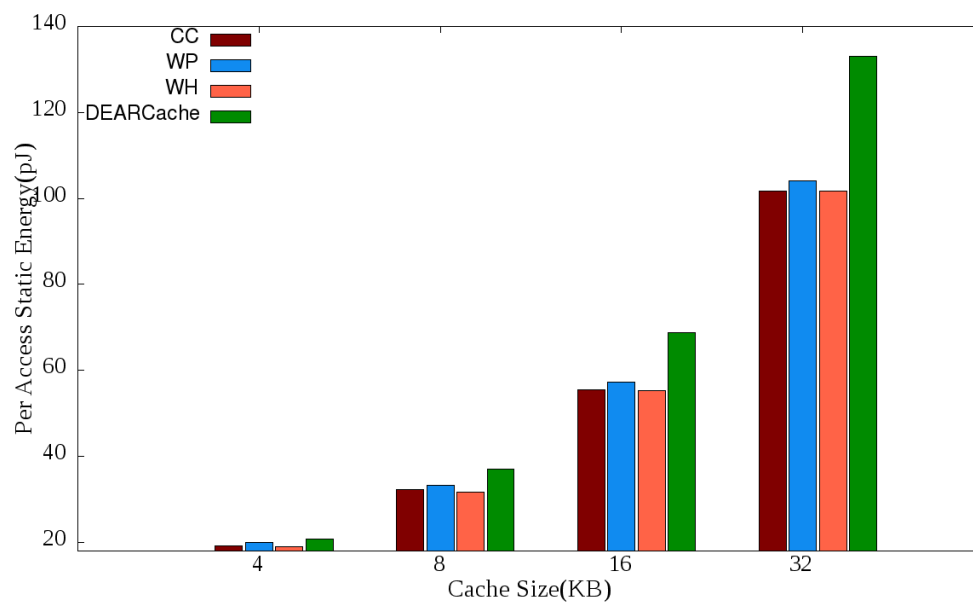


Figure 6.12: Per access Static energy for varying line size with 10 preemptions, 8KB cache size and associativity as 4 way

accessed increases dynamic energy consumption of CC.

WH consumes the least dynamic energy. The number of ways accessed by WH is lesser than CC and WP. The prediction miss in WP results in accessing all other ways. This increases dynamic energy consumption of WP over WH.

Identification based reservation in DEARCache reduces associativity per task. The reduced associativity decreases hit rate. The decreased hit rate increases dynamic energy consumption of DEARCache over WP and WH. The number of ways accessed in case of DEARCache is less than CC hence dynamic energy consumption of DEARCache is better than CC.

CC consumes the least static energy. The prediction circuit and halting circuit incurs additional static energy in case of WP and WH respectively. The least hit rate of DEARCache along with additional DEARCache overhead results in highest static energy is consumption of DEARCache.

Early detection of cache misses due to halt tag misses in case of WH gives it the least response time. Prediction misses in case of WP results in additional response time over CC. The response time of DEARCache is the highest among all the cache architectures.

DEARCache can be made energy efficient by incorporating prediction circuit and halt tag circuit along with process identification. This will reduce the dynamic energy consumption of DEARCache further. The static energy consumption and response time of DEARCache can be improved by using shared way which will be discussed in chapter 7. Data shared between task can be stored in shared way. This results in improving DEARCache hit rate and hence decreases dynamic energy consumption, static energy consumption and response time.

6.4 Conclusion

Meeting all deadlines with least energy consumption is the major design constraints of battery powered hard real-time systems. Real-time scheduling algorithms perform schedulability analysis by considering the pessimistic WCET. In conventional systems, the WCET is loosely bound because of the non-deterministic nature of the memory subsystem. Cache memory is an integral and crucial part of the memory subsystem. This chapter proposed DEARCACHE a deterministic real-time cache memory. DEARCACHE eliminates intertask interference by allocating dedicated cache ways to tasks in execution. It obtains tighter upper bound on number of cache misses. DEARCACHE reduces dynamic energy consumption by 38.10% for 4-way set associative cache configuration over CC with 4.39% overhead of static energy. Response time of DEARCACHE is improved 3.87 times over NO CACHE model and with an additional requirement of 4.37% of response time as compared to CC. The WCET of DEARCache is reduced by 1.12%. The reduced WCET can be used to schedule more number of task. DEARCache can be made energy efficient by using prediction circuit and halt tag array along with process identification. This will reduce the dynamic energy consumption of DEARCache further. The static energy consumption and response time of DEARCache can be improved by using shared cache to store shared data.

Chapter 7

DREAM - Deterministic Memory Subsystem

7.1 Introduction

Tighter schedulability analysis is crucial in hard real-time systems as deadline misses may lead to catastrophic failure. The tighter upper bound on WCET at all levels of memory subsystem - TLB, various levels of caches and main memory - required to ensure scheduling feasibility. This chapter designs memory subsystem to get deterministic performance at all levels. The inter-task and intra-task interference at all the levels of memory hierarchy makes the memory sub-system non-deterministic. To get deterministic performance of L2 cache, this chapter proposes a Deterministic Energy Efficient Process aware(DEEP) design. DEEP cache is shared among all tasks running on different cores of the processor. It allocates minimum number of ways to each task which is identified as a result of static analysis. It dynamically increases/decreases the number of allocated ways based on task requirements. It backs up allocated way(s) of task in backup storage if the running job is in need of more ways and if the WCET calculation incorporates this time. This chapter also proposes an integrated design of deterministic memory

named Deterministic REAL-time Memory system (DREAM). DREAM achieves deterministic performance at TLB and L1 cache by incorporating DTLB and DEARCache with DEEP as L2 cache. Detailed discussion about DTLB and DEARCache are done in chapter 5 and chapter 6 respectively.

7.2 DREAM Architecture

Each core in DREAM architecture consists of a TLB, private split L1 instruction and data caches, shared unified L2 cache and main memory. To avoid the complication of secondary storage, this work assumes that the main memory is large enough to hold the instructions, stack, heap and paging information of the running jobs [72] [71]. This can be very well relaxed with a semi-conductor memory like SSD as secondary storage. Figure 7.1 shows the proposed memory subsystem architecture of DREAM.

DTLB offers tighter WCET bound on real-time tasks. It guarantees minimum number of TLB misses. To achieve deterministic performance, the L1 cache is designed as a DEARCache. L2 is designed as DEEP shared cache which locks cache ways for tasks to achieve deterministic performance in MC systems [73]. It is shared among tasks and partitioning is identified based on task IDs (PID/TID). The DEEP cache is a V-way cache implementation that has a conflict miss counter for each set [73]. It increases the associativity of an index if the number of conflict misses in that index increases beyond a particular threshold. On increasing the associativity of an index, the associativity of another index is decreased such that the total number of enabled cache lines remain constant. The data mapping information is stored in the tag entries

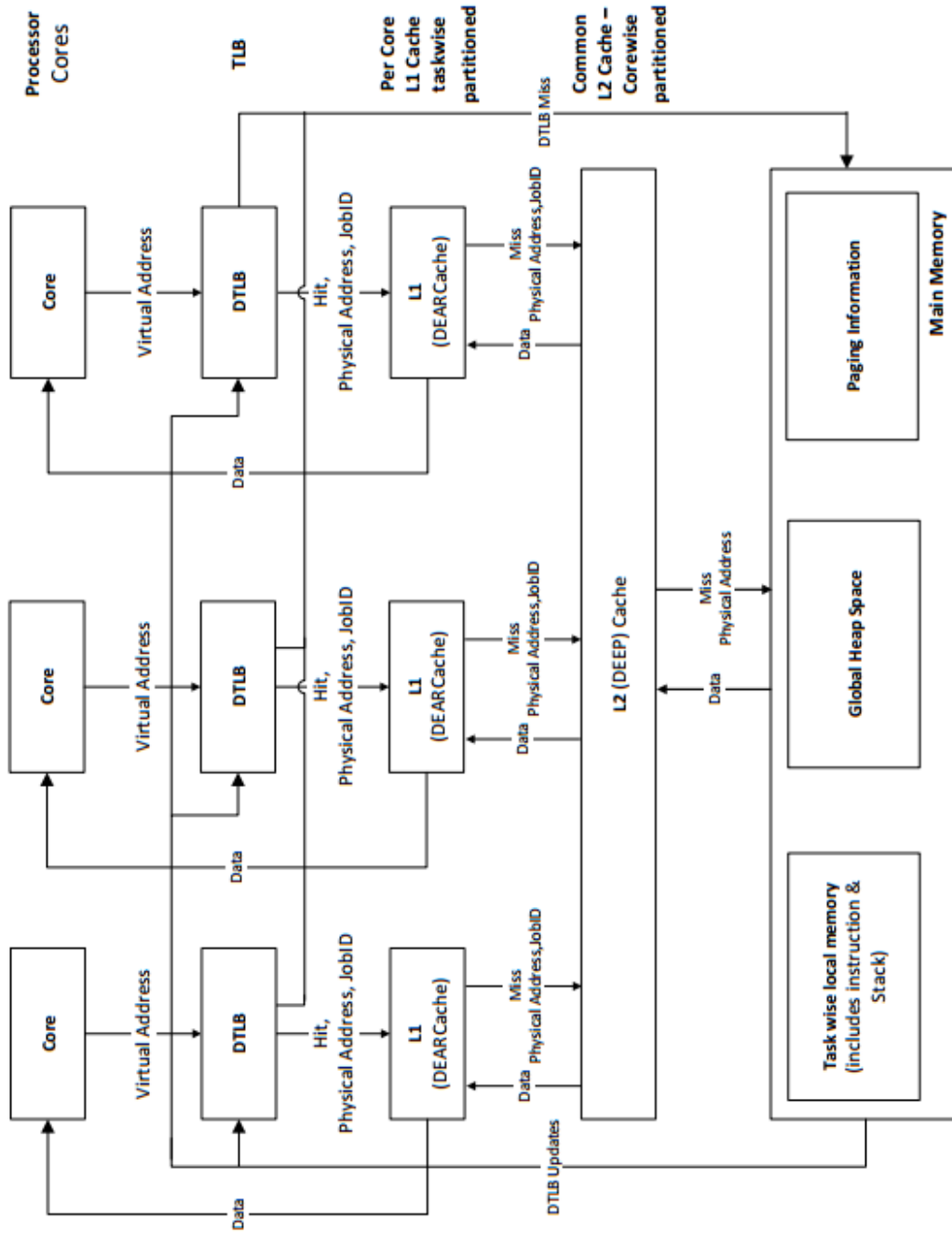


Figure 7.1: DREAM Memory Subsystem

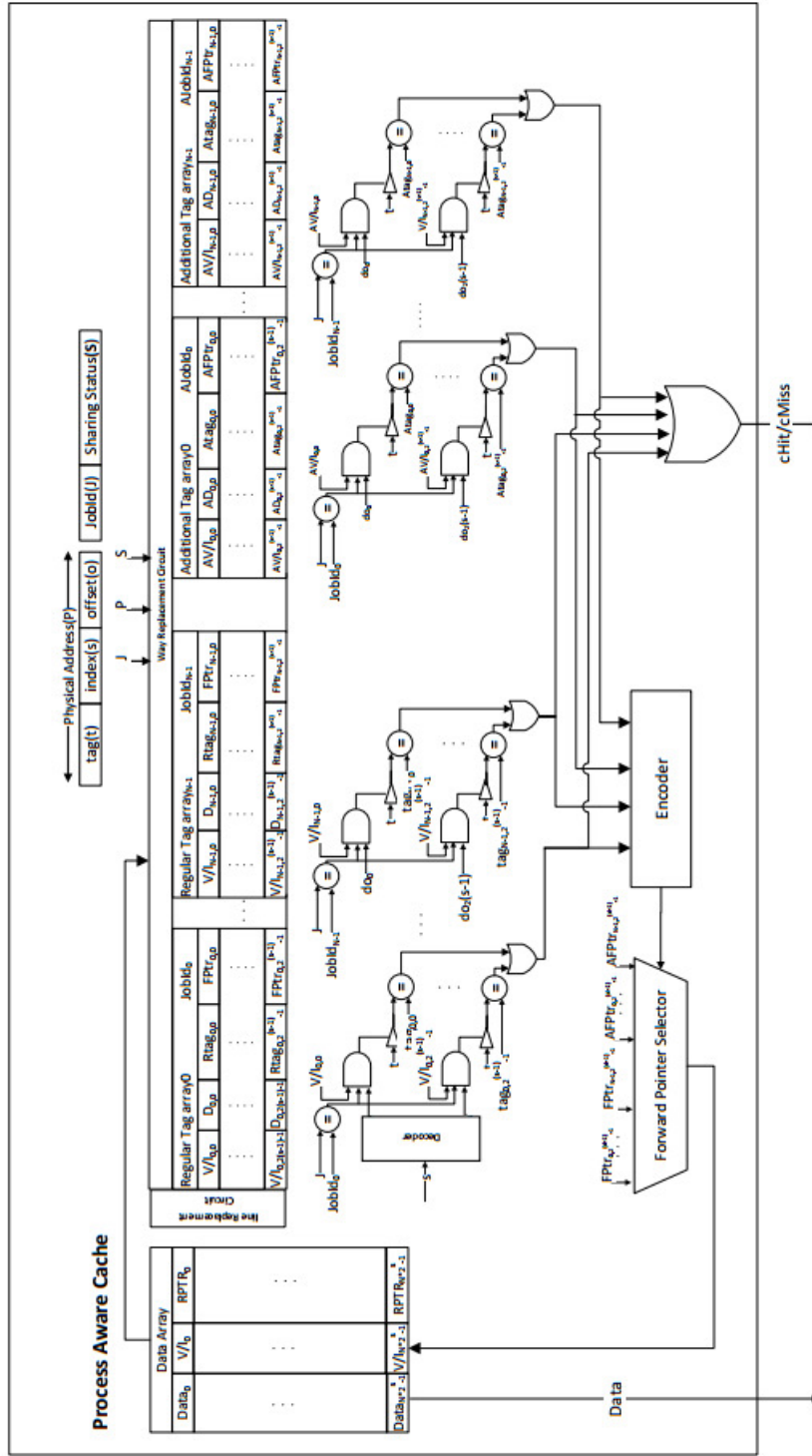


Figure 7.2: Deterministic Energy Efficient Process aware(DEEP) Cache

such that it can be directly mapped to one location in the data array. The data corresponding to tags in the extra or extended tag array are stored in locations corresponding to the disabled cache lines of other indexes. Figure 7.2 shows detailed architecture of DEEP cache. DEEP cache controller is provided with job identification J and physical address P . Tag comparisons and validity check is done only for the ways allocated to job J . On cache hit, data from data array is accessed by using forward pointers stored in tag array. Performance of the proposed system can be improved by providing a shared cache along with the partitioned cache. Shared cache improves cache hit rate when entries are shared between multiple jobs. The allotted partitions are accessed for the non shared cache accesses and shared cache is accessed otherwise. The detailed algorithm which illustrates the working of L2 cache architecture is given in algorithm 2. Detailed architecture of the complete memory subsystem architecture of DREAM with Shared way (DREAMS) is as shown in Figure 7.3. The shared space can be N-way set associative to fully associative where N is at least double in number than the associativity of the partitioned cache. Partitioned cache is accessed only for non-shared accesses. This reduces number of active components in the system.

7.3 Time, Power and Energy Modeling

The cache components used for the performance and energy consumption evaluation of DREAM are given in Table 3.1. SESC time, power and energy model is used for finding the energy and time parameters from Table 3.1.

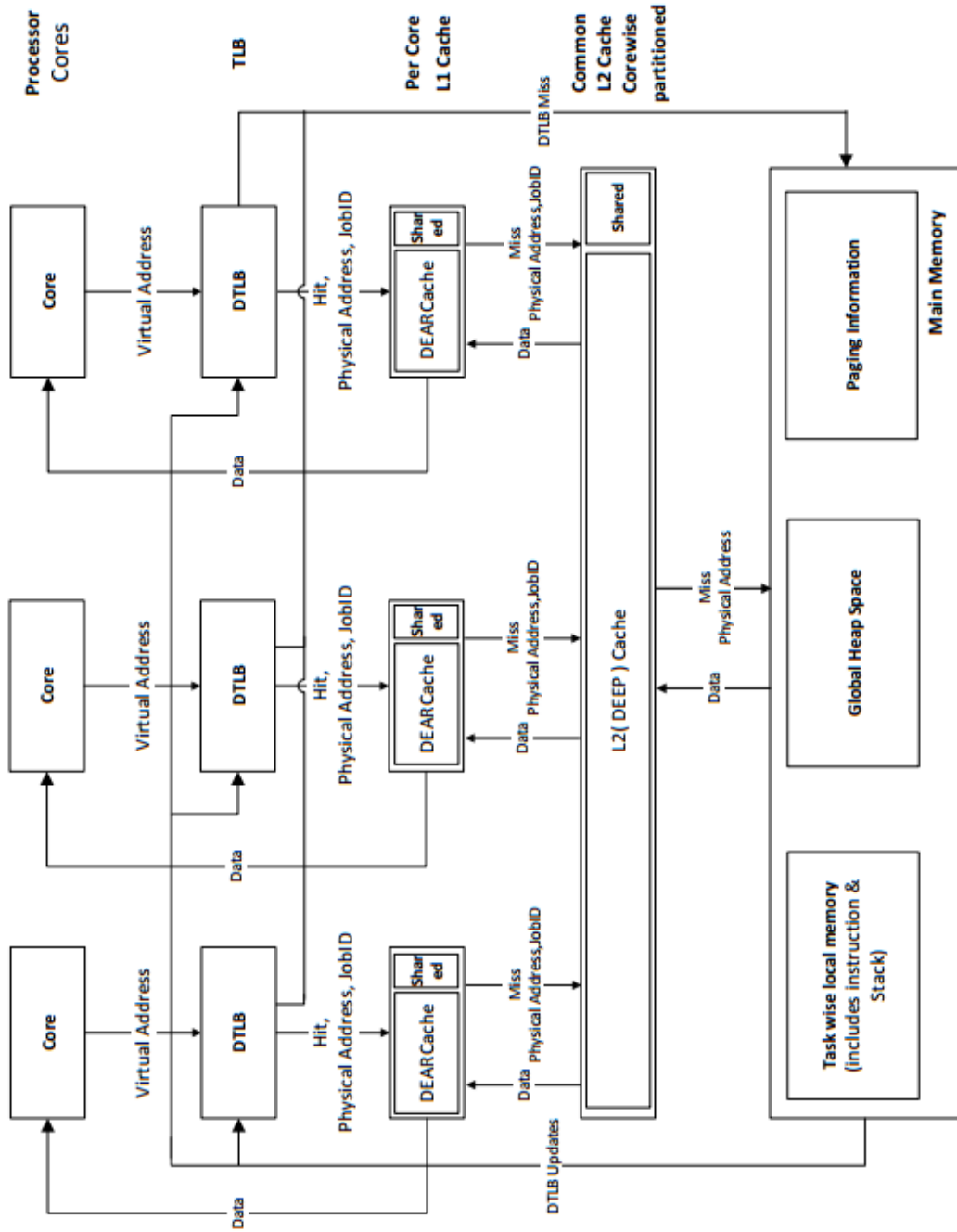


Figure 7.3: DREAM Memory Subsystem with shared way

Algorithm 2: Accessing Shared DEEP cache

Input: Physical Address P , Cache sharing status S , and task Identifier T

Output: Process aware cache or Shared cache hit/miss and data

```

1 begin
2   if  $P$  is shared then
3     Access shared L2 cache;
4     if Match found for tag bits of  $P$  in tag array of shared cache
       then
5       | Transfer requested data from / to L1; return;
6     else
7       | Select a cache line as victim cache line in shared cache for
         replacement;
8       | Transfer data from memory to L2 cache line and transfer
         data from / to L1 cache; return;
9     end
10  end
11  else
12    Extract tag bits  $t$ , set index bits  $s$ , and offset bits  $o$  from  $P$ ;
13    Identify  $w$  ways allotted to task,  $T_i$ ;
14    Compare tag bits  $t$ , with tags of  $w$  ways of task  $T_i$  in the desired
       set  $s$ ;
15    if Match found then
16      | Transfer requested data from / to L1 cache; return;
17    if Any Cacheline at  $s$  in  $w$  ways is free then
18      | Transfer data from memory to free cache line in L2 and
         transfer it to L1; return;
19    else
20       $conflictMissCnt[s]++$ ;
21      if  $w < maxAllotment_j$  then
22        if  $conflictMissCnt[s] \leq conflictThreshold$  then
23          |  $allotedWay =$  Allot L2 cache way with regular tag
             array;
24        else
25          |  $allotedWay =$  Allot L2 cache way with additional
             tag array;
26          |  $conflictMissCnt[s] = 0$ ;
27        end
28      else
29        | Select  $allotedWay$  using replacement policy;
30      end
31      Transfer data from memory to cacheline at( $s, allotedWay$ )
         and transfer data from / to L1 cache; return;
32    end
33  end

```

7.3.1 Energy Modeling of DEEP Cache with Shared way - DEEPS

Total energy consumption of DEEPS, (E_{DEEPS}), is summation of dynamic energy consumption and static energy consumption.

$$\begin{aligned} E_{\text{DEEPS}} = & \textit{DynamicEnergyDEEPS} (DE_{\text{DEEPS}}) \\ & + \textit{StaticEnergyDEEPS} (SE_{\text{DEEPS}}) \end{aligned} \quad (7.1)$$

The dynamic energy consumption calculation is shown in equation 7.2

$$\begin{aligned} DE_{\text{DEEPS}} = & N_{\text{accesses_PC}} * (E_{\text{dyn_dec}} + E_{\text{dyn_op_drv}}) + E_{\text{dyn_x}} * w + \\ & N_{\text{accesses_SC}} * (E_{\text{dyn_dec_SC}} + E_{\text{dyn_op_drv_SC}} + \textit{Associativity_SC} * \\ & E_{\text{dyn_x_SC}}) + N_{\text{misses}} * DE_{Tx} + DE_{\text{DEARCACHE_Overhead}} \end{aligned} \quad (7.2)$$

where w , $N_{\text{accesses_PC}}$, $N_{\text{accesses_SC}}$ and N_{misses} represent the number of enabled ways, non-shared accesses, shared accesses and misses respectively. The $DE_{\text{DEEPS_Overhead}}$ and DE_{Tx} represent the dynamic energy overhead and dynamic energy required to transfer cache block from next level memory respectively. $E_{\text{dyn_dec_SC}}$, $E_{\text{dyn_x_SC}}$, $E_{\text{dyn_op_drv_SC}}$ and $\textit{Associativity_SC}$ represents decoder energy, access energy, output driver energy and associativity related to shared cache.

Calculations of static energy consumption is as per equation 7.3

$$SE_{\text{DEEPS}} = (P_{\text{leakage}} + P_{\text{DEEPS_Overhead}}) * RT_{\text{DEEPS}} \quad (7.3)$$

Where $P_{DEEPS_Overhead}$ is leakage power to maintain associated circuitry to maintain DEEPS and RT_{DEEPS} is the response time of DEEPS. RT_{DEEPS} is given by :

$$RT_{DEEPS} = \text{total cycles required for completion of program} * \text{cycle time} \quad (7.4)$$

7.3.2 Energy Modeling of DREAM system with Shared cache - DREAMS

Energy consumption of DREAMS, (E_{DREAMS}), is summation of energy consumption of DTLB, DEARCACHE and DEEPS.

$$E_{DREAMS} = E_{TLB_DTLB} + E_{DEARCACHE} + E_{DEEPS} \quad (7.5)$$

7.3.3 Time Modeling of DEEPS

$$T_{DEEPS} = T_{access} + N_{misses_DEEPS} * T_{Tx} \quad (7.6)$$

where T_{access} and N_{misses_DEEPS} represents response time and miss rate of DEEPS respectively.

7.3.4 Time Modeling of DREAMS

Total execution time of DREAMS is summation of time of DTLB, DEARCACHE and DEEPS.

$$T_{DREAMS} = T_{TLB_DTLB} + T_{DEARCACHE} + T_{DEEPS} \quad (7.7)$$

7.4 Performance Evaluation

SESC is used as framework for the analysis of DEEPS and DREAMS is discussed in section 3.5 of chapter 3.

7.4.1 Experimental Analysis - L2 as DEEP Cache

7.4.1.1 Dynamic Energy per Access

Per access dynamic energy consumption of CC, DEEP and DEEPS by varying number of preemptions, cache size, cache line size and associativity is shown in figures 7.4, 7.5, 7.6 and 7.7.

The average cache hit rate of CC and DEEP cache with varying number of preemptions is 88.59% and 85.92% respectively. The average number of ways accessed in case of CC and DEEP cache with varying number of preemptions is 4 and 1.57 ways respectively. Although the per access ways of DEEP cache is lower than CC, whenever the hit rate of CC is substantially higher over DEEP, the per access dynamic energy of CC cache is lower than that of DEEP. The dynamic energy consumption of DEEP cache can be reduced by using a shared way. The shared entries between tasks are transferred to shared way. This improves the DEEPS cache hit rate and hence reduces the dynamic energy consumption. On an average per access dynamic energy of DEEP cache is higher than CC by 1.12% and per access time of CC cache is higher than DEEPS by 71.69%.

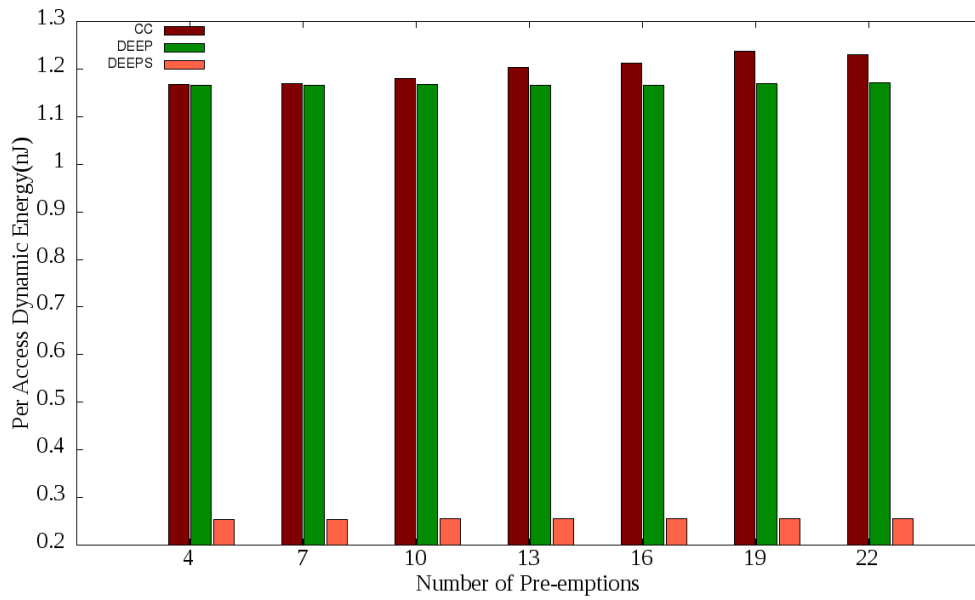


Figure 7.4: Per access dynamic energy for a 4 way, 32B, 8KB cache with varying preemptions

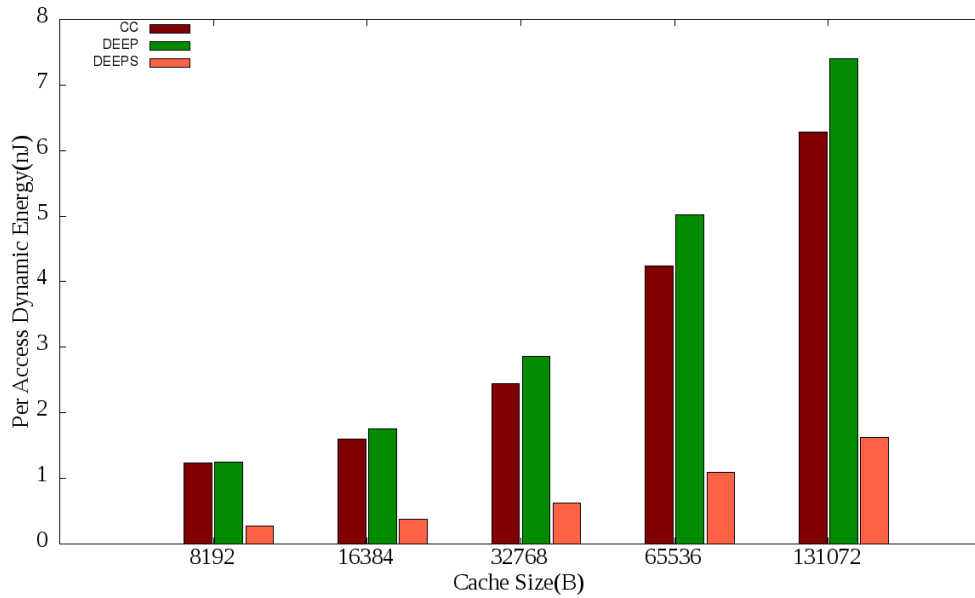


Figure 7.5: Per access dynamic energy for a 4 way, 32B line size with varying cache size [#preemptions = 10]

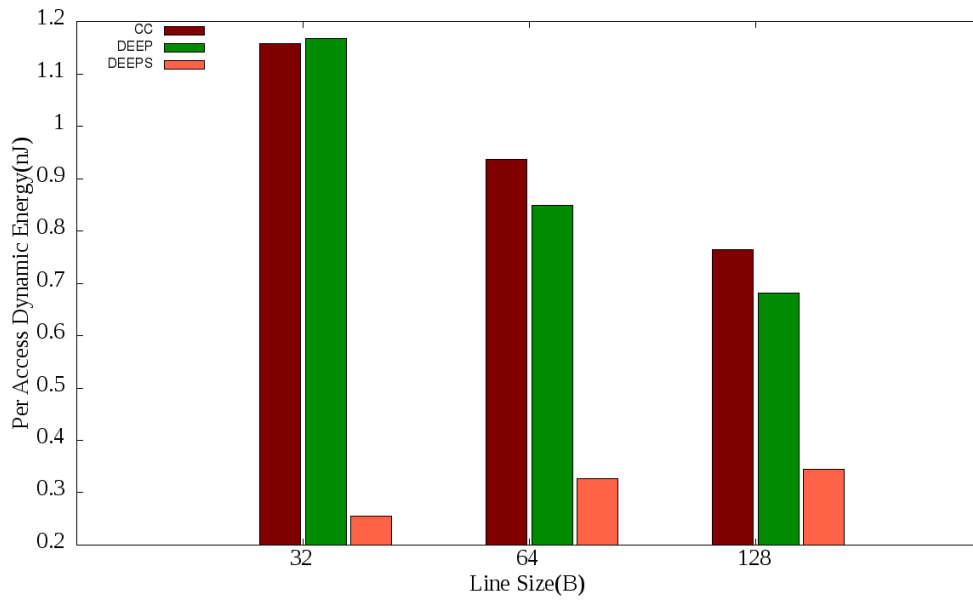


Figure 7.6: Per access dynamic energy for a 4 way, 8KB cache with varying line size [#preemptions = 10]

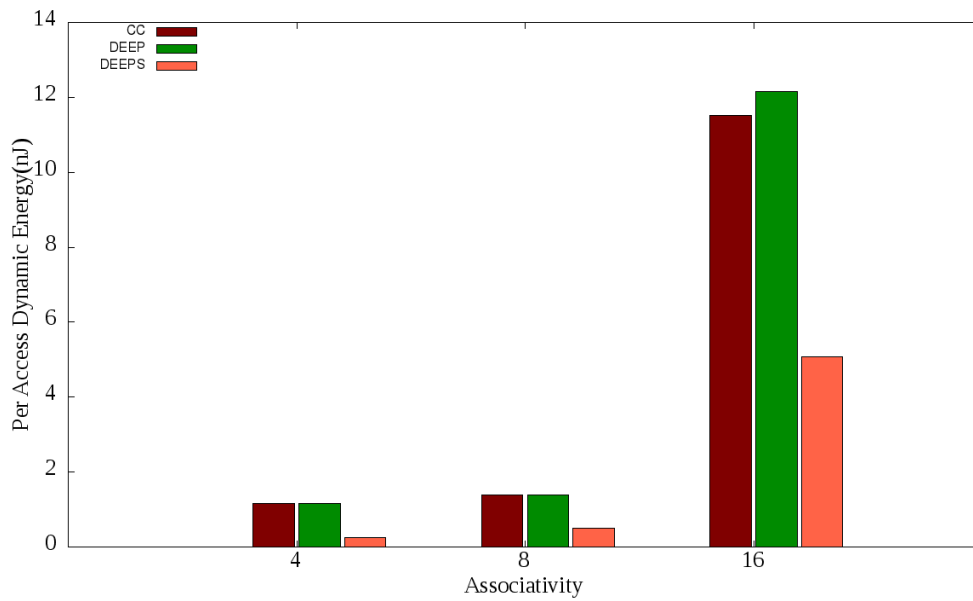


Figure 7.7: Per access dynamic energy for a 32B, 8KB cache with varying associativity [#preemptions = 10]

7.4.1.2 Static Energy per Access

Per access static energy consumption of CC, DEEP and DEEPS by varying number of preemptions, cache size, cache line size and associativity is shown in figures 7.8, 7.9, 7.10 and 7.11.

The per access static energy consumption of DEEP cache is higher than CC.

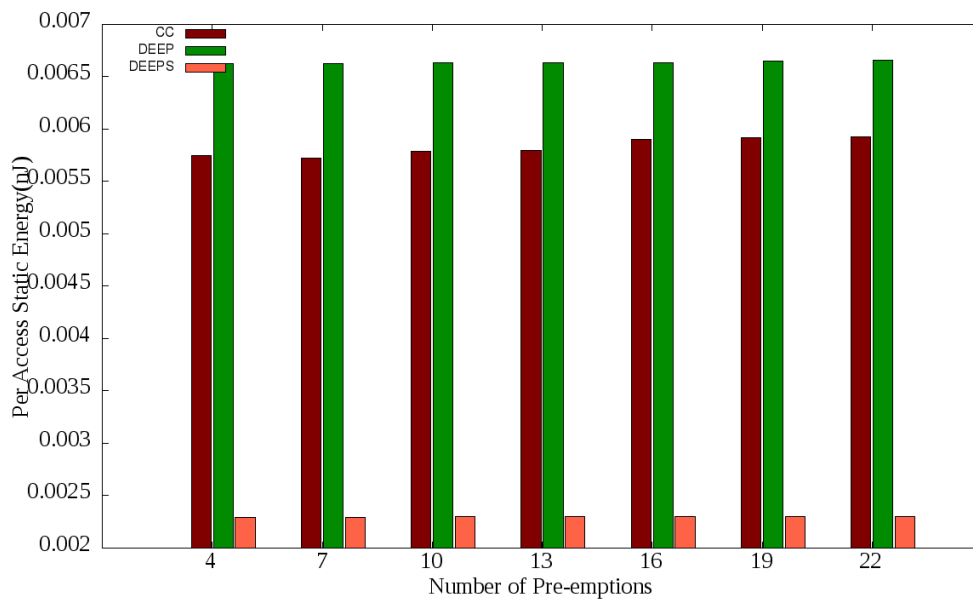


Figure 7.8: Per access static energy for a 4 way, 32B, 8KB cache with varying preemptions

DEEP cache has additional static energy for maintaining core identification, way replacement circuit and extra tag array. Static energy requirement of DEEP cache can be reduced by making one of the way as shared way. The shared way improved the cache hit rate and hence reduced static energy. On an average per access static energy of DEEP cache is higher than CC by 15.78% and per access time of CC cache is higher than DEEPS by 50.02%.

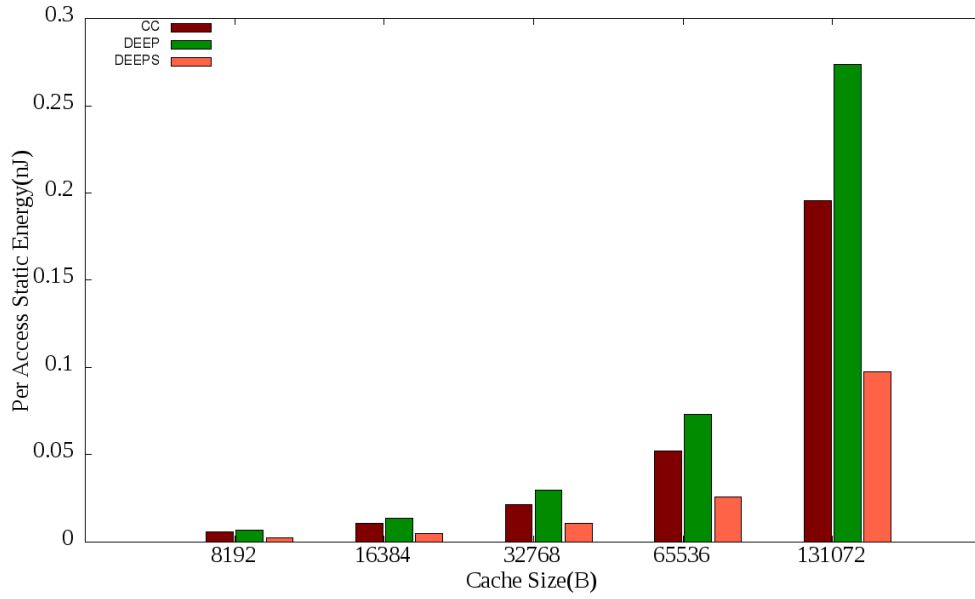


Figure 7.9: Per access static energy for a 4 way, 32B line size with varying cache size [#preemptions = 10]

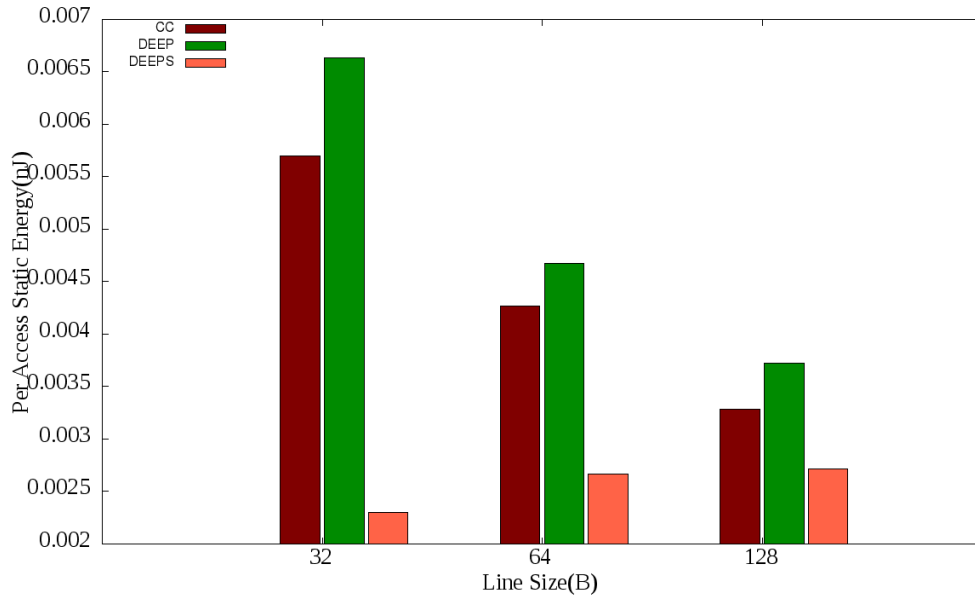


Figure 7.10: Per access static energy for a 4 way, 8KB cache with varying line size [#preemptions = 10]

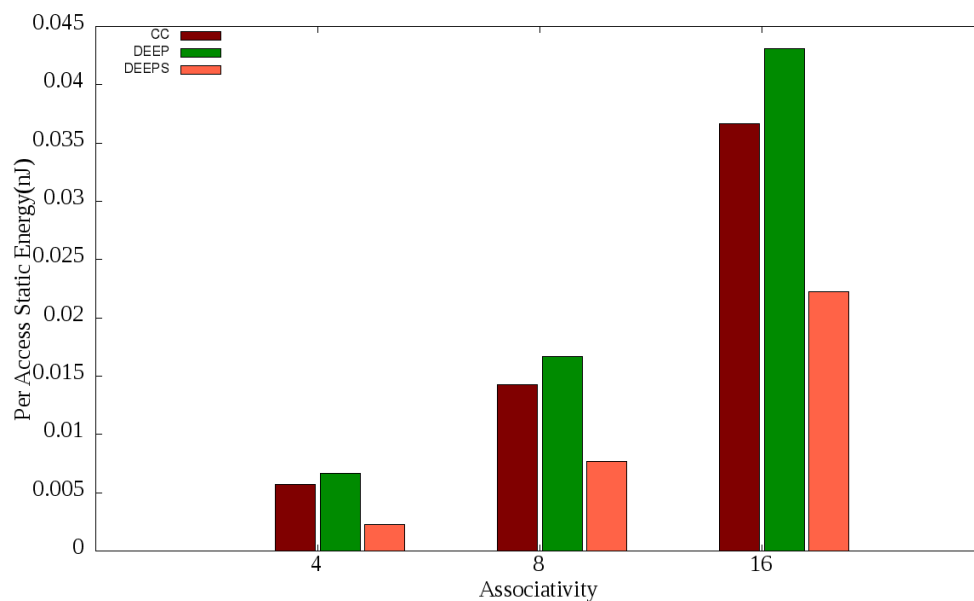


Figure 7.11: Per access static energy for a 32B, 8KB cache with varying associativity [#preemptions = 10]

7.4.1.3 Effective per Access Time

Per access time with varying number of preemptions, cache size, cache line size and associativity is shown in figures 7.12, 7.13, 7.14 and 7.15.

Per access time is directly proportional to the hit rate. The per access time of DEEP cache is higher than that of CC whenever the hit rate of CC is higher than that of DEEP cache. DEEPS cache offers the highest hit rate in comparison with CC and DEEP cache. On an average, per access time of DEEP cache is higher than CC by 10.48% and per access time of CC cache is higher than DEEPS by 38.50%.

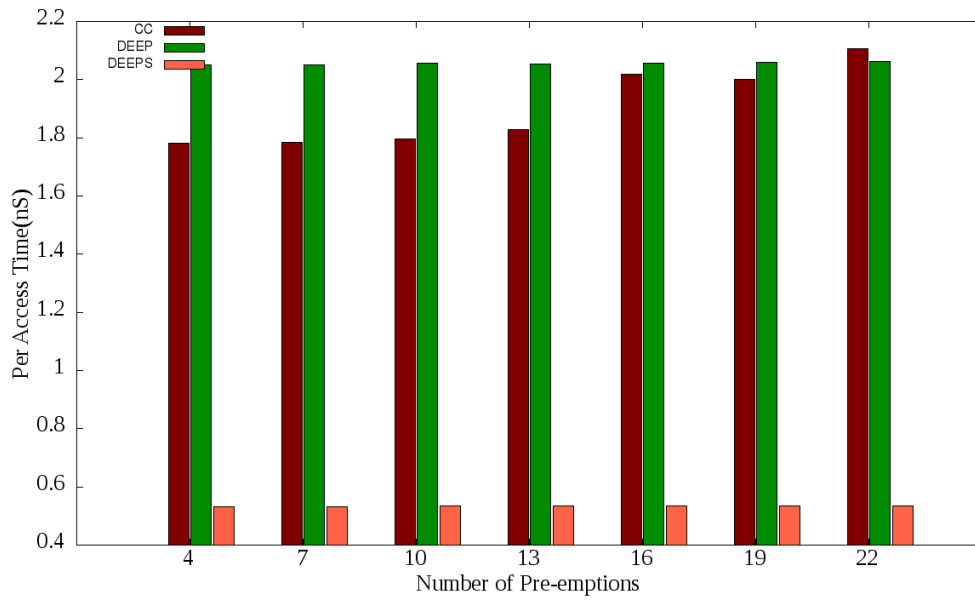


Figure 7.12: Per access time for a 4 way, 32B, 8KB cache with varying preemptions

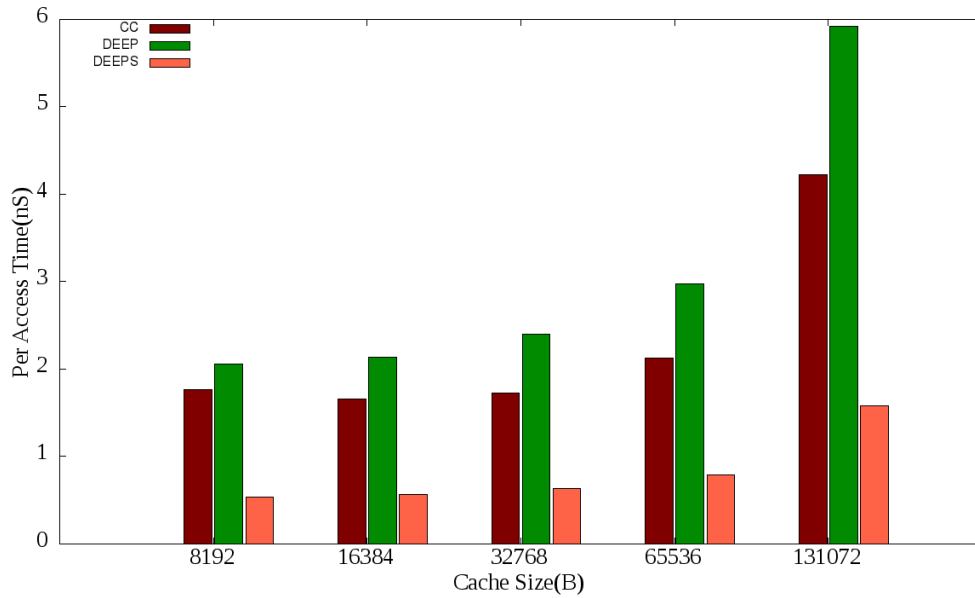


Figure 7.13: Per access time for a 4 way, 32B line size with varying cache size [#preemptions = 10]

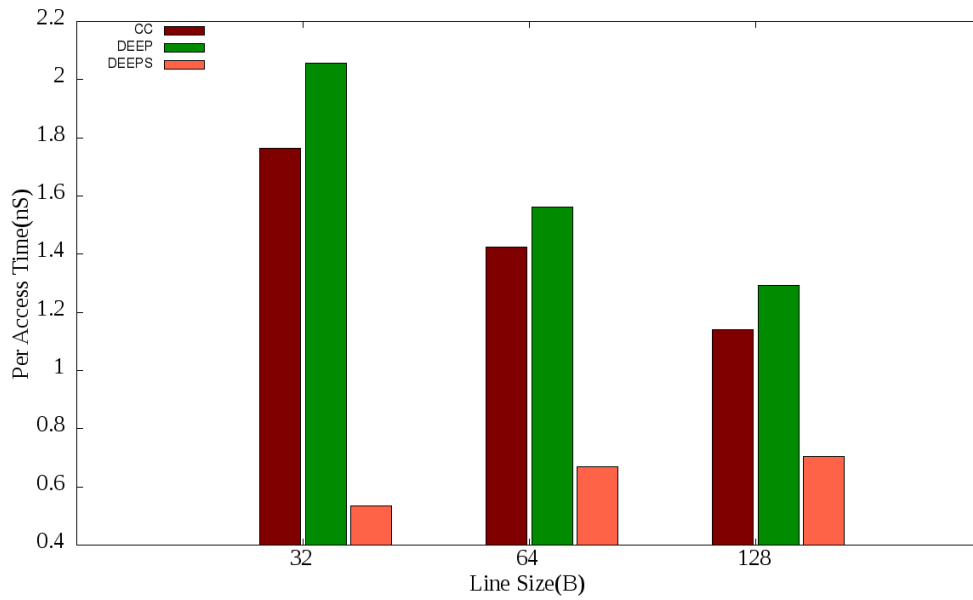


Figure 7.14: Per access time for a 4 way, 8KB cache with varying line size [#preemptions = 10]

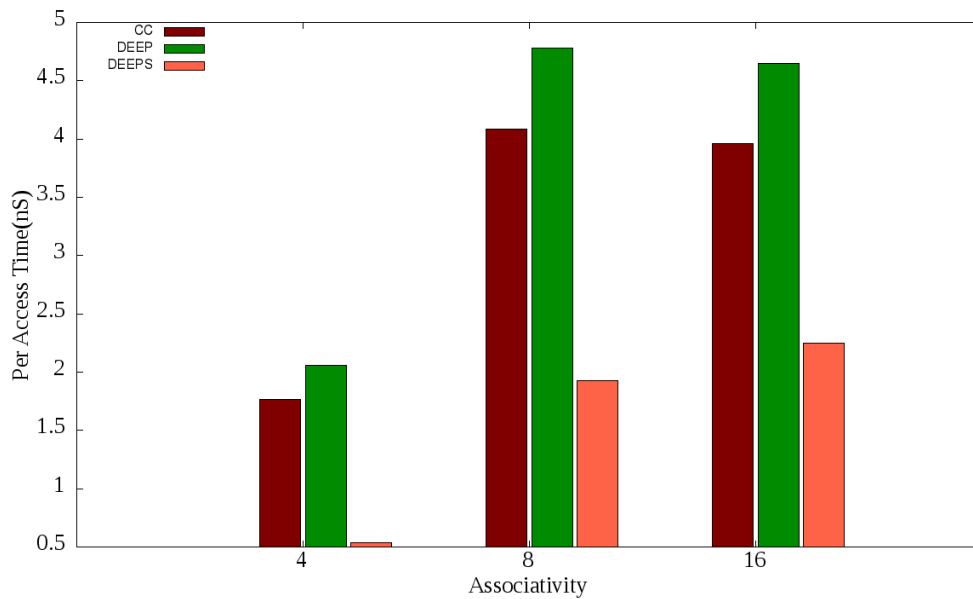


Figure 7.15: Per access time for a 32B, 8KB cache with varying associativity [#preemptions = 10]

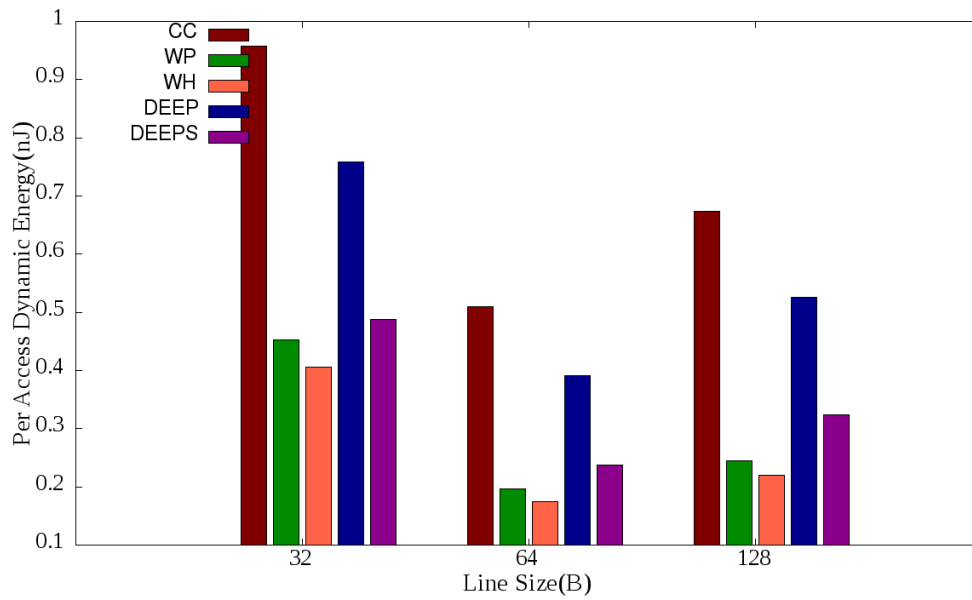


Figure 7.16: Per access dynamic energy for varying line size with 10 preemptions, 8KB cache size and associativity as 4 way

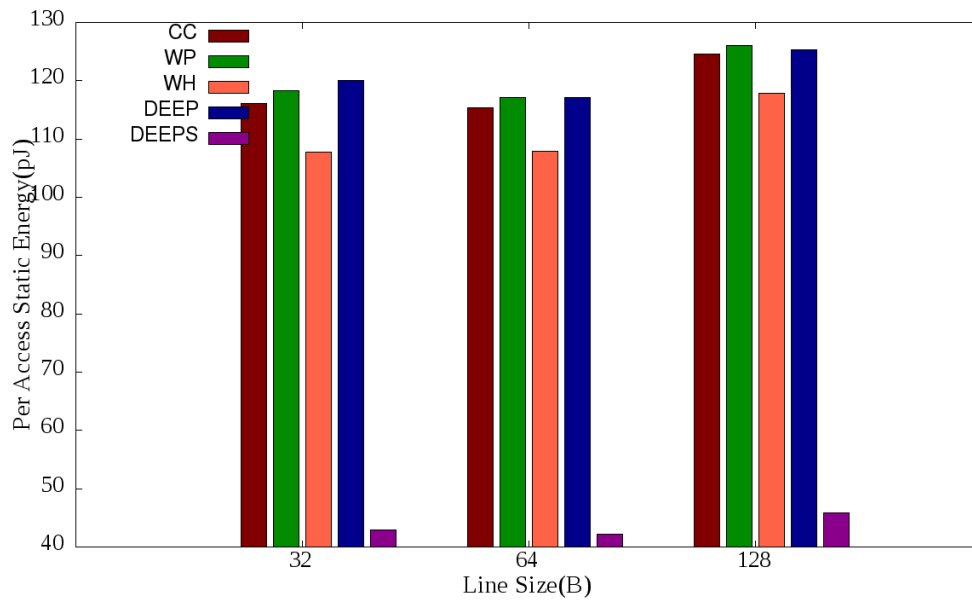


Figure 7.17: Per access Static energy for varying line size with 10 preemptions, 8KB cache size and associativity as 4 way

7.4.1.4 Energy and Time comparison with energy efficient caches

The comparison of dynamic and static energy consumption of DEEP with CC, WP, WH and DEEPS is as shown in Figures 7.16 and 7.17 respectively. Hit rate of CC, WP and WH is the same and is higher than DEEP. The hit rate of DEEP cache is improved by using shared cache in DEEPS. CC accesses all the ways for every access. The number of ways accessed increases dynamic energy consumption of CC. WH cache consumes the least dynamic energy. The number of ways accessed by WH cache is lesser than other caches. The prediction miss in WP cache results in accessing all other ways. This increases dynamic energy consumption of WP over WH cache. Identification based reservation in DEEP reduces associativity per task. The reduced associativity decreases hit rate. The decreased hit rate increases dynamic energy consumption of DEEP over WP and WH cache. Hit rate improvement in case of DEEPS cache results in reducing dynamic energy.

The improved hit rate in DEEPS cache results in consuming the least static energy. The prediction circuit incurs additional static energy in case of WP. Early detection of cache misses in case of WH results in reduced static energy consumption in comparison with CC. The least hit rate of DEEP along with additional DEEP overhead results in increases static energy consumption.

DEEP can be made more energy efficient by using prediction circuit and halt tag bits along with process identification. This will reduce the dynamic energy consumption. The static energy consumption and response time of DEEP is improved by using shared way. Data shared between task is stored in shared way. This results in improving DEEP hit rate and hence energy consumption.

7.4.2 Experimental Analysis - Complete Memory Model

Simulation results are analyzed by varying L1 cache size from 8K to 32K, L2 cache size from 32K to 128K. Results are obtained using following configurations :

CC : L1 and L2 cache as CC.

CD : L1 cache as CC and L2 cache as DEEPS.

DC : L1 cache as DEARCACHE and L2 cache as CC.

DD : L1 cache as DEARCACHE and L2 cache as DEEPS.

7.4.2.1 Dynamic Energy per Access

Figure 7.18 show per access dynamic energy consumption of various cache models simulated with 8-way, 64B, 64KB L2 cache, with varying L1 cache size. Figure 7.19 show per access dynamic energy consumption of various cache models simulated with 4-way, 32B, 8KB L1 cache, with varying L2 cache size. With increase in cache size, hit rate and per access dynamic energy consumption increases. This increase in per access energy is majorly due to increase in data access energy with cache size. Per access dynamic energy is highest when both L2 cache and L1 cache are implemented as CC.

Per access dynamic energy of memory model decreases when L2 cache is implemented as DEEP cache and L1 cache as CC. This decrease in per access dynamic energy is due to reduction in number of active ways for L2 cache from 8 ways to one way and increase in hit rate by and 5.33%. Further reduction in per access dynamic energy consumption can be achieved when L1

cache is implemented as DEARCACHE and L2 cache as CC. This decrease is due to reduction in number of ways from 4 ways to 1.02 ways and increase in hit rate by 9.68%. Per access dynamic energy is the least when L1 cache is implemented as DEARCACHE and L2 cache is implemented as DEEP cache respectively.

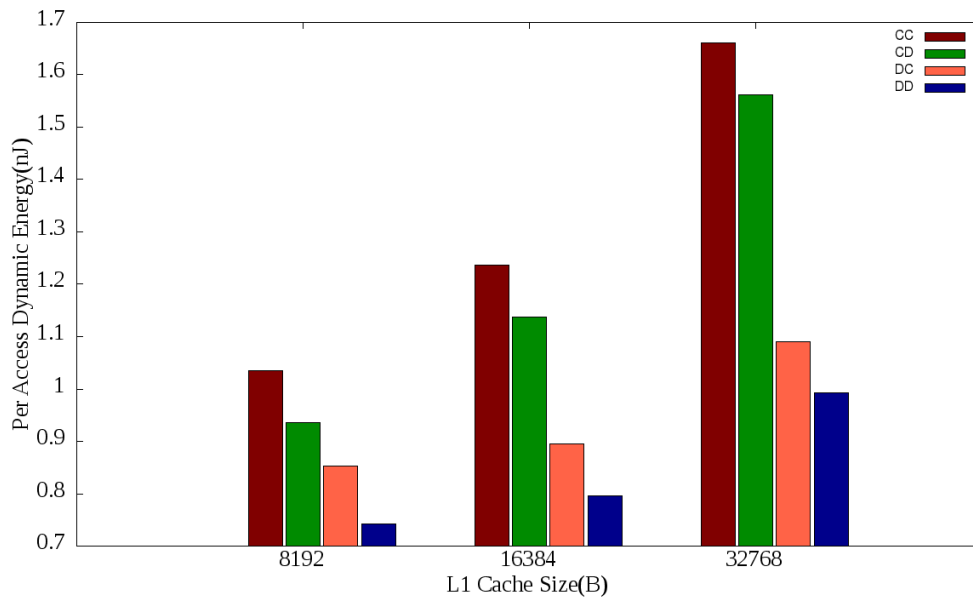


Figure 7.18: Per access dynamic energy for a 64KB L2 cache with varying L1 cache size

7.4.2.2 Static Energy per Access

Figure 7.20 and 7.21 show per access static energy dissipation of various memory models by varying L1 cache size and L2 cache size respectively. The static energy dissipation because of the internal components increases with increase in cache size. So the per access static energy increases with increase in cache size.

Per access static energy dissipation is the highest when both L1 cache and

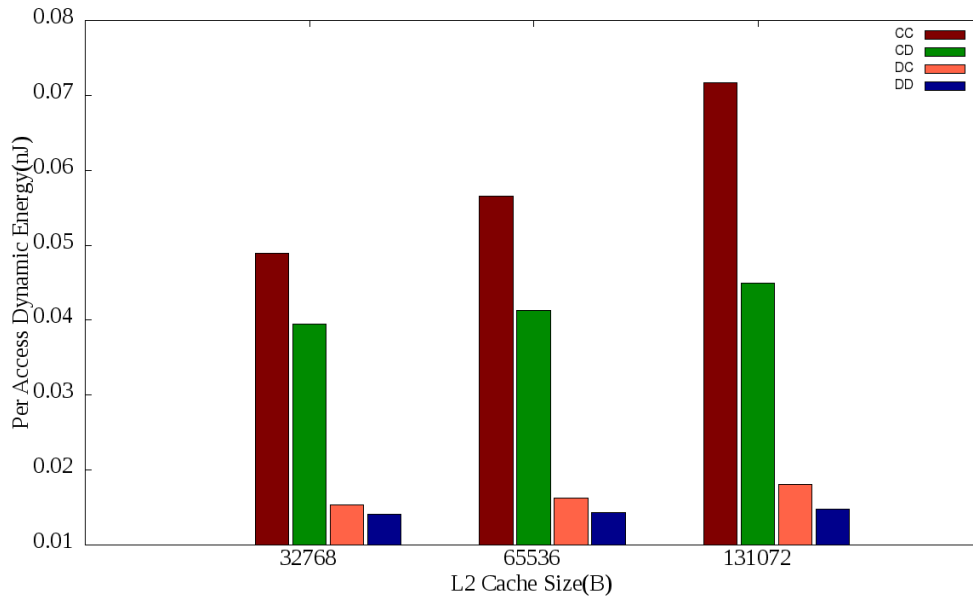


Figure 7.19: Per access dynamic energy for a 8KB L1 cache with varying L2 cache size

L2 cache are implemented as CC. When L1 cache and L2 cache is designed as DEARCACHE and DEEP cache with shared way, hit rate is improved by 9.68% and 5.33% over its respective CC implementations. Increase in hit rate decreases per access time and hence per access static energy dissipation. The per access static energy is the least when L1 cache and L2 cache are designed as DEARCACHE and DEEP cache respectively.

7.4.2.3 Per Access Time

Figure 7.22 and 7.23 show per access time of various cache models simulated with 8-way, 64KB, 64B L2 cache, varying L1 cache size and with 4-way, 8KB, 32B L1 cache, varying L2 cache size respectively. With increase in cache size, hit rate increases, and per access time decreases. Per access time of memory model decreases when L2 cache is implemented as DEEP cache and

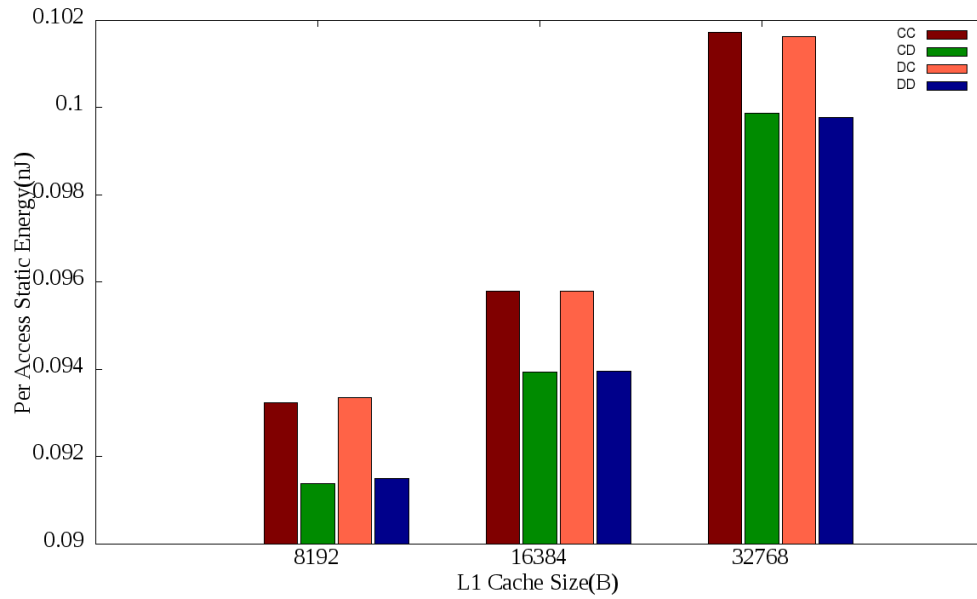


Figure 7.20: Per access static energy for a 64KB L2 cache with varying L1 cache size

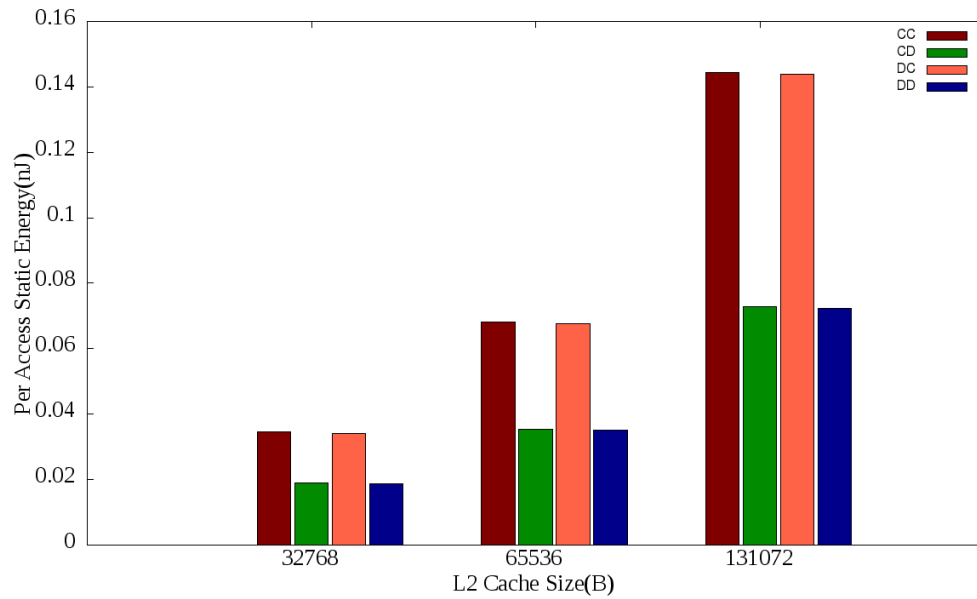


Figure 7.21: Per access static energy for a 8KB L1 cache with varying L2 cache size

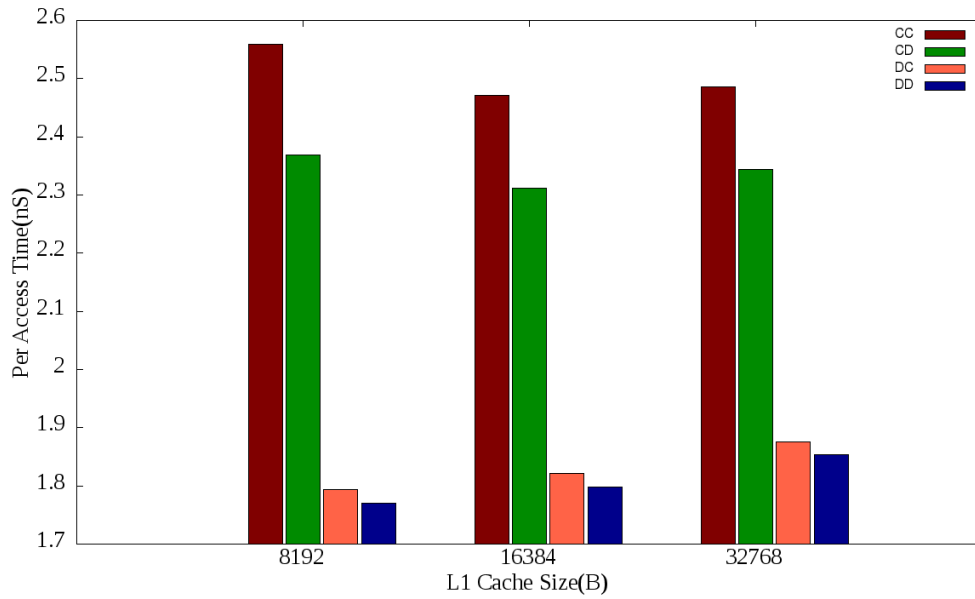


Figure 7.22: Per access time for a 64KB L2 cache with varying L1 cache size

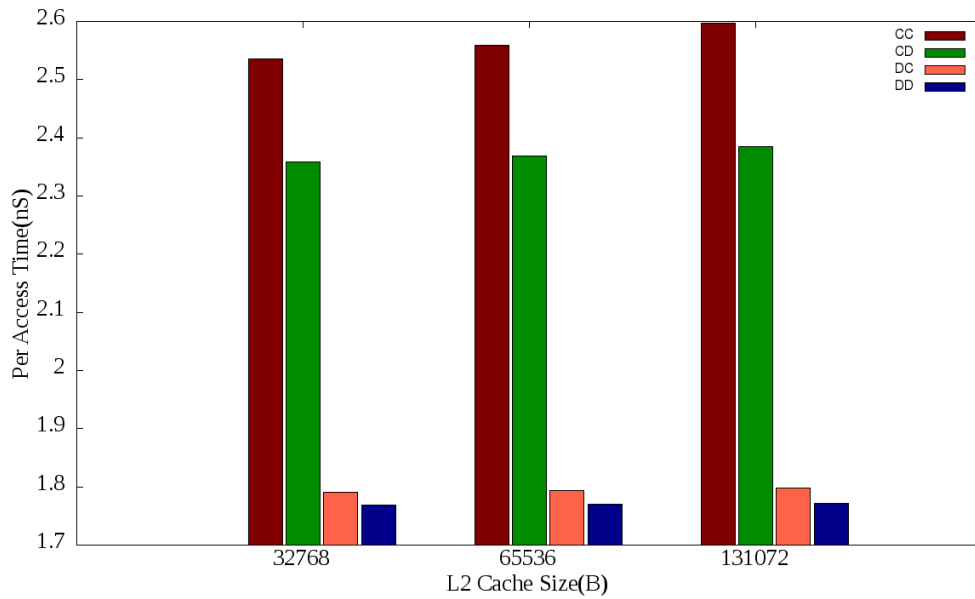


Figure 7.23: Per access time for an 8KB L1 cache with varying L2 cache size

L1 cache as CC. This decrease in per access time is due to increase in hit rate by 5.33% over L2 cache as CC. Further reduction in per access time can be achieved when L1 cache is implemented as DEARCACHE with increase in hit rate by 9.68%. Per access time is the least when L1 cache is implemented as DEARCACHE and L2 cache is implemented as DEEP cache respectively.

7.5 Conclusion

Tighter schedulability analysis is crucial in hard real-time systems as deadline misses may lead to catastrophic failure. The tighter upper bound on WCET at all levels of memory subsystem from TLB, L1 cache, L2 cache and main memory is required to ensure scheduling feasibility. This chapter designs memory subsystem to get deterministic performance at all levels. This chapter proposed deterministic memory subsystem by making TLB, L1 cache and L2 cache deterministic. This chapter combines implementation of DTLB and DEARCACHE with DEEP cache. The DTLB, DEARCACHE and DEEP cache gives tighter upper bound on misses of TLB, L1 cache and L2 cache respectively and hence gives tighter upper bound on cache memory related execution time. On an average per access time, per access dynamic energy and per access static energy of DEEP is higher than CC by 10.48%, 2.13%, 15.78% respectively. The per access time, per access dynamic energy and per access static energy of CC is higher than DEEPS by 38.50%, 71.69% and 50.01% respectively. The per access time, per access dynamic energy and per access static energy of CC is higher than DD by 27.85%, 71.40% and 46.75% respectively.

Chapter 8

Conclusion and Future

Directions

Energy efficiency is one of the major design considerations of the modern day processors. Memory subsystem consumes major portion of the on-chip energy. Architects are motivated to design the cache memory subsystem with the least possible energy consumption without much performance degradation. This thesis addresses static and dynamic energy consumptions of uncore and multicore systems. The thesis also addresses mechanisms to provide tighter upper bound on worst case execution time on memory sub-system performance in order to achieve deterministic memory performance.

This thesis proposed an energy efficient cache architecture - Way Halted Prediction. WHP cache uses halt tag array and prediction circuit to achieve reduced energy consumption and response time. The combination of halt tag and prediction circuit reduces the number of ways to be activated for cache access. In WHP cache, the number of active ways are reduced from k ways in case of WH to one way with the help of prediction circuit. As the prediction circuit is enabled only when $k > 1$, the performance of WHP cache is improved with respect to energy and time. The simulation results show that WHP offers better energy efficiency over the other architectures

analyzed. WHP offers 46.64%, 6.45% and 4.15% dynamic energy saving and 1.04%, 2.92% and -0.05% saving in response time over the CC, WP and WH respectively.

Cache coherence protocols achieve data consistency and coherency at the cost of performance degradation with respect to time and energy. The additional overhead can be minimized by optimizing the usage of interconnection bandwidth. This thesis discussed MOESIF protocol which improves the off chip bandwidth by reducing write backs to next level memory and the on-chip bandwidth by reducing the number of responders to a cache miss when multiple copies of data exists in private L1 caches of various cores. For varying cache sizes, energy consumption in MESI, MOESI, MESIF and MOESIF protocols is reduced by 51.41%, 94.20%, 51.66% and 94.49% respectively over MI protocol. The energy savings of MOESIF protocol over MESI, MOESI and MESIF protocol is 88.58%, 4.33% and 88.52% respectively. For varying cache sizes, per access time of MESI, MOESI, MESIF and MOESIF protocols is reduced by 52.04%, 95.59%, 52.31% and 95.86% respectively over MI. The per access time saving of MOESIF protocol over MESI, MOESI and MESIF protocol is 91.37%, 6.17% and 91.32% respectively.

Meeting all deadlines with least energy consumption is the major design consideration of battery powered hard real-time systems. Real-time scheduling algorithms perform schedulability analysis by considering the pessimistic WCET. In conventional systems, the WCET is loosely bound because of the non-deterministic nature of the memory subsystem. Cache memory is an integral and crucial part of the memory subsystem. This thesis proposed DEARCACHE, a deterministic real-time cache memory. DEARCACHE elim-

inates intertask interference by allocating dedicated cache ways to tasks in execution. It obtains tighter upper bound on number of cache misses. DEARCACHE reduces dynamic energy consumption by 38.10% for 4-way set associative cache configuration over CC with 4.39% overhead of static energy. Per access time of DEARCACHE improved 3.87 times over NO CACHE model with an additional requirement of 4.37% of per access time as compared to CC.

Tighter schedulability analysis is crucial in hard real-time systems as deadline misses may lead to catastrophic failure. The tighter upper bound on WCET at all levels of memory subsystem from TLB, L1 cache, L2 cache and main memory is required to ensure scheduling feasibility. This thesis designs memory subsystem to get deterministic performance at all levels. This thesis proposed deterministic memory subsystem by making TLB, L1 cache and L2 cache deterministic. This work combines implementation of DTLB and DEARCACHE with DEEP cache. The DTLB, DEARCACHE and DEEP cache gives tighter upper bound on misses of TLB, L1 cache and L2 cache respectively and hence gives tighter upper bound on cache memory related execution time. On an average per access time, per access dynamic energy and per access static energy of DEEP is higher than CC by 10.48%, 2.13%, 15.78% respectively. The per access time, per access dynamic energy and per access static energy of CC is higher than DEEPS by 38.50%, 71.69% and 50.01% respectively. The per access time, per access dynamic energy and per access static energy of CC is higher than DD by 27.85%, 71.40% and 46.75% respectively.

This thesis addresses deterministic cache designs for homogeneous multi-core

systems. We intend to extent this work for heterogeneous MC systems with point to point interconnections. We also intent to address deterministic performance for homogeneous many-core systems where switched-fabric is used as the interconnection mechanism. We also intend to address deterministic GPU performance with tighter upper bound on local, global and shared memory access times.

Publications based on the research work

Paper A: Geeta Patil, Parag Panda and Biju Raveendran, “A Survey on Replacement Strategies in Cache Memory for Embedded Systems,” IEEE Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), Mangalore, 2016, pp. 12-17.

Paper B: Geeta Patil, Neethu Bal Mallya and Biju Raveendran, “Way Halted Prediction Cache: An Energy Efficient Cache Architecture for Embedded Processors,” 28th International Conference on VLSI Design, Bangalore, 2015, pp. 65-70

Paper C: Geeta Patil, Neethu Bal Mallya and Biju Raveendran, “Simulation based Performance Study of Cache Coherence Protocols,” IEEE International Symposium on Nanoelectronic and Information Systems, Indore, 2015, pp. 125-130.

Paper D: Geeta Patil, Neethu Bal Mallya and Biju Raveendran, “MOESIF: A MC/MP Cache Coherence Protocol with Improved Bandwidth Utilization,” International Journal of Embedded Systems (In Press).

Paper E: Geeta Patil, Kajal Varma and Biju Raveendran, “DTLB: Deterministic TLB for Tightly Bound Hard Real-Time Systems,” 30th International Conference on VLSI Design and 16th International Conference on Embedded Systems, Hyderabad, 2017, pp. 207-212.

A brief biography of the candidate

Ms. Geeta Patil, is currently working as a Lecturer in the Department of Computer Science and Information Systems, BITS PILANI K. K. BIRLA GOA CAMPUS, GOA, INDIA. She received her Bachelor's degree in Computer Engineering from Goa Engineering College, Goa University in the year 2001. She did Master's degree in Computer Engineering in the year 2010 from Gogte Institute of Technology, Visvesvaraya Technological University, Belgaum. She is currently pursuing Ph.D. from BITS PILANI K. K. BIRLA GOA CAMPUS, GOA. Her research interests are in areas of Cache Architecture, Multi-core / Many-core systems, Multi-processors and Real time systems.

A brief biography of the supervisor

Dr. Biju K. Raveendran is currently serving as Assistant Professor in the Department of Computer Science and Information Systems, BITS PILANI K. K. BIRLA GOA CAMPUS, GOA, INDIA. He received his Ph.D. degree from BITS PILANI, PILANI CAMPUS, RAJASTHAN in the year 2009. He heads the Computer Centre Unit at Goa campus which is responsible for the central networking and computing facilities of the campus. His research area includes Energy Efficient Multi-core / Many-core Real-time Scheduling, Energy Efficient Memory Architecture for Multi-core / Many-core Embedded Systems, Predictable and Dependable Real-time / Embedded System Design and Big Data Systems. He was one of the five recipients of Microsoft Research India Fellowship in the year 2005 for his Ph.D. work. His passion is teaching.

He is a recipient of Microsoft young faculty award in the year 2009. He is also a recipient of Best Faculty Award by BITSAA in the year 2013. He is actively involved in collaborative projects with industries like Microsoft and Aditya Birla Group.

A brief biography of the co-supervisor

Dr. Lucy J. Gudino is Assistant Professor in the Department of Computer Science and Information Systems at BITS-Pilani, K K Birla Goa campus, Goa, India. She received B.E. degree in Electronics and Communications Engineering from Kuvempu University, M.Tech. and Ph.D. in Computer Science from Vishveswaraya Technological University, Karnataka, India. She has 21 years of teaching experience in the field of Electronics and Communications and Computer Science. Her research interests are computer architecture, wireless communications, adaptive arrays for cellular base stations and digital filter design.

Bibliography

- [1] C.-K. Luk, S. Hong, and H. Kim, “Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping,” in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, (New York, NY, USA), pp. 45–55, ACM, 2009.
- [2] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 3 ed., 2003.
- [3] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The splash-2 programs: Characterization and methodological considerations,” in *Proceedings of the 22Nd Annual International Symposium on Computer Architecture*, ISCA '95, (New York, NY, USA), pp. 24–36, ACM, 1995.
- [4] K. Inoue, T. Ishihara, and K. Murakami, “Way-predicting set-associative cache for high performance and low energy consumption,” in *Proceedings of the 1999 international symposium on Low power electronics and design*, pp. 273–275, ACM, 1999.
- [5] C. Zhang, F. Vahid, J. Yang, and W. Najjar, “A way-halting cache for low-energy high-performance systems,” *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 2, no. 1, pp. 34–54, 2005.
- [6] M. H. L. John Paul Shen, *Modern Processor Design*. New Delhi, India: Tata McGraw Hill, 3 ed., 2011.

-
- [7] D. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.
- [8] C. Ferdinand, “Worst case execution time prediction by static program analysis,” in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, pp. 125–, April 2004.
- [9] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, “High performance cache replacement using re-reference interval prediction (rrip),” *SIGARCH Comput. Archit. News*, vol. 38, pp. 60–71, June 2010.
- [10] J. Jeong and M. Dubois, “Optimal replacements in caches with two miss costs,” in *Proceedings of the Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '99*, (New York, NY, USA), pp. 155–164, ACM, 1999.
- [11] H. Al-Zoubi, A. Milenkovic, and M. Milenkovic, “Performance evaluation of cache replacement policies for the spec cpu2000 benchmark suite,” in *Proceedings of the 42Nd Annual Southeast Regional Conference, ACM-SE 42*, (New York, NY, USA), pp. 267–272, ACM, 2004.
- [12] K. So and R. N. Rechtschaffen, “Cache operations by mru change,” *IEEE Transactions on Computers*, vol. 37, pp. 700–709, Jun 1988.
- [13] J. Alghazo, A. Akaaboune, and N. Botros, “Sf-lru cache replacement algorithm,” in *Records of the 2004 International Workshop on Memory Technology, Design and Testing, 2004.*, pp. 19–24, Aug 2004.

-
- [14] C. T. Do, H.-J. Choi, J. M. Kim, and C. H. Kim, “A new cache replacement algorithm for last-level caches by exploiting tag-distance correlation of cache lines,” *Microprocess. Microsyst.*, vol. 39, pp. 286–295, June 2015.
- [15] M. Kampe, P. Stenstrom, and M. Dubois, “Self-correcting lru replacement policies,” in *IN PROCEEDINGS OF THE 1ST CONFERENCE ON COMPUTING FRONTIERS*, pp. 181–191, 2004.
- [16] “Intel Corp. 1997. Embedded Intel486 Processor Family Developer’s Manual Technical Report 273021-001,” tech. rep.
- [17] J. L. Henning, “Spec cpu2006 benchmark descriptions,” *SIGARCH Comput. Archit. News*, vol. 34, pp. 1–17, Sept. 2006.
- [18] T. Austin, E. Larson, and D. Ernst, “SimpleScalar: An infrastructure for computer system modeling,” *Computer*, vol. 35, pp. 59–67, Feb. 2002.
- [19] C.-L. Su and A. M. Despain, “Cache design trade-offs for power and performance optimization: A case study,” in *Proceedings of the 1995 International Symposium on Low Power Design, ISLPED ’95*, (New York, NY, USA), pp. 63–68, ACM, 1995.
- [20] L. Liu, Y. Li, Z. Cui, Y. Bao, M. Chen, and C. Wu, “Going vertical in memory management: Handling multiplicity by multi-policy,” in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 169–180, June 2014.
- [21] J. Kin, M. Gupta, and W. H. Mangione-Smith, “The filter cache: an energy efficient memory structure,” in *Proceedings of 30th Annual International Symposium on Microarchitecture*, pp. 184–193, Dec 1997.

-
- [22] R. S. Bajwa, M. Hiraki, H. Kojima, D. J. Gorny, K.-i. Nitta, A. Shridhar, K. Seki, and K. Sasaki, "Instruction buffering to reduce power in processors for signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 4, pp. 417–424, 1997.
- [23] V. Guzma, T. Pitknen, and J. Takala, "Reducing instruction memory energy consumption by using instruction buffer and after scheduling analysis," in *2010 International Symposium on System on Chip*, pp. 99–102, Sept 2010.
- [24] L. H. Lee, B. Moyer, and J. Arends, "Instruction fetch energy reduction using loop caches for embedded applications with small tight loops," in *Proceedings of the 1999 International Symposium on Low Power Electronics and Design, ISLPED '99*, (New York, NY, USA), pp. 267–269, ACM, 1999.
- [25] J. W. Kwak and Y. T. Jeon, "Compressed tag architecture for low-power embedded cache systems," *Journal of Systems Architecture*, vol. 56, no. 9, pp. 419 – 428, 2010.
- [26] T. M. Jones, S. Bartolini, B. D. Bus, J. Cavazos, and M. F. P. O'Boyle, "Instruction cache energy saving through compiler way-placement," in *2008 Design, Automation and Test in Europe*, pp. 1196–1201, March 2008.
- [27] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache for low energy embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 4, pp. 363–387, May 2005.

-
- [28] G. Bournoutian and A. Orailoglu, “Application-aware adaptive cache architecture for power-sensitive mobile processors,” *ACM Trans. Embed. Comput. Syst.*, vol. 13, pp. 41:1–41:26, Dec. 2013.
- [29] A. Sembrant, E. Hagersten, and D. Black-Shaffer, “Tlc: a tag-less cache for reducing dynamic first level cache energy,” in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 49–61, ACM, 2013.
- [30] R. K. Megalingam, K. B. Deepu, I. P. Joseph, and V. Vikram, “Phased set associative cache design for reduced power consumption,” *2009 2nd IEEE International Conference on Computer Science and Information Technology*, pp. 551–556, 2009.
- [31] Z. Zhu and X. Zhang, “Access-mode predictions for low-power cache design,” *IEEE Micro*, vol. 22, pp. 58–71, Mar 2002.
- [32] B. Batson and T. Vijaykumar, “Reactive-associative caches,” in *Parallel Architectures and Compilation Techniques, 2001. Proceedings. 2001 International Conference on*, pp. 49–60, IEEE, 2001.
- [33] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, “An evaluation of directory schemes for cache coherence,” *SIGARCH Comput. Archit. News*, vol. 16, pp. 280–298, May 1988.
- [34] R. Komuravelli, S. V. Adve, and C.-T. Chou, “Revisiting the complexity of hardware cache coherence and some implications,” *ACM Trans. Archit. Code Optim.*, vol. 11, pp. 37:1–37:22, Dec. 2014.

-
- [35] B. Choi, R. Komuravelli, H. Sung, R. Smolinski, N. Honarmand, S. V. Adve, V. S. Adve, N. P. Carter, and C.-T. Chou, “Denovo: Rethinking the memory hierarchy for disciplined parallelism,” in *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques, PACT '11*, (Washington, DC, USA), pp. 155–166, IEEE Computer Society, 2011.
- [36] J. K. Archibald, *The Cache Coherence Problem in Shared-memory Multiprocessors*. PhD thesis, Seattle, WA, USA, 1987. UMI Order No. GAX87-06505.
- [37] T. M. Chaves, E. A. Carara, and F. G. Moraes, “Energy-efficient cache coherence protocol for noc-based mpsoes,” in *Proceedings of the 24th Symposium on Integrated Circuits and Systems Design, SBCCI '11*, (New York, NY, USA), pp. 215–220, ACM, 2011.
- [38] P. Stenstrom, “A survey of cache coherence schemes for multiprocessors,” *Computer*, vol. 23, pp. 12–24, June 1990.
- [39] J. Archibald and J.-L. Baer, “Cache coherence protocols: Evaluation using a multiprocessor simulation model,” *ACM Trans. Comput. Syst.*, vol. 4, pp. 273–298, Sept. 1986.
- [40] M. Loghi, M. Poncino, and L. Benini, “Cache coherence tradeoffs in shared-memory mpsoes,” *ACM Trans. Embed. Comput. Syst.*, vol. 5, pp. 383–407, May 2006.

-
- [41] N. P. Jouppi, “Cache write policies and performance,” in *Proceedings of the 20th Annual International Symposium on Computer Architecture, ISCA '93*, (New York, NY, USA), pp. 191–201, ACM, 1993.
- [42] G. Bournoutian and A. Orailoglu, “Dynamic, multi-core cache coherence architecture for power-sensitive mobile processors,” in *2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 89–97, Oct 2011.
- [43] A. Kayi and T. El-Ghazawi, “An adaptive cache coherence protocol for chip multiprocessors,” in *Proceedings of the Second International Forum on Next-Generation Multicore/Manycore Technologies, IFMT '10*, (New York, NY, USA), pp. 4:1–4:10, ACM, 2010.
- [44] S. Manne, A. Klauser, D. C. Grunwald, and F. Somenzi, “Low power tlb design for high performance microprocessors,” 1997.
- [45] J.-H. Lee, G. ho Park, S.-B. Park, and S.-D. Kim, “A selective filter-bank tlb system [embedded processor mmu for low power],” in *Low Power Electronics and Design, 2003. ISLPED '03. Proceedings of the 2003 International Symposium on*, pp. 312–317, Aug 2003.
- [46] M. Talluri and M. D. Hill, “Surpassing the tlb performance of superpages with less operating system support,” *SIGPLAN Not.*, vol. 29, pp. 171–182, Nov. 1994.
- [47] I. Chukhman and P. Petrov, “Context-aware tlb preloading for interference reduction in embedded multi-tasked systems,” in *Proceedings of*

- the 20th Symposium on Great Lakes Symposium on VLSI, GLSVLSI '10*, (New York, NY, USA), pp. 401–404, ACM, 2010.
- [48] A. Saulsbury, F. Dahlgren, and P. Stenström, “Recency-based tlb preloading,” in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, (New York, NY, USA), pp. 117–127, ACM, 2000.
- [49] M. Kandemir, I. Kadayif, and G. Chen, “Compiler-directed code restructuring for reducing data tlb energy,” in *Proceedings of the 2Nd IEEE/ACM/IFIP International Conference on Hardware/Software Code-sign and System Synthesis*, CODES+ISSS '04, (New York, NY, USA), pp. 98–103, ACM, 2004.
- [50] G. Venkatasubramanian, R. J. Figueiredo, and R. Illikkal, “On the performance of tagged translation lookaside buffers: A simulation-driven analysis,” in *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 139–149, July 2011.
- [51] Y. Li, R. Melhem, and A. K. Jones, “Ps-tlb: Leveraging page classification information for fast, scalable and efficient translation for future cmps,” *ACM Trans. Archit. Code Optim.*, vol. 9, pp. 28:1–28:21, Jan. 2013.
- [52] R. J. Bril, J. J. Lukkien, and W. F. Verhaegh, “Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption,” *Real-Time Systems*, vol. 42, no. 1-3, pp. 63–119, 2009.

-
- [53] S. Altmeyer, R. I. Davis, and C. Maiza, “Pre-emption cost aware response time analysis for fixed priority pre-emptive systems,” *32nd RTSS*, 2011.
- [54] X. Lin, Y. Wang, Q. Xie, and M. Pedram, “Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment,” *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 175–186, 2015.
- [55] A. P. Florence, V. Shanthi, and C. Simon, “Energy conservation using dynamic voltage frequency scaling for computational cloud,” *The Scientific World Journal*, vol. 2016, 2016.
- [56] V. Legout, M. Jan, and L. Pautet, “A scheduling algorithm to reduce the static energy consumption of multiprocessor real-time systems,” in *Proceedings of the 21st International Conference on Real-Time Networks and Systems, RTNS '13*, (New York, NY, USA), pp. 99–108, ACM, 2013.
- [57] R. I. Davis and A. Burns, “A survey of hard real-time scheduling for multiprocessor systems,” *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.
- [58] F. Zhang and A. Burns, “Schedulability analysis for real-time systems with edf scheduling,” *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1250–1258, 2009.
- [59] L. Ju, S. Chakraborty, and A. Roychoudhury, “Accounting for cache-related preemption delay in dynamic priority schedulability analysis,” in *2007 Design, Automation & Test in Europe Conference & Exhibition*, pp. 1–6, IEEE, 2007.

-
- [60] R. I. Davis, S. Altmeyer, and J. Reineke, “Analysis of write-back caches under fixed-priority preemptive and non-preemptive scheduling,” in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pp. 309–318, ACM, 2016.
- [61] W.-N. Chin, H. H. Nguyen, C. Popeea, and S. Qin, “Analysing memory resource bounds for low-level programs,” in *Proceedings of the 7th international symposium on Memory management*, pp. 151–160, ACM, 2008.
- [62] J. Whitham, N. C. Audsley, and R. I. Davis, “Explicit reservation of cache memory in a predictable, preemptive multitasking real-time system,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4s, p. 120, 2014.
- [63] J. Chang and G. S. Sohi, “Cooperative cache partitioning for chip multiprocessors,” in *ACM International Conference on Supercomputing 25th Anniversary Volume*, pp. 402–412, ACM, 2014.
- [64] H. Falk, S. Plazar, and H. Theiling, “Compile-time decided instruction cache locking using worst-case execution paths,” in *2007 5th IEEE/ACM/IFIP International Conference on Hardware/Software Code-sign and System Synthesis (CODES+ISSS)*, pp. 143–148, Sept 2007.
- [65] A. Arnaud and I. Puaut, “Dynamic instruction cache locking in hard real-time systems,” in *In RTNS*.
- [66] P. M. Ortego and P. Sack, “Sesc: Superescalar simulator,” tech. rep., 2004.

-
- [67] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, “Cacti 6.0: A tool to model large caches,” *HP Laboratories*, pp. 22–31, 2009.
- [68] B. Pham, V. Vaidyanathan, A. Jaleel, and A. Bhattacharjee, “Colt: Coalesced large-reach tlbs,” in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, (Washington, DC, USA), pp. 258–269, IEEE Computer Society, 2012.
- [69] Y.-J. Chang, “An ultra low-power tlb design,” in *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings*, DATE '06, (3001 Leuven, Belgium, Belgium), pp. 1122–1127, European Design and Automation Association, 2006.
- [70] J. Whitham, N. C. Audsley, and R. I. Davis, “Explicit reservation of cache memory in a predictable, preemptive multitasking real-time system,” *ACM Trans. Embed. Comput. Syst.*, vol. 13, pp. 120:1–120:25, Apr. 2014.
- [71] J. Feljan, *Task Allocation Optimization for Multicore Embedded Systems*. PhD thesis, Mälardalen University, December 2015.
- [72] A. Thekkilakattil, *Limited Preemptive Scheduling in Real-time Systems*. PhD thesis, Mälardalen University, May 2016. The faculty examiner is Associate Professor Reinder Bril, Eindhoven University of Technology; and the examining committee consists of Professor Giorgio Buttazzo, SantAnna School of Advance studies-Pisa; Professor Gerhard Fohler, Technical University Kaiserslautern; Associate Professor Liliana Cucu-Grosjean, INRIA.Reserve: Associate Professor Damir Isovich, MDH.

-
- [73] M. K. Qureshi, D. Thompson, and Y. N. Patt, “The v-way cache: demand-based associativity via global replacement,” in *32nd International Symposium on Computer Architecture (ISCA’05)*, pp. 544–555, IEEE, 2005.