

# **A Study of Low Power and High Performance Cache Hierarchy for Multi-Core Processor**

by

**Geng Tian**

B.Engineering. (Electrical and Electronic),  
Xi'an Jiaotong University, 2006

M.Engineering. (Electrical and Electronic),  
University of Adelaide, 2009

Thesis submitted for the degree of

**Doctor of Philosophy**

in

Electrical and Electronic Engineering  
University of Adelaide

2015

© 2015  
Geng Tian  
All Rights Reserved



# Contents

<b>Contents</b>	<b>iii</b>
<b>Abstract</b>	<b>vii</b>
<b>Statement of Originality</b>	<b>ix</b>
<b>Acknowledgments</b>	<b>xi</b>
<b>Publications</b>	<b>xiii</b>
<b>List of Figures</b>	<b>xv</b>
<b>List of Tables</b>	<b>xxiii</b>
<b>Chapter 1. Introduction and Background</b>	<b>1</b>
1.1 Introduction and Motivation . . . . .	2
1.2 Background: Tiled CMP Architectures and Their Cache . . . . .	3
1.2.1 Tiled CMP Architecture . . . . .	3
1.2.2 Cache Design Background . . . . .	8
1.2.3 Non-Uniform Cache Architecture . . . . .	13
1.2.4 Power Dissipation Model and Simulation Method . . . . .	19
1.3 Related Work . . . . .	22
1.3.1 Power Saving Cache . . . . .	22
1.3.2 Novel Cache Replacement Policy . . . . .	24
1.3.3 Cache Access Latency Reduction . . . . .	26
1.4 Thesis overview: Significance and Contributions . . . . .	29
<b>Chapter 2. Utility Based Cache Resizing</b>	<b>31</b>
2.1 Introduction . . . . .	32
2.2 Overall Power Analyses . . . . .	33

- 2.3 Utility of the Cache Block . . . . . 34
- 2.4 Utility-Based Monitoring . . . . . 35
  - 2.4.1 Stack Distance . . . . . 35
  - 2.4.2 Performance vs Cache Resources . . . . . 36
- 2.5 Increasing Cache Algorithm . . . . . 38
  - 2.5.1 Rationale of the Size-up Approach . . . . . 38
  - 2.5.2 Concept of BEM . . . . . 40
  - 2.5.3 L1 Early Eviction . . . . . 44
  - 2.5.4 Evaluation Platform and Benchmarks . . . . . 49
  - 2.5.5 Up-Sizing Profile . . . . . 50
  - 2.5.6 Evaluation of Up Sizing Only Scheme . . . . . 51
- 2.6 Bi-Directional Tuning Scheme . . . . . 53
  - 2.6.1 Pre-reduction Preparation . . . . . 56
  - 2.6.2 Thrashing Protection . . . . . 58
  - 2.6.3 Down-sizing Evaluation . . . . . 59
- 2.7 Tile Level Evaluation . . . . . 63
- 2.8 Summary . . . . . 65

**Chapter 3. An Effectiveness-Based Adaptive Cache Replacement Policy 67**

- 3.1 Introduction . . . . . 68
- 3.2 Re-definition of Cache Optimal Replacement . . . . . 69
- 3.3 Study of Different Replacement Policy . . . . . 77
  - 3.3.1 Optimal Replacement Policy . . . . . 77
  - 3.3.2 Least Recent Used Replacement Policy (LRU): . . . . . 77
  - 3.3.3 Least Frequently Used Replacement Policy (LFU) : . . . . . 80
- 3.4 Effectiveness Based Replacement Policy (EBR): . . . . . 81
- 3.5 Evaluation Environment . . . . . 83
  - 3.5.1 Evaluation Platform . . . . . 83
  - 3.5.2 Benchmark . . . . . 84
- 3.6 Evaluation of EBR . . . . . 85
- 3.7 Dynamic EBR . . . . . 86

3.8	Evaluation of DEBR . . . . .	89
3.9	Sensitivity Study for Rank Granularity . . . . .	90
3.10	Hardware Implementation and Overhead . . . . .	98
3.10.1	Replacement Logic . . . . .	98
3.10.2	Storage Overhead . . . . .	99
3.11	Summary . . . . .	100
<b>Chapter 4. Cache Dynamic Resize using EBR based Cache Utility Evaluation</b>		<b>103</b>
4.1	Introduction . . . . .	104
4.2	Further Study of <i>LRU</i> Prediction . . . . .	105
4.2.1	Future Access Trace . . . . .	105
4.2.2	Rank Distance Estimation . . . . .	108
4.3	Algorithm of <i>EBR</i> based increasing only mechanism . . . . .	117
4.4	False Evicting Rank Reuse . . . . .	117
4.5	Deterministic Evaluation Analyse . . . . .	120
4.5.1	Non Deterministic Simulation . . . . .	120
4.5.2	Trace Driven Evaluation . . . . .	123
4.6	<i>LRU</i> -based vs <i>EBR</i> -based Increasing Scheme . . . . .	126
4.7	<i>EBR</i> -based Reducing Scheme . . . . .	137
4.8	<i>EBR</i> Reduced Evaluation and Comparison with <i>LRU</i> based Reducing . .	140
4.9	Hardware Overhead Analysis . . . . .	145
4.10	Dynamic <i>EBR</i> resizing . . . . .	150
4.11	Summary . . . . .	153
<b>Chapter 5. EBR based Dynamic Private and Shared Cache Partitioning</b>		<b>157</b>
5.1	Introduction and motivation . . . . .	158
5.2	Latency analysis . . . . .	159
5.3	Private/Shared Partition . . . . .	161
5.3.1	L1 Victim can be Useful . . . . .	162
5.3.2	Foundation of Partition . . . . .	162
5.3.3	Cache Query Process . . . . .	164

## Contents

---

5.3.4	L1 Cache Eviction . . . . .	165
5.3.5	Coherency Protocol . . . . .	165
5.3.6	Replacement Policy Partitioning . . . . .	166
5.4	Dynamic Partitioning . . . . .	167
5.5	Performance Evaluation . . . . .	170
5.5.1	Fixed Partition Evaluation . . . . .	170
5.5.2	Dynamic Partitioning Evaluation . . . . .	173
5.6	Dynamic Partition With Resize . . . . .	178
5.7	Summary . . . . .	179
<b>Chapter 6. Conclusion and Future Work</b>		<b>181</b>
6.1	Conclusion . . . . .	182
6.2	Future Work . . . . .	183
6.2.1	EBR Based Bypass . . . . .	184
6.2.2	Private Cache Evaluation . . . . .	184
6.2.3	Cache Resizing Without Set-Associativity . . . . .	184
<b>Bibliography</b>		<b>185</b>

# Abstract

The increasing levels of transistor density have enabled integration of an increasing number of cores and cache resources on a single chip. However, power, as a first order design constraint may bring this trend to a dead end. Recently, the primary design objective has been shifted from pursuing faster speed to higher power-performance efficiency. This is also reflected by the fact that design preference has transitioned from fast super-scalar architecture to slower multi-core architecture.

Tiled chip multiprocessors (CMPs) have shown unmatched advantages in recent years, and they are very likely to be the mainstream in the future. Meanwhile, increasing number of cores will exert higher pressure on the cache system. Expanding cache storage can ease the pressure but will incur higher static power consumption. More importantly, very large caches in future multi-core systems may not be fully utilised. Under-utilised caches consume static power for no productivity. Off-line profiling of applications to determine optimal cache size and configuration is not practical.

This thesis describes dynamic cache reallocation techniques for tiled multi core architectures. We proposed the idea of Break Even number of Misses (BEM). BEM defines, for a given cache configuration and time interval, the maximum number of misses that can be tolerated without increasing the energy delay product. We use BEM as the upper bound to determine a set of thresholds that are used to periodically evaluate the utility of cache. Based on this scheme, we then propose a conservative increase-only resizing method to tune the cache size at tile-level granularity. The increasing only method can be further extended to a dynamic downsizing scheme.

In simulations, our tuning scheme can reduce the static power significantly with a very minor degradation of IPC (Instruction Per Cycle). One thing that can be improved from our resizing scheme is the replacement policy. The estimation of cache utility is based on stack-distance which relies on the recency position of a real *LRU* (least recently used) replacement policy. As is commonly known, *LRU* is not easy to implement, especially in high associativity caches, which are becoming more commonly used. *LRU* also suffers from “cache thrashing”, “scan” and “inter-thread interference”. To solve

these three problems, we further propose a novel replacement policy, the Effectiveness-Based Replacement policy (*EBR*) and a refinement, Dynamic *EBR* (*D-EBR*), which combines measures of recency and frequency to form a rank sequence inside each set and evict blocks with lowest rank. To evaluate our design, we simulated all 30 applications from SPEC CPU2006 for uni-core system and a set of combinations for 4-core systems, for different cache sizes. The results show that both of them can achieve higher performance with only half the hardware overhead of real *LRU*. With the help of *EBR*, we further extend our last level cache resize scheme. We discuss how to estimate equivalent utility of cache using *EBR* replacement policy rather than *LRU*, and introduce an *EBR* based resizing scheme. Since *EBR* replacement policy is hardware economical and cache thrashing protected, it is more suitable for the utility estimation.

Finally, to shorten average cache access latency, we propose the idea of using a private replica region to store useful data replicas. Keeping them close to the requester can significantly reduce average access latency, however, it also reduces effective storage size causing higher cache miss rates. We leverage our *EBR* based cache utility estimation method to dynamically change the partition based on cache access patterns to achieve a near optimal result.

# Statement of Originality

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

---

Signed

---

Date



# Acknowledgments

First and foremost, I would like to thank my supervisors A.Prof Michael Liebelt and Dr Braden Phillips sincerely. Without their guidance and encouragement, I would not be able to finish my research. It is my honour to pursue Ph.D under their supervision. They have taught me not only how to do research in this particular area but also taught me how to divide and conquer the puzzle.

I also want to acknowledge my colleagues and academic members in the computer system research group for their help and advice. I would like to thank the staff of School of Electrical and Electronics Engineering at the University of Adelaide for their assistance and technical support.

I would like to acknowledge Microsoft for giving me a job before I finish my thesis which makes my thesis writing much easier.

Finally, I would like to express my gratitude to my wife Jessica for her encouragement, support and patience throughout my entire candidature.



# Publications

## Journal

---

Geng Tian, Michael Liebelt, (2014). An effectiveness-based adaptive cache replacement policy, *Microprocess. Microsyst.* 38, 1 (February 2014), 98-111. DOI=10.1016/j.micpro.2013.11.011 <http://dx.doi.org/10.1016/j.micpro.2013.11.011> . Published.

## Conference

---

Geng Tian, Michael Liebelt, (2011). Utility-based dynamic cache resizing, *Computer Science and Network Technology (ICCSNT), 2011 International Conference on* , vol.1, no., pp.610,616, 24-26 Dec. 2011 . Published.



# List of Figures

1.1	Two commonly used multi core approaches . . . . .	5
1.2	Topology:ring . . . . .	6
1.3	Topology:symmetric . . . . .	6
1.4	Topology:bus . . . . .	7
1.5	Topology:crossbar . . . . .	7
1.6	Topology:2D mesh,packet-switched NoCs . . . . .	8
1.7	Base system of our dynamic cache design . . . . .	9
1.8	Computer memory hierarchy . . . . .	10
1.9	Different ways of mapping data from lower level memory to higher level	11
1.10	Example of cache incoherency . . . . .	12
1.11	Comparison between UCA with one bank and 64 cycle average access and NUCA with 16 banks . . . . .	14
1.12	Layout of private cache and shared cache . . . . .	15
1.13	Break down cache mapping on physical address . . . . .	18
1.14	Block diagram of the McPAT framework (Li <i>et al.</i> 2009) . . . . .	21
<hr/>		
2.1	Overall power breakdown . . . . .	34
2.2	The minimal cache size (represented by number of ways) to achieve 95% 99% and 99.9% of original performance during the runtime of <i>Libquantum</i> (Uni-Core / 128 Sets / 16 Way-Associativity/ 64 KB block size) . . .	35
2.3	Hits vs Misses for different configurations . . . . .	37
2.4	Performance VS Cache size . . . . .	38
2.5	<i>ferret</i> VS <i>blackscholes</i> , number of hits at different stack distances . . . . .	39
2.6	Architecture of overall tuning framework . . . . .	41
2.7	A demonstration of early eviction . . . . .	44
2.8	Stack distance based estimation of total L2 misses versus measurement for different number of ways along with the total number of L1 invali- dations due to L2 replacement for each size ( <i>blackscholes</i> ) . . . . .	46

## List of Figures

---

2.9	Stack distance based estimation of total L2 misses versus measurement for different number of ways along with the total number of L1 invalidations due to L2 replacement for each size ( <i>x264</i> ) . . . . .	47
2.10	Adjusted stack distance based estimation of total L2 misses versus measurement for different number of ways along with the total number of L1 invalidations due to L2 replacement for each size ( <i>blackscholes</i> ) . . . . .	48
2.11	Adjusted stack distance based estimation of total L2 misses versus measurement for different number of ways along with the total number of L1 invalidations due to L2 replacement for each size ( <i>x264</i> ) . . . . .	48
2.12	The tuning procedure of multi core framework . . . . .	52
2.13	The average performance loss versus power saving . . . . .	53
2.14	Temporary cache utility variation over time . . . . .	54
2.15	Temporary cache utility variation over time . . . . .	55
2.16	Example snapshot of cache content before way shut off . . . . .	58
2.17	Utility estimation using different time intervals . . . . .	59
2.18	Temporary cache utility variation over time . . . . .	61
2.19	Temporary cache utility variation over time . . . . .	62
2.20	Comparison between the increase-only and increase/reduce tuning schemes	63
2.21	L2 demand of each tile for different applications, the sixteen tiles are represented as a grid in the horizontal plane of each graph . . . . .	64
2.22	Actual L2 resource allocation after applying resize tuning . . . . .	65
<hr/>		
3.1	Illustration of effectiveness example 1 : Future access sequence of block A and block B . . . . .	71
3.2	Illustration of effectiveness example 1 : Block B remains in the cache all the time . . . . .	72
3.3	Illustration of effectiveness example 1 : Swap block B with block A between T2 to T5 . . . . .	72
3.4	Example 2 to illustrate block effectiveness:Future access sequence of Block A and Block B . . . . .	73
3.5	Example 2 to illustrate block effectiveness:Swap block B with block A upon each miss . . . . .	74

---

3.6	Example 2 to illustrate block effectiveness:Block B remains in the cache all the time . . . . .	74
3.7	Illustration of effectiveness example 3 . . . . .	76
3.8	Three different cache access patterns . . . . .	79
3.9	MITT Demonstration . . . . .	83
3.10	Rank Sequence . . . . .	84
3.11	Normalized total number of misses using EBR with 1MB cache . . . . .	87
3.12	Normalized total number of misses using EBR with 2MB cache . . . . .	87
3.13	Normalized total number of misses using EBR with 4MB cache . . . . .	87
3.14	MITT 1 . . . . .	89
3.15	Normalized total number of misses achieved by EBR with different MITT compared to dynamic EBR (Type 1 applications) . . . . .	91
3.16	Normalized total number of misses achieved by EBR with different MITT compared to dynamic EBR (Type 2 applications) . . . . .	92
3.17	Normalized total number of misses achieved by EBR with different MITT compared to dynamic EBR (Type 3 applications) . . . . .	93
3.18	Normalized Total Number of Misses using Dynamic EBR with 1MB cache	94
3.19	Normalized total number of misses using dynamic EBR with 2MB cache	94
3.20	Normalized total number of misses using dynamic EBR with 4MB cache	94
3.21	Normalized total number of misses compared among different D-EBR complexities for 1M 16way cache . . . . .	96
3.22	Normalized total number of misses compared among different D-EBR complexities for 2M 16way cache . . . . .	96
3.23	Normalized total number of misses compared among different D-EBR complexities for 4M 16way cache . . . . .	97
3.24	Normalized total number of misses compared among different D-EBR complexities for 4M 64way cache . . . . .	97
3.25	Simulation result of D-EBR for multi-application . . . . .	98
3.26	Hardware requirements for EBR . . . . .	99

---

4.1	Example of future access trace . . . . .	106
-----	--	-----

---

4.2	The number of extra misses that would be incurred by reducing cache associativity by 1 from different cache size. Test-bench: SPEC CPU2006 : <i>games</i> , 1 billion instruction, uni-core, 4 MByte L2Cache . . . . .	114
4.3	The number of extra misses that would be incurred when reducing cache associativity by 1 from different cache sizes. Test-bench: SPEC CPU2006 : <i>namd</i> , 1 billion instruction, Uni-core, 4 MByte L2Cache . . . . .	115
4.4	The number of extra misses would incur when reduce cache associativity by 1 from different cache size. Compare among actual measurement, estimation with knowledge of evicting rank width and estimation without knowledge of evicting rank width. Test-bench: SPEC CPU2006 : <i>libquantum</i> , 1 billion instruction, uni-core, 4 MByte L2Cache . . . . .	116
4.5	Runtime utility (the minimum run-time cache size to ensure a certain percentage of performance compared with a fully activated cache) of <i>bodytrack</i> and timing of way allocation events at each tile, Test-bench: PARSEC2 : <i>bodytrack</i> , 1 billion instruction, 16-core, 1 MByte x 16 Shared L2 cache, core1 to core4 first row from left to right, core5 to core8 second row from left to right, core9 to core12 third row from left to right, core13 to core16 last row from left to right . . . . .	118
4.6	Runtime utility (the minimum run-time cache size to ensure a certain percentage of performance compared with a fully activated cache) of <i>bodytrack</i> and timing of way allocation events at each tile. Test-bench: PARSEC2 : <i>bodytrack</i> , 1 billion instruction, 16-core, 1 MByte x 16 Shared L2 cache, core 1 to core 4 first row from left to right, core 5 to core 8 Second row from left to right, core 9 to core 12 third row from left to right, core 13 to core 16 last row from left to right . . . . .	120
4.7	Increased only mechanism, LRU and EBR based comparison in performance, L2 leakage power, total power, EDP . . . . .	127
4.8	Runtime heatmap of cache utility during LRU based increase-only resize when running <i>Blackscholes</i> , Y axis represents cache associativity which ranging from 2 to 16, the heat color represents the cache reuse would happen at that stack depth during run-time. On top of the heat map is the timing of cache way-allocation event happened at each tile with different threshold during runtime . . . . .	129

---

4.9	Runtime heatmap of cache utility during EBR based increase-only resize when running <i>Blackscholes</i> , Y axis represents cache associativity which ranging from 2 to 16, the heat color represents the cache reuse would happen at that stack depth during run-time. On top of the heat map is the timing of cache way-allocation event happened at each tile with different threshold during runtime . . . . .	130
4.10	Runtime heatmap of cache utility during LRU based increase-only re-size when running <i>Bodytrack</i> , Y axis represents cache associativity which ranging from 2 to 16, the heat color represents the cache reuse would happen at that stack depth during run-time. On top of the heat map is the timing of cache way-allocation event happened at each tile with different threshold during runtime . . . . .	131
4.11	Runtime heatmap of Cache Utility during EBR based increase-only re-size when running <i>Bodytrack</i> , Y axis represents cache associativity which ranging from 2 to 16, the heat color represents the cache reuse would happen at that stack depth during run-time. On top of the heat map is the timing of cache way-allocation event happened at each tile with different threshold during runtime . . . . .	132
4.12	Runtime heatmap of Cache Utility during LRU based increase-only re-size when running <i>Canneal</i> , Y axis represents cache associativity which ranging from 2 to 16, the heat color represents the cache reuse would happen at that stack depth during run-time. On top of the heat map is the timing of cache way-allocation event happened at each tile with different threshold during runtime . . . . .	133
4.13	Runtime heatmap of Cache Utility during EBR based increase-only re-size when running <i>Canneal</i> , Y axis represents cache associativity which ranging from 2 to 16, the heat color represents the cache reuse would happen at that stack depth during run-time. On top of the heat map is the timing of cache way-allocation event happened at each tile with different threshold during runtime . . . . .	134

---

4.14	Runtime heatmap of Cache Utility during LRU based increase-only re-size when running <i>X264</i> , Y axis represents cache associativity which ranging from 2 to 16, the heat color represents the cache reuse would happen at that stack depth during run-time. On top of the heat map is the timing of cache way-allocation event happened at each tile with different threshold during runtime . . . . .	135
4.15	Runtime heatmap of Cache Utility during EBR based increase-only re-size when running <i>X264</i> , Y axis represents cache associativity which ranging from 2 to 16, the heat color represents the cache reuse would happen at that stack depth during run-time. On top of the heat map is the timing of cache way-allocation event happened at each tile with different threshold during runtime . . . . .	136
4.16	State machine of reducing state bit . . . . .	139
4.17	Increase/reduce mechanism, LRU and EBR based comparison in performance, L2 leakage power, total power, EDP . . . . .	141
4.18	Comparison of performance, L2 leakage power, total power, EDP between increasing only mechanism and increase/reduce mechanism . . . . .	144
4.19	Runtime heatmap of cache utility during EBR based increase/reduce resize when running <i>Blackscholes</i> , Y axis represents cache associativity which ranging from 2 to 16, the heat color represents the cache reuse would happen at that stack depth during run-time. On top of the heat map is the timing of cache way-allocation event happened at each tile with different threshold during runtime . . . . .	146
4.20	Runtime heatmap of cache utility during EBR based increase/reduce re-size when running <i>bodytrack</i> , Y axis represents cache associativity which ranging from 2 to 16, the heat color represents the cache reuse would happen at that stack depth during run-time. On top of the heat map is the timing of cache way-allocation event happened at each tile with different threshold during runtime . . . . .	147
4.21	Runtime heatmap of cache utility during EBR based increase/reduce resize when running <i>canneal</i> , Y axis represents cache associativity which ranging from 2 to 16, the heat color represents the cache reuse would happen at that stack depth during run-time. On top of the heat map is the timing of cache way-allocation event happened at each tile with different threshold during runtime . . . . .	148

---

4.22	Runtime heatmap of cache utility during EBR based increase/reduce resize when running $x264$ , Y axis represents cache associativity which ranging from 2 to 16, the heat color represents the cache reuse would happen at that stack depth during run-time. On top of the heat map is the timing of cache way-allocation event happened at each tile with different threshold during runtime . . . . .	149
4.23	Hardware implementation of <i>EBR</i> based cache resizing and overhead at each set . . . . .	151
4.24	Comparison between dynamic-EBR based tuning scheme and static-EBR based tuning scheme, in performance, L2 leakage power, total power, EDP when threshold $T=8$ . . . . .	154

---

5.1	Breakdown of total memory access latency of 16 Core system . . . . .	160
5.2	Breakdown of total memory access latency of 64 Core system . . . . .	160
5.3	Base system of our dynamic cache design . . . . .	161
5.4	Different ways to interpret a data address when the cache size is different . . . . .	163
5.5	An illustration of way partitioning . . . . .	164
5.6	A demonstration of how P/S controller works . . . . .	167
5.7	P/S resizeable EBR replacement policy hardware . . . . .	168
5.8	Exploring all possible static partitions - 1 . . . . .	171
5.9	Exploring all possible static partitions - 2 . . . . .	172
5.10	Dynamic private shared cache partition performance normalized to the fixed partition (2P:14S) . . . . .	174
5.11	Partition achieved by dynamic partitioning vs optimal partitioning . . .	175
5.12	Dynamic private shared cache partitioning performance normalized to the optimal scheme . . . . .	177
5.13	Tuning procedure in time . . . . .	177
5.14	Dynamic partitioning with cache resizing integrated (Number of misses and access latency are normalized to running under fully activated shared cache) . . . . .	179

---



# List of Tables

1.1	Flynn’s taxonomy . . . . .	4
1.2	Example of directory state entry . . . . .	13
2.1	Recency position distribution upon hits . . . . .	36
2.2	A set of example parameters based on our simulation framework . . . . .	42
2.3	Parameters for the simulation framework: multi core platform and single core platform . . . . .	49
2.4	Proportion of different block status upon way shut off . . . . .	57
2.5	Average performance (CPI), leakage power, total power and EDP of different tunings scheme and different threshold value relative to a full size cache . . . . .	60
3.1	Parameters for the simulation framework . . . . .	84
3.2	Simulation workloads . . . . .	85
3.3	Parameters for the simulation framework . . . . .	100
4.1	Replacement policy related status . . . . .	106
4.2	Count reuse at evicting rank . . . . .	109
4.3	Evicting rank size upon reuse SPEC CPU 2006 gamess associativity=16 .	111
4.4	Evicting rank size upon reuse SPEC CPU 2006 gamess associativity=5 .	113
4.5	PARSEC2 16MB cache 16 core : MRU re-reference/total cache re-reference	113
4.6	SPEC CPU2006 4MB cache uni core : MRU re-reference/total cache re-reference . . . . .	114
4.7	PARSEC2: streamcluster 16MB cache 16 core: run 1 and run 2 . . . . .	122
4.8	PARSEC2: blackscholes: performance estimation verification . . . . .	124
4.9	PARSEC2: streamcluster performance estimation verification . . . . .	124
4.10	PARSEC2: ferret: performance estimation verification . . . . .	126

## List of Tables

---

4.11	Average EDP of all 8 applications from PARSEC2 achieved by different threshold with EBR based increasing only tuning mechanism and LRU based increasing only tuning mechanism, both normalized to LRU running with full size cache . . . . .	137
4.12	Average EDP of all 8 applications from <i>PARSEC2</i> achieved by different threshold with <i>EBR</i> based increasing only tuning mechanism and <i>LRU</i> based increasing only tuning mechanism, both normalized to their respective full size cache . . . . .	138
4.13	Trade off between power and performance of <i>bodytrack</i> with different threshold . . . . .	138
4.14	Average total power of all 8 applications from PARSEC2 relative to LRU with a fully active cache, increasing only versus increase/reduce . . . .	140
4.15	Threshold $T=8$ , the total power comparison between <i>EBR</i> based increasing only and <i>EBR</i> based increase/reduce . . . . .	142
4.16	Average L2 usage using EBR based increasing only method and the average activated L2 at the end of simulation . . . . .	143
4.17	Average L2 usage using EBR based Increasing/Reduce method and the Average activated L2 at the end of simulation . . . . .	145
4.18	MITT interval at different associativity for static EBR policy . . . . .	152
4.19	Normalized performance of <i>streamcluster</i> with different MITT . . . . .	153
5.1	Average L2 hit latency for different applications (cycles) . . . . .	160
5.2	Proportion of reused L1 victims amongst L2 access requests . . . . .	162
5.3	Parameters for the simulation framework . . . . .	170
5.4	Dynamic scheme performance and optimal scheme performance normalized to fixed scheme . . . . .	175