Memory Subsystem Optimization Techniques for Modern High-Performance General-Purpose Processors

by

Akhil Arunkumar

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved September 2018 by the
Graduate Supervisory Committee:

Carole-Jean Wu, Chair
Aviral Shrivastava
Yann-Hang Lee
Evgeny Bolotin

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

General-purpose processors propel the advances and innovations that are the subject of humanity's many endeavors. Catering to this demand, chip-multiprocessors (CMPs) and general-purpose graphics processing units (GPGPUs) have seen many high-performance innovations in their architectures. With these advances, the memory subsystem has become the performance- and energy-limiting aspect of CMPs and GPGPUs alike. This dissertation identifies and mitigates the key performance and energy-efficiency bottlenecks in the memory subsystem of general-purpose processors via novel, practical, microarchitecture and system-architecture solutions.

Addressing the important Last Level Cache (LLC) management problem in CMPs, I observe that LLC management decisions made in isolation, as in prior proposals, often lead to sub-optimal system performance. I demonstrate that in order to maximize system performance, it is essential to manage the LLCs while being cognizant of its interaction with the system main memory. I propose ReMAP, which reduces the net memory access cost by evicting cache lines that either have no reuse, or have low memory access cost. ReMAP improves the performance of the CMP system by as much as 13%, and by an average of 6.5%.

Rather than the LLC, the L1 data cache has a pronounced impact on GPGPU performance by acting as the bandwidth filter for the rest of the memory subsystem. Prior work has shown that the severely constrained data cache capacity in GPGPUs leads to sub-optimal performance. In this thesis, I propose two novel techniques that address the GPGPU data cache capacity problem. I propose ID-Cache that performs effective cache bypassing and cache line size selection to improve cache capacity utilization. Next, I propose LATTE-CC that considers the GPU's latency tolerance feature and adaptively compresses the data stored in the data cache, thereby

increasing its effective capacity. ID-Cache and LATTE-CC are shown to achieve 71% and 19.2% speedup, respectively, over a wide variety of GPGPU applications.

Complementing the aforementioned microarchitecture techniques, I identify the need for system architecture innovations to sustain performance scalability of GPGPUs in the face of slowing Moore's Law. I propose a novel GPU architecture called the Multi-Chip-Module GPU (MCM-GPU) that integrates multiple GPU modules to form a single logical GPU. With intelligent memory subsystem optimizations tailored for MCM-GPUs, it can achieve within 7% of the performance of a similar but hypothetical monolithic die GPU. Taking a step further, I present an in-depth study of the energy-efficiency characteristics of future MCM-GPUs. I demonstrate that the inherent non-uniform memory access side-effects form the key energy-efficiency bottleneck in the future.

In summary, this thesis offers key insights into the performance and energy-efficiency bottlenecks in CMPs and GPGPUs, which can guide future architects towards developing high-performance and energy-efficient general-purpose processors.

To Amma, Appa and Nanditha

ACKNOWLEDGMENTS

First, and foremost, I would like to thank God for blessing me with the inspiration, courage, and strength to embark on this life-changing journey of a Ph.D.

I would like to express my deep gratitude to my advisor, Prof. Carole-Jean Wu. At every step, Prof. Wu has provided me with opportunities, and also enabled me to develop the necessary skills to be an independent and successful researcher. She has helped me cement the attitude of always aiming for the best. I am very thankful for her continuous guidance and support throughout my Ph.D. journey. No matter how busy she was, she has always made time to help me. I have always found inspiration in our numerous technical discussions together. I consider myself fortunate for having an encouraging, and supportive advisor like her.

I am very thankful to Prof. Aviral Shrivastava, Prof. Yann-Hang Lee, and Dr. Evgeny Bolotin for taking time to serve on my thesis committee and offering valuable feedback on my work. My interactions with Prof. Shrivastava early on in the program helped me become secure in the self-confidence that if at all anyone could do something, I could do it. I am also thankful to other faculty and staff members at ASU for their support. I would like to thank Christina Sebring for helping me ensure that I follow all the requirements of my degree correctly at all times. I would also like to thank Pamela Dunn, Monica Dugan, and Theresa Chai for helping me with the funding, scheduling, and travel needs during my stay at ASU.

During the course of my Ph.D. I have had the good fortune of gaining invaluable industry experience through multiple internships. I would like to thank Prof. Wu for encouraging me to take up these internships periodically. These internships have allowed me to expand my professional horizons and have resulted in fruitful research collaborations. My internships have been great learning experiences thanks to the mentors I have had. Through my internship at Nvidia Research, I have had the

opportunity to meet and work with Evgeny and for an extended period of time. I am very thankful for Evgeny's technical guidance during my internship and for his continued interest in my research work. In addition, I am deeply indebted to his constant support and encouragement during the final years of my Ph.D. I am also thankful for the support I received from Dr. David Nellans and Dr. Aamer Jaleel during my internship. I am grateful for the technical and career advice I have received from Aamer. I would also like to thank Dr. Tarun Nakra for mentoring me during one of my first industry research internships at Samsung SARC. Through our discussions, Tarun helped me broaden my research vision. Moreover, he also helped me understand and appreciate the process of exploring a research idea, and eventually seeing it to fruition as a product feature that impacts millions of users.

My Ph.D. experience was made a pleasant one by my colleagues and friends. Shin-Ying, Jhe-Yu, Davesh, Vignesh, Hsing-Min, and Ben thank you for your support and the opportunity to have great collaborative works together. Davesh, Amrit, Ujjwal, DK, Moslem, Niranjan and Sudhi, thank you for your support.

This journey of mine would not have been possible without the support of my family. I thank my parents for their unwavering love and support. They have always believed in me and encouraged me to go after my dreams with the confidence that they are always there, rooting for me. It is because of them, I am what I am today. I want to thank my entire family for cheering and praying for me. Special thanks to Vinay and Pooja for being there for me at all times. Finally, I want to thank my dear wife Nanditha for her unconditional love and support during this journey. Her appreciative and encouraging words helped me keep my head down and focus on the goal, even during otherwise difficult times. I dedicate this thesis to my family.

LIST OF PUBLICATIONS BY THE AUTHOR

1. **A. Arunkumar**, E. Bolotin, D. Nellans, and C.-J. Wu, "Understanding the Future of Energy Efficiency in Multi-Module GPUs", to appear in "Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)", 2019.

2. **A. Arunkumar**, S.-Y. Lee, V. Soundararajan, and C.-J. Wu, "LATTE-CC: Latency Tolerance Aware Adaptive Cache Compression Management for Energy-Efficient GPUs", in "Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)", 2018.

3. **A. Arunkumar**, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability", in "Proceedings of the International Symposium on Computer Architecture (ISCA)", 2017.

4. **A. Arunkumar**, S.-Y. Lee, and C.-J. Wu, "ID-Cache: Instruction and Memory Divergence Based Cache Management for GPUs", in "Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)", 2016.

5. **A. Arunkumar** and C.-J. Wu, "ReMAP: Reuse and Memory Access Cost Aware Eviction Policy for Last Level Cache Management", in "Proceedings of the IEEE International Conference on Computer Design (ICCD)", 2014.

Publications not part of this dissertation:

6. D. Shingari*, **A. Arunkumar***, B. Gaudette, S. Vrudhula, and C.-J. Wu, "DORA: Optimizing Smartphone Energy Efficiency and Web Browser Performance under Interference", in "Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)", 2018. (*equal contribution)

7. U.Milic, O. Villa, E. Bolotin, **A. Arunkumar**, E. Ebrahimi, A. Jaleel, A. Ramirez, and D. Nellans, "Beyond the Socket: NUMA-Aware GPUs", in "Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)", 2017.

8. H.-M. Chen, S. Jeloka, **A. Arunkumar**, D. Blaauw, C.-J. Wu, T. Mudge, and C. Chakrabarti, "Using Low Cost Erasure and Error Correction Schemes to Improve Reliability of Commodity DRAM Systems", in "IEEE Transactions on Computers (TC)", April 2016.

9. S.-Y. Lee, **A. Arunkumar**, and C.-J. Wu, "CAWA: Coordinated Warp Scheduling and Cache Prioritization for Critical Warp Acceleration of GPGPU Workloads", in "Proceedings of the International Symposium on Computer Architecture (ISCA)", 2015.

10. D. Shingari, **A. Arunkumar**, and C.-J. Wu, "Characterization and Throttling-based Mitigation of Memory Interference for Heterogeneous Smartphones", in "Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)", 2015.

11. H.-M. Chen, **A. Arunkumar**, C.-J. Wu, T. Mudge, and C. Chakrabarti, "E-ECC: Low Power Erasure and Error Correction Schemes for Increasing Reliability of Commodity DRAM Systems", in "Proceedings of the International Symposium on Memory Systems (MEMSYS)", 2015.

12. **A. Arunkumar**, A. Panday, B. Joshi, A. Ravindran, and H. P. Zaveri, "Estimating Correlation for a Real-Time Measure of Connectivity", in "Proceedings of the International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)", 2012.

TABLE OF CONTENTS

Page

LIST OF TABLES xiv

LIST OF FIGURES xv

CHAPTER

1 Introduction 1

1.1 Background: Memory Subsystem Inefficiencies in General Purpose Processors 2

1.1.1 Chip-Multiprocessors and the Last Level Cache 2

1.1.2 The GPGPU Execution Model and Constrained Data Cache Capacity 3

1.2 Research Overview 5

1.2.1 Main Memory Aware Last Level Cache Management for CMPs 6

1.2.2 Divergence Aware Data Cache Management for GPGPUs ... 7

1.2.3 Adaptive Cache Compression Management for GPGPU Data Caches 8

1.2.4 GPGPUs and the Memory Subsystem Design for the Post-Moore's Law Era 9

1.2.5 Energy Efficiency Scaling for Future Multi-Module GPGPUs 10

1.3 Contributions 11

1.3.1 Thesis Outline 12

2 Reuse and Memory Access Cost Aware Cache Management for CMP Systems 14

2.1 Background and Motivation 14

2.2 ReMAP Design and Implementation 18

2.2.1 Recency Estimation 19

CHAPTER Page

2.2.2 Post Eviction Reuse Distance Estimation 20
2.2.3 Memory Access Cost Determination 22
2.2.4 α, β, and γ Parameters in $\text{Effective}_{\text{Cost}}$ Computation 23
2.2.5 Implementation and Hardware Overhead 23
2.3 Evaluation and Analysis 25
2.3.1 Simulation Infrastructure 25
2.3.2 Workload Construction 25
2.3.3 Sequential Workloads Results 28
2.3.4 Benefit of Using PERD and MAC Information in Isolation .. 31
2.3.5 Sensitivity to Victim Buffer Storage 32
2.3.6 Sensitivity to System Parameters 32
2.3.7 Multiprogrammed Workloads Results 33
2.4 Related Work 34
2.4.1 Reuse Distance Prediction 34
2.4.2 Dead Block Prediction 35
2.4.3 Coordinating LLC Management with DRAM 36
2.5 Chapter Summary 36
3 Instruction and Memory Divergence Based Cache Management for GPGPU Systems 38
3.1 Background and Motivation 38
3.1.1 Application Sensitivity to Cache Capacity and Interconnect Bandwidth 40
3.1.2 Inefficient Cache Utilization in GPGPUs 42

CHAPTER Page

3.1.3 Inefficient Cache and Bandwidth Utilization due to Default Fixed Cache Line Size Configuration 43
3.2 ID-Cache Design and Implementation 46
3.2.1 Towards Effective Cache Bypassing 46
3.2.2 PC and Memory Divergence Pattern Guided Bypassing 50
3.2.3 Towards Efficient Cache Line Size Selection 51
3.2.4 Divergence Guided Adaptive Line Size Insertion (ALSI) 55
3.2.5 ID-Cache: Instruction and Divergence Based Cache Management 56
3.3 Evaluation and Analysis 58
3.3.1 Simulation Infrastructure 58
3.3.2 Workload Construction 59
3.3.3 PC and Memory Divergence Pattern Guided Bypassing 61
3.3.4 Divergence Based Adaptive Line Size Insertion (ALSI) 63
3.3.5 ID-Cache - Instruction and Divergence Based Cache Management 64
3.4 Related Work 66
3.5 Chapter Summary 67
4 Latency Tolerance Aware Cache Compression Management for GPGPUs 68
4.1 Background and Motivation 68
4.1.1 GPGPU Workload Data Compressibility 69
4.1.2 Latency Tolerance of GPGPUs 72
4.1.3 Adaptive Compression in GPGPUs 73
4.2 LATTE-CC Design and Implementation 76

CHAPTER Page

4.2.1 Minimizing $AMAT_{GPU}$ for Optimal Compression Mode Selection . . . 77
4.2.2 Dynamic Estimation of $AMAT_{GPU}$ . . . 78
4.2.3 Putting it all Together . . . 81
4.3 Evaluation and Analysis . . . 82
4.3.1 Simulation Infrastructure . . . 82
4.3.2 Workload Construction . . . 83
4.3.3 Component Compression Policy Implementation Details . . . 85
4.3.4 Overall Performance and Energy Impact . . . 87
4.3.5 Comparing LATTE-CC with an Offline Optimal Policy . . . 90
4.3.6 An Illustrating Application Example: Similarity Score (SS) . . . 92
4.3.7 Benefits of Latency Tolerance Awareness . . . 94
4.3.8 Flexibility of LATTE-CC Design . . . 95
4.4 Related Work . . . 97
4.4.1 Data Compression in CMPs . . . 97
4.4.2 Data Compression for GPU Memory . . . 98
4.5 Chapter Summary . . . 99
5 Multi-Chip-Module GPGPUs and the Memory Subsystem Design for the Post Moore's Law Era . . . 100
5.1 Background and Motivation . . . 100
5.1.1 GPU Application Scalability . . . 101
5.1.2 Multi-GPU Alternative . . . 103
5.1.3 Package-Level Integration . . . 103
5.2 Multi-Chip-Module GPU Design . . . 104

CHAPTER Page

5.2.1 MCM-GPU Organization 105
5.2.2 MCM-GPU and GPM Architecture 106
5.2.3 On-Package Bandwidth Considerations 108
5.3 Evaluation and Analysis 110
5.3.1 Simulation Infrastructure 111
5.3.2 Workload Construction 111
5.3.3 Revisiting MCM-GPU Cache Architecture 112
5.3.4 CTA Scheduling for GPM Locality 117
5.3.5 Data Partitioning for GPM Locality 120
5.3.6 Optimized MCM-GPU Performance Summary 125
5.3.7 MCM-GPU Performance vs Multi-GPU 127
5.4 Related Work 129
5.5 Chapter Summary 130
6 Understanding the Energy Efficiency of Multi-Chip-Module GPGPUs and the Dependence on the Memory Subsystem 132
6.1 Background and Motivation 132
6.2 EDPSE: Quantifying GPU Energy Efficiency at Scale 136
6.3 GPUJoule: A GPU Energy Estimation Framework 138
6.3.1 Micro-Benchmark Construction 141
6.3.2 Generating Energy Estimates 144
6.4 Energy Efficiency and the Future of Multi-Module GPUs 146
6.4.1 Experimental Methodology 146
6.4.2 Understanding Energy Efficiency 150
6.4.3 Optimizing for Energy Efficiency 153

CHAPTER Page

6.4.4 Decomposing EDSPE Improvements 156
6.4.5 Discussion 158
6.5 Related Work 159
6.6 Chapter Summary 160
7 Conclusion 162
REFERENCES 167

LIST OF TABLES

Table		Page
2.1	Post Eviction Reuse Distance Encoding.	21
2.2	Memory Access Cost Classification and Encoding.	22
2.3	Simulated System Architectural Details.	26
2.4	Sequential Workloads.	26
2.5	Multiprogrammed Workload Mixes.	27
3.1	ID-Cache: GPGPU-sim simulation configurations.	58
3.2	ID-Cache: GPGPU Benchmarks.	59
4.1	Comparison between the state-of-the-art cache compression algorithms.	69
4.2	LATTE-CC: Baseline system configurations.	83
4.3	LATTE-CC: GPGPU benchmarks and input sets.	84
5.1	Key characteristics of recent NVIDIA GPUs.	101
5.2	Approximate bandwidth and energy parameters for different integration domains.	104
5.3	Baseline MCM-GPU simulation configuration.	112
5.4	The high parallelism, memory intensive workloads for MCM-GPU evaluation	113
6.1	The NVIDIA Tesla K40 experimental platform.	143
6.2	GPU applications and their input sets. C: Compute intensive benchmarks and M: Memory bandwidth intensive benchmarks.	147
6.3	Simulated multi-module GPU configurations.	149
6.4	Simulated per-GPM I/O bandwidth.	150

LIST OF FIGURES

Figure Page

1.1 An abstract view of the chip-multiprocessor architecture and its memory subsystem. . . . 3

1.2 An abstract view of the general purpose GPU architecture showing the streaming multiprocessors (SM), execution lanes (L), and the memory subsystem. . . . 4

2.1 Memory access cost and oracle reuse information of evicted lines at LLC for SPEC2006 benchmarks. . . . 16

2.2 Cascading bloom filter as victim buffer for PERD estimation. . . . 21

2.3 ReMAP cache organization. . . . 25

2.4 ReMAP: Reduction in LLC Misses over LRU. . . . 29

2.5 ReMAP: Improvement in IPC over LRU. . . . 29

2.6 ReMAP: performance using PERD and MAC information in isolation. . . 31

2.7 ReMAP:sensitivity to victim buffer storage. . . . 32

2.8 ReMAP: Performance for multiprogrammed workloads . . . 34

3.1 GPGPU application performance sensitivity to data cache capacity and interconnect bandwidth . . . 40

3.2 Fraction of zero reuse cache lines in GPGPU applications . . . 42

3.3 The distribution of spatial utilization of cache lines in GPGPU applications . . . 44

3.4 The performance of GPGPU applications with different L1 cache line sizes and turning off L1 data caches. . . . 45

3.5 Cache line spatial utilization over time . . . 45

3.6 The distribution of L1 data cache reuse distances seen in GPGPU benchmarks . . . 46

Figure Page

3.7 The distribution of L1 data cache reuse distance with different load instructions 48
3.8 The distribution of L1 data cache reuse distance with degree of memory divergence of a load instruction 48
3.9 The distribution of cache hits and misses in GPGPU applications. 53
3.10 The distribution of L1 data cache utilization vs degree of memory divergence 54
3.11 Performance of GPGPU applications with **offline trained** PC-Based and PC+Div-Based bypassing methods. 61
3.12 Performance of GPGPU applications with **online** PC-Based and PC+Div-Based bypassing designs. 62
3.13 The performance improvement under Adaptive Line Size Insertion. 63
3.14 Performance improvement with ID-Cache. 64
3.15 L1 data cache hit rate improvement and interconnect busy stall reduction provided by ID-Cache. 65
4.1 Compression ratio achieved by the state-of-the-art compression algorithms 70
4.2 Performance impact of the effective cache capacity increase provided by data compression. 72
4.3 Performance degradation with increase in cache hit latency due to decompression. 73
4.4 GPU latency tolerance characterization for `SS` GPGPU benchmark. 74
4.5 Potential performance and energy impact of adaptive compression 75
4.6 A conceptual overview of LATTE-CC. 76

Figure Page

4.7 Block diagram of the modern GPU architecture with the LATTE-CC components. 77
4.8 Modified set sampling in the LATTE-CC data caches. 80
4.9 A temporal representation of LATTE-CC. 82
4.10 Performance improvement with LATTE-CC. 88
4.11 L1 cache miss reduction with LATTE-CC. 88
4.12 GPU energy consumption comparison. 89
4.13 Breakdown of GPU Energy Reduction achieved by LATTE-CC. 90
4.14 Comparison of LATTE-CC's compression mode decision with decision given by Kernel-OPT. 91
4.15 Effective cache capacity variation over time for Similarity Score (SS) application. 93
4.16 Performance comparison of LATTE-CC, Adaptive-Hit-Count, and Adaptive-CMP[20] policies for C-Sens workloads. 94
4.17 LATTE-CC performance with an alternative underlying compression algorithm. 96
5.1 Hypothetical GPU performance scaling with growing number of SMs and memory system. 102
5.2 Basic MCM-GPU architecture comprising four GPU modules (GPMs). 106
5.3 Relative performance sensitivity to inter-GPM link bandwidth for a 4-GPM, 256SM MCM-GPU system. 109
5.4 MCM-GPU architecture equipped with L1.5 GPM-side cache 114

Figure Page

5.5 Performance of 256 SM, 768 GB/s inter-GPM BW MCM-GPU with 8MB (iso-transistor), 16 MB (iso-transistor), and 32 MB (non-iso-transistor) L1.5 caches. 116
5.6 Total inter-GPM bandwidth in baseline MCM-GPU architecture and with a 16MB *remote-only* L1.5 cache. 117
5.7 An example of exploiting inter-CTA data locality with CTA scheduling in MCM-GPU. 118
5.8 Performance of MCM-GPU system with a distributed scheduler. 120
5.9 Reduction in inter-GPM bandwidth with a distributed scheduler 120
5.10 First Touch page mapping policy: (a) Access order. (b) Proposed page mapping policy. 121
5.11 Exploiting cross-kernel CTA locality with First Touch page placement and distributed CTA scheduling. 123
5.12 Performance of MCM-GPU with First Touch page placement. 123
5.13 Reduction in inter-GPM bandwidth with First Touch page placement. . 124
5.14 S-curve summarizing the optimized MCM-GPU performance speedups for all workloads. 124
5.15 Breakdown of the sources of performance improvements of optimized MCM-GPU when applied alone and together 126
5.16 Performance comparison of MCM-GPU and Multi-GPU. 127
6.1 Scaling of future GPU designs via multi-module GPU architecture. 133
6.2 The average energy cost of strong scaling, when growing the number of GPMs using on-board integration. 135
6.3 GPUJoule top-down instruction-based energy modeling methodology. . . 139

Figure Page

6.4 GPUJoule energy estimation accuracy. Comparison of energy estimations with hardware measurements on the Tesla K40 GPU. 145
6.5 EDPSE for across compute intensive, memory intensive, and all workloads, for baseline on-package configuration (2x-BW). 151
6.6 Performance speedup (left axis) and energy increase (right axis) compared to each preceding multi-GPM configuration. Energy consumption is broken down by component. 152
6.7 EDPSE as a function of interconnect bandwidth settings. 154
6.8 EDPSE for on-board ring and switched interconnection networks 156
6.9 Speedup and energy consumption when varying interconnect bandwidth, and applying on-package energy optimization at the 2x-BW and 4x-BW points 157

Chapter 1

INTRODUCTION

Today's high-performance general-purpose processors propel the advances and innovations that are the subject of humanity's many endeavors. In turn, the associated demands placed on these processors have also led to a plethora of innovations in the processors themselves. Modern Chip-Multiprocessors (CMPs) are equipped with advanced speculation support [60, 64, 77, 78, 121, 160, 178], large Last Level Caches (LLCs), and sophisticated LLC management techniques [74, 142, 143, 172, 173]. As a result, they are able to provide high performance for a wide variety of applications. Furthermore, to continue the performance scaling trend and exploit available parallelism, computer architects have transformed Graphics Processing Units (GPUs) into accelerators that are specialized for general purpose parallel applications, i.e., General Purpose Graphics Processing Units (GPGPUs). GPGPUs utilize a combination of thousands of simple processor cores, massive multi-threading, and fast context-switching to deliver upto 110 Tera-Flops of performance [130].

Powered by the Moore's Law [120], the compute capability of CMPs and GPGPUs has grown consistently over the years. On the flip side, this growth has closely been followed by a massive increase in the complexity and the amount of data that is processed by these processors as well. This trend has made the memory subsystem a significant performance and energy bottleneck in these processors. Therefore, today's computer architects face the crucial challenge of delivering data to these general-purpose processors in an efficient manner. To this end, *the overarching research goal of this thesis is to identify and mitigate the major performance and energy efficiency*

bottlenecks in the memory subsystem of current and future general purpose processors via novel, practical, microarchitecture and system architecture solutions.

1.1 Background: Memory Subsystem Inefficiencies in General Purpose Processors

1.1.1 Chip-Multiprocessors and the Last Level Cache

For a broad set of important general-purpose memory-intensive applications, the performance of a CMP is effectively dictated by the performance of the memory subsystem. Figure 1.1 shows an abstract view of the CMP architecture and memory subsystem. The CMP memory subsystem is typically comprised of multi-level cache hierarchies with smaller lower level caches (L1/L2 cache) and a large Last Level Cache (LLC). Figure 1.1 also shows the typical latency cost incurred while accessing each level of the memory hierarchy. While accessing the L1 cache takes only $2ns$, delivering data from the DRAM to the processor could take multiple orders of magnitude longer at $100ns$ - $200ns$ per access. Therefore, the LLC plays a crucial role in the performance of the memory subsystem as it avoids the significant penalty of long latency main memory accesses by keeping frequently used data closer to the processor.

While it is desirable to continually increase the LLC capacity available on the system to capture more of the application's data within the LLC, it is not a practical option as it can lead to prohibitively high silicon costs. This has led to numerous research efforts being directed towards effectively managing the contents of the LLC. Prior work has shown that by intelligently retaining only the data that's most likely to be reused in the future, the LLC can improve the memory subsystem and the overall system performance significantly [33, 45, 53, 72, 74, 86, 142, 143, 164, 172]. Furthermore, the LLC management decisions can influence the performance of the other memory subsystem components it closely interacts with — the L1/L2 caches and the DRAM. Depending on the inclusion policy employed within the L1 and L2

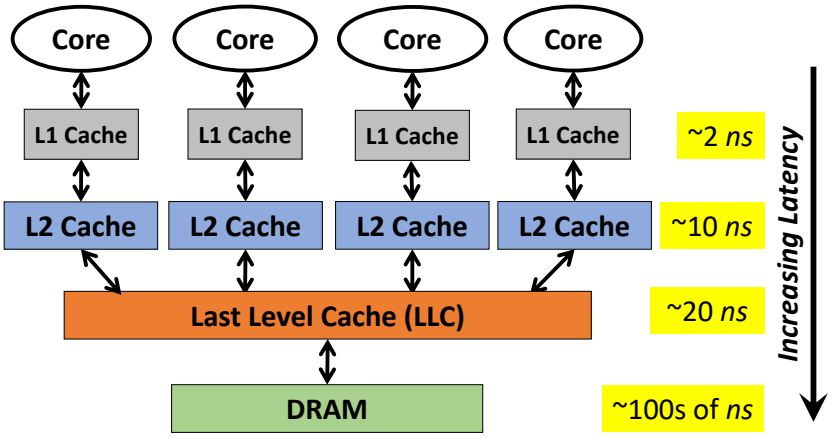


Figure 1.1: An abstract view of the chip-multiprocessor architecture and its memory subsystem.

caches, an LLC eviction could trigger an eviction from the L1 and L2 caches as well. When it comes to the DRAM, an LLC miss could result in data being flushed from the row-buffers of the DRAM, making subsequent accesses to such data more expensive. Furthermore, certain LLC misses could experience bank conflicts that lead to exacerbated memory access latency costs. As I demonstrate later in this thesis, being oblivious to the interaction of the LLC with other memory subsystem components can result in sub-optimal performance. While a few prior works address the interaction of the LLC with the L1/L2 caches [45, 72], a key challenge remains in *how to best design a practical LLC management method that not only retains valuable data within the LLC, but is also cognizant of the impact of such management decisions on the performance of the DRAM.*

1.1.2 The GPGPU Execution Model and Constrained Data Cache Capacity

GPGPUs deliver their superior performance acceleration by exploiting high degree of parallelism that is available in many general-purpose applications. They concur-

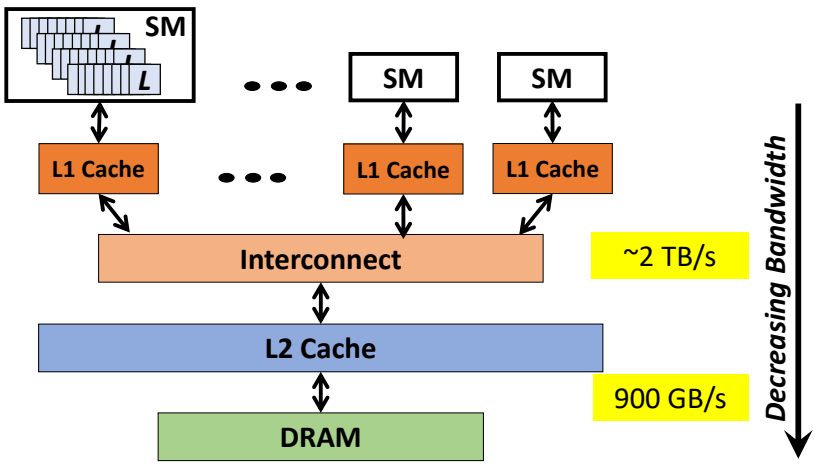


Figure 1.2: An abstract view of the general purpose GPU architecture showing the streaming multiprocessors (SM), execution lanes (L), and the memory subsystem.

rently execute thousands of parallel threads on each *streaming multiprocessor* (SM) in a batched fashion, in order to hide long latency memory accesses. A batch of parallel threads called a *warp* or a *wavefront* executes simultaneously on simple processing cores (*lanes*) that are present within the SMs. Whenever a warp encounters a long latency memory access, it is switched out of execution and is replaced by another ready warp. Due to this execution model, the GPU pipeline is kept busy, hiding the long latency memory access and thus delivering superior performance acceleration.

This massive parallelism execution paradigm places significant stresses on the memory subsystem of GPGPUs. Due to the large number of threads that are concurrently executing on the GPU, the amount of data that is requested from the memory subsystem, or in other words the memory bandwidth demand, is increased significantly. Adapting to this new execution model, the memory hierarchy of GPGPUs has also evolved accordingly — incorporating high bandwidth memories and network-on-chips, and multi-level cache hierarchies as shown in Figure 1.2. In fact, as we proceed lower down the memory hierarchy of GPGPUs, the available memory

bandwidth capacity reduces. As a result, differing from CMPs, the multi-level cache hierarchy of GPGPUs is meant to act as a bandwidth filter to alleviate the high bandwidth demand. However, relative to the thousands of threads that are executing concurrently on the GPU, the available cache capacity is significantly constrained - only on the order of a few bytes per thread. This constrained cache capacity coupled with the high bandwidth demand leads to severe cache thrashing, especially in the data caches of GPGPUs. This in turn increases the bandwidth demand from the lower levels of the memory hierarchy. Therefore, *it is imperative to find effective solutions to the constrained data cache capacity problem in GPGPUs.*

1.2 Research Overview

The focus of my dissertation is to address the key performance and energy inefficiencies in the memory subsystem of general-purpose processors through novel and practical designs. This thesis offers detailed characterization of data reuse, data compressibility, performance scalability, and energy efficiency characteristics of general purpose workloads. These workloads span a variety of important application domains including machine learning, scientific simulations, audio coding, voice recognition etc. Beyond the inefficiencies highlighted in Section 1.1, this thesis illuminates an impending performance scalability problem in GPGPUs. Addressing these, I develop and evaluate three novel microarchitecture proposals for the CMP and GPGPU memory subsystems. Additionally, to ensure continued performance acceleration, I propose a novel GPU system architecture for future GPGPUs. Taking a step further, I present a detailed analysis of the energy consumption and energy-efficiency trends as applicable to future GPGPUs. Together, these evaluations and proposals advance the state-of-the-art architecture design, and improve the performance and energy efficiency of general-purpose processors significantly.

1.2.1 Main Memory Aware Last Level Cache Management for CMPs

The performance critical role of the LLC in CMP systems has motivated a wide body of research work that offer techniques for sophisticated LLC management [32, 33, 34, 44, 45, 53, 58, 72, 82, 87, 110, 117, 142, 144, 164, 177]. However, most of these techniques were developed by considering the LLC performance in isolation. In this thesis, I observe that since the LLC interacts closely with the main memory, the prior approaches to LLC management lead to performance sub-optimality. It is imperative that an effective LLC management scheme be cognizant of the interaction between LLC management decisions and the system main memory.

It is important to note that not all cache misses or memory accesses are the same. Due to the hierarchical structure of DRAMs, different memory accesses can incur diverse memory access costs. Depending on whether an access experiences a row-buffer hit/miss or bank conflict at the DRAM, it can incur varied access latencies ranging from as low as $45ns$ to as high as $180ns$. In Chapter 2 of this thesis, I propose a novel LLC management policy, Reuse and Memory Access cost aware eviction Policy (ReMAP) [29] that takes this interaction between LLC management decisions and main memory access costs into account. ReMAP estimates the eviction priority of different cache lines by considering multiple factors such as the recency, the post-eviction reuse distance, and the memory access cost of cache lines. Upon cache line eviction, ReMAP aims to evict a cache line that is 1) expected to not receive any reuse in the near future and 2) expected to incur lower memory access cost than the other lines present in the cache. My detailed evaluation with SPEC2006 applications shows that ReMAP reduces the effective memory access cost experienced by the system and improves system performance by an average of 6.5% and by as much as 13% over the baseline.

1.2.2 Divergence Aware Data Cache Management for GPGPUs

While the performance of CMPs is significantly influenced by the LLC, this role is played by the L1 data cache on GPGPUs. Prior work has shown that the constrained L1 data cache capacity problem described in Section 1.1, leads to performance sub-optimality for many GPGPU applications [76, 101, 103, 104, 108, 132, 147, 165, 175, 176]. These applications suffer from two important caching inefficiencies. First, GPGPU applications suffer from a high degree of cache thrashing. Due to the thousands of threads that are concurrently executed on the GPUs, most of the cache lines brought into the cache end up being evicted before they receive any reuse. Second, the spatial locality observed in cache lines varies significantly, and can be extremely low in many GPGPU applications. The aforementioned issues lead to inefficient utilization of an already constrained cache space. In Chapter 3 of this thesis, I propose Instruction and memory Divergence based Cache management (ID-Cache) [28], a novel cache management technique that addresses both these problems.

Chapter 3 identifies that both the cache line reuse behavior and the extent of spatial locality can be predicted by augmenting commonly-used instruction or program counter information, with GPU specific characteristics such as the degree of memory divergence. ID-Cache leverages this insight to first make fine-grained and accurate cache bypassing decisions such that only the cache lines predicted to have reuse are inserted and retained in the cache. Second, ID-Cache predicts the degree of spacial locality and appropriately fetches cache lines of variable sizes into the cache. By employing fine-grained cache bypassing and adaptive cache line size selection together, ID-Cache not only improves the cache capacity utilization, but also reduces the bandwidth demand on the interconnect — thus achieving an average of 71% performance improvement across a wide range of cache-sensitive GPGPU applications.

1.2.3 Adaptive Cache Compression Management for GPGPU Data Caches

An orthogonal approach to alleviate the cache capacity problem is to adopt cache compression. Cache compression is a technique used to increase the effective capacity of on-chip caches by compressing the cache lines prior to their insertion. Although, cache compression comes with a decompression latency overhead, it can be particularly well-suited for GPGPU systems due to their inherent latency tolerance feature. However, I observe that cache compression techniques, typically designed for CMPs, cannot be directly applied to GPGPUs data caches. This is because 1) the data compressibility and the compression ratio achieved by different compression algorithms vary significantly across GPGPU applications, and 2) compression algorithms come with varying decompression latencies and many a times the associated latency overhead might not be hidden in GPGPUs [40, 132].

In this thesis, I take the first steps to quantify the limits of the "latency tolerance" that is available in GPGPUs, and present a detailed characterization of the data compressibility of GPGPU applications. Furthermore, I observe that unique to GPGPUs, there exists a three-way trade-off between compression ratio of different compression algorithms, their associated decompression penalty, and the available GPU latency tolerance. With these insights, I design a LATency Tolerence awarE Cache Compression management technique (LATTE-CC) [27] that is presented in Chapter 4 of this thesis. LATTE-CC is able to effectively navigate the three-way trade-off described above by adaptively choosing one of the three compression modes — no-compression mode, low-latency mode, or high-capacity mode. In doing so, LATTE-CC intelligently leverages the time-varying latency tolerance feature of GPGPUs to choose the compression mode that gives the highest effective cache capacity increase, and whose decompression penalty can be well hidden. Outperforming state-of-the-art

cache compression techniques, LATTE-CC is able to improve GPGPU performance by 19.2% on average across a wide range of cache-sensitive applications. Additionally, LATTE-CC reduces the overall GPGPU energy consumption by 10%, twice as much as what the state-of-the-art cache compression technique achieves [138].

1.2.4 GPGPUs and the Memory Subsystem Design for the Post-Moore's Law Era

In addition to the microarchitectural innovations similar to the ones presented above, the performance scaling of GPGPUs over the past decade has been supported by scaling the number of transistors and by increasing the die sizes. However, transistor scaling (Moore's Law [120]) is slowing down and is expected to soon come to an end. Thus, designing future GPU dies with transistor counts significantly larger than today's is going to be difficult, if not impossible. Additionally, due to the optical limitations of lithography, GPU die sizes can no longer be increased. In the face of these two challenges, the performance scaling trend of GPGPUs could soon plateau. To address this problem, Chapter 5 of this thesis proposes a novel GPU system architecture called the Multi-Chip-Module GPU (MCM-GPU) [25].

MCM-GPU moves away from the current monolithic die architecture of GPGPUs and proposes to integrate multiple GPU modules (GPMs) within the same package. With such a multi-chip-module design, future GPGPUs are expected to be in-package Non-Uniform Memory Access (NUMA) systems. Thus, the performance of such in-package NUMA MCM-GPU depends significantly on the memory subsystem, especially the available on-package bandwidth between the GPMs. I demonstrate that with a holistic memory subsystem design tailored for such architectures, the sensitivity to the inter-GPM bandwidth can almost be completely eliminated. Specifically, I propose a coordinated design, consisting of a new cache hierarchy design, a thread-block scheduling technique, and a memory page placement technique, to reduce the

demand placed on the inter-GPM links. Simulation-based evaluation results show that these optimizations bring the MCM-GPU within 8% of the performance of a hypothetical monolithic die GPU of similar capabilities, which cannot be built due to the technology limitations discussed above.

1.2.5 Energy Efficiency Scaling for Future Multi-Module GPGPUs

The MCM-GPU architecture proposed in Chapter 5 offers a promising path forward for future GPGPU performance scalability via multi-module integration. To cater to the energy efficiency expectation of the future, it is necessary to consider the energy cost of performance scaling for the new multi-module GPGPUs. To enable us to reason about the performance and energy overhead together, I develop a novel efficiency scaling metric called EDP Scaling Efficiency (EDPSE), in Chapter 6. EDPSE allows us to understand energy efficient scalability by considering performance, energy costs, and the amount of scaled resources together. In addition, to enable accurate estimation of energy consumption in modular GPUs, in Chapter 6, I propose an instruction based GPU energy estimation framework called GPUJoule. GPUJoule has been thoroughly validated against real hardware and found to achieve 90% energy estimation accuracy on average. Utilizing EDPSE and GPUJoule in conjunction with a GPU performance simulator, Chapter 6 presents an in-depth study that uncovers multiple new energy efficiency trends that are likely to impact future multi-module GPU designs.

I observe that future GPU energy efficiency is less likely to depend on the GPM microarchitecture energy efficiency nor the intrinsic energy costs of data movement, including off-die, inter-module data movement. This finding is particularly meaningful because the common belief places significant value on both these factors as the key drivers of energy efficiency for the future GPUs. In contrast, the inherent

NUMA side-effects of the multi-module GPU architecture are going to be the key energy efficiency bottleneck. A congested inter-module interconnect would not only degrade performance in these multi-module GPUs but would also increase the energy consumption significantly. These findings further underscore the significance of data locality optimizations within the memory subsystem of GPUs described in this thesis. Furthermore, these results highlight a pressing need for future research to focus on alleviating the NUMA side-effects in multi-module GPUs by enhancing the locality captured within the GPMs.

1.3 Contributions

This thesis highlights two key challenges faced within the memory subsystems of today's high-performance general-purpose processors and offers novel microarchitectural solutions for the same. Beyond the microarchitecture level, this thesis identifies the need for a novel GPU system architecture to ensure performance scalability in the future. With the help of advanced CMP last level cache management, GPGPU data cache management, a novel future GPGPU architecture, and finally a thorough energy efficiency analysis for future GPGPUs, this thesis pushes the horizons of general-purpose processor designs a step forward. Overall, this work advances the state-of-the-art by:

1. Offering detailed characterizations of the data reuse, data compressibility, performance scalability, and energy efficiency characteristics of a wide variety of general purpose applications executed on high performance CMPs and GPGPUs. The insights thus derived, lead to opportunities for memory subsystem optimization in CMPs and GPGPUs.

2. Highlighting the need for cache management techniques to consider the impact of seemingly independent aspects of the system architecture, such as (a) the LLC and main memory interaction in CMPs, and (b) memory divergence and its relation to data reuse and spatial locality in GPGPUs. These are shown to have a pronounced impact on the performance of memory subsystem as well as the overall system performance.

3. Offering an in-depth study and quantification of the impact of latency tolerance in GPGPUs. Furthermore, by intelligently leveraging the latency tolerating ability, I propose an efficient adaptive cache compression management method for GPGPUs.

4. Illustrating the impending performance scaling problem in GPGPU systems and proposing the novel MCM-GPU architecture along with its NUMA-aware memory subsystem design. Taking a step further, I present an in-depth study showing the main energy efficiency bottlenecks in future MCM-GPU like architectures.

5. Developing a new efficiency scaling metric and a new GPU energy estimation framework. These tools are particularly apt for future exploratory studies by the research community at a time when energy efficiency is a first order design concern.

1.3.1 Thesis Outline

The following chapters of this thesis present my research accomplishments in detail. Chapter 2 proposes ReMAP, a novel LLC management technique for CMP systems [29]. Next, Chapters 3 and 4 present two novel techniques, ID-Cache [28] and LATTE-CC [27], that alleviate the data cache capacity problem in GPGPUs.

Following that, Chapter 5 proposes a novel GPU architecture for performance scaling in the post Moore's law era and describes a holistic memory subsystem design for future GPGPUs [25]. Taking a step further, Chapter 6 offers an in-depth study for the energy consumption and energy efficiency characteristics of the new GPU architecture [26]. Finally, Chapter 7 summarizes and concludes my thesis work.

Chapter 2

REUSE AND MEMORY ACCESS COST AWARE CACHE MANAGEMENT FOR CMP SYSTEMS

Modern general-purpose chip-multiprocessors (CMPs) consist of significantly large last level caches (LLCs) that bridge the long latency gap between the processor and the main memory. By storing frequently used data closer to the processor, the LLC plays a crucial role in the performance of a CMP system. Prior work has shown that intelligent management of LLC contents is key to deliver high performance with CMPs. In this chapter, I demonstrate that prior techniques which approach LLC management, while being oblivious of its impact on the rest of the memory subsystem lead to sub-optimal performance. With this insight, this chapter proposes a novel cache line reuse and memory access cost aware eviction policy for LLC management.

2.1 Background and Motivation

There has been a wide body of research literature that is directed towards improving cache management. This has led to many innovations in insertion, promotion and replacement policies, and dead block prediction [32, 34, 44, 45, 72, 82, 87, 110, 143, 172, 177]. Most of the aforementioned studies focus on optimizing the LLC performance in isolation. Though the improvement in LLC performance leads to a significant reduction in the gap between memory and processor speeds, LLC is going to be most effective when its working is well coordinated with the levels of memory above and below it, i.e., the L1 and L2 private caches and the DRAM.

In the widely prevalent open page DRAM designs, not all LLC misses experience a fixed memory access cost. The memory access cost can vary from approximately 15ns

to 150ns (for a 2GHz processor, this corresponds to 30 cycles to 300 cycles). This is owing to the fact that LLC misses could result in row buffer hits, row buffer misses, or map to conflicting banks in the memory. To address this, Qureshi et al. [143] made a compelling case for taking memory level parallelism into consideration while making the LLC eviction decision. The memory access cost is lower for parallel LLC misses because the cost is amortized over multiple misses.

LLC misses that incur the least memory access cost are the ones that 1) do not cause bank conflict and 2) hit in the row buffer. The memory access cost incurred by such a reference is proportional to the time taken to place the data on the data lines from the sense amplifiers. This is called Column Address Strobe (CAS) latency (CL). Typically CL is about 15ns for a DDR3 SDRAM [55]. Therefore, for a 2GHz processor, the overall memory cost to fetch data from the memory is,

$$MemoryCost_{row\_buffer\_hit} \propto CL \approx 15ns = 30\ cycles \tag{2.1}$$

However, when the row buffer of the bank does not contain the row of data for the referenced address, a row buffer miss is incurred. In this case, when the data request is presented to the memory, the row that is open in the row buffer is closed (row is precharged) and the row corresponding to the new reference is brought to the row buffer. Finally, the data is placed on the data lines. The time taken to bring data into the row buffer is referred to as Row Address Strobe (RAS) to Column Address Strobe (CAS) delay. In this case, the overall memory access cost becomes the sum of the time taken for row precharge (tRP), time to bring data into row buffer (RAS to CAS delay or tRCD), and CAS latency. Typically for a DDR3 SDRAM, tRCD and tRP are about 15ns [55]. Therefore for a 2GHz processor, the overall cost to fetch

data from the memory is,

$$MemoryCost_{row\_buffer\_miss} \propto tRP + tRCD + CL$$
$$\approx 15ns + 15ns + 15ns = 45ns = 90\ cycles \quad (2.2)$$

When an LLC miss is mapped to a conflicting bank, the memory access cost experienced varies (depending on the memory access costs of earlier misses that are waiting to be serviced by this particular bank), and a cascading effect influences the memory access cost. If two requests map to a bank when another bank is idle, the second request experiences a memory access cost that is the sum of the memory access cost experienced by the first access and the second request's own memory access cost. If both the first and second requests experience a row buffer miss, this could be as high as 180 cycles (90 cycles for the first request and 90 cycles for this request).

Next, I present detailed characterization results to highlight that we can choose the eviction candidates in the LLC more intelligently, if the knowledge of the memory access cost and reuse pattern is available at the time of cache line replacement. For various SPEC2006 benchmarks, Figure 2.1 shows the memory access cost breakdown of the lines that are evicted from a typical LLC performing least recently used (LRU) replacement. With such oracular information, we can see that the fraction of evicted

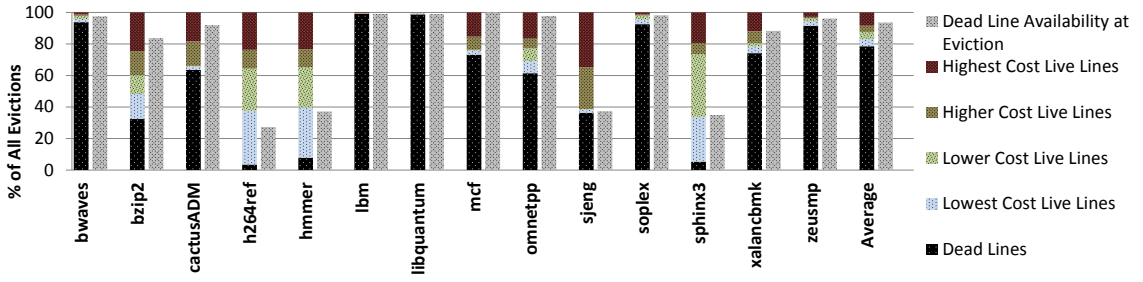


Figure 2.1: Memory access cost and oracle reuse information of evicted lines at LLC for SPEC2006 benchmarks.

lines that would have been reused in the future is typically high. I notice that **live cache lines are being evicted from the LLC while there are one or more other cache lines in the same set, that are dead**. Among all evicted cache lines, the fraction of cache lines that are indeed dead (with no future or distant reuse) is represented by the black bars labeled "Dead Lines" in Figure 2.1. The adjacent grey bars show the fraction of evictions when there is at least one cache line in the same set, that is dead. The difference between these two bars gives the fraction of times where a live line was evicted even though a dead line was available in the cache. We can see that this difference is significant for many benchmarks – 15% on an average and as much as 50%, of all evictions. This is because using only recency information, as done by LRU and other replacement policies, is ineffective in identifying the best cache eviction candidates.

While rescuing cache lines that are still useful in the near future can improve the LLC performance, the memory access penalties for these cache lines can vary significantly. For example, some cache lines that could be reused in the near future have longer memory access cost than others. This leads to the second important observation: **cache lines being evicted from the LLC are not always the ones that have the least memory access cost.** Often there are one or more cache lines in the same set whose memory access cost is lower than the memory access cost of the chosen eviction candidate. The darker bars in Figure 2.1 represent the fraction of evictions where live lines with higher memory access cost were evicted. This undesirable behavior occurs to 10% of the evictions on average (and can happen to as much as 60% of the evictions). For most of these occasions, there is opportunity to convert higher memory access cost evictions to dead line evictions or lower memory access cost evictions. Therefore, we need a cache replacement policy that prioritizes

cache lines with longer memory access cost and farther or no reuse over other lines in the set at the eviction time.

From the above insights, we can see that standard recency-based cache replacement policies leave sufficient room for improvement. Leveraging on post eviction reuse distance (PERD) and memory access cost (MAC) information along with recency information can provide additional performance benefits. In this chapter, I propose ReMAP, Reuse and Memory Access cost aware eviction Policy [29], that takes cache line reuse characteristics and memory access behavior into consideration when making cache line eviction decision. This allows ReMAP to mitigate the two undesirable effects described above and achieve higher performance compared to other recency based policies.

2.2 ReMAP Design and Implementation

Conceptually in an LRU-based cache replacement policy, each cache line in a cache set is given a reuse counter that records how long ago the particular cache line was last reused. For example in a 16-way cache, each cache line in a set is assigned a number between 0 and 15. Every time a cache line is accessed, its counter is reset to zero while all other cache lines' counters increase by 1. When a cache line needs to be replaced, the eviction candidate is selected by choosing the cache line that has the largest counter value. Instead of assigning a predetermined counter value as in LRU, ReMAP assigns a *cost* to each cache line, indicating the cost of evicting a particular cache line versus keeping it in the cache. For example, the cost is higher if a cache line to be evicted will experience a longer memory access latency when it is accessed next time. Therefore when selecting an eviction candidate, ReMAP looks at the cost for each cache line in the set and picks the line that has the *least* cost, contrary to an LRU-based system.

ReMAP determines the cost of the eviction candidate by considering a cache line's recency *(R)*, predicted post eviction reuse distance *(PERD)*, and memory access cost *(MAC)*. While the recency information gives us an insight into the line's liveliness when the line is still in the cache, PERD provides us with additional information about how soon a line would be recalled into the cache after it has been evicted. Finally, MAC provides additional information on the associated latency for main memory access when the line gets recalled. These three vital pieces of information help in assessing the worthiness of a cache line. At the time of eviction, an effective cost of each cache line is determined using a linear relationship between the aforementioned parameters.

$$Effective_{cost} = \alpha * R + \beta * PERD + \gamma * MAC \tag{2.3}$$

Intuitively, the cache line with the least $Effective_{cost}$ is less important. Therefore, ReMAP always selects the cache line with the least $Effective_{cost}$ for eviction. The pseudo code for cache line eviction selection is illustrated in Algorithm 1.

2.2.1 Recency Estimation

Recency *(R)* is typically available from the underlying cache replacement policy. For example, the recency counter in a LRU based cache indicates how recently a cache line was used. Similarly, in another state-of-the-art cache replacement policy, RRIP [74], the re-reference interval predicted value (RRPV) provides a measure of the recency of a cache line. ReMAP uses RRPV in estimating a cache line's recency (R) component of the effective cost calculation.

Input: EvictionSet

//make eviction decision

for *all lines in EvictionSet* **do**

$cache.line.EffectiveCost =$

$\alpha * cache.line.R + \beta * cache.line.PERD + \gamma * cache.line.MAC$;

if $cache.line.EffectiveCost < cache.MinCost$ **then**

$cache.MinCost = cache.line.EffectiveCost$;

$cache.EvictionCandidate = cache.line$;

end

end

//insert to victim buffer $insert\_to\_vb(cache.EvictionCandidate)$;

Algorithm 1: ReMAP eviction decision algorithm.

2.2.2 Post Eviction Reuse Distance Estimation

To learn a cache line's reuse behavior and predict the post eviction reuse distance, ReMAP uses a bloom filter [38] based victim buffer that records the address of every cache line that is evicted from the cache. Upon every cache miss, the victim buffer is looked up for the missing line address. The victim buffer is empirically designed to hold *cache_entries* entries. In order to facilitate the PERD estimation with a practical hardware overhead, the victim buffer is designed by cascading three bloom filters in a hierarchical fashion as shown in Figure 2.2. Upon eviction, a cache line is inserted into the first stage of the victim buffer. When the number of entries in the victim buffer is equal to one-third the number of entries in the cache, the entries in each stage is flushed down to the stage below it.

To classify the PERD for the entries in the victim buffer, ReMAP uses the following heuristics. If the missing address is found within the first $1/3 * cache\_entries$

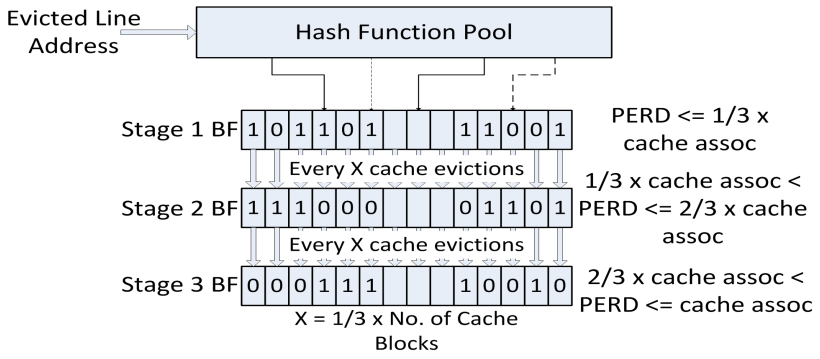


Figure 2.2: Cascading bloom filter as victim buffer for PERD estimation.

(Stage 1 BF in Figure 2.2) in the victim buffer, the line's PERD is predicted to be "near". If the missing address is found within $2/3 * cache\_entries$ (Stage 2 BF in Figure 2.2), the line's PERD is predicted to be "intermediate". If the missing address is found within $cache\_entries$ (Stage 3 BF in Figure 2.2), the line's PERD is predicted to be "far". If the missing address is not found within $cache\_entries$ entries, the line is predicted to have no reuse, or "dead". This predicted PERD is recorded with each cache line using two additional bits. The PERD encoding is described in Table 2.1. The hardware overhead of the victim buffer is described in the Section 2.2.5.

Table 2.1: Post Eviction Reuse Distance Encoding.

Reuse Dist.	PERD	Encoding
near	$0 < PERD <= 4$	3
intermediate	$5 < PERD <= 9$	2
far	$10 < PERD <= 15$	1
distant	$PERD > 15$	0

2.2.3 Memory Access Cost Determination

In addition to obtaining the recency and PERD information, it is necessary to obtain the associated memory access cost. To obtain the MAC, ReMAP uses a small auxiliary structure called MAC estimation table. The MAC estimation table holds a small memory access trace of current reference and previous references by storing the row addresses of two references preceding the current reference for each bank, and the number of references waiting to be serviced by each bank.

At the time of insertion, based on whether the bank has requests waiting and if the current row address matches the previous two row addresses, the MAC is determined as described in Table 2.2. The predicted MAC of the new inserted cache line is then recorded and used to calculate the overall effective cost at the time of eviction.

It is important to note that this mechanism for estimating the MAC of an incoming cache line relies on the similarity of row access behavior of the current and the

Table 2.2: Memory Access Cost Classification and Encoding.

Access Type	MAC	Encoding
No bank conflict; row buffer hit	Lowest	0
Bank conflict; row buffer hit;	Lower	1
No bank conflict; row buffer miss	Higher	2
Bank conflict; at least one row buffer miss from earlier misses	Highest	3

previous reference. As a previous study [143] showed, the row buffer access patterns of the current and the previous references in the LLC are highly correlated. ReMAP estimates MAC based on this insight – row access behavior of current and its previous references are similar. Table 2.2 shows the different memory access cost classification used in this study for different bank and row access scenarios. ReMAP stores the predicted MAC per cache line, as a 2-bit value.

2.2.4 α, β, and γ Parameters in $Effective_{Cost}$ Computation

The parameters α, β, and γ in the $Effective_{cost}$ calculation (Equation 2.3) represent the importance of each of the three pieces of knowledge, i.e recency, post eviction reuse distance, and memory access cost, for estimating the cost of LLC misses. I qualitatively explore different combinations of values for these parameters exhaustively. I evaluated ReMAP for setups that give equal importance to R, PERD, and MAC, higher importance to one of the three, and lastly, giving higher importance to two of the three, pieces of knowledge.

The results show that while some applications benefit from $\alpha = 1$, $\beta = 1$, $\gamma = 4$ and some other applications benefit from $\alpha = 1$, $\beta = 4$, $\gamma = 1$. Therefore, ReMAP implements a set dueling mechanism that dynamically selects the eviction policy that minimizes the total memory access cost (as compared to number of misses in traditional set dueling schemes).

2.2.5 Implementation and Hardware Overhead

Algorithm 2 shows the ReMAP algorithm and Figure 2.3 shows the hardware structures used in ReMAP. ReMAP uses the PERD estimation victim buffer which is a multilevel bloom filter. Each cache line has two 2-bit fields to record PERD and MAC estimations.

Input: cache_access

//Upon cache miss

1. Issue request to DRAM
2. Compute Effective Cost for all cache lines (Off critical path)
3. Find eviction candidate with minimum effective cost (Off critical path)
4. Perform PERD estimation victim buffer access for missing address (Off critical path)
5. Perform MAC estimation for missing address (Off critical path)

//Upon cache insertion

1. Attach PERD value to new cache line
2. Attach MAC value to new cache line
3. Complete cache insertion

Algorithm 2: ReMAP algorithm.

The PERD estimation victim buffer is implemented as a set of three bit arrays as shown in Figure 2.2. These bit arrays are of size $5 * c$ bits, where c is 10. Each cache line inserted into the victim buffer is represented by "k" bits. These "k" bits are identified by a set of "k" hash functions. The victim buffer hardware overhead for the above setup is 18.75 KB. This additional hardware requirement is reasonable given the significant performance gain and is comparable to the hardware overhead of recently proposed state-of-the-art replacement policies [45, 153, 172].

Apart from the PERD estimation victim buffer, ReMAP consists of negligible logic overhead from the effective cost calculation and 4 bits per cache line to store the line's MAC and PERD values.

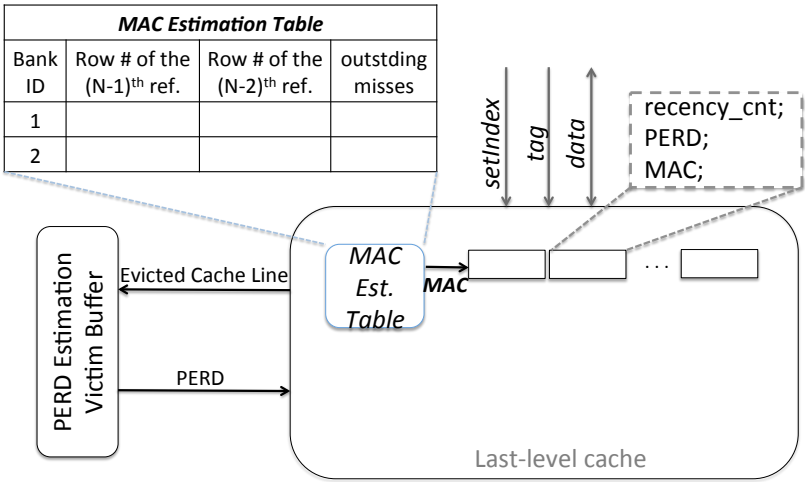


Figure 2.3: ReMAP cache organization.

2.3 Evaluation and Analysis

2.3.1 Simulation Infrastructure

I evaluate ReMAP using an open source full system simulator, gem5 [36]. I model a 4-way out-of-order processor with a 128-entry reorder buffer, a three-level non-inclusive cache hierarchy, and a multi-channel, multi-bank DRAM. The memory hierarchy is based on an Intel Core i7 system [69]. The L1 and L2 caches are private to each core and implement the LRU replacement policy. The configurations of my setup is summarized in Table 2.3. This setup is similar to the setup used in other recent studies [32, 44, 45, 72, 82, 87, 110, 143, 177].

I build ReMAP on top of a recently proposed cache replacement policy, Static Re-Reference Interval Prediction (SRRIP)[74] because SRRIP requires less hardware overhead than LRU and outperforms LRU [74].

2.3.2 Workload Construction

I evaluate ReMAP for both sequential and multiprogrammed workloads.

Table 2.3: Simulated System Architectural Details.

Processor	1 GHz, 4-way out of order, 128 entry reorder buffer
L1-I Cache	32KB, 4-way associative, 1 cycle latency, 64B block size
L1-D Cache	32KB, 8-way associative, 1 cycle latency, 64KB block size
Unified L2 Cache	256KB, 8-way associative, 10 cycle latency, 64B block size
Shared L3 Cache	1MB per core, 16-way associative, 30 cycle latency, 64B block size
Main Memory	DDR3, FCFS scheduling, open page policy, 13.75ns precharge time, 13.75ns CAS latency, 13.75ns RAS to CAS latency, 1 Channel, 8 Banks, 8 KB row buffers

Table 2.4: Sequential Workloads.

Category	Benchmarks
Memory Sensitive	h264ref, lbm, mcf, omnetpp, soplex, sphinx3
Streaming or Large Working Set	cactusADM, libquantum

Sequential Workloads: For my study with sequential workloads, I use 8 memory sensitive (MS), and streaming or large working set (Str) benchmarks from the SPEC2006 benchmark suite. I use Simpoints [139] methodology to identify single 250 million instruction representative region for each benchmark and use this for my study. Table 2.4 shows the benchmarks used in my study.

Multiprogrammed Workloads: For my study with multiprogrammed workloads, I add one memory sensitive (MS), two streaming (str), and three compute intensive (CI) benchmarks to model realistic multiprogrammed application execution scenarios.

Table 2.5: Multiprogrammed Workload Mixes.

WL#	Benchmarks	Category
1	bzip2, mcf, omnetpp, soplex	MS, MS, MS, MS
2	bzip2, mcf, lbm, sphinx3	MS, MS, MS, MS
3	bzip2, mcf, omnetpp, sphinx3	MS, MS, MS, MS
4	bzip2, soplex, omnetpp, sphinx3	MS, MS, MS, MS
5	bzip2, mcf, sphinx3, bwaves	MS, MS, MS, Str
6	bzip2, mcf, omnetpp, libquantum	MS, MS, MS, Str
7	omnetpp, soplex, zeusmp, bwaves	MS, MS, Str, Str
8	sphinx3, mcf, libquantum, bwaves	MS, MS, Str, Str
9	mcf, sphinx3, cactusADM, zeusmp	MS, MS, Str, Str
10	zeusmp, libquantum, cactusADM, bwaves	Str, Str, Str, Str
11	bzip2, mcf, sphinx3, hmmer	MS, MS, MS, CI
12	omnetpp, mcf, sphinx3, h264ref	MS, MS, MS, MS
13	bzip, soplex, h264ref, hmmer	MS, MS, MS, CI
14	bzip2, mcf, sjeng, hmmer	MS, MS, CI, CI
15	sphinx3, sjeng, h264ref, hmmer	MS, CI, MS, CI
16	sjeng, xalancbmk, h264ref, hmmer	CI, CI, MS, CI
17	xalancbmk, bwaves, h264ref, hmmer	CI, Str, MS, CI
18	libquantum, bwaves, cactusADM, hmmer	Str, Str, Str, CI

I choose 4-core combinations of the workload types and create 18 workload mixes such that all combinations of the different types are covered. The workload mixes are shown in Table 2.5.

For the multiprogrammed simulations, I fast-forward two billion instructions from the program start and simulate in detail until all the benchmarks have simulated for at least 250 million instructions. The benchmarks continue to run after they have finished executing 250 million instructions until all other benchmarks within that set have completed simulating 250 million instructions. This is done so that the faster benchmark continues to offer cache contention while the slower benchmark is running. However, in such a case, the statistics are collected only for the first 250 million instructions.

2.3.3 Sequential Workloads Results

I evaluate ReMAP by comparing its performance with LRU, DRRIP [74] and the MLP-aware replacement policy (MLP-aware) [143] as these policies are most closely related to ReMAP. I perform sensitivity studies to determine the weights given to R, MAC, and PERD in the effective cost computations. I observe that the optimal configuration varies from application to application. Therefore, I use set dueling [143] to determine the weights for MAC, R, and PERD dynamically at runtime. Specifically, for the effective cost computation, I employ set dueling between "$\alpha = 1$, $\beta = 1$, $\gamma = 4$" and "$\alpha = 1$, $\beta = 4$, $\gamma = 1$" for the performance studies. Having values which are powers of two for β and γ makes the hardware implementation simpler. Unlike in Qureshi et al. [143], the policy selector counter is updated based on the total MAC incurred by the component policies instead of the total number of misses incurred.

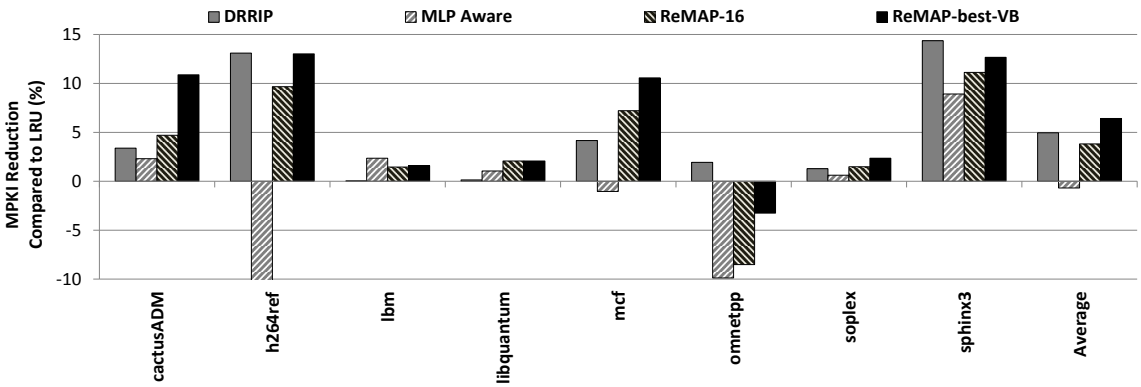


Figure 2.4: ReMAP: Reduction in LLC Misses over LRU.

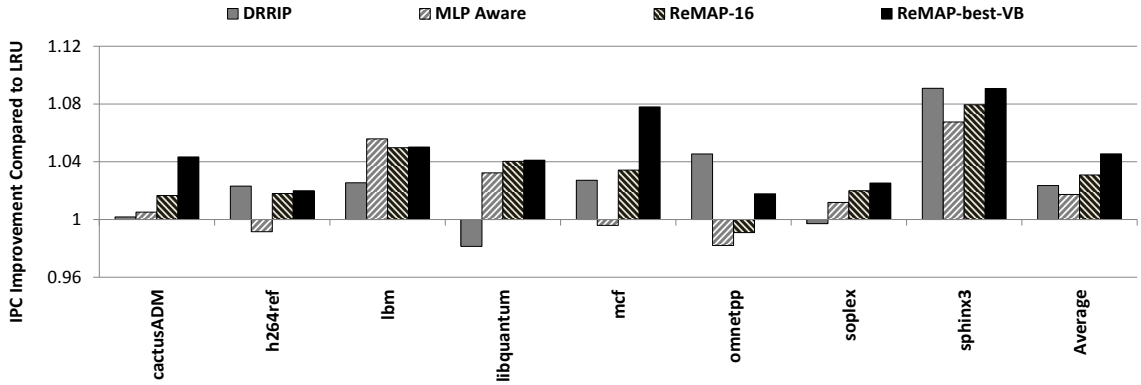


Figure 2.5: ReMAP: Improvement in IPC over LRU.

Figures 2.4 and 2.5 compare the cache miss reduction and IPC improvement experienced by the sequential workloads under the different policies: DRRIP, MLP-aware, ReMAP-16, ReMAP-best-VB. ReMAP-16 represents the performance of ReMAP with 16 entry per set victim buffer and ReMAP-best-VB represents the performance of ReMAP when victim buffer size is fine tuned for each benchmark. To identify the best victim buffer setup for each benchmark, ReMAP searches through victim buffers having 8 through 64 entries per cache set. Though most applications are found to perform best when the victim buffer sizes are less than 24 entries, applications such

as cactusADM, mcf, and soplex perform best with victim buffers containing upto 48 entries per cache set.

While all three policies reduce application LLC misses, the miss reduction does not always translate to IPC performance improvement. For example, for sphinx3, DRRIP reduces the number of misses the most, but because of the memory access cost disparity among the misses, the benefit of miss reduction is not reflected in application IPC performance improvement. Overall, ReMAP reduces the number of misses of SPEC2006 applications by as much as 13% over the baseline LRU replacement and by an average of 6.5% while MLP-aware replacement and DRRIP reduce the miss counts by -0.7% and 5% respectively. More importantly, when looking at application IPC performance improvement, ReMAP-best-VB achieves an average of 4.6% performance gain across the SPEC2006 applications while MLP-aware replacement and DRRIP see only 1.7% and 2.3% respectively. Here onward, to maintain generality I only discuss the results of ReMAP-16.

To illustrate the importance of considering the post eviction reuse distance and memory access cost in LLC management, I take a closer look at cactusADM. Figure 2.4 shows that all three replacement policies reduce the LLC miss count for cactusADM by 3-5%. However, because not all cache lines are equally important, the LLC miss count reduction does not translate to IPC performance improvement linearly. Figure 2.5 shows that ReMAP improves the performance of cactusADM by 2% while DRRIP and MLP-aware improve its performance by 0.1% and 0.5% respectively. The reason for the IPC performance gap can be explained by Figure 2.1. The figure shows that, for 40% of cache line evictions, a live line with higher memory access cost is chosen as the eviction candidate under LRU. In contrast, ReMAP is able to identify and prioritize cache lines with higher memory access cost over those with lower memory access cost.

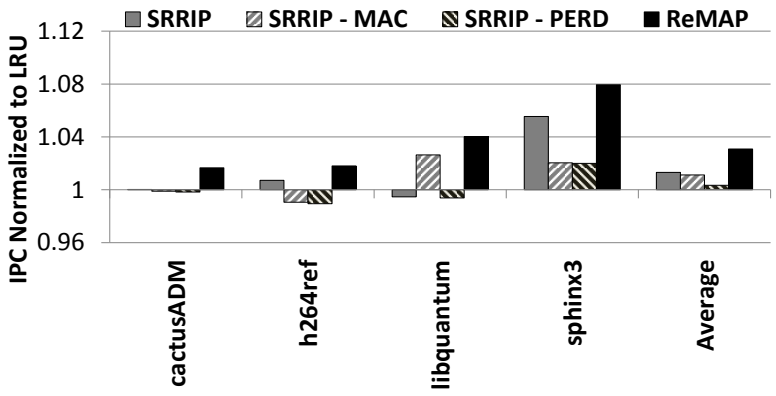


Figure 2.6: ReMAP: performance using PERD and MAC information in isolation.

ReMAP adopts a holistic approach towards LLC management and this is highlighted in the cases of libquantum and soplex. For both these applications all three policies achieve similar MPKI reduction. However, ReMAP achieved superior IPC improvement compared to DRRIP and MLP-Aware policies.

2.3.4 Benefit of Using PERD and MAC Information in Isolation

In order to understand the contributions of each of the individual components of ReMAP, i.e. the post eviction reuse distance and memory access cost, I study the performance benefit achieved by each component in isolation for a few interesting applications. Figure 2.6 shows the performance results of using PERD and MAC information in isolation, on top of SRRIP. SRRIP-PERD represents the setup where only PERD information is used to make eviction decisions along with the recency information. Similarly SRRIP-MAC represents the setup where in addition to the recency information, only MAC information is used to make eviction decisions. Figure 2.6 highlights the importance of the considering all three parameters, recency, PERD, and MAC, together while making the eviction decision.

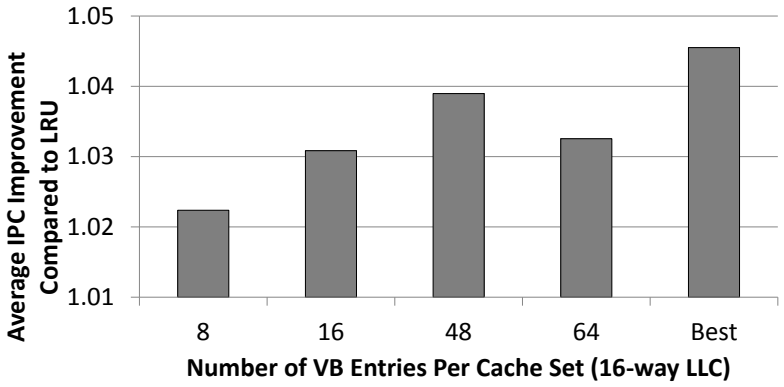


Figure 2.7: ReMAP:sensitivity to victim buffer storage.

2.3.5 Sensitivity to Victim Buffer Storage

The multi-level bloom filter based victim buffer consists of three stages of bloom filters cascaded together. It can be seen that the hardware overhead depends largely on the number of bit array size of the bloom filter. As the bit array size increases, the bloom filter false positive rate decreases. I observe that we can achieve a reasonable accuracy (false positive rate $\approx$ 1%) when the bit array size is 10$\times$ the number of entries in the victim buffer. Furthermore, as the number of entries in the victim buffer increase, we will be able to capture the reuse behavior more accurately. However if the number of entries is very large, the reuse behavior can be misguided and in turn hurt performance. I observe a similar trend as can be seen in Figure 2.7.

2.3.6 Sensitivity to System Parameters

ReMAP performance can be influenced by system parameters such as the baseline replacement policy and memory scheduling policy. In addition to RRIP, I conducted experiments with ReMAP built on top of the LRU policy. I observe that ReMAP shows similar performance improvement when the recency information is provided by LRU.

I also conducted studies to understand the sensitivity of ReMAP to architectural parameters such as cache sizes and cache set associativities. I conduct sensitivity studies with cache sizes from 1MB through 32MB, and associativities from 16-way, through 64-way configurations. I observe that ReMAP continues to provide significant performance benefit ranging from 2% to 7% on average and as high as 25% in case of benchmarks such as libquantum and mcf.

The performance of ReMAP can be sensitive to the underlying memory scheduling policy. Different memory scheduling policies can alter the MAC of the cache lines differently. Furthermore, estimating MAC under more sophisticated policies can be non trivial. In such cases, MAC value can be communicated from the main memory to the last level cache with negligible overhead on the bandwidth demand. I expect that the MAC information and PERD information will continue to be important pieces of information that can assist LLC management even under more sophisticated memory scheduling policies as well.

2.3.7 Multiprogrammed Workloads Results

I evaluate the heterogeneous multiprogrammed workloads for overall system throughput and fairness. I measure overall system throughput using the normalized average throughput and normalized average weighted speedup metrics. The normalized average throughput is given by $\frac{GM(IPC_{i\_policy})}{GM(IPC_{i\_LRU})}$, for (i = 0, 1, 2, 3). This metric indicates the overall throughput improvement of the system. I use the minimum normalized throughput achieved across all threads as a fairness metric. This metric gives a conservative measure of fairness of the new policy relative to the fairness of the baseline. This is given by $Min_i(\frac{(IPC_{i\_policy})}{(IPC_{i\_LRU})})$.

Figure 2.8 summarizes the normalized average throughput achieved by ReMAP, MLP-Aware [143], and TA-DRRIP [73] policies for different workload mixes. Across

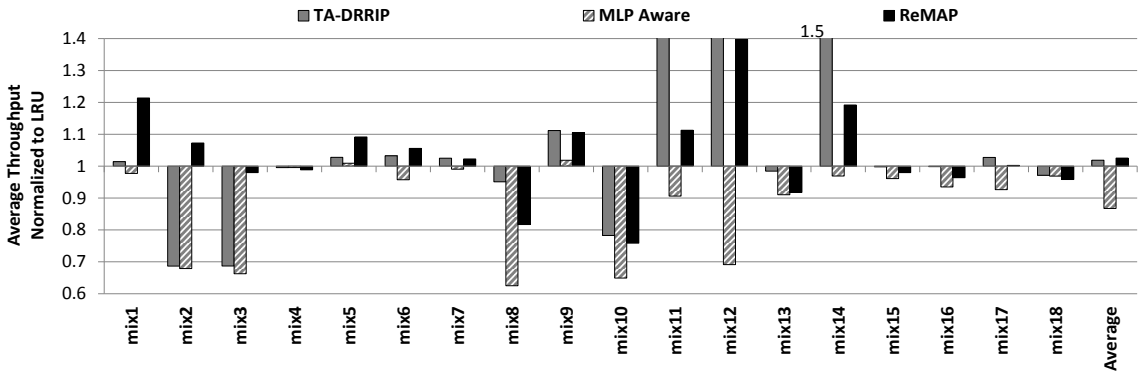


Figure 2.8: ReMAP: Performance for multiprogrammed workloads

all workload mixes, ReMAP improves average throughput by 2.5% compared to LRU. TA-DRRIP and MLP-Aware policies improve throughput by 1.8% and -14% respectively, compared to LRU.

Overall ReMAP performs better than both TA-DRRIP and MLP-aware policies on the fairness metric as well. ReMAP gives a normalized minimum throughput of 0.9 compared to LRU while TA-DRRIP and MLP-aware policies give 0.8 and 0.7 respectively.

2.4 Related Work

There has been a significant research effort directed towards the innovations in cache management research [32, 34, 44, 45, 58, 72, 82, 87, 110, 117, 142, 144, 177], I discuss prior works that closely resemble ReMAP in this section.

2.4.1 Reuse Distance Prediction

Jaleel et al. [74] proposed SRRIP and DRRIP to learn reuse behavior of applications and manage the last level cache accordingly. DRRIP provides both scan and thrashing resistance by performing set-dueling [142] between its two component poli-

cies, SRRIP and BRRIP. SRRIP provides scan resistance by inserting cache lines with "long" reference interval prediction. BRRIP on the other hand, provides both thrashing resistance by predicting "distant" re-reference interval most of the times and "long" re-reference interval infrequently. The recently proposed EAF-cache [153] predicts whether a cache line will receive reuse or not at the time of insertion, based on it's own past behavior. Though RRIP and EAF-cache predict the reuse behavior of cache lines, their predictions are limited to insertion time. On the contrary, ReMAP uses post eviction reuse distance prediction. This helps ReMAP to predict if a cache line will be recalled to the cache or not, and also how soon would a line be recalled once it is evicted from the cache.

Rajan et al. proposed shepherd cache [145] to emulate optimal replacement in the cache. They use four shepherd ways in a 16-way cache that will keep track of partial reuse distances and allowing to evict the the lines that are reused farther into the future. Their proposal allows for limited look ahead and ReMAP on the other hand is able to predict reuse distances much which are much farther, upto three times the associativity of the cache.

2.4.2 Dead Block Prediction

Many works have also used a variation of recency, instruction traces, or address traces to predict blocks that are dead [67, 85, 93]. Sampling Dead Block Prediction [86] proposed by Khan et al. predicts cache blocks that are dead in the cache based on the last touched instructions. They replace these predicted dead blocks prior to the LRU replacement candidate. Chaudhuri et al. proposed cache hierarchy-aware replacement (CHAR) [45] policy where they mined the private L2 cache eviction stream for information that identifies certain blocks to be dead and passed the hint to the LLC.

While ReMAP's PERD estimation is similar to dead block prediction, it is important to note that ReMAP predicts the reuse distance in a finer grainularity, and this chapter shows that the finer-grained PERD prediction contributes to further performance improvement.

2.4.3 Coordinating LLC Management with DRAM

Qureshi et al. first identified the potential in considering DRAM access costs in managing the LLC in [143]. They assign cost to each cache line based on the amount of memory level parallelism the cache line would present when it misses in the cache. They adopt a linear relationship between recency and the MLP cost of cache lines to determine the total cost. I present a more fine-grained memory access cost analysis and combine that with post eviction reuse distance along with recency information to assign effective costs to cache lines. This coordinated approach enables ReMAP to provide higher performance improvement than MLP-aware replacement.

2.5 Chapter Summary

The miss rate reduction achieved by most state-of-the-art cache management policies does not translate to corresponding IPC improvement at all times. This is because of the wide disparity in memory access costs experienced by different LLC misses. Hence, it is vital to manage the last level cache while considering memory access behavior of cache lines. Furthermore, the system performance is understandably affected by the current and future reuse of cache lines. With this insight, in this chapter, I proposed ReMAP, a reuse and memory access cost aware eviction policy that takes recency, post eviction reuse distance, and memory access costs to make better-informed eviction decisions at LLC. ReMAP is able to preserve most valuable cache lines, i.e lines that have near reuse and high memory access cost, in the LLC

and thereby providing superior performance. I demonstrate with extensive performance evaluation using wide variety of workloads that ReMAP performs consistently better than related state-of-the-art policies such as MLP-aware replacement, DRRIP and TA-DRRIP. ReMAP improves performance by as much as 13% and on average by 6.5% over the baseline.

Chapter 3

INSTRUCTION AND MEMORY DIVERGENCE BASED CACHE MANAGEMENT FOR GPGPU SYSTEMS

In Chapter 2, I described ReMAP, a novel cache management technique for last level caches (LLCs). ReMAP combines recency, future reuse and memory access cost information together to perform intelligent eviction decisions. By doing so, ReMAP is able to take into account the interaction between LLC and the next level of memory, the DRAM, and achieve high system performance. However, as accelerated computing becomes prevalent, the memory subsystem of general purpose graphics processing units (GPGPUs) also becomes a key factor affecting system performance. ReMAP and other techniques developed for CMPs, although applicable, might lead to sub-optimal performance in case of GPGPUs. Due to the GPGPU execution model, its performance bottlenecks differ significantly from that of the CMPs. Prior work has shown that the constrained data cache capacity is a key problem leading to sub-optimal performance in GPGPUs. In this chapter, I demonstrate that the constrained cache capacity problem can be effectively alleviated by using GPU specific information such as degree of memory divergence to predict both temporal and spacial locality behaviors.

3.1 Background and Motivation

GPGPUs execute instructions in the single instruction multiple thread (SIMT) manner. That is, multiple threads execute the same instruction on different data concurrently. For example, in recent NVIDIA GPUs, 32 threads are grouped into a single "warp" (wavefront in AMD terminology) for execution concurrently. In case

of memory load / store instructions, every single thread in a warp requests for its own piece of data from the memory subsystem. This places immense pressure on the memory subsystem of the GPGPUs. To mitigate this pressure, GPGPUs employ coalescers that try to merge multiple requests from a single warp into as few memory requests as possible. In the optimal case, all 32 requests originating from a warp, coalesce to present a single request to the memory system. However, as GPGPU applications get more and more general-purpose, the coalescer is often unable to coalesce all the requests originating from a warp to a single memory access. This situation is referred to as memory divergence. In the worst case, memory divergence could result in up to 32 individual cache lines being demanded and brought into the cache.

Furthermore, GPGPUs employ massive multithreading and fast context-switching to hide long memory access latencies. That is, whenever a warp experiences a long latency memory access, it is switched out of execution and another ready warp is switched into execution. This allows GPGPUs to achieve high pipeline utilization and consequently, high throughput. On the flip-side, if multiple warps place their memory requests concurrently, this results in a high bandwidth demand from the GPGPU memory subsystem. Thus, the massive multithreading execution model and the presence of memory divergence result in a few key challenges in the memory subsystem design of GPGPUs.

On the one hand, due to the large number of memory requests that are presented to the GPGPU memory systems, the level one (L1) data caches in GPGPUs tend to get thrashed often. On the other hand, GPGPU applications experience minimal spatial locality and spatial utilization of cache lines. These factors make the data caches and the interconnect significant performance bottlenecks in GPGPUs. In this chapter, I propose Instruction and memory Divergence based Cache, ID-Cache [28]

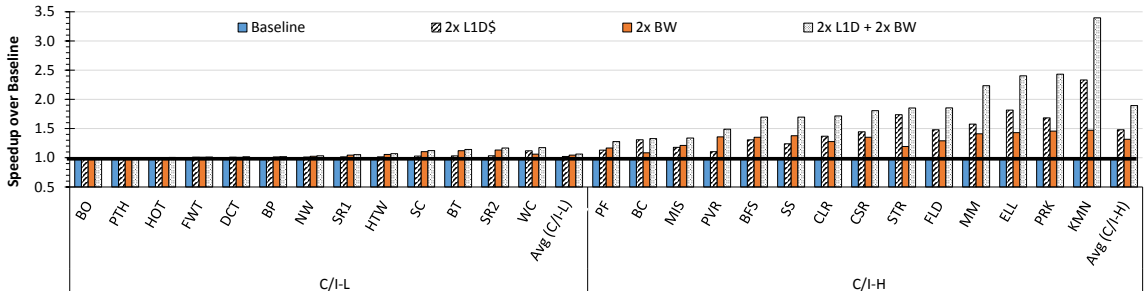


Figure 3.1: GPGPU application performance sensitivity to data cache capacity and interconnect bandwidth. The applications are categorized into two groups based on whether they are sensitive to data cache capacity and interconnect bandwidth or not[1].

that alleviates both the data cache capacity problem and interconnect bandwidth problem faced by today's GPGPUs.

3.1.1 Application Sensitivity to Cache Capacity and Interconnect Bandwidth

As mentioned above, GPGPUs adopt the massive multithreading execution model. For example, on the Kepler and Maxwell architectures released in 2012 and 2014 [125, 127], more than 2000 concurrent threads are supported on a streaming multiprocessor. These threads share a 16KB L1 data cache resulting in as few as 8B data capacity per thread. Similarly, the massive multithreading operation also puts immense pressure on other parts of the memory system, such as the on-chip and off-chip interconnect bandwidth. This makes data cache capacity and the on-chip interconnect, critical resources for GPGPUs.

To understand the impact of cache capacity and interconnect bandwidth on GPGPU applications, I conduct a sensitivity study. Figure 3.1 shows the performance sensitivity of GPGPU workloads when L1 data cache capacity and interconnect bandwidth

are increased. The x-axis represents the wide variety of GPGPU applications studied in this chapter and the y-axis represents the execution time speedup normalized to the baseline configuration (16KB L1 data cache with 32B Flits interconnect configuration). The first bar represents the baseline configuration (always 1). The second and third bars represent settings where the data cache capacity and interconnect bandwidth are doubled separately, 2x L1D$ and 2x BW (with 64B Flits interconnect configuration) respectively. Finally, the last bar represents a configuration where both the data cache capacity and interconnect bandwidth are doubled together (2x L1D$ + 2x BW).

The applications classified under the label C/I-L are insensitive to data cache capacity and interconnect bandwidth and experience negligible performance impact with the increase in cache capacity or bandwidth. The C/I-H applications on the other hand are highly sensitive to data cache capacity and interconnection bandwidth. For these applications, 89% performance improvement is obtained when both the cache and interconnection bandwidth capacities are doubled.

Data streaming applications are more sensitive to the bandwidth than cache capacity. Streaming applications such as `SC`, `SR2`, and `PVR` receive negligible performance improvement with the 2x L1D$ configuration while receiving significant speedup with the 2x BW configuration shown in Figure 3.1. On the other hand, applications that possess cache friendly access patterns benefit significantly from the increased cache capacity. Applications such as `BC` and `STR` in Figure 3.1 exemplify this scenario. Nonetheless, most applications in the C/I-H category are sensitive to both data cache capacity and interconnect bandwidth resources. Since simply increasing cache and

[1] The details of simulation infrastructure, workload selection and classification are described in Section 3.3

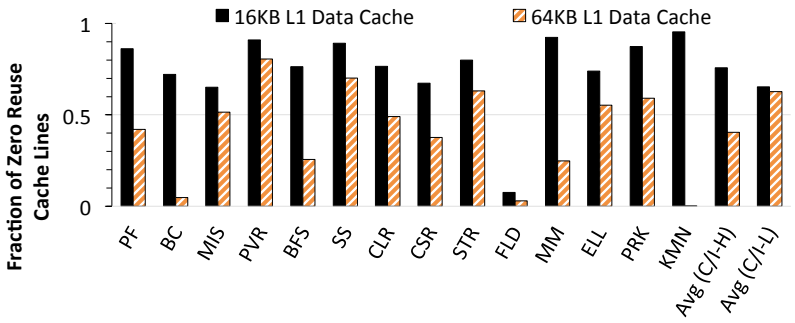


Figure 3.2: The fraction of zero reuse cache lines in the baseline 16KB and 4x larger 64KB data cache configurations. Zero reuse cache lines are cache lines that are brought to the cache but never reused.

bandwidth capacities is not always a viable option, there is a need for simple designs to optimize the utilization of these two critical resources.

3.1.2 Inefficient Cache Utilization in GPGPUs

To understand the utilization of the data caches in GPGPUs, I study the presence of "zero reuse" lines in the cache. Zero reuse lines are cache lines that are allocated in the cache but do not receive reuse before they are evicted. My study shows that in the baseline data cache, more than 70% of the lines allocated in the data cache turn out to be zero reuse lines. Motivated by similar observations, recent prior works such as [75, 76, 175], have pointed out that employing cache memories in GPGPUs may rather degrade performance significantly for some GPGPU workloads due to pipeline stalls incurred by resource contention (e.g., MSHR entries) and additional queuing latencies introduced by unnecessary data traffic.

Although the above observation might seem to indicate that the GPGPU applications do not possess data locality that can be exploited by the caches, closer analysis suggests otherwise. Figure 3.2 shows the comparison of the fraction of zero reuse cache lines in the default 16KB L1 data cache (black bars) and that in a 4x larger

64KB cache (orange diagonal bars). On average, the fraction of zero reuse lines reduces from 75% to 40% for the C/I-H benchmarks. The large disparity between the number of zero reuse cache lines in the two configurations illustrates that data locality exists in GPGPU workloads but the baseline architecture is unable to effectively capture it. The main reason is the commonly-observed mixed reuse patterns in GPGPU applications. That is, due to massive multithreading execution, lines that possess excellent reuse behavior are often interleaved with lines that do not. This results in lines being evicted before they can be reused, wasting precious cache capacity and interconnect bandwidth. Therefore, we need a way to accurately separate the cache lines that possess reuse from the ones that do not. *By accurately predicting and bypassing the cache lines that do not possess any reuse, GPGPU performance can be significantly improved.*

3.1.3 Inefficient Cache and Bandwidth Utilization due to Default Fixed Cache Line Size Configuration

Having highlighted the problem of zero reuse cache lines in Section 3.1.2, I now focus this section on another key contributor to sub-optimal memory system performance in GPGPUs. I observe that a large part of the bytes in the default 128B cache line remain unused. I refer to this problem as the low spatial utilization problem. Low spatial utilization results in wastage in precious cache capacity and bandwidth resources.

To understand the spatial utilization of cache lines, I analyze the fraction of bytes utilized in each cache line throughout the execution period of GPGPU applications. Figure 3.3 shows the utilization distribution of data in the cache line granularity for the C/I-H GPGPU applications. From Figure 3.3, we can observe that while the cache line spatial utilization is fairly high for C/I-L applications, for C/I-H applications,

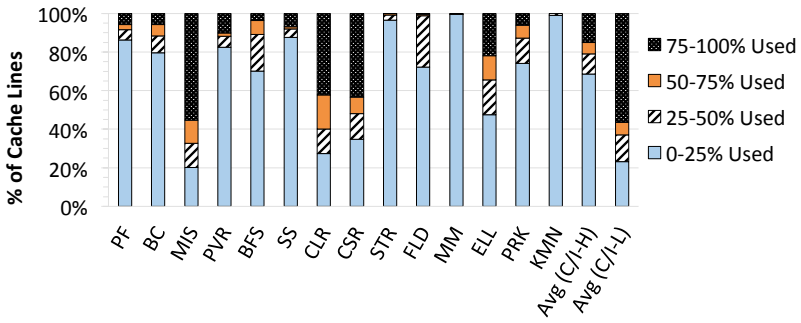


Figure 3.3: The distribution of spatial utilization of cache lines. The stacks represent different spatial utilization categories measured at the granularity of 32B and the y-axis represents the percentage of cache lines that belong to a particular utilization category, e.g., "0-25% Used" represents the fraction of cache lines with 0-25% spatial utilization.

an opposite behavior is observed—more than 70% of cache lines have less than 25% spatial utilization. For an overwhelming majority of the cache lines, at least 75% of the data is brought to the cache and is stored unnecessarily. This inefficient use of cache and bandwidth capacities presents an opportunity for performance optimization for GPGPUs.

One way to minimize cache capacity and bandwidth wastage due to low spatial utilization is to reduce the cache line size. For the same purpose, the L1 data cache can even be turned off.Figure 3.4 shows the performance of GPGPU applications with different cache line sizes and with the L1 data caches turned off. Each of the configuration shows bipolar behavior across the different kinds of workloads. For example, the 32B line size configuration provides performance speedup that varies from -50% (`FWT`) to as much as 3.7X (`KMN`). Similar behavior can be observed for the 64B cache line size configuration as well as when L1 data cache is turned off. This performance variability is undesirable.

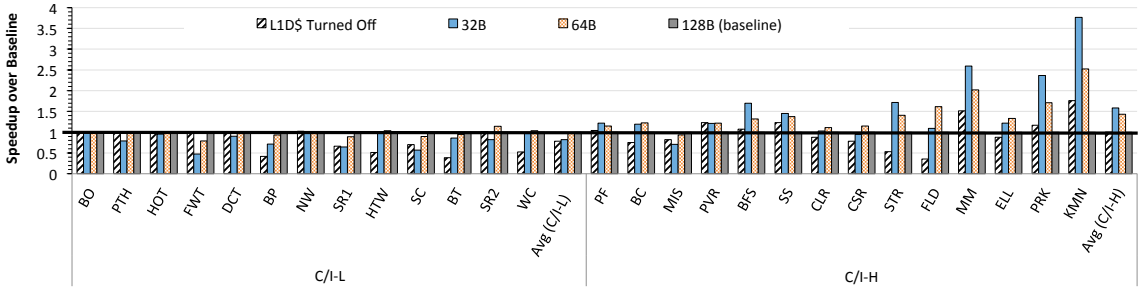


Figure 3.4: The performance of GPGPU applications with different L1 cache line sizes and turning off L1 data caches.

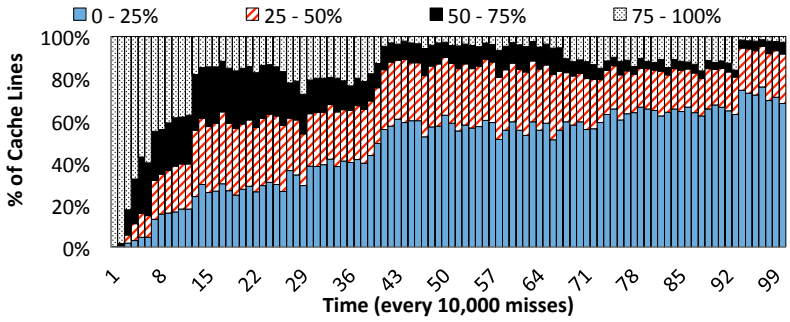


Figure 3.5: Cache line spatial utilization over time (an example from PVR).

The performance variability arises from the fact that the spatial utilization of cache lines varies significantly from one application to another, as seen in Figure 3.3. Furthermore, as Figure 3.5 shows, the spatial utilization of cache lines can vary significantly within an application over its execution as well. Therefore, there is not a simple way to select a cache line size that minimizes cache capacity wastage and bandwidth usage while ensuring there is no performance degradation. Therefore, *there is a need for a* **dynamic** *method that predicts and inserts cache lines with an optimal line size configuration.*

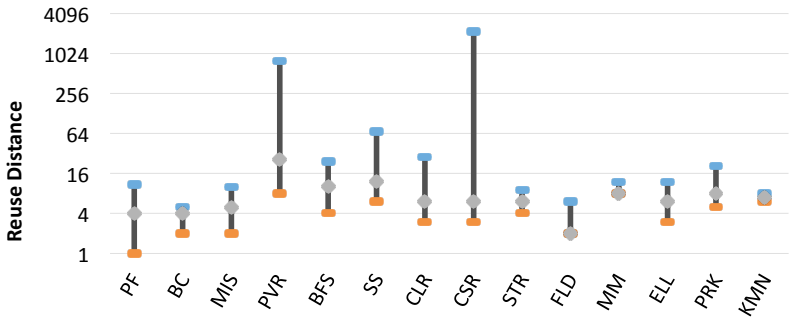


Figure 3.6: The distribution of L1 data cache reuse distances seen in GPGPU benchmarks. The grey diamonds represent the median reuse distances. The high-low bars represent 75th and 25th percentile reuse distances.

3.2 ID-Cache Design and Implementation

I focus this section on designing a dynamic method, ID-Cache that addresses the cache thrashing and low spatial utilization problems highlighted in Sections 3.1.2 and 3.1.3.

3.2.1 Towards Effective Cache Bypassing

To further investigate the memory reuse patterns, I analyze the reuse distances seen in GPGPU applications. Figure 3.6 shows the distribution of reuse distances for all memory requests in the C/I-H category. The median reuse distance is depicted by the diamonds whereas the 75th and 25th percentile reuse distances are shown by the high and low markers respectively. We can observe that across the wide variety of applications, the reuse distance of data is extremely diverse. Due to the long reuse distances, some GPGPU applications might appear as streaming applications under the default cache configuration e.g., `PVR` and `CSR`. Furthermore, Figure 3.6 demonstrates the mixed reuse behavior discussed above. That is, for many applications, some of the requests have reuse distances less than four which is the associativity

of the baseline data cache, while other requests do not. Under such a scenario, the requests that have low reuse distance could experience interference from other requests that have long reuse distances, resulting in a loss of locality and higher cache misses. Therefore, it is important to protect these low reuse distance requests from the interference originating from other long reuse distance requests. This turns out to be a non-trivial task in the massive multithreading operating paradigm.

Using Instruction Program Counter to Classify Requests

The problem of identifying, separating, and protecting memory references that have excellent locality from other references that do not, has been investigated in the context of CMP caches. A common piece of information that has been well explored for this purpose is the instruction program counter (PC) [85, 86, 93, 113, 166, 172] All of the above works exploit the property that the reuse behavior, or the lack thereof, of cache lines brought by a particular load is typically homogeneous. This property is a result of the typical features of structured programming. For example, during each iteration of a loop, a load instruction will access data that is indexed by some function of the iteration number in a strided fashion. If this access stride is smaller than the cache line size, all the cache lines brought by this load will receive a reuse hit due to spatial locality. Similarly, if we consider any producer-consumer relationship, an instruction that is part of the producer would typically bring a cache line into the cache and update it before the data is used by the consumer. This process could result in cache hits due to temporal locality. On the other hand, if the access stride is too long, or if there is a large number of conflicting accesses between when a piece of data is produced and consumed, the data that is brought to the cache will not be reused.

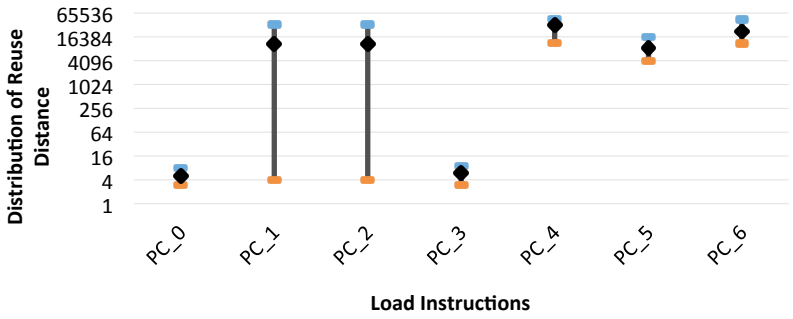


Figure 3.7: The distribution of L1 data cache reuse distance with different load instructions in ELL. The grey diamonds represent the median reuse distances. The high-low bars represent 75$^{\text{th}}$ and 25$^{\text{th}}$ percentile reuse distances.

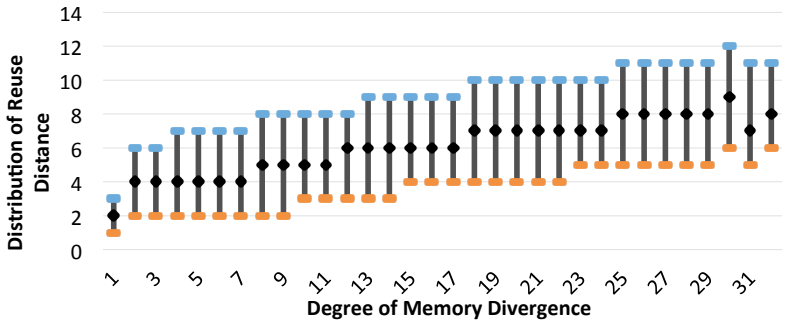


Figure 3.8: The distribution of L1 data cache reuse distance with degree of memory divergence of a load instruction, i.e., ELL's PC_3. The grey diamonds represent the median reuse distances. The high-low bars represent 75$^{\text{th}}$ and 25$^{\text{th}}$ percentile reuse distances.

Following the same insight, a few recent works in GPGPU cache designs have also proposed to use instruction specific information to identify the reuse behavior of different load instructions. Many of these works use compiler analysis to learn the reuse behavior of specific load instructions [75, 108, 175]. These works either require significant support from the compiler or offline profiling to work well. The adaptive bypassing method proposed by Tian et al. [165] utilized the *last touched PC*

information to predict zero reuse cache lines and bypass them from the cache. Their design achieves a modest 3% performance improvement.

To understand how PC information can potentially be used to segregate memory references in C/I-H kind of GPGPU applications, I analyze the reuse distance distribution of memory references from the different PCs. Figure 3.7 shows the reuse distance distribution of memory references from the different PCs in an example application, `ELL`. From the figure, we can make a couple of important observations. The reuse distances of PCs in an application vary widely. A significant part of memory references brought in by a small set of PCs have long reuse distances, e.g., PC_4, PC_5, and PC_6. Such reuses cannot be captured by caches of realistic sizes and hence should be bypassed from the cache. On the other hand, the applications also possess certain PCs whose memory requests have short reuse distances which could potentially result in cache hits. Memory references from such PCs should be preserved in the cache.

Furthermore, while memory references associated with a particular PC exhibit "similar" reuse distance for a majority of PCs in an application, there are other PCs that do not follow this pattern, e.g., PC_1 and PC_2 in `ELL`. For these instructions, some references exhibit short reuse distances that could result in cache hits, while others, although in the same PC bucket, exhibit long reuse distances. Therefore, *we need additional information to achieve a finer resolution in segregating memory references, such that, references that can potentially receive reuse hits are protected in the cache while others are bypassed more intelligently.*

Using Memory Divergence Behavior to Classify Requests

Memory divergence is a property unique to GPGPU applications, that could have a pronounced impact on the reuse behavior of an application. As the degree of

memory divergence increases, the number of memory requests sent to the cache also increase. Consequently, the reuse distances of lines inserted by divergent instructions are typically longer. I use this insight to segregate memory references that exhibit short reuse distances from others that do not. Specifically I analyze the relationship between reuse distance and degree of memory divergence of load instructions. Figure 3.8 shows the reuse distance distribution for memory references from a specific PC in `ELL`. We can see that with the help of the memory divergence information, we can further separate memory references in a finer granularity such that we can identify specific memory references that have similar reuse distances. That is, in the example of `ELL`'s PC_3 (Figure 3.8), we observe that a memory load instruction with a high degree of memory divergence, e.g., PC_3 that generates more than 15 requests in the case of `ELL`, typically all have reuse distances greater than four, i.e., the set associativity of the baseline cache. Such requests are unlikely to receive reuse hits in the cache and therefore are candidates for bypassing.

3.2.2 PC and Memory Divergence Pattern Guided Bypassing

The observations made in the previous sections motivate the exploration of using PC and the memory divergence patterns to manage cache bypassing in order to improve cache utilization in GPGPUs. I design and evaluate two simple bypassing techniques — a PC (PC-only) based method and a combined PC and degree of divergence (PC+Div) based method.

Offline-Trained Bypassing: To quantify the performance potential enabled by the consideration of the PC and memory divergence information, I obtain the reuse distance distribution for PC-only and PC+Div offline. Then, in the second pass, I use the reuse distance information obtained to guide the cache line bypassing decision. To

compensate the effect of bypassing on the reuse distance distribution obtained offline, the bypassing decision is made when the reuse distance of a PC is larger than 8 for the 4-way set associative L1 data caches. Then, for the PC+Div bypassing technique, a second filter is applied by using the instruction's degree of memory divergence. That is, for each PC, if the reuse distance for memory references with a particular degree of memory divergence d, is higher than 6, then the future memory references from that PC, having degree of memory divergence greater than d are bypassed from the cache.

Dynamic Bypassing: For the PC-only and PC+Div based bypassing techniques to be practical, the reuse distance information must be learned dynamically at runtime. To do so, I design a 128-entry reuse distance prediction table of saturating counters for the PC-only scheme. In case of the PC+Div scheme, for each PC, I use four bins to learn the reuse behavior of instructions with different divergence degrees. Therefore, I use a 512-entry reuse distance prediction table of saturating counters for the PC+Div scheme. The table learns and predicts a PC's reuse characteristics in a similar manner as a recent prior work [172]. The prediction table is indexed by the lower 7 bits of an instruction PC and the entry value indicates the predicted reuse behavior of the instruction. Algorithm 3 describes the learning and prediction steps of the design.

Having addressed the cache thrashing problem with PC+Div based bypassing, I focus the next section to address the low spatial utilization problem in GPGPUs.

3.2.3 Towards Efficient Cache Line Size Selection

I first analyze the sources of cache line spatial utilization. In a GPGPU system, the data within a cache line could be consumed in two ways: 1) Future reuses to the cache

Input: request,access.status

if $PC - only$ **then**

  $index = hash(request.Inst.PC)$;

end

else

  $index = hash(request.Inst.PC, request.Inst.deg\_divergence)$;

end

if $access.status = MISS$ **then**

  if $table[index] > 0$ **then**

    $BypassDecision = INSERT$;

  end

  else

    $BypassDecision = BYPASS$;

  end

end

else if $access.status = HIT$ **then**

  $table[index] = table[index] + 1$;

end

else if $access.status = EVICT$ **then**

  if $Evicted\_line.Reused = False$ **then**

    $table[index] = table[index] - 1$;

  end

end

Algorithm 3: The reuse prediction algorithm for PC-only and PC+Div based bypassing.

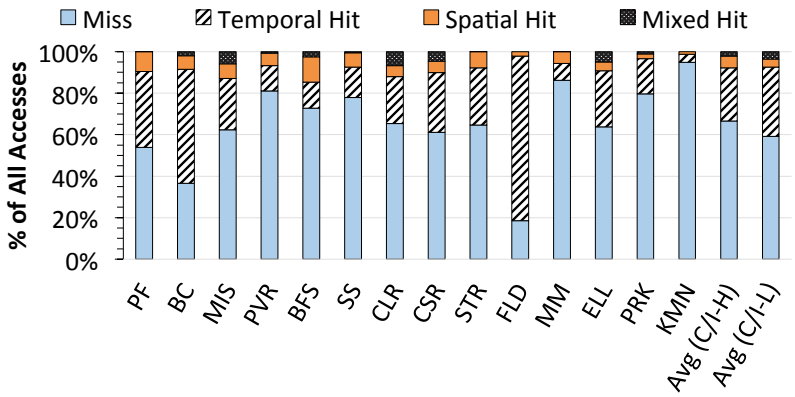


Figure 3.9: The distribution of cache hits and misses in GPGPU applications.

line consume data that was not consumed upon the line's insertion (traditionally, spatial locality) 2) Requests from multiple threads in a warp are coalesced together to consume adjacent data upon a line's insertion. To delve deeper, I look at the significance of each of these in GPGPU applications.

In Figure 3.9 I show a breakdown for all cache accesses by separating them into misses, temporal hits, spatial hits and mixed hits. Temporal hits signify the situation where all data that is reused on a particular access have been touched before. Spatial hits on the other hand represent the hits where all data that is being touched for the first time. Similarly, mixed hits refer to the case where a part of the data that is being reused has been touched before and the rest has not. From Figure 3.9, we can observe that the fraction of accesses that result in hits due to spatial locality (spatial hits and mixed hits) are a mere 7% on average for both C/I-L and C/I-H workloads, respectively. That is, the amount of spatial locality that is exploited in a GPGPU system is minimal.

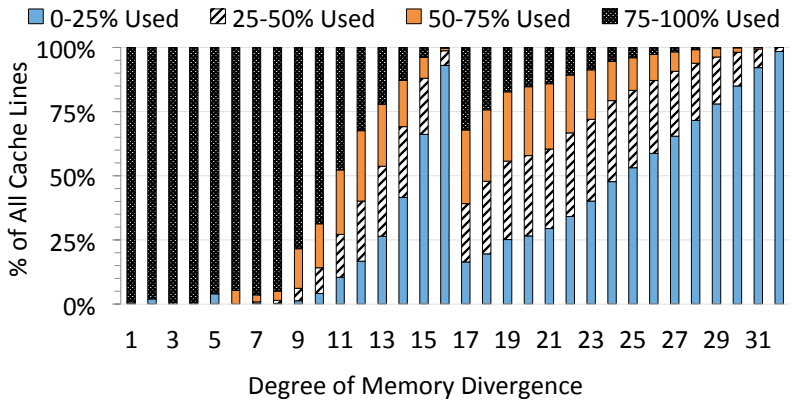


Figure 3.10: The distribution of L1 data cache utilization vs degree of memory divergence (x-axis) for `PVR` application. The stacks represent different spatial utilization categories measured at the granularity of 32B and the y-axis represents the percentage of cache lines that belong to a category.

High Correlation between Cache Line Spatial Utilization and Memory Divergence Patterns

Although the coalescer attempts to combine requests from multiple threads that access adjacent data together, it might not always be successful in doing so. This results in memory divergence. The degree of memory divergence is a property of GPGPU applications that affects the spatial utilization of cache lines significantly. For example, when the degree of memory divergence is one (i.e., a convergent instruction) and each thread accesses 4B data, all 128 bytes of the cache line are utilized, leading to 100% spatial utilization. However, on the other extreme, when the degree of memory divergence is 32, 1 to 8 bytes (depending on the access data type) of the cache line would be used, leading to low spatial utilization.

I investigate the relationship between cache line spatial utilization and the memory divergence patterns by analyzing the variation of spatial utilization for cache lines that are inserted by instructions of different degrees of divergence. Figure 3.10 shows the

variation of spatial utilization of cache lines with varying degree of memory divergence for one example application — PVR. We notice that most of the cache lines brought to the cache by instructions having lower degrees of divergence (e.g. 1-8) have higher spatial utilization. On the other hand, most of the cache lines that are brought in by instructions with high degrees of divergence have much lower utilization. It is apparent that *the spatial utilization has a fairly predictable behavior with respect to the degree of divergence* [2] . Therefore, I leverage this piece of information to dynamically optimize both cache capacity utilization and bandwidth consumption.

3.2.4 Divergence Guided Adaptive Line Size Insertion (ALSI)

Based on the observations made in the previous sections, I arrive at the intuition that both cache capacity and interconnect bandwidth utilization can be optimized together by inserting cache lines of different sizes based on their spatial utilization. A similar intuition was used by Rhu et al. to optimize for bandwidth consumption [146]. I will highlight the differences between their work and ours in detail in Section 3.4. In order to store data of variable size granularities, e.g., 32B, 64B, or 128B, I modify the L1 data cache architecture. I use a previously-proposed cache architecture, called Amoeba cache [91]. Amoeba cache is a cache architecture proposed for CMP LLCs that treats each cache set as an array of small blocks (8B size each) that can be used to hold either tag or data information. Therefore, a cache line of any size can be held in the cache using a set of contiguous blocks.

I design a simple divergence based approach for Adaptive Line Size Insertion (ALSI). I modify the cache line size dynamically at runtime based on the degree of memory divergence. Specifically, I assume that as the degree of memory divergence

[2] I also evaluate the correlation between PC and spatial utilization and find the degree of low spatial utilization to be more closely related to the degree of memory divergence.

Input: request

if $request.Inst.divergence\_deg = 1$ **then**

  $LineSizeDecision = 128B$;

end

if $request.Inst.divergence\_deg < 4$ **then**

  $LineSizeDecision = 64B$;

end

else

  $LineSizeDecision = 32B$;

end

Algorithm 4: Algorithm for Divergence Based Adaptive Line Size Selection (ALSI).

increases, the spatial utilization of cache lines reduces and therefore cache lines are inserted using a smaller line size configuration, e.g., 32B. On the other hand, when the load instruction is convergent, the spatial utilization of cache lines is likely to be 100% and therefore cache lines are inserted using the 128B line size configuration. The algorithm for ALSI is described in Algorithm 4.

3.2.5 ID-Cache: Instruction and Divergence Based Cache Management

Thus far, I describe two designs—PC+Div-based bypassing and adaptive line size insertion (ALSI)—that improve the efficiency of the memory subsystem by minimizing zero reuse lines (Section 3.2.2) and increasing spatial utilization (Section 3.2.4), respectively. Since cache bypassing and variable line size insertion are closely related to each other, I integrate the two designs together to jointly optimize the performance of the memory subsystem. I propose **ID-Cache** to improve the performance of the L1 data caches and the interconnect bandwidth utilization. ID-Cache is a simple design that optimizes the GPGPU performance by using instruction-related

Input: request, access

$Bypass = BypassDecision$;

//Predict whether to bypass or not based on Algorithm 3

if $Bypass = TRUE$ **then**

    //Take bypass path for this access

    $access.bypass = TRUE$;

end

else

    //Insert line to cache

    //Predict Line Size according to ALSI Algorithm 4

    $LineSize = LineSizeDecision$;

    //Complete cache line insertion with predicted line size

    $access.line\_size = LineSize$;

end

Algorithm 5: Algorithm for ID-Cache Bypass and ALSI selection logic.

information, i.e., the reuse distance characteristics and memory divergence behavior of instructions. ID-Cache improves the cache capacity utilization by bypassing memory requests from instructions that generate long reuse requests and have high degree of memory divergence. This component is the PC+Div based bypassing design described in Section 3.2.2 and is called ID-Cache Bypass. Furthermore, for lines to be inserted into the cache, ID-Cache uses the degree of memory divergence to determine the size configuration and thereby improves the utilization of precious cache capacity and the interconnect bandwidth. This component is the ALSI design described in Section 3.2.4. The pseudo-code implementation of ID-Cache is described in Algorithm 5.

Table 3.1: ID-Cache: GPGPU-sim simulation configurations.

Architecture	*NVIDIA Fermi GTX480*
Num. of SMs	*15*
Max. # of Warps per SM	*48*
Max. # of Blocks per SM	*8*
# of Schedulers per SM	*2*
# of Registers per SM	*32768*
Shared Memory	*48KB*
L1 Data Cache	*16KB per SM (32-sets/4-ways), LRU*
L1 Inst Cache	*2KB per SM (4-sets/4-ways), LRU*
L2 Cache	*768KB unified cache (64-sets/16-ways/6-banks), LRU*
Min. L2 Access Latency	*120 cycles*
Min. DRAM Access Latency	*220 cycles*
Warp Size (SIMD Width)	*32 threads*
Warp Scheduler	*GTO [147]*

3.3 Evaluation and Analysis

3.3.1 Simulation Infrastructure

I use GPGPU-sim simulator (version 3.2.2) [31] to characterize the behavior of the GPGPU memory subsystems. GPGPU-sim is a cycle-level performance simulator that models a general-purpose GPGPU architecture. I utilize GPGPU-sim's default configuration representing the NVIDIA Fermi GTX480 architecture [123]. Table 3.1 shows the detailed configuration of my experimental setup.

3.3.2 Workload Construction

I select a broad set of GPGPU applications from the Mars [65], NVIDIA SDK [124], Pannotia [46], and Rodinia [47, 48] benchmark suites to represent the diverse behavior present in GPGPU workloads. I utilize these GPGPU applications to quantify and evaluate the efficiency of memory subsystem designs. I classify the applications into two categories based on the speedup achieved when both the L1 data cache capacity and interconnect bandwidth are doubled — cache/interconnect insensitive (C/I-L) (speedup < 1.2x), and highly cache/interconnect sensitive (C/I-H) (speedup > 1.2x). Table 4.3 lists the details of these benchmarks and their input data sets. I simulate all benchmarks except three, to completion [3] . I present detailed characterization and analysis for the C/I-H benchmarks throughout this chapter, while only presenting results for C/I-L benchmarks when necessary for completion.

Table 3.2: ID-Cache: GPGPU Benchmarks.

Abbr	Application	Input	Cat.
BO	Binomial Options [124]	512 Options	C/I-L
PTH	Path Finder [47]	100k nodes	
HOT	Hotspot [47]	512x512 nodes	
FWT	Fast Walsh Trans. [124]	32k samples	
DCT	Discreet Cosine Trans. [124]	10 blocks	
BP	Back Propagation [47]	65536 nodes	
NW	Needleman-Wunsh [47]	1024x1024 nodes	
SR1	SRAD1 [47]	502x458 nodes	
HTW	Heartwall [48]	656x744 AVI	

[3] To keep the simulation times manageable, I restrict CSR, ELL and KMN to one billion instructions.

SC	Streamcluster [47]	32x4096 nodes	
BT	B+Tree [47]	1M nodes	
SR2	SRAD2 [47]	2048x2048 nodes	
WC	Word Count [65]	86kB text file	
PF	Particle Filter [47]	28x128x10 nodes	C/I-H
BC	Betweenness Centrality [46]	1K (V), 128K (E)	
MIS	Maximal Ind. Set [46]	ecology	
PVR	Page View Rank [65]	1M data entries	
BFS	Breadth First Search [47]	65536 nodes	
SS	Similarity Score [65]	1024x256 points	
CLR	Graph Coloring [46]	ecology	
CSR	Dijkstra-CSR [46]	USA road NY	
STR	String Match [65]	165k words	
FLD	Floyd Warshall [46]	256(V), 16K (E)	
MM	Matrix Multiplication [65]	1024x1024	
ELL	Dijkstra-ELL [46]	USA road NY	
PRK	Pagerank (SPMV) [46]	Co-Author DBLP	
KMN	K-Means [47]	494020 objects	

Next, I present detailed simulation results evaluating ID-Cache and its component policies, PC+DIV-based bypassing, and Adaptive Line Size Insertion.

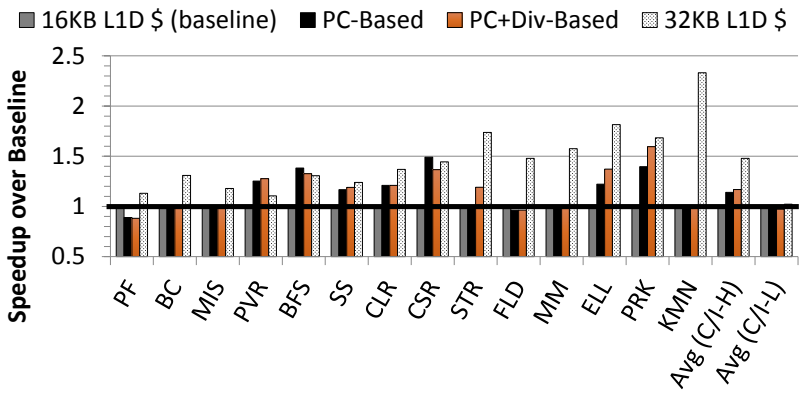


Figure 3.11: Performance of GPGPU applications with **offline trained** PC-Based and PC+Div-Based bypassing designs.

3.3.3 PC and Memory Divergence Pattern Guided Bypassing

Offline-Trained Bypassing: Figure 3.11 shows the performance of the PC-only and PC+Div based bypassing methods. The PC-only and PC+Div methods result in an average speedup of 14% and 17%, respectively, for C/I-H workloads. As can be expected, both techniques have negligible impact on the C/I-L workloads. When compared to a 32KB cache, the PC-only and PC+Div based methods can bridge the performance gap between 16KB and 32KB caches by 29% and 35%, respectively.

Adding an additional layer of information, namely, the memory divergence patterns, can help prune the incoming memory requests in a finer granularity. The benefit of doing so can be witnessed in workloads such as `STR`, `ELL` and `PRK`. For these workloads, PC+Div based method improves the performance by a significant extent than the PC-only method. In the case of `STR`, while the PC-only approach does not bypass any requests and performs exactly the same as the baseline, the PC+Div based approach bypasses 10% of the requests, translating to a significant 19% performance gain.

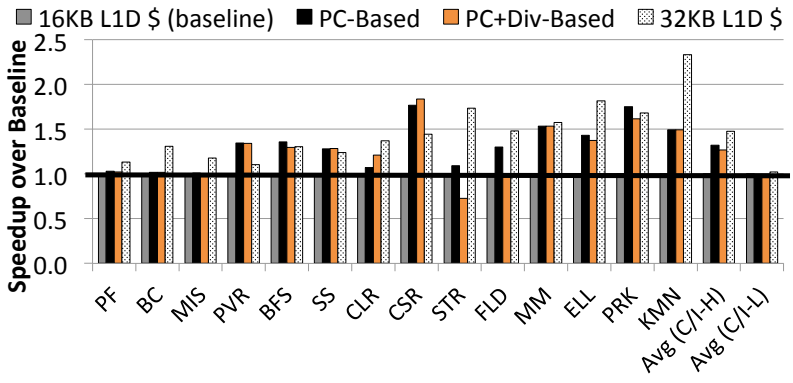


Figure 3.12: Performance of GPGPU applications with **online** PC-Based and PC+Div-Based bypassing designs. These designs require no offline training.

The performance of the PC-only and PC+Div based approaches also depends on the aggressiveness of bypassing that is carried out. This aggressiveness is dictated by the reuse distance thresholds used in the bypassing decision. If the aggressiveness is too low, then not many requests are bypassed from the cache and this results in applications performing exactly the same as the baseline, e.g., `MIS`, `FLD`, and `KMN` in Figure 3.11. On the other hand, if the aggressiveness is too high, it could result in useful requests being bypassed from the cache. Due to this reason, applications such as `PF` experience an 11% performance degradation with both the PC-only and PC+Div based methods.

Dynamic Bypassing: Figure 3.12 shows the performance improvement achieved by the online PC-only and PC+Div based bypassing designs described above. On average, the online PC-only and PC+Div-based designs improve performance by 29% and 22%, respectively for C/I-H workloads and have negligible impact on the C/I-L workloads. Since the online PC-only and PC+Div based designs can adapt to the runtime changes in reuse behavior of different load instructions, the online approaches perform better than the offline trained ones in Figure 3.11 for a number of

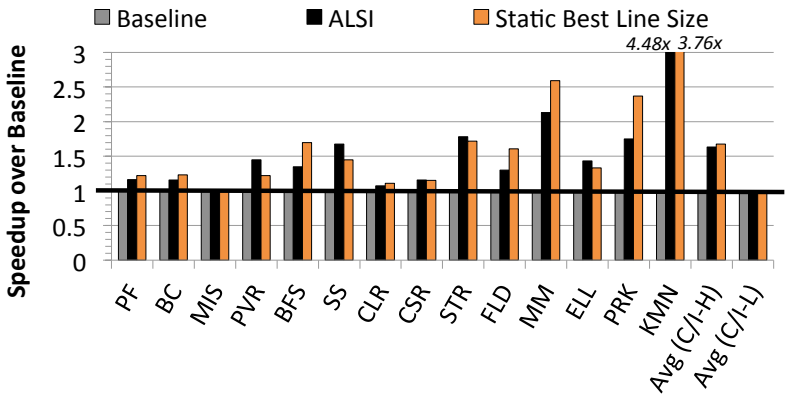


Figure 3.13: The performance improvement under Adaptive Line Size Insertion.

applications, such as PF, SS, CSR, and KMN. Furthermore, both the online PC-only and PC+Div based designs achieve more than 50% of the performance benefit brought by a 32KB cache for the C/I-H workloads.

The benefit from the degree of divergence information is more modest in the case of the online PC+Div based design. The workloads that received the highest benefits from the degree of divergence information in the offline trained approach (STR, ELL, and PRK) fail to benefit from the same information in the simple online design described here. A more advanced design that accurately captures the reuse distance behavior across the different divergence degrees is necessary.

3.3.4 Divergence Based Adaptive Line Size Insertion (ALSI)

In this section, I present evaluation results for ALSI described in Section 3.2.4. Figure 3.13 shows the performance improvement achieved by ALSI. On average, ALSI improves the performance of the C/I-H applications by 64% and does not affect/degrade the performance of the C/I-I applications. Furthermore, I compare the performance of ALSI to a static, best line size configuration i.e., a per-application line size configuration (32B, 64B, or 128B) which gives the best per-application per-

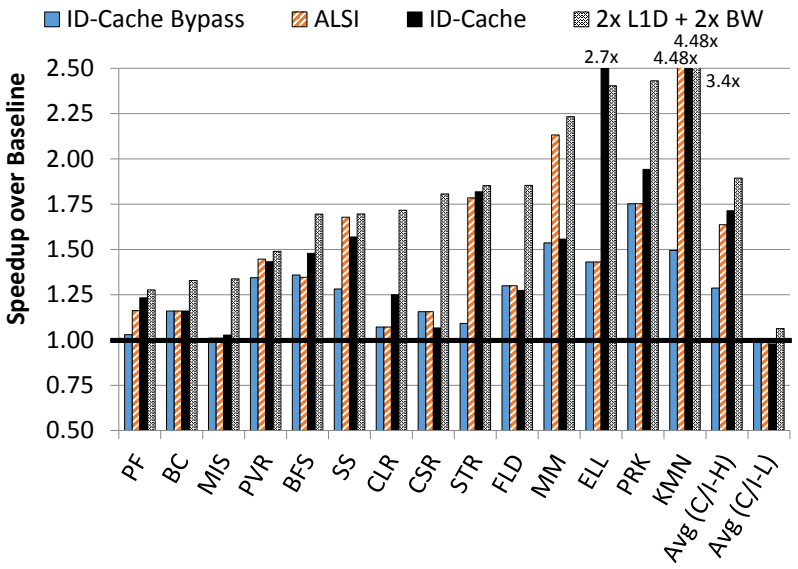


Figure 3.14: Performance improvement with ID-Cache.

formance. ALSI performs almost as well as the static best line size configuration and achieves 96% of the performance gain given by the optimal setting. This shows that degree of divergence information can be used effectively to predict the spatial utilization and hence the insertion cache line size. Furthermore, a runtime adaptive system such as ALSI would be able to capture the change in spatial utilization over different application phases. This results in ALSI outperforming the static best line size configuration for a number of workloads such as `PVR`, `SS`, and `KMN`.

3.3.5 ID-Cache - Instruction and Divergence Based Cache Management

ID-Cache is composed of its component policies PC+DIV based bypassing and adaptive line size insertion as described in Section 3.2.5. Overall, when compared with the baseline architecture, ID-Cache achieves an average of 71% performance improvement for the cache and bandwidth capacity sensitive workloads, as Figure 3.14 shows. The significant performance gain from ID-Cache matches 90% of that from a GPU with doubled cache and bandwidth capacities (2x L1D$ + 2x BW). Figure 3.14

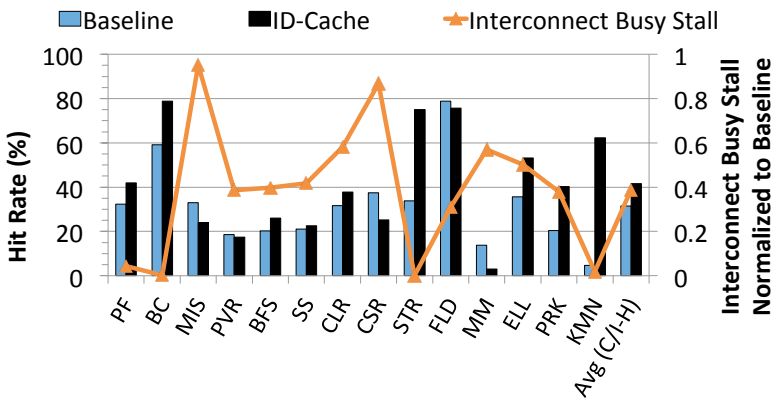


Figure 3.15: L1 data cache hit rate improvement and interconnect busy stall reduction provided by ID-Cache.

also shows the performance of ID-Cache's component policies—ID-Cache Bypass and ALSI—individually. Generally, ID-Cache performs better than its component policies. Combining a more intelligent bypassing scheme and adaptive cache line size insertion results in added performance gain for most of the workloads. For a few workloads, i.e., `SS`, `CSR` and `MM`, while ID-Cache improves the performance, the performance gain is lower than that of its component policy, ALSI. This is because the reuse behavior changes with the varying cache line sizes and a simple bypass predictor (ID-Cache Bypass) is unable to learn the changing reuse behavior introduced by the varying cache line sizes inserted by ALSI, leading to bypassing cache lines too aggressively.

To understand the source of the large performance gain brought by ID-Cache, I take a closer look at the cache and interconnect performance. Figure 3.15 shows that L1 data cache hit rate improvement and the reduction in interconnect busy stalls, that is achieved by ID-Cache. The hit rate improvement signifies the improved utilization of cache capacity under ID-Cache. On the other hand, the reduction in interconnect busy stalls demonstrates ID-Cache's improved bandwidth utilization.

ID-Cache increases L1D cache hit rate by 10% and reduces interconnect busy stalls by 60% for C/I-H workloads. This shows that, by utilizing program level information such as instruction PC and runtime information such as memory divergence patterns intelligently, the performance of GPGPU applications can be significantly enhanced.

3.4 Related Work

In order to alleviate cache thrashing and resource contention, many prior works focused on designing cache bypassing algorithms tailed-made for GPGPUs. Jia et al. [75], Xie et al. [175], and Xie et al. [176] proposed using compilers to perform offline analysis and identify memory regions which have long reuse distances. These memory regions are then bypassed from the cache. Jia et al. [76], Chen et al. [51], and Khairy et al. [84] demonstrated that bypassing memory accesses whenever resource contention is detected could effectively improve the performance of GPGPUs. Tian et al. [165] built additional hardware in L2 caches to collect and predict the reuse pattern of L1 cache, whereas Li et al. [104] proposed using decoupled tag arrays to calculate the reuse distance. Lee et al. [100] proposed CAWA which uses instruction level information to predict reuse distance. The goal of CAWA is to accelerate the performance of the critical, i.e., the slowest running, warp within a thread block. In contrast, all these cache bypassing and modified insertion/replacement schemes only take temporal locality into account. This work characterizes the efficacy of utilizing program level information such as insertion PCs and runtime information such as memory divergence to predict reuse behavior and the spatial utilization patterns of cache lines.

Rhu et al. [146] observed that data caches in GPGPUs usually have low cache line utilization, i.e., only a small portion of data within a cache line are referenced during the line's lifetime. As a result, a large amount of data traffic across the

interconnection is redundant. Thus, the authors proposed LAMAR, a low overhead bloom filter and sectored cache based design to (1) reduce data traffic by bringing and storing segments of cache lines into the cache and (2) improve the energy efficiency by turning off the unused portion of the caches. This proposal, on the other hand, demonstrates that there is minimal amount of spatial locality that can be exploited in a wide variety of GPGPU applications and the spatial utilization of cache lines is highly correlated with the degree of divergence. Thus, instead of bringing a smaller amount of data based on first touched patterns, ID-Cache determines the amount of data to bring and store in the L1 caches based on memory divergence patterns.

3.5 Chapter Summary

In this chapter I identified key sources of inefficiencies in the memory subsystem of GPGPUs. My analysis indicated that there is an ample room for performance improvement which can be achieved by effective management of GPGPU L1 data caches and the interconnect bandwidth. I showed that the reuse behavior of cache lines is well correlated with program level information such as memory load/store instructions and runtime information such as memory divergence patterns. Based on the insights from the characterization results, I design ID-Cache, a simple, yet effective, cache management mechanism. ID-Cache identifies and bypasses zero reuse cache lines intelligently (PC+DIV Bypass) while inserting useful data into caches with appropriate size granularities (ALSI). ID-Cache achieves a significant 71% performance improvement by alleviating the severe data cache capacity problem faced by GPGPUs.

Chapter 4

LATENCY TOLERANCE AWARE CACHE COMPRESSION MANAGEMENT FOR GPGPUS

Chapter 3 of my thesis highlights an important cause of performance sub-optimality in GPGPUs — constrained data capacity. I proposed ID-Cache, a cache bypassing and adaptive line size insertion technique that alleviates the constrained data cache capacity problem. Orthogonally, cache compression is a common way to increase the effective capacity of caches with low overheads. This chapter focuses on the feasibility and applicability of cache compression as a solution to the constrained data capacity problem faced by GPGPUs.

To evaluate the applicability of cache compression, I first carry out a thorough characterization study to understand the data compressibility and the impact of decompression latency on GPGPU applications. I then propose LATency Tolerance awarE Cache Compression (LATTE-CC) [27] which intelligently leverages the GPGPU's latency tolerating ability to adaptively choose the best compression technique whose decompression latency can be hidden.

4.1 Background and Motivation

Cache data compression is a natural approach to increase the effective cache capacity in an energy efficient way. For data compression to be beneficial 1) the data used by the applications must be compressible, and 2) the performance benefit given by the effective capacity increase must be greater than the penalty incurred by the increase in cache hit time. In other words, the decompression latency must be partially or entirely hidden from the performance critical path of an application execution.

Table 4.1: Comparison between the state-of-the-art cache compression algorithms.

Algorithm	Decomp. Lat.	Value Locality	Compress-ibility	Complex.
Base Delta Immediate (BDI) [138]	2	Spatial	Higher	Low
Frequent Pattern Compression (FPC) [21]	5	Spatial	Low	High
Cache Packer + Zero Value Compression (CPACK + Z) [52, 57]	8	Both	Low	High
Bit Plane Compression (BPC) [90]	11	Spatial	High	Moderate
Statistical Compression (SC) [24]	14	Temporal	Highest	High

Section 4.1.1 first shows a detailed characterization study for data compressibility of GPGPU workloads and evaluates the performance benefit brought by the effective capacity increase. Section 4.1.2 assesses the degree of the decompression latency penalty that can be hidden in these workloads and by the architecture. Lastly, Section 4.1.3 motivates the need for adaptive compression designs in GPGPUs.

4.1.1 GPGPU Workload Data Compressibility

The effective capacity increase provided by data compression is a direct function of the data compressibility of applications. This data compressibility is dictated by the data values used and the algorithm itself. Prior work has observed value locality—data accessed by applications often has same or similar values during program execution [23]. Additionally, value locality can be extended to temporal value locality and spatial value locality [149]. Temporal value locality is the phenomenon where a particular data value is accessed repeatedly and spatial value locality is the phenomenon where the data values in adjacent memory locations are similar to each

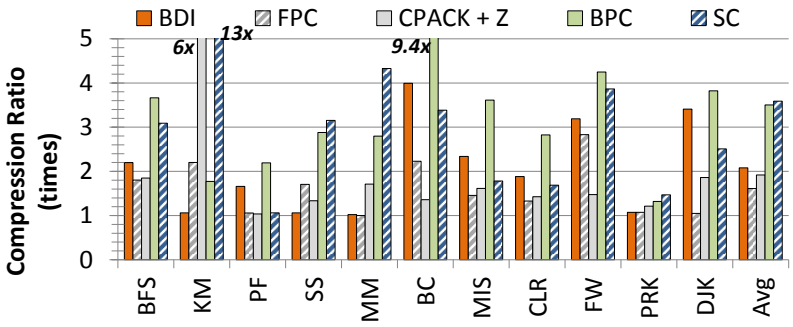


Figure 4.1: Compression ratio achieved by the state-of-the-art compression algorithms, i.e., BDI [138], FPC [21], CPACK-Z [52], BPC [90], and SC [24].

other. Data compression algorithms are designed to exploit the two distinct value locality properties.

The efficiency of a compression algorithm is measured as the achieved compression ratio — the ratio of the original data size and the resulting compressed size. To quantify the data compressibility of the workloads, I evaluate the compression ratio of all cache lines inserted in the L1 data caches with five state-of-the-art cache compression algorithms summarized in Table 4.1: base delta immediate compression (BDI) [138], frequent pattern compression (FPC) [21], dictionary-based compression with zero-block detection (CPACK-Z) [52, 57], bit plane compression (BPC) [90], and huffman-coding based statistical compression (SC) [24]. Algorithms such as BDI, FPC, and BPC perform value compression by compacting identical or similar values within cache lines, exploiting spatial value locality. On the other hand, CPACK-Z and SC exploit temporal value locality by replacing identical values across multiple memory locations with shorter codes.

Figure 4.1 shows the varying degree of the data compression ratios achieved for a wide range of GPGPU workloads [1] . We observe that almost all applications exhibit

[1]Workload selection is described in detail in Section 4.3 and Table 4.3.

a high degree of data compressibility. Applications, such as BFS, BC, FW, and DJK, achieve significant cache line size reduction with multiple compression algorithms and show both spatial and temporal localities in their data values. On the other hand, applications, such as KM, SS, MM, and PRK, show a significant affinity to the compression algorithms that exploit temporal value locality, whereas PF achieves more significant compression ratio with BDI and BPC, compression algorithms that exploits spatial value locality. This is due to the fact that the presence of spatial or temporal value locality in applications depends on the data types that are used in the applications [22]. Applications that operate on pointer and integer data typically contain low variance in data bits, thus exhibiting high degree of spatial value locality. On the other hand, high precision floating point data inherently contains high variance in the data bits. Thus, applications that operate on floating point data often have poor spatial value locality but exhibit high temporal value locality. This indicates a need for an adaptive algorithm that can exploit both the spatial and temporal value localities that exists but varies across workloads.

From Figure 4.1, we can also observe that commonly-used cache compression algorithms for CMP caches, i.e., FPC and CPACK+Z, do not achieve high compression ratios compared to BDI, BPC, and SC. Note that SC and BDI compression exploit complementary kinds of value locality and also represent two compression schemes with diverse decompression latencies. Thus, I focus the design and analysis with the combination of BDI and SC for the purpose of GPGPU's memory hierarchy optimization. Since there are a few workloads that prefer BPC compression in particular, I will study the inclusion of BPC compression in LATTE-CC later in Section 4.3.8

Next, I characterize the expected performance gain that can be attained due to the increase in L1 data cache capacity. To isolate the performance improvement potential from the decompression latency penalty, I increase the cache capacity by

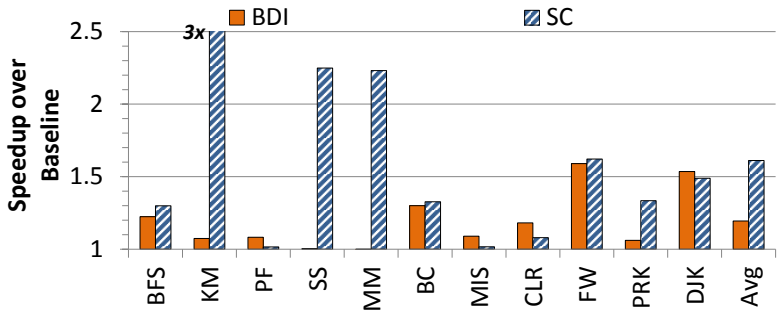


Figure 4.2: Performance impact of the effective cache capacity increase provided by data compression. The decompression latency is not taken into consideration here. Thus, the performance speedup shown here is the performance upper bound for static application of BDI and SC.

employing static compression while assuming a zero decompression latency. Figure 4.2 shows that significant performance improvement can be achieved for a majority of the workloads. This serves as the performance upper bound for the workloads under the static applications of BDI and SC, respectively.

4.1.2 Latency Tolerance of GPGPUs

GPGPUs group a number of parallel threads and execute them simultaneously in single instruction multiple thread (SIMT) fashion. This group of threads that execute simultaneously is called a warp. GPGPUs are able to hide the stall latency from a warp with useful instruction execution from another warp through fast context-switching. By taking advantage of this latency hiding feature, I expect to see a part of or all of the decompression latency to be overlapped with the execution of other available warps in the GPGPU pipeline.

The availability of this latency hiding feature depends mostly on two factors. Firstly, the regularity in an application's memory access behavior influences the available latency tolerance. For instance, depending on the underlying warp scheduling

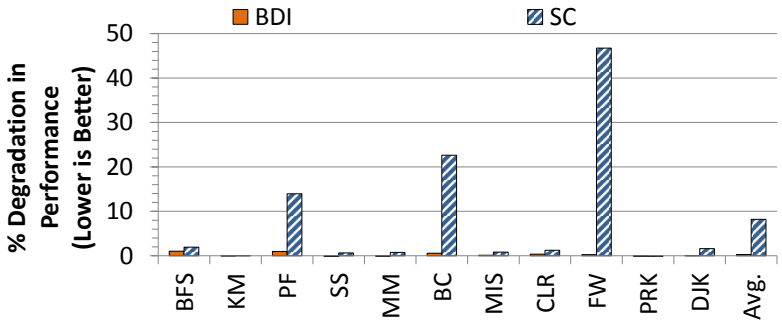


Figure 4.3: Performance degradation with increase in cache hit latency due to decompression. The cache capacity increase is not taken into consideration here.

policy, all warps in a GPU application could experience long latency memory access stalls at the same time. This results in low latency tolerance. Second, the GPGPU application might be characterized by varying amount of warp-level parallelism, possibly due to branch divergence.

To quantify the available latency tolerance in GPGPU workloads I measure the performance degradation caused by the decompression latencies of BDI and SC compression algorithms [2] . From Figure 4.3, we can see that some applications are highly sensitive to the decompression latency, while others are not. Applications, such as `FW` and `BC`, undergo significant performance degradation (47% and 22%, respectively), whereas `PRK` is able to tolerate the 14-cycle decompression latency of SC without experiencing any performance degradation.

4.1.3 Adaptive Compression in GPGPUs

Besides the varying degree and different forms of data value locality, and the varying degree of latency tolerance across different workloads, I also observe that the latency hiding ability of GPGPUs varies over time. Motivated by this, I delve deeper

[2]The decompression latencies are detailed in Section 5.3.1

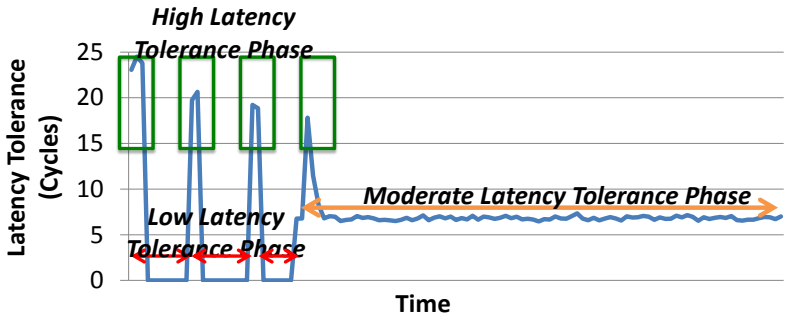


Figure 4.4: GPU latency tolerance characterization for SS GPGPU benchmark.

into investigating the temporal characteristics of GPU latency tolerance. I use the number of available warps in a GPU Streaming Multiprocessor (SM) as a proxy for the degree of latency tolerance and examine the time-varying latency tolerance for SS in Figure 4.4 as an example. The x-axis represents the application execution over time while the y-axis plots the latency tolerance. The latency tolerance represents the number of latency cycles that can be hidden by the GPGPU, described in detail in Section 4.2.2. We can see that SS goes through phases of varying degrees of latency tolerance, which dictates whether the decompression latency can be hidden or not. Therefore, exploiting the temporal variation in latency tolerance is critical to maximizing performance.

I characterize the performance improvement and energy reduction when BDI and SC are directly applied to the L1 data caches, taking into account both the capacity benefit and latency increase. Figure 4.5(a) shows the performance speedup for the GPGPU workloads under BDI (the first bars) and SC (the second bars). There is a significant variation from +48% to -52% in performance when a static cache compression method is applied. Similarly, a significant variation can be seen in the energy consumption (1.36x to 0.76x) when a static compression method is applied (Figure 4.5(b)). This is a compound effect of the performance gain from the increased

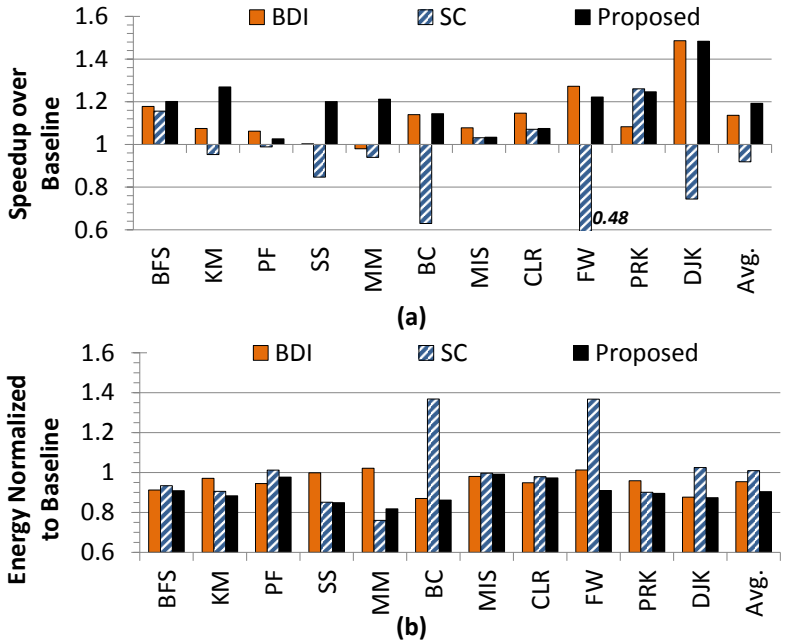


Figure 4.5: (a) Potential performance impact and (b) potential energy impact when BDI and SC are directly applied, and when an adaptive technique like LATTE-CC is applied.

cache capacity, the latency penalty from decompression and the temporal variations of latency tolerance. With a design that is able to exploit the variations of latency tolerance (the third bars) by switching between the available compression modes, additional performance and energy savings can be achieved. This is particularly significant for `KM`, `SS`, and `MM`.

Therefore, to achieve consistent high performance speedup and energy reduction, it is necessary to adopt a compression algorithm that achieves a higher compression ratio at the cost of longer decompression latency, during the execution phases of high latency tolerance. Similarly, it is also important to revert to a compression algorithm that incurs lower decompression latency at the cost of achieving potentially lower compression ratio during the execution phases of low latency tolerance. Finally,

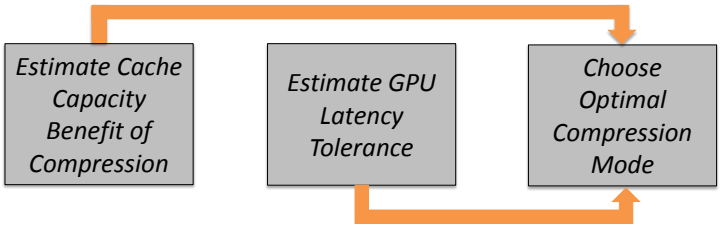


Figure 4.6: A conceptual overview of LATTE-CC.

when compression brings no added benefit, it might even be necessary to switch off the compression feature. With this insight, I design LATTE-CC, an adaptive technique that learns the runtime latency tolerance of GPGPU workloads, estimates the performance benefit of different compression methods, and determines the best cache compression operation mode to maximize performance.

4.2 LATTE-CC Design and Implementation

I propose an adaptive compression management approach, LATency Tolerance AwarE Cache Compression or LATTE-CC for the L1 data caches of GPUs. The key component of LATTE-CC is the design of an adaptive algorithm that dynamically predicts the best compression operating mode among the three choices: no compression (baseline), low-latency, and high-capacity modes, at runtime. The low-latency mode implements the BDI compression algorithm that exploits spatial value locality while the high-performance mode implements the SC compression algorithm which exploits temporal value locality available in applications. LATTE-CC is agnostic to the underlying compression algorithms and can be implemented with different compression hardware as well.

The dynamic compression mode selection is designed based on the performance trade-off of three important factors: cache capacity benefit brought by compression, decompression latency overhead, and the extent of GPU latency tolerance. Depending

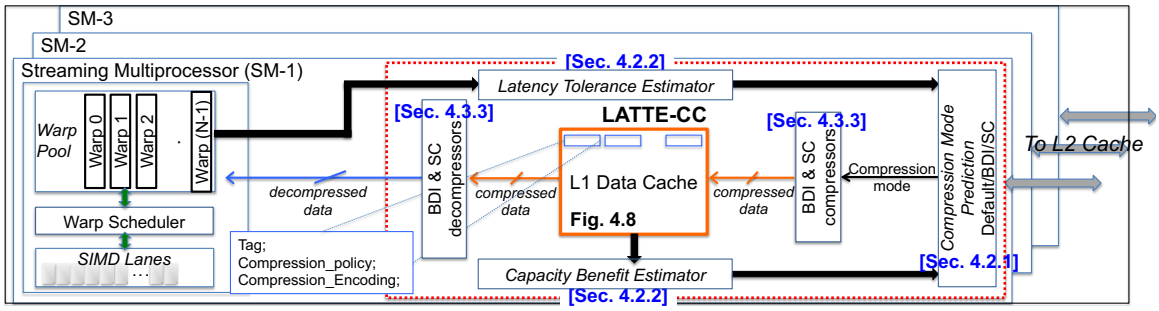


Figure 4.7: Block diagram of the modern GPU architecture with the LATTE-CC components.

on the application locality and data value characteristics, the different compression modes result in different degrees of performance improvement. On the other hand, depending on the dynamically varying latency hiding ability of the pipeline, a varying degree of the decompression latency can be hidden. Thus, LATTE-CC is designed to adopt the compression decision to maximize the net performance improvement (Section 4.2.1) by estimating both the benefit of cache capacity increase offered by the different compression modes (Section 4.2.2) and the dynamically varying latency tolerance of the GPU (Section 4.2.2). Figure 4.6 shows a conceptual overview of LATTE-CC's design and Figure 4.7 illustrates the LATTE-CC architecture and its three major components: the adaptive compression mode prediction algorithm, the cache capacity benefit estimator, and the latency tolerance estimator, in the context of a GPU.

4.2.1 Minimizing $AMAT_{GPU}$ for Optimal Compression Mode Selection

I use the average memory access time (AMAT) as a metric to combine the performance effects of cache capacity increase and decompression latency in the presence of a GPU's latency tolerance. An application receives more performance gain from using

one compression algorithm (Compr1) than using a different compression algorithm (Compr2) if $AMAT_{Compr1}$ is less than $AMAT_{Compr2}$.

Conventionally, $AMAT$ is given by

$$AMAT = \frac{total\_hit\_latency + total\_miss\_latency}{N_{hits} + N_{misses}} \tag{4.1}$$

where,

$$total\_hit\_latency = (N_{hits} * hit\_latency)$$

$$total\_miss\_latency = (N_{misses} * miss\_latency)$$

However, in the context of GPUs, the average memory access time that is experienced by the pipeline also depends on the GPU pipeline's latency hiding ability or latency tolerance. Therefore, $AMAT_{GPU}$ for a GPU should be expressed as

$$AMAT_{GPU} = \frac{total\_hit\_latency_{GPU} + total\_miss\_latency}{N_{hits} + N_{misses}} \tag{4.2}$$

where,

$$total\_hit\_latency_{GPU} = N_{hits} * min[(hit\_latency - latency\_tolerance), 0]$$

$latency\_tolerance$ = latency tolerance of GPU

$$total\_miss\_latency = N_{misses} * miss\_latency$$

I utilize this notion of $AMAT_{GPU}$ to dynamically determine the better operating compression mode. In other words, LATTE-CC estimates the $AMAT_{GPU}$ for the different compression modes periodically and chooses the compression mode that minimizes the average memory access time experienced by the application.

4.2.2 Dynamic Estimation of $AMAT_{GPU}$

LATTE-CC is designed with the goal of capturing the dynamic application phase behavior. To accomplish this, LATTE-CC uses a dynamic profiling technique to

estimate $AMAT_{GPU}$ for the different compression modes periodically. LATTE-CC breaks down the application execution into multiple smaller periods that consists of learning and adaptive phases, each comprised of one or more *Experimental Phases (EPs)*.

Estimating Performance Improvement From Increased Effective Capacity

LATTE-CC uses the learning phase of each period to estimate the cache capacity benefit brought by different compression modes. This is done using a modified set sampling-based dynamic profiling method [143] as shown in Figure 4.8. During the learning phase EPs, LATTE-CC operates a small number of cache sets of the L1 data cache as the dedicated sets for the Default, BDI, and SC compression modes. Then, in the following EPs in the adaptive phase, the dedicated sets are operated as the follower sets (to minimize set sampling overhead) which always apply the winning compression mode.

In order to estimate the performance of the different compression modes, I implement two counters for each mode—one counts the number of cache line insertions of a compression mode ($N_{miss,mode_i}$) and the other counts the number of cache hits ($N_{hit,mode_i}$). These counters are incremented only on accesses to the corresponding dedicated sets. That is, during the learning phase EPs, $N_{miss,mode_{default/BDI/SC}}$ and $N_{hit,mode_{default/BDI/SC}}$ are incremented. Note that the benefit of compression might manifest later in time relative to the insertion time. Therefore, I allow the counters, $N_{hit,mode_{default/BDI/SC}}$ to continue their update on hits in the dedicated sets during one subsequent EP following the learning phase EPs. Since cache line reuses exhibit generational behavior, I do not expect to see many more cache hits beyond the EP following the learning phase. Thus, if the learning phase spans one EP, then $N_{hit,mode_i}$ is designed to count the number of hits during the first and second EPs of each period.

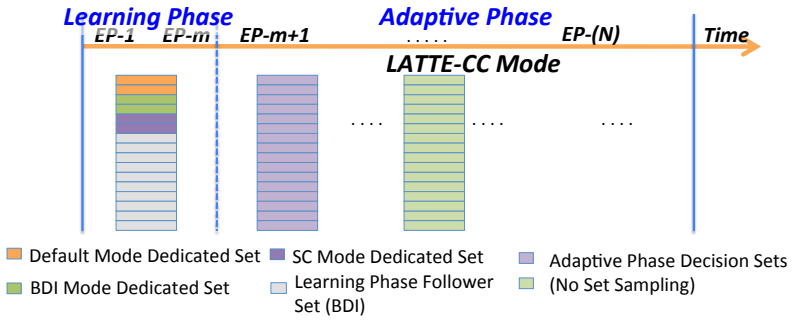


Figure 4.8: Modified set sampling in the LATTE-CC data caches.

Following the end of the learning phase, LATTE-CC is able to estimate the cache performance under the three operating compression modes using the dynamically measured hit and insertion counts from the dedicated sets.

Estimating Performance Penalty From Increased Hit Latency

In addition to the key element of exploiting data compressibility in GPU workloads, another important design question to address is whether and how the increase in cache hit latency can be overlapped with useful instruction execution in the GPU pipeline. An effective cache compression design has to take into account the time-varying degree of latency tolerance in GPU SMs such that the hit latency due to decompression can be hidden as much as possible.

I estimate the effective hit latency experienced by a compressed cache line as the sum of the decompression latency and the amount of time the line waits for service from the decompression unit (in a decompression queue). Therefore, the effective hit latency is

$$effective\_hit\_latency = decompression\_latency * (queue\_insertion\_pos + 1) \tag{4.3}$$

where,

$$queue\_insertion\_pos = the\ insertion\ position\ of\ the$$
$$line\ in\ the\ decompression\ queue$$

To determine whether or not the $effective\_hit\_latency$ can be hidden, LATTE-CC uses the number of available warps as a proxy for the degree of pipeline latency tolerance. For example, when a compressed cache line receives a hit, the additional decompression latency is incurred for the de-compressor to provide the data to the requesting warp. If there are other ready warps available for execution, the decompression latency becomes hidden and is overlapped with the execution of other warps.

In a round-robin warp scheduler, the degree of latency tolerance can be simply estimated as the number of available warps in the warp scheduler as the scheduler executes one instruction from each available warp before switching to the next. I utilize a more advanced Greedy-Then-Oldest (GTO) scheduler [147] in LATTE-CC. In schedulers similar to the state-of-the-art GTO scheduler, the scheduler tries to execute as many instructions as possible from each available warp before switching to the next. In such a scenario, the degree of latency tolerance can be estimated as follows:

$$latency\_tolerance = average\_warp\_available* \\ average\_execution\_cycles\_per\_schedule \tag{4.4}$$

4.2.3 Putting it all Together

Figure 4.9 shows the LATTE-CC execution over time. LATTE-CC goes through a number of learning and adaptive phases to adapt to the run-time phase behavior in an application. LATTE-CC learns and predicts the operating mode that results in better cache hit performance in the learning phase. Furthermore, LATTE-CC continuously

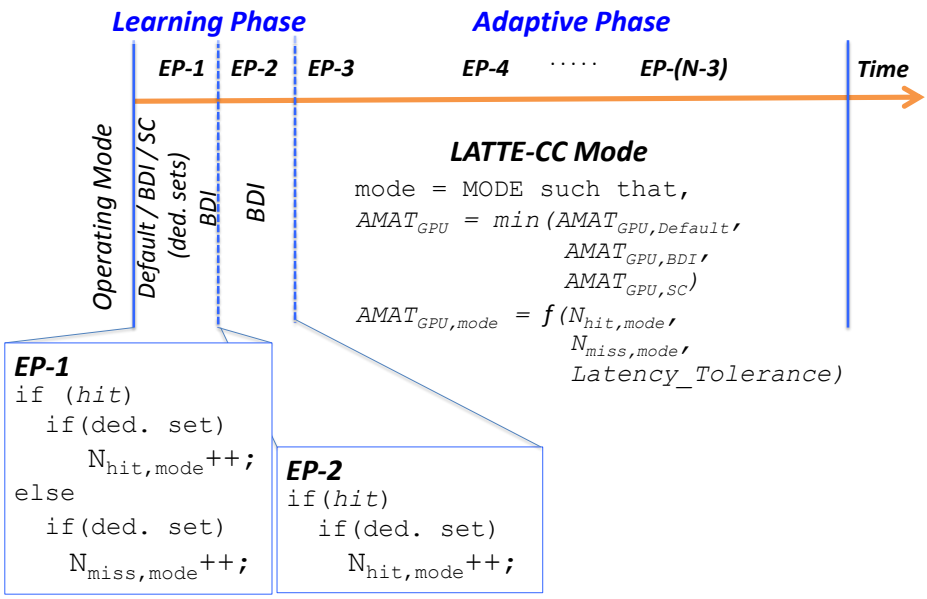


Figure 4.9: A temporal representation of LATTE-CC.

estimates the latency tolerance of the GPU pipeline in each EP of the adaptive phase. Finally, LATTE-CC chooses the optimal compression mode that maximizes the cache hit performance subject to the current degree of latency tolerance in each EP. By doing so, LATTE-CC always chooses the operating mode for each EP that results in the lowest $AMAT_{GPU}$.

4.3 Evaluation and Analysis

4.3.1 Simulation Infrastructure

I model LATTE-CC with GPGPU-Sim (version 3.2.2), a cycle-level GPU simulator [31]. The details of the simulated baseline system are given in Table 4.2. This setup is similar to the baseline configurations used in other recent works [100, 105, 137, 147]. I implement BDI and SC compressors/decompressors, and a compressed data cache in GPGPU-Sim. The compressed cache is provisioned with four times the tag blocks and allows data to be stored in 32B sub blocks. This compressed cache

Table 4.2: LATTE-CC: Baseline system configurations.

Parameter	Value(s)
Num. of SMs	*15*
Max. # of Warps per SM	*48*
Max. # of Blocks per SM	*8*
# of Schedulers per SM	*2*
# of Registers per SM	*32768*
Shared Memory	*48KB*
L1 Data Cache	*16KB per SM (128B lines/4-ways)*
L1 Inst Cache	*2KB per SM (128B lines/4-ways)*
L2 Cache	*768KB unified cache (128B lines/8-ways/12-banks)*
Min. L2 Access Latency	120 cycles
Min. DRAM access Latency	220 cycles
Warp Size (SIMD Width)	*32 threads*
Warp Scheduler	GTO [147]

organization is a simple modification to the existing data cache and similar cache organizations have been used in prior works [20, 22, 63].

To analyze the energy consumption of LATTE-CC and other compression methods, I use a modified version of GPUWattch [102] that is augmented with the BDI and SC compressor and decompressor power models.

4.3.2 Workload Construction

I use a wide variety of GPU workloads taken from Pannotia [46], Rodinia [47], Mars [65], and NVIDIA SDK [124] benchmark suites to evaluate LATTE-CC and

Table 4.3: LATTE-CC: GPGPU benchmarks and input sets.

Abbr	Application	Input	Cat.
BO	Binomial Options [124]	512 Options	C-InSens
PTH	Path Finder [47]	100k nodes	
HOT	Hotspot [47]	512x512 nodes	
FWT	Fast Walsh Trans. [124]	32k samples	
BP	Back Propagation [47]	65536 nodes	
NW	Needleman-Wunsh [47]	1024x1024 nodes	
SR1	SRAD1 [47]	502x458 nodes	
HTW	Heartwall [48]	656x744 AVI	
SC	Streamcluster [47]	32x4096 nodes	
BT	B+Tree [47]	1M nodes	
WC	Word Count [65]	86kB text file	
BFS	Breadth First Search [47]	65536 nodes	C-Sens
KM	K-Means [47]	494020 objects	
PF	Particle Filter [47]	28x128x10 nodes	
SS	Similarity Score [65]	1024x256 points	
MM	Matrix Multiplication [65]	1024x1024	
BC	Betweenness Centrality [46]	1K (V), 128K (E)	
MIS	Maximal Ind. Set [46]	ecology	
CLR	Graph Coloring [46]	ecology	
FW	Floyd Warshall [46]	256(V), 16K (E)	
PRK	Pagerank (SPMV) [46]	Co-Author DBLP	
DJK	Dijkstra-ELL [46]	USA road NY	

compare its performance with other designs. These workloads represent important computing domains such as web document clustering, web search, medical imaging, data mining, social network and graph analysis, financial modeling, and scientific simulations. I classify the applications into two categories based on their sensitivity to data cache capacity. I classify a workload as cache insensitive (C-InSens) if it experiences less than 20% performance speedup in the presence of a 4x larger data cache and as cache sensitive (C-Sens) if it experiences more than 20% performance speedup. The workloads and their input sets are summarized in Table 4.3. I simulate each benchmark for 1 billion instructions or to completion, whichever is earlier. Similar methodology for workload selection is used in recent GPGPU works [28, 43, 105]

4.3.3 Component Compression Policy Implementation Details

BDI Compressor/Decompressor Details

I model a 2/2-cycle compression/decompression latency, and 0.192/0.056 nJ compression/decompression access energy for BDI [22]. The BDI compression algorithm chooses a 2, 4, or 8B *base*, divides the cache line into blocks of size equal to *base*, and represents each block as a *delta* which is the difference of the value of the block from the *base*. This results in 10 possible encoding combinations of different *base* and *delta* as follows: (1) All zero; (2) *base* = 8B, *delta* = 0 (all blocks are same); (3) *base* = 8B, *delta* = 1, 2, or 4B; (4) *base* = 4B, *delta* = 0 (all blocks are same); *base* = 4B, *delta* = 1 or 2B; (5) *base* = 2B, *delta* = 0; (6) *base* = 2B, *delta* = 1B. These encodings are stored as part of the metadata in the 4 bit *compression_enc* field within each tag block.

SC Compressor/Decompressor Details

I model a 6/14-cycle compression/decompression latency and 0.42/0.336 nJ compression/decompression access energy for SC [22]. The SC compression algorithm uses a Huffman coding based compression technique [68] to compress the cache lines. Huffman coding based compression assigns variable length codes to data values based on the probability of their occurrence. A shorter code is applied to a value which has a higher probability of occurrence. In order to generate Huffman codes for compression, a value-frequency table (VFT), that holds the data value and the frequency of it's use needs to be built.

To exploit the generational behavior of cache accesses in GPU applications, I revise the SC compression algorithm such that, a 1024-entry VFT with 12-bit counters, is built during the first EP of the first period, and is re-built during the final EP of each period. Therefore, during each period of EPs, SC (and LATTE-CC) uses a newly generated set of codes to perform SC compression. SC (and LATTE-CC) invalidates older cache lines when a new period starts.

SC utilizes a table of code-words in the compressor, and a lookup table for decompression (DeLUT). The hardware overhead is 5.5KB for VFT, 7KB for the compressor, and 3KB for the DeLUT. This translates to a total of 6.45% of the total data cache capacity (15.5KB/(16KB/SM*15 SMs)) for the GPU. Note, the bandwidth requirement of the SC decompressor is determined largely by the L1 data cache hit rate, which is typically in the range of 40% - 50% for GPGPUs. This places a relatively low bandwidth demand on the decompressor as compared to CMPs, whose L1 data cache hit rates are typically greater than 90%.

LATTE-CC Parameters

I empirically set the period size of LATTE-CC to be 10 EPs and the length of the learning phase is set to be one EP. Finally, I set each EP to be 256 accesses long. During the learning phase, I use four dedicated sets per compression mode. I utilize two additional bits per tag blocks to store the *compression_policy* information for each cache line.

Based on performance characterization, I observe that the write policy employed for GPU L1 caches has negligible impact on performance. Therefore, I model L1 caches as write-evict caches. This allows LATTE-CC the choice of not having to potentially evict other cache lines on write hits.

4.3.4 Overall Performance and Energy Impact

Overall, LATTE-CC improves GPU performance by an average of 19.2% (by as much as 48.4%) and reduces L1 data cache misses by 24.6% compared to the baseline uncompressed cache for the C-Sens category workloads. While Static-BDI achieves 13.6% speedup and 19.2% reduction in L1 data cache misses, Static-SC incurs a performance degradation of 8.2% despite achieving an impressive 28.7% reduction in cache misses for these workloads. For C-InSens category workloads, LATTE-CC and Static-BDI result in negligible performance change as these workloads are not sensitive to the additional cache capacity brought by compression. On the other hand, the decompression latency penalty incurred by Static-SC results in a significant performance degradation for many applications (e.g. `HTW`, `SC`, `BT`), leading to 13.4% performance degradation on average for C-InSens workloads. Figures 4.10 and 4.11 show the application performance speedup and the L1 miss reduction comparison for Static-BDI, Static-SC, and LATTE-CC compression designs.

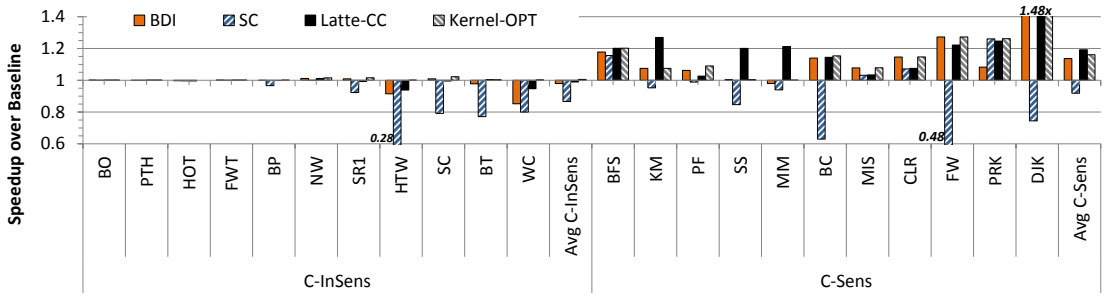


Figure 4.10: Performance improvement with LATTE-CC.

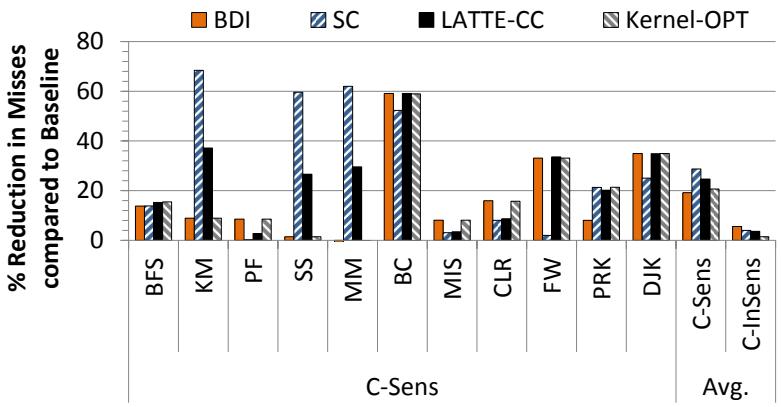


Figure 4.11: L1 cache miss reduction with LATTE-CC.

Furthermore, as we see from Figure 4.1, some C-Sens workloads favor BDI compression (e.g. `BC`, `FW`, `DJK`), while others favor SC (e.g. `PRK`, `KM`). Additionally, although applications such as `KM`, `SS`, and `MM` achieve higher compression ratio and lower cache miss rate (Figure 4.11) with SC, Static-SC is unable to translate this into performance improvement due to the high degree of unhidden decompression cost.

One of LATTE-CC's key design feature is to predict and adopt the best performing cache compression mode while taking into account the degree of GPU's latency tolerance, dynamically. By doing so, it captures the diverse and time-varying behavior of the workloads. That is, for workloads such as `BC`, `FW`, `DJK` and others, LATTE-CC is able to get the performance benefits of BDI compression while realizing the increased capacity benefits of the SC compression for workloads such as `KM`, `MM` and others. This

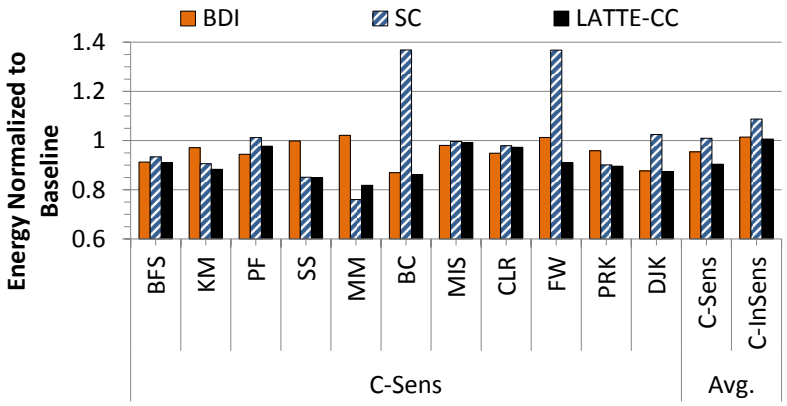


Figure 4.12: GPU energy consumption comparison.

results in LATTE-CC achieving superior performance across diverse application behaviors with a robust 19% average speedup and 24.6% reduction in misses compared to the baseline.

Energy Saving: I observe that LATTE-CC is able to achieve significant energy savings compared to the baseline. LATTE-CC's energy impact comes from the following sources: reduction in application execution time, reduction in data movement in the cache hierarchy, overhead associated with compression and decompression operations, and reduction in the L2 cache energy due to reduced accesses. I take all these factors into consideration.

Figure 4.12 shows the energy consumption of the GPU, for the different compression schemes. For C-Sens workloads, LATTE-CC reduces the energy consumption by 10% while Static-BDI does so by 5%. Static-SC on the other hand does not provide any energy savings on average. Among the C-InSens workloads, while LATTE-CC and Static-BDI do not alter the energy consumption considerably, Static-SC increases the energy consumption by 8.7% on average and by much as 53% for `HTW` workload.

Next, I take a closer look to understand the sources of energy reduction achieved by LATTE-CC. Figure 4.13 shows the breakdown of the energy reduction achieved

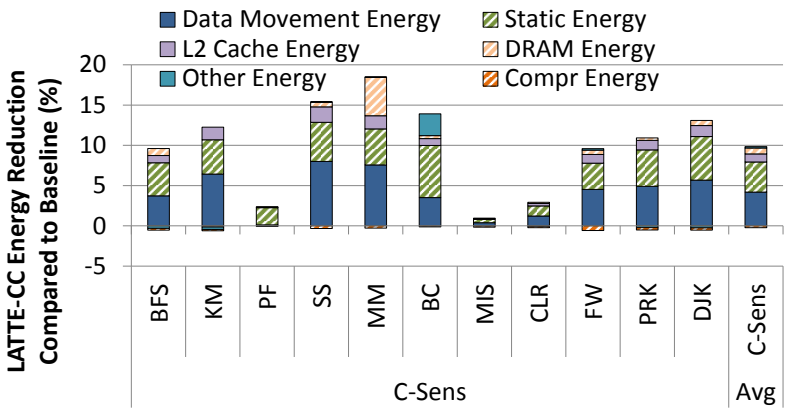


Figure 4.13: Breakdown of GPU Energy Reduction achieved by LATTE-CC.

by LATTE-CC for C-Sens workloads. I find that the reduction in data movement and static energy make up the bulk of the energy savings, providing 4.2% and 3.7% GPU energy reduction on average. Finally, I observe that the cost of compression and decompression energies is $< 0.25\%$ of the total GPU energy on average. *The energy analysis highlights the effectiveness of data compression in reducing the energy consumption in addition to GPU performance improvement.*

4.3.5 Comparing LATTE-CC with an Offline Optimal Policy

I also compare LATTE-CC to an oracular compression policy, Kernel-OPT. Kernel-OPT uses oracle knowledge from the end of each kernel of the application [3] to choose the compression mode that gives the lowest execution time for that kernel [4] . That is, while Kernel-OPT performs adaptive compression at a coarse kernel boundary granularity, LATTE-CC performs adaptive compression at a finer granularity within each kernel. As seen in Figure 4.10, LATTE-CC is able to perform slightly better than

[3]Note that a kernel is the block of parallel execution running on the GPU which consists of multiple LATTE-CC learning and adaptive phases.

[4]Though such a policy cannot be implemented in hardware, it serves as a reference point for my study.

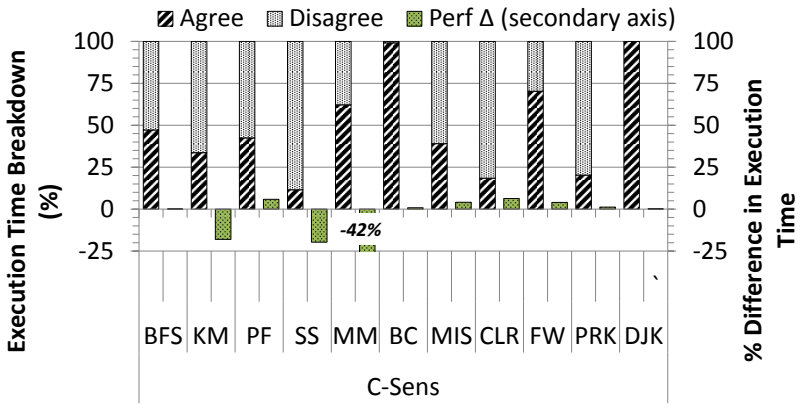


Figure 4.14: Comparison of LATTE-CC's compression mode decision with decision given by Kernel-OPT. (*Perf* Δ: Execution time difference between LATTE-CC and Kernel-OPT. Negative value means LATTE-CC performs better than Kernel-OPT.)

Kernel-OPT, achieving 3% higher speedup on average and 4% greater miss reduction for the C-Sens workloads.

To compare LATTE-CC's compression mode decisions with those suggested by Kernel-OPT, I measure the fraction of execution time where LATTE-CC's prediction agrees with Kernel-OPT's, shown in Figure 4.14. The x-axis shows the different benchmarks and the primary y-axis shows the fraction of the total application execution time where LATTE-CC's decision agrees/disagrees with the decision given by Kernel-OPT. This execution breakdown is shown in the first column for each application. The secondary y-axis represents the performance gap between Kernel-OPT and LATTE-CC for the different benchmarks. This is shown in the second column for each application.

For applications such as `BC` and `DJK`, the compression mode decision of LATTE-CC is highly correlated with the decision given by Kernel-OPT. However, for others, the compression mode selected by LATTE-CC is different from the decision given by Kernel-OPT. This results in some lost opportunity for performance improvement in applications such as `PF`, `CLR` and `PRK`. This lost opportunity is shown by the *Perf* Δ bar

in Figure 4.14. It is important to note that LATTE-CC is not designed to necessarily agree with Kernel-OPT as it operates at a much finer granularity than Kernel-OPT. This fine-grained runtime adaptation is particularly important for applications whose best compression operating mode changes over time. For such applications, LATTE-CC is able to achieve performance improvement that is significantly greater than what is suggested by Kernel-OPT (by as much as an additional 42%). `KM`, `SS`, and `MM` are applications that particularly benefit from the fine-grained adaptation as seen from the corresponding *Perf* Δ bars of Figure 4.14. Next, I examine the performance of `SS` in more detail.

4.3.6 An Illustrating Application Example: Similarity Score (SS)

`SS` presents an interesting case. `SS` is a memory-intensive application whose performance is mainly restricted by the efficiency of the data cache. `SS` achieves a modest 0.3% performance improvement with Static-BDI, -15.3% with Static-SC and 20% with LATTE-CC. To investigate the performance of `SS` further, I measure the effective cache capacity relative to the baseline cache in Figure 4.15. The effective cache capacity is calculated as the integral sum of the uncompressed size of all valid compressed cache lines. Over time, Static-BDI consistently offers very small capacity benefit to `SS`. This is because BDI is unable to compress the data values used by `SS` significantly (Figure 4.1). On the other hand, SC achieves an impressive compression ratio of 3.2x. Although Static-SC results in the highest effective cache capacity increase for `SS` (Figure 4.15), its application performance is significantly *degraded*. This is due to the performance penalty from the high hit latency which cannot be easily hidden. This cost outweighs the benefit brought by the capacity improvement.

To address the aforementioned shortcoming in the static schemes and to fully exploit the potential benefit brought by SC's high compression ratio, LATTE-CC dy-

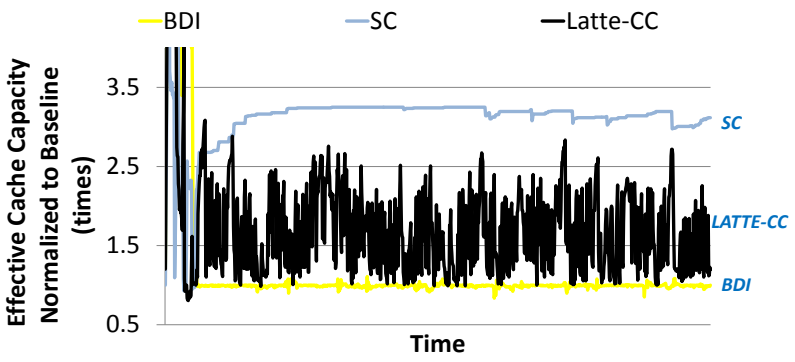


Figure 4.15: Effective cache capacity variation over time for Similarity Score (SS) application.

namically assesses the degree of latency tolerance in the GPU pipeline and switches among the three compression modes, depending on the degree of latency tolerance, respectively. Over its execution period, SS goes through phases of high, moderate, and low latency tolerance (Figure 4.4). Furthermore, my analysis shows that SS, like many other applications, possesses ample data locality. LATTE-CC is able to take advantage of the high and medium latency tolerance phases to choose SC compression during the periods of high data locality. This enables LATTE-CC to opportunistically achieve higher cache capacity when it is most beneficial. As can be seen in Figure 4.15, LATTE-CC's effective cache capacity hovers between 1-2X, resulting in significant net performance gain. This is possible only due to LATTE-CC's feature of fine-grained adaptive compression mode selection. Therefore, LATTE-CC is able to achieve 20% performance improvement, significantly higher than Static-BDI and Static-SC compression. LATTE-CC also results in 26.6% decrease in L1 cache misses whereas Static-BDI, and Static-SC achieve 1.4%, and 59.6% miss reduction, respectively.

LATTE-CC's performance is also much higher than that of Kernel-OPT. By operating at a much coarser, kernel boundary granularity, Kernel-OPT loses the opportu-

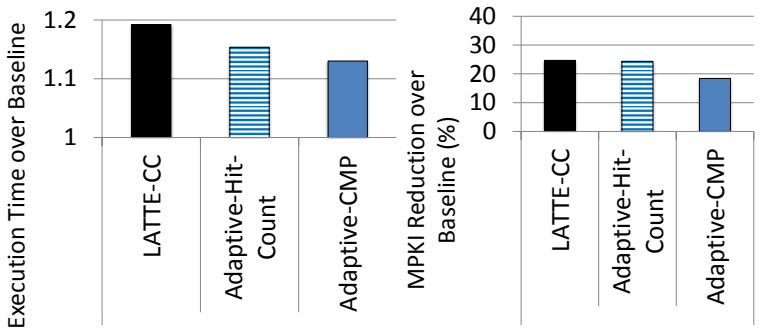


Figure 4.16: Performance comparison of LATTE-CC, Adaptive-Hit-Count, and Adaptive-CMP[20] policies for C-Sens workloads.

nity to take advantage of the runtime changes in latency tolerance within the kernel execution. This results in Kernel-OPT achieving only 0.3% performance improvement over the baseline. I observe a similar behavior with `KM` and `MM` workloads which experience a speedup of 26.9%, and 21.2% under LATTE-CC; significantly larger than that with Static-BDI, Static-SC, and Kernel-OPT for these workloads. *These applications highlight the advantage of LATTE-CC's fine-grained adaptive compression mode selection feature for GPUs.*

4.3.7 Benefits of Latency Tolerance Awareness

Next, I show that the optimization goal conventionally used for CMP caches— *the higher the cache hit rate, the better the performance is*—does not hold true for GPU compressed data caches. By accounting for the runtime latency tolerance of GPUs, LATTE-CC is able to achieve higher performance by sacrificing some cache hits in the process.

To illustrate the benefit of the latency tolerance awareness feature of LATTE-CC, I compare LATTE-CC to two other adaptive policies. First, I implement an adaptive policy that is purely based on the hit counts of the different compression modes—

Adaptive-Hit-Count. The Adaptive-Hit-Count policy is based on the modified set sampling policy described in Section 4.2.2 without taking into account the decompression latency or runtime latency tolerance variation in the GPU pipeline. Second, I compare LATTE-CC to an adaptive compression management method proposed for CMPs [20] that takes into account the effect of decompression latency but not the latency tolerance of GPUs. I refer to this policy as *Adaptive-CMP*.

Figure 4.16 shows the performance comparison of LATTE-CC in comparison with Adaptive-Hit-Count, and Adaptive-CMP policies. We can observe that although Adaptive-Hit-Count reduces misses by an average of 24.3% which is similar to that of LATTE-CC's miss reduction, this reduction in misses is not translated to performance improvement entirely. The *Adaptive-Hit-Count* policy experiences lower performance improvement compared to LATTE-CC, improving performance by only 15% over baseline. Similarly, Adaptive-CMP policy that is not aware of the GPU latency tolerance and hence performs sub-optimally compared to LATTE-CC. It achieves only 13% speedup over the baseline.

The performance of *Adaptive-Hit-Count* and *Adaptive-CMP* policies highlights two important aspects of GPU data cache designs—(1) Designs aimed to minimize miss counts, typically targeting CMP systems, are not always the best choice for GPUs. (2) The knowledge of GPU's time-varying latency tolerance is crucial and can be leveraged to achieve additional performance improvements.

4.3.8 Flexibility of LATTE-CC Design

Thus far, I have focused LATTE-CC's design and evaluation, having BDI, and SC as component compression algorithms. This combination of algorithms offers qualitative diversity in terms of the kind of value locality exploited and the decompression

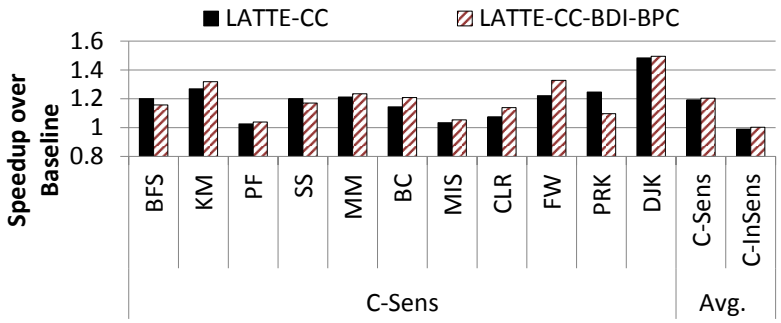


Figure 4.17: LATTE-CC performance with an alternative underlying compression algorithm.

latency incurred. However, it is important to note that LATTE-CC's adaptive algorithm design is agnostic to the underlying compression algorithms.

From Figure 4.1, we can observe that there are a few workloads such as `PF`, `MIS`, `CLR`, and `FW`, that show affinity to BPC. In fact, we see that on average, BPC achieves a similar compression ratio as SC. Therefore, BPC can be a plausible alternative to SC compression. Figure 4.17 shows the performance of LATTE-CC when it adaptively chooses between no-compression, BDI, and BPC compression modes (LATTE-CC-BDI-BPC). On average, I see that LATTE-CC-BDI-BPC performs similarly as LATTE-CC. This is reasonable considering that on average, BPC achieves a similar compression ratio (3.5x) as SC (3.6x), and its decompression latency (11 cycles) is also comparable to that of SC (14 cycles). Furthermore, I see that LATTE-CC-BDI-BPC performs better than LATTE-CC for workloads that show affinity to BPC compression i.e. `PF`, `MIS`, `CLR` and `FW`. *LATTE-CC is a flexible compression management design that can adapt to and maximize the performance advantages from the different underlying compression algorithms.*

4.4 Related Work

Compressed cache designs have been studied extensively for CMPs. Owing to the increase in hit latencies, they are typically not employed on the upper level of caches. This is the first work that carefully exploits the latency tolerating ability of GPGPUs to adaptively compress GPGPU L1 caches.

4.4.1 Data Compression in CMPs

Data compression for CMPs can be broadly categorized as compressed cache architectures [24, 63, 150, 151], compression algorithms [21, 52, 57, 138], cache replacement for compressed caches [30, 137], and main memory compression [59]. While these prior works all address the various design aspects of compressed caches for CMPs, the two closest related works that focus on adaptive compression techniques are [20, 22].

Alameldeen et al. [20] proposed a method to adaptively compress or not compress individual cache lines in a cache set. They consider the effect of decompression latency by noting that cache compression will be beneficial only if the performance benefit gained by compression offsets the decompression penalty that is incurred. However since their technique was proposed for CMP caches, it doesn't consider the impact of GPU latency tolerance.

More recently, Arelakis et al. [22] proposed a method to adaptively use one of several compression methods tailored to the data types of data being compressed. They develop heuristics to predict and identify different data types in hardware and choose a compression method that is known to yield the maximum compression ratio for a given data type. While their work estimates the benefit of cache compression that can be attributed to the increased cache capacity, they do not take into account the effect of increased hit latency or the latency tolerance that is available in GPUs. A direct

application of such techniques would lead to sub-optimal performance improvement for reasons similar to those detailed in Section 4.1.2.

4.4.2 Data Compression for GPU Memory

While this is the first work that explores the possibility of compressing L1 data caches in GPUs, data compression has been employed in GPU register files and off chip interconnect in GPUs. With the goal of reducing GPU register file power consumption, Lee et al. [99] proposed a method to compress the GPU register file. They observe that data held in registers exhibit low dynamic range for GPGPU applications. With this insight they use BDI [138] compression algorithm to compress the GPU register file and design the supporting compressed register file microarchitecture.

Pekhimenko et al. [136] proposed a toggle aware compression technique to reduce energy consumption while transferring compressed data, across the GPU interconnect. They noted that compression typically reduces the redundancy in bits and thereby increasing the amount of randomness that is seen per bit and thus leading to significant additional energy consumption due to bit-toggles. Vijaykumar et al. [169] propose to utilize idle cycles to compress the interconnect traffic with the help of assist warps. Another recent work by Sathish et al. [152] proposed a lossy technique to compress the traffic on the off-chip interconnect of GPGPU systems.

Kim et al. [90] propose bit plane compression, a compression algorithm tailored for GPUs achieves high compression ratio by employing data transformation techniques to enhance and exploit spatial value locality in cache lines. They utilize BPC to compress the interconnect traffic on GPUs. Similarly, Lal et al. [94] propose to compress the interconnect traffic on GPUs using huffman compression, similar to SC compression [24] employed in this work. Compressing the interconnect traffic reduces bandwidth consumption significantly, and could result in significant performance im-

provement and energy reduction due to reduced congestion on the interconnect. The benefit provided by such designs is orthogonal to the benefit provided by LATTE-CC.

4.5 Chapter Summary

This chapter performs a detailed performance characterization quantifying the impact of the GPGPU latency tolerance feature, and data value compressibility on system performance. By leveraging the latency tolerance of GPGPUs, I design LATTE-CC, a new adaptive latency tolerance aware cache compression management technique for GPGPU L1 data caches. LATTE-CC assesses the trade-off between the capacity benefit given by multiple compression schemes that exploit different kinds of value locality and the performance penalty introduced by the corresponding decompression latencies. It then adaptively applies the best-performing compression mode whose decompression penalty can be hidden by the runtime latency tolerance of the GPU. LATTE-CC can accurately predict the latency tolerance variation over application phases and from applications to applications. By operating at a finer-grainularity in time, LATTE-CC is demonstrated to perform better than an oracular scheme (Kernel-OPT) which applies the static oracle compression decision at the application kernel boundary. Overall, LATTE-CC improves GPGPU performance by an average of 19.2% and reduces L1 misses by an average of 24.6% for a wide range of cache-sensitive GPGPU applications, resulting in a 10% reduction in overall GPU energy consumption.

Chapter 5

MULTI-CHIP-MODULE GPGPUS AND THE MEMORY SUBSYSTEM DESIGN FOR THE POST MOORE'S LAW ERA

Chapters 3 and 4 identifies a crucial constrained data cache capacity problem within the modern GPGPU microarchitecture. With accurate cache bypassing, adaptive line size insertion, and latency tolerance aware cache compression, this thesis proposes effective solutions that address the data cache capacity problem and provide significant performance improvements.

However, in order to sustain GPGPU performance scaling, further to the microarchitecture optimizations described thus far, this thesis recognizes that GPU system-architecture innovations are necessary. As we look at the future GPGPU designs, we see that GPGPUs are faced with a crucial performance scalibility problem. The performance scaling of GPGPUs over the past decade has been significantly supported by transistor scaling and increasing GPGPU die sizes. However, due to technology limitations including the slowdown of transistor scaling (slowdown of Moore's Law [71, 120]) and die photoreticle size limitations, the path to future GPGPU performance scaling is unclear. This chapter focuses on describing a novel GPU architecture called the Multi-Chip-Module GPU (MCM-GPU) [25]. The MCM-GPU architecture offers a promising path forward for continued performance scaling of GPGPUs in the face of slowing Moore's law.

5.1 Background and Motivation

Modern GPUs accelerate a wide spectrum of parallel applications in the fields of scientific computing, data analytics, and machine learning. The abundant par-

Table 5.1: Key characteristics of recent NVIDIA GPUs.

	Fermi	Kepler	Maxwell	Pascal
SMs	16	15	24	56
BW (GB/s)	177	288	288	720
L2 (KB)	768	1536	3072	4096
Transistors (B)	3.0	7.1	8.0	15.3
Tech. node (nm)	40	28	28	16
Chip size (mm2)	529	551	601	610

allelism available in these applications continually increases the demands for higher performing GPUs. Table 5.1 lists different generations of NVIDIA GPUs released in the past decade. The table shows an increasing trend for the number of streaming multiprocessors (SMs), memory bandwidth, and number of transistors with each new GPU generation [10].

5.1.1 GPU Application Scalability

To understand the benefits of increasing the number of GPU SMs, Figure 5.1 shows performance as a function of the number of SMs on a GPU. The L2 cache and DRAM bandwidth capacities are scaled up proportionally with the SM count, i.e., 384 GB/s for a 32-SM GPU and 3 TB/s for a 256-SM GPU [1] . The figure shows two different performance behaviors with increasing SM counts. First is the trend of applications with limited parallelism whose performance plateaus with increasing SM count (Limited Parallelism Apps). These applications exhibit poor performance scalability (15 of the total 48 applications evaluated) due to the lack of available

[1]See Section 5.3.1 for details on the experimental methodology

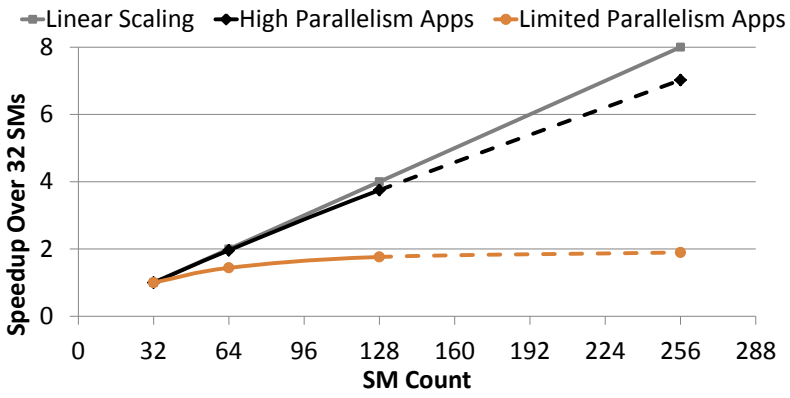


Figure 5.1: Hypothetical GPU performance scaling with growing number of SMs and memory system. 48 applications are grouped into 33 that have enough parallelism to fill a 256 SMs GPU, and 15 that do not.

parallelism (i.e. number of threads) to fully utilize larger number of SMs. On the other hand, I find that 33 of the 48 applications exhibit a high degree of parallelism and fully utilize a 256-SM GPU. Note that such a GPU is substantially larger (4.5×) than GPUs available today. For these High-Parallelism Apps, 87.8% of the linearly-scaled theoretical performance improvement can potentially be achieved if such a large GPU could be manufactured.

Unfortunately, despite the application performance scalability with the increasing number of SMs, *the observed performance gains are unrealizable with a monolithic single-die GPU design.* This is because the slowdown in transistor scaling [71] eventually limits the number of SMs that can be integrated onto a given die area. Additionally, conventional photolithography technology limits the maximum possible reticle size and hence the maximum possible die size. For example, $\approx 800mm^2$ is expected to be the maximum possible die size that can be manufactured [12, 158]. For the purpose of this work I assume that GPUs with greater than 128 SMs are

not manufacturable on a monolithic die. I illustrate the performance of such an unmanufacturable GPU with dotted lines in Figure 5.1.

5.1.2 Multi-GPU Alternative

An alternative approach is to stop scaling single GPU performance, and increase application performance via board- and system-level integration, by connecting multiple maximally sized monolithic GPUs into a multi-GPU system. While conceptually simple, multi-GPU systems present a set of critical challenges. For instance, work distribution across GPUs cannot be done easily and transparently; necessitating significant programmer expertise [35, 49, 50, 88, 119, 162]. Automated multi-GPU runtime and system-software approaches also face challenges with respect to work partitioning, load balancing, and synchronization [41, 161].

Moreover, a multi-GPU approach heavily relies on multiple levels of system interconnections. It is important to note that the data movement and synchronization energy dissipated along these interconnects significantly affects the overall performance and energy efficiency of such multi-GPU systems. Unfortunately, the quality of interconnect technology in terms of available bandwidth and energy per bit becomes progressively worse as communication moves off-package, off-board, and eventually off-node, as shown in Table 5.2 [8, 9, 11, 81, 157]. While the above integration tiers are an essential part of large systems (e.g. [13]), it is more desirable to reduce the off-board and off-node communication by building more capable GPUs.

5.1.3 Package-Level Integration

Recent advances in organic package technology are expected to address today's challenges and enable on-package integration of active components. For example, next generation packages are expected to support a $77mm$ substrate dimension [70],

Table 5.2: Approximate bandwidth and energy parameters for different integration domains.

	Chip	Package	Board	System
BW	10s TB/s	1.5 TB/s	256 GB/s	12.5 GB/s
Energy	80 fJ/bit	0.5 pJ/bit	10 pJ/bit	250 pJ/bit
Overhead	Low	Medium	High	Very High

providing enough room to integrate the MCM-GPU architecture described in this thesis. Furthermore, advances in package level signaling technologies such as NVIDIA's Ground-Referenced Signaling (GRS), offer the necessary high-speed, high-bandwidth signaling for organic package substrates. GRS signaling can operate at 20 Gb/s while consuming just 0.54 pJ/bit in a standard 28nm process [141]. As this technology evolves, we can expect it to support up to multiple TB/s of on-package bandwidth. This makes the on-package signaling bandwidth eight times larger than that of on-board signaling.

The aforementioned factors make package level integration a promising integration tier, that qualitatively falls in between chip- and board-level integration tiers (See Table 5.2). In this thesis, I aim to take advantage of this integration tier and set the ambitious goal of exploring how to manufacture a 2× more capable GPU, comprising 256 or more SMs within a single GPU package.

5.2 Multi-Chip-Module GPU Design

The proposed Multi-Chip Module GPU (MCM-GPU) architecture is based on aggregating multiple GPU modules (GPMs) within a single package, as opposed to today's GPU architecture based on a single monolithic die. This enables scaling single GPU performance by increasing the number of transistors, DRAM, and I/O

bandwidth per GPU. Figure 5.2 shows an example of an MCM-GPU architecture with four GPMs on a single package that potentially enables up to 4× the number of SMs (chip area) and 2× the memory bandwidth (edge size) compared to the largest GPU in production today.

5.2.1 MCM-GPU Organization

In this chapter I propose the MCM-GPU as a collection of GPMs that share resources and are presented to software and programmers as a single monolithic GPU. Pooled hardware resources, and shared I/O are concentrated in a shared on-package module. The goal for this MCM-GPU is to provide the same performance characteristics as a single (unmanufacturable) monolithic die. By doing so, the operating system and programmers are isolated from the fact that a single logical GPU may now be several GPMs working in conjunction. There are two key advantages to this organization. First, it enables resource sharing of underutilized structures within a single GPU and eliminates hardware replication among GPMs. Second, applications will be able to transparently leverage bigger and more capable GPUs, without any additional programming effort.

Alternatively, on-package GPMs could be organized as multiple fully functional and autonomous GPUs with very high speed interconnects. However, I do not propose this approach due to its drawbacks and inefficient use of resources. For example, if implemented as multiple GPUs, splitting the off-package I/O bandwidth across GPMs may hurt overall bandwidth utilization. Other common architectural components such as virtual memory management, DMA engines, and hardware context management would also be private rather than pooled resources. Moreover, operating systems and programmers would have to be aware of potential load imbalance and

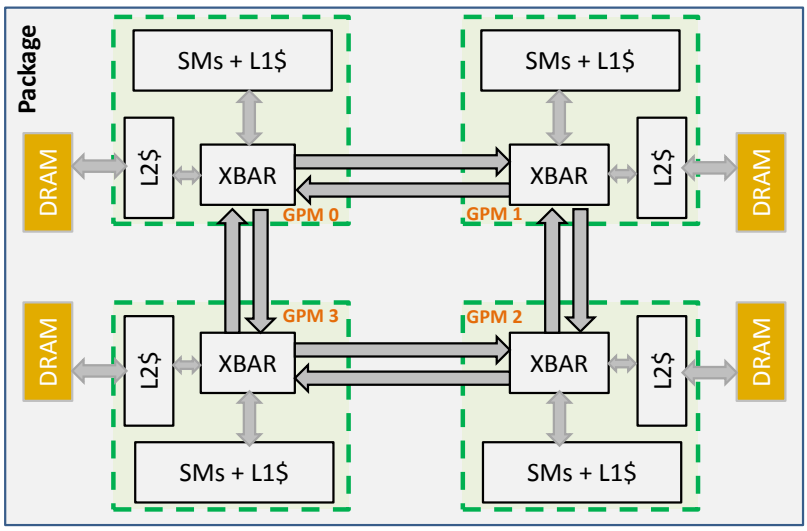


Figure 5.2: Basic MCM-GPU architecture comprising four GPU modules (GPMs).

data partitioning between tasks running on such an MCM-GPU that is organized as multiple independent GPUs in a single package.

5.2.2 MCM-GPU and GPM Architecture

As discussed in Section 5.1, moving forward beyond 128 SM counts will almost certainly require at least two GPMs in a GPU. Since smaller GPMs are significantly more cost-effective [79], in this chapter I evaluate building a 256 SM GPU out of four GPMs of 64 SMs each. This way each GPM is configured very similarly to today's biggest GPUs. Area-wise each GPM is expected to be 40% - 60% smaller than today's biggest GPU assuming the process node shrinks to 10nm or 7nm. Each GPM consists of multiple SMs along with their private L1 caches. SMs are connected through the GPM-Xbar to a GPM memory subsystem comprising a local memory-side L2 cache and DRAM partition. The GPM-Xbar also provides connectivity to adjacent GPMs via on-package GRS [141] inter-GPM links.

Figure 5.2 shows the high-level diagram of this 4-GPM MCM-GPU. Such an MCM-GPU is expected to be equipped with 3TB/s of total DRAM bandwidth and 16MB of total L2 cache. All DRAM partitions provide a globally shared memory address space across all GPMs. Addresses are fine-grain interleaved across all physical DRAM partitions for maximum resource utilization. GPM-Xbars route memory accesses to the proper location (either the local or a remote L2 cache bank) based on the physical address. They also collectively provide a modular on-package ring or mesh interconnect network. Such organization provides spatial traffic locality among local SMs and memory partitions, and reduces on-package bandwidth requirements. Other network topologies are also possible especially with growing number of GPMs, but a full exploration of inter-GPM network topologies is outside the scope of this thesis. The L2 cache is a memory-side cache, caching data only from its local DRAM partition. As such, there is only one location for each cache line, and no cache coherency is required across the L2 cache banks. In the baseline MCM-GPU architecture I employ a centralized CTA scheduler that schedules CTAs to MCM-GPU SMs globally in a round-robin manner as SMs become available for execution, as in the case of a typical monolithic GPU.

The MCM-GPU memory system is a Non Uniform Memory Access (NUMA) architecture, as its inter-GPM links are not expected to provide full aggregated DRAM bandwidth to each GPM. Moreover, an additional latency penalty is expected when accessing memory on remote GPMs. This latency includes data movement time within the local GPM to the edge of the die, serialization and deserialization latency over the inter-GPM link, and the wire latency to the next GPM. I estimate each additional inter-GPM hop latency, for a potentially multi-hop path in the on-package interconnect as 32 cycles. Each additional hop also adds an energy cost compared to a local DRAM access. Even though I expect the MCM-GPU architecture to incur these

bandwidth, latency, and energy penalties, I expect them to be much lower compared to off-package interconnects in a multi-GPU system (see Table 5.2).

5.2.3 On-Package Bandwidth Considerations

Estimation of On-package Bandwidth Requirements

I calculate the required inter-GPM bandwidth in a generic MCM-GPU. The basic principle for this analysis is that on-package links need to be sufficiently sized to allow full utilization of expensive DRAM bandwidth resources. Let us consider a 4-GPM system with an aggregate DRAM bandwidth of $4b$ units (3TB/s in this example), such that b units of bandwidth (768 GB/s in this example) are delivered by the local memory partition directly attached to each GPM. Assuming an L2 cache hit-rate of $\sim 50\%$ for the average case, $2b$ units of bandwidth would be supplied from each L2 cache partition. In a statistically uniform address distribution scenario, $2b$ units of bandwidth out of each memory partition would be equally consumed by all four GPMs. Extending this exercise to capture inter-GPM communication to and from all memory partitions results in the total inter-GPM bandwidth requirement of the MCM-GPU. A link bandwidth of $4b$ would be necessary to provide $4b$ total DRAM bandwidth. In the 4-GPM MCM-GPU example with 3TB/s of DRAM bandwidth ($4b$), link bandwidth settings of less than 3TB/s are expected to result in performance degradation due to NUMA effects. Alternatively, inter-GPM bandwidth settings greater than 3TB/s are not expected to yield any additional performance.

Performance Sensitivity to On-Package Bandwidth

Figure 5.3 shows performance sensitivity of a 256 SM MCM-GPU system as the inter-GPM bandwidth is decreased from an abundant 6TB/s per link all the way to

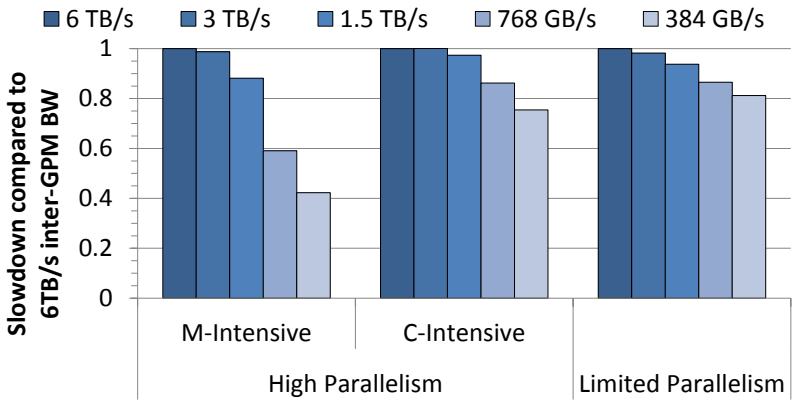


Figure 5.3: Relative performance sensitivity to inter-GPM link bandwidth for a 4-GPM, 256SM MCM-GPU system.

384GB/s. The applications are grouped into two major categories of high- and low-parallelism, similar to Figure 5.1. The scalable high-parallelism category is further subdivided into memory-intensive and compute-intensive applications (For further details about application categories and simulation methodology see Section 5.3.1).

The simulation results support the analytical estimations above. Increasing link bandwidth to 6TB/s yields diminishing or even no return for an entire suite of applications. As expected, MCM-GPU performance is significantly affected by the inter-GPM link bandwidth settings lower than 3TB/s. For example, applications in the memory-intensive category are the most sensitive to link bandwidth, with 12%, 40%, and 57% performance degradation for 1.5TB/s, 768GB/s, and 384GB/s settings respectively. Compute-intensive applications are also sensitive to lower link bandwidth settings, however with lower performance degradation. Surprisingly, even the non-scalable applications with limited parallelism and low memory intensity show performance sensitivity to the inter-GPM link bandwidth due to increased queuing delays and growing communication latencies in the low bandwidth scenarios.

On-Package Link Bandwidth Configuration

NVIDIA's GRS technology can provide signaling rates up to 20 Gbps per wire. The actual on-package link bandwidth settings for the 256 SM MCM-GPU can vary based on the amount of design effort and cost associated with the actual link design complexity, the choice of packaging technology, and the number of package routing layers. I assume, an inter-GPM GRS link bandwidth of 768 GB/s (equal to the local DRAM partition bandwidth) is realizable. Larger bandwidth settings such as 1.5 TB/s might be possible, albeit harder to achieve, and a 3TB/s link would require further investment and innovations in signaling and packaging technology. Moreover, higher than necessary link bandwidth settings would result in additional silicon cost and power overheads. Even though on-package interconnect is more efficient than its on-board counterpart, it is still substantially less efficient than on-chip wires and thus we *must* minimize inter-GPM link bandwidth consumption as much as possible.

In this thesis I assume a low-effort, low-cost, and low-energy link design point of 768GB/s and make an attempt to bridge the performance gap due to relatively lower bandwidth settings via architectural innovations that improve communication locality and essentially eliminate the need for more costly and less energy efficient links. The rest of the chapter evaluates architectural mechanisms to capture data-locality within GPM modules, which eliminate the need for costly inter-GPM bandwidth solutions.

5.3 Evaluation and Analysis

In this section, I first describe the simulation methodology and then progressively evaluate multiple locality-aware designs for the MCM-GPU. In the process I propose three mechanisms to minimize inter-GPM bandwidth by capturing data locality within a GPM. First, I revisit the MCM-GPU cache hierarchy and propose a GPM-

side hardware cache. Second, I augment the MCM-GPU architecture with distributed CTA scheduling to exploit inter-CTA data locality within the GPM-side cache and in memory. Finally, I propose data partitioning and locality-aware page placement to further reduce on-package bandwidth requirements. The three mechanisms combined significantly improve MCM-GPU performance.

5.3.1 Simulation Infrastructure

I use an NVIDIA in-house simulator to conduct the performance studies. I model the GPU to be similar to, but extrapolated in size compared to the recently released NVIDIA Pascal GPU [128]. The SMs are modeled as in-order execution processors that accurately model warp-level parallelism. I model a multi-level cache hierarchy with a private L1 cache per SM and a shared L2 cache. Caches are banked such that they can provide the necessary parallelism to saturate DRAM bandwidth. I model software based cache coherence in the private caches, similar to state-of-the-art GPUs. Table 5.3 summarizes baseline simulation parameters.

5.3.2 Workload Construction

I study a diverse set of 48 benchmarks that are taken from four benchmark suites. My evaluation includes a set of production class HPC benchmarks from the CORAL benchmarks [6], graph applications from Lonestar suite [133], compute applications from Rodinia [47], and a set of NVIDIA in-house CUDA benchmarks. The application set covers a wide range of GPU application domains including machine learning, deep neural networks, fluid dynamics, medical imaging, graph search, etc. I classify the applications into two categories based on the available parallelism — high parallelism applications (parallel efficiency $\geq 25\%$) and limited parallelism applications

[2]Other evaluated compute intensive and limited parallelism workloads are not shown in Table 5.4.

Table 5.3: Baseline MCM-GPU simulation configuration.

Number of GPMs	4
Total number of SMs.	256
GPU frequency	1GHz
Max number of warps	64 per SM
Warp scheduler	Greedy then Round Robin
L1 data cache	128 KB per SM, 128B lines, 4 ways
Total L2 cache	16MB, 128B lines, 16 ways
Inter-GPM interconnect	768GB/s per link, Ring, 32 cycles/hop
Total DRAM bandwidth	3 TB/s
DRAM latency	100ns

(parallel efficiency $< 25\%$). I further categorize the high parallelism applications based on whether they are memory-intensive (M-Intensive) or compute-intensive (C-Intensive). I classify an application as memory-intensive if it suffers from more than 20% performance degradation if the system memory bandwidth is halved. In the interest of space, I present the detailed per-application results for the M-Intensive category workloads and present only the average numbers for the C-Intensive and limited-parallelism workloads. The set of M-Intensive benchmarks, and their memory footprints are detailed in Table 5.4. I simulate all the benchmarks for one billion warp instructions, or to completion, whichever occurs first.

5.3.3 Revisiting MCM-GPU Cache Architecture

Introducing L1.5 Cache

The first mechanism I propose to reduce on-package link bandwidth is to enhance the MCM-GPU cache hierarchy. I propose to augment the baseline GPM architecture

Table 5.4: The high parallelism, memory intensive workloads for MCM-GPU evaluation and their memory footprints[2].

Benchmark	Abbr.	Memory Footprint (MB)
Algebraic multigrid solver	AMG	5430
Neural Network Convolution	NN-Conv	496
Breadth First Search	BFS	37
CFD Euler3D	CFD	25
Classic Molecular Dynamics	CoMD	385
Kmeans clustering	Kmeans	216
Lulesh (size 150)	Lulesh1	1891
Lulesh (size 190)	Lulesh2	4309
Lulesh unstructured	Lulesh3	203
Adaptive Mesh Refinement	MiniAMR	5407
Mini Contact Solid Mechanics	MnCtct	251
Minimum Spanning Tree	MST	73
Nekbone solver (size 18)	Nekbone1	1746
Nekbone solver (size 12)	Nekbone2	287
SRAD (v2)	Srad-v2	96
Shortest path	SSSP	37
Stream Triad	Stream	3072

in Figure 5.2 with a GPM-side cache that resides between the L1 and L2 caches. I call this new cache level the L1.5 cache as shown in Figure 5.4. Architecturally, the L1.5 cache can be viewed as an extension of the L1 cache and is shared by all SMs inside a GPM. I propose that the L1.5 cache stores remote data accesses made by a GPM partition. In other words, all local memory accesses will bypass the L1.5 cache.

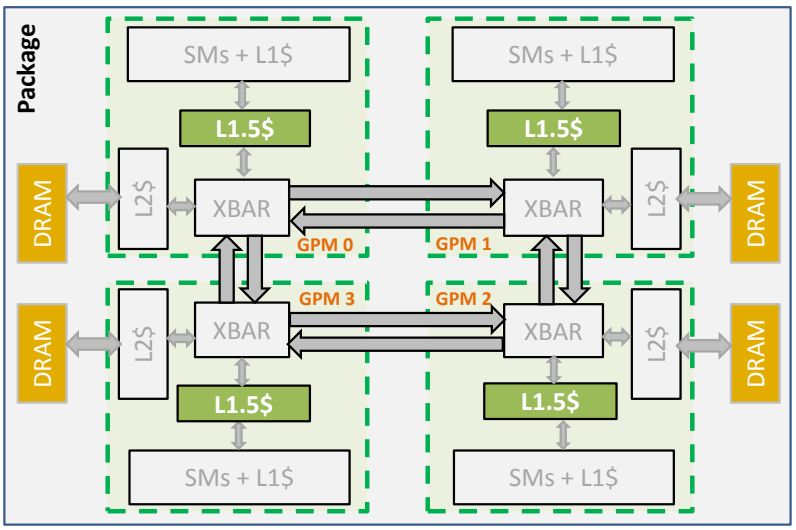


Figure 5.4: MCM-GPU architecture equipped with L1.5 GPM-side cache to capture remote data and effectively reduce inter-GPM bandwidth and data access latency.

Doing so reduces both remote data access latency and inter-GPM bandwidth. Both these properties improve performance and reduce energy consumption by avoiding inter-GPM communication.

To avoid increasing on-die transistor overhead for the L1.5 cache, I add it by rebalancing the cache capacity between the L2 and L1.5 caches in an iso-transistor manner. I extend the GPU L1 cache coherence mechanism to the GPM-side L1.5 caches as well. This way, whenever an L1 cache is flushed on a synchronization event such as reaching a kernel execution boundary, the L1.5 cache is flushed as well. Since the L1.5 cache can receive multiple invalidation commands from GPM SMs, I make sure that the L1.5 cache is invalidated only once for each synchronization event.

Design Space Exploration for the L1.5 Cache

I evaluate MCM-GPU performance for three different L1.5 cache capacities: an 8MB L1.5 cache where half of the memory-side L2 cache capacity is moved to the L1.5

caches, a 16MB L1.5 cache where almost all of the memory-side L2 cache is moved to the L1.5 caches [3] , and finally a 32MB L1.5 cache, a non iso-transistor scenario where in addition to moving the entire L2 cache capacity to the L1.5 caches I add an additional 16MB of cache capacity. As the primary objective of the L1.5 cache is to reduce the inter-GPM bandwidth consumption, I evaluate different cache allocation policies based on whether accesses are to the local or remote DRAM partitions.

Figure 5.5 summarizes the MCM-GPU performance for different L1.5 cache sizes. I report the average performance speedups for each category, and focus on the memory-intensive category by showing its individual application speedups. I observe that performance for the memory-intensive applications is sensitive to the L1.5 cache capacity, while applications in the compute-intensive and limited-parallelism categories show very little sensitivity to various cache configurations. When focusing on the memory-intensive applications, an 8MB iso-transistor L1.5 cache achieves 4% average performance improvement compared to the baseline MCM-GPU. A 16MB iso-transistor L1.5 cache achieves 8% performance improvement, and a 32MB L1.5 cache that doubles the transistor budget achieves an 18.3% performance improvement. I choose the 16MB cache capacity for the L1.5 and keep the total cache area constant.

The simulation results confirm the intuition that the best allocation policy for the L1.5 cache is to only cache remote accesses, and therefore I employ a remote-only allocation policy in this cache. From Figure 5.5 we can see that such a configuration achieves the highest average performance speedup among the two iso-transistor configurations. It achieves an 11.4% speedup over the baseline for the memory-intensive GPU applications. While the GPM-side L1.5 cache has minimal impact on the compute-intensive GPU applications, it is able to capture the relatively small

[3] A small cache capacity of 32KB is maintained in the memory-side L2 cache to accelerate atomic operations.

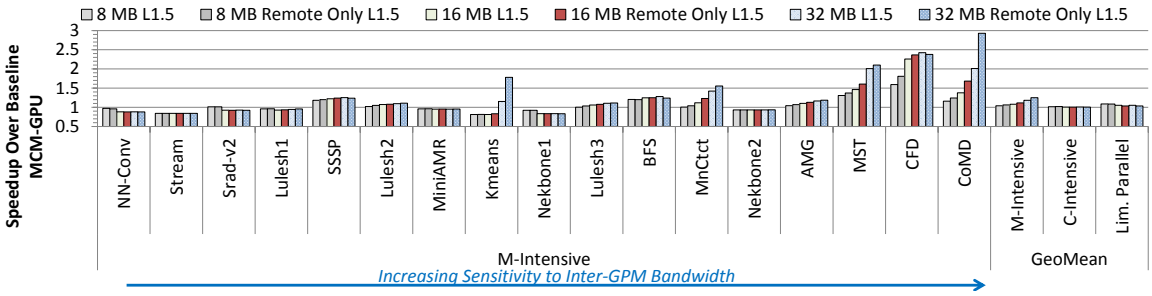


Figure 5.5: Performance of 256 SM, 768 GB/s inter-GPM BW MCM-GPU with 8MB (iso-transistor), 16 MB (iso-transistor), and 32 MB (non-iso-transistor) L1.5 caches. The M-Intensive applications are sorted by their sensitivity to inter-GPM bandwidth.

working sets of the limited-parallelism GPU applications and provide a performance speedup of 3.5% over the baseline. Finally, Figure 5.5 shows that the L1.5 cache generally helps applications that incur significant performance loss when moving from a 6TB/s inter-GPM bandwidth setting to 768GB/s. This trend can be seen in the figure as the memory-intensive applications are sorted by their inter-GPM bandwidth sensitivity from left to right.

In addition to improving MCM-GPU performance, the GPM-side L1.5 cache helps to significantly reduce the inter-GPM communication energy associated with on-package data movements. This is illustrated by Figure 5.6 which summarizes the total inter-GPM bandwidth with and without the L1.5 cache. Among the memory-intensive workloads, inter-GPM bandwidth is reduced by as much as 39.9% for the `SSSP` application and by an average of 16.9%, 36.4%, and 32.9% for memory-intensive, compute-intensive, and limited-parallelism workloads respectively. On average across all evaluated workloads, I observe that inter-GPM bandwidth utilization is reduced by 28% due to the introduction of the GPM-side L1.5 cache.

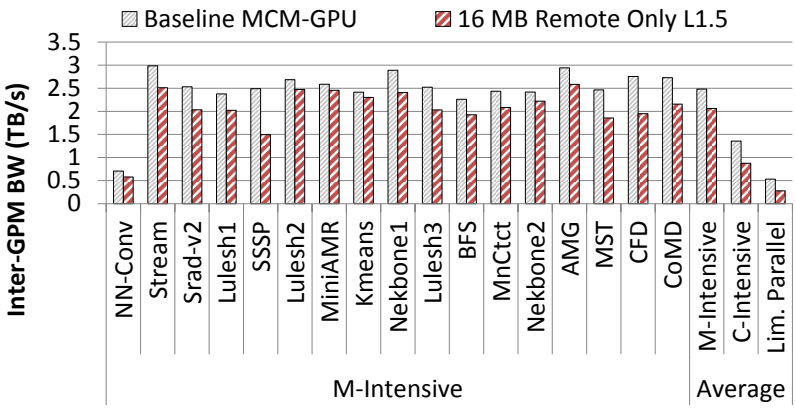
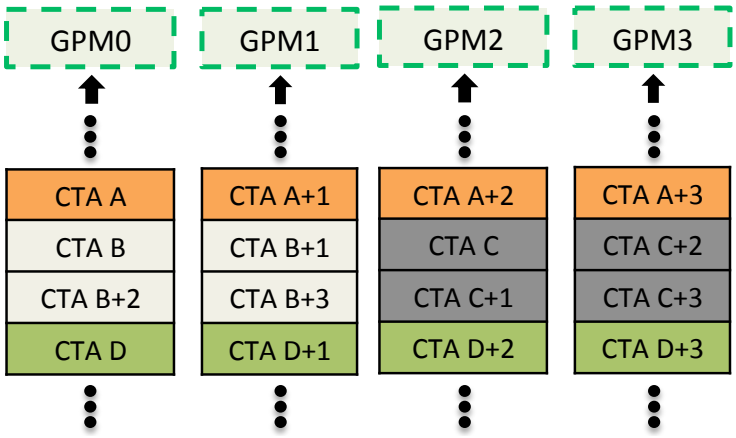


Figure 5.6: Total inter-GPM bandwidth in baseline MCM-GPU architecture and with a 16MB *remote-only* L1.5 cache.

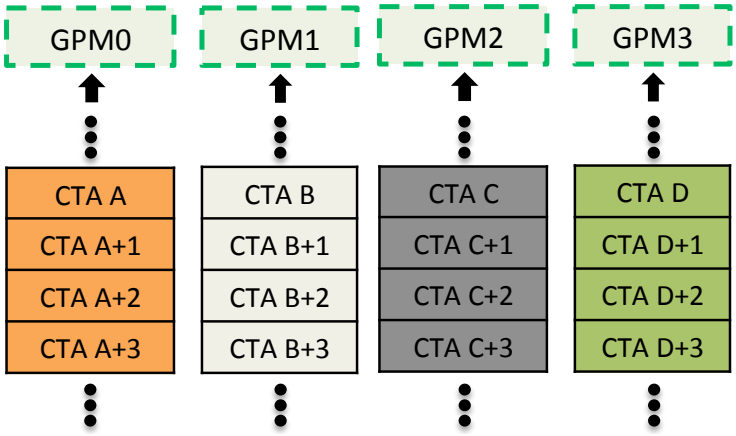
5.3.4 CTA Scheduling for GPM Locality

In a baseline MCM-GPU similar to monolithic GPU, at kernel launch, a first batch of CTAs are scheduled to the SMs by a centralized scheduler in-order. However during kernel execution, CTAs are allocated to SMs in a round-robin order based on the availability of resources in the SMs to execute a given CTA. In steady state application execution, this could result in consecutive CTAs being scheduled on SMs in different GPMs as shown in Figure 5.7(a). The colors in this figure represent four groups of contiguous CTAs that could potentially enjoy data locality if they were scheduled in close proximity and share memory system resources. While prior work has attempted to exploit such inter-CTA locality in the private L1 cache [98], here I propose a CTA scheduling policy to exploit this locality across all memory system components associated with GPMs due to the NUMA nature of the MCM-GPU design.

To this end, I propose using a distributed CTA scheduler for the MCM-GPU. With the distributed CTA scheduler, a group of contiguous CTAs are sent to the same GPM as shown in Figure 5.7(b). Here we can see that all four contiguous CTAs



(a) Centralized CTA Scheduling in an MCM-GPU.



(b) Distributed CTA Scheduling in an MCM-GPU.

Figure 5.7: An example of exploiting inter-CTA data locality with CTA scheduling in MCM-GPU.

of a particular group are assigned to the same GPM. In the context of the MCM-GPU, doing so enables better cache hit rates in the L1.5 caches and also reduces inter-GPM communication. The reduced inter-GPM communication occurs due to contiguous CTAs sharing data in the L1.5 cache and avoiding data movement over the inter-GPM links. In the example shown in Figure 5.7, the four groups of contiguous

CTAs are scheduled to run on one GPM each, to potentially exploit inter-CTA spatial data locality.

I choose to divide the total number of CTAs in a kernel equally among the number of GPMs, and assign a group of contiguous CTAs to a GPM. Figures 5.8 and 5.9 show the performance improvement and bandwidth reduction provided by this proposal when combined with the L1.5 cache described in the previous section. On average, the combination of these proposals improves performance by 23.4% / 1.9% / 5.2% on memory-intensive, compute-intensive, and limited-parallelism workloads respectively. In addition, inter-GPM bandwidth is reduced further by the combination of these proposals. On average across all evaluated workloads, I observe that inter-GPM bandwidth utilization is reduced by 33%.

For workloads such as `Srad-v2`, and `Kmeans`, the combination of distributed scheduling and remote-only caching provides significant performance improvement while remote-only caching does not improve performance in isolation (Figure 5.5). This is due to the improved inter-CTA data reuse in the L1.5 cache when distributed scheduling is applied. Although distributed scheduling provides significant additional performance benefit for a number of evaluated workloads, I observe that it causes some applications to experience degradation in performance. Such workloads tend to suffer from the coarse granularity of CTA division and may perform better with a smaller number of contiguous CTAs assigned to each GPM. A case for a dynamic mechanism for choosing the group size could be made. While I do not explore such a design in this chapter, I expect a dynamic CTA scheduler to obtain further performance gain.

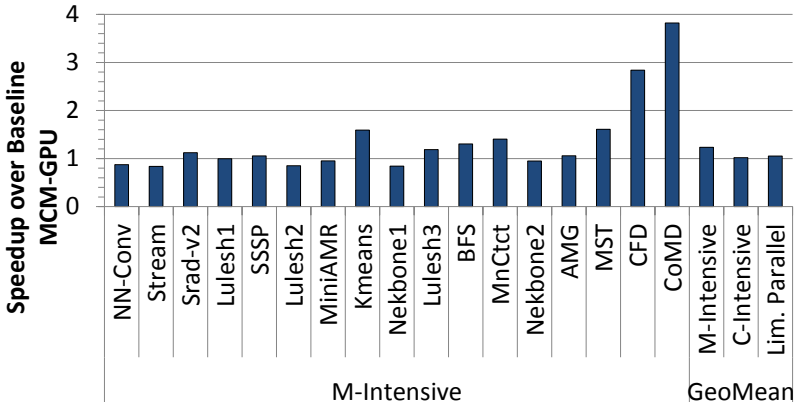


Figure 5.8: Performance of MCM-GPU system with a distributed scheduler.

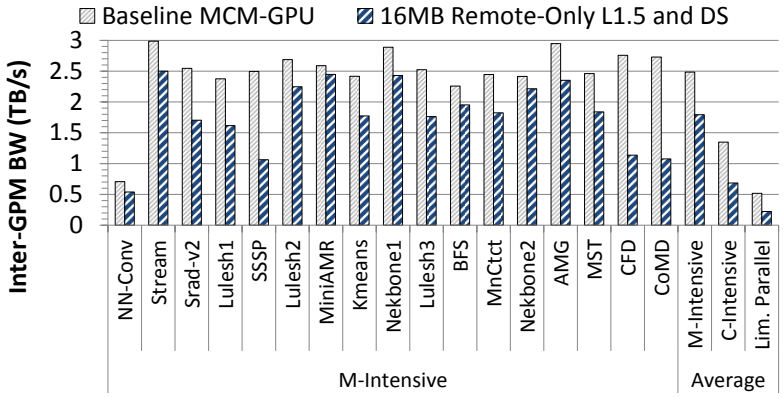


Figure 5.9: Reduction in inter-GPM bandwidth with a distributed scheduler compared to baseline MCM-GPU architecture.

5.3.5 Data Partitioning for GPM Locality

Prior work on NUMA systems focuses on co-locating code and data by scheduling threads and placing pages accessed by those threads in close proximity [54, 111, 171]. Doing so limits the negative performance impact of high-latency low-bandwidth inter-node links by reducing remote accesses. In an MCM-GPU system, while the properties of inter-GPM links are superior to traditional inter-package links assumed in prior work (i.e., the ratio of local memory bandwidth compared to remote memory

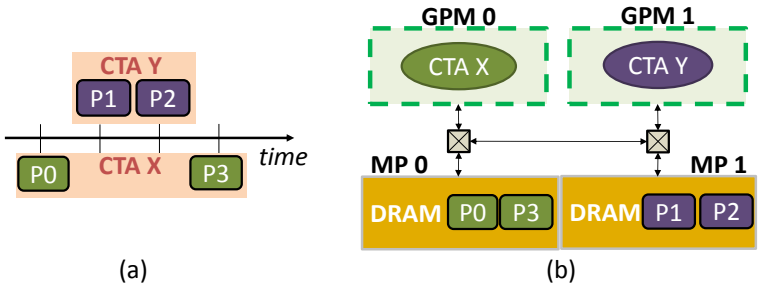


Figure 5.10: First Touch page mapping policy: (a) Access order. (b) Proposed page mapping policy.

bandwidth is much greater and latency much lower for inter-package links), I revisit page placement policies to reduce inter-GPM bandwidth.

To improve MCM-GPU performance, special care is needed for page placement to reduce inter-GPM traffic when possible. Ideally, it is beneficial to map memory pages to physical DRAM partitions such that they would incur as many local memory accesses as possible. In order to maximize DRAM bandwidth utilization and prevent camping on memory channels within the memory partitions, I continue to interleave addresses at a fine granularity across the memory channels of each memory partition (analogous to the baseline described in Section 5.2.2).

Figure 5.10 shows a schematic representation of the first touch (FT) page mapping policy I employ in the MCM-GPU. When a page is referenced for the first time in the FT policy, the page mapping mechanism checks which GPM the reference is from and maps the page to the local memory partition (MP) of that GPM. For example, in the figure, page P0 is first accessed by CTA-X which is executing on GPM0. This results in P0 being allocated in MP0. Subsequently, pages P1 and P2 are first accessed by CTA-Y executing on GPM1, which maps those pages to MP1. Following this, page P3 is first accessed by CTA-X, which maps the page to MP0. This policy results in

keeping DRAM accesses mostly local. Regardless of the referencing order, if a page is first referenced from CTA-X in GPM0, then the page will be mapped to the MP0, which would keep accesses to that page local and avoid inter-GPM communication. This page placement mechanism is implemented in the software layer by extending current GPU driver functionality. Such driver modification is transparent to the OS, and does not require any special handling from the programmer.

An important benefit that comes from the first touch mapping policy is its synergy with the CTA scheduling policy described in Section 5.3.4. I observe that inter-CTA locality exists across multiple kernels and within each kernel at a page granularity. For example, the same kernel is launched iteratively within a loop in applications that contain convergence loops and CTAs with the same indices are likely to access the same pages. Figure 5.11 shows an example of this. As a result of the distributed CTA scheduling policy and the first touch page mapping policy described above, MCM-GPU is able to exploit inter-CTA locality across the kernel execution boundary as well. This is enabled due to the fact that CTAs with the same indices are bound to the same GPM on multiple iterative launches of the kernel, therefore allowing the memory pages brought to a GPM's memory partition to continue to be local across subsequent kernel launches. Note that this locality does not show itself without the first touch page mapping policy as it does not increase L1.5 cache hit rates since the caches are flushed at kernel boundaries. However, MCM-GPU benefits significantly from more local accesses when distributed scheduling is combined with first touch mapping.

FT also allows for much more efficient use of the cache hierarchy. Since FT page placement keeps many accesses local to the memory partition of a CTA's GPM, it reduces pressure on the need for an L1.5 cache to keep requests from going to remote memory partitions. In fact using the first touch policy shifts the performance bot-

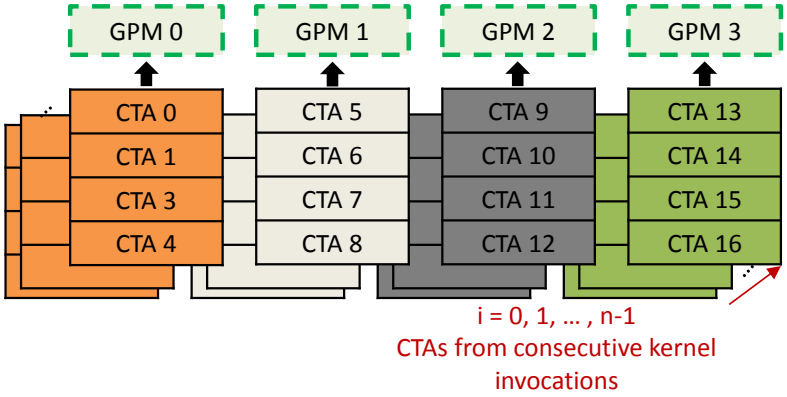


Figure 5.11: Exploiting cross-kernel CTA locality with First Touch page placement and distributed CTA scheduling.

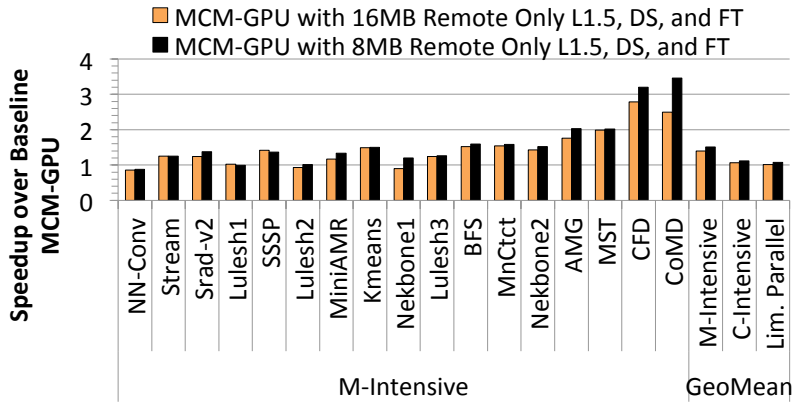


Figure 5.12: Performance of MCM-GPU with First Touch page placement.

tleneck from inter-GPM bandwidth to local memory bandwidth. Figure 5.12 shows this effect. In this figure, I show two bars for each benchmark — FT with DS and 16MB remote-only L1.5 cache, and FT with DS and 8MB remote-only L1.5 cache. The 16MB L1.5 cache leaves room for only 32KB worth of L2 cache in each GPM. This results in sub-optimal performance as there is insufficient cache capacity that is allocated to local memory traffic. I observe that in the presence of FT, an 8MB L1.5 cache along with a larger 8MB L2 achieves better performance. The results show that with this configuration the optimized MCM-GPU can obtain 51% /11.3% / 7.9% per-

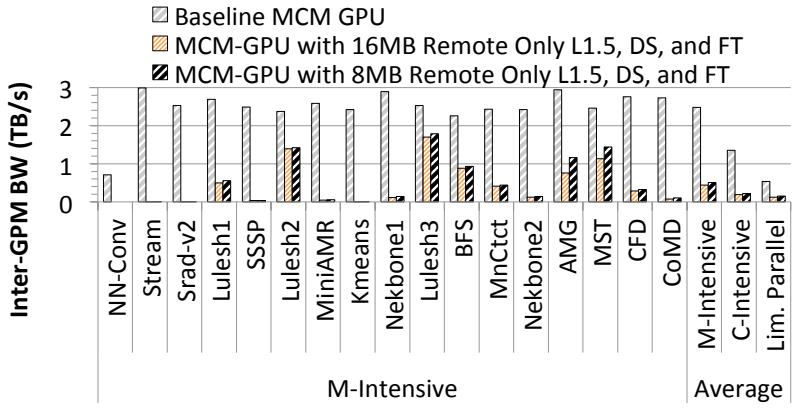


Figure 5.13: Reduction in inter-GPM bandwidth with First Touch page placement.

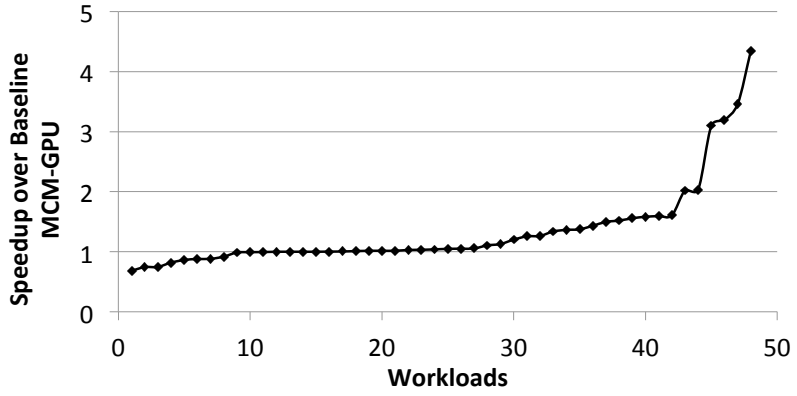


Figure 5.14: S-curve summarizing the optimized MCM-GPU performance speedups for all workloads.

formance improvements compared to the baseline MCM-GPU in memory-intensive, compute-intensive, and limited parallelism applications respectively. Finally, Figure 5.13 shows that with FT page placement a multitude of workloads experience a drastic reduction in their inter-GPM traffic, sometimes almost eliminating it completely. On average the proposed MCM-GPU achieves a 5× reduction in inter-GPM bandwidth compared to the baseline MCM-GPU.

5.3.6 Optimized MCM-GPU Performance Summary

Figure 5.14 shows the s-curve depicting the performance improvement of MCM-GPU for all workloads in this study. Of the evaluated 48 workloads, 31 workloads experience performance improvement while 9 workloads suffer some performance loss. M-Intensive workloads such as `CFD`, `CoMD` and others experience drastic reduction in inter-GPM traffic due to the optimizations and thus experience significant performance gains of up to 3.2× and 3.5× respectively. Workloads in the C-Intensive and limited parallelism categories that show high sensitivity to inter-GPM bandwidth also experience significant performance gains (e.g. 4.4× for `SP` and 3.1× for `XSBench`). On the flip side, I observe two side-effects of the proposed optimizations. For example, for workloads such as `DWT` and `NN` that have limited parallelism and are inherently insensitive to inter-GPM bandwidth, the additional latency introduced by the presence of the L1.5 cache can lead to performance degradation by up to 14.6%. Another reason for potential performance loss as observed in `Streamcluster` is due to the reduced capacity of on-chip writeback L2 caches [4] which leads to increased write traffic to DRAM. This results in performance loss of up to 25.3% in this application. Finally, I observe that there are workloads (two in the evaluation set) where different CTAs perform unequal amount of work. This leads to workload imbalance due to the coarse-grained distributed scheduling. The MCM-GPU architecture can be further optimized by taking advantage of this potential opportunity resulting in better performance.

In summary, I have proposed three important mircroarchitectural enhancements to the baseline MCM-GPU architecture: (i) a remote-only L1.5 cache, (ii) a distributed CTA scheduler, and (iii) a first touch data page placement policy. It is important

[4] L1.5 caches are set up as write-through to support software based GPU coherence implementation

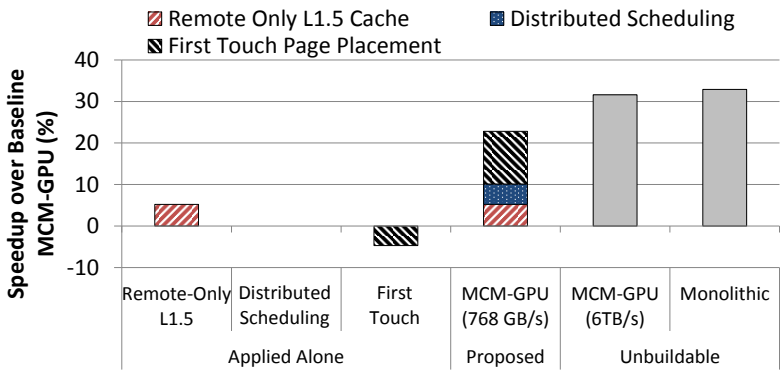


Figure 5.15: Breakdown of the sources of performance improvements of optimized MCM-GPU when applied alone and together. Three proposed architectural improvements for MCM-GPU almost close the gap with unbuildable monolithic GPU.

to note that these independent optimizations, work best when they are combined together. Figure 5.15 shows the performance benefit of employing the three mechanisms individually. The introduction of the L1.5 cache provides a 5.2% performance. Distributed scheduling and first touch page placement on the other hand, do not improve performance at all when applied individually. In fact they can even lead to performance degradation, e.g., -4.7% for the first touch page placement policy.

However, when all three mechanisms are applied together, I observe that the optimized MCM-GPU, achieves a speedup of 22.8% as shown in Figure 5.15. I observe that combining distributed scheduling with the remote-only cache improves cache performance and reduces the inter-GPM bandwidth further. This results in an additional 4.9% performance benefit compared to having just the remote-only cache while also reducing inter-GPM bandwidth by an additional 5%. Similarly, when first touch page placement is employed in conjunction with the remote-only cache and distributed scheduling, it provides an additional speedup of 12.7% and reduces inter-GPM bandwidth by an additional 47.2%. These results demonstrate that the proposed enhancements not only *exploit* the currently available data locality within

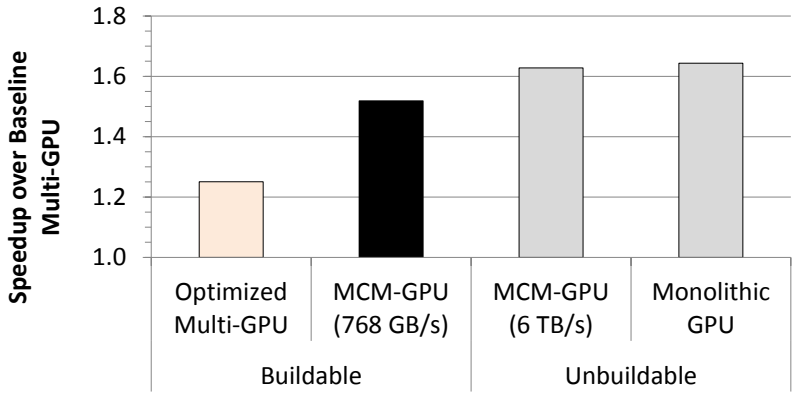


Figure 5.16: Performance comparison of MCM-GPU and Multi-GPU.

a program but also *improve* it. Collectively, all three locality-enhancement mechanisms achieve a 5× reduction in inter-GPM bandwidth. These optimizations enable the proposed MCM-GPU to achieve a 45.5% speedup compared to the largest implementable monolithic GPU and be within 8% of an equally equipped albeit *unbuildable* monolithic GPU.

5.3.7 MCM-GPU Performance vs Multi-GPU

A system with 256 SMs can also be built by interconnecting two maximally sized discrete GPUs of 128 SMs each. Similar to the MCM-GPU proposal, each GPU has a private 128KB L1 cache per SM, an 8MB memory-side cache, and 1.5 TB/s of DRAM bandwidth. I assume such a configuration as a maximally sized future monolithic GPU design. I assume that the two GPUs are interconnected via the next generation of on-board level links with 256 GB/s of aggregate bandwidth, improving upon the 160 GB/s commercially available today [128]. I assume the multi-GPU to be fully transparent to the programmer. This is accomplished by assuming the following two features: (i) a unified memory architecture between two peer GPUs, where both GPUs can access local and remote DRAM resources with load/store semantics, (ii) a

combination of system software and hardware which automatically distributes CTAs of the same kernel across GPUs.

In such a multi-GPU system the challenges of load imbalance, data placement, workload distribution and interconnection bandwidth discussed in Sections 5.2, are amplified due to severe NUMA effects from the lower inter-GPU bandwidth. Distributed CTA scheduling together with the first-touch page allocation mechanism (described respectively in Sections 5.3.4 and 5.3.5) are also applied to the multi-GPU. I refer to this design as a *baseline multi-GPU* system. Although a full study of various multi-GPU design options was not performed, alternative options for CTA scheduling and page allocation were investigated. For instance, a fine grain CTA assignment across GPUs was explored but it performed very poorly due to the high interconnect latency across GPUs. Similarly, round-robin page allocation results in very low and inconsistent performance across the benchmarks.

Remote memory accesses are even more expensive in a multi-GPU when compared to MCM-GPU due to the relative lower quality of on-board interconnect. I optimize the multi-GPU baseline by adding GPU-side hardware caching of remote GPU memory, similar to the L1.5 cache proposed for MCM-GPU. I have explored various L1.5 cache allocation configurations, and observed the best average performance with a half of the L2 cache capacity moved to the L1.5 caches that are dedicated to caching remote DRAM accesses, and another half retained as the L2 cache for caching local DRAM accesses. I refer to this as the *optimized multi-GPU*.

Figure 5.16 summarizes the performance results for different buildable GPU organizations and unrealizable hypothetical designs, all normalized to the baseline multi-GPU configuration. The optimized multi-GPU which has GPU-side caches outperforms the baseline multi-GPU by an average of 25.1%. The proposed MCM-GPU on

the other hand, outperforms the baseline multi-GPU by an average of 51.9% mainly due to higher quality on-package interconnect.

5.4 Related Work

Multi-Chip-Modules are an attractive design point that have been extensively used in the industry to integrate multiple heterogeneous or homogeneous chips in the same package. On the homogeneous front, IBM Power 7 [5] integrates 4 modules of 8 cores each, and AMD Opteron 6300 [4] integrates 2 modules of 8 cores each. On the heterogeneous front, the IBM z196 [3] integrates 6 processors with 4 cores each and 2 storage controller units in the same package. The Xenos processor used in the Microsoft Xbox360 [1] integrates a GPU and an EDRAM memory module with its memory controller. Similarly, Intel offers heterogeneous and homogeneous MCM designs such as the Iris Pro [7] and the Xeon X5365 [2] processors respectively. While MCMs are popular in various domains, I am unaware of any attempt to integrate homogeneous high performance GPU modules on the same package in an OS and programmer transparent fashion. To the best of my knowledge, this is the first effort to utilize MCM technology to scale GPU performance.

MCM package level integration requires efficient signaling technologies. Recently, Kannan et al. [79] explored various packaging and architectural options for disintegrating multi-core CPU chips and studied its suitability to provide cache-coherent traffic in an efficient manner. Most recent work in the area of low-power links has focused on differential signaling because of its better noise immunity and lower noise generation [114, 140]. Some contemporary MCMs, like those used in the Power 6 processors, have over 800 single-ended links, operating at speeds of up to 3.2 Gbps, from a single processor [56]. NVIDIA's Ground-Referenced Signaling (GRS) technol-

ogy for organic package substrates has been demonstrated to work at 20 Gbps while consuming just 0.54pJ/bit in a standard 28nm process [141].

The MCM-GPU design exposes a NUMA architecture. One of the main mechanisms to improve the performance of NUMA systems is to preserve locality by assigning threads in close proximity to the data. In a multi-core domain, existing work tries to minimize the memory access latency by thread-to-core mapping [37, 106, 163], or memory allocation policy [39, 54, 95]. Similar problems exist in MCM-GPU systems where the primary bottleneck is the inter-GPM interconnection bandwidth. Moreover, improved CTA scheduling has been proposed to exploit the inter-CTA locality, higher cache hit ratios, and memory bank-level parallelism [98, 115, 170] for monolithic GPUs. In case of MCM-GPU, distributed CTA scheduling along with the first-touch memory mapping policy exploits inter-CTA localities both within a kernel and across multiple kernels, and improves the efficiency of the newly introduced GPM-side L1.5 cache.

Finally, I propose to expose the MCM-GPU as a single logical GPU via hardware innovations and extensions to the driver software to provide programmer- and OS-transparent execution. While there have been studies that propose techniques to efficiently utilize multi-GPU systems [35, 41, 88, 96], none of the proposals provide a fully transparent approach suitable for MCM- GPUs.

5.5 Chapter Summary

Many of today's important GPGPU applications scale well with GPU compute capabilities, and future progress in many fields such as exascale computing and artificial intelligence will depend on continued GPU performance growth. The greatest challenge towards building more powerful GPUs comes from reaching the end of transistor density scaling, combined with the inability to further grow the area of a single

monolithic GPU die. In this chapter I proposed MCM-GPU, a novel GPU architecture that extends GPU performance scaling at a package level, beyond what is possible today. I do this by partitioning the GPU into easily manufacturable basic building blocks (GPMs), and by taking advantage of the advances in signaling technologies developed by the circuits community to connect GPMs on-package in an energy efficient manner.

I discuss the details of the MCM-GPU architecture and design locality aware optimizations for the MCM-GPU. I explore the interplay of hardware caches, CTA scheduling, and data placement in MCM-GPUs to optimize this architecture. I show that with these optimizations, a 256 SMs MCM-GPU achieves 45.5% speedup over the largest possible monolithic GPU with 128 SMs. Furthermore, it performs 26.8% better than an equally equipped discrete multi-GPU, and its performance is within 8% of that of a hypothetical monolithic GPU that cannot be built based on today's technology roadmap.

Chapter 6

UNDERSTANDING THE ENERGY EFFICIENCY OF MULTI-CHIP-MODULE GPGPUS AND THE DEPENDENCE ON THE MEMORY SUBSYSTEM

The previous chapter proposed the Multi-Chip-Module GPU (MCM-GPU) as a solution to address the impending GPGPU performance scalability problem. With the slowdown of transistor scaling and the hard optical limitations of lithography, GPGPUs are necessitated to embrace modular scaling as shown in Figure 6.1. Chapter 5 and other prior works [35, 49, 50, 118, 119, 162] have addressed the scalability of multi-chip-module GPGPUs from a performance standpoint for the on-package and off-package integration scenarios. However, considering the growing energy constraints of today's data centers, it is essential to understand the energy consumption and energy efficiency characteristics of this new GPU architecture.

This chapter perform an in-depth study of the scalability challenges of multi-chip-module GPGPUs that are exposed when energy efficiency expectations are taken into account. I develop a new GPGPU efficiency metric and propose a top-down instruction based GPGPU energy estimation framework, that together allow us to simultaneously reason about GPGPU performance scalability and energy costs of achieving the same [26].

6.1 Background and Motivation

Scaling GPU performance through the integration of multiple GPU-modules (GPMs) along with their local and remote memories similar to MCM-GPU (Chapter 5) results in a NUMA GPU architecture as shown in Figure 6.1. NUMA-based multi-module GPUs are expected to have unique characteristics affecting their performance and

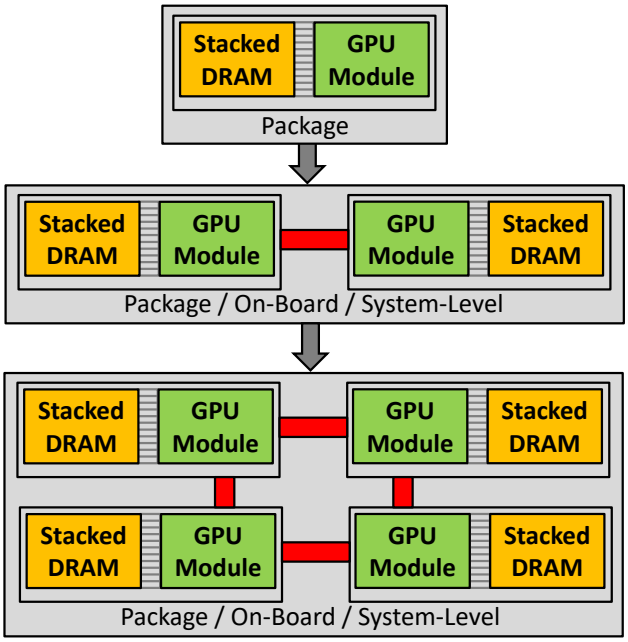


Figure 6.1: Scaling of future GPU designs via multi-module GPU architecture.

energy efficiency. For example, work partitioning and data locality nuances will dictate the inter-module communication requirements. Additionally, the GPMs could be integrated leveraging diverse integration domains and the available inter-module bandwidth may vary significantly based on the integration domain and signaling technology employed. While state-of-the-art on-board interconnects can easily provide 300 GB/s of interconnect bandwidth per GPU [11, 17], this is still 3× lower than the 900 GB/s of DRAM bandwidth available to GPUs today. Future on-package integration technologies utilizing high density interconnect along with high speed signaling [25, 167] are expected to provide substantially higher bandwidth that may match or exceed today's DRAM bandwidth but undoubtedly, DRAM bandwidth will continue to grow in the future as well.

Depending on the integration domain, inter-module communication costs may vary substantially. On-package communication has an energy overhead of 0.54 pJ/bit [141]

that is an order of magnitude lower than the energy overhead of on-board integration (10 pJ/bit) [25]. So while it is advantageous to integrate in the most efficient domain possible, on-package integration is not expected to scale to a very large number of modules due to package size limitations. On-board integration can also leverage efficient interconnect topologies such as high-radix switch chips [17], whereas on-package environments will likely utilize multi-hop technologies without having dedicated switches, as these are the most suitable for their planar nature and limited resources.

Although it is evident that multi-module GPUs can provide significant strong scaling performance improvements, this is likely to be coupled with high energy overheads when compared to traditional monolithic GPU scaling. As a result, it is important to take into consideration not only performance, but also energy efficiency, when designing and evaluating multi-module GPU architectures. Prior work has shown that it may be possible to build 4, 8, or even 16× larger GPUs than what exists today by integrating them either on-package or on a single PCB. Using the tools developed in this chapter, Figure 6.2 shows the relative increase in energy required to compute the solution for fixed problem sizes (average across 18 workloads) for a GPU built out of multiple GPMs with on-board integration. While this hypothetical 32× larger GPU may be able to ideally compute the solution to a problem 32× faster (if it could achieve ideal strong scaling), it is going to consume 2× more energy to do-so. Thus, for a fixed problem size, this hypothetical MCM-GPU is only half as energy efficient as the baseline GPU.

For many GPU users, this energy differential may not be consequential as they are solely focused on time to solution or their private datacenter has additional energy headroom to spare. However, professional datacenters and cloud service providers often operate at near peak energy threshold in order to achieve cost efficiency. There-

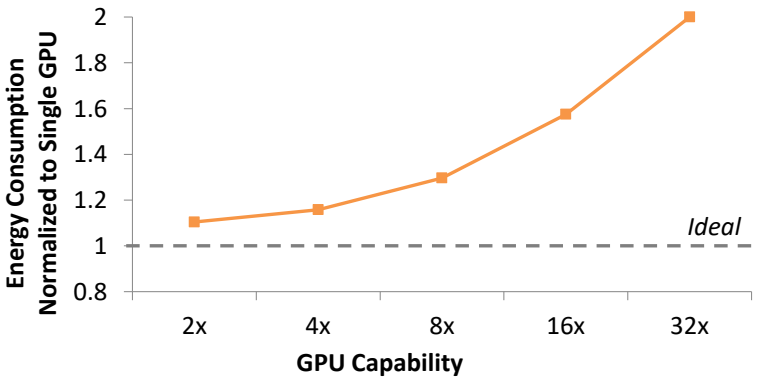


Figure 6.2: The average energy cost of strong scaling, when growing the number of GPMs using on-board integration.

fore, upgrading components to less energy efficient versions (despite them being higher performance) will likely lead to issues in power delivery, cooling, and provisioning. This trend towards energy *inefficiency* is what motivates me to quantify and analyze what minimum amount of energy growth GPUs must endure in the pursuit of strong scaling. By quantifying these effects one can then understand what the main architectural bottlenecks are that affect such energy-performance trade-offs and thus, how to optimize them.

To be able to effectively model energy efficiency, accurate performance and energy estimation models and simulators are needed. Commonly used GPU energy estimation frameworks adopt a bottom-up modeling approach [66, 102], where the energy cost of each key microarchitectural component is combined with the corresponding switching activities to determine the overall GPU energy consumption. While this potentially results in very accurate analysis, it also comes with some significant drawbacks. First, since most of the microarchitectural details of modern GPUs are confidential, researchers are forced to guess and assume particular organizations. This leads a lack of accuracy in energy projections, and necessitate the need for statistical (or hand-tuned) error correction methods (fudge factors) to be applied. Adopting

these error correction methods reduces the flexibility of the model [122]. In addition, a relatively complex retraining process needs to be carried out for every new GPU generation in order to achieve acceptable levels of fidelity. For example, my analysis showed that adopting a commonly used bottom-up energy model that was tuned for NVIDIA's Fermi architecture without retraining it to NVIDIA's Kepler generation architecture, led to an average error of over 100% when trying to project the energy consumption of an NVIDIA Kepler GPU.

This retraining process requires maintaining a set of complicated microbenchmarks that are designed to reverse engineer and isolate various microarchitectural details of any given design. With the rate of GPU microarchitectural innovation, reverse engineering the details of every new GPU generation is a very challenging task and models that rely on this process unsurprisingly lag behind hardware and become irrelevant. Therefore, to understand and project GPU energy into the future, GPU researchers need a different modeling approach that is more sustainable. Prior work by Shao et al. [156] has shown that top down models can in fact be both accurate and flexible in the CPU domain and I believe that a top-down energy modeling approach should be able to provide these qualities in the GPU domain as well.

6.2 EDPSE: Quantifying GPU Energy Efficiency at Scale

Comparing two individual hardware designs can easily be done using point metrics like performance, power or energy (e.g. Design A achieves performance X, while design B achieves performance Y). For systems with a (mostly) fixed set of hardware resources, this lets us focus on performance improvement as the figure of merit, because other metrics like energy consumption are unlikely to vary substantially without large changes in hardware. There are even metrics like energy delay product (EDP),

or ED^2, that explicitly combine energy and performance to allow comparisons between two substantially different designs on equal footing.

While very useful for comparing point-wise designs, these metrics are not suitable for exploring the efficiency of scalable GPU designs because they give us no notion of the scaling efficiency (or lack thereof) of an architecture as it evolves from 1–N GPU modules within a design. One prior approach that attempts to capture this effect, is Parallel Efficiency, a commonly used metric to quantify the efficiency of strong performance scaling on parallel systems [92]. Parallel efficiency is a measure of the fraction of ideal speedup that is realized with scaling. If the execution time with 1 processor is t_1 and the execution time with N processors is t_N, then:

$$Parallel\ Efficiency = \frac{t_1 \times 100}{N \times t_N} \tag{6.1}$$

However, we can see from equation 6.1, parallel efficiency does not take into account the energy cost of the system needed to achieve that parallel performance. Therefore, in order to make comparative decisions about energy efficiency scaling across a variety of design points, a metric that considers performance, energy, and the number of replicated resources that is needed. Thus I propose *EDP Scaling Efficiency* (EDPSE) to be able quantify these factors. EDPSE is defined as:

$$EDP\ Scaling\ Efficiency = \frac{EDP_1 \times 100}{N \times EDP_N} \tag{6.2}$$

EDP_1 is the EDP with one processor, N is the number of processors in the scaled configuration and EDP_N is the EDP with N processors. Extending the rationale of parallel efficiency (Equation 6.1), EDPSE is designed to measure the fraction of linear EDP scaling that is realized in a particular design. In a design that achieves linear performance speedup, strong scaling with N processors would reduce execution time (or delay) by a factor of N, while keeping the energy consumption constant; this

would lead to an EDPSE of 100%. If performance achieves sub-linear scaling or energy consumption increases as the GPU design grows, both factors will reduce the design's EDPSE [1] .

EDPSE can also be extended to provide different relative weights to performance and energy factors, reflecting different design priorities for a given architecture. In general, if an energy delay combination that takes the i^{th} power of delay i.e. ED^{i}P is deemed to be the metric of interest, then ED^{i}PSE can be defined as:

$$ED^iPSE = \frac{ED^iP_1 \times 100}{N^i \times ED^iP_N} \tag{6.3}$$

ED^iP_1 is the ED^{i}P metric with one processor and ED^iP_N is the ED^{i}P metric with N processors.

For GPU architects, evaluating designs based on EDPSE provides a relatively simple metric to understand the scaling efficiency achieved by future designs on an iso-energy efficiency basis. Simply put, the higher the EDPSE metric, the better the design. Considering the growing significance of energy efficiency in today's datacenters and personal computers (where GPUs may be the dominant energy cost), I believe that GPU architects of the future will have to satisfy EDPSE designs thresholds (e.g. 50%) to justify architectural improvements.

6.3 GPUJoule: A GPU Energy Estimation Framework

Having defined an appropriate metric to reason about the scaling efficiency of future multi-module GPUs, I now present the instruction based energy estimation framework for GPUs, called *GPUJoule*, that feeds into the EDPSE analyses later in this work. GPUJoule is based on publicly available information about GPU architectures to provide an accurate and reproducible estimation of GPU energy consumption.

[1] It is possible to achieve an EDPSE that is greater than 100% in cases of super-linear speedup or a decrease in energy consumption

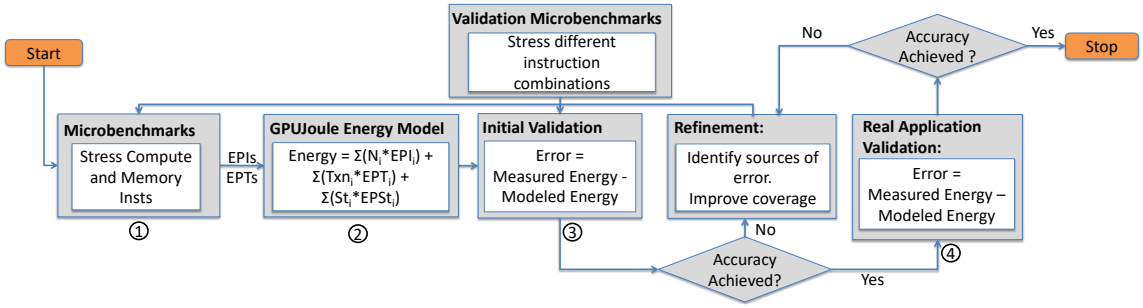


Figure 6.3: GPUJoule top-down instruction-based energy modeling methodology.

Its design leverages the insight that the total energy consumption of a GPU is the sum of the energy consumption of each instruction executed on the GPU, plus any constant overheads present within the GPU.

For example, a GPU is capable of executing compute instructions of type $1, 2, ...N_c$ and generates memory transactions of type $1, 2, ...N_m$ across the various levels of the memory hierarchy. To predict the total energy consumption of the GPU, I can thus estimate the Energy-Per-Instruction (EPI) and Energy-Per-Transaction (EPT) for both GPU pipeline instructions and memory system operations, while knowing very little about the specific microarchitectural implementation of the GPU itself. Utilizing the EPI and EPT information, the energy consumption of the GPU is then predicted as follows:

$$\begin{aligned} E_{GPU} = & \sum_{c=1}^{c=N_c} (EPI_c \times IC_c) + \\ & \sum_{m=1}^{d=N_m} (EPT_m \times TC_m) + \\ & (EPStall \times stalls) + \\ & (Const\_Power \times Execution\_Time) \end{aligned} \tag{6.4}$$

EPI_c and IC_c represent the energy-per-instruction and instruction count of *compute* instruction of type *'c'*, respectively. EPT_m and TC_m represent the energy-per-

transaction and transaction count of *memory* transaction of type *'m'*, respectively. $EPStall$ and $stalls$ represent the energy-per-stall of a compute lane stall and number of lane stalls, respectively which together account for the varying degree of parallelism in the GPU applications. $Const\_Power$ is the baseline idle power consumption of the GPU which accounts for the power consumed by voltage regulators, the power delivery network, I/O to the host, and the static power dissipation of the GPU.

Figure 6.3 shows the modeling framework of GPUJoule that generates the data which feeds into this energy estimate. Before diving into the specifics of each step I provide a high level overview of the broad GPUJoule workflow. At a high level, GPUJoule uses exhaustive set of microbenchmarks to stress the execution of different GPU instructions in isolation (1), while also utilizing the GPU's parallelism to average the behavior of each instruction across thousands of iterations and all compute units (also known as SMs) in the system. Similarly, the memory microbenchmarks carefully designed to isolate accesses and cause memory movement between different levels of the GPU memory hierarchy. These microbenchmarks allow me to derive EPI_c and EPT_m values for individual native ISA (PTX) instructions. The EPI and EPT estimates then combine with the compute instruction and memory transaction event counts to feed into the constitute the initial GPUJoule energy model (2). GPUJoule then iteratively improves upon the initial microbenchmark suite using two validation steps. First I design a set of synthetic microbenchmarks that combine different instruction types and correlate the modeled and measured energy values from real hardware to expose any coverage or instruction interaction issues overlooked in the initial microbenchmark design process (3). Finally, I validate the energy model by correlating the modeled and measured energy for real GPU applications (4). Overall, GPUJoule's estimation and validation process is fairly comparable to prior works that focus on instruction-based energy estimation in the server [83], mobile [134], and

1: **procedure** FMA-KERNEL(res, N)

▷ Location for result and number of iterations

2:

▷ Declare and initialize registers

$$\_\_asm\ volatile($$
$$\text{“ } .reg\ .f32\ \%r1;''$$
$$\text{“ } mov.f32\ \%r1, k1;''$$
$$...);$$

3: **for** $i = 0$ *To* $i < num\_iterations$ **do**

▷ Benchmark ROI operation

$$\_\_asm\ volatile($$
$$\text{“}fma.rn.f32\ \%r3,\ \%r1,\ \%r3,\ \%r2;''$$
$$\ldots$$
$$);$$

end

4: **end procedure**

Algorithm 6: Compute instruction microbenchmark example - FMA instruction

many-core architecture [156] domains but adjusted for the unique properties of GPU architectures.

6.3.1 Micro-Benchmark Construction

Modern GPUs support many native general purpose computation and data movement instructions such as `ADD`, `MUL`, `SQRT`, `AND` [129]. Depending on the instruction

and data type, each instruction will utilize a variety of functional units. To extract the energy expenditure across all instructions, I develop two classes of microbenchmarks; compute microbenchmarks that work at the PTX ISA level, and data movement microbenchmarks that move data between different levels of the GPU memory hierarchy.

The compute microbenchmarks are designed to execute a particular instruction repeatedly so that the steady state power and energy consumption of the instruction execution can be determined. When designing these benchmarks I intentionally disable compiler optimizations and use in-lined assembly to ensure that the benchmarks are faithfully executed on hardware. Algorithm 6 shows an example microbenchmark designed to stress an `FMA` instruction. In each benchmark, I first declare and initialize the registers (line 2) used by the microbenchmark before the region-of-interest (ROI) of the microbenchmark is reached. In the ROI of the microbenchmark (lines 3 and 4) the instruction of interest is executed on the GPU iteratively and the corresponding power and energy consumption are measured using NVIDIA provided power measurement tools. Microbenchmarks similar to Algorithm 6 have been developed for all compute instructions in the PTX ISA.

To stress the data movement operations that fetch data from different levels of the GPU memory hierarchy such as shared memory, L1 cache, L2 cache, and the DRAM a different type of benchmark is needed. These microbenchmarks are designed to first initialize a data set that completely fits within the particular level of the memory hierarchy (for which specifications are typically publicly available), and then consistently access the data set within that level of the hierarchy In order to isolate the data movement operations, I employ pointer-chasing microbenchmarks using known methodology defined previously [83, 134]. To account for the nuances of the GPU execution model, I ensure that memory accesses originating from a single warp can be coalesced to a single cacheline. I also ensure accesses stress the right level of the

Table 6.1: The NVIDIA Tesla K40 experimental platform.

(a) Important specifications of the GPU.

NVIDIA Tesla K40	
Architecture	Kepler
Process node	28nm
SM count	15
Shared memory/L1 cache	16KB/48KB, 32KB/32KB, 48KB/16KB
L2 cache	1.5MB
DRAM	12GB, 280 GB/s

(b) Energy of operations as measured on real hardware.

	EPI/EPT [nJ]	Energy per bit [pJ/bit]
PTX Instructions		
32b float ADD, MUL, FMA	0.06, 0.05, 0.05	N/A
32b int ADD, SUB	0.07, 0.07	N/A
32b bitwise AND, OR, XOR	0.06, 0.06, 0.06	N/A
32b float SINE, COS	0.10, 0.10	N/A
32b int MUL, MAD	0.13, 0.15	N/A
64b float ADD, MUL, FMA	0.15, 0.13, 0.16	N/A
32b float SQRT, LOG2, EXP2, RCP	0.02, 0.03, 0.08, 0.31	N/A
Data Movement Transactions		
Shared Memory to Register File	5.45	5.32
L1 Cache to Register File	5.99	5.85
L2 Cache to L1 Cache	3.96	15.48
DRAM to L2 Cache	7.82	30.55

memory hierarchy by managing warp-level and thread-block level data locality as needed. For example, when stressing the L2 cache, I ensure there are no hits due to inter-warp or intra-warp locality in the L1 caches.

6.3.2 Generating Energy Estimates

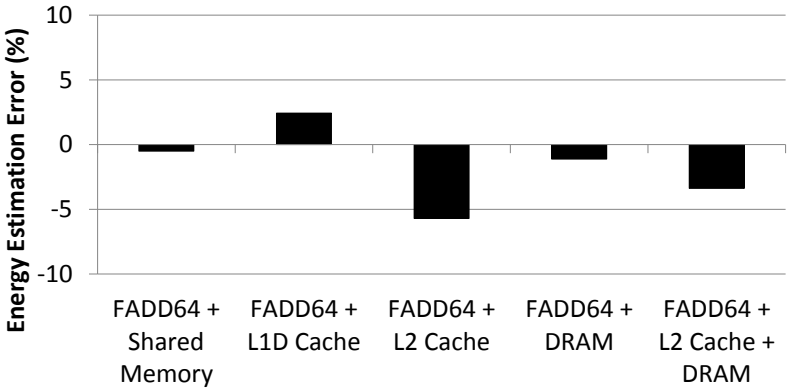
In this work I choose to validate GPUJoule using a NVIDIA Tesla K40 GPU [125] for which specifications are provided in Table 6.1a. I use the NVIDIA management library (NVML) [126] to query the on-board power sensor and look at the steady state power consumption incurred by different microbenchmarks to discern the EPI and EPT values for this GPU. The EPI for an instruction is computed as following:

$$EPI = \frac{(Power_{active} - Power_{idle}) \times Exec.\ Time}{Num.\ of\ Instructions} \tag{6.5}$$

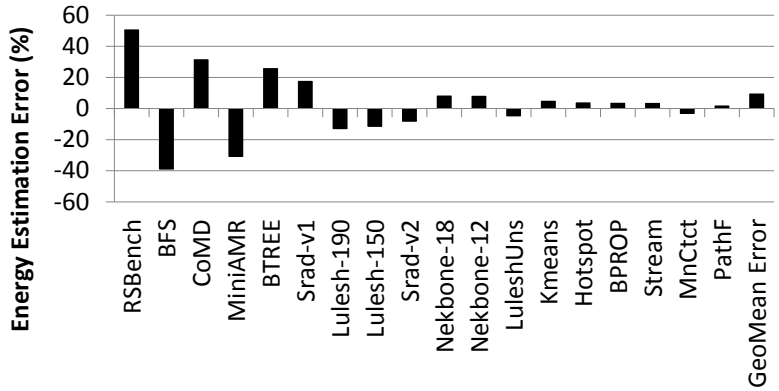
$Power_{idle}$ is the idle power consumption of the GPU. The EPT for data transfer operations is similarly determined.

Energy Per Instruction Estimates

Table 6.1b shows the EPI values for respective GPU compute instructions and the EPT values for GPU memory system operations. I observe that the energy consumption of GPU pipeline instructions have some variability depending on the instruction type and that both data type and width, perhaps unsurprisingly, affect the energy expended per operation. We can see that the energy cost of moving data from the shared memory or L1 cache to the registers is much lower than among other component of the memory hierarchy. Also, per bit energy expenditure increases as the data is brought from farther levels of the memory hierarchy. Moving data from the DRAM to the register file costs nearly 10 times the energy of moving data from L1 cache/shared memory to the register file, and delivering data to a floating point



(a) Energy estimation errors for microbenchmarks using combination of different types of instructions.



(b) Energy estimation errors for real GPU Applications.

Figure 6.4: GPUJoule energy estimation accuracy. Comparison of energy estimations with hardware measurements on the Tesla K40 GPU.

unit from the DRAM expends 80× the energy of performing the a computation on that data.

GPUJoule Validation to Silicon

To validate the energy estimates against silicon, I leverage the validation microbenchmarks as shown in Figure 6.3. Furthermore, I use an additional set of 18 GPU applications from Rodinia [48], Stream [116] and the CORAL benchmark suites [6],

summarized later in Table 6.2. Figure 6.4a shows the accuracy observed for the different microbenchmarks with mixed instruction types. I see that the energy estimation error is within +2.5% and -6% indicating GPUJoule has good fidelity compared to real hardware on a per instruction basis. Figure 6.4b shows the end-to-end relative error in energy estimation for the entire workload suite. While the correlation is very good for most benchmarks, with a 9.4% mean error across all benchmarks, I see that four applications experience an absolute error >30%. For applications such as RSBench and CoMD, I find the utilization of the memory subsystem is very low and the GPUJoule energy model may underestimate memory consumption. While I could have introduced fudge factors to try and improve the correlation of all outliers, that would reduce the generality of the model and its appropriateness for GPU scalability studies.

6.4 Energy Efficiency and the Future of Multi-Module GPUs

In this section I lay out the final details of the simulation methodology and simulated multi-GPM architecture. Then using GPUJoule and EDPSE, I perform a detailed analysis of the energy efficiency and performance scalability of several multi-module GPU configurations. I identify the key limitations affecting both energy and performance scaling and explore the solution space on the road to efficient scaling using multi-module GPUs.

6.4.1 Experimental Methodology

In this work, I integrate the GPUJoule energy model with a system level performance simulator used in prior works [25, 118]. I utilize a subset of 14 workloads selected from Table 6.2 that have enough inherent parallelism to fill a GPU with 32× the capability of the basic GPU module building block.

Table 6.2: GPU applications and their input sets. C: Compute intensive benchmarks and M: Memory bandwidth intensive benchmarks.

Benchmark	Input	Abbr.	Cat.
Back Propagation [48]	65536	BPROP	C
B+Tree [48]	1 Million	BTREE	C
Classic Molecular Dynamics [6]	49 bodies	CoMD	C
Hotspot [48]	1024x1024	Hotspot	C
Lulesh [6]	Unstrc Mesh	LuleshUns	C
Path Finder [48]	1 Million	PathF	C
RSBench [6]	1 Million	RSBench	C
SRAD (v1) [48]	100, 0.5, 502, 458	Srad-v1	C
Adaptive Mesh Refinement [6]	15,000	MiniAMR	M
Breadth First Search [48]	Graph1MW	BFS	M
Kmeans clustering [48]	819200	Kmeans	M
Lulesh [6]	size 150	Lulesh-150	M
Lulesh [6]	size 190	Lulesh-190	M
Nekbone solver [6]	size 12	Nekbone-12	M
Nekbone solver [6]	size 18	Nekbone-18	M
Mini Contact [6]	Mas1_2	MnCtct	M
SRAD (v2) [48]	2048x2048	Srad-v2	M
Stream Triad [116]	2^{26} elements	Stream	M

Simulated Architecture

The simulations configure a basic GPM to be similar to an NVIDIA Tesla K40 GPU. It comprises 16 SMs, equipped with a 32 KB L1 cache each, a shared L2 cache of 2 MB, and one HBM [89] memory stack with the DRAM bandwidth of 256 GB/s (see the 1-GPM configuration in Table 6.3). GPMs are expected to be individually smaller than the largest GPUs available today [14] because smaller die sizes yield significant cost and yield advantages for manufacturers [79, 167]. The GPUJoule energy model has been purposely trained on this GPU generation to increase the confidence of its projections and I note that because the compute to memory bandwidth ratio has remained relatively similar across GPU generations [14, 125, 130]; the conclusions drawn with these configurations should be applicable to multi-module GPUs with larger and more capable GPMs, as long as the baseline ratios between compute throughput, DRAM bandwidth, and I/O bandwidth do not change dramatically for a given GPM.

As shown in Table 6.3, I scale the number of GPU modules in the system from 1–32 and employ compute scheduling and DRAM page placement locality optimizations proposed in Chapter 5 and other prior works [25, 118]. In-line with these prior multi-module works, in 2-GPM configurations and beyond, I shift the L2 cache from being a dedicated memory-side cache to become a module-side cache, and the GPMs are interconnected in a ring topology. I evaluate three different per-GPM I/O bandwidth settings, shown in Table 6.4, representing bandwidth that is 2× lower, equal, and 2× larger than local GPM DRAM bandwidth. For example, current NVIDIA Volta GPUs support a inter-GPU to DRAM bandwidth ratio of 1:3. In this analysis I assume a slightly improved I/O to DRAM bandwidth ratio of 1:2 (called 1x-BW), to reflect future on-board integration capabilities. Similarly, a bandwidth ratio of 1:1 (called

2x-BW) reflects current projections for on-package I/O bandwidth used in Chapter 5 and the 2:1 ratio (called 4x-BW) corresponds to higher on-package BW settings that may become available through use of next generation signaling technologies [141].

Table 6.3: Simulated multi-module GPU configurations.

Config.	1-GPM	2-GPM	4-GPM	8-GPM	16-GPM	32-GPM
Number of Modules	1	2	4	8	16	32
Total SM count	16	32	64	128	256	512
L1 cache per SM	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB
Total L2 cache	2 MB	4 MB	8 MB	16 MB	32 MB	64 MB
Total DRAM Bandwidth	256 GB/s	512 GB/s	1024 GB/s	2048 GB/s	4096 GB/s	8192 GB/s

Energy Model Considerations

The vast majority of the GPM energy model parameters come directly from GPUJoule, as described in Section 6.3. To accurately reflect future scaling parameters I augment the energy model to reflect the use of HBM (vs GDDR5 memory) and the inclusion of signaling overheads for on and off package links. Specifically I use the published energy costs of GDDR5 and HBM technologies [131] to approximate the

Table 6.4: Simulated per-GPM I/O bandwidth.

Config.	Inter-GPM BW	Inter-GPM to DRAM BW	Integration Domain
1x-BW	128 GB/s	1:2	on-board
2x-BW	256 GB/s	1:1	on-package
4x-BW	512 GB/s	2:1	on-package

DRAM to L2 cache energy cost of 21.1 pJ/bit for the GPU system with HBM and I use the published [141] energy costs of on-package integration as 0.54 pJ/bit, and estimate 10 pJ/bit for on-board links though I later discuss the implications of this link potentially having up to 4× this energy overhead.

On-board integration of discrete GPM components comes with many per-GPM static energy overheads including voltage regulators, fans, system I/O, etc. To model on-board integration, I scale the static energy component linearly with the number of GPMs. However, as I will show later in this section, we can assume that certain on-platform components can be shared across multiple GPMs on package. Therefore, I scale only part of the measured static energy per GPM with the number of GPMs, I term this *Constant Energy Amortization*. To account for constant energy amortization under on-package integration, I assume that only 50% of the original per-GPM constant energy grows linearly with the number of GPMs. I also study the sensitivity to this parameter later in Section 6.4.3.

6.4.2 Understanding Energy Efficiency

Figure 6.5 shows the EDPSE of future GPU designs as they scale the number of on-package GPMs, using an on-package integration domain. I observe that compute intensive workloads achieve significantly higher EDPSE than their memory intensive

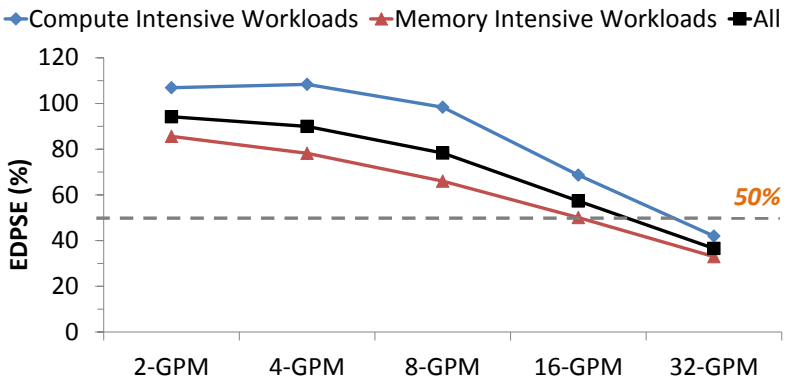


Figure 6.5: EDPSE for across compute intensive, memory intensive, and all workloads, for baseline on-package configuration (2x-BW).

counterparts. In fact, compute intensive workloads achieve an EDPSE that is higher than 100% for small GPM counts, as these workloads are able to leverage the increased caching resources and reduce their dependence on memory bandwidth, while incurring negligible or low energy costs.

A second trend is that EDPSE starts to decrease dramatically at high GPM counts. Maximum EDPSE achieved (when averaged across all workloads) is 94% under the 2-GPM configuration, and decreases to 36% when scaled to 32 GPMs. I find this is primarily due to the NUMA bandwidth limitations that are amplified when growing the number of GPMs in a ring topology. Based on the commonly used parallel efficiency threshold of 50%, I observe that on-package multi-module GPUs may start running into limitations when scaled beyond 16 GPMs.

To understand the decreasing EDPSE values I analyzed the speedup and energy consumption at each scaling step. Figure 6.6 shows the incremental speedup and energy increase *compared to the preceding point*, as the number of GPMs is scaled. I observe that the incremental speedup achieved at each scaling step decreases, sometimes dramatically. For example, scaling from the the 1-GPM to 2-GPM configuration

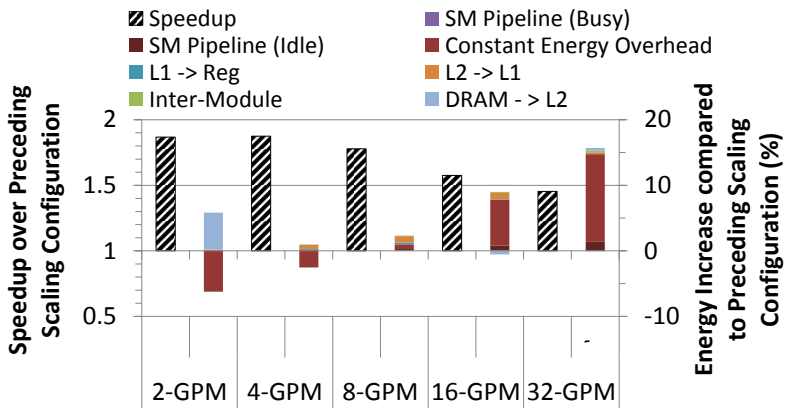


Figure 6.6: Performance speedup (left axis) and energy increase (right axis) compared to each preceding multi-GPM configuration. Energy consumption is broken down by component.

achieves 86.8% speedup, whereas scaling from the 16-GPM to the 32-GPM achieves only 47% speedup. I note, that experiments on the same set of applications on similarly equipped hypothetical monolithic GPU with unlimited on-chip bandwidth resources, achieves 80.8% speedup when scaling from the 16-GPM to 32-GPM point. I conclude that the observed performance penalty can primarily be attributed to the NUMA-related bottlenecks of multi-module GPUs.

Figure 6.6 also demonstrates the growing relative energy cost at each scaling step across a range of metrics available in the GPUJoule energy model. Scaling from the 16-GPM to 32-GPM configuration results in a 15.7% increase in energy consumption. Further analyzing the sources of energy growth at each step, I observe that when I first scale from the 1-GPM to 2-GPM configuration, the energy cost associated with new NUMA architecture becomes immediately visible in the form of increased DRAM to L2 cache energy consumption. However, due a slight super-linear performance speedup in some benchmarks and a reduction in constant energy overhead (as redundant components can be eliminated via on-package integration)

this additional data movement energy overhead is mostly offset. As a result, the 2-GPM configuration experiences only a minor decrease in EDPSE overall.

Unfortunately, as as the number of GPMs grows the energy consumption due to constant energy overhead also grows substantially. Analysis shows that this unfortunate effect is associated with an increasing number of GPM pipeline stalls (GPMs being idle and waiting on remote memory access) due to increased inter-module bandwidth pressure at large GPM counts. This, in turn, increases the relative contribution of static energy in high GPM-count designs and also contributes to the sharp decline in EDPSE. I conclude that as module based GPU designs become commonplace, NUMA effects and specifically inter-GPM bandwidth will be the primary challenge in achieving future GPU energy efficiency.

6.4.3 Optimizing for Energy Efficiency

Interconnect Bandwidth

Thus far, I have evaluated multi-module GPUs using the on-package integration domain. However the integration domain alone does not determine the intra-GPM bandwidth that is available in a system and architect's implementation choices, may also change the available bandwidth by 2×, either lower or higher. Figure 6.7 shows the impact of using different interconnect bandwidths across both on-package and on-board integration domains as described in Table 6.4. Inter-module bandwidth has pronounced effects on EDP scaling efficiency, and at high GPM counts, EDPSE improves by a factor of 3 when inter-module bandwidth increases by a factor of 4. This supports the conclusion that providing adequate levels of inter-module bandwidth is going to be the most important factor in maintaining high levels of energy efficiency in future multi-module GPUs.

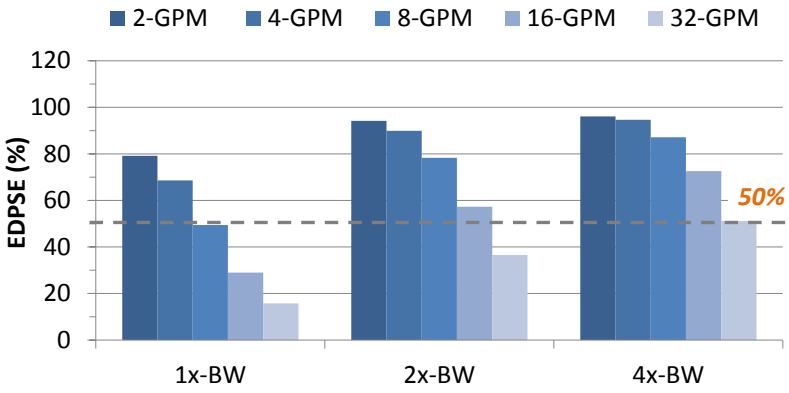


Figure 6.7: EDPSE as a function of interconnect bandwidth settings.

Interconnect Energy

As the number of GPMs in future GPUs grow the amount of data that is transferred on the inter-module links will increase. Common architectural practice place significant importance on reducing the intrinsic energy cost of interconnect technologies. Yet Figure 6.6 indicates that the overall GPU energy growth in high module-count systems that can be attributed to this increased data movement is relatively low (the "inter-module" stack in Figure 6.6). To understand this issue better, I performed a point study on the GPU's overall energy sensitivity when varying the inter-module interconnect energy consumption, but leaving bandwidth unchanged.

Using the 32-GPM design in an on-board integration domain (described as 1x-BW), I increased the per/bit interconnect energy cost by a factor of 2 and 4× over the baseline (10 pJ/bit). I observe that even with a 4× increase in the interconnect energy cost, the net impact on the EDPSE is below 1%. This is a significant result, when we consider that there is nearly a 2× improvement in EDPSE when doubling the inter-GPM bandwidth from the 1x-BW to 2x-BW configurations I conclude that, counter to design trends in monolithic GPUs, multi-module GPU architects should

not focus on driving down the intrinsic energy cost of interconnect technologies, but instead work towards maximizing bandwidth density, even at the expense of per bit transfer energy. The right architectural trade off may in fact be to prefer the highest bandwidth interconnect available, regardless of energy cost, because it leads to the highest possible global EDPSE. For example, my analysis shows that if the 4× higher interconnect energy cost could be used to achieve 2× higher interconnect bandwidth, it would cause 8.8% increase in EDPSE for a 32-GPM design.

Impact of Integration Domain

On-package integration comes not only with improved interconnect technologies that enable significant bandwidth advantages, but also the opportunities to reduce constant energy overheads. Tighter on-package integration of GPMs provides an opportunity to share the previously per-GPM energy burden of various on-platform components such as cooling, power delivery, etc. Because at high GPM counts, SM idle time, and thus constant energy overheads is a significant factor in overall energy efficiency, the effect of overhead amortization among on-package GPMs can be significant. I analyze a hypothetical 32-GPM system using on-package integration (with a 2x-BW configuration) where fixed energy overheads can not be amortized or amortized at a 50% rate, as described in Section 6.4.1. When compared to having no amortization, on-package integration achieves an impressive 22.3% decrease in absolute energy consumption and 8.1% increase in EDPSE. If the assumed amortization rate is reduced to 25% for example, then the energy saving would reach 10.4% on average compared to having no amortization at all, with a 3.5% increase in EDPSE.

Impact of an On-Board High-Radix Switch

To alleviate the impact of low inter-module bandwidth in on-board integration sce-

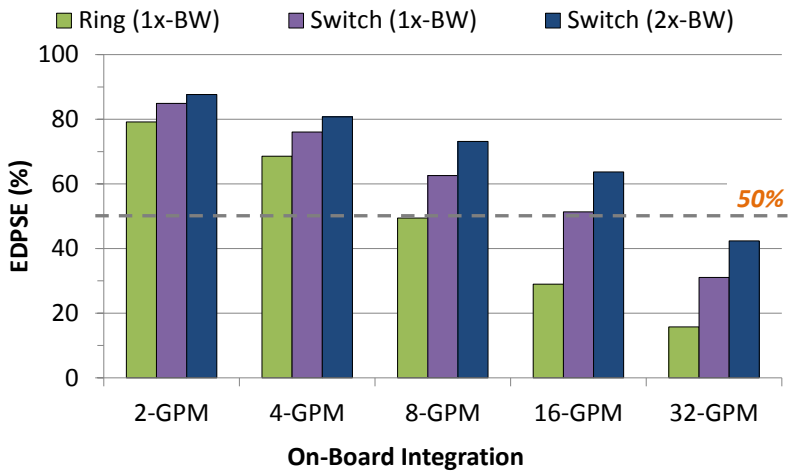


Figure 6.8: EDPSE for on-board ring and switched interconnection networks

narios, GPU manufacturers have recently introduced high-bandwidth and high-radix switch chips for these systems [15, 16, 17]. Compared to ring topologies, switched networks reduces the number of hops between source and destination, improving inter-module bandwidth congestion, despite absolute inter-GPU bandwidth remaining unchanged. Figure 6.8 shows the EDPSE achieved for on-board multi-module GPUs in the presence of a switched network. [2] I see that the introduction of a switch over a basic ring topology can improve EDPSE by nearly 2× in the 32-GPM case, despite no absolute change in interconnect-bandwidth. This further underscores the importance of reducing NUMA-related inter-module bandwidth bottlenecks at all cost, not just via increased link bandwidth, but also via network topology innovations.

6.4.4 Decomposing EDSPE Improvements

In this work I have argued EDSPE is the appropriate metric for analyzing the efficiency of multi-module GPU designs. However, there is a risk in using metrics such as EDSPE because individual constraints on either energy or performance can

[2] I assume an additional 10 pJ/bit data movement energy cost through the switch.

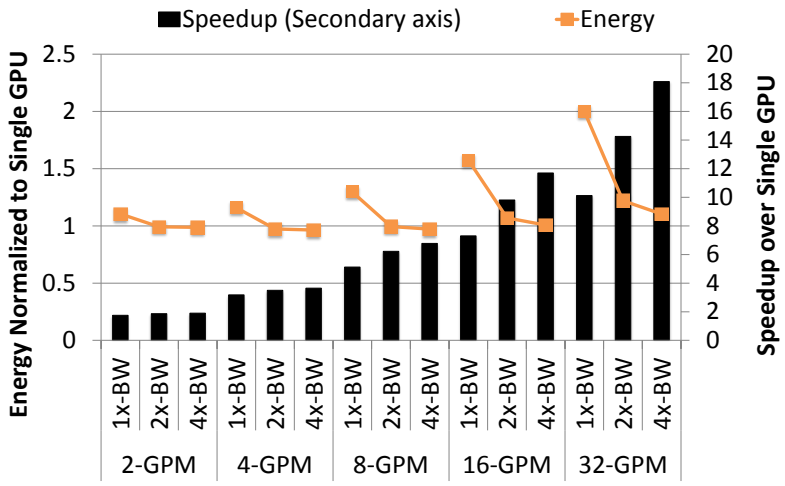


Figure 6.9: Speedup and energy consumption when varying interconnect bandwidth, and applying on-package energy optimization at the 2x-BW and 4x-BW points

be overlooked by designers. While variations of EDPSE can be constructed (as shown in Section 6.2) to try and capture these requirements, understanding the relative contributions of energy and performance into these synthetic metrics is still important.

Figure 6.9 summarizes the absolute speedup and energy expenditure across all GPM counts, and at all three bandwidth configurations. This data takes into account the aforementioned optimization allowing amortization of constant energy across all GPMs when moving from on-board (1x-BW) to on-package domains (2x-BW and 4x-BW), but still assumes a ring topology in all integration domains. As the number of GPMs are scaled , we can see that at a higher number of modules (GPM-8, 16, and 32), the achieved speedup is primarily associated with improvements in inter-GPM bandwidth. So significant is this effect, that a 16-GPM design with 2x-BW will outperform a 32-GPM design with only half the inter-module bandwidth, while expending just half the energy. In fact, a 32-GPM system configured with just 1x-BW (in an on-board domain) can reduce the energy required to compute a fixed size problem by doing nothing other than increasing the inter-GPM bandwidth by a factor

of four, by 27.4% on average. Furthermore, if the integration is subsequently moved to the on-package domain, and the effect of constant energy amortization is taken into account, this energy reduction increases to 45% on average.

I conclude that improving inter-GPM interconnect technologies and topologies, along with less traditional architecture measures, such as sharing the per-GPM platform related constant energy overheads across multiple GPMs will play a crucial role in making strong scaling with multi-module GPUs worthwhile from an energy efficiency point of view. My findings indicate that GPU architects will have to start making new design trade offs between interconnect topologies, link bandwidth, link energy, integration domains and to create opportunities to holistically share the overall energy burden of the GPU most efficiently. At extreme scales, architects may be forced to turn to extreme measures such as reallocation of costly on-chip pin-outs to re-balance local DRAM bandwidth versus inter-GPM bandwidth if the ratio of local to remote memory access continues to skew towards the latter.

6.4.5 Discussion

This work illuminates multiple key energy efficiency trends in future multi-module GPUs. I identify that without architecture and system level innovations, multi-module GPUs will quickly run into energy efficiency concerns when trying to scale performance at all costs. In light of these observations, traditional approaches to energy efficiency improvement within the GPU modules themselves will only manifest as second order effects in the future. The results highlight the pressing need for future research to focus on reducing the impact of the NUMA effects on multi-module GPUs. While prior works have attempted to address similar aspects in NUMA CPU systems [42, 54, 111, 171], techniques appropriate for multi-module GPUs deserve a close attention. Furthermore, techniques such as locality aware thread-block (CTA)

scheduling and data placement [25, 168], sophisticated cache management strategies [28, 51, 101, 118, 146, 147, 165], and data compression techniques [27, 90, 99, 169], need to be re-applied not just within today's GPU, but now among GPU modules. In addition, system-level techniques that reduce the impact of constant power in the presence of large number of GPU modules, either via integration technology innovations or through aggressive and intelligent clock-gating, can measurably improve energy efficiency of multi-module GPUs.

6.5 Related Work

GPU energy efficiency has been addressed in many prior works [19, 61, 80, 97, 99, 102, 112, 135, 136, 146, 148, 154, 155]. However this work was all done in context of monolithic die GPUs, and focused on minimizing energy consumption through microarchitectural innovation. More recently, Milic et al. [118] found that inter-module bandwidth plays a key role in the performance scalability of multi-modules GPUs (similar to my finding in Chapter 5) but did not address the concern of energy efficiency in these proposals. Vijayaraghavan et al. [167], perform a first order characterization of performance, power, and temperature using a specific multi-module GPU within an exascale node architecture (ENA). However, they do not address the scalability issues inherent in multi-module GPU designs.

Inspiring the design of GPUJoule, Wu et al. [174] design a machine learning based power model that estimates the power consumption of future chips. However, such approaches do not provide insights into the energy consumption trends or understanding of specific bottlenecks. To achieve this kind of detail, most prior work takes a bottom-up approach to power modeling; such as Leng et al. [102] who design a widely used cycle-level architecture-level power model based on McPat [107]. Guerreiro et al. [62] have also proposed a DVFS-aware power model for GPUs. Bottom up approaches

have key advantages by offering cycle-level power estimations and an understanding of the power consumption of fine-grained microarchitectural structures. However, as described in Section 6.1, these and other bottom-up power models [18, 66, 109, 159] are very difficult to maintain and keep current.

Most similar to GPUJoule, Kestor et al. [83], Pandiyan et al. [134] and Shao et al. [156] take top down approaches to characterize the energy consumption of server, mobile and the XeonPhi many-core processors respectively. Since these models were developed for CPU-like processors, they do not handle the subtleties of GPUs and can not be easily re-applied but reinforce the value of the GPUJoule approach.

6.6 Chapter Summary

The future of GPU computing relies on achieving strong performance scaling using modular designs, in an energy efficient manner. While prior works including the work proposed in Chapter 5 had addressed the performance scalability of multi-chip-module GPGPUs, the aspect of energy efficiency has been largely unexplored. In this chapter, I propose a new GPU efficiency scaling metric and a novel energy projection tool, that collectively let us reason about both the performance and energy efficiency of future multi-module GPGPUs. The in-depth scalability analysis reveals two key findings. First, I demonstrate that the dominant factor affecting the energy efficiency is the NUMA nature of these multi-chip-module GPGPUs. Consequently, congested inter-GPM interconnect in future designs increases GPM idle time, hampering performance scalability while simultaneously exposing the (relatively) increasing overhead of constant energy components in the system. Second, I demonstrate how an analysis-driven choice of interconnect technology, can provide counter-intuitive results and encourage architects to make energy-inefficient choices locally (in the inter-GPM interconnect) to help maximize energy efficiency globally. Using this analysis, I show it is possible

to reduce future multi-chip-module GPGPU energy consumption growth from over 100% to just 10% (compared to a single GPU) while improving strong scaling performance by a factor of 18×, paving the way to further architectural enhancements that will enable aggressive strong scaling performance that are not at odds with GPU energy efficiency. Furthermore, the findings presented in this chapter underscores the importance of data locality in future multi-chip-module GPGPUs and motivates further research focus to address the NUMA-effects within these multi-chip-module GPUs via architectural or system-level GPM locality management.

Chapter 7

CONCLUSION

The efficiency of the memory subsystem and the data delivery mechanisms have become first order design concerns as general purpose computing advances towards the era of exascale computing. The memory subsystem not only has a significant impact on the overall performance of the CMPs and GPGPUs, but also on their energy efficiency. As presented in the prior chapters, my thesis characterizes the performance and energy bottlenecks in modern CMPs and GPGPUs, and proposes practical microarchitecture and system architecture solutions to address the same. Specifically, my thesis proposes three novel microarchitectural techniques, a novel GPGPU system architecture, a new scaling efficiency metric, and an energy estimation framework that together advance the state-of-the-art general-purpose processor design, while enhancing the performance, scalability, and energy efficiency significantly.

In Chapter 2, I demonstrate that the impact of the performance critical CMP on-chip resource, the last-level-cache (LLC), can be significantly improved if the LLC is managed while considering its interaction with other memory system components, especially the main memory. With insights garnered via a detailed characterization of the reuse behavior of cache lines and their memory access costs, this thesis proposes a novel, low-overhead LLC management technique, *ReMAP*. ReMAP considers the current and predicted future reuse in conjunction with the diverse memory access costs in order to preserve highly valuable cache lines in the LLC. In doing so, ReMAP is able to achieve efficient cache management and superior performance in CMP systems.

While sophisticated LLCs have a pronounced influence on the performance of state-of-the-art CMPs, this responsibility is shouldered by the on-chip data cache in

modern GPGPU architectures. Relative to the massive number of concurrent threads executing on the GPU, the available on-chip data cache capacity happens to be extremely constrained. As a result, many prior works have directed their focus towards alleviating the constrained cache capacity problem in GPUs. Rather than taking approaches that reduce the TLP, this thesis explores and identifies GPU specific characteristics that can be leveraged to significantly improve the effective cache utilization, while keeping the TLP intact. Chapter 3 proposes a simple cache bypassing and cache line size selection method, *ID-Cache*, that leverages memory divergence information to retain only the most valuable data in the cache. Orthogonally, Chapter 4 proposes *LATTE-CC*, which leverages the GPU's inherent latency tolerance feature to adaptively compress the data cache. Such an adaptive technique allows us to navigate a three-way trade-off between the data value compressibility and compression latency of different compression algorithms, and the GPU latency tolerance to enhance the effective data cache capacity on GPU systems. These methods provide significant performance improvements while also providing considerable energy savings.

Complementing the aforementioned microarchitectural techniques, my thesis recognizes the need for system architecture innovations to ensure continued performance scalability of GPGPUs in the face of slowing Moore's law. Due to the impending end of transistor scaling and the optical limitations of lithography, it might no longer be possible to scale GPU performance in the traditional manner i.e., by increasing the number or transistors on a single die or by increasing the die sizes of GPUs. In this thesis, I propose a novel GPU architecture called *MCM-GPU*, that attains performance scaling by integrating multiple GPU-modules (GPMs) within a package (Chapter 5). Taking a step further, Chapter 5 also proposes architectural techniques spanning the GPM/GPU system architecture, which work together to alleviate the inherent NUMA side-effects of the MCM-GPU architecture. Thus, the MCM-GPU

architecture provides a promising path forward to ensure GPU performance scaling well into the future with modular scaling of GPMs.

Beyond ensuring performance scalability, it is equally important to consider the energy cost of scaling as we cater to the energy efficiency expectations of the future. Chapter 6 presents an in-depth study of the energy consumption and energy efficiency characteristics of modular scaling of GPMs in MCM-GPU like architectures. To allow us to understand and reason about the performance and energy costs of such scalable GPUs, I develop a novel efficiency scaling metric, *EDP Scaling Efficiency* (EDPSE), and an instruction-based GPU energy estimation tool, *GPUJoule*. Together, EDPSE and GPUJoule enable quantification of energy efficient scalability in such multi-modular GPU systems. Through the detailed analysis presented in Chapter 6, I find multiple key trends that are likely to impact future GPU energy efficiency. Contrary to common knowledge, I find that neither the energy efficiency of the GPM microarchitecture, nor the intrinsic energy cost of data movement would play a primary role in the overall energy efficiency of these GPUs. In fact, the inherent NUMA-effects would form the key energy efficiency bottleneck in the future. These findings further underscore the significance of data locality optimizations within the memory subsystem of GPUs described in this thesis.

Overall, my thesis advances the state-of-the-art general purpose processor design as follows:

- This thesis offers a detailed characterization of the reuse behavior, memory access costs, and data value compressibility of cache lines in CMP and GPGPU systems across a broad set of applications. These applications span a variety of domains including machine learning, audio processing, HPC, and scientific simulations. The characterizations presented offer key insights towards holistic management of the memory subsystem resources on CMPs and GPUs.

- This thesis illuminates the benefits of managing the LLC in CMPs and the L1 data cache in GPUs while considering their interaction with the other seemingly independent aspects of the system architecture, which eventually allow for the most valuable cache lines to be retained in these caches. Building on this insight, this thesis proposes targeted microarchitecture level solutions that help manage the crucial cache resources within the memory subsystem of general purpose processors. In addition to ReMAP that improves CMP performance by as much as 13%, this thesis proposes ID-Cache and LATTE-CC that improve GPGPU performance by an average of 71% and 19.2%, respectively. The high performance of the proposed schemes is enabled only due to the identification of precise set of information from across the system architecture such as main memory access cost (ReMAP), degree of memory divergence (ID-Cache), and system latency tolerance (LATTE-CC), which can enhance the effectiveness of CMP LLCs and GPU L1 data caches.

- At the broader system architecture level, this thesis proposes the novel MCM-GPU architecture and thus presents a promising path forward for continued performance scalability of GPUs in the face of slowing Moore's law. The MCM-GPU architecture achieves performance scaling via multi-modular integration of GPU Modules (GPMs); thereby decoupling the advancement of GPU architectures from the transistor scaling that is expected to come to a halt. Taking a step further, this thesis carries out an in-depth analysis of the energy consumption and energy efficiency trends of scaling GPU performance with multi-module integration. The detailed study presented here illuminates surprising trends in future multi-module GPUs, including the fact that neither the GPM microarchitecture energy efficiency, nor the intrinsic energy costs of data movement

are going to be first order concerns in these GPUs. These results are all the more meaningful considering that common knowledge places significant stress on both those factors as the key drivers of energy efficiency in the future. My work on the MCM-GPU architecture forms the first step in the design of scalable GPGPU architectures in the future and more importantly, calls for future research focus to be directed towards managing the NUMA-side effects in these systems, at scale.

- Finally, this thesis presents a new efficiency scaling metric, EDPSE, that considers both performance scalability of systems, as well as the energy overhead of achieving the performance scaling. In addition, to help analyze the energy consumption of current and future GPUs with relative ease, this thesis presents a new GPU energy estimation framework, GPUJoule. EDPSE and GPUJoule together are particularly well suited for exploratory studies for future GPUs.

In summary, my thesis offers novel insights pertaining to data reuse, data compressbility, performance scalability, and energy efficiency in modern computing systems, that are backed by detailed characterizations and evaluations of hardware proposals. Building on these insights will enable future CMP and GPGPU architects to design high-performance and energy efficient general purpose processors.

[1] "Xenos: XBOX360 GPU", URL http://fileadmin.cs.lth.se/cs/Personal/Michael_Doggett/talks/eg05-xenos-doggett.pdf, accessed: 2016-08-19 (2005).

[2] "The Xeon X5365", URL http://ark.intel.com/products/30702/Intel-Xeon-Processor-X5365-8M-Cache-3_00-GHz-1333-MHz-FSB, accessed: 2016-08-19 (2007).

[3] "IBM zEnterprise 196 technical guide", URL http://www.redbooks.ibm.com/redbooks/pdfs/sg247833.pdf, accessed: 2016-08-19 (2011).

[4] "AMD server solutions playbook", URL http://www.amd.com/Documents/AMD_Opteron_ServerPlaybook.pdf, accessed: 2016-08-19 (2012).

[5] "IBM power systems deep dive", URL http://www-05.ibm.com/cz/events/febannouncement2012/pdf/power_architecture.pdf, accessed: 2016-08-19 (2012).

[6] "Coral benchmarks", URL https://asc.llnl.gov/CORAL-benchmarks/ (2014).

[7] "The compute architecture of Intel Processor Graphics Gen8", URL https://software.intel.com, accessed: 2016-08-19 (2015).

[8] "Switch-IB 2 EDR switch silicon - world's first smart switch", URL http://www.mellanox.com/related-docs/prod_silicon/PB_SwitchIB2_EDR_Switch_Silicon.pdf, accessed: 2016-06-20 (2015).

[9] "ConnectX-4 VPI single and dual port QSFP28 adapter card user manual", URL http://www.mellanox.com/related-docs/user_manuals/ConnectX-4_VPI_Single_and_Dual_QSFP28_Port_Adapter_Card_User_Manual.pdf, accessed: 2016-06-20 (2016).

[10] "Inside pascal: Nvidia's newest computing platform", URL https://devblogs.nvidia.com/parallelforall/inside-pascal, accessed: 2016-06-20 (2016).

[11] "NVIDIA NVLink high-speed interconnect", URL http://www.nvidia.com/object/nvlink.html, accessed: 2016-06-20 (2016).

[12] "The TWINSCAN NXT:1950i dual-stage immersion lithography system", URL https://www.asml.com/products/systems/twinscan-nxt/twinscan-nxt1950i/en/s46772?dfp_product_id=822, accessed: 2016-11-18 (2016).

[13] "Titan : The world's #1 open science super computer", URL https://www.olcf.ornl.gov/titan/ (2016).

[14] "NVIDIA Volta unvieled", URL https://www.anandtech.com/show/11367/nvidia-volta-unveiled-gv100-gpu-and-tesla-v100-accelerator-announced (2017).

[15] "NVIDIA DGX-1", URL https://www.nvidia.com/en-us/data-center/dgx-1/ (2018).

[16] "NVIDIA HGX-2", URL https://www.nvidia.com/en-us/data-center/hgx/ (2018).

[17] "NVIDIA NVSWITCH", URL http://images.nvidia.com/content/pdf/nvswitch-technical-overview.pdf (2018).

[18] Abe, Y., H. Sasaki, S. Kato, K. Inoue, M. Edahiro and M. Peres, "Power and performance characterization and modeling of GPU-accelerated systems", in "Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium", (2014).

[19] Adhinarayanan, V., I. Paul, J. L. Greathouse, W. Huang, A. Pattnaik and W. c. Feng, "Measuring and modeling on-chip interconnect power on real hardware", in "Proceedings of the IEEE International Symposium on Workload Characterization", (2016).

[20] Alameldeen, A. R. and D. A. Wood, "Adaptive cache compression for high-performance processors", in "Proceedings of the 31st Annual International Symposium on Computer Architecture", (2004).

[21] Alameldeen, A. R. and D. A. Wood, "Frequent pattern compression: A significance-based compression scheme for L2 caches", (2004).

[22] Arelakis, A., F. Dahlgren and P. Stenstrom, "HyComp: a hybrid cache compression method for selection of data-type-specific compression methods", in "Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture", (2015).

[23] Arelakis, A. and P. Stenstrom, "A case for a value-aware cache", IEEE Computer Architecture Letters **13**, 1, 1–4 (2014).

[24] Arelakis, A. and P. Stenstrom, "SC2: A statistical compression cache scheme", in "Proceedings of the 41st Annual International Symposium on Computer Architecuture", (2014).

[25] Arunkumar, A., E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu and D. Nellans, "MCM-GPU: Multi-Chip-Module GPUs for continued performance scalability", in "Proceedings of the 44th Annual International Symposium on Computer Architecture", (2017).

[26] Arunkumar, A., E. Bolotin, D. Nellans and C.-J. Wu, "Understanding the future of energy efficiency in multi-module GPUs", in "(to appear) Proceedings of the IEEE International Symposium on High Performance Computer Architecture", (2019).

[27] Arunkumar, A., S.-Y. Lee, V. Soundararajan and C.-J. Wu, "LATTE-CC: Latency tolerance aware adaptive cache compression management for energy efficient GPUs", in "Proceedings of the IEEE International Symposium on High Performance Computer Architecture", (2018).

[28] Arunkumar, A., S.-Y. Lee and C.-J. Wu, "ID-Cache: Instruction and memory divergence based cache management for GPUs", in "Proceedings of the IEEE International Symposium on Workload Characterization", (2016).

[29] Arunkumar, A. and C.-J. Wu, "ReMAP: Reuse and memory access cost aware eviction policy for last level cache management", in "Proceedings of the 32nd IEEE International Conference on Computer Design", (2014).

[30] Baek, S., H. G. Lee, C. Nicopoulos, J. Lee and J. Kim, "ECM: Effective capacity maximizer for high-performance compressed caching", in "Proceedings of the IEEE 19th International Symposium on High Performance Computer Architecture,", (2013).

[31] Bakhoda, A., G. Yuan, W. W. L. Fung, H. Wong and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator", in "Proceedings of the International Symposium on Analysis of Systems and Software", (2009).

[32] Basu, A., N. Kirman, M. Kirman and M. Chaudhuri, "Scavenger: A new last level cache architecture with global block priority", in "Proceedings of the 40th International Symposium on Microarchitecture", (2007).

[33] Beckmann, N. and D. Sanchez, "Talus: A simple way to remove cliffs in cache performance", in "Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture", (2015).

[34] Belady, L. A., "A study of replacement algorithms for a virtual storage computer", in "IBM Syst. J.", vol. 5 (1966).

[35] Ben-Nun, T., E. Levy, A. Barak and E. Rubin, "Memory access patterns: the missing piece of the multi-GPU puzzle", in "Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis", (2015).

[36] Binkert, N., B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill and D. A. Wood, "The gem5 simulator", SIGARCH Computer Architecture News **39**, 2, 1–7 (2011).

[37] Blagodurov, S., A. Fedorova, S. Zhuravlev and A. Kamali, "A case for NUMA-aware contention management on multicore systems", in "Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques", (2010).

[38] Bloom, B. H., "Space/time trade-offs in hash coding with allowable errors", Commun. ACM **13**, 7, 422–426 (1970).

[39] Bolosky, W. L., R. P. Fitzgerald and M. L. Scott, "Simple but effective techniques for numa memory management", in "Proceedings of the Twelfth ACM Symposium on Operating Systems Principles", (1989).

[40] Burtscher, M., R. Nasre and K. Pingali, "A quantitative study of irregular programs on GPUs", in "Proceedings of the IEEE International Symposium on Workload Characterization", (2012).

[41] Cabezas, J., L. Vilanova, I. Gelado, T. B. Jablin, N. Navarro and W.-m. W. Hwu, "Automatic parallelization of kernels in shared-memory multi-gpu nodes", in "Proceedings of the 29th ACM on International Conference on Supercomputing", (2015).

[42] Chandra, R., S. Devine, B. Verghese, A. Gupta and M. Rosenblum, "Scheduling and page migration for multiprocessor compute servers", in "Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems", (1994).

[43] Chatterjee, N., M. O'Connor, G. H. Loh, N. Jayasena and R. Balasubramonia, "Managing DRAM latency divergence in irregular GPGPU applications", in "Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis", (2014).

[44] Chaudhuri, M., "Pseudo-LIFO: the foundation of a new family of replacement policies for LLCs", in "Proceedings of the 42nd International Symposium on Microarchitecture", (2009).

[45] Chaudhuri, M., J. Gaur, N. Bashyam, S. Subramoney and J. Nuzman, "Introducing hierarchy-awareness in replacement and bypass algorithms for last-level caches", in "Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques", (2012).

[46] Che, S., B. M. Beckmann, S. K. Reinhardt and K. Skadron, "Pannotia: Understading irregular GPGPU graph applications", in "Proceedings of the International Symposium on Workload Characterization", (2013).

[47] Che, S., M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing", in "Proceedings of the International Symposium on Workload Characterization", (2009).

[48] Che, S., J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang and K. Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads", in "Proceedings of the International Symposium on Workload Characterization", (2010).

[49] Chen, L., O. Villa and G. R. Gao, "Exploring fine-grained task-based execution on multi-GPU systems", in "Proceedings of the IEEE International Conference on Cluster Computing", (2011).

[50] Chen, L., O. Villa, S. Krishnamoorthy and G. R. Gao, "Dynamic load balancing on single- and multi-GPU systems", in "Proceedings of the IEEE International Symposium on Parallel Distributed Processing", (2010).

[51] Chen, X., L.-W. Chang, C. I. Rodrigues, J. Lv, Z. Wang and W.-M. Hwu, "Adaptive cache management for energy-efficient GPU computing", in "Proceedings of the International Symposium on Microarchitecture", (2014).

[52] Chen, X., L. Yang, R. Dick, L. Shang and H. Lekatsas, "C-Pack: a high-performance microprocessor cache compression algorithm", IEEE Transactions on Very Large Scale Integration (VLSI) Systems **18**, 8, 1196–1208 (2010).

[53] Das, S., T. M. Aamodt and W. J. Dally, "Reuse distance-based probabilistic cache replacement", ACM Transactions on Architecture Code Optimization **12**, 4, 33:1–33:22 (2015).

[54] Dashti, M., A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema and M. Roth, "Traffic management: A holistic approach to memory placement on NUMA systems", in "Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems", (2013).

[55] DDR3, "Micron DDR3 SDRAM http://www.micron.com/products/dram/ddr3-sdram", (2007).

[56] Dreps, D., "The 3rd generation of ibm's elastic interface on power6", in "Proceedings of the IEEE Hot Chips 19 Symposium", HCS '19 (2007).

[57] Dusser, J., T. Piquet and A. Seznec, "Zero-content augmented caches", in "Proceedings of the 23rd international conference on Supercomputing", (2009).

[58] Dybdahl, H. and P. Stenstrom, "An adaptive shared/private NUCA cache partitioning scheme for chip multiprocessors", in "Proceedings of the IEEE 13th International Symposium onHigh Performance Computer Architecture", (2007).

[59] Ekman, M. and P. Stenstrom, "A robust main-memory compression scheme", in "Proceedings of the 32nd Annual International Symposium on Computer Architecture", (2005).

[60] Farooq, M. U., Khubaib and L. K. John, "Store-load-branch (SLB) predictor: A compiler assisted branch prediction for data dependent branches", in "Proceedings of the IEEE 19th International Symposium on High Performance Computer Architecture", (2013).

[61] Gebhart, M., D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm and K. Skadron, "Energy-efficient mechanisms for managing thread context in throughput processors", in "Proceedings of the 38th Annual International Symposium on Computer Architecture", (2011).

[62] Guerreiro, J., A. Ilic, N. Roma and P. Tomas, "GPGPU power modeling for multi-domain voltage-frequency scaling", in "Proceedings of the IEEE International Symposium on High Performance Computer Architecture", (2018).

[63] Hallnor, E. and S. Reinhardt, "A unified compressed memory hierarchy", in "Proceedings of the 11th International Symposium on High-Performance Computer Architecture", (2005).

[64] Hashemi, M. and Y. N. Patt, "Filtered runahead execution with a runahead buffer", in "Proceedings of the 48th International Symposium on Microarchitecture", (2015).

[65] He, B., W. Fang, N. K. Govindaraju, Q. Luo and T. Wang, "Mars: A mapreduce framework on graphics processors", in "Proceedings of the International Conference on Parallel Architectures and Compilation Techniques", (2008).

[66] Hong, S. and H. Kim, "An integrated GPU power and performance model", in "Proceedings of the 37th Annual International Symposium on Computer Architecture", (2010).

[67] Hu, Z., S. Kaxiras and M. Martonosi, "Timekeeping in the memory system: Predicting and optimizing memory behavior", in "Proceedings of the 29th Annual International Symposium on Computer Architecture", ISCA (2002).

[68] Huffman, D., "A method for the construction of minimum-redundancy codes", Proc. of the IRE (1952).

[69] Intel, "Intel Core i7 Processors http://www.intel.com/products/processor/corei7/", (2008).

[70] Ishida, M., "Kyocera APX - An Advanced Organic Technology for 2.5D Interposers", URL `https://www.ectc.net`, accessed: 2016-06-20 (2014).

[71] ITRS, "International technology roadmap for semiconductors 2.0", URL `http://www.itrs2.net/itrs-reports.html` (2015).

[72] Jaleel, A., E. Borch, M. Bhandaru, S. Steely and J. Emer, "Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (TLA) cache management policies", in "Proceedings of 43rd Annual IEEE/ACM International Symposium on Microarchitecture", (2010).

[73] Jaleel, A., W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely, Jr. and J. Emer, "Adaptive insertion policies for managing shared caches", in "Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques", (2008).

[74] Jaleel, A., K. B. Theobald, S. C. Steely, Jr. and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)", in "Proceedings of the 37th Annual International Symposium on Computer Architecture", (2010).

[75] Jia, W., K. A. Shaw and M. Martonosi, "Characterizing and improving the use of demand-fetched caches in GPUs", in "Proceedings of the 26th ACM International Conference on Supercomputing", (2012).

[76] Jia, W., K. A. Shaw and M. Martonosi, "MRPB: Memory request prioritization for massively parallel processors", in "Proceedings of the IEEE 20th International Symposium on High Performance Computer Architecture", (2014).

[77] Jimenez, D. A. and C. Lin, "Dynamic branch prediction with perceptrons", in "Proceedings of the Seventh International Symposium on High-Performance Computer Architecture", (2001).

[78] Kadjo, D., J. Kim, P. Sharma, R. Panda, P. Gratz and D. Jimenez, "B-Fetch: Branch prediction directed prefetching for chip-multiprocessors", in "Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture", (2014).

[79] Kannan, A., N. D. E. Jerger and G. H. Loh, "Enabling interposer-based disintegration of multi-core processors", in "Proceedings of the 48th International Symposium on Microarchitecture", (2015).

[80] Kayiran, O., A. Jog, A. Pattnaik, R. Ausavarungnirun, X. Tang, M. T. Kandemir, G. H. Loh, O. Mutlu and C. R. Das, "uC-States: Fine-grained GPU datapath power management", in "Proceedings of the International Conference on Parallel Architectures and Compilation", (2016).

[81] Keckler, S. W., W. J. Dally, B. Khailany, M. Garland and D. Glasco, "GPUs and the future of parallel computing", IEEE Micro **31**, 5, 7–17 (2011).

[82] Keramidas, G., P. Petoumenos and S. Kaxiras, "Cache replacement based on reuse-distance prediction", in "Proceedings of the 25th International Conference on Computer Design", (2007).

[83] Kestor, G., R. Gioiosa, D. J. Kerbyson and A. Hoisie, "Quantifying the energy cost of data movement in scientific applications", in "Proceedings of the IEEE International Symposium on Workload Characterization", (2013).

[84] Khairy, M., M. Zahran and A. G. Wassal, "Efficient utilization of GPGPU cache hierarchy", in "Proceedings of the Workshop on General Purpose Processing Using GPUs", (2015).

[85] Khan, S. M., D. A. Jiménez, D. Burger and B. Falsafi, "Using dead blocks as a virtual victim cache", in "Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques", (2010).

[86] Khan, S. M., Y. Tian and D. A. Jimenez, "Sampling dead block prediction for last-level caches", in "Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture", (2010).

[87] Kharbutli, M. and Y. Solihin, "Counter-based cache replacement and bypassing algorithms", in "IEEE Transactions on Computing", vol. 57 (2008).

[88] Kim, J., H. Kim, J. H. Lee and J. Lee, "Achieving a single compute device image in opencl for multiple GPUs", in "Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming", (2011).

[89] Kim, J. and Y. Kim, "HBM: Memory solution for bandwidth-hungry processors", in "Proceedings of the IEEE Hot Chips 26 Symposium", (2014).

[90] Kim, J., M. Sullivan, E. Choukse and M. Erez, "Bit-plane compression: Transforming data for better compression in many-core architectures", in "Proceedings of the 43rd International Symposium on Computer Architecture", (2016).

[91] Kumar, S., H. Zhao, A. Shriraman, E. Matthews, S. Dwarkadas and L. Shannon, "Amoeba-cache: Adaptive blocks for eliminating waste in the memory hierarchy", in "Proceedings of the International Symposium on Microarchitecture", (2012).

[92] Kumar, V., A. Grama, A. Gupta and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms* (Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994).

[93] Lai, A.-C., C. Fide and B. Falsafi, "Dead-block prediction & dead-block correlating prefetchers", in "proceedings of the International Symposium on Computer Architecture", (2001).

[94] Lal, S., J. Lucas and B. Juurlink, "E2MC: Entropy encoding based memory compression for GPUs", in "Proceedings of the IEEE International Parallel and Distributed Processing Symposium", (2017).

[95] LaRowe Jr., R. P., J. T. Wilkes and C. S. Ellis, "Exploiting operating system support for dynamic page placement on a NUMA shared memory multiprocessor", in "Proceedings of the Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming", (1991).

[96] Lee, J., M. Samadi, Y. Park and S. Mahlke, "Transparent CPU-GPU collaboration for data-parallel kernels on heterogeneous systems", in "Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques", (2013).

[97] Lee, J., V. Sathisha, M. Schulte, K. Compton and N. S. Kim, "Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling", in "Proceedings of the International Conference on Parallel Architectures and Compilation Techniques", (2011).

[98] Lee, M., S. Song, J. Moon, J. Kim, W. Seo, Y. Cho and S. Ryu, "Improving GPGPU resource utilization through alternative thread block scheduling", in "Proceedings of the IEEE 20th International Symposium on High Performance Computer Architecture", (2014).

[99] Lee, S., K. Kim, G. Koo, H. Jeon, W. W. Ro and M. Annavaram, "Warped-compression: Enabling power efficient GPUs through register compression", in "Proceedings of the 42nd Annual International Symposium on Computer Architecture", (2015).

[100] Lee, S.-Y., A. Arunkumar and C.-J. Wu, "CAWA: Coordinated warp scheduling and cache prioritization for critical warp acceleration of GPGPU workloads", in "Proceedings of the 42nd Annual International Symposium on Computer Architecture", (2015).

[101] Lee, S.-Y. and C.-J. Wu, "Ctrl-C: Instruction-aware control loop based adaptive cache bypassing for GPUss", in "Proceedings of the IEEE 34th International Conference on Computer Design", (2016).

[102] Leng, J., T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt and V. J. Reddi, "GPUWattch: Enabling energy optimizations in GPGPUs", in "Proceedings of the 40th Annual International Symposium on Computer Architecture", (2013).

[103] Li, A., G. J. van den Braak, A. Kumar and H. Corporaal, "Adaptive and transparent cache bypassing for GPUs", in "Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis", (2015).

[104] Li, C., S. L. Song, H. Dai, A. Sidelnik, S. K. S. Hari and H. Zhou, "Locality-driven dynamic GPU cache bypassing", in "Proceedings of the International Conference on Supercomputing", (2015).

[105] Li, D., M. Rhu, D. R. Johnson, M. O'Connor, M. Erez, D. Burger, D. S. Fussell and S. W. Keckler, "Priority-based cache allocation in throughput processors", in "Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture", (2015).

[106] Li, H., S. Tandri, M. Stumm and K. C. Sevcik, "Locality and loop scheduling on NUMA multiprocessors", in "Proceedings of the International Conference on Parallel Processing - Volume 02", (1993).

[107] Li, S., J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures", in "Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture", (2009).

[108] Liang, Y., Y. Wang and G. Sun, "Coordinated static and dynamic cache bypassing for GPUs", in "Proceedings of the Symposium on High Performance Computer Architecture", (2015).

[109] Lim, J., N. B. Lakshminarayana, H. Kim, W. J. Song, S. Yalamanchili and W. Sung, "Power modeling for GPU architectures using McPAT", ACM Transactions on Design Automation of Electronic Systems **19**, 3, 26:1–26:24 (2014).

[110] Liu, H., M. Ferdman, J. Huh and D. Burger, "Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency", in "Proceedings of the 41st International Symposium on Microarchitecture", (2008).

[111] Majo, Z. and T. R. Gross, "Matching memory access patterns and data placement for NUMA systems", in "Proceedings of the Tenth International Symposium on Code Generation and Optimization", (2012).

[112] Majumdar, A., L. Piga, I. Paul, J. L. Greathouse, W. Huang and D. H. Albonesi, "Dynamic GPGPU power management using adaptive model predictive control", in "Proceedings of the IEEE International Symposium on High Performance Computer Architecture", (2017).

[113] Manikantan, R., K. Rajan and R. Govindarajan, "NUcache: an efficient multicore cache organization based on next-use distance", in "proceedings of the International Symposium on High Performance Architecture", (2011).

[114] Mansuri, M., J. E. Jaussi, J. T. Kennedy, T.-C. Hsueh, S. Shekhar, G. Balamurugan, F. O'Mahony, C. Roberts, R. Mooney and B. Casper, "A scalable 0.128-to-1tb/s 0.8-to-2.6pj/b 64-lane parallel i/o in 32nm cmos", in "IEEE International Solid-State Circuits Conference Digest of Technical Papers", (2013).

[115] Mao, M., W. Wen, X. Liu, J. Hu, D. Wang, Y. Chen and H. Li, "Temp: Thread batch enabled memory partitioning for gpu", in "Proceedings of the 53rd Annual Design Automation Conference", (2016).

[116] McCalpin, J. D., "Memory bandwidth and machine balance in current high performance computers", IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter pp. 19–25 (1995).

[117] Meng, J. and K. Skadron, "Avoiding cache thrashing due to private data placement in last-level cache for manycore scaling", in "Proceedings of the IEEE International Conference on Computer Design.", (2009).

[118] Milic, U., O. Villa, E. Bolotin, A. Arunkumar, E. Ebrahimi, A. Jaleel, A. Ramirez and D. Nellans, "Beyond the socket: NUMA-aware GPUs", in "Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture", (2017).

[119] Mitsuishi, T., J. Suzuki, Y. Hayashi, M. Kan and H. Amano, "Breadth first search on cost-efficient multi-gpu systems", SIGARCH Comput. Archit. News **43**, 4, 58–63 (2016).

[120] Moore, G. E., "Cramming more components onto integrated circuits", Electronics **38**, 8 (1965).

[121] Mutlu, O., H. Kim and Y. N. Patt, "Techniques for efficient processing in runahead execution engines", in "Proceedings of the 32nd International Symposium on Computer Architecture", (2005).

[122] Nowatzki, T., J. Menon, C. han Ho and K. Sankaralingam, "gem5, GPGPUSim, McPAT, GPUWattch, "your favorite simulator here" considered harmful", (2014).

[123] NVIDIA, "NVIDIA GeForce GTX 480/470/465 GPU datasheet", URL http://www.nvidia.co.uk/docs/IO/90201/GTX-480-470-Web-Datasheet-Final4.pdf (2010).

[124] NVIDIA, "CUDA C/C++ SDK code samples v4.0", URL http://docs.nvidia.com/cuda/cuda-samples (2011).

[125] NVIDIA, "NVIDIA kepler GK110 architecture", URL https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf (2012).

[126] NVIDIA, "NVML api reference manual", URL http://developer.download.nvidia.com/assets/cuda/files/CUDADownloads/NVML/nvml.pdf (2012).

[127] NVIDIA, "NVIDIA GeForce GTX 750 Ti", URL http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce-GTX-750-Ti-Whitepaper.pdf (2014).

[128] NVIDIA, "NVIDIA tesla P100 architecture", URL https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdff (2016).

[129] NVIDIA, "CUDA C Programming Guide", URL http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (2017).

[130] NVIDIA, "NVIDIA tesla V100 architecture", URL http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf (2017).

[131] O'Connor, M., N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler and W. J. Dally, "Fine-grained DRAM: Energy-efficient DRAM for extreme bandwidth systems", in "Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture", (2017).

[132] O'Neil, M. A. and M. Burtscher, "Microarchitectural performance characterization of irregular GPU kernels", in "Proceedings of the IEEE International Symposium on Workload Characterization", (2014).

[133] O'Neil, M. A. and M. Burtscher, "Microarchitectural performance characterization of irregular gpu kernels", in "Proceedings of the IEEE International Symposium on Workload Characterization", (2014).

[134] Pandiyan, D. and C.-J. Wu, "Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms", in "Proceedings of the IEEE International Symposium on Workload Characterization", (2014).

[135] Paul, I., V. Ravi, S. Manne, M. Arora and S. Yalamanchili, "Coordinated energy management in heterogeneous processors", in "Proceedings of theInternational Conference for High Performance Computing, Networking, Storage and Analysis", (2013).

[136] Pekhimenko, G., E. Bolotin, N. Vijaykumar, O. Mutlu, T. C. Mowry and S. W. Keckler, "A case for toggle-aware compression for GPU systems", in "Proceedings of the IEEE International Symposium on High Performance Computer Architecture", (2016).

[137] Pekhimenko, G., T. Huberty, R. Cai, O. Mutlu, P. B. Gibbons, M. A. Kozuch and T. C. Mowry, "Exploiting compressed block size as an indicator of future reuse", in "Proceedings of the IEEE 19th International Symposium on High Performance Computer Architecture", (2015).

[138] Pekhimenko, G., V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches", in "Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques", (2012).

[139] Perelman, E., G. Hamerly, M. Van Biesbrouck, T. Sherwood and B. Calder, "Using simpoint for accurate and efficient simulation", in "Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems", (2003).

[140] Poulton, J., R. Palmer, A. M. Fuller, T. Greer, J. Eyles, W. J. Dally and M. Horowitz, "A 14-mw 6.25-gb/s transceiver in 90-nm cmos", IEEE Journal of Solid-State Circuits **42**, 12, 2745–2757 (2007).

[141] Poulton, J. W., W. J. Dally, X. Chen, J. G. Eyles, T. H. Greer, S. G. Tell, J. M. Wilson and C. T. Gray, "A 0.54 pJ/b 20 Gb/s ground-referenced single-ended short-reach serial link in 28 nm CMOS for advanced packaging applications", IEEE Journal of Solid-State Circuits **48**, 12, 3206–3218 (2013).

[142] Qureshi, M. K., A. Jaleel, Y. N. Patt, S. C. Steely and J. Emer, "Adaptive insertion policies for high performance caching", in "Proceedings of the 34th Annual International Symposium on Computer Architecture", (2007).

[143] Qureshi, M. K., D. N. Lynch, O. Mutlu and Y. N. Patt, "A case for MLP-aware cache replacement", in "Proceedings of the 33rd Annual International Symposium on Computer Architecture", (2006).

[144] Qureshi, M. K., D. Thompson and Y. N. Patt, "The V-Way Cache: Demand based associativity via global replacement", in "Proceedings of the 32nd Annual International Symposium on Computer Architecture", (2005).

[145] Rajan, K. and R. Govindarajan, "Emulating optimal replacement with a shepherd cache", in "Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture", (2007).

[146] Rhu, M., M. Sullivan, J. Leng and M. Erez, "A locality-aware memory hierarchy for energy-efficient GPU architectures", in "Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture", (2013).

[147] Rogers, T. G., M. O'Connor and T. M. Aamodt, "Cache-conscious wavefront scheduling", in "Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture", (2012).

[148] Santriaji, M. H. and H. Hoffmann, "GRAPE: Minimizing energy for GPU applications with performance requirements", in "Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture", (2016).

[149] Sardashti, S., A. Arelakis, P. Stenström and D. A. Wood, *A Primer on Compression in the Memory Hierarchy*, Synthesis Lectures on Computer Architecture (Morgan & Claypool Publishers, 2015).

[150] Sardashti, S., A. Seznec and D. A. Wood, "Skewed compressed caches", in "Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture", (2014).

[151] Sardashti, S. and D. A. Wood, "Decoupled compressed cache: Exploiting spatial locality for energy-optimized compressed caching", in "Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture", (2013).

[152] Sathish, V., M. J. Schulte and N. S. Kim, "Lossless and lossy memory I/O link compression for improving performance of GPGPU workloads", in "Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques", (2012).

[153] Seshadri, V., O. Mutlu, M. A. Kozuch and T. C. Mowry, "The evicted-address filter: A unified mechanism to address both cache pollution and thrashing", in "Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques", (2012).

[154] Sethia, A., G. Dasika, M. Samadi and S. Mahlke, "APOGEE: Adaptive prefetching on GPUs for energy efficiency", in "Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques", (2013).

[155] Sethia, A. and S. Mahlke, "Equalizer: Dynamic tuning of GPU resources for efficient execution", in "Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture", (2014).

[156] Shao, Y. S. and D. Brooks, "Energy characterization and instruction-level energy model of Intel's Xeon Phi processor", in "Proceedings of the International Symposium on Low Power Electronics and Design", (2013).

[157] Sharma, D. D., "PCI Express 3.0 Features and Requirements Gathering for beyond", URL `https://www.openfabrics.org/downloads/Media/Monterey_2011/Apr5_pcie%20gen3.pdf`, accessed: 2016-06-20 (2014).

[158] Smith, B. W. and K. Suzuki, *Microlithography: Science and Technology, Second Edition*, Optical science and engineering (2007), URL `https://books.google.com/books?id=_hTLDCeIYxoC`.

[159] Song, S., C. Su, B. Rountree and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent GPU architectures", in "Proceedings of the IEEE 27th International Symposium on Parallel and Distributed Processing", (2013).

[160] Srinath, S., O. Mutlu, H. Kim and Y. N. Patt, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers", in "Proceedings of the IEEE 13th International Symposium on High Performance Computer Architecture", (2007).

[161] Stuart, J. A. and J. D. Owens, "Message passing on data-parallel architectures", in "Proceedings of the IEEE International Symposium on Parallel & Distributed Processing", (2009).

[162] Stuart, J. A. and J. D. Owens, "Multi-GPU mapreduce on GPU clusters", in "Proceedings of the IEEE International Parallel Distributed Processing Symposium", (2011).

[163] Tam, D., R. Azimi and M. Stumm, "Thread clustering: Sharing-aware scheduling on smp-cmp-smt multiprocessors", in "Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems", (2007).

[164] Teran, E., Z. Wang and D. A. Jiménez, "Perceptron learning for reuse prediction", in "Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture", (2016).

[165] Tian, Y., S. Puthoor, J. L. Greathouse, B. M. Beckmann and D. A. Jiménez, "Adaptive GPU cache bypassing", in "Proceedings of the 8th Workshop on General Purpose Processing Using GPUs", (2015).

[166] Tyson, G., M. Farrens, J. Matthews and A. R. Pleszkun, "A modified approach to data cache management", in "proceedings of the International Symposium on Microarchitecture", (1995).

[167] Vijayaraghavan, T., Y. Eckert, G. H. Loh, M. J. Schulte, M. Ignatowski, B. M. Beckmann, W. C. Brantley, J. L. Greathouse, W. Huang, A. Karunanithi, O. Kayiran, M. Meswani, I. Paul, M. Poremba, S. Raasch, S. K. Reinhardt, G. Sadowski and V. Sridharan, "Design and analysis of an APU for exascale computing", in "Proceedings of the IEEE International Symposium on High Performance Computer Architecture", (2017).

[168] Vijaykumar, N., E. Ebrahimi, K. Hsieh, P. B. Gibbons and O. Mutlu, "The locality descriptor: A holistic cross-layer abstraction to express data locality in GPUs", in "Proceedings of the 45th Annual International Symposium on Computer Architecture", (2018).

[169] Vijaykumar, N., G. Pekhimenko, A. Jog, A. Bhowmick, R. Ausavarungnirun, C. Das, M. Kandemir, T. C. Mowry and O. Mutlu, "A case for core-assisted bottleneck acceleration in GPUs: Enabling flexible data compression with assist warps", in "Proceedings of the 42nd Annual International Symposium on Computer Architecture", (2015).

[170] Wang, J., N. Rubin, A. Sidelnik and S. Yalamanchili, "Laperm: Locality aware scheduler for dynamic parallelism on gpus", in "Proceedings of the 43rd International Symposium on Computer Architecture", (2016).

[171] Wilson, K. M. and B. B. Aglietti, "Dynamic page placement to improve locality in CC-NUMA multiprocessors for TPC-C", in "Proceedings of the ACM/IEEE Conference on Supercomputing", (2001).

[172] Wu, C.-J., A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely and J. Emer, "SHiP: Signature-based hit predictor for high performance caching", in "Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture", (2011).

[173] Wu, C.-J., A. Jaleel, M. Martonosi, S. C. Steely, Jr. and J. Emer, "PACMan: Prefetch-aware cache management for high performance caching", in "Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture", (2011).

[174] Wu, G., J. L. Greathouse, A. Lyashevsky, N. Jayasena and D. Chiou, "GPGPU performance and power estimation using machine learning", in "Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture", (2015).

[175] Xie, X., Y. Liang, G. Sun and D. Chen, "An efficient compiler framework for cache bypassing on GPUs", in "Proceedings of the IEEE/ACM International Conference on Computer-Aided Design", (2013).

[176] Xie, X., Y. Liang, Y. Wang, G. Sun and T. Wang, "Coordinated static and dynamic cache bypassing for GPUs", in "Proceedings of the IEEE 21st International Symposium on High Performance Computer Architecture", (2015).

[177] Xie, Y. and G. Loh, "PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches", in "Proceedings of the 37th International Symposium on Computer Architecture", (2009).

[178] Yeh, T.-Y., D. T. Marr and Y. N. Patt, "Increasing the instruction fetch rate via multiple branch prediction and a branch address cache", in "Proceedings of the ACM International Conference on Supercomputing 25th Anniversary Volume", (2014).