

MEMORY HIERARCHY PARTITIONING AND DATA MAPPING TECHNIQUES FOR SCRATCH PAD BASED ARCHITECTURES

PRASENJIT CHAKRABORTY



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY DELHI
JUNE 2015

©Indian Institute of Technology Delhi (IITD), New Delhi, 2015

MEMORY HIERARCHY PARTITIONING AND DATA MAPPING TECHNIQUES FOR SCRATCH PAD BASED ARCHITECTURES

by

PRASENJIT CHAKRABORTY

Department of Computer Science and Engineering

Submitted

in fulfillment of the requirements of the degree of
Doctor of Philosophy

to the



Indian Institute of Technology Delhi

June 2015

Certificate

This is to certify that the thesis titled **Memory Hierarchy Partitioning and Data Mapping Techniques for Scratch Pad Based Architectures** being submitted by **Prasenjit Chakraborty** for the award of **Doctor of Philosophy** in Computer Science & Engg. is a record of bona-fide work carried out by him under my guidance and supervision at the Department of Computer Science & Engineering, Indian Institute of Technology Delhi. The work presented in this thesis has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma.

Preeti Ranjan Panda

Professor

Department of Computer Science and Engineering

Indian Institute of Technology Delhi

New Delhi - 110016

Dedication

To my uncle, Late Dr. Monotosh Chakraborty.

Thank you for your love, support and encouraging me for heights!

Acknowledgments

There are several people I would like to thank, without them this thesis would not have been possible. First and foremost, I would like to thank my advisor, Prof. Preeti Ranjan Panda. My first interaction with him was during my application process for part-time PhD and he was instrumental in enabling me to get admitted to IIT Delhi and for giving me an opportunity to work under his supervision. His constant guidance, advice and technical mentorship were valuable to make this thesis a reality and I have learned a great deal from him. I thank Prof. Sandeep Sen for providing the theoretical formulation and proof of the work presented in this thesis. I would like to thank many other professors at IIT Delhi that I have interacted over the last 7 years.

I would like to express my deepest gratitude to IBM India for permitting me to pursue part-time PhD studies. My sincere thanks to Sriram Vajapeyam for instigating the thought to pursue PhD and embarking me into the world of research. Special thanks to Manish Gupta and Reena Malangone for instrumenting this into a reality within the realm of office work. I also thank Intel India for all the support.

Thanks to all my PhD research colleagues - Ayesha, Vaibhav, Gayathri, Namita, Krishnaiah, Sandeep, Sharat for their support during various stages over the course of my PhD studies. Thanks to Sahu, Anant, Lava and Kameshwar for all the discussions and company over the coffee breaks.

I thank my wife, Ananya, for the understanding over the past few arduous years while I was too busy managing PhD research and office work. I thank my daughter Pratyusha and son Adhvik for their love; they make life worth looking forward to. Finally, I thank my mother for her affection and unconditional love.

Prasenjit Chakraborty

Abstract

Modern computer systems lay greater emphasis on energy-efficiency as performance ceases to be the sole criteria for processor design. Therefore, architecting systems that improve performance yet remain within the power budget becomes necessary. The memory subsystem plays a dominant role in this endeavor of system redesign. Due to the growing gap between the CPU and DRAM performance/energy, innovating on-chip memory storage becomes paramount. Scratch-Pad memory (SPM) is considered a useful component in the memory hierarchy, solely or along with caches, to meet the power and energy constraints. Although the efficiency of SPM is well-known, its usage has been restricted owing to difficulties in programmability. Real applications usually have regions that are amenable to exploitation by either SPM or Cache, and hence can improve if the two are used in conjunction. Dynamically adjusting the on-chip local memory resources to suit application demand, can significantly improve the efficiency of the overall system. In this work, we propose a compiler technique to map application data objects to SPM/cache and also partition the local memory between SPM and cache depending on the dynamic requirement of the application.

First, we design a framework to characterize application data objects and partition the on-chip memory space between the objects. This helps alleviate the manual effort required of the developers in programming SPM based systems. This profile-based solution is obtained by a quick one-time execution of an instrumented binary instead of slow and detailed simulations of all possible local memory configurations. We apply detailed compiler based analysis for further refining the allocation and partition sizes. Second, we develop a novel cost based cache aware algorithm to decide the mapping of the application

objects between SPM and cache at different regions of the program. Our data mapping strategy is able to seamlessly integrate existing SPM mapping techniques, and is structured as an enhancement to assign the cache to data objects where it is considered to be more appropriate than SPM. We demonstrate its utility by executing applications on Cell BE hardware which resulted in performance improvement over previous SWC-based strategies.

Finally, in this research work we introduce a novel graph based structure to tackle data mapping decisions in a complex application. We use this to present a data mapping heuristic to map program objects for a fixed sized SPM-Cache hybrid system that targets whole program optimization. We extend this formulation to adapt the SPM and cache sizes, as well as the data mapping as per the requirement of different application regions. We study the applicability of the technique on various workloads targeted at both SPM-Only and hardware reconfigurable memory system, and obtain the performance and energy results using simulations and compare it against existing techniques. We demonstrate that our compiler based allocation scheme is applicable for complex programs targeted at all flavors of SPM based architectures and results in significant performance efficiency improvement over existing competitive techniques.

Contents

Certificate	i
Acknowledgments	v
Abstract	vii
List of Figures	xvii
List of Tables	xxi
1 Introduction	1
1.1 SPM Based Architectures	2
1.2 Thesis Statement	5
1.3 Research Contribution	6
1.4 Dissertation Organization	7
2 Background	9
2.1 Cache vs. Scratch Pad Memory	10
2.1.1 Design	10
2.1.2 On-Chip Memory Utilization	10
2.1.3 Latency Tolerance	11
2.1.4 Energy Consumption	11
2.1.5 Programmability	11
2.2 Implementation of SPM Based Architectures	12

2.2.1	Cacheless Local Memory	12
2.2.2	Hybrid Local Memory	14
2.2.3	Reconfigurable Local Memory	16
2.3	Power-Performance Evaluation of SPM Based Architectures	17
2.4	Baseline SPM Based Architecture	20
2.5	SPM Programming Model	22
3	Related Work	25
3.1	Static Scratchpad Allocation Techniques	26
3.2	Dynamic Scratchpad Allocation Techniques	27
3.3	Scratchpad Allocation in Presence of Caches	30
3.4	Reconfigurable Architectures	32
3.5	Tools for Assisting Programs Targeted to SPM	35
4	Framework for Assisting Data Partitioning of Application	37
4.1	Introduction	38
4.2	Overview of the Framework	40
4.3	Reuse Distance Based Analysis	43
4.4	Object Allocation Tracking	46
4.5	Object Categorization and Mapping Criteria	47
4.6	Category Size Estimation	50
4.7	Critical Object Identification	53
4.8	Analysis and Evaluation	54
4.8.1	Benchmark Characteristics	54
4.8.2	Profile Distribution and Object Categorization	57
4.8.3	Critical Object Identification	61
4.8.4	Evaluation of Partition Size	64
4.9	Summary	69

5	Data Mapping to SPM and Cache	71
5.1	Motivating Examples	72
5.1.1	Blocking and Double Buffering	72
5.1.2	Scatter-Gather Processing	74
5.1.3	Conditional Array Accesses	75
5.1.4	Temporal Locality	77
5.1.5	Mode Transitions in Array Accesses	79
5.2	Framework for Data Mapping	79
5.2.1	Deriving Regions in the Program	80
5.2.2	Array Access Graph	82
5.2.3	Node Cost Estimates	82
5.2.4	Edge Cost Estimates	83
5.2.5	Example AAG	88
5.2.6	Special Cases	89
5.3	Mapping Strategy	91
5.3.1	Prioritization of Arrays	92
5.3.2	Building Array Access Graph	92
5.3.3	Least Cost Path for an Array	96
5.3.4	Mapping All Arrays	99
5.4	Experiments	102
5.4.1	Setup	102
5.4.2	Benchmark Selection	103
5.4.3	Results	104
5.5	Summary	108
6	Whole Program Analysis for Partitioning and Mapping	111
6.1	Loop Transition Graph	112
6.2	Constructing the Loop Transition Graph	114
6.3	Fixed Partition Mapping	116

6.3.1	Cost Estimation of FCLTG	117
6.3.2	LTG Decomposition	119
6.3.3	Algorithm to Derive FCLTG	123
6.4	Dynamic Partitioning of L1 Memory	126
6.5	Minimum Cost Partition	128
7	Implementation and Results	135
7.1	Implementation Framework	135
7.2	Simulation Infrastructure	136
7.3	Benchmark Selection	138
7.4	Experimental Comparisons	140
7.5	Results	142
7.5.1	Effectiveness of <i>LTG-Fixed</i>	145
7.5.2	Effectiveness of <i>LTG-Reconfig</i>	147
7.5.3	Energy-Delay Product	148
7.6	Effect of Reconfiguration	149
7.7	Dynamic Variation of Memory Configuration	151
7.8	Summary	160
8	Conclusion and Future Work	163
8.1	Future Research	166
8.1.1	<i>SPM-Sieve</i> Framework Enhancement	166
8.1.2	Optimization for Performance/Energy Efficiency	166
8.1.3	Mapping for Multi Hierarchy On-chip Storage	167
8.1.4	Multi-processor/Parallel System	167
8.1.5	Multiple Concurrent Software Cache	167
8.1.6	Reconfiguration Enhancement	168
	References	169

List of Publications	185
Biography	187

List of Figures

1.1	Impact of Cache Size on Hit Rate of Spec 2000 suite	4
2.1	Cell Broadband Engine	13
2.2	Cacheless Memory Architecture	14
2.3	SPM-Cache hybrid memory system	15
2.4	Hardware Reconfigurable Memory Architecture	16
2.5	Example illustrating regular and irregular code sections in an application .	18
2.6	Comparison of Performance and Energy : SPM-Only, Cache-Only, Fixed SPM-Cache and Reconfigurable Architecture	20
2.7	Work proposed for the Target Architecture	22
2.8	Programming a DAXPY kernel in a SPM system	24
4.1	High-level workflow of SPM-Sieve framework	41
4.2	Analyzing Application using PIN	42
4.3	Reuse Distance Distribution and Miss Rate Estimation for 186.crafty . . .	45
4.4	Per-category Size Estimation Methodology (RD=Reuse Distance)	52
4.5	Distribution of memory footprint for SPEC 2000 benchmarks	58
4.6	Breakdown of memory profile	59
4.7	L1 D allocation estimated by SPM-Sieve	61

4.8	Impact of varying partition sizes against SPM-Sieve suggested baseline partition. For <i>188.amm</i> and <i>300.twolf</i> , D2x is simulated at D1.5x. For <i>179.art</i> D2x and D4x are simulated at D64x and D128x respectively. Simulation not performed due to infeasible configurations: <i>300.twolf</i> (D4x), <i>179.art</i> (Sx/4,Sx/2,Dx/4,Dx/2), <i>186.crafty</i> (S4x), <i>188.amm</i> (D4x)	66
4.9	Comparing allocation suggested by SPM-Sieve with the best configuration achieved using detailed simulation	68
5.1	Restructuring of Code to use Single and Double Buffer	73
5.2	Compute Communication Timeline	74
5.3	Gather DMA operation	75
5.4	Access under a conditional	77
5.5	Presence of Temporal Locality	78
5.6	Multiple access modes for same array	79
5.7	Execution flow with Regions and Timestamps	81
5.8	Array Access Graph	82
5.9	Example illustrating mapping of <i>ftab</i> array in <i>bzip2</i>	89
5.10	Result comparing our technique with others for Cell BE	105
6.1	A simple example introducing LTG	113
6.2	More examples to illustrate LTG	114
6.3	Various ways the LTG in Figure 6.2b can be decomposed into AAG	117
6.4	An example to illustrate the LTG decomposition process and derivation of FCLTG : colors represent mapping decisions	120
6.5	Example of shared node scenario and its LTG	121
6.6	LTG decomposition and handling of shared nodes	122
6.7	Multi-Configuration LTG for Configuration C1 - Cn	127
6.8	Example for Selection of Node Pair in MCLTG	132
6.9	Example for Selection of a Node in MCLTG	132

7.1	High Level Workflow of Analysis and Simulation Infrastructure	136
7.2	Performance Comparison of LTG Based Allocation Heuristics. In Cacheless architecture, a part of the SPM is treated as a Software Cache (SWC) . . .	143
7.3	Energy Comparison of LTG Based Allocation Heuristics	144
7.4	L1 Memory Access Distribution (Non Stack)	145
7.5	ExD Comparison of LTG Based Allocation Heuristics	149
7.6	Reconfiguration of L1 Memory by <i>LTG-Reconfig</i>	150
7.7	Minimum Cost Selection MCLTG : mcf	152
7.8	Minimum Cost Selection MCLTG : equake	153
7.9	Minimum Cost Selection MCLTG : bzip2	154
7.10	Minimum Cost Selection MCLTG : cjpeg	155
7.11	Minimum Cost Selection MCLTG : amg2013	156
7.12	Dynamic Reconfiguration during Execution of mcf : complete execution timeline	158
7.13	Dynamic Reconfiguration during Execution of equake : partial execution timeline (log scale)	158
7.14	Dynamic Reconfiguration during Execution of bzip2 : partial execution timeline	159
7.15	Dynamic Reconfiguration during Execution of cjpeg : partial execution timeline	159
7.16	Dynamic Reconfiguration during Execution of amg2013 : partial execution timeline	160

List of Tables

2.1	SPM and Cache Latencies Obtained using CACTI and used for the Comparison	19
4.1	Example illustrating Reuse Distance	44
4.2	Collecting Reuse Distances into buckets	45
4.3	Object Categorization and Mapping Decision	49
4.4	SPEC 2000 Object Categorization	50
4.5	Estimating object category size for 186.crafty with 128KB L1 memory . . .	53
4.6	SPEC 2000 Benchmark Characteristics	56
4.7	Impact of Cache Size on Hit Rate (%)	60
4.8	SPEC 2000 Critical Objects	63
4.9	Baseline Configuration: SPM-Sieve Suggested Partition Sizes in Bytes . . .	64
5.1	Edge Transition Costs for SPM - SPM Allocation	86
5.2	Edge Transition Costs for SPM - CACHE Allocation	86
5.3	Edge Transition Costs for CACHE - SPM Allocation	87
5.4	Delta Blocks Estimation for CACHE - CACHE Allocation	88
5.5	Latencies in cycles of Cell BE	103
5.6	Benchmark Characteristics	103
5.7	Number of DMA setups and Cache Performance (in Millions)	108
7.1	Architecture parameters	137
7.2	Benchmarks and inputs	138

7.3	Characteristics of Benchmarks	139
7.4	Number of Data Objects: Threshold size 1 KB	140
7.5	Memory Access Distribution (%)	140
7.6	Misses per Kilo Accesses (MPKA)	140
7.7	L1-Data Memory Partition Sizes (KB): SPM-Cache partition size adjusted for Fixed Size Mapping Techniques obtained by profiling	141