ARCHITECTURAL SUPPORT FOR A VARIABLE GRANULARITY CACHE MEMORY SYSTEM

by

Snehasish Kumar
B.Tech, Biju Patnaik University of Technology, 2010

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Fall 2012

APPROVAL

Name: Snehasish Kumar

Degree: Master of Science

Title of Thesis: Architectural support for a variable granularity cache memory system

Examining Committee: Dr. Hu Kaers
Chair

Dr. Arrvindh Shriraman,
Senior Supervisor

Dr. Alexandra Federova,
Supervisor

Date Approved: _____

Partial Copyright Licence

Abstract

Memory in modern computing systems are hierarchial in nature. Maintaining a memory hierarchy enables the system to service frequently requested data from a small low latency store located close to the processor. The design paradigms of the memory hierachy have been mostly unchanged since their inception in the late 1960's. However in the meantime there have been significant changes in the tasks computers perform and the way they are programmed. Modern computing systems perform more data centric tasks and are programmed in higher level languages which introduce many layers of abstraction between the programmer and the system.

Waste in the memory hierarchy refers to the under utilised space in the memory system and consequently wasted energy and time. The data access patterns of modern workloads are increasingly less uniform which makes it hard to design a memory hierarchy with rigid design principles that performs optimally for a wide range of workloads. The problem is exacerbated by the implications of the growing fraction of dark silicon on a processor chip.

This dissertation proposes and evaluates the benefits of a novel architecture for the on chip memory hierarchy which would allow it to dynamically adapt to the requirements of the application. We propose a design that can support a variable number of cache blocks, each of a different granularity. It employs a novel organization that completely eliminates the tag array, treating the storage array as uniform and morphable between tags and data. This enables the cache to harvest space from unused words in blocks for additional tag storage, thereby supporting a variable number of tags (and correspondingly, blocks). The design adjusts individual cache line granularities according to the spatial locality in the application. It adapts to the appropriate granularity both for different data objects in an

application as well as for different phases of access to the same data.

Compared to a fixed granularity cache, improves cache utilization to 90% - 99% for most applications, saves miss rate by up to 73% at the L1 level and up to 88% at the LLC level, and reduces miss bandwidth by up to 84% at the L1 and 92% at the LLC. Correspondingly reduces on-chip memory hierarchy energy by as much as 36% and improves performance by as much as 50%.

To whomever whoever reads this!

"Don't worry, Gromit. Everything's under control!"

— *The Wrong Trousers*, Aardman Animations, *1993*

Acknowledgments

Here go all the people you want to thank.

Contents

List of Tables

List of Figures

List of Programs

Preface

Here go all the interesting reasons why you decided to write this thesis.

Chapter 1

Introduction

Memory systems are an integral part of computer architecture whose overall design and organisation have remained unchanged since their inception. Early mainframe computers in the 1960's were known to use a hierarchial memory organisation. The memory technologies include semi-conductor, magnetic core, drum and disc. Caching would be used to fetch data and instructions into the fastest memory ahead of CPU accesses. Initally, accessing memory was only slightly slower than register access however as the difference grew, the need to mitigate the delay incurred for a memory access became extremely important. The rate at which computations were performed kept increasing however the rate at which data was fed to the processor from the memory system did not grow at the same rate. In order to alleviate the effects of slow memory, smaller faster memory was built close to the processor to cache frquently used data. The first documented use of a data cache was in the IBM System/360 Model 85[3]. Now multilevel memory hierarchies are used which are composed of fast static random access memory and slower dynamic random access memory before going to disk.

1.1 Cache Memory Systems

Long story about cache memory systems and how they are organised Diagrams of standard caches

1.2 Motivation for Change

In traditional caches, the cache block defines the fundamental unit of data movement and space allocation in caches. The blocks in the data array are uniformly sized to simplify the insertion/removal of blocks, simplify cache refill requests, and support low complexity tag organization. Unfortunately, conventional caches are inflexible (fixed block granularity and fixed # of blocks) and caching efficiency is poor for applications that lack high spatial locality. Cache blocks influence multiple system metrics including bandwidth, miss rate, and cache utilization. The block granularity plays a key role in exploiting spatial locality by effectively prefetching neighboring words all at once. However, the neighboring words could go unused due to the low lifespan of a cache block. The unused words occupy interconnect bandwidth and pollute the cache, which increases the # of misses. We evaluate the influence of a fixed granularity block below.

1.2.1 Cache Utilization

In the absence of spatial locality, multi-word cache blocks (typically 64 bytes on existing processors) tend to increase cache pollution and fill the cache with words unlikely to be used. To quantify this pollution, we segment the cache line into words (8 bytes) and track the words touched before the block is evicted. We define utilization as the average # of words touched in a cache block before it is evicted. We study a comprehensive collection of workloads from a variety of domains: 6 from PARSEC [1], 7 from SPEC2006, 2 from SPEC2000, 3 Java workloads from DaCapo [2], 3 commercial workloads (Apache, SpecJBB2005, and TPC-C [4]), and the Firefox web browser. Subsets within benchmark suites were chosen based on demonstrated miss rates on the fixed granularity cache (i.e., whose working sets did not fit in the cache size evaluated) and with a spread and diversity in cache utilization. We classify the benchmarks into 3 groups based on the utilization they exhibit: Low (<33%), Moderate (33%—66%), and High (66%+) utilization (see Table 1.1).

Figure 1.1 shows the histogram of words touched at the time of eviction in a cache line of a 64K, 4-way cache (64-byte block, 8 words per block) across the different benchmarks. Seven applications have less than 33% utilization and 12 of them are dominated (¿50%) by 1-2 word accesses. In applications with good spatial locality (cactus, ferret, tradesoap, milc, eclipse) more than 50% of the evicted blocks have 7-8 words touched. Despite similar average utilization for applications such as astar and h2 (39%), their distributions are dissimilar;

Table 1.1: Benchmark Groups

Group	Utilization %	Benchmarks
Low	0 — 33%	art, soplex, twolf, mcf, canneal, lbm, omnetpp
Moderate	34 — 66%	astar, h2, jbb, apache, x264, firefox, tpc-c, freqmine, fluidanimate
High	67 — 100%	tradesoap, facesim, eclipse, cactus, milc, ferret

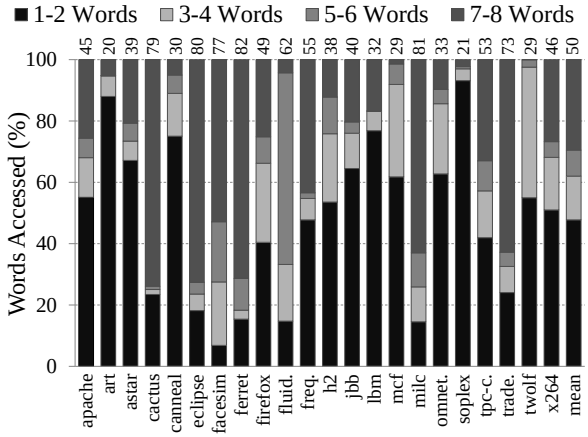


Figure 1.1: Distribution of words touched in a cache block. Avg. utilization is on top. (Config: 64K, 4 way, 64-byte block.)

≃70% of the blocks in astar have 1-2 words accessed at the time of eviction, whereas ≃50% of the blocks in h2 have 1-2 words accessed per block. Utilization for a single application also changes over time; for example, ferret's average utilization, measured as the average fraction of words used in evicted cache lines over 50 million instruction windows, varies from 50% to 95% with a periodicity of roughly 400 million instructions.

1.2.2 Effect of Block Granularity on Miss Rate and Bandwidth

Cache miss rate directly correlates with performance, while under current and future wire-limited technologies, bandwidth directly correlates with dynamic energy. Figure 1.2 shows the influence of block granularity on miss rate and bandwidth for a 64K L1 cache and a 1M L2 cache keeping the number of ways constant. For the 64K L1, the plots highlight the pitfalls of simply decreasing the block size to accommodate the Low group of applications;

miss rate increases by 2× for the High group when the block size is changed from 64B to 32B; it increases by 30% for the Moderate group. A smaller block size decreases bandwidth proportionately but increases miss rate. With a 1M L2 cache, the lifetime of the cache lines increases significantly, improving overall utilization. Increasing the block size from 64→256 halves the miss rate for all application groups. The bandwidth is increased by 2× for the Low and Moderate.

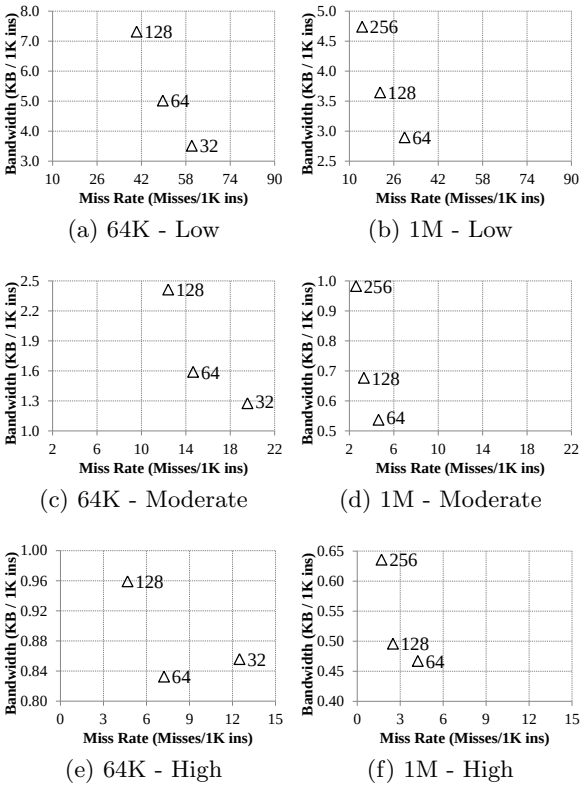
Since miss rate and bandwidth have different optimal block granularities, we use the following metric: $\frac{1}{MissRate \times Bandwidth}$ to determine a fixed block granularity suited to an application that takes both criteria into account. Table 1.2 shows the block size that maximizes the metric for each application. It can be seen that different applications have different block granularity requirements. For example, the metric is maximized for apache at 128 bytes and for firefox (similar utilization) at 32 bytes. Furthermore, the optimal block sizes vary with the cache size as the cache lifespan changes. This highlights the challenge of picking a single block size at design time especially when the working set does not fit in the cache.

1.2.3 Need for adaptive cache blocks

Our observations motivate the need for adaptive cache line granularities that match the spatial locality of the data access patterns in an application. In summary:

- Smaller cache lines improve utilization but tend to increase miss rate and potentially traffic for applications with good spatial locality, affecting the overall performance.
- Large cache lines pollute the cache space and interconnect with unused words for applications with poor spatial locality, significantly decreasing the caching efficiency.
- Many applications waste a significant fraction of the cache space. Spatial locality varies not only across applications but also within each application, for different data structures as well as different phases of access over time.

1.3 Dissertation Outline



(a) 64K - Low (b) 1M - Low

(c) 64K - Moderate (d) 1M - Moderate

(e) 64K - High (f) 1M - High

Figure 1.2: Bandwidth vs. Miss Rate. (a),(c),(e): 64K, 4-way L1. (b),(d),(f): 1M, 8-way LLC. Markers on the plot indicate cache block size. Note the different scales for different groups.

Table 1.2: Optimal block size. Metric: $\frac{\mathbf{1}}{\mathbf{Miss-rate} \times \mathbf{Bandwidth}}$

64K, 4-way	
Block	Benchmarks
32B	cactus, eclipse, facesim, ferret, firefox, fluidanimate,freqmine, milc, tpc-c, tradesoap
64B	art
128B	apache, astar, canneal, h2, jbb, lbm, mcf, omnetpp, soplex, twolf, x264
1M, 8-way	
Block	Benchmarks
64B	apache, astar, cactus, eclipse, facesim, ferret, firefox, freqmine, h2, lbm, milc, omnetpp, tradesoap, x264
128B	art
256B	canneal, fluidanimate, jbb, mcf, soplex, tpc-c, twolf

Chapter 2

Background

2.1 Cache Memory Architecture

2.1.1 Fundamental Building Blocks

2.1.2 Standard Cache Operations

2.1.3 Replacement Policies

2.1.4 Hierarchial Cache Memory Systems

2.1.5 On-chip Interconnection Network

2.1.6 Multi Core and Coherence

2.2 Related Work

2.2.1 Sector Caches

2.2.2 Cache Filtering

2.2.3 Spatial Access Pattern Prediction

2.2.4 Software based approaches

Chapter 3

Amoeba Cache Architecture

3.1 Fundamental Building Blocks

3.2 Cache Management

3.2.1 Standard Operations

3.2.2 Hardware Aids

3.2.3 Replacement Policy

3.3 Spatial Pattern Predictor

3.3.1 Oracle

3.3.2 Region based Approach

3.3.3 Program Counter based Approach

3.4 Hierarchial Amoeba Cache Architecture

3.5 On-chip Interconnection Network

3.6 Multi Core and Coherence

Chapter 4

Implementation

4.1 Application Traces

4.1.1 Intel Pin

4.1.2 Generating a memory access trace

4.1.3 Workload selection

4.2 GEMS Infrastructure

4.2.1 Introduction

4.2.2 Components

4.2.3 SLICC

4.2.4 Amoeba-Single Protocol

Chapter 5

Evaluation

5.1 Best Effort - Oracle

5.1.1 Miss Rate - Performance

5.1.2 Bandwidth - Energy

5.2 Amoeba Cache vs Other Approaches

5.2.1 Sector Caches

5.2.2 Sector Caches with Prefetching

5.2.3 Line Distillation

5.2.4 Multi Cache

5.3 A feasible online approach

5.4 Multi Core Shared Cache

Chapter 6

Conclusion

6.1 Summary

6.2 Future Work

Bibliography

[1] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM.

[2] Stephen M. Blackburn, Robin Garner, Chris Hoffmann, Asjad M. Khang, Kathryn S. McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. The dacapo benchmarks: java benchmarking development and analysis. In *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, OOPSLA '06, pages 169–190, New York, NY, USA, 2006. ACM.

[3] J. S. Liptay. Structural aspects of the system/360 model 85: Ii the cache. *IBM Syst. J.*, 7(1):15–21, March 1968.

[4] Diego R. Llanos. Tpcc-uva: an open-source tpc-c implementation for global performance measurement of computer systems. *SIGMOD Rec.*, 35(4):6–15, December 2006.