
HIGH PERFORMANCE INSTRUCTION FETCH USING SOFTWARE AND HARDWARE CO-DESIGN

Alex Ramirez

Department of Computer Architecture
Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain. April 2002

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

©2002 - Alex Ramirez

All rights reserved.

Thesis advisors
Josep L. Larriba-Pey and Mateo Valero

Author
Alex Ramirez

High performance instruction fetch using software and hardware co-design

Abstract

The design of higher performance processors has been following two major trends: increasing the pipeline depth to allow faster clock rates, and widening the pipeline to allow parallel execution of more instructions. Designing a higher-performance processor implies balancing all the pipeline stages to ensure that overall performance is not dominated by any of them. This means that a faster execution engine also requires a faster fetch engine, to ensure that it is possible to read and decode enough instructions to keep the pipeline full, and the functional units busy.

This thesis explores the challenges faced by the design of the instruction fetch engine from a dual perspective: design a better software for the existing fetch architectures, and design a better hardware for the newly constructed software.

Our approach to the design of a better software has been the proposal of a novel code layout algorithm which targets both the instruction cache performance and the effective fetch width. Based on the analysis of the behavior of these optimized codes, we also propose a modification to the trace cache mechanism to make a more efficient use of the available storage space, and a novel branch predictor which exploits the available profile information to obtain higher prediction accuracy.

Finally, we propose a novel fetch architecture designed to exploit the special characteristics of optimized codes. The proposed fetch engine has low cost and complexity, but provides very high fetch performance.

*To my mother, who wanted me to get the best education.
I hope you can see this from where you are.*

Acknowledgments

To Josep L. Larriba-Pey who took me into this PhD adventure without giving me a chance to think about other options, for which I thank him.

To Mateo Valero who provided energy and tons of technical discussions on endless subjects which could not be fully developed to be included here.

To Luiz Barroso, Kourosh Gharachorloo, Robert Cohn, Geoff Lawney, and the whole Western Research Lab team, for hosting me for two consecutive summers, for the chance to work with them, and for plenty of cakes.

To John Shen, Hong Wang, Ed Grochowsky, and the MRL teams in Santa Clara, Austin, and Oregon, for bringing me in for an excellent summer which provided me a broader view of this field, for their technical expertise, and for their friendship.

To Jesus Corbal, Dani Jimenez, Carlos Navarro, Daniel Ortega, Xavi Serrano, Josep Torrellas, and all my fellow PBCs, who have all contributed significantly to this thesis. I could not have gone this far without them.

To my sister Marta, and my girlfriend Alicia, who have endured this long path with me, providing support when it was most needed.

And to my advisors again, for this opportunity to work with such fine people.

Contents

Abstract	3
Acknowledgments	7
1 Introduction	13
1.1 Motivation	13
1.1.1 Superscalar processor architecture	13
1.1.2 Objectives	16
1.2 Thesis overview	16
1.2.1 Compiler optimizations for improved fetch performance	16
1.2.2 Hardware modifications to exploit software characteristics	17
1.2.3 Exploiting layout optimized codes	17
1.3 Document structure	18
2 State of the Art	21
2.1 Code layout optimizations	21
2.1.1 Basic block chaining	22
2.1.2 Procedure splitting	23
2.1.3 Procedure mapping	24
2.2 Processor architecture	26
2.2.1 Pipelined processors	26
2.2.2 Superscalar processors	29
2.2.3 Wide superscalar processors	37
2.3 Conclusions	44
2.4 Historical context	44
3 Platform, Tools, and Benchmarks	47
3.1 Benchmarks	47
3.1.1 SPEC int 95	48
3.1.2 PostgreSQL and TPC-D	49
3.1.3 Oracle and TPC-B	50
3.2 Optimized code generation	51
3.2.1 Profiling tools	51
3.2.2 Code optimizers	52
3.3 Simulators	53
3.3.1 Fetch engine simulation	53
3.3.2 The Simplescalar toolset	54
3.3.3 Branch predictor simulation	55

3.3.4	SimOS	55
3.3.5	Real machine runs	56
3.3.6	Ideal pipeline simulator	57
3.4	Final remarks	58
4	Software Trace Cache	59
4.1	Placement algorithm	59
4.1.1	Seed selection	60
4.1.2	Trace construction	60
4.1.3	Trace mapping	62
4.2	Performance impact	63
4.2.1	Impact on the instruction cache	64
4.2.2	Impact on the fetch width	69
4.2.3	Impact on the branch predictor	72
4.2.4	Overall performance impact	79
4.3	Conclusions	83
5	Selective Trace Storage	85
5.1	Introduction	85
5.2	Trace cache redundancy	86
5.3	Selective Trace Storage	88
5.4	Evaluation	88
5.4.1	Realistic branch prediction	88
5.4.2	Perfect branch prediction	92
5.5	Conclusions	94
6	The <i>agbias</i> Branch Predictor	97
6.1	Introduction	97
6.2	Using profile data in dynamic prediction	98
6.2.1	The gshare predictor	98
6.2.2	The agree predictor	99
6.2.3	The bimode predictor	100
6.2.4	The gskew predictor	101
6.2.5	Combining dynamic and static predictors	102
6.2.6	The <i>agbias</i> predictor	104
6.3	Performance evaluation	106
6.3.1	Prediction table interference	106
6.3.2	BHR filtering	107
6.4	Conclusions	108
7	Fetching Instruction Streams	109
7.1	Introduction	109
7.2	Fetching instruction streams	110
7.3	Performance evaluation	116
7.4	Conclusions	122

8 Conclusions	123
Bibliography	127
List of Figures	135
List of Tables	139

