

Compilation Techniques for Reconfigurable Architectures

João M. P. Cardoso • Pedro C. Diniz

Compilation Techniques for Reconfigurable Architectures

 Springer

João M. P. Cardoso
Technical University of Lisbon/IST
INESC-ID, Rua Alves Redol n.9
1000-029 Lisboa, Portugal
jmpc@acm.org

Pedro C. Diniz
Technical University of Lisbon/IST
INESC-ID, Rua Alves Redol n.9
1000-029 Lisboa, Portugal
pedro@isi.edu

ISBN: 978-0-387-09670-4 e-ISBN: 978-0-387-09671-1
DOI: 10.1007/978-0-387-09671-1

Library of Congress Control Number: 2008929498

© 2009 Springer Science+Business Media, LLC

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

springer.com

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark. All trademarks mentioned in this book are the property of their respective owners.

ARM[®] is a registered trademark of ARM, Ltd.

Cantata[®] is a registered trademark of Khoral, Inc.

CoCentric[™] is a trademark of Synopsys, Inc.

Excalibur[™] is a trademark of Altera, Corp.

Java[™] is a registered trademark of Sun Microsystems, Inc.

MATLAB[®] is a registered trademark of MathWorks, Inc.

MicroBlaze[®] is a registered trademark of Xilinx, Inc.

MIPS[®] is a registered trademark of MIPS Technologies, Inc.

Nios[®] and Nios-II[®] are registered trademarks of Altera, Corp.

PowerPC[®] is a registered trademark of the IBM, Corp.

Stratix[™] is a trademark of Altera, Corp.

Verilog[®] is a registered trademark of Cadence Design Systems, Inc.

Virtex[™], Virtex-II[™], Virtex-II Pro[™] are trademarks of Xilinx, Inc.

Virtex-4[™] and Virtex-5[™] are trademarks of Xilinx, Inc.

WildStar[™] is a trademark of Annapolis Micro Systems Inc.

XPP[®] is a registered trademark of the PACT XPP Technologies, AG.

XTensa[®] is a registered trademark of Tensilica, Inc.

CLAY[™] is a trademark of National Semiconductors, Corp.

occam[®] is a registered trademarks of INMOS Limited.

To Teresa, Rodrigo, and Frederico
(João M. P. Cardoso)

To Mariana de Sena and Rafael Nuno
(Pedro C. Diniz)

Preface

The extreme flexibility of reconfigurable architectures and their performance potential have made them a vehicle of choice in a wide range of computing domains, from rapid circuit prototyping to high-performance computing. The increasing availability of transistors on a die has allowed the emergence of reconfigurable architectures with a large number of computing resources and interconnection topologies. To exploit the potential of these reconfigurable architectures, programmers are forced to map their applications, typically written in high-level imperative programming languages, such as C or MATLAB, to hardware-oriented languages such as VHDL or Verilog. In this process, they must assume the role of hardware designers and software programmers and navigate a maze of program transformations, mapping, and synthesis steps to produce efficient reconfigurable computing implementations. The richness and sophistication of any of these application mapping steps make the mapping of computations to these architectures an increasingly daunting process. It is thus widely believed that automatic compilation from high-level programming languages is the key to the success of reconfigurable computing.

This book describes a wide range of code transformations and mapping techniques for programs described in high-level programming languages, most notably imperative languages, to reconfigurable architectures. While many of these transformations and mapping techniques have been developed in the context of compilation for traditional architectures and high-level synthesis, their application to reconfigurable architectures poses a whole new set of challenges, in particular when targeting fine-grained reconfigurable architectures such as contemporary Field-Programmable Gate-Arrays (FPGAs). Their ability to emulate virtually any execution paradigm and to configure their logic blocks as either storage or computing resources forces compilers to evaluate a huge number of possible mapping alternatives in search for effective hardware implementations on these reconfigurable architectures.

This book is primarily intended for researchers and graduate students in the areas of study of hardware compilation and advanced computing architectures in the fields of Electrical and Computer Engineering and Computer Science. As it focuses on the specific topic of compilation from high-level program descriptions to reconfigurable

architectures, this book can easily support advanced compiler and computer architecture courses related to reconfigurable computing. Through this book, we hope to motivate further discussions on the challenging topic of compilation for reconfigurable architectures, and also hope this book can be a reference for researchers, educators, and students to learn more about the most prominent efforts on this subject in recent years.

This book was only made possible with the unabated comprehension, relentless, and unconditional support of our families who allowed us to devote to it countless many hours, some of them late at night. It is to them we dedicate this book, in particular to Teresa, Rodrigo, Frederico, Mariana, and Rafael. We are also truly indebted to our parents, Cristina and Luís and Mariana and Mário, for their lifelong support and encouragement. Lastly, we would also like to thank our friends for their support and friendship.

We would like to take this opportunity to acknowledge the support of the Springer staff, Amy Brais (Springer Publishing Editor) for the opportunity to publish this book, and Deborah Doherty (Springer Author Support) for her prompt help and guidance throughout the editing and publishing process.

We would also like to acknowledge the financial support of the Portuguese Foundation for Science and Technology (“Fundação para a Ciência e Tecnologia, FCT”) under the research grants PTDC/EEA-ELC/71556/2006, PTDC/EEA-ELC/70272/2006, and PTDC/EIA/70271/2006.

We would like to acknowledge and thank the many contributions to a survey about compilation techniques for reconfigurable computing platforms that inspired this book by Markus Weinhardt (PACT XPP Technologies, AG., Germany). We are also grateful to a number of colleagues who have carefully reviewed early drafts of the chapters in this book, in particular, Leonel Sousa (INESC-ID/IST/UTL, Lisbon, Portugal), José Alves (FEUP, Porto, Portugal), Koen Bertels (TU Delft, Delft, The Netherlands), Horácio Neto (INESC-ID/IST/UTL, Lisbon, Portugal), Mihai Budiu (Microsoft Research SVC, Mountain View, CA, USA), Timothy Callahan (CMU, PA, USA), Michael Hübner (Univ. of Karlsruhe, Karlsruhe, Germany), Benjamin Gerdemann (INESC-ID/IST/UTL, Lisbon, Portugal), Ricardo Ferreira (Federal Univ. of Viçosa, Brazil) and Jecel Assumpção, Jr. (Univ. of São Paulo, São Carlos, Brazil). Lastly, a very special thanks to Eduardo Marques (Univ. of São Paulo, São Carlos, Brazil) for all his support while completing the final stages of this book during our visit to the University of São Paulo in São Carlos, Brazil. Lastly, we are grateful to Markus Weinhardt for his contributions to a survey about compilation techniques for reconfigurable computing platforms that inspired this book.

Lisbon, Portugal,
April 2008

João M. P. Cardoso
Pedro C. Diniz

Contents

- 1 Introduction** 1
 - 1.1 The Promise of Reconfigurable Architectures and Systems 1
 - 1.2 The Challenge: How to Program and Compile for Reconfigurable Systems? 3
 - 1.3 This Book: Key Techniques when Compiling to Reconfigurable Architecture 4
 - 1.4 Organization of this Book 5

- 2 Overview of Reconfigurable Architectures** 7
 - 2.1 Evolution of Reconfigurable Architectures 7
 - 2.2 Reconfigurable Architectures: Key Characteristics 8
 - 2.3 Granularity 10
 - 2.3.1 Fine-Grained Reconfigurable Architectures 12
 - 2.3.2 Coarse-Grained Reconfigurable Architectures 14
 - 2.3.3 Hybrid Reconfigurable Architectures 16
 - 2.3.4 Granularity and Mapping 19
 - 2.4 Interconnection Topologies 20
 - 2.5 System-Level Integration 21
 - 2.6 Dynamic Reconfiguration 24
 - 2.7 Computational and Execution Models 29
 - 2.8 Streaming Data Input and Output 31
 - 2.9 Summary 31

- 3 Compilation and Synthesis Flows** 33
 - 3.1 Overview 33
 - 3.1.1 Front-End 34
 - 3.1.2 Middle-End 35
 - 3.1.3 Back-End 37
 - 3.2 Hardware Compilation and High-Level Synthesis 39
 - 3.2.1 Generic High-Level Synthesis 40
 - 3.2.2 Customized High-Level Synthesis for Fine-Grained Reconfigurable Architectures 41

3.2.3	Register-Transfer-Level/Logic Synthesis	45
3.2.4	High-Level Compilation for Coarse-Grained Reconfigurable Architectures	48
3.2.5	Placement and Routing	49
3.3	Illustrative Example	51
3.3.1	High-Level Source Code Example	51
3.3.2	Data-Flow Representation	52
3.3.3	Computation-Oriented Mapping and Scheduling	53
3.3.4	Data-Oriented Mapping and Transformations	55
3.3.5	Translation to Hardware	58
3.4	Reconfigurable Computing Issues and Their Impact on Compilation	59
3.4.1	Programming Languages and Execution Models	61
3.4.2	Intermediate Representations	62
3.4.3	Target Reconfigurable Architecture Features	64
3.5	Summary	65
4	Code Transformations	67
4.1	Bit-Level Transformations	67
4.1.1	Bit-Width Narrowing	68
4.1.2	Bit-Level Optimizations	72
4.1.3	Conversion from Floating- to Fixed-Point Representations	75
4.1.4	Nonstandard Floating-Point Formats	77
4.2	Instruction-Level Transformations	77
4.2.1	Operator Strength Reduction	78
4.2.2	Height Reduction	80
4.2.3	Code Motion	84
4.3	Loop-Level Transformations	87
4.3.1	Loop Unrolling	87
4.3.2	Loop Tiling and Loop Strip-Mining	90
4.3.3	Loop Merging and Loop Distribution	94
4.4	Data-Oriented Transformations	95
4.4.1	Data Distribution	95
4.4.2	Data Replication	96
4.4.3	Data Reuse and Scalar Replacement in Registers and Internal RAMs	96
4.4.4	Other Data-Oriented Transformations	99
4.5	Function-Oriented Transformations	101
4.5.1	Function Inlining and Outlining	101
4.5.2	Recursive Functions	104
4.6	Which Code Transformations to Choose?	105
4.7	Summary	107

5	Mapping and Execution Optimizations	109
5.1	Hardware Execution Techniques	109
5.1.1	Instruction-Level Parallelism	110
5.1.2	Speculative Execution	112
5.1.3	Predication and if-conversion	114
5.1.4	Multi Tasking	116
5.2	Partitioning	118
5.2.1	Temporal Partitioning	119
5.2.2	Spatial Partitioning	124
5.2.3	Illustrative Example	125
5.3	Mapping Program Constructs to Resources	127
5.3.1	Mapping Scalar Variables to Registers	127
5.3.2	Mapping of Operations to FUs	129
5.3.3	Mapping of Selection Structures	130
5.3.4	Sharing Functional Units FUs	131
5.3.5	Combining Instructions for RFUs	132
5.4	Pipelining	134
5.4.1	Pipelined Functional and Execution Units	135
5.4.2	Pipelining Memory Accesses	138
5.4.3	Loop Pipelining	139
5.4.4	Coarse-Grained Pipelining	144
5.4.5	Pipelining Configuration–Computation Sequences	145
5.5	Memory Accesses	146
5.5.1	Partitioning and Mapping of Arrays to Memory Resources	146
5.5.2	Improving Memory Accesses	148
5.6	Back-End Support	150
5.6.1	Allocation, Scheduling, and Binding	150
5.6.2	Module Generation	151
5.6.3	Mapping, Placement, and Routing	153
5.7	Summary	153
6	Compilers for Reconfigurable Architectures	155
6.1	Early Compilation Efforts	155
6.2	Compilers for FPGA-Based Systems	157
6.2.1	The SPC Compiler	157
6.2.2	A C to Fine-Grained Pipelining Compiler	158
6.2.3	The DeepC Silicon Compiler	158
6.2.4	The COBRA-ABS Tool	158
6.2.5	The DEFACTO Compiler	159
6.2.6	The Streams-C Compiler	159
6.2.7	The Cameron Compiler	160
6.2.8	The MATCH Compiler	160
6.2.9	The Galadriel and Nanya Compilers	161
6.2.10	The Sea Cucumber Compiler	161

6.2.11	The Abstract-Machines Compiler	161
6.2.12	The CHAMPION Software Design Environment	162
6.2.13	The SPARCS Tool	163
6.2.14	The ROCCC Compiler	163
6.2.15	The DWARV Compiler	163
6.3	Compilers for Coarse-Grained Reconfigurable Architectures	164
6.3.1	The DIL Compiler	164
6.3.2	The RaPiD-C Compiler	165
6.3.3	The CoDe-X Compiler	165
6.3.4	The XPP-VC Compiler	166
6.3.5	The DRESC Compiler	166
6.4	Compilers for Hybrid Reconfigurable Architectures	167
6.4.1	The Chimaera-C Compiler	167
6.4.2	The Garp and the Nimble C Compilers	168
6.4.3	The NAPA-C Compiler	168
6.5	Compilation Efforts Summary	169
7	Perspectives on Programming Reconfigurable Computing Platforms	177
7.1	How to Make Reconfigurable Computing a Reality?	177
7.1.1	Easy of Programming	178
7.1.2	Program Portability and Legacy Code Migration	179
7.1.3	Performance Portability	180
7.2	Research Directions in Compilation for Reconfigurable Architectures	181
7.2.1	Programming Language Design	181
7.2.2	Intermediate Representation	181
7.2.3	Mapping to Multiple Computing Engines	182
7.2.4	Code Transformations	182
7.2.5	Design-Space Exploration and Compilation Time	183
7.2.6	Pipelined Execution	184
7.2.7	Memory Mapping Optimizations	185
7.2.8	Application-Specific Compilers and Cores	185
7.2.9	Resource Virtualization	186
7.2.10	Dynamic and Incremental Compilation	186
7.3	Tackling the Compilation Challenge for Reconfigurable Architectures	187
7.4	Reconfigurable Architectures and Nanotechnology	189
7.5	Summary	189
8	Final Remarks	191
	References	193
	List of Acronyms	213
	Index	217