

Compiling Algorithms for Heterogeneous Systems

Synthesis Lectures on Computer Architecture

Editor

Margaret Martonosi, *Princeton University*

Founding Editor Emeritus

Mark D. Hill, *University of Wisconsin, Madison*

Synthesis Lectures on Computer Architecture publishes 50- to 100-page publications on topics pertaining to the science and art of designing, analyzing, selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals. The scope will largely follow the purview of premier computer architecture conferences, such as ISCA, HPCA, MICRO, and ASPLOS.

Compiling Algorithms for Heterogeneous Systems

Steven Bell, Jing Pu, James Hegarty, and Mark Horowitz
2018

Architectural and Operating System Support for Virtual Memory

Abhishek Bhattacharjee and Daniel Lustig
2017

Deep Learning for Computer Architects

Brandon Reagen, Robert Adolf, Paul Whatmough, Gu-Yeon Wei, and David Brooks
2017

On-Chip Networks, Second Edition

Natalie Enright Jerger, Tushar Krishna, and Li-Shiuan Peh
2017

Space-Time Computing with Temporal Neural Networks

James E. Smith
2017

Hardware and Software Support for Virtualization

Edouard Bugnion, Jason Nieh, and Dan Tsafir
2017

[Datacenter Design and Management: A Computer Architect's Perspective](#)

Benjamin C. Lee

2016

[A Primer on Compression in the Memory Hierarchy](#)

Somayeh Sardashti, Angelos Arelakis, Per Stenström, and David A. Wood

2015

[Research Infrastructures for Hardware Accelerators](#)

Yakun Sophia Shao and David Brooks

2015

[Analyzing Analytics](#)

Rajesh Bordawekar, Bob Blainey, and Ruchir Puri

2015

[Customizable Computing](#)

Yu-Ting Chen, Jason Cong, Michael Gill, Glenn Reinman, and Bingjun Xiao

2015

[Die-stacking Architecture](#)

Yuan Xie and Jishen Zhao

2015

[Single-Instruction Multiple-Data Execution](#)

Christopher J. Hughes

2015

[Power-Efficient Computer Architectures: Recent Advances](#)

Magnus Sjölander, Margaret Martonosi, and Stefanos Kaxiras

2014

[FPGA-Accelerated Simulation of Computer Systems](#)

Hari Angepat, Derek Chiou, Eric S. Chung, and James C. Hoe

2014

[A Primer on Hardware Prefetching](#)

Babak Falsafi and Thomas F. Wenisch

2014

[On-Chip Photonic Interconnects: A Computer Architect's Perspective](#)

Christopher J. Nitta, Matthew K. Farrens, and Venkatesh Akella

2013

Optimization and Mathematical Modeling in Computer Architecture

Tony Nowatzki, Michael Ferris, Karthikeyan Sankaralingam, Cristian Estan, Nilay Vaish, and David Wood

2013

Security Basics for Computer Architects

Ruby B. Lee

2013

The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition

Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle

2013

Shared-Memory Synchronization

Michael L. Scott

2013

Resilient Architecture Design for Voltage Variation

Vijay Janapa Reddi and Meeta Sharma Gupta

2013

Multithreading Architecture

Mario Nemirovsky and Dean M. Tullsen

2013

Performance Analysis and Tuning for General Purpose Graphics Processing Units (GPGPU)

Hyesoon Kim, Richard Vuduc, Sara Bagsorkhi, Jee Choi, and Wen-mei Hwu

2012

Automatic Parallelization: An Overview of Fundamental Compiler Techniques

Samuel P. Midkiff

2012

Phase Change Memory: From Devices to Systems

Moinuddin K. Qureshi, Sudhanva Gurumurthi, and Bipin Rajendran

2011

Multi-Core Cache Hierarchies

Rajeev Balasubramonian, Norman P. Jouppi, and Naveen Muralimanohar

2011

A Primer on Memory Consistency and Cache Coherence

Daniel J. Sorin, Mark D. Hill, and David A. Wood

2011

[Dynamic Binary Modification: Tools, Techniques, and Applications](#)

Kim Hazelwood

2011

[Quantum Computing for Computer Architects, Second Edition](#)

Tzvetan S. Metodi, Arvin I. Faruque, and Frederic T. Chong

2011

[High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities](#)

Dennis Abts and John Kim

2011

[Processor Microarchitecture: An Implementation Perspective](#)

Antonio González, Fernando Latorre, and Grigorios Magklis

2010

[Transactional Memory, 2nd edition](#)

Tim Harris, James Larus, and Ravi Rajwar

2010

[Computer Architecture Performance Evaluation Methods](#)

Lieven Eeckhout

2010

[Introduction to Reconfigurable Supercomputing](#)

Marco Lanzagorta, Stephen Bique, and Robert Rosenberg

2009

[On-Chip Networks](#)

Natalie Enright Jerger and Li-Shiuan Peh

2009

[The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It](#)

Bruce Jacob

2009

[Fault Tolerant Computer Architecture](#)

Daniel J. Sorin

2009

[The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines](#)

Luiz André Barroso and Urs Hölzle

2009

Computer Architecture Techniques for Power-Efficiency

Stefanos Kaxiras and Margaret Martonosi

2008

Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency

Kunle Olukotun, Lance Hammond, and James Laudon

2007

Transactional Memory

James R. Larus and Ravi Rajwar

2006

Quantum Computing for Computer Architects

Tzvetan S. Metodi and Frederic T. Chong

2006

© Springer Nature Switzerland AG 2022
Reprint of original edition © Morgan & Claypool 2018

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopy, recording, or any other except for brief quotations in printed reviews, without the prior permission of the publisher.

Compiling Algorithms for Heterogeneous Systems
Steven Bell, Jing Pu, James Hegarty, and Mark Horowitz

ISBN: 978-3-031-00630-2 paperback
ISBN: 978-3-031-01758-2 ebook
ISBN: 978-3-031-00055-3 hardcover

DOI10.1007/978-3-031-01758-2

A Publication in the Springer series
SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE

Lecture #43
Series Editor: Margaret Martonosi, *Princeton University*
Founding Editor Emeritus: Mark D. Hill, *University of Wisconsin, Madison*
Series ISSN
Print 1935-3235 Electronic 1935-3243

Compiling Algorithms for Heterogeneous Systems

Steven Bell
Stanford University

Jing Pu
Google

James Hegarty
Oculus

Mark Horowitz
Stanford University

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE #43

ABSTRACT

Most emerging applications in imaging and machine learning must perform immense amounts of computation while holding to strict limits on energy and power. To meet these goals, architects are building increasingly specialized compute engines tailored for these specific tasks. The resulting computer systems are heterogeneous, containing multiple processing cores with wildly different execution models. Unfortunately, the cost of producing this specialized hardware—and the software to control it—is astronomical. Moreover, the task of porting algorithms to these heterogeneous machines typically requires that the algorithm be partitioned across the machine and rewritten for each specific architecture, which is time consuming and prone to error.

Over the last several years, the authors have approached this problem using domain-specific languages (DSLs): high-level programming languages customized for specific domains, such as database manipulation, machine learning, or image processing. By giving up generality, these languages are able to provide high-level abstractions to the developer while producing high-performance output. The purpose of this book is to spur the adoption and the creation of domain-specific languages, especially for the task of creating hardware designs.

In the first chapter, a short historical journey explains the forces driving computer architecture today. Chapter 2 describes the various methods for producing designs for accelerators, outlining the push for more abstraction and the tools that enable designers to work at a higher conceptual level. From there, Chapter 3 provides a brief introduction to image processing algorithms and hardware design patterns for implementing them. Chapters 4 and 5 describe and compare Darkroom and Halide, two domain-specific languages created for image processing that produce high-performance designs for both FPGAs and CPUs from the same source code, enabling rapid design cycles and quick porting of algorithms. The final section describes how the DSL approach also simplifies the problem of interfacing between application code and the accelerator by generating the driver stack in addition to the accelerator configuration.

This book should serve as a useful introduction to domain-specialized computing for computer architecture students and as a primer on domain-specific languages and image processing hardware for those with more experience in the field.

KEYWORDS

domain-specific languages, high-level synthesis, compilers, image processing accelerators, stencil computation

Contents

Preface	xiii
Acknowledgments	xv
1 Introduction	1
1.1 CMOS Scaling and the Rise of Specialization	2
1.2 What Will We Build Now?	7
1.2.1 Performance, Power, and Area	7
1.2.2 Flexibility	9
1.3 The Cost of Specialization	10
1.4 Good Applications for Acceleration	13
2 Computations and Compilers	17
2.1 Direct Specification	18
2.2 Compilers	19
2.3 High-level Synthesis	21
2.4 Domain-specific Languages	24
3 Image Processing with Stencil Pipelines	27
3.1 Image Signal Processors	27
3.2 Example Applications	29
4 Darkroom: A Stencil Language for Image Processing	33
4.1 Language Description	34
4.2 A Simple Pipeline in Darkroom	36
4.3 Optimal Synthesis of Line-buffered Pipelines	37
4.3.1 Generating Line-buffered Pipelines	37
4.3.2 Shift Operator	39
4.3.3 Finding Optimal Shifts	39
4.4 Implementation	42
4.4.1 ASIC and FPGA Synthesis	43
4.4.2 CPU Compilation	44

4.5	Evaluation	46
4.5.1	Scheduling for Hardware Synthesis	47
4.5.2	Scheduling for General-purpose Processors	49
4.6	Summary	50
5	Programming CPU/FPGA Systems from Halide	51
5.1	The Halide Language	52
5.2	Mapping Halide to Hardware	53
5.3	Compiler Implementation	57
5.3.1	Architecture Parameter Extraction	57
5.3.2	IR Transformation	59
5.3.3	Loop Perfection Optimization	59
5.3.4	Code Generation	61
5.4	Implementation and Evaluation	63
5.4.1	Programmability and Efficiency	63
5.4.2	Quality of Hardware Generation	66
5.5	Conclusion	68
6	Interfacing with Specialized Hardware	69
6.1	Common Interfaces	69
6.2	The Challenge of Interfaces	70
6.3	Solutions to the Interface Problem	71
6.3.1	Compiler Support	71
6.3.2	Library Interface	71
6.3.3	API plus DSL	71
6.4	Drivers for Darkroom and Halide on FPGA	72
6.4.1	Memory and Coherency	73
6.4.2	Running the Hardware	74
6.4.3	Generating Systems and Drivers	76
6.4.4	Generating the Whole Stack with Halide	76
6.4.5	Heterogeneous System Performance	77
7	Conclusions and Future Directions	81
	Bibliography	83
	Authors' Biographies	89

Preface

Cameras are ubiquitous, and computers are increasingly being used to process image data to produce better images, recognize objects, build representations of the physical world, and extract salient bits from massive streams of video, among countless other things. But while the data deluge continues to increase, and while the number of transistors that can be cost-effectively placed on a silicon die is still going up (for now), limitations on power and energy mean that traditional CPUs alone are insufficient to meet the demand. As a result, architects are building more and more specialized compute engines tailored to provide energy and performance gains on these specific tasks.

Unfortunately, the cost of producing this specialized hardware—and the software to control it—is astronomical. Moreover, the resulting computer systems are heterogeneous, containing multiple processing cores with wildly different execution models. The task of porting algorithms to these heterogeneous machines typically requires that the algorithm be partitioned across the machine and rewritten for each specific architecture, which is time consuming and prone to error.

Over the last several years, we have approached this problem using domain-specific languages (DSLs)—high-level programming languages customized for specific domains, such as database manipulation, machine learning, or image processing. By giving up generality, these languages are able to provide high-level abstractions to the developer while producing high-performance output. Our purpose in writing this book is to spur the adoption and the creation of domain-specific languages, especially for the task of creating hardware designs.

This book is not an exhaustive description of image processing accelerators, nor of domain-specific languages. Instead, we aim to show why DSLs make sense in light of the current state of computer architecture and development tools, and to illustrate with some specific examples what advantages DSLs provide, and what tradeoffs must be made when designing them. Our examples will come from image processing, and our primary targets are mixed CPU/FPGA systems, but the underlying techniques and principles apply to other domains and platforms as well. We assume only passing familiarity with image processing, and focus our discussion on the architecture and compiler sides of the problem.

In the first chapter, we take a short historical journey to explain the forces driving computer architecture today. Chapter 2 describes the various methods for producing designs for accelerators, outlining the push for more abstraction and the tools that enable designers to work at a higher conceptual level. In Chapter 3, we provide a brief introduction to image processing algorithms and hardware design patterns for implementing them, which we use through the rest of the book. Chapters 4 and 5 describe Darkroom and Halide, two domain-specific lan-

guages created for image processing. Both are able to produce high-performance designs for both FPGAs and CPUs from the same source code, enabling rapid design cycles and quick porting of algorithms. We present both of these examples because comparing and contrasting them illustrates some of the tradeoffs and design decisions encountered when creating a DSL. The final portion of the book discusses the task of controlling specialized hardware within a heterogeneous system running a multiuser operating system. We give a brief overview of how this works on Linux and show how DSLs enable us to automatically generate the necessary driver and interface code, greatly simplifying the creation of that interface.

This book assumes at least some background in computer architecture, such as an advanced undergraduate or early graduate course in CPU architecture. We also build on ideas from compilers, programming languages, FPGA synthesis, and operating systems, but the book should be accessible to those without extensive study on these topics.

Steven Bell, Jing Pu, James Hegarty, and Mark Horowitz
January 2018

Acknowledgments

Any work of this size is necessarily the result of many collaborations. We are grateful to John Brunhaver, Zachary DeVito, Pat Hanrahan, Jonathan Ragan-Kelley, Steve Richardson, Jeff Setter, Artem Vasilyev, and Xuan Yang, who influenced our thinking on these topics and helped develop portions of the systems described in this book. We're also thankful to Mike Morgan, Margaret Martonosi, and the team at Morgan & Claypool for shepherding us through the writing and production process, and to the reviewers whose feedback made this a much better manuscript than it would have been otherwise.

Steven Bell, Jing Pu, James Hegarty, and Mark Horowitz
January 2018