Compiling Algorithms for Heterogeneous Systems

Synthesis Lectures on Computer Architecture

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

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 languages 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