

# The Compiler Design Handbook Optimizations And Machine Code Generation

This new, expanded textbook describes all phases of a modern compiler: lexical analysis, parsing, abstract syntax, semantic actions, intermediate representations, instruction selection via tree matching, dataflow analysis, graph-coloring register allocation, and runtime systems. It includes good coverage of current techniques in code generation and register allocation, as well as functional and object-oriented languages, that are missing from most books. In addition, more advanced chapters are now included so that it can be used as the basis for a two-semester or graduate course. The most accepted and successful techniques are described in a concise way, rather than as an exhaustive catalog of every possible variant. Detailed descriptions of the interfaces between modules of a compiler are illustrated with actual C header files. The first part of the book, Fundamentals of Compilation, is suitable for a one-semester first course in compiler design. The second part, Advanced Topics, which includes the advanced chapters, covers the compilation of object-oriented and functional languages, garbage collection, loop optimizations, SSA form, loop scheduling, and optimization for cache-memory hierarchies.

While compilers for high-level programming languages are large complex software systems, they have particular characteristics that differentiate them from other software systems. Their functionality is almost completely well-defined – ideally there exist complete precise descriptions of the source and target languages. Additional descriptions of the interfaces to the operating system, programming system and programming environment, and to other compilers and libraries are often available. This book deals with the analysis phase of translators for programming languages. It describes lexical, syntactic and semantic analysis, specification mechanisms for these tasks from the theory of formal languages, and methods for automatic generation based on the theory of automata. The authors present a conceptual translation structure, i.e., a division into a set of modules, which transform an input program into a sequence of steps in a machine program, and they then describe the interfaces between the modules. Finally, the structures of real translators are outlined. The book contains the necessary theory and advice for implementation. This book is intended for students of computer science. The book is supported throughout with examples, exercises and program fragments.

Get to grips with various performance improvement techniques such as concurrency, lock-free programming, atomic operations, parallelism, and memory management

**Key Features**

- Understand the limitations of modern CPUs and their performance impact
- Find out how you can avoid writing inefficient code and get the best optimizations from the compiler
- Learn the tradeoffs and costs of writing high-performance programs

**Book Description**

The great free lunch of "performance taking care of itself" is over. Until recently, programs got faster by themselves as CPUs were upgraded, but that doesn't happen anymore. The clock frequency of new processors has almost peaked. New architectures provide small improvements to existing programs, but this only helps slightly. Processors do get larger and more powerful, but most of this new power is consumed by the increased number of processing cores and other "extra" computing

## Access Free The Compiler Design Handbook Optimizations And Machine Code Generation

units. To write efficient software, you now have to know how to program by making good use of the available computing resources, and this book will teach you how to do that. The book covers all the major aspects of writing efficient programs, such as using CPU resources and memory efficiently, avoiding unnecessary computations, measuring performance, and how to put concurrency and multithreading to good use. You'll also learn about compiler optimizations and how to use the programming language (C++) more efficiently. Finally, you'll understand how design decisions impact performance. By the end of this book, you'll not only have enough knowledge of processors and compilers to write efficient programs, but you'll also be able to understand which techniques to use and what to measure while improving performance. At its core, this book is about learning how to learn. What you will learn Discover how to use the hardware computing resources in your programs effectively Understand the relationship between memory order and memory barriers Familiarize yourself with the performance implications of different data structures and organizations Assess the performance impact of concurrent memory accessed and how to minimize it Discover when to use and when not to use lock-free programming techniques Explore different ways to improve the effectiveness of compiler optimizations Design APIs for concurrent data structures and high-performance data structures to avoid inefficiencies Who this book is for This book is for experienced developers and programmers who work on performance-critical projects and want to learn different techniques to improve the performance of their code. Programmers who belong to algorithmic trading, gaming, bioinformatics, computational genomics, or computational fluid dynamics communities can learn various techniques from this book and apply them in their domain of work. Although this book uses the C++ language, the concepts demonstrated in the book can be easily transferred or applied to other compiled languages such as C, Java, Rust, Go, and more.

LLVM is one of the most popular and powerful frameworks for creating programming language tools such as compilers, source analyzers, and dynamic execution runtime. With this book, you'll discover some of the most critical parts of LLVM and get hands-on using LLVM to build projects in no time.

This entirely revised second edition of *Engineering a Compiler* is full of technical updates and new material covering the latest developments in compiler technology. In this comprehensive text you will learn important techniques for constructing a modern compiler. Leading educators and researchers Keith Cooper and Linda Torczon combine basic principles with pragmatic insights from their experience building state-of-the-art compilers. They will help you fully understand important techniques such as compilation of imperative and object-oriented languages, construction of static single assignment forms, instruction scheduling, and graph-coloring register allocation. In-depth treatment of algorithms and techniques used in the front end of a modern compiler Focus on code optimization and code generation, the primary areas of recent research and development Improvements in presentation including conceptual overviews for each chapter, summaries and review questions for sections, and prominent placement of definitions for new terms Examples drawn from several different programming languages

Software -- Operating Systems.

This book explores break-through approaches to tackling and mitigating the well-known

## Access Free The Compiler Design Handbook Optimizations And Machine Code Generation

problems of compiler optimization using design space exploration and machine learning techniques. It demonstrates that not all the optimization passes are suitable for use within an optimization sequence and that, in fact, many of the available passes tend to counteract one another. After providing a comprehensive survey of currently available methodologies, including many experimental comparisons with state-of-the-art compiler frameworks, the book describes new approaches to solving the problem of selecting the best compiler optimizations and the phase-ordering problem, allowing readers to overcome the enormous complexity of choosing the right order of optimizations for each code segment in an application. As such, the book offers a valuable resource for a broad readership, including researchers interested in Computer Architecture, Electronic Design Automation and Machine Learning, as well as computer architects and compiler developers.

This book provides readers with a single-source reference to static-single assignment (SSA)-based compiler design. It is the first (and up-to-now only) book that covers in a deep and comprehensive way how an optimizing compiler can be designed using the SSA form. State-of-the-art experts on the subject from globally recognized compiler research centers and engineering practitioners from companies such as Google, Facebook, IBM, and Amazon have collaborated to create a coherent and uniform presentation, in terms of structure, sequence, terminology, and notations. Provides first single-source reference to widely adopted, static-single assignment (SSA) form of compiler design; Offers readers state-of-the-art, advanced compiler optimization techniques; Includes content from experts on the subject from globally recognized compiler research centers and engineering practitioners.

Modern computer architectures designed with high-performance microprocessors offer tremendous potential gains in performance over previous designs. Yet their very complexity makes it increasingly difficult to produce efficient code and to realize their full potential. This landmark text from two leaders in the field focuses on the pivotal role that compilers can play in addressing this critical issue. The basis for all the methods presented in this book is data dependence, a fundamental compiler analysis tool for optimizing programs on high-performance microprocessors and parallel architectures. It enables compiler designers to write compilers that automatically transform simple, sequential programs into forms that can exploit special features of these modern architectures. The text provides a broad introduction to data dependence, to the many transformation strategies it supports, and to its applications to important optimization problems such as parallelization, compiler memory hierarchy management, and instruction scheduling. The authors demonstrate the importance and wide applicability of dependence-based compiler optimizations and give the compiler writer the basics needed to understand and implement them. They also offer cookbook explanations for transforming applications by hand to computational scientists and engineers who are driven to obtain the best possible performance of their complex applications. The approaches presented are based on research conducted over the past two decades, emphasizing the strategies implemented in research prototypes at Rice University and in several associated commercial systems. Randy Allen and Ken Kennedy have provided an indispensable resource for researchers, practicing professionals, and graduate students engaged in designing and optimizing compilers for modern computer architectures. \* Offers a guide to the simple, practical algorithms and approaches that are most effective in real-world, high-performance microprocessor and parallel systems. \* Demonstrates each transformation in worked examples. \* Examines how two case study compilers implement the theories and practices described in each chapter. \* Presents the most complete treatment of memory hierarchy

## Access Free The Compiler Design Handbook Optimizations And Machine Code Generation

issues of any compiler text. \* Illustrates ordering relationships with dependence graphs throughout the book. \* Applies the techniques to a variety of languages, including Fortran 77, C, hardware definition languages, Fortran 90, and High Performance Fortran. \* Provides extensive references to the most sophisticated algorithms known in research.

While compilers for high-level programming languages are large complex software systems, they have particular characteristics that differentiate them from other software systems. Their functionality is almost completely well-defined – ideally there exist complete precise descriptions of the source and target languages. Additional descriptions of the interfaces to the operating system, programming system and programming environment, and to other compilers and libraries are often available. The final stage of a compiler is generating efficient code for the target microprocessor. The applied techniques are different from usual compiler optimizations because code generation has to take into account the resource constraints of the processor – it has a limited number of registers, functional units, instruction decoders, and so on. The efficiency of the generated code significantly depends on the algorithms used to map the program to the processor, however these algorithms themselves depend not only on the target processor but also on several design decisions in the compiler itself – e.g., the program representation used in machine-independent optimization. In this book, the authors discuss classical code generation approaches that are well suited to existing compiler infrastructures, and they also present new algorithms based on state-of-the-art program representations as used in modern compilers and virtual machines using just-in-time compilation. This book is intended for students of computer science. The book is supported throughout with examples, exercises and program fragments.

Broad in scope, involving theory, the application of that theory, and programming technology, compiler construction is a moving target, with constant advances in compiler technology taking place. Today, a renewed focus on do-it-yourself programming makes a quality textbook on compilers, that both students and instructors will enjoy using, of even more vital importance. This book covers every topic essential to learning compilers from the ground up and is accompanied by a powerful and flexible software package for evaluating projects, as well as several tutorials, well-defined projects, and test cases.

The control and data flow of a program can be represented using continuations, a concept from denotational semantics that has practical application in real compilers. This book shows how continuation-passing style is used as an intermediate representation on which to perform optimisations and program transformations. Continuations can be used to compile most programming languages. The method is illustrated in a compiler for the programming language Standard ML. However, prior knowledge of ML is not necessary, as the author carefully explains each concept as it arises. This is the first book to show how concepts from the theory of programming languages can be applied to the production of practical optimising compilers for modern languages like ML. This book will be essential reading for compiler writers in both industry and academe, as well as for students and researchers in programming language theory.

An industry insider explains why there is so much bad software—and why academia doesn't teach programmers what industry wants them to know. Why is software so prone to bugs? So vulnerable to viruses? Why are software products so often delayed, or even canceled? Is software development really hard, or are software developers just not that good at it? In *The Problem with Software*, Adam Barr examines the proliferation of bad software, explains what causes it, and offers some suggestions on how to improve the situation. For one thing, Barr points out, academia doesn't teach programmers what they actually need to know to do their jobs: how to work in a team to create code that works reliably and can be maintained by somebody other than the original authors. As the size and complexity of commercial software have grown, the gap between academic computer science and industry has widened. It's an



## Access Free The Compiler Design Handbook Optimizations And Machine Code Generation

open secret that there is little engineering in software engineering, which continues to rely not on codified scientific knowledge but on intuition and experience. Barr, who worked as a programmer for more than twenty years, describes how the industry has evolved, from the era of mainframes and Fortran to today's embrace of the cloud. He explains bugs and why software has so many of them, and why today's interconnected computers offer fertile ground for viruses and worms. The difference between good and bad software can be a single line of code, and Barr includes code to illustrate the consequences of seemingly inconsequential choices by programmers. Looking to the future, Barr writes that the best prospect for improving software engineering is the move to the cloud. When software is a service and not a product, companies will have more incentive to make it good rather than "good enough to ship."

"Modern Compiler Design" makes the topic of compiler design more accessible by focusing on principles and techniques of wide application. By carefully distinguishing between the essential (material that has a high chance of being useful) and the incidental (material that will be of benefit only in exceptional cases) much useful information was packed in this comprehensive volume. The student who has finished this book can expect to understand the workings of and add to a language processor for each of the modern paradigms, and be able to read the literature on how to proceed. The first provides a firm basis, the second potential for growth. For real-time systems, the worst-case execution time (WCET) is the key objective to be considered. Traditionally, code for real-time systems is generated without taking this objective into account and the WCET is computed only after code generation. Worst-Case Execution Time Aware Compilation Techniques for Real-Time Systems presents the first comprehensive approach integrating WCET considerations into the code generation process. Based on the proposed reconciliation between a compiler and a timing analyzer, a wide range of novel optimization techniques is provided. Among others, the techniques cover source code and assembly level optimizations, exploit machine learning techniques and address the design of modern systems that have to meet multiple objectives. Using these optimizations, the WCET of real-time applications can be reduced by about 30% to 45% on the average. This opens opportunities for decreasing clock speeds, costs and energy consumption of embedded processors. The proposed techniques can be used for all types real-time systems, including automotive and avionics IT systems.

Obtain better system performance, lower energy consumption, and avoid hand-coding arithmetic functions with this concise guide to automated optimization techniques for hardware and software design. High-level compiler optimizations and high-speed architectures for implementing FIR filters are covered, which can improve performance in communications, signal processing, computer graphics, and cryptography. Clearly explained algorithms and illustrative examples throughout make it easy to understand the techniques and write software for their implementation. Background information on the synthesis of arithmetic expressions and computer arithmetic is also included, making the book ideal for newcomers to the subject. This is an invaluable resource for researchers, professionals, and graduate students working in system level design and automation, compilers, and VLSI CAD.

A refreshing antidote to heavy theoretical tomes, this book is a concise, practical guide to modern compiler design and construction by an acknowledged master. Readers are taken step-by-step through each stage of compiler design, using the simple yet powerful method of recursive descent to create a compiler for Oberon-0, a subset of the author's Oberon language. A disk provided with the book gives full listings of the Oberon-0 compiler and associated tools. The hands-on, pragmatic approach makes the book equally attractive for project-oriented courses in compiler design and for software engineers wishing to develop their skills in system software.

The widespread use of object-oriented languages and Internet security concerns are

## Access Free The Compiler Design Handbook Optimizations And Machine Code Generation

just the beginning. Add embedded systems, multiple memory banks, highly pipelined units operating in parallel, and a host of other advances and it becomes clear that current and future computer architectures pose immense challenges to compiler designers-challenges th

The Compiler Design Handbook Optimizations and Machine Code Generation, Second Edition CRC Press

Building an Optimizing Compiler provides a high-level design for a thorough optimizer, code generator, scheduler, and register allocator for a generic modern RISC processor. In the process it addresses the small issues that have a large impact on the implementation. The book approaches this subject from a practical viewpoint. Theory is introduced where intuitive arguments are insufficient; however, the theory is described in practical terms. Building an Optimizing Compiler provides a complete theory for static single assignment methods and partial redundancy methods for code optimization. It also provides a new generalization of register allocation techniques. A single running example is used throughout the book to illustrate the compilation process.

This book brings a unique treatment of compiler design to the professional who seeks an in-depth examination of a real-world compiler. Chris Fraser of AT & T Bell Laboratories and David Hanson of Princeton University codeveloped lcc, the retargetable ANSI C compiler that is the focus of this book. They provide complete source code for lcc; a target-independent front end and three target-dependent back ends are packaged as a single program designed to run on three different platforms. Rather than transfer code into a text file, the book and the compiler itself are generated from a single source to ensure accuracy.

Software -- Programming Languages.

This book is an introduction to the field of compiler construction. It combines a detailed study of the theory underlying the modern approach to compiler design, together with many practical examples, and a complete description, with source code, of a compiler for a small language. It is specifically designed for use in an introductory course on compiler design or compiler construction at the advanced undergraduate level. This textbook is intended for an introductory course on Compiler Design, suitable for use in an undergraduate programme in computer science or related fields. This book undertakes to provide the proper balance between theory and practice, and to provide enough actual implementation detail to give a real flavor for the techniques without overwhelming the reader. In this text, I provide a complete compiler for a small language, written in C, and developed using the different techniques studied in each chapter. In addition, detailed descriptions of coding techniques for additional language examples are given as the associated topics are studied. Finally, each chapter concludes with an extensive set of exercises, which are divided into two sections. The first contains those of the more pencil-and-paper variety involving little programming. The second contains those involving a significant amount of programming. Simply In Depth.....

This book presents a novel approach for Architecture Description Language (ADL)-based instruction-set description that enables the automatic retargeting of the complete software toolkit from a single ADL processor model.

Designed for an introductory course, this text encapsulates the topics essential for a freshman course on compilers. The book provides a balanced coverage of both theoretical and practical

## Access Free The Compiler Design Handbook Optimizations And Machine Code Generation

aspects. The text helps the readers understand the process of compilation and proceeds to explain the design and construction of compilers in detail. The concepts are supported by a good number of compelling examples and exercises.

A compiler translates a program written in a high level language into a program written in a lower level language. For students of computer science, building a compiler from scratch is a rite of passage: a challenging and fun project that offers insight into many different aspects of computer science, some deeply theoretical, and others highly practical. This book offers a one semester introduction into compiler construction, enabling the reader to build a simple compiler that accepts a C-like language and translates it into working X86 or ARM assembly language. It is most suitable for undergraduate students who have some experience programming in C, and have taken courses in data structures and computer architecture.

Coding and testing are often considered separate areas of expertise. In this comprehensive guide, author and Java expert Scott Oaks takes the approach that anyone who works with Java should be equally adept at understanding how code behaves in the JVM, as well as the tunings likely to help its performance. You'll gain in-depth knowledge of Java application performance, using the Java Virtual Machine (JVM) and the Java platform, including the language and API. Developers and performance engineers alike will learn a variety of features, tools, and processes for improving the way Java 7 and 8 applications perform. Apply four principles for obtaining the best results from performance testing Use JDK tools to collect data on how a Java application is performing Understand the advantages and disadvantages of using a JIT compiler Tune JVM garbage collectors to affect programs as little as possible Use techniques to manage heap memory and JVM native memory Maximize Java threading and synchronization performance features Tackle performance issues in Java EE and Java SE APIs Improve Java-driven database application performance

A compiler translates a high-level language program into a functionally equivalent low-level language program that can be understood and executed by the computer. Crucial to any computer system, effective compiler design is also one of the most complex areas of system development. Before any code for a modern compiler is even written, many students and even experienced programmers have difficulty with the high-level algorithms that will be necessary for the compiler to function. Written with this in mind, Algorithms for Compiler Design teaches the fundamental algorithms that underlie modern compilers. The book focuses on the "front-end" of compiler design: lexical analysis, parsing, and syntax. Blending theory with practical examples throughout, the book presents these difficult topics clearly and thoroughly. The final chapters on code generation and optimization complete a solid foundation for learning the broader requirements of an entire compiler design.

Maintaining a balance between a theoretical and practical approach to this important subject, Elements of Compiler Design serves as an introduction to compiler writing for undergraduate students. From a theoretical viewpoint, it introduces rudimentary models, such as automata and grammars, that underlie compilation and its essential phases. Based on these models, the author details the concepts, methods, and techniques employed in compiler design in a clear and easy-to-follow way. From a practical point of view, the book describes how compilation techniques are implemented. In fact, throughout the text, a case study illustrates the design of a new programming language and the construction of its compiler. While discussing various compilation techniques, the author demonstrates their implementation through this case study. In addition, the book presents many detailed examples and computer programs to emphasize the applications of the compiler algorithms. After studying this self-contained textbook, students should understand the compilation process, be able to write a simple real compiler, and easily follow advanced books on the subject.

Become familiar with the LLVM infrastructure and start using LLVM libraries to design a compiler About This Book Learn to use the LLVM libraries to emit intermediate representation

## Access Free The Compiler Design Handbook Optimizations And Machine Code Generation

(IR) from high-level language Build your own optimization pass for better code generation Understand AST generation and use it in a meaningful way Who This Book Is For This book is intended for those who already know some of the concepts of compilers and want to quickly get familiar with the LLVM infrastructure and the rich set of libraries that it provides. What You Will Learn Get an introduction to LLVM modular design and LLVM tools Convert frontend code to LLVM IR Implement advanced LLVM IR paradigms Understand the LLVM IR Optimization Pass Manager infrastructure and write an optimization pass Absorb LLVM IR transformations Understand the steps involved in converting LLVM IR to Selection DAG Implement a custom target using the LLVM infrastructure Get a grasp of C's frontend clang, an AST dump, and static analysis In Detail LLVM is currently the point of interest for many firms, and has a very active open source community. It provides us with a compiler infrastructure that can be used to write a compiler for a language. It provides us with a set of reusable libraries that can be used to optimize code, and a target-independent code generator to generate code for different backends. It also provides us with a lot of other utility tools that can be easily integrated into compiler projects. This book details how you can use the LLVM compiler infrastructure libraries effectively, and will enable you to design your own custom compiler with LLVM in a snap. We start with the basics, where you'll get to know all about LLVM. We then cover how you can use LLVM library calls to emit intermediate representation (IR) of simple and complex high-level language paradigms. Moving on, we show you how to implement optimizations at different levels, write an optimization pass, generate code that is independent of a target, and then map the code generated to a backend. The book also walks you through CLANG, IR to IR transformations, advanced IR block transformations, and target machines. By the end of this book, you'll be able to easily utilize the LLVM libraries in your own projects. Style and approach This book deals with topics sequentially, increasing the difficulty level in a step-by-step approach. Each topic is explained with a detailed example, and screenshots are included to help you understand the examples.

LEARNING STARTS WITH VIEWING THE WORLD DIFFERENTLY. Knowledge flow — A mobile learning platform provides Apps and Books. Knowledge flow provides learning book of Compiler Design. This book is for all information technology, computer science and students, teachers and professionals across the world. Compiler design principles explain in-depth view of translation and optimization process. This compiler design book delivers the updated information and basic concepts. Contents: 1. Introduction Compiler Design 2. Phases of Compiler 3. Cousins of compiler and Construction tools 4. Lexical Analysis 5. Syntax Analysis 6. Syntax Directed Translation 7. Type Checking 8. Intermediate Code Generation 9. Code Generation 10. Code Optimizer

Scalable parallel systems or, more generally, distributed memory systems offer a challenging model of computing and pose fascinating problems regarding compiler optimization, ranging from language design to run time systems. Research in this area is foundational to many challenges from memory hierarchy optimizations to communication optimization. This unique, handbook-like monograph assesses the state of the art in the area in a systematic and comprehensive way. The 21 coherent chapters by leading researchers provide complete and competent coverage of all relevant aspects of compiler optimization for scalable parallel systems. The book is divided into five parts on languages, analysis, communication optimizations, code generation, and run time systems. This book will serve as a landmark source for education, information, and reference to students, practitioners, professionals, and researchers interested in updating their knowledge about or active in parallel computing. In today's fast and competitive world, a program's performance is just as important to customers as the features it provides. This practical guide teaches developers performance-tuning principles that enable optimization in C++. You'll learn how to make code that already embodies best practices of C++ design run faster and consume fewer



## Access Free The Compiler Design Handbook Optimizations And Machine Code Generation

resources on any computer--whether it's a watch, phone, workstation, supercomputer, or globe-spanning network of servers. Author Kurt Guntheroth provides several running examples that demonstrate how to apply these principles incrementally to improve existing code so it meets customer requirements for responsiveness and throughput. The advice in this book will prove itself the first time you hear a colleague exclaim, "Wow, that was fast. Who fixed something?"

Locate performance hot spots using the profiler and software timers  
Learn to perform repeatable experiments to measure performance of code changes  
Optimize use of dynamically allocated variables  
Improve performance of hot loops and functions  
Speed up string handling functions  
Recognize efficient algorithms and optimization patterns  
Learn the strengths--and weaknesses--of C++ container classes  
View searching and sorting through an optimizer's eye  
Make efficient use of C++ streaming I/O functions  
Use C++ thread-based concurrency features effectively

Compilers and operating systems constitute the basic interfaces between a programmer and the machine for which he is developing software. In this book we are concerned with the construction of the former. Our intent is to provide the reader with a firm theoretical basis for compiler construction and sound engineering principles for selecting alternate methods, implementing them, and integrating them into a reliable, economically viable product. The emphasis is upon a clean decomposition employing modules that can be re-used for many compilers, separation of concerns to facilitate team programming, and flexibility to accommodate hardware and system constraints. A reader should be able to understand the questions he must ask when designing a compiler for language X on machine Y, what tradeoffs are possible, and what performance might be obtained. He should not feel that any part of the design rests on whim; each decision must be based upon specific, identifiable characteristics of the source and target languages or upon design goals of the compiler. The vast majority of computer professionals will never write a compiler. Nevertheless, study of compiler technology provides important benefits for almost everyone in the field . • It focuses attention on the basic relationships between languages and machines. Understanding of these relationships eases the inevitable transitions to new hardware and programming languages and improves a person's ability to make appropriate tradeoffs in design and implementation .

While compilers for high-level programming languages are large complex software systems, they have particular characteristics that differentiate them from other software systems. Their functionality is almost completely well-defined - ideally there exist complete precise descriptions of the source and target languages. Additional descriptions of the interfaces to the operating system, programming system and programming environment, and to other compilers and libraries are often available. The book deals with the optimization phase of compilers. In this phase, programs are transformed in order to increase their efficiency. To preserve the semantics of the programs in these transformations, the compiler has to meet the associated applicability conditions. These are checked using static analysis of the programs. In this book the authors systematically describe the analysis and transformation of imperative and functional programs. In addition to a detailed description of important efficiency-improving transformations, the book offers a concise introduction to the necessary concepts and methods, namely to operational semantics, lattices, and fixed-point

