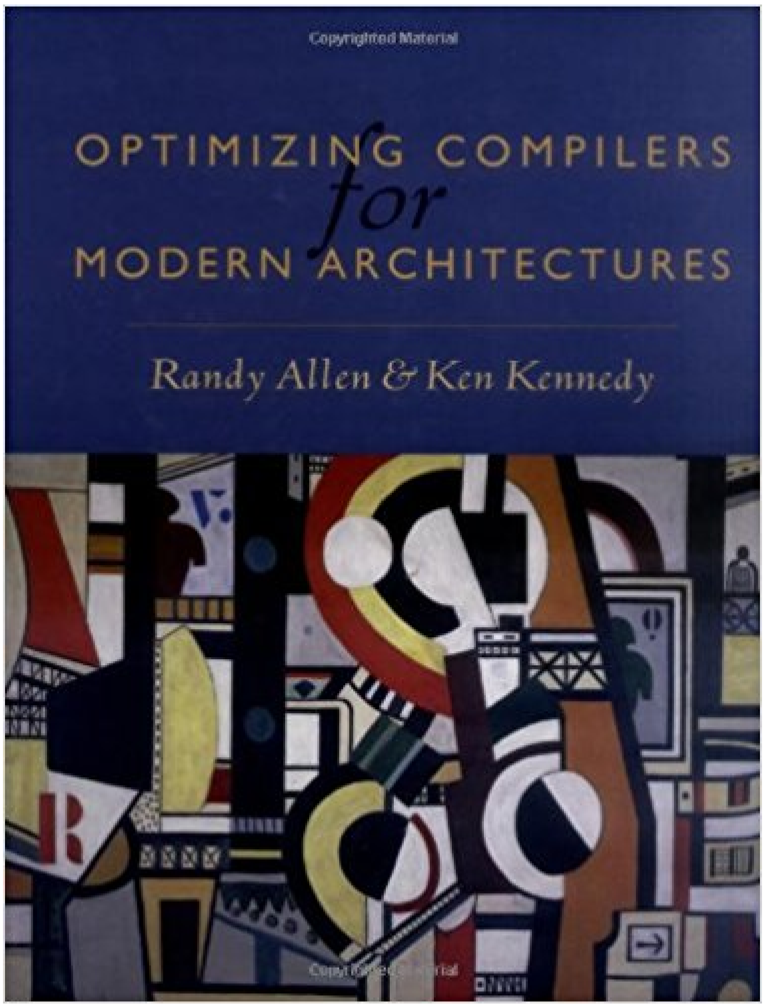
The book was found

Optimizing Compilers For Modern Architectures: A Dependence-based Approach



Synopsis

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 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.

Book Information

Hardcover: 790 pages

Publisher: Morgan Kaufmann; 1 edition (October 10, 2001)

Language: English

ISBN-10: 1558602860

ISBN-13: 978-1558602861

Product Dimensions: 7.7 x 1.7 x 9.6 inches

Shipping Weight: 3.3 pounds (View shipping rates and policies)

Average Customer Review: 4.0 out of 5 starsÂ Â See all reviewsÂ (8 customer reviews)

Best Sellers Rank: #1,424,838 in Books (See Top 100 in Books)   #91 inÂ Books > Computers & Technology > Programming > Languages & Tools > Compiler Design   #277 inÂ Books > Computers & Technology > Programming > Languages & Tools > Compilers   #665 inÂ Books > Computers & Technology > Hardware & DIY > Design & Architecture

Customer Reviews

Allen and Kennedy (A&K) haven’t written your first compiler book. There’s nothing about syntax analysis, code generation, instruction scheduling, or intermediate representations. You already know all that part, or you won’t get very far in this book. Once you have the basics down, A&K is an irreplaceable reference.It centers heavily on Fortran - even today, a mainstay of scientific computing and an active area of language development. Today, just as 50 years ago, the language’s straightforward structure makes detailed behavioral analysis relatively easy. That’s especially true in handling the array computations that soak up so many dozens (as of this writing) of CPU-hours per second on todays largest machines. There’s far too much to summarize here, but A&K cover a huge range of processor features, including caches, multiple ALUs, vector units, chaining, and more. C code gets some attention as well, much needed because of the cultural weirdness around array handling in C. In every case, the focus is on the real-world kernels that need the help and on explicit ways of identifying and manipulating those code structures. As a result, the authors disregard the unreal situations that sometimes arise, e.g. in "while (--n) *a++ = *b++ * *c++;"Yes, the arrays pointed to by a, b, and c can overlap. But the pointer a can also point to a, b, c, or n, somewhere in its range - and likewise for pointers b and c, or all three. There is essentially no limit to how bad this can get, e.g when n is an alias for a, b, or c. Yes these are rare situations and generally errors - but I’ve seen on-the-fly code generation in production environments, so even the A&K example isn’t as bad as it gets.

Download to continue reading...

Optimizing Compilers for Modern Architectures: A Dependence-based Approach Writing Compilers and Interpreters: A Software Engineering Approach Algorithms, Languages, Automata, And