Start at 13:15 local time

DF00100

Advanced Compiler Construction

2023

Organization, Motivation, Overview

https://www.ida.liu.se/~chrke/courses/ACC

Christoph Kessler, IDA,
Linköping University

Staff 2023

- Lectures / Presentation session / Examination
  - Christoph Kessler, IDA, Linköping University
    christoph.kessler \at liu.se
  - Welf Löwe, Linnaeus-University, Växjö,
    welf.lowe \at lnu.se (guest lecturer / guest examiner)
  - Martin Sjölund, IDA, Linköping University (guest lecturer)
  - August Ernstsson, IDA, Linköping University (guest lecturer)
  - Mattias Eriksson, Ericsson, Linköping (guest lecturer)
- Lessons
  - Christoph Kessler
- Course administrator
  - Anne Moe, IDA, anne.moe \at liu.se

Course moments (total: 9 6 hp)

- **Lectures and exam**
  - 2 lecture blocks (week 6 + week 7)
  - See course web page for schedule, contents
  - Written/oral exam 27 April 2023 09:00-13:00, 4.5hp
  - Mandatory presence 50% of the lectures + lessons for admission to presentation and exam
- **Labs**, 3 hp (could be done in groups of 2)
  - LLVM open-source compiler framework, llvm.org
  - Lab part 1: IR and program analysis, 1.5hp
  - Lab part 2: Code generation, 1.5hp
- **Presentation** 30 March 2023 09:15-… (whole day), 1.5hp
  - of a recent compiler research paper
  - Opposition on another presentation
  - Written summary with your own words, ca. 2 pages

Lessons ~~and Labs~~

Lessons:

- Theory exercises, good as preparation for the exam
- To get out most of the lessons for yourself:
  - Prepare your solutions ahead of time
  - Present your solution in class

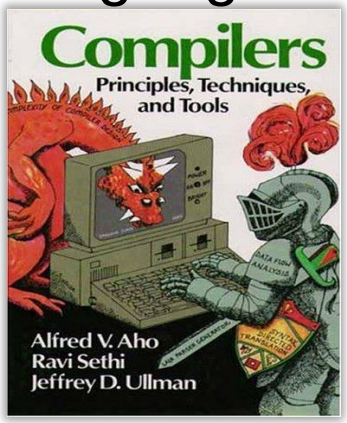
Labs: No labs in 2023

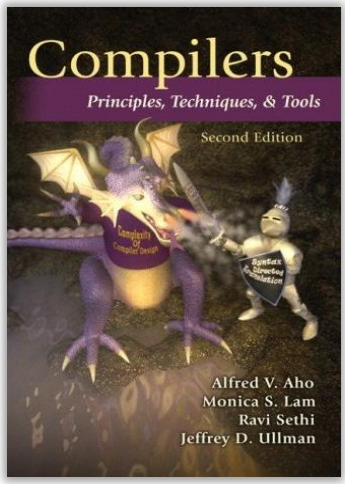
- Lab introduction today (Tuesday) at 13:15
- Mission critical, attendance is highly recommended

Why Another Compiler Course? (1)

Focus of traditional compiler courses (e.g., TDDB44, TDDD55):

- Understand concepts of programming languages
  - Syntax, semantics
- Good application of formal languages and automata theory
  - Lexing, parsing
- Toy languages and toy target architectures
- Front-end, parser generators, symbol table, AST, syntax-driven translation, quadruples, simple code generation
- Technology well-established since 1970s





Why Another Compiler Course? (2)

Current compiler technology R&D has a different focus:

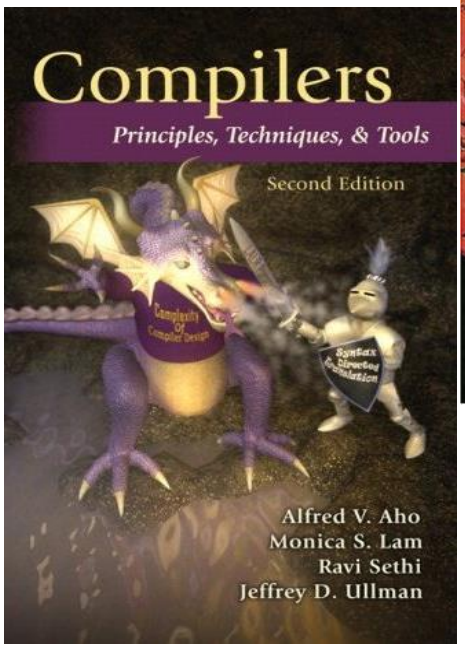
- Rate of programming language introduction is rather low
  - Mostly, DSLs (usually compile to C/C++)
  - Few students will be hired to write industrial frontends
- Rate of architectural change and variety is high
  - Processor architecture: DSP, superscalar, VLIW/EPIC, SIMD, GPU, ML accelerators, SMP, NUMA, Cluster, Multicore, MPSoC, reconfigurable, FPGA
  - Memory architecture: memory hierarchy, HBM, prefetching, device memory, local memory, memory banks, in-memory-computing, …
  - A new computer architecture does not sell without a (~C) compiler
  - Optimizing compilers vs. Manual low-level coding and tuning
- High requirements on code
  - Performance, Realtime constraints, Code size, Energy efficiency
  - Performance portability, utilization of accelerators
  - Moore's Law is slowing down → more performance growth must come from the software – most conveniently from an optimizing compiler
- Hot issues: Automatic program optimization, vectorization and parallelization, accelerator use, high-quality target code generation; run-time adaptivity; coupling with libraries
  - Required for this: Static analysis of programs, multi-level internal representations, graph algorithms, combinatorial optimization, architecture modeling
- Also hot, but not covered here: Static analysis for correctness and security

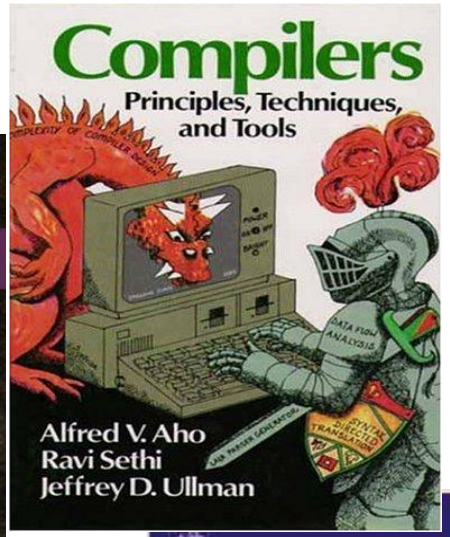
Contents

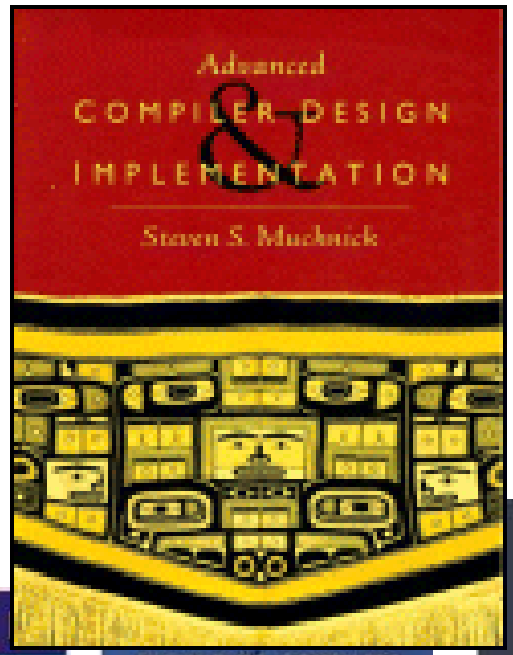
- **Advanced Intermediate Representation Design**
  - Multi-Level IRs
  - Static Single Assignment (SSA) Form
- **Static Analysis of Programs**
  - Control Flow Analysis
  - Data Flow Analysis
  - Abstract Interpretation
  - Points-to Analysis
  - Dependence Analysis
  - WCET Analysis (not this time)
- **High-Level Optimizations**
  - Loop Optimizations e.g. for Data Locality; Loop Parallelization; …
  - Task Fusion, Resource Allocation, Mapping, Scheduling, DVFS
- **Optimized Target Code Generation**
  - Instruction Selection, Instruction Scheduling, Register Allocation, …
- Code generation and optimization for special target architectures
  - DSP, clustered VLIW, SIMD, parallel, GPU, ...
- **Adaptivity, Autotuning and Other Issues** (as time permits)
- Guest lectures e.g. about **DSL compilation**, e.g. Modelica compiler, SkePU
- Student presentations about selected recent compiler research papers
- Labs: LLVM (not this time)

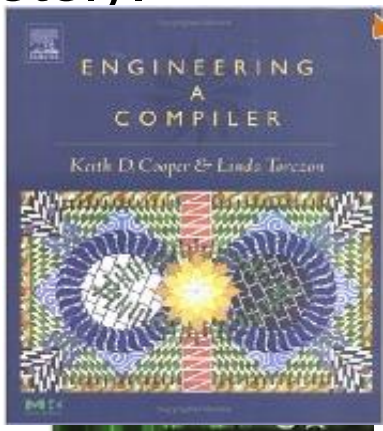
Literature

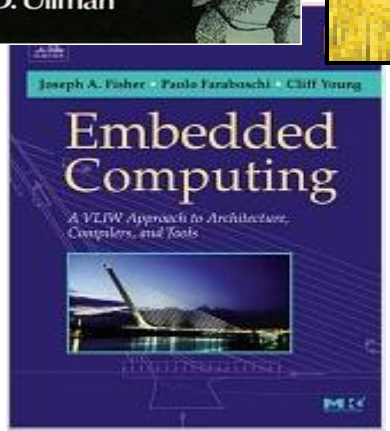
- No single book covers the course contents completely.
  - → Combine different book chapters and papers
- List on course homepage
- In the library

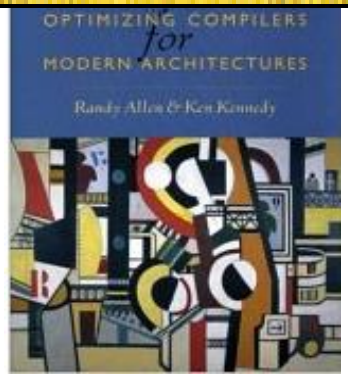






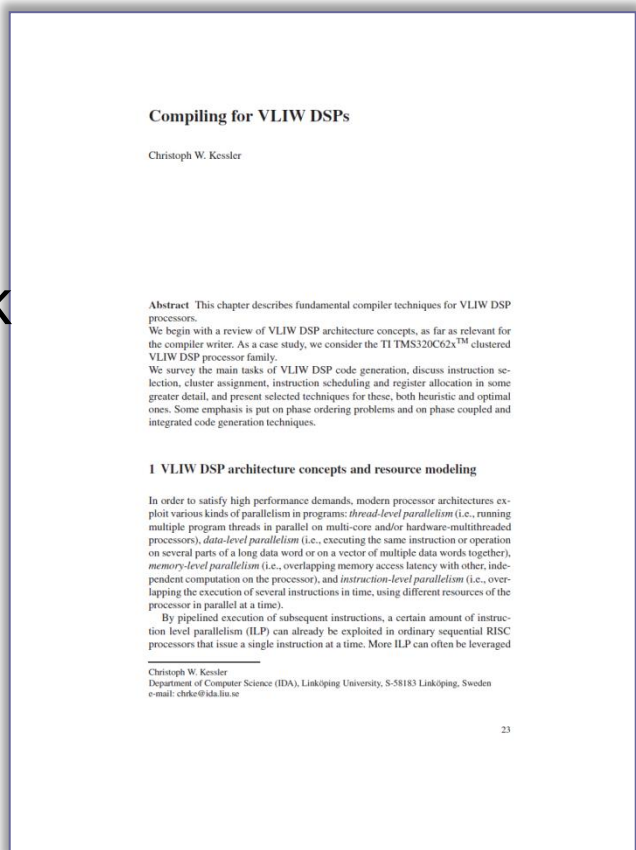






Literature (cont.)

- C. Kessler: Compiling for VLIW DSPs.
  Book chapter, in S. Bhattacharyya, E. Deprettere, R. Leupers and J. Takala, eds., *Handbook of Signal Processing Systems*, 3rd Edition, Springer, 2019. Ca. 41 pages.
  - Preprint handed out
  - Mandatory course literature for the code generation part
  - TekNat-Library has the complete book



Compiling for VLIW DSPs

Christoph W. Kessler

Abstract This chapter describes fundamental compiler techniques for VLIW DSP processors.
We begin with a review of VLIW DSP architecture concepts, as far as relevant for the compiler writer. As a case study, we consider the TI TMS320C62x™ clustered VLIW DSP processor family.
We survey the main tasks of VLIW DSP code generation, discuss instruction selection, cluster assignment, instruction scheduling and register allocation in some greater detail, and present selected techniques for these, both heuristic and optimal ones. Some emphasis is put on phase ordering problems and on phase coupled and integrated code generation techniques.

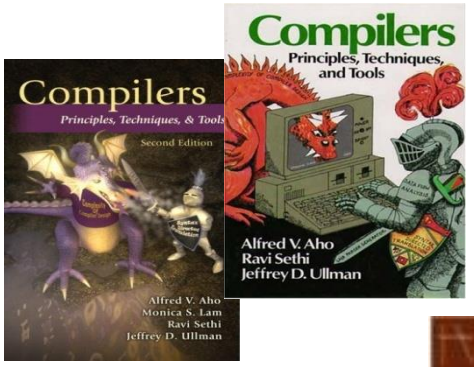
1 VLIW DSP architecture concepts and resource modeling

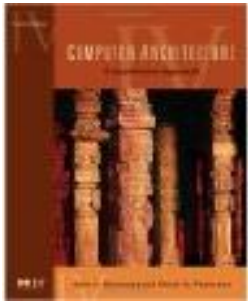
In order to satisfy high performance demands, modern processor architectures exploit various kinds of parallelism in programs: *thread-level parallelism* (i.e., running multiple program threads in parallel on multi-core and/or hardware-multithreaded processors), *data-level parallelism* (i.e., executing the same instruction or operation on several parts of a long data word or on a vector of multiple data words together), *memory-level parallelism* (i.e., overlapping memory access latency with other, independent computation on the processor), and *instruction-level parallelism* (i.e., overlapping the execution of several instructions in time, using different resources of the processor in parallel at a time).

By pipelined execution of subsequent instructions, a certain amount of instruction level parallelism (ILP) can already be exploited in ordinary sequential RISC processors that issue a single instruction at a time. More ILP can often be leveraged

Christoph W. Kessler
Department of Computer Science (IDA), Linköping University, S-58183 Linköping, Sweden
e-mail: chrke@ida.liu.se

Prerequisites





- A first course in **compiler construction**
  - TDDB44, TDDD55 or similar
  - or read the Dragon book in advance
- A course in **computer architecture**
  - Processor structure, pipelining, assembler language…
  - or read Hennessy/Patterson: *Computer Architecture*
  - or watch Onur Mutlu's *Computer architecture lectures* videos on youtube:
    https://www.youtube.com/watch?reload=9&v=c3mPdZA-Fmc&list=PL5Q2soXY2Zi9xidyIgBxUz7xRPS-wisBN
    https://www.youtube.com/@OnurMutluLectures/videos
- Background in **discrete maths**, **data structures** and **algorithms**
  - Graphs, trees; depth-first search; connected components; backtracking, dynamic programming, branch-and-bound,…
  - Integer linear programming
- Some repetition material available on course homepage