Minutes of the EuroLLVM 2016 "Compilers in Education" BoF
Introduction slides: https://docs.google.com/presentation/d/13aamtJ9SVuUKDv7eLpfmUs_O6Y6QY2un12Imdpzl_Fw/edit?usp=sharing

TU Eindhoven course overview:
Four assignments
1. Backend programming with AVR
Teaches compiler organization (AST, IR, MI), instruction selection (using DAG and patterns), scheduling, and register allocation
2. Optimization layer, writing passes
Organization of LLVM and basic analysis of IR. Also includes writing a basic list scheduling algorithm.
3. Code optimization and loop transformations
The effect of loop optimizations, kinds of optimizations possible, analyzed using an example image processing program. Teaches profiling with perf to understand the effects of transformations. Also covers auto vectorization and polyhedral optimizations.
4. Code generation for heterogeneous architectures
OpenMP4, CUDA, OpenCL. Tools, usage, and internal design issues for the compiler, includes run-time libraries and linker support.

Organized as 8 weeks, each with 2 lectures and 2 hours of assignment support + online forum for questions.

Cambridge course materials:

- http://compilerteaching.github.io

Cambridge course overview:
https://www.cl.cam.ac.uk/teaching/1516/L25/

David Cisnel's priorities
OpenCL type languages
Scripting languages like Javascript

Overview of issues to consider
* Most courses are based on flex/bison, which is now very little of what compilers do
* Students need to understand back-end transformation, code generation, register allocation but studying one back-end may vendor-lock them.
* Teaching multiple backends at a lower depth might help.
* Teaching weird architectures (heterogeneous), too
* Middle level optimisations may prove hard to re-run a course once they go upstream
* Finding new stuff every year may take it to a too advanced level for undergraduate, or even master courses
* Handing assignment grading is a pain and error prone when the complexity of the code moves away from the dragon book
* We need both practical and abstract approach (academia and industry need both at different levels)
* Harder courses will train better students, but may spook many good students
* The software engineering training is as important as the content, since compilers are generally very large and complex systems
* Proposing industry projects to students (mainly master) is an intersting avenue

Floor comments

Importance of nuts and bolts.Give students a toolbox
Want al courses to be like this
  David Chisnall will write the book - one
Want students to understand ramifications of SSA
More graph theory
More profiling
  In Cambridge this is part of OS
  TU Eindhoven teaches students the use and understanding of the perf tool
More modern loop analysis. E.g compute on the fly
Example course from floor
  Broad theory course
  Then write full C++ cimpiler so see why theory matters
Note value of students working on a very large prog.
Issue of vendor lock in
  Note Roel teaches multiple architectures
    More specifically, AVR, ARM, and x86, but others are used as examples

If you have a 40 hour project suitable for a student tell David