Historic Compilers

Copyright 2003, Keith D. Cooper, Ken Kennedy & Linda Torczon, all rights reserved. Students enrolled in Comp 412 at Rice University have explicit permission to make copies of these materials for their personal use.
1957: The FORTRAN Automatic Coding System

- Six passes in a fixed order
- Generated good code
  
  **Assumed unlimited index registers**
  
  **Code motion out of loops, with ifs and gotos**
  
  **Did flow analysis & register allocation**
1969: IBM’s FORTRAN H Compiler

- Used low-level IR (quads), identified loops with dominators
- Focused on optimizing loops (“inside out” order)
  
  **Passes are familiar today**
- Simple front end, simple back end for IBM 370
1975: BLISS-11 compiler (Wulf et al., CMU)

- The great compiler for the PDP-11
- Seven passes in a fixed order
- Focused on code shape & instruction selection

LexSynFlo did preliminary flow analysis
Final included a grab-bag of peephole optimizations
Classic Compilers

1980: IBM’s PL.8 Compiler

- Many passes, one front end, several back ends
- Collection of 10 or more passes
  - Repeat some passes and analyses
  - Represent complex operations at 2 levels
  - Below machine-level IR

Dead code elimination
Global cse
Code motion
Constant folding
Strength reduction
Value numbering
Dead store elimination
Code straightening
Trap elimination
Algebraic reassociation

Multi-level IR has become common wisdom
1986: HP’s PA-RISC Compiler

- Several front ends, an optimizer, and a back end
- Four fixed-order choices for optimization (9 passes)
- Coloring allocator, instruction scheduler, peephole optimizer
1999: The SUIF Compiler System

Another classically-built compiler
- 3 front ends, 3 back ends
- 18 passes, configurable order
- Two-level IR (High SUIF, Low SUIF)
- Intended as research infrastructure

Classic Compilers

Data dependence analysis
Scalar & array privatization
Reduction recognition
Pointer analysis
Affine loop transformations
Capturing object definitions
Virtual function call elimination
Garbage collection

SSA construction
Dead code elimination
Partial redundancy elimination
Constant propagation
Global value numbering
Strength reduction
Reassociation
Instruction scheduling
Register allocation
Classic Compilers

2000: The SGI Pro64 Compiler (now Open64 from Intel)

Open source optimizing compiler for IA 64

- 3 front ends, 1 back end
- Five-levels of IR
- Gradual lowering of abstraction level
Classic Compilers

2000: The SGI Pro64 Compiler (now Open64 from Intel)

Interprocedural
Classic analysis
Inlining (user & library code)
Cloning (constants & locality)
Dead function elimination
Dead variable elimination

Loop Nest Optimization
Dependence analysis
Parallelization
Loop transformations (fission, fusion, interchange, peeling, tiling, unroll & jam)
Array privatization

Global Optimization
SSA-based analysis & opt’n
Constant propagation, PRE,
OSR+LFTR, DVNT, DCE
(also used by other phases)

Code Generation
If conversion & predication
Code motion
Scheduling (inc. sw pipelining)
Allocation
Peephole optimization
Classic Compilers

Even a 2000 JIT fits the mold, albeit with fewer passes

- Front end tasks are handled elsewhere
- Few (if any) optimizations
  
  Avoid expensive analysis
  
  Emphasis on generating native code
  
 Compilation must be profitable