Optimal Register Allocation and Instruction Scheduling for LLVM

Roberto Castañeda Lozano – SICS, KTH

joint work with:
G. Hjort Blindell – KTH, SICS  M. Carlsson – SICS
F. Drejhammar – SICS  C. Schulte – KTH, SICS

This research has been partially funded by LM Ericsson AB and the Swedish Research Council (VR 621-2011-6229)
Code Generation in LLVM

... → PreRA
Code Generation in LLVM

... → PreRA → instruction scheduling
Code Generation in LLVM

... → PreRA → instruction scheduling → register allocation
Code Generation in LLVM

... → PreRA → instruction scheduling → register allocation → instruction scheduling
Code Generation in LLVM
Code Generation in LLVM

- PreRA
- Instruction Scheduling
- Register Allocation
- Instruction Scheduling
- PreEmit

Stages, heuristics

Pros: compilation speed
Cons: suboptimal, complex
Code Generation in LLVM

- PreRA
- Instruction scheduling
- Register allocation
- Instruction scheduling
- PreEmit

- Stages, heuristics
Code Generation in LLVM

- Stages, heuristics
- Pros: compilation speed
Code Generation in LLVM

- PreRA
- Instruction scheduling
- Register allocation
- Instruction scheduling
- PreEmit

- Stages, heuristics
- Pros: compilation speed
- Cons: suboptimal, complex
Introducing Unison

... → PreRA
Introducing Unison

PreRA

register allocation

instruction scheduling

Pros: simple, optimal
Cons: compilation slowdown
Introducing Unison

PreRA → register allocation

PreRA → instruction scheduling

unified combinatorial problem
Introducing Unison

minimize $\sum_{b \in B} \text{weight}(b) \times \text{cost}(b)$ subject to

\[ l_t \iff \exists p \in P : \left( \text{use}(p) \land y_p = t \right) \quad \forall t \in T \]

\[ a_{\text{definer}}(t) \iff l_t \quad \forall t \in T \]

\[ a_o \iff y_p \neq \bot \quad \forall o \in O, \forall p \in \text{operands}(o) \]

\[ a_o \iff i_o \neq \bot \quad \forall o \in O \]

\[ r_{yp} \in \text{class}(i_o, p) \quad \forall o \in O, \forall p \in \text{operands}(o) \]

\[ \text{disjoint2} \left( \{(r_t, r_t + \text{width}(t) \times l_t, l_{st}, l_{et}) : t \in T(b)\} \right) \quad \forall b \in B \]

\[ r_{yp} = r \quad \forall p \in P : p \triangleright r \]

\[ r_{yp} = r_{yq} \quad \forall p, q \in P : p \equiv q \]

\[ l_t \rightarrow l_{st} = c_{\text{definer}}(t) \quad \forall t \in T \]

\[ l_t \rightarrow l_{et} = \max_{o \in \text{users}(t)} c_o \quad \forall t \in T \]

\[ a_o \rightarrow c_o \geq c_{\text{definer}}(y_p) + \text{lat}(i_{\text{definer}}(y_p)) \quad \forall o \in O, \forall p \in \text{operands}(o) : \text{use}(p) \]

\[ \text{cumulative} \left( \{(c_o, \text{con}(i_o, r), \text{dur}(i_o, r)) : o \in O(b)\}, \text{cap}(r) \right) \quad \forall b \in B, \forall r \in R \]
Introducing Unison

... → PreRA → register allocation → instruction scheduling → unified combinatorial problem

Pros: simple, optimal
Cons: compilation slowdown
Introducing Unison

... → PreRA → register allocation → instruction scheduling → unified combinatorial problem → constraint solver
Introducing Unison

PreRA

- register allocation
- instruction scheduling

unified combinatorial problem

constraint solver

PreEmit

Pros: simple, optimal
Cons: compilation slowdown
Introducing Unison

- PreRA
- Register allocation
- Instruction scheduling
- Unified combinatorial problem
- Constraint solver
- PreEmit

Pros: simple, optimal
Cons: compilation slowdown
Introducing Unison

Integration, combinatorial optimization
Introducing Unison

- Integration, combinatorial optimization
- Pros: simple, optimal
Introducing Unison

- Integration, combinatorial optimization
- Pros: simple, optimal
- Cons: compilation slowdown
Introducing Unison

Integration, combinatorial optimization
Pros: simple, optimal
Cons: compilation slowdown

perfect complement to LLVM!
Speedup over LLVM 3.8

- 50 MediaBench functions
- Hexagon V4 processor
Speedup over LLVM 3.8

- 50 MediaBench functions
- Hexagon V4 processor
- Provably optimal (◼) for 54% of the functions
Speedup over LLVM 3.8

- 50 MediaBench functions
- Hexagon V4 processor
- Provably optimal (■) for 54% of the functions
- Compilation time: from seconds to minutes
Unison Is Practical and Effective

- Integrated
  - register allocation
  - instruction scheduling
Unison Is Practical and Effective

- Integrated
  - register allocation
  - instruction scheduling
- Simple, optimal, slower
Unison Is Practical and Effective

- Integrated
  - register allocation
  - instruction scheduling
- Simple, optimal, slower
- Complements LLVM:
  - traditional LLVM for compile/debug cycle
Unison Is Practical and Effective

- Integrated
  - register allocation
  - instruction scheduling

- Simple, optimal, slower

- Complements LLVM:
  - traditional LLVM for compile/debug cycle
  - LLVM + Unison for release builds
Unison Is Practical and Effective

- Integrated
  - register allocation
  - instruction scheduling
- Simple, optimal, slower
- Complements LLVM:
  - traditional LLVM for compile/debug cycle
  - LLVM + Unison for release builds
- Useful analysis tool for LLVM developers
  - how good is my heuristic?
Demo at CC2016 (12:20)

www.sics.se/~rcas/unison-demo