Register Allocation via Chordal Graph Coloring for a HOT Compiler

Daniel Lee and Bryan Mills
15-745 Optimizing Compilers, Spring 2006
Prof. Peter Lee
RA via Chordal Graph Coloring

- Register allocation requires coloring interference graphs for temps.
- In general, known to be NP-complete.
- There exist linear graph coloring algorithms for chordal graphs.
- One uses maximal cardinality search to find a simplicial elimination ordering.
- Recent result: programs in strict-SSA form have chordal interference graphs.
RA via Chordal Graph Coloring

- Our RA based on the Pereira-Palsberg algorithm:
  
built $\rightarrow$ MCS $\rightarrow$ greedy color $\rightarrow$ spill $\rightarrow$ transform

- We pay close attention to make sure the transform stage is done safely.
- This involved going into much more depth about spilling than Pereira and Palsberg did.
Continuation passing style is a common IR for compilers that support higher order functions. All control flow is modeled as jumping to a continuation. A continuation has no join points. CPS code is strict-SSA without $\phi$-nodes. So we get optimal graph coloring and avoid having to do SSA-elimination! Downside: $\phi$-nodes internalized as arguments thrown to continuations.
A HOT Compiler Backend

- Implemented in SML as a backend to a higher-order, typed compiler for ML.
- CPS conversion and closure conversion done a la “From System F to Typed Assembly Language”.
- Generating code for a simple TAL we designed.
- The simple TAL instruction set is RISC-like with a stack, parameterized on the number of registers.
- Fairly trivial to translate our TAL to a real architecture.
- Having our own simulator makes it easy to collect data.
Post Spilling

- When all temps cannot be colored with k-registers, some must be spilled.
- P-P heuristics spill until a subgraph fits into k-colors.
- They do not explain what to do about shuttling spilled values back and forth from memory.
- Simple solution: reserve registers (architecture dependent, usually 2) for shuttling data.
- Losing 2 registers is very bad! (esp on x86)
- Our solution: use better heuristics to make efficient use of shuttle registers.
Simple Spill Heuristics

- Base Spillers: Spillers that will color in k-2 colors and spill everything else, shuttle spill data with 2 shuttle registers.
  - spill-highest : heuristic
    - spill the highest used colors in the coloring
    - very simple to implement
  - spill-least-used : heuristic
    - spills the colors used least in the coloring
    - allocates the colors used used the most
Spill Heuristics as Combinators

- Spiller transformers: Takes a spiller and makes a new one. Functions of type heuristic -> heuristic.
- Goal of transformers: make heuristics that color to shuttle registers when safe.
- try-in-k : heuristic -> heuristic
  - tries input heuristic only if graph is not k colorable
Partition Coloring

- partition-optimize : heuristic -> heuristic
  - runs input heuristic
  - generates interference graph of spilled nodes
  - partitions IG into connected subgraphs
  - colors 2-colorable graphs with shuttle colors, spills rest

- According to our results, a good optimization in practice.
- More aggressive optimizations possible.
Spill Heuristic Transformers

- Useful for more than just building better spillers!
- coalesce : heuristic -> heuristic
  - greedy coalescing
  - runs input heuristic
  - coalesces the output
  - coalesces both registers and stack slots
- Example:
  coalesce (try-in-k (partition-optimize spill-least-used))