CSE 582 – Compilers

Register Allocation
Hal Perkins
Autumn 2002

Agenda

- Register allocation constraints
- Top-down and bottom-up local allocation
- Global allocation – register coloring

Credits: Adapted from slides by Keith Cooper, Rice University

k

- Intermediate code typically assumes infinite number of registers
- Real machine has k registers available
- Goals
  - Produce correct code that uses k or fewer registers
  - Minimize added loads and stores
  - Minimize space needed for spilled values
  - Do this efficiently – O(n), O(n log n), maybe O(n^2)

Register Allocation

- Task
  - At each point in the code, pick the values to keep in registers
  - Insert code to move values between registers and memory
    - No additional transformations – scheduling should have done its job
  - Minimize inserted code, both dynamically and statically

Allocation vs Assignment

- Allocation: deciding which values to keep in registers
- Assignment: choosing specific registers for values
- Compiler must do both

Basic Blocks

- A *basic block* is a maximal length segment of straight-line code (i.e., no branches)
- Significance
  - If any statement executes, they all execute
    - Barring exceptions or other unusual circumstances
  - Execution totally ordered
  - Many techniques for improving basic blocks – simplest and strongest methods

Local Register Allocation

- Transformation on basic blocks
- Produces decent register usage inside a block
  - Need to be careful of inefficiencies at boundaries between blocks
- Global register allocation can do better, but is more complex

Allocation Constraints

- Allocator typically won't allocate all registers to values
- Generally reserve some minimal set of registers F used only for spilling (i.e., don't dedicate to a particular value

Liveness

- A value is *live* between its *definition* and *use*.
  - Find definitions (x = …) and uses (… = … x …)
  - Live range is the interval from definition to last use
    - Can represent live range as an interval [i,j] in the block

Top-Down Allocator

- Idea
  - Keep busiest values in a dedicated registers
  - Use reserved set, F, for the rest
- Algorithm
  - Rank values by number of occurrences
  - Allocate first k-F values to registers
  - Add code to move other values between reserved registers and memory

Bottom-Up Allocator

- Idea
  - Focus on replacement rather than allocation
  - Keep values used "soon" in registers
- Algorithm
  - Start with empty register set
  - Load on demand
  - When no register available, free one
- Replacement
  - Spill value whose next use is farthest in the future
  - Prefer clean value to dirty value
  - Sound familiar?

Bottom-Up Allocator

- Invented about once per decade
  - Sheldon Best, 1955, for Fortran I
  - Laslo Belady, 1965, for analyzing paging algorithms
  - William Harrison, 1975, ECS compiler work
  - Chris Fraser, 1989, LCC compiler
  - Vincenzo Liberatore, 1997, Rutgers
- Will be reinvented again, no doubt
- Many arguments for optimality of this

Global Register Allocation

- A standard technique is *graph coloring*
- Use control and dataflow graphs to derive *interference graph*
  - Nodes are virtual registers (the infinite set)
  - Edge between (t1,t2) when t1 and t2 cannot be assigned to the same register
    - Most commonly, t1 and t2 are both live at the same time
    - Can also use to express constraints about registers, etc.
- Then color the nodes in the graph
  - Two nodes connected by an edge may not have same color
  - If more than k colors are needed, insert spill code

Coming Attractions

- Dataflow and Control flow analysis
- Overview of optimizations