Low-Level Issues

Last lecture
- Liveness analysis

Today
- Register allocation

Later
- More register allocation
- Instruction scheduling

Register Allocation

Problem
- Assign an unbounded number of symbolic registers to a fixed number of architectural registers
- Simultaneously live data must be assigned to different architectural registers

Goal
- Minimize overhead of accessing data
  - Memory operations (loads & stores)
  - Register moves

Scope of Register Allocation

Expression
Local
Loop
Global
Interprocedural

Granularity of Allocation

What is allocated to registers?
- Variables
- Live ranges/Web (i.e., du-chains with common uses)
- Values (i.e., definitions; same as variables with SSA)

What are the tradeoffs?
Each allocation unit is given a symbolic register name (e.g., s1, s2, etc.)
Global Register Allocation by Graph Coloring

Idea [Cocke 71], First allocator [Chaitin 81]
1. Construct interference graph $G=(N,E)$
   - Represents notion of “simultaneously live”
   - Nodes are units of allocation (e.g., variables, live ranges)
   - There is an edge $(n_i, n_j) \in E$ if $n_i$ and $n_j$ are simultaneously live
   - Symmetric (not reflexive nor transitive)
2. Find $k$-coloring of $G$ (for $k$ registers)
   - Adjacent nodes can’t have the same color
3. Allocate the same register to all allocation units of the same color
   - Adjacent nodes must be allocated to distinct registers

Interference Graph Example (Variables)

Computing the Interference Graph

Use results of live variable analysis

for each symbolic-register $s_i$ do
   for each symbolic-register $s_j$ (j < i) do
      for each def $\in \{\text{definitions of } s_i\}$ do
         if ($s_j$ is live at def) then
            $E \leftarrow E \cup \{s_i, s_j\}$

Options
- treat all instructions the same
- treat MOVE instructions special
- which is better?

Allocating Registers Using the Interference Graph

$k$-coloring
- Color graph nodes using up to $k$ colors
- Adjacent nodes must have different colors

Allocating to $k$ registers $\Rightarrow$ finding a $k$-coloring of the interference graph
- Adjacent nodes must be allocated to distinct registers

But...
- Optimal graph coloring is NP-complete
- Register allocation is NP-complete, too (must approximate)
- What if we can’t $k$-color a graph? (must spill)
Register Allocation: Spilling

If we can’t find a k-coloring of the interference graph
- Spill variables (nodes) until the graph is colorable

Choosing variables to spill
- Choose arbitrarily or
- Choose least frequently accessed variables
- Break ties by choosing nodes with the most conflicts in the interference graph
- Yes, these are heuristics!

Spilling (Original CFG and Interference Graph)

Spilling (After spilling b)

Simple Greedy Algorithm for Register Allocation

for each \( n \in N \) do  
  \{ select \( n \) in decreasing order of weight \}  
  if \( n \) can be colored then  
    do it  
    \{ reserve a register for \( n \) \}  
  else  
    Remove \( n \) (and its edges) from graph  
    \{ allocate \( n \) to stack (spill) \}

(After spilling b)
Example

Attempt to 3-color this graph ( , , )

Weighted order:

What if you use a different order?

Example

Attempt to 2-color this graph ( , )

Weighted order:

Improvement #1: Simplification Phase [Chaitin 81]

Idea

– Nodes with $< k$ neighbors are guaranteed colorable

Remove them from the graph first

– Reduces the degree of the remaining nodes

Must spill only when all remaining nodes have degree $\geq k$

Simplifying Graph Allocators
Algorithm [Chaitin81]

while interference graph not empty do
  while ∃ a node \( n \) with < \( k \) neighbors do
    \{ simplify \}
    Remove \( n \) from the graph
    Push \( n \) on a stack
  \{ spill \}
  if any nodes remain in the graph then
    Pick a node \( n \) to spill
    Add \( n \) to spill set
    Remove \( n \) from the graph
  \{ blocked with \( \geq k \) edges \}
  if spill set not empty then
    Insert spill code for all spilled nodes
    \{ store after def; load before use \}
    Reconstruct interference graph & start over
  \{ lowest spill-cost or \}
  while stack not empty do
    \{ highest degree \}
    Pop node \( n \) from stack
    Allocate \( n \) to a register
  \{ color or select \}

Example

Attempt to 3-color this graph ( )

More on Spilling

Chaitin’s algorithm restarts the whole process on spill
- Necessary, because spill code (loads/stores) uses registers
- Okay, because it usually only happens a couple times

Alternative
- Reserve 2-3 registers for spilling
- Don’t need to start over
- But have fewer registers to work with

Concepts

Register allocation
- scope of allocation
- granularity: what is being allocated to a register
- order that allocation units are visited in matters in all heuristic algorithms

Global approaches
- greedy coloring
- coloring with simplification
Next Time

Lecture
- More register allocation
  - Allocation across procedure calls