Register Allocation

• Given: k registers and code with n registers
• Goal: transform code to use only k registers
• For every instruction we will:
  - Determine which values are in registers
  - Select a register for each value
• Global register allocation is of course NP-complete
• Develop heuristics which minimize running time

Register Allocation

• Until register allocation we assume an infinite set of registers (aka "temps" or "pseudo-registers").
• But real machines have a fixed set of registers.
• The register allocator must assign each temp to a machine register.

Interference

• Consider two temps, t0 and t1.
• If the live ranges for t0 and t1 overlap, we say that they interfere.
• First rule of register allocation:
  - Temps with interfering live ranges may not be assigned to the same machine register.

General Plan

• Construct an interference graph
• Respect special registers
  - avoid reserved registers
  - Use registers properly
  - respect distinction between callee/caller save registers
• Map temps to registers
• Generate code to save & restore
• Deal with spills

Optimistic Graph Coloring

• Construct Interference Graph
  - Use liveness information
  - Each node in graph is a temp
  - \((u,v) \in G \iff u \& v \text{ can't be in the same hard register, i.e., they interfere}\)
• Color Graph
  - Assign to each node a color from a set of k colors, \(k = 1 \mid \text{register set} \mid\)
• Spill
  - If can't color graph with \(\leq k\) colors then spill some temps into memory. Regenerate asm code and start over.
An Example, k=4

\[
\begin{align*}
    v &\leftarrow 1 \\
w &\leftarrow v + 3 \\
x &\leftarrow w + v \\
u &\leftarrow v \\
t &\leftarrow u + x \\
w &\leftarrow t \\
t &\leftarrow u \\
\end{align*}
\]

Construct the interference graph

Compute live ranges

Voila, registers are assigned!

But, can we do better?

\[
\begin{align*}
a &\leftarrow x + y + z \\
b &\leftarrow a
\end{align*}
\]

u & v are special. They interfere, but only through a move!
Rewrite the code to coalesce u & v

Is Coalescing always good?

Was 2-colorable, now it needs 3 colors

So, we treat moves specially.

Interference from moves become "move edges."

Interference from moves become "move edges."

Construct the interference graph

Compute live ranges

Compute live ranges

Construct the interference graph

Was 2-colorable, now it needs 3 colors

So, we treat moves specially.
So, we need to spill

Spilling reduces live ranges, which decreases register pressure.

Recalculate interference graph

Recalculate live ranges

Recolor graph
So far

- Interference Graph
- Coalescing
- Coloring
- Spilling

Interference Graph Creation

- Use liveness information to determine if two temps have overlapping live ranges
- nodes in the interference graph represent temps or hard registers in IR
- \((u,v) \in G\) iff \(u\) and \(v\) interfere
- What does it mean for two temps to interfere? I.e., What edges should be included in \(G\)?
- What does it mean to be a node in \(G\)?

Meaning of interference

- Overlapping live ranges?

Nodes of Interference Graph

- One node for \(i\) or two?
  - for \((i=0; i<10; i++)\) {
    - 
  } for \((i=0; i<n; i++)\) {
    - SSA?
  }
  - Reaching Defs?

We really don't want the first register assignment to influence which register the second \(i\) is assigned to. How can we automate this?

Building Webs

- Use reaching defs to create du-chains
- Use du-chains to create "Webs"
- A Web is a collection of du-chains such that for any 2 du-chains in a web they have a stmt in common
Building Webs
- Use reachingdefs to create du-chains
- Use du-chains to create "webs"
- A Web is a collection of du-chains such that for any 2 du-chains in a web they have a stmt in common

Build the Interference Graph
- Compute liveness
- Compute webs
- Foreach instruction, i, that defines T
  - Determine Web, W, for T
  - Foreach live variable at i
    - Determine Web w, for i
    - Insert (W, w) in G

Web Creation Algorithm
Compute reaching definitions
\{(def0: i00, i01, ..., i0n), (def1: i10, i11, ..., i1n), ...\}
For each def: (d: i0, i1, ..., in)
if d is not marked
create new web, w.
Put d into bag, B
While B is not empty
  remove a def, e, from B
  make e part of W
  mark e
  foreach use of d, i, in (i0, i1, ..., in)
  make i part of W
  for each def, r, that reaches i, add r to B
Special Registers

- Which registers can be used?
  - Some registers have special uses.
    - Register 0 or 31 is often hardwired to contain 0.
    - Special registers to hold return address, stack pointer, frame pointer, global area, etc.
  - Reserved registers for operating system.
  - Typically, leaves about 20 or so registers for other general uses.
- Second rule of register allocation:
  - Temps should be assigned only to the non-reserved registers.

Register Usage Conventions

- Certain registers are used for specific purposes by standard calling convention.
  - 4-6 argument registers.
    - The first 4-6 arguments to procedures/functions are always passed in these registers.
  - ~8 callee-save registers.
    - These registers must be preserved across procedure calls. Thus, if a procedure wants to use a callee-save register, it must first save the old value and then restore it before returning.
    - The remainder are caller-save registers.
      - These are not preserved across procedure calls. Thus, a procedure is free to use them without saving first.
      - Includes the argument registers.

Creating the Interference Graph

- Create precolored nodes for hard registers.
- Foreach instruction, augment set of defines as necessary.
- Compute liveness.
- Compute webs.
- Foreach instruction, i, that defines T:
  - Foreach t ∈ T:
    - Determine Web, W, for t.
    - Foreach live variable, v, at i:
      - Determine Web w, for v.
      - Insert (W, w) in G.

K-coloring a graph

- Let's say we have a node, n, s.t. n° < k and let G' = G - (n), then
  - if G' can be k-colored, then G can be k-colored.
- Proof?
  - This suggests the following optimistic heuristic:
    - While |G| > 0
      - Choose some n with degree < k
      - Push n on stack
      - Remove n from G
    - While |S| > 0
      - Pop n from S
      - Color with a legal color.
Example: $K=3$.

**Alg not perfect**

What should we do when there is no node of degree < $k$?

---

**Where We Are**

- Build
- Simplify
- Potential Spill
- Select
- Actual Spill

**Coalescing**

Can $u$ & $v$ be coalesced? Should $u$ & $v$ be coalesced?

---

**Briggs**

- Can coalesce $a$ and $b$ if ($\#$ of neighbors of $ab$ with degree $\geq k) < k$
- Why?
  - Simplify removes all nodes with degree $< k$
  - $\#$ of remaining nodes $< k$
  - Thus, $ab$ can be simplified

**Preston**

- Can coalesce $a$ and $b$ if foreach neighbor of $t$ of $a$
  - $t$ interferes with $b$, or,
  - degree of $t + k$
- Why?
  - let $S$ be set of neighbors of $a$ with degree $< k$
  - If no coalescing, simplify removes all nodes in $S$, call that graph $G^1$
  - If we coalesce we can still remove all nodes in $S$, call that graph $G^2$
  - $G^2$ is a subgraph of $G^1$
Register Allocation

Why Two Methods?

- With Briggs one needs to look at all neighbors of a & b
- With Preston, only need to look at neighbors of a.
- We need to insert hard registers in graph and they will have LARGE adjacency lists.
  - So
    - Precolored nodes have infinite degree
    - No other precolored nodes in adj list
    - Use Preston if one of a & b is precolored
    - Use Briggs if both are temps

Where We Are

Build
Simplify
Coalesce
Potential Spill
Select
Actual Spill

Actually, one more step: Freeze

Spilling

- What should we spill?

Spilling

- What should we spill?
  - Something that will eliminate a lot of interference edges
  - Something that is used infrequently
  - Maybe something that is live across a lot of calls?
- One Heuristic:
  - spill cheapest Web
  - Cost = (\sum_{\text{def} \in W} 10^{\text{depth(def)}} + \sum_{\text{use} \in W} 10^{\text{depth(use)}})/\text{degree}

Setting Up For Better Spills

- We want vars not-live across procedures to be allocated to caller-save registers. Why?
- We want vars live across many procs to be in callee-save registers
- We want live ranges of precolored nodes to be short!
- We prefer to use callee-save registers last.
Register Allocation

Avoiding Callee-registers

- Move callee-reg to temp at start of proc
- Move it back at end of proc.
- What happens if there is no register pressure?
- What happens if there is a lot of register pressure?

entry: define r
        temp ← r
        ...
exit:  r ← temp
      use r

Allocating long-lived vars to callee-save registers

- CALL instruction “defines” all caller-save regs
  entry: define r_s
  t_1 ← r_s
  x ← ...
  call ...
  x ← x
  x ← x
  exit: r_s ← t_1
        use r_s

Keeping the Frame Size Down

- How do you allocate spilled vars?
- What about mov a, b where both a & b have been spilled?
- Use graph-coloring with aggressive coalescing!
- Use liveness info to create an interference graph of the spilled nodes
- Coalesce ALL non-interfering moves between spilled nodes
- Simply/Select
- Colors map to frame locations
- NB: Do this before rewriting the program so the moves are eliminated.