Lecture 6
Register Allocation

I. Introduction
II. Abstraction and the Problem
III. Algorithm

Reading: Chapter 8.8.4
Before next class: Chapter 10.1 - 10.2
I. Motivation

• **Problem**
  – Allocation of variables (pseudo-registers) to hardware registers in a procedure

• **Perhaps the most important optimization**
  – Directly reduces running time
    • memory access $\rightarrow$ register access
  – Useful for other optimizations
    • e.g. PRE assumes old values are kept in registers

• **General Lessons**
  – How to abstract a problem according to program/machine characteristics?
  – important & often-overlooked approach to NP-Complete problems
Goal

• Find an assignment for all pseudo-registers, if possible.
  – Not trying to minimize the number of registers used

• If there are not enough registers in the machine, choose registers to spill to memory
Example

A = ...
IF A goto L1

B = ...
  = A
D =
  = B + D

L1: C = ...
  = A
D =
  = C + D
II. An Abstraction for Allocation & Assignment

• **Intuitively**
  – Two pseudo-registers interfere if at some point in the program they cannot both occupy the same register.

• **Interference graph**: an undirected graph, where
  – nodes = pseudo-registers
  – there is an edge between two nodes if their corresponding pseudo-registers interfere

• **What is not represented**
  – The extent of the interference between uses of different variables
  – Where in the program is the interference

**Quiz:** Why is this a good representation?
Register Allocation and Coloring

• A graph is \textit{n-colorable} if:
  – every node in the graph can be colored with one of the \textit{n} colors such that two adjacent nodes do not have the same color.

• \textbf{Assigning \textit{n} register (without spilling) = Coloring with \textit{n} colors}
  – assign a node to a register (color) such that no two adjacent nodes are assigned same registers(colors)

• \textbf{Is spilling necessary? = Is the graph \textit{n}-colorable?}

• \textbf{To determine if a graph is \textit{n}-colorable is NP-complete, for \textit{n}>2}
  • Too expensive
  • Heuristics
Quick Notes on NP-Completeness

• NP = P?
  – P: Polynomial
  – NP: Non-deterministic Polynomial
    • Exponential time on deterministic machines
  – A famous open problem in theory (unsolved after much research)

• NP-complete problems
  – If any can be solved in polynomial time, then NP = P

• Proving a problem is NP-complete → License to use heuristics
III. Algorithm

Step 1. Build an interference graph
   a. refining notion of a node
   b. finding the edges

Step 2. Coloring Algorithm (NP-Complete Problem)
Step 1a. Nodes in an Interference Graph

Quiz: What is the new inference graph with the additional code?

Answer: Interference applies only to live variables
Step 1a. Nodes in an Interference Graph

Quiz: What is the new inference graph with the additional code?

Answer: Use different registers for same variables if possible.
Live Ranges and Merged Live Ranges

- **Motivation:** to create an interference graph that is easier to color
  - Eliminate interference in a variable’s “dead” zones.
  - Increase flexibility in allocation:
    - can allocate same variable to different registers
- A **live range** consists of a definition and all the points in a program (e.g. end of an instruction) in which that definition is live.
  - How to compute a live range? (homework)
- Two overlapping live ranges for the **same** variable must be merged
Example (Revisited)

\[
\begin{align*}
A &= \ldots \quad (A_1) \\
\text{IF } A \text{ go to } L1 \\
B &= \ldots \\
D &= (D_2) \\
D &= B + D \\
L1: C &= \ldots \\
D &= (D_1) \\
D &= D + C \\
A &= \ldots \quad (A_2) \\
D &= \ldots \\
= A \\
\end{align*}
\]

liveness: \{ \} \quad \text{reaching-def: } \{ \}

\[
\begin{align*}
\{ A \} &\quad \{ A_1 \} \\
\{ A, B \} &\quad \{ A_1, B \} \\
\{ B \} &\quad \{ A_1, B \} \\
\{ B, D \} &\quad \{ A_1, B, D_2 \} \\
\{ D \} &\quad \{ A_1, B, D_2 \} \\
\{ \} &\quad \{ A_2, B, C, D_1, D_2 \} \\
\end{align*}
\]

(Does not use A, B, C, or D.)
Merging Live Ranges

- **Merging definitions into equivalence classes**
  - Start by putting each definition in a different equivalence class
  - For each point in a program:
    - if (i) variable is live, and (ii) there are multiple reaching definitions for the variable, then:
      - merge the equivalence classes of all such definitions into one equivalence class

- **From now on, refer to merged live ranges simply as live ranges**

Given:

\[
\begin{align*}
A_1 & \text{ overlaps with } A_2 \\
A_3 & \text{ overlaps with } A_4 \\
A_1 & \text{ overlaps with } A_3
\end{align*}
\]

Quiz: How many merged live ranges are here?
Step 1b. Edges of Interference Graph

• Intuitively:
  – Two live ranges (necessarily of different variables) may interfere if they overlap at some point in the program.
  – Algorithm:
    • At each point in the program:
      – enter an edge for every pair of live ranges at that point.

An optimized definition & algorithm for edges:
  – Algorithm:
    • check for interference only at the starts of each merged live range
  – Faster
  – Better quality

Quiz: Is this correct?
Example 2

Quiz: How many registers do we need for A and B?

IF .. goto L1

A = ...

L1: B = ...

IF .. goto L2

... = A

L2: ... = B

Lesson: Watch out for corner cases! Make sure the algorithm is correct!
Algorithm

Step 1. Build an interference graph
   a. refining notion of a node
   b. finding the edges

Step 2. Coloring Algorithm (NP-Complete Problem)

Quiz: What would you do?
Observations

• **Reminder**: coloring for $n > 2$ is NP-complete

• **Observations:**
  
  – a node with degree $< n$ ⇒
    
    • can always color it successfully, given its neighbors’ colors

  – a node with degree $= n$ ⇒

  – a node with degree $> n$ ⇒
Coloring Algorithm

- **Algorithm:**
  - Iterate until stuck or done
    - Pick any node with degree < n
    - Remove the node and its edges from the graph
  - If done (no nodes left)
    - reverse process and add colors

- **Example (n = 3):**

![Diagram of a graph with nodes B, E, A, C, and D]

- **Note:** degree of a node may drop in iteration
- **Avoids making arbitrary decisions that make coloring fail**
What Does Coloring Accomplish?

- **Done:**
  - colorable, also obtained an assignment
- **Stuck:**
  - colorable or not?
- **Example:** $n = 2$

Even if the algorithm gets stuck, it does not mean that it is not colorable.
What if Coloring Fails?

- **Use heuristics to improve its chance of success and to spill code**

Build interference graph

Iterative until there are no nodes left
- If there exists a node $v$ with less than $n$ neighbors
  place $v$ on stack to register allocate
else
  $v =$ node chosen by heuristics
  (least frequently executed)
  place $v$ on stack to register allocate (mark as spilled)
remove $v$ and its edges from graph

While stack is not empty
  Remove $v$ from stack
  Reinsert $v$ and its edges into the graph
  Assign $v$ a color that differs from all its neighbors if possible
  (guaranteed to be possible only for nodes not marked as spilled)
Summary

• **Problems:**
  – Given n registers in a machine, is spilling avoided?
  – Find an assignment for all pseudo-registers, whenever possible.

• **Solution:**
  – **Abstraction:** an interference graph
    • nodes: live ranges
    • edges: presence of live range at time of definition
  – **Register Allocation and Assignment problems**
    • equivalent to n-colorability of interference graph
      ➔ **NP-complete**
  – **Heuristics** to find an assignment for n colors
    • successful: colorable, and finds assignment
    • not successful: colorability unknown & no assignment

• **General lessons:**
  – Problem abstraction depends on program/machine characteristics
  – Minimize making arbitrary decisions for NP-complete problems
  – Careful about corner cases