Lecture 13

Register Allocation: Coalescing

I. Motivation

II. Coalescing Overview

III. Algorithms:

• Simple & Safe Algorithm
• Briggs’ Algorithm
• George’s Algorithm
Review: Register Allocation without Spilling

• **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
Review: Coloring Heuristic

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

- Avoids making arbitrary decisions that make coloring fail (e.g., B, A, D different colors)
Review: Computing Live Ranges

Live Variables

Reaching Definitions

Variable \( v \) is live at point \( p \) if the value of \( v \) is used on some path starting at \( p \)

A = ... \( (A_1) \)
IF A goto L1

{A}  \{A\}  \{A_1\}
{A}  \{A\}  \{A_1\}

B = ... \( (B_1) \)
= A
D = ... \( (D_1) \)
= B + D

L1: C = ... \( (C_1) \)
= A
D = ... \( (D_2) \)
= D + C

{A}  \{A\}  \{A_1\}
{A}  \{A,C\}  \{A_1,C_1\}
{B}  \{B\}  \{B_1\}
{B}  \{B\}  \{B_1\}
{C}  \{C\}  \{C\}
{C}  \{C\}  \{C_1\}
{D}  \{D\}  \{D\}
{D}  \{D\}  \{D_1\}

Variable v is live at point p if the value of v is used on some path starting at p

A = 2 \( (A_2) \)

{A,D}  \{A,D\}  \{A_2,B_1,C_1,D_1,D_2\}
{D}  \{D\}  \{A_2,B_1,C_1,D_1,D_2\}

= A

ret D

Must merge

Overlapping live ranges for the same variable must be merged
Review: Register Allocation with Spilling

• **A pseudo-register is**
  – Colored successfully: allocated a hardware register
  – Not colored: left in memory

• **Objective function**
  – **Cost** of an uncolored node:
    • proportional to number of uses/definitions (dynamically)
    • one estimate = $(\# \text{ defs} \& \text{ uses}) \times 10^{\text{loop-nest-depth}}$
    • Objective: minimize sum of cost of uncolored nodes

• **Heuristics**
  – **Benefit** of spilling a pseudo-register:
    • increases colorability of pseudo-registers it interferes with
    • can approximate by its degree in interference graph
  – **Greedy heuristic**
    • spill the pseudo-register with lowest cost-to-benefit ratio, whenever spilling is necessary
Review: Live-Range Splitting

- **Observation**: spilling is absolutely necessary if
  - number of live ranges active at a program point > n

- **Apply live-range splitting before coloring**
  - Identify a point where number of live ranges > n
  - Among those live ranges, choose the one with the largest inactive region
  - Split the inactive region from the live range
  - Repeat as needed

Split & spill x, then can color rest

Spill cost? 2

\[ x = \]
\[ i = i + 1 \]
\[ j = j + 1 \]
\[ k = k + 1 \]
\[ = x \]
I. Register Coalescing Motivation: Copy Instructions

- Two optimizations that help optimize away copy instructions:
  - Copy Propagation
  - Dead Code Elimination

- Can all copy instructions be eliminated using this pair of optimizations?

\[
\begin{align*}
X &= A + B \\
& \quad \ldots \\
Y &= X \\
& \quad \ldots \\
Z &= Y + 4
\end{align*}
\]

\[
\begin{align*}
X &= A + B \\
& \quad \ldots \\
// deleted \\
& \quad \ldots \\
Z &= X + 4
\end{align*}
\]
Example Where Copy Propagation Fails

- Use of copy target has multiple (conflicting) reaching definitions

\[
\begin{align*}
X &= A + B \\
Y &= C \\
Z &= Y + 4 \\
Y &= X
\end{align*}
\]
Another Example Where the Copy Instruction Remains

- Copy target (Y) still live even after some successful copy propagations

- **Bottom line**: copy instructions may still exist at the time register allocation is performed
II. Coalescing: Overview

• What clever thing might the register allocator do for copy instructions?

...  
Y = X  
...  

...  
r7 = r7  
...  

• If we can assign both the source and target of the copy to the same register:
  – then we don’t need to perform the copy instruction at all!
  – the copy instruction can be removed from the code
    • even though the optimizer was unable to do this earlier
• One way to do this:
  – treat the copy source and target as the same node in the interference graph
    • then the coloring algorithm will naturally assign them to the same register
  – this is called “coalescing”
Simple Example: Without Coalescing

- **Without coalescing**, X and Y can end up in different registers
  - cannot eliminate the copy instruction

\[
\begin{align*}
X &= \ldots \\
A &= 5 \\
Y &= X \\
B &= A + 2 \\
Z &= Y + B \\
\text{return } Z
\end{align*}
\]

Valid coloring with 3 registers
Example Revisited: With Coalescing

• **With coalescing**, \(X\) and \(Y\) are now guaranteed to end up in the same register
  – the copy instruction can now be eliminated

• **Great! So should we go ahead and do this for every copy instruction?**
Should We Coalesce $X$ and $Y$ In This Case?

It is legal to coalesce $X$ and $Y$ for a "$Y = X$" copy instruction if:

- the live ranges of $X$ and $Y$ do not overlap

But just because it is legal doesn’t mean that it is a good idea…

No! That would result in incorrect behavior if this branch is taken.

$X = A + B$

$Y = X$

$Z = Y + X$

$X = 2$
Why Coalescing May Be Undesirable, Even If Legal

\[
\begin{align*}
X &= A + B \\
\ldots &// 100 \text{ instructions} \\
Y &= X &// \text{last use of } X \\
\ldots &// 100 \text{ instructions} \\
Z &= Y + 4
\end{align*}
\]

• What is the likely impact of coalescing \(X\) and \(Y\) on:
  – live range size(s)?
    • recall our discussion of live range splitting
  – colorability of the interference graph?
• Fundamentally, coalescing adds further constraints to the coloring problem
  – doesn’t make coloring easier; may make it more difficult
• If we coalesce in this case, we may:
  – save a copy instruction, BUT
  – cause significant spilling overhead if we can no longer color the graph
Legal to Coalesce $X$ and $Y$?

- It is legal to coalesce $X$ and $Y$ for a "$Y = X$" copy instruction if:
  - the live ranges of $X$ and $Y$ do not overlap

\[
\begin{align*}
X &= A + B \\
X &= 2 \\
Y &= X \\
Z &= Y + 2 \\
\end{align*}
\]

Not by our (conservative) rule: live ranges overlap

But actually would be ok in this case to use same register for $X$ and $Y$
When to Coalesce

• Goal when coalescing is legal:
  – coalesce *unless* it would make a colorable graph non-colorable
• The bad news:
  – predicting colorability is tricky!
    • it depends on the shape of the graph
    • graph coloring is NP-hard
• Example: assuming 2 registers, should we coalesce X and Y?

2-colorable

Not 2-colorable
Representing Coalescing Candidates in the Interference Graph

- To decide whether to coalesce, we augment the interference graph.
- Coalescing candidates are represented by a new type of interference graph edge:
  - dotted lines: coalescing candidates
    - *try* to assign vertices the same color
    - (unless that is problematic, in which case they can be given different colors)
  - solid lines: interference (i.e., live ranges overlap)
    - vertices *must* be assigned different colors

```
X = ...
A = 5
Y = X
B = A + 2
Z = Y + B
return Z
```
How Do We Know When Coalescing Will Not Cause Spilling?

- **Key insight:**
  - Recall from the coloring algorithm:
    - we can always successfully $N$-color a node if its degree is $< N$

- To ensure that coalescing does not cause spilling:
  - check that the degree $< N$ invariant is still locally preserved after coalescing
    - if so, then coalescing won’t cause the graph to become non-colorable

- **Note:**
  - We do NOT need to determine whether the full graph is colorable or not
  - Just need to check that coalescing does not cause a colorable graph to become non-colorable
III. Algorithms

- Simple and Safe Algorithm
- Briggs’ Algorithm
- George’s Algorithm
Simple and Safe Coalescing Algorithm

- We can safely coalesce nodes \( X \) and \( Y \) with a coalescing edge if \((|X| + |Y|) < N\)
  - Note: \(|X| = \) degree of node \( X \) counting only interference (not coalescing) edges

- Example:
  \[
  (|X| + |Y|) = (1 + 2) = 3
  \]
  Degree of coalesced node can be no larger than 3

- if \( N \geq 4 \), it would always be safe to coalesce these two nodes
  - this cannot cause new spilling that would not have occurred with the original graph
- if \( N < 4 \), it is unclear

*How can we (safely) be more aggressive than this?*
What About This Example?

- Assume $N = 3$
- Is it safe to coalesce $X$ and $Y$?

![Diagram showing nodes A, B, X, Y, and Z with edges connecting them.]

$(|X| + |Y|) = (1 + 2) = 3$

\[(Not \ less \ than \ N)\]

- **Note**: $X$ and $Y$ share a common (interference) neighbor: node $A$
  - hence the degree of the coalesced $X/Y$ node is actually 2 (not 3)
  - therefore coalescing $X$ and $Y$ is guaranteed to be safe when $N = 3$
- How can we adjust the algorithm to capture this?
Another Helpful Insight

- Colors are not assigned until nodes are popped off the stack
  - nodes with degree \(< N\) are pushed on the stack first
  - when a node is popped off the stack, we know that it can be colored
    - because the number of potentially conflicting neighbors must be \(< N\)
- Spilling only occurs if there is no node with degree \(< N\) to push on the stack

- Example: \((N=2)\)

\[
\begin{array}{cccccc}
X & A & B & C & D & E \\
Y & J & I & H & G & F \\
\end{array}
\]

\(|X| = 5 \\
|Y| = 5 \\
2\text{-colorable after coalescing } X \text{ and } Y? \\
Yes: X/Y \text{ gets 1 color, A-J get 1 color}

Carnegie Mellon
Building on This Insight

- When would coalescing cause the stack pushing (aka “simplification”) to get stuck?
  1. coalesced node must have a degree $\geq N$
     - otherwise, it can be pushed on the stack, and we are not stuck
  2. AND it must have at least $N$ neighbors that each have a degree $\geq N$
     - otherwise, all neighbors with degree $< N$ can be pushed before this node
       - reducing this node’s degree below $N$ (and therefore we aren’t stuck)

- To coalesce more aggressively (and safely), let’s exploit this second requirement
  - which involves looking at the degree of a coalescing candidate’s neighbors
    - not just the degree of the coalescing candidates themselves
Briggs’ Algorithm

• Nodes \( X \) and \( Y \) (with a coalescing edge) can be coalesced if:
  – (number of neighbors of \( X/Y \) with degree \( \geq N \)) \( < N \)

• Works because:
  – all other neighbors can be pushed on the stack before this node,
  – and then its degree is \( < N \), so then it can be pushed

• Example: \( (N = 2) \)
Briggs’ Algorithm

• **Nodes X and Y can be coalesced if:**
  – \((\text{number of neighbors of } X/Y \text{ with degree } \geq N) < N\)

• **More extreme example: \((N = 2)\)**

![Graph showing nodes X and Y with connected neighbors]

<table>
<thead>
<tr>
<th>X/Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td>H</td>
</tr>
<tr>
<td>G</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>D</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>A</td>
</tr>
</tbody>
</table>
George’s Algorithm

Motivation:
• imagine that $X$ has a very high degree, but $Y$ has a much smaller degree
  – (perhaps because $X$ has a large live range)

• With Briggs’ algorithm, we would inspect all neighbors of both $X$ and $Y$
  – but $X$ has a lot of neighbors!

• Can we get away with just inspecting the neighbors of $Y$?
  – while showing that coalescing makes coloring no worse than it was given $X$?
George’s Algorithm

• Coalescing $X$ and $Y$ does no harm if:
  – foreach neighbor $T$ of $Y$, either:
    1. degree of $T$ is $< N$, or $\leftarrow$ similar to Briggs: $T$ will be pushed before $X/Y$
    2. $T$ interferes with $X$ $\leftarrow$ hence no change compared with coloring $X$

• Example: ($N=2$)
Summary

• *Coalescing* can enable register allocation to *eliminate copy instructions*
  – if both source and target of copy can be allocated to the same register
• However, coalescing must be applied with care to *avoid causing register spilling*
• Augment the interference graph:
  – dotted lines for coalescing candidate edges
  – try to allocate to same register, unless this may cause spilling
• **Three Coalescing Algorithms:**
  – Simplest: based solely on *degree of coalescing candidate nodes* (X and Y)
  – Briggs’ algorithm
    • look at degree of neighboring nodes of X and Y
  – George’s algorithm
    • asymmetrical: *look at neighbors of lower degree node Y*
      (examine degree and interference with X)
Today’s Class

I. Motivation

II. Coalescing Overview

III. Algorithms:
   - Simple & Safe Algorithm
   - Briggs’ Algorithm
   - George’s Algorithm

Friday’s Class

• Discussion of Assignment 1 and 2 homework problems

No Class on Monday