Lecture 16
Register Allocation:
Coalescing
Let’s Focus on Copy Instructions

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

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

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

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

- Use of copy target has multiple (conflicting) reaching definitions
Another Example Where the Copy Instruction Remains

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

- Copy target (Y) still live even after some successful copy propagations
- **Bottom line:**
  - copy instructions may still exist when we perform register allocation
Copy Instructions and Register Allocation

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

![Image of code snippet]

• 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

```plaintext
X = ...;
A = 5;
Y = X;
B = A + 2;
Z = Y + B;
return Z;
```
Example Revisited: With Coalescing

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

Valid coloring with 3 registers

- 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 iff:
  - initial definition of \( Y \)’s live range is this copy instruction, AND
  - the live ranges of \( X \) and \( Y \) do not interfere otherwise

- 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.

\[
\begin{align*}
X &= A + B; \\
Y &= X; \\
Z &= Y + X; \\
X &= 2;
\end{align*}
\]
Why Coalescing May Be Undesirable, Even If Legal

\[
X = A + B; \\
\ldots \quad // 100 \text{ instructions} \\
Y = X; \\
\ldots \quad // 100 \text{ instructions} \\
Z = Y + 4; \\
\]

• 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
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 graph](image1)
- ![Not 2-colorable graph](image2)
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
    - vertices must be assigned different colors

\[
\begin{align*}
X &= \ldots; \\
A &= 5; \\
Y &= X; \\
B &= A + 2; \\
Z &= Y + B; \\
return Z;
\end{align*}
\]
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
  – no need to inspect the entire interference graph, or do trial-and-error

• **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
Simple and Safe Coalescing Algorithm

- We can safely coalesce nodes $X$ and $Y$ if $(|X| + |Y|) < N$
  - Note: $|X|$ = degree of node $X$ counting 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 \( Y \)?

\[
(|X| + |Y|) = (1 + 2) = 3
\]

(Not less than \( N \))

- Notice: \( X \) 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)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>J</td>
</tr>
<tr>
<td>B</td>
<td>I</td>
</tr>
<tr>
<td>C</td>
<td>H</td>
</tr>
<tr>
<td>D</td>
<td>G</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$|X| = 5$

$|Y| = 5$

2-colorable after coalescing X and Y?
Building on This Insight

• When would coalescing cause the stack pushing (aka “simplification”) to get stuck?
  1. coalesced node must have a degree >= 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 >= 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’s Algorithm

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

• 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: \((N = 2)\)

![Diagram showing coalescing of nodes X and Y, with example](image.png)
Briggs’s Algorithm

• Nodes X and Y can be coalesced if:
  – (number of neighbors of X/Y with degree >= N) < N

• More extreme example: (N = 2)
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’s algorithm, we would inspect all neighbors both \( X \) and \( Y \)
  – but \( X \) has a lot of neighbors!

• Can we get away with just inspecting the neighbors of \( Y \)?
  – 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
• **Coalescing Algorithms**:
  – simply based upon *degree of coalescing candidate nodes* (*X* and *Y*)
  – Briggs’s algorithm
    • look at *degree of neighboring nodes of X and Y*
  – George’s algorithm
    • asymmetrical: *look at neighbors of Y* (degree and interference with *X*)