Translation Out of SSA Form

SSA Deconstruction

At some point, we need executable code
• Do machines implement $\phi$-operations?
• Need to fix up the flow of values

Original idea [CFRWZ, 110]
• Replace $\phi$-function with copies in predecessor blocks
• Simple algorithm
  ♦ Works in most cases
• Adds lots of copies
  ♦ Most of them coalesce away
  ♦ Copy-coalescing is well understood

→ See, for example, Chaitin-Briggs GCRA [75,56]
or Budimlic et al. in PLDI 2002
Translation Out of SSA Form

Two “classic” problems arise in SSA deconstruction

• Lost-copy problem
• Swap problem

In each case, simple copy insertion produces incorrect code

Critical Observation

• Both “problems” are caused by transformations that rewrite the code and move definitions and uses
• Some of the complication arises from the shift between the parallel semantics of the $\Phi$–functions and the sequential semantics of copy

These problems were identified by Cliff Click and first published by Briggs et al. in 1997 [50]. They presented ad-hoc ways of solving the problems. (Click is not an author; he found the problems.)

This lecture is based on the work of Boissinot et al. and presents a more systematic approach to solving the problems inherent in translation out of SSA form. See “Revisiting Out-of-SSA Translation for Correctness, Code Quality, and Efficiency,” by Benoit Boissinot, Alain Darte, Benoit Dupont de Dinechin, Christophe Guillon, and Fabrice Rastello in CGO 2009.
The assignment to $z$ now receives the wrong value.

The Lost Copy Problem

In optimization, a critical edge in a control-flow graph is one that runs from a node with multiple successors to a node with multiple predecessors.

In out-of-SSA translation, critical edges create serious problems for copy insertion.

Copy folding is a simple transformation that creates the problem. Other transformations can have the same effect.

To fix this problem, the compiler either needs to create a temporary name to hold the penultimate value of $i$, or to insert the copy in the loop-closing branch.
This problem arises when a $\Phi$-function argument is defined by a $\Phi$-function in the same block. To generate correct code, the compiler will need to insert one or more additional copy operations and temporary names.

*
Translation Out of SSA Form

The Big Picture

• Swap problem & lost-copy problem arise from messing with $\Phi$-function parameters
  ♦ Renaming $\Phi$-function parameters, moving their definitions, ...
  ♦ Underlying issue is the parallel semantics of $\Phi$-function evaluation

• One way to simplify out-of-SSA translation is to separate the parallelism from the $\Phi$-functions and tackle it directly
  ♦ Convert to a form of SSA where eliminating the subscripts produces correct code
  ♦ We call that form “Conventional SSA” or CSSA
Translation Out of SSA Form

The Big Picture

- Insert parallel copies to convert SSA to conventional SSA
  - In CSSA, we can just drop the subscripts on SSA name
  - Introduces a new set of names
- Rename out of CSSA by replacing introduced names
- Eliminate \( \Phi \) functions as in original paper
  - Insert copies at end of the predecessor blocks
- Sequentialize parallel copies (may introduce new temporaries)
- Aggressive copy coalescing to remove copies
  - Can coalesce copies before renaming or after we are done
  - Coalescing parallel copies requires some care
  - May be easier, and clearer, to coalesce after sequentialization

Moves the issues created by parallel \( \Phi \) function semantics back into pred. blocks (in parallel copies).

The Individual Steps

Convert SSA to CSSA

For a $\phi$-function $a_0 \leftarrow \phi(a_1, a_2, \ldots, a_n)$:
1. Insert a parallel copy $a_1' \leftarrow a_1$ at the end of the block corresponding to $a_1$
2. Replace $a_0 \leftarrow \phi(a_1, a_2, \ldots, a_n)$ with $a_0' \leftarrow \phi(a_1', a_2', \ldots, a_n')$
3. Insert a parallel copy $a_0 \leftarrow a_0'$ after the $\phi$-function

Rename Out Of CSSA

$\forall$ primed name, $i_j'$, drop the subscripts and replace each $i'$ with a new name

Remove $\Phi$ Functions

- Replace $\Phi$-functions with copies in predecessor blocks as in CFRWZ paper
  ♦ Use parallel copies for good measure

Serialize The Parallel Copies

- Build a dependence graph and break cycles with a new name

Copies inserted for different $\phi$-functions in the same block form a “parallel copy group”. 

COMP 506, Rice University
Sequentializing Parallel Copies

A parallel copy group forms a graph

```
        a ←₄ b; b ←₄ c; c ←₄ a; d ←₄ a
```

Parallel copies

Corresponding graph

Graph is either a tree or it has a cycle

- Schedule a tree, bottom up from the leaves
- Must break each cycle with an extra copy
  - May require a new name
  - Other copies may avoid the extra name
    → If possible, break on the value preserved in a non-cycle copy

```
d ←₄ a  
   a ←₄ b 
   b ←₄ c 
   c ←₄ d 
```

Serialized copies

Subscripts on the copies indicate parallel copy group membership.
The assignment to $z$ now receives the wrong value.

**Reminder from earlier slide**

**Translation Out of SSA Form**

**The Lost Copy Problem**

Original code

- $i \leftarrow 1$
- $y \leftarrow i$
- $i \leftarrow i + 1$
- $d \leftarrow y + ...$

In SSA form

- $i_0 \leftarrow 1$
- $i_1 \leftarrow \Phi(i_0, i_2)$
- $y_0 \leftarrow i_1$
- $i_2 \leftarrow i_1 + 1$
- $z_0 \leftarrow y_0 + ...$

With copies folded

- $i_0 \leftarrow 1$
- $i_1 \leftarrow \Phi(i_0, i_2)$
- $i_2 \leftarrow i_1 + 1$
- $z_0 \leftarrow i_1 + ...$

Copies naively inserted

- $i_0 \leftarrow 1$
- $i_1 \leftarrow i_0$
- $i_2 \leftarrow i_1 + 1$
- $i_1 \leftarrow i_2$
- $z_0 \leftarrow i_1 + ...$

Copy folding is a simple transformation that creates the problem. Other transformations can have the same effect.

To fix this problem, the compiler needs to create a temporary name to hold the penultimate value of $i$. 

The assignment to $z$ now receives the wrong value.
The Lost Copy Problem via CSSA

Original code

\[ i \leftarrow 1 \]
\[ y \leftarrow i \]
\[ i \leftarrow i + 1 \]
\[ z \leftarrow y + \ldots \]

In SSA with copies folded

\[ i_0 \leftarrow 1 \]
\[ i_1 \leftarrow \Phi(i_0, i_2) \]
\[ i_2 \leftarrow i_1 + 1 \]
\[ z_0 \leftarrow i_1 + \ldots \]

Insert parallel copies to create CSSA

\[ i_0' \leftarrow i_0 \]
\[ i_1' \leftarrow \Phi(i_0', i_2') \]
\[ i_1 \leftarrow i_1' \]
\[ i_2 \leftarrow i_1 + 1 \]
\[ z_0 \leftarrow i_1 + \ldots \]

Rename out of CSSA

\[ i_0 \leftarrow 1 \]
\[ x \leftarrow i_0 \]
\[ i_1 \leftarrow x \]
\[ i_2 \leftarrow i_1 + 1 \]
\[ x \leftarrow i_2 \]

With only one \( \Phi \), parallel is a singleton.

Each use of \( i' \) replaced with \( x \). Use of \( i_1 \) to compute \( z_0 \) is correct.
The Lost Copy Problem via CSSA

\[
\begin{align*}
i_0 &\leftarrow 1 \\
x &\leftarrow i_0 \\
x &\leftarrow \Phi(x, x) \\
i_1 &\leftarrow x \\
i_2 &\leftarrow i_1 + 1 \\
x &\leftarrow i_2 \\
z_0 &\leftarrow i_1 + \ldots
\end{align*}
\]

From previous slide

\[
\begin{align*}
i_0 &\leftarrow 1 \\
x &\leftarrow i_0 \\
x &\leftarrow x \\
i_1 &\leftarrow x \\
i_2 &\leftarrow i_1 + 1 \\
x &\leftarrow i_2 \\
z_0 &\leftarrow i_1 + \ldots
\end{align*}
\]

Replace \(\Phi\)'s with copies

\[
\begin{align*}
x &\leftarrow 1 \\
i_1 &\leftarrow x \\
x &\leftarrow i_1 + 1 \\
z_0 &\leftarrow i_1 + \ldots
\end{align*}
\]

After coalescing

\[
\begin{align*}
i &\leftarrow 1 \\
y &\leftarrow i \\
i &\leftarrow i + 1 \\
z_0 &\leftarrow y + \ldots
\end{align*}
\]

The original code

Copies look rather stupid because it is a simple case, but it is correct.
This problem arises when a $\Phi$-function argument is defined by a $\Phi$-function in the same block. To generate correct code, the compiler will need to insert one or more additional copy operations and temporary names.
The Swap Problem

Starting Point
(code in SSA form)

Convert to CSSA

Rename primed variables

Eliminate $\Phi$ functions
(used parallel copies)

Subscript on $\leftarrow$ indicates a parallel copy group
Parallel copy groups 1, 2, 3, & 4 have acyclic dependence graphs. They can be scheduled in either order.

Groups 1, 2, 3, & 4 sequentialized
TRANSLATION OUT OF SSA FORM

The Swap Problem

Must break this cycle with an extra copy & a new name.

Groups 1, 2, 3, & 4 sequentialized

All copies are now sequential

The Swap Problem

Cyclic dependence graph for parallel copy group 5

These copies coalesce away

These copies are useless

These copies are LIVE because of next group

These copies are useless

Groups 1, 2, 3, & 4 sequentialized
Translation Out of SSA Form

**To recap, the algorithm is**

- Insert parallel copies to convert SSA to conventional SSA
  - In CSSA, we can just drop the subscripts on SSA name
  - Introduces a new set of names
- Rename out of CSSA by replacing introduced names
- Eliminate $\Phi$ functions by inserting parallel copies in prior blocks
  - May look redundant, but is necessary to handle the swap problem
- Sequentialize parallel copies \((\text{may introduce new temporaries})\)
- Aggressive copy coalescing to remove copies
  - Can coalesce copies before renaming or after we are done
  - Coalescing parallel copies requires some care
  - May be easier, and clearer, to coalesce after sequentialization

*e.g., Chaitin-Briggs unrestricted coalescing or Budimlic et. al. PLDI 2002.*
Using SSA Form in a Compiler

Need to translate source or IR into SSA form
• Algorithm from earlier lecture [110,50]
• IR needs to represent both $\Phi$–functions and $\Phi$–argument to CFG edge correspondance

Working with SSA Form
• The SSA name space makes some kinds of transformations harder
  ♦ Code motion past $\Phi$-functions is problematic
• SSA provides a useful sparse representation that can lead to efficiency
  ♦ Wegman-Zadeck SCP and SCCP as examples [347]

Need to translate out of SSA form
• Algorithms from this lecture
• LLVM comes out of SSA in code generator (selection, allocation, scheduling)