SSA-Form Register Allocation
Foundations

Sebastian Hack

Compiler Construction Course
Winter Term 2009/2010
Overview

1 Graph Theory
   - Perfect Graphs
   - Chordal Graphs

2 SSA Form
   - Dominance
   - $\phi$-functions

3 Interference Graphs
   - Non-SSA Interference Graphs
   - Perfect Elimination Orders
   - Chordal Graphs

4 Interference Graphs of SSA-form Programs
   - Dominance and Liveness
   - Dominance and Interference
   - Spilling
   - Implementing $\phi$-functions

5 Intuition
Overview

1. Graph Theory
   - Perfect Graphs
   - Chordal Graphs

2. SSA Form
   - Dominance
   - $\phi$-functions

3. Interference Graphs
   - Non-SSA Interference Graphs
   - Perfect Elimination Orders
   - Chordal Graphs

4. Interference Graphs of SSA-form Programs
   - Dominance and Liveness
   - Dominance and Interference
   - Spilling
   - Implementing $\phi$-functions

5. Intuition
Complete Graphs and Cycles

Complete Graph $K^5$

Cycle $C^5$
Induced Subgraphs

Graph with a $C^4$ subgraph

Graph with a $C^4$ induced subgraph
Induced Subgraphs

Graph with a $C^4$ subgraph

Graph with a $C^4$ induced subgraph

Note

Induced complete graphs are called cliques
Clique number and Chromatic number

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega(G)$ Size of the largest clique in $G$</td>
</tr>
<tr>
<td>$\chi(G)$ Number of colors in a minimum coloring of $G$</td>
</tr>
</tbody>
</table>
Clique number and Chromatic number

**Definition**

\[ \omega(G) \text{ Size of the largest clique in } G \]

\[ \chi(G) \text{ Number of colors in a minimum coloring of } G \]

**Corollary**

\[ \omega(G) \leq \chi(G) \text{ holds for each graph } G \]
Clique number and Chromatic number

Definition

\( \omega(G) \) Size of the largest clique in \( G \)

\( \chi(G) \) Number of colors in a minimum coloring of \( G \)

Corollary

\( \omega(G) \leq \chi(G) \) holds for each graph \( G \)

<table>
<thead>
<tr>
<th>( \omega(G) )</th>
<th>( \chi(G) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Perfect Graphs

**Definition**

$G$ is perfect $\iff \chi(H) = \omega(H)$ for each induced subgraph $H$ of $G$
Perfect Graphs

Definition

$G$ is perfect $\iff \chi(H) = \omega(H)$ for each induced subgraph $H$ of $G$
Perfect Graphs

Definition

$G$ is perfect $\iff \chi(H) = \omega(H)$ for each induced subgraph $H$ of $G$
Chordal Graphs

Definition

\( G \text{ is chordal} \iff G \text{ contains no induced cycles longer than 3} \)
Chordal Graphs

Definition

$G$ is chordal $\iff G$ contains no induced cycles longer than 3

chordal?
Chordal Graphs

Definition

G is chordal ⇐⇒ G contains no induced cycles longer than 3

Theorem

Chordal graphs are perfect
Chordal Graphs

Definition

$G$ is chordal $\iff G$ contains no induced cycles longer than 3

Theorem

Chordal graphs are perfect

Theorem

Chordal graphs can be colored optimally in $O(|V| \cdot \omega(G))$
Overview

1 Graph Theory
   - Perfect Graphs
   - Chordal Graphs

2 SSA Form
   - Dominance
   - $\phi$-functions

3 Interference Graphs
   - Non-SSA Interference Graphs
   - Perfect Elimination Orders
   - Chordal Graphs

4 Interference Graphs of SSA-form Programs
   - Dominance and Liveness
   - Dominance and Interference
   - Spilling
   - Implementing $\phi$-functions

5 Intuition
Dominance

**Definition**

Every use of a variable is dominated by its definition

\[ v \leftarrow \cdots \Rightarrow \cdots \leftarrow v \]

You cannot reach the use without passing by the definition. Else, you could use uninitialized variables.

Dominance induces a tree on the control flow graph.

Sometimes called strict SSA.
Dominance

Definition

Every use of a variable is dominated by its definition

- You cannot reach the use without passing by the definition
- Else, you could use uninitialized variables
- Dominance induces a tree on the control flow graph
- Sometimes called strict SSA
What do $\phi$-functions mean?

$z_1 \leftarrow \phi(x_1, y_1)
z_2 \leftarrow \phi(x_2, y_2)
z_3 \leftarrow \phi(x_3, y_3)$

Frequent misconception

Put a sequence of copies in the predecessors
What do $\phi$-functions mean?

Frequent misconception
Put a sequence of copies in the predecessors
What do \( \phi \)-functions mean?

Lost Copies

- Cannot simply push copies in predecessor
- Copies are also executed if we jump from \( B \) to \( C \)
- Need to remove critical edges (edge from \( B \) to \( A \))
What do $\phi$-functions mean?

Lost Copies

- Cannot simply push copies in predecessor
- Copies are also executed if we jump from $B$ to $C$
- Need to remove critical edges (edge from $B$ to $A$)
What do \( \phi \)-functions mean?

\( \phi \)-swap

\[
\begin{align*}
z_1 & \leftarrow \phi(\cdot, z_2) \\
z_2 & \leftarrow \phi(\cdot, z_1)
\end{align*}
\]

- \( z_1 \) overwritten before used

\[
\begin{align*}
z_1 & \leftarrow z_2 \\
z_2 & \leftarrow z_1
\end{align*}
\]
What do $\phi$-functions mean?

$\phi$-swap

\[
\begin{align*}
z_1 &\leftarrow \phi(\cdot, z_2) \\
z_2 &\leftarrow \phi(\cdot, z_1)
\end{align*}
\]

- $z_1$ overwritten before used
What do $\phi$-functions mean?

$z_1 \leftarrow \phi(x_1, y_1)$
$z_2 \leftarrow \phi(x_2, y_2)$
$z_3 \leftarrow \phi(x_3, y_3)$

$(z_1, z_2, z_3) \leftarrow (x_1, x_2, x_3)$
$(z_1, z_2, z_3) \leftarrow (y_1, y_2, y_3)$

The Reality

$\phi$-functions correspond to parallel copies on the incoming edges
\(\phi\)-functions and uses

- Does not fulfill dominance property
- \(\phi\)s do not use their operands in the \(\phi\)-block
- Uses happen in the predecessors

\[ z_1 \leftarrow \phi(x_1, y_1) \]
\[ z_2 \leftarrow \phi(x_2, y_2) \]
\[ z_3 \leftarrow \phi(x_3, y_3) \]
Does not fulfill dominance property

\(\phi\)s do not use their operands in the \(\phi\)-block

Uses happen in the predecessors

Split \(\phi\)-functions in two parts:

- Split critical edges
- Read part \((\phi^r)\) in the predecessors
- Write part \((\phi^w)\) in the block
- Correct modelling of liveness
Overview

1. Graph Theory
   - Perfect Graphs
   - Chordal Graphs

2. SSA Form
   - Dominance
   - $\phi$-functions

3. Interference Graphs
   - Non-SSA Interference Graphs
   - Perfect Elimination Orders
   - Chordal Graphs

4. Interference Graphs of SSA-form Programs
   - Dominance and Liveness
   - Dominance and Interference
   - Spilling
   - Implementing $\phi$-functions

5. Intuition
Non-SSA Interference Graphs

An inconvenient property

Program

```
Program

$ a \leftarrow 1$

$ d \leftarrow 1$
$ e \leftarrow a + 1$
$ \leftarrow d$

$ b \leftarrow a + a$
$ c \leftarrow a + 1$
$ e \leftarrow b + 1$
$ \leftarrow c$
```

Interference Graph

The number of live variables at each instruction (register pressure) is 2

However, we need 3 registers for a correct register allocation

In theory, this gap can be arbitrarily large (Mycielski Graphs)
Graph-Coloring Register Allocation

[Chaitin '80, Briggs '92, Appel & George '96, Park & Moon '04]

- Every undirected graph can occur as an interference graph
  \(\implies\) Finding a \(k\)-coloring is NP-complete

- Color using heuristic
  \(\implies\) Iteration necessary

- Might introduce spills although IG is \(k\)-colorable

- Rebuilding the IG each iteration is costly
Graph-Coloring Register Allocation

[Chaitin ’80, Briggs ’92, Appel & George ’96, Park & Moon ’04]

- Spill-code insertion is **crucial** for the program’s performance
- Hence, it should be very sensitive to the structure of the program
  - Place load and stores carefully
  - Avoid spilling in loops!
- Here, it is merely a fail-safe for coloring
Subsequently remove the nodes from the graph
Subsequently remove the nodes from the graph

elimination order

$d,$
Subsequently remove the nodes from the graph.

This graph is 3-colorable. We obviously picked a wrong order.

Elimination order: d, e,
Subsequently remove the nodes from the graph

elimination order

\[ d, e, c, \]
Subsequently remove the nodes from the graph

elimination order
d, e, c, a,
Subsequently remove the nodes from the graph.

Elimination order: d, e, c, a, b

But... this graph is 3-colorable. We obviously picked a wrong order.
Coloring

- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color

Elimination order: $d, e, c, a, b$
Coloring

- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color

Elimination order:

- d, e, c, a,
Coloring

- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color

![Graph Diagram]

Elimination order: d, e, c,
Coloring

- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color

\[
\begin{array}{c}
\text{d, e,} \\
\text{elimination order}
\end{array}
\]
Coloring

- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color

![Graph diagram]

Elimination order: d, e, a, b, c

But... this graph is 3-colorable. We obviously picked a wrong order.
Coloring

- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color

![Graph diagram]

elimination order
Coloring

- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color

But... this graph is 3-colorable. We obviously picked a wrong order.
Coloring

- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color

But... this graph is 3-colorable. We obviously picked a wrong order.
Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected.
Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected.
**Perfect Elimination Order (PEO)**

All not yet eliminated neighbors of a node are mutually connected.

**Diagram:**

```
  d -- e
 /    \
 a      b
     /  \
     c
```

**Elimination Order:**
a, c,
Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected.

Elimination order: a, c, d,
Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected

elimination order
a, c, d, e,
Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected

Elimination order:

\[ a, c, d, e, b \]
Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected.

Elimination order:

\[ a, c, d, e, \]
Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected.

Elimination order:

a, c, d,
Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected

**Elimination Order**

\[ a, c, \]

From Graph Theory [Berge '60, Fulkerson/Gross '65, Gavril '72]
Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected

elimination order

a,
Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected.
Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected.

From Graph Theory [Berge ’60, Fulkerson/Gross ’65, Gavril ’72]

- A PEO allows for an optimal coloring in $k \times |V|$.
- The number of colors is bound by the size of the largest clique.
Coloring

PEOs

- Graphs with holes larger than 3 have no PEO, e.g.

- $G$ has a PEO $\iff G$ is chordal
Coloring

PEOs

- Graphs with holes larger than 3 have no PEO, e.g.

\[ G \text{ has a PEO} \iff G \text{ is chordal} \]

Core Theorem of SSA Register Allocation

- The dominance relation in SSA programs induces a PEO in the IG
- Thus, SSA IGs are chordal
Overview

1 Graph Theory
   - Perfect Graphs
   - Chordal Graphs

2 SSA Form
   - Dominance
   - $\phi$-functions

3 Interference Graphs
   - Non-SSA Interference Graphs
   - Perfect Elimination Orders
   - Chordal Graphs

4 Interference Graphs of SSA-form Programs
   - Dominance and Liveness
   - Dominance and Interference
   - Spilling
   - Implementing $\phi$-functions

5 Intuition
Liveness and Dominance

- Each instruction $\ell$ where a variable $v$ is live, is dominated by $v$
Liveness and Dominance

- Each instruction $\ell$ where a variable $v$ is live, is dominated by $v$

Why?

- Assume $\ell$ is not dominated by $v$
- Then there's a path from start to some usage of $v$ not containing the definition of $v$
- This cannot be since each value must have been defined before it is used
Liveness and Dominance

- Each instruction $\ell$ where a variable $v$ is live, is dominated by $v$

**Why?**

- Assume $\ell$ is not dominated by $v$
- Then there's a path from start to some usage of $v$ not containing the definition of $v$
- This cannot be since each value must have been defined before it is used
Interference and Dominance

- Assume $v, w$ interfere, i.e. they are live at some instruction $\ell$
- Then, $v \succeq \ell$ and $w \succeq \ell$
- Since dominance is a tree, either $v \succeq w$ or $w \succeq v$

\[
\begin{array}{c}
\text{v} \\
\hline
\{\succeq, \preceq\}
\end{array}
\]

\[
\text{w}
\]
Interference and Dominance

- Assume \( v, w \) interfere, i.e. they are live at some instruction \( \ell \)
- Then, \( v \succeq \ell \) and \( w \succeq \ell \)
- Since dominance is a tree, either \( v \succeq w \) or \( w \succeq v \)

Consequences

- Each edge in the IG is directed by dominance
- The interference graph is an “excerpt” of the dominance relation
Interference and Dominance

- Assume $v \succeq w$
- Then, $v$ is live at $w$
Interference and Dominance

- Assume $v \preceq w$

- Then, $v$ is live at $w$

Why?

- If $v$ and $w$ interfere then there is a place where both are live
- $w$ dominates all places where $w$ is live
- Hence, $v$ is live inside $w$’s dominance tree
- Thus, $v$ is live at $w$
Interference and Dominance

Consider three nodes $u, v, w$ in the IG:

Thus, they interfere

Conclusion

All variables that interfere with $w$ dominate $w$. . . are mutually connected in the IG
Interference and Dominance

Consider three nodes $u, v, w$ in the IG:

- $u$ is live at $w$
- $v$ is live at $w$

Thus, they interfere

Conclusion

All variables that . . . interfere with $w$ . . . are mutually connected in the IG

???
Interference and Dominance

Consider three nodes $u, v, w$ in the IG:

- $u$ is live at $w$
- $v$ is live at $w$
- Thus, they interfere

Conclusion: All variables that... are mutually connected in the IG.
Interference and Dominance

Consider three nodes $u, v, w$ in the IG:

$\geq$ or $\leq$

- $u$ is live at $w$
- $v$ is live at $w$
- Thus, they interfere

Conclusion

All variables that ... 
- interfere with $w$
- dominate $w$

... are mutually connected in the IG
Dominance and PEOs

- Before a value $v$ is added to a PEO, add all values whose definitions are dominated by $v$
- A post order walk of the dominance tree defines a PEO
- A pre order walk of the dominance tree yields a coloring sequence
- IGs of SSA-form programs can be colored **optimally** in $O(\omega(G) \cdot |V|)$
- **Without** constructing the interference graph itself
## Theorem

For each clique in the IG there is a program point where all nodes in the clique are live.
Spilling

Theorem

For each clique in the IG there is a program point where all nodes in the clique are live.

- Dominance induces a total order inside the clique
  \[ \Rightarrow \text{ There is a “smallest” value } d \]

- All others are live at the definition of \( d \)
Spilling

Consequences

- The chromatic number of the IG is exactly determined by the number of live variables at the labels.
- Lowering the number of values live at each label to $k$ makes the IG $k$-colorable.
- We know in advance where values must be spilled $\Rightarrow$ All labels where the pressure is larger than $k$.
- Spilling can be done before coloring and coloring will always succeed afterwards.
Spilling

Consequences

- The chromatic number of the IG is exactly determined by the number of live variables at the labels.
- Lowering the number of values live at each label to $k$ makes the IG $k$-colorable.
- We know in advance where values must be spilled. \[\Rightarrow\] All labels where the pressure is larger than $k$.
- Spilling can be done before coloring and
- Coloring will always succeed afterwards.

Conclusion

- No iteration as in Chaitin/Briggs-allocators.
- No interference graph necessary.
Getting out of SSA

- We now have a $k$-coloring of the SSA interference graph.
- Can we turn that program into a non-SSA program and maintain the coloring?
Getting out of SSA

- We now have a $k$-coloring of the SSA interference graph
- Can we turn that program into a non-SSA program and maintain the coloring?

Central question

What to do about $\phi$-functions?
Φ-Functions

Consider the following example:

\[ z_1 \leftarrow \phi(x_1, y_1) \]
\[ z_2 \leftarrow \phi(x_2, y_2) \]
\[ z_3 \leftarrow \phi(x_3, y_3) \]
Φ-Functions

- Consider following example

\[
\begin{align*}
(z_1, z_2, z_3) &\leftarrow (x_1, x_2, x_3) \\
(z_1, z_2, z_3) &\leftarrow (y_1, y_2, y_3)
\end{align*}
\]

\[
\begin{align*}
z_1 &\leftarrow \phi(x_1, y_1) \\
z_2 &\leftarrow \phi(x_2, y_2) \\
z_3 &\leftarrow \phi(x_3, y_3)
\end{align*}
\]

- Φ-functions are parallel copies on control flow edges
Φ-Functions

- Consider following example

\[
\begin{align*}
(z_1, z_2, z_3) & \leftarrow (x_1, x_2, x_3) \\
(z_1, z_2, z_3) & \leftarrow (y_1, y_2, y_3)
\end{align*}
\]

\[
\begin{align*}
z_1 & \leftarrow \phi(x_1, y_1) \\
z_2 & \leftarrow \phi(x_2, y_2) \\
z_3 & \leftarrow \phi(x_3, y_3)
\end{align*}
\]

- Φ-functions are **parallel copies** on control flow edges

- Considering assigned registers . . .
Φ-Functions

- Consider following example

\[
\begin{align*}
z_1 & \leftarrow \phi(x_1, y_1) \\
z_2 & \leftarrow \phi(x_2, y_2) \\
z_3 & \leftarrow \phi(x_3, y_3)
\end{align*}
\]

- Φ-functions are parallel copies on control flow edges
- Considering assigned registers …
- … Φs represent register permutations
Permutations

- A permutation can be implemented with copies if one auxiliary register is available.

- Permutations can be implemented by a series of transpositions (i.e. swaps).

- A transposition can be implemented by three $\text{xors}$ without a third register.
Intuition: Why do SSA IGs do not have cycles?

Why are SSA IGs chordal?

Program

\[
\begin{align*}
    a & \leftarrow \cdots \\
    b & \leftarrow \cdots \\
    c & \leftarrow \cdots \\
    d & \leftarrow a + b \\
    e & \leftarrow c + 1
\end{align*}
\]

Live Ranges

How can we create a 4-cycle \( \{a, c, d, e\} \)?

Interference Graph
Intuition: Why do SSA IGs do not have cycles?

Why are SSA IGs chordal?

Program

\[
\begin{align*}
a & \leftarrow \cdots \\
b & \leftarrow \cdots \\
c & \leftarrow \cdots \\
d & \leftarrow a + b \\
e & \leftarrow c + 1 \\
a & \leftarrow \cdots 
\end{align*}
\]

Live Ranges

\[
\begin{align*}
a & \\
b & \\
c & \\
d & \\
e & \\
a & 
\end{align*}
\]

Interference Graph

How can we create a 4-cycle \( \{a, c, d, e\} \)?

- Redefine \( a \) \(\implies\) SSA violated!
Intuition: $\phi$-functions break cycles in the IG

Program and live ranges:

- $a \leftarrow \cdots$
- $d \leftarrow \cdots$
- $e \leftarrow a + \cdots$
- $\leftarrow d$
- $b \leftarrow \cdots$
- $c \leftarrow a + \cdots$
- $e \leftarrow b$
- $\leftarrow c$

Interference Graph:
Intuition: $\phi$-functions break cycles in the IG

Program and live ranges

\begin{align*}
d &\leftarrow \cdots \\
e_1 &\leftarrow a + \cdots \\
&\leftarrow d \\
e_3 &\leftarrow \phi(e_1, e_2) \\
a &\leftarrow \cdots \\
b &\leftarrow \cdots \\
c &\leftarrow a + \cdots \\
e_2 &\leftarrow b \\
&\leftarrow c
\end{align*}
Intuition: Why destroying SSA before RA is bad

Parallel copies

\((a', b', c', d') \leftarrow (a, b, c, d)\)

Sequential copies

\[
\begin{align*}
d' & \leftarrow d \\
c' & \leftarrow c \\
b' & \leftarrow b \\
a' & \leftarrow a
\end{align*}
\]
Intuition: Why destroying SSA before RA is bad

Parallel copies

\[(a', b', c', d') \leftarrow (a, b, c, d)\]

Sequential copies

\[
\begin{align*}
d' & \leftarrow d \\
c' & \leftarrow c \\
b' & \leftarrow b \\
a' & \leftarrow a
\end{align*}
\]
Intuition: Why destroying SSA before RA is bad

Parallel copies

\((a', b', c', d') \leftarrow (a, b, c, d)\)

Sequential copies

\(d' \leftarrow d\)
\(c' \leftarrow c\)
\(b' \leftarrow b\)
\(a' \leftarrow a\)
IGs of SSA-form programs are chordal
The dominance relation induces a PEO
No further spills after pressure is lowered
Register assignment optimal in linear time
Do not need to construct interference graph
Allocator without iteration