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

**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 \( G \)

\[
\begin{array}{c}
\omega(G) & 3 \\
\chi(G) & 2
\end{array}
\]
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 \( G \)
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

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>G</em> is perfect ⇐⇒ $\chi(H) = \omega(H)$ for each induced subgraph <em>H</em> of <em>G</em></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 $G$

<table>
<thead>
<tr>
<th>perfect?</th>
<th>✓</th>
<th>✓</th>
</tr>
</thead>
</table>
Chordal Graphs

Definition

$G$ is chordal $\iff G$ contains no induced cycles longer than 3
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 $O(|V| \cdot \omega(G))$
Chordal Graphs

Definition

\[ G \text{ is chordal } \iff G \text{ contains no induced cycles longer than 3} \]

Theorem

Chordal graphs are perfect

\[ \text{Chordal graphs are perfect} \]
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

\[ v \leftarrow \cdots \]

You cannot reach the use without passing by the definition. Otherwise, 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 Copy Problem

$\begin{align*}
&\ x_1 \leftarrow \\
&\ x_3 \leftarrow x_1 \\
&\ x_3 \leftarrow \phi(x_1, x_2) \\
&\ x_2 \leftarrow x_3 + 1 \\
\end{align*}$

$\begin{align*}
&\ x_2 \leftarrow x_3 + 1 \\
&\ x_3 \leftarrow x_2 \\
\end{align*}$

Cannot simply push copies in predecessor

Copies are also executed if we jump out of the loop

Need to remove critical edges (loopback edge)
What do $\phi$-functions mean?

Lost Copy Problem

- Cannot simply push copies in predecessor
- Copies are also executed if we jump out of the loop
- Need to remove critical edges (loopback edge)
What do $\phi$-functions mean?

Swap Problem

\[
\begin{align*}
a_1 &\leftarrow \\
b_1 &\leftarrow \\
a_2 &\leftarrow \phi(a_1, b_2) \\
b_2 &\leftarrow \phi(b_1, a_2)
\end{align*}
\]

\[
\begin{align*}
a_1 &\leftarrow \\
b_1 &\leftarrow \\
a_2 &\leftarrow a_1 \\
b_2 &\leftarrow b_1
\end{align*}
\]

\[
\begin{align*}
a_1 &\leftarrow \\
b_1 &\leftarrow \\
a_2 &\leftarrow b_2 \\
b_2 &\leftarrow a_2
\end{align*}
\]
What do $\phi$-functions mean?

Swap Problem

- $a_2$ overwritten before used
- All $\phi$s in a block need to be evaluated simultaneously
What do $\phi$-functions mean?

The Reality

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

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

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

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

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

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
  \[\Rightarrow\] Finding a $k$-coloring is NP-complete

- Color using heuristic
  \[\Rightarrow\] Iteration necessary

- Might introduce spills although IG is $k$-colorable

- Rebuilding the IG each iteration is costly
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

Example graph:

```
  a -- b -- c
   |    |
  d---e
```

**elimination order**
Subsequently remove the nodes from the graph.
Subsequently remove the nodes from the graph

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

elimination order
\[ d, e, \]
Coloring

- Subsequently remove the nodes from the graph

![Graph Diagram]

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

elimination order

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

elimination order
d, e, c, a,
Coloring

- Subsequently remove the nodes from the graph

![Graph diagram]

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

![Graph diagram]

- 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,
```

```latex
\begin{tikzpicture}
  \node (a) at (0,0) [fill, circle, inner sep=2pt] {a};
  \node (b) at (1,0) [fill, circle, inner sep=2pt, red] {b};
  \node (c) at (2,0) [fill, circle, inner sep=2pt] {c};
  \node (d) at (1,1.5) [fill, circle, inner sep=2pt] {d};
  \node (e) at (2,1.5) [fill, circle, inner sep=2pt] {e};

  \draw (a) -- (b) -- (c) -- (a);
  \draw (d) -- (b) -- (e) -- (d);
\end{tikzpicture}
```
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.

elimination order

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

Re-insert the nodes in reverse order

Assign each node the next possible color

elimination order

d,
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.

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.
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.

![Graph showing Perfect Elimination Order](image)

**elimination order**

a,
Perfect Elimination Order (PEO)

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

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

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

**Diagram**

- Nodes: a, b, c, d, e
- 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

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, 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.

![Graph Diagram]

**elimination order**

a, c,
Coloring
PEOs

Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected

d, e, c, a, b

elimination order
a,
Perfect Elimination Order (PEO)

All not yet eliminated neighbors of a node are mutually connected

Elimination order:

A PEO allows for an optimal coloring in $k \times |V|$.

The number of colors is bound by the size of the largest clique 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.

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

\[ \text{\( G \) has a PEO} \iff \text{\( G \) 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 \( v \)

```
start

v ← ···

ℓ : ···

··· ← v
```
Liveness and Dominance

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

```
start

v ← ···

\ell : ···

··· ← 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, \( v \geq \ell \) and \( w \geq \ell \)
- Since dominance is a tree, either \( v \geq w \) or \( w \geq v \)

![Diagram showing the relationship between \( v \) and \( w \) with \( \geq \) and \( \leq \) relations]

\( v \) \( \{\geq, \leq\} \) \( w \)
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 \( w \succeq v \)

\[ v \xrightarrow{\{\succeq, \preceq\}} w \]

Consequences

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

- Assume $v \trianglerighteq w$
- 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**

- 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
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 ... 
- 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.
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 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

\[
(z_1, z_2, z_3) \leftarrow (x_1, x_2, x_3)
\]

\[
(z_1, z_2, z_3) \leftarrow (y_1, y_2, y_3)
\]

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

- 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

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

Φ-functions are parallel copies on control flow edges

Considering assigned registers . . .

. . . Φs represent register 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 XORs without a third register.
Intuition: Why do SSA IGs do not have cycles?
Why are SSA IGs chordal?

Program       Live Ranges
\[ a \leftarrow \cdots \]
\[ b \leftarrow \cdots \]
\[ c \leftarrow \cdots \]
\[ d \leftarrow a + b \]
\[ e \leftarrow c + 1 \]

How can we create a 4-cycle \( \{a, c, d, e\} \)?
Intuition: Why do SSA IGs do not have cycles?

Why are SSA IGs chordal?

Program   Live Ranges

\[ a \leftarrow \cdots \]
\[ b \leftarrow \cdots \]
\[ c \leftarrow \cdots \]
\[ d \leftarrow a + b \]
\[ e \leftarrow c + 1 \]
\[ a \leftarrow \cdots \]

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

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

Program and live ranges

\[
\begin{align*}
  d & \leftarrow \cdots \\
  e_1 & \leftarrow a + \cdots \\
  & \leftarrow d \\
  e_3 & \leftarrow \phi(e_1, e_2)
\end{align*}
\]

Interference Graph

\[
\begin{align*}
  a & \rightarrow d \\
  b & \rightarrow e_1 \\
  & \rightarrow e_2 \\
  c & \rightarrow e_2 \\
  & \rightarrow e_3 \\
\end{align*}
\]
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

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

Summary