The Development of Static Single Assignment Form

Kenneth Zadeck
NaturalBridge, Inc.
zadeck@naturalbridge.com
Ken's Graduate Optimization Seminar

• We learned:
  1. what kinds of problems could be addressed by compiler optimization.
  2. how to formulate optimization problems as dataflow equations.
  3. how to solve dataflow equations.
Ken's Graduate Optimization Seminar

• We learned:
  1. what kinds of problems could be addressed by compiler optimization.
  2. how to formulate optimization problems as dataflow equations.
  3. how to solve dataflow equations.
• Because of my dyslexia, I am really bad at 2.
Ken's Graduate Optimization Seminar

- We learned:
  1. what kinds of problems could be addressed by compiler optimization.
  2. how to formulate optimization problems as dataflow equations.
  3. how to solve dataflow equations.
- Because of my dyslexia, I am really bad at 2.
- I was able to reason about dataflow problems geometrically.
Variable by Variable Analysis.

- Viewing the program variable by variable exposes structure that is obscured by the dataflow model:
  - A kill allows the cfg to be clipped.
  - The dataflow for a single variable can be solved without iteration.
The Dataflow Abstraction

Dataflow analysis is an abstraction:

• Get:
  – Use bit vectors for simple problems.
  – Use interval analysis to solve equations quickly.
The Dataflow Abstraction

Dataflow analysis is an abstraction:

- **Get:**
  - Use bit vectors for simple problems.
  - Use interval analysis to solve equations quickly.

- **Give:**
  - Cannot play games with kill sets.
The Dataflow Abstraction

Dataflow analysis is an abstraction:

- **Get:**
  - Use bit vectors for simple problems.
  - Use interval analysis to solve equations quickly.

- **Give:**
  - Cannot play games with kill sets.
  - Cannot do SSA form.
Constant Propagation

\[
\begin{align*}
j &= 0 \\
k &= 1 \\
\text{if } (j > 0) & \quad \text{then } k = 4 \\
k \ ?
\end{align*}
\]
The Development of SSA Form

Constant Propagation - Kildall

\[
\begin{align*}
j & = 0 \\
k & = 1 \\
\text{if } (j > 0) & \\
\quad & \text{then } k = 4 \\
k & ?
\end{align*}
\]
Constant Propagation - Kildall

\[ j = 0 \]
\[ k = 1 \]
\[ \text{if } (j > 0) \]
\[ \quad \text{then } k = 4 \]
\[ k? \]
Constant Propagation - Kildall

\[
j = 0
\]
\[
k = 1
\]
\[
\text{if } (j > 0)\text{ then } k = 4
\]
\[k ?\]
The Development of SSA Form

Constant Propagation - Kildall

\[
\begin{align*}
  j &= 0 \\
  k &= 1 \\
  \text{if } (j > 0) & \\
  \quad \text{then } k &= 4 \\
\end{align*}
\]

\[
\begin{array}{cc}
  j & k \\
  0 & T \\
  0 & 1 \\
  0 & 1 \\
\end{array}
\]
Constant Propagation - Kildall

\[
\begin{array}{r}
  j = 0 \\
  k = 1 \\
  \text{if } (j > 0) \\
  \quad \text{then } k = 4 \\
  \end{array}
\]

\[
\begin{array}{r}
  j & k \\
  0 & T \\
  0 & 1 \\
  0 & 4 \\
  \end{array}
\]
Constant Propagation - Kildall

\[
\begin{align*}
\quad & j = 0 & j & 0 \\
\quad & k = 1 & k & 1 \\
\text{if } (j > 0) & & & \\
\quad & \text{then } k = 4 & & 4 \\
\quad & k ? & k & 0
\end{align*}
\]
Constant Propagation - Wegbreit

\[
\begin{align*}
 j &= 0 & 1 \\
 k &= 1 & 2 \\
 \text{if } (j > 0) & \\n \quad & \text{then } k = 4 & 4 \\
 \end{align*}
\]

\[ j \quad k \quad (3 \wedge 4) \quad 3 \]

k ? 5

The Development of SSA Form
Constant Propagation - Wegbreit

\[
\begin{align*}
j &= 0 & 1 \\
k &= 1 & 2 \\
\text{if } (j > 0) & & 3 \\
\text{then } k &= 4 & 4 \\

k ? & & 5
\end{align*}
\]
### Constant Propagation - Wegbreit

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if (j &gt; 0)</td>
<td>then k = 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>k ?</td>
<td>5</td>
<td></td>
<td>0 1</td>
</tr>
</tbody>
</table>

The Development of SSA Form
Constant Propagation – Reif & Lewis

j = 0
k = 1
if (j > 0)
    then k = 4

k = k
k ?

• Add Reif and Tarjan birthpoints.
### Constant Propagation – Reif & Lewis

1. \( j = 0 \)
2. \( k = 1 \)
3. \( \text{if } (j > 0) \) then \( k = 4 \)
4. \( k = k \)
5. \( k ? \)

- Add Reif and Tarjan birthpoints.
- Add def-use chains.

---

The Development of SSA Form
The Development of SSA Form

Constant Propagation – Reif & Lewis

\[ j = 0 \]
\[ k = 1 \]
\[ \text{if } (j > 0) \]
\[ \quad \text{then } k = 4 \]
\[ k = k \]
\[ k ? \]
Constant Propagation – Reif & Lewis

The Development of SSA Form
The Development of SSA Form

Constant Propagation – Reif & Lewis

\[ j = 0 \]
\[ k = 1 \]
\[ \text{if } (j > 0) \text{ then } k = 4 \]
\[ k = k \]
\[ k ? \]

<table>
<thead>
<tr>
<th></th>
<th>j1</th>
<th>k2</th>
<th>k4</th>
<th>k5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Constant Propagation – Reif & Lewis

\[
\begin{align*}
j &= 0 & j_1 & k_2 & k_4 & k_5 \\
k &= 1 & 1 & 0 \\
\text{if } (j > 0) & 2 & 1 \\
\text{then } k &= 4 & 4 \\
k &= k & 5 \\
k &? & 6
\end{align*}
\]
Constant Propagation – Reif & Lewis

j = 0
k = 1
if (j > 0)
then k = 4
k = k
k = ?

j1 k2 k4 k5
1 0
2 1
3
4 4
5
6
Constant Propagation – Wegman & Zadeck

- Add more identity assignments.
Constant Propagation – Wegman & Zadeck

- Add more identity assignments.
- Propagate values along def-use edges iff statement is executable.
The Development of SSA Form
The Development of SSA Form

Constant Propagation – Wegman & Zadeck

```
j = 0
k = 1
if (j > 0)
    then k = 4
    else k = k
k = k
k ?
```

<table>
<thead>
<tr>
<th></th>
<th>k1</th>
<th>k2</th>
<th>k4</th>
<th>k5</th>
<th>k6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Development of SSA Form

**Constant Propagation – Wegman & Zadeck**

```
j = 0
k = 1
if (j > 0)
    then k = 4
else k = k
k = k
k ?
```

```
<table>
<thead>
<tr>
<th></th>
<th>k1</th>
<th>k2</th>
<th>k4</th>
<th>k5</th>
<th>k6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The Development of SSA Form
The Development of SSA Form
The Development of SSA Form

Constant Propagation – Wegman & Zadeck

\[
\begin{align*}
j &= 0 \\
k &= 1 \\
\text{if } (j > 0) & \quad \text{then } k = 4 \\
\text{else } k &= k \\
k &= k
\end{align*}
\]

\[
\begin{array}{cccccc}
k_1 & k_2 & k_4 & k_5 & k_6 \\
0 & 1 & 1 & 1 & 1
\end{array}
\]
Constant Propagation – Time and Power

- Kildall and Wegbreit use a conventional dataflow framework.
- The time to run these is between $O(N \log NV)$ and $O(N^2 V)$ depending on the type of control flow graph processing.
- Reif & Lewis and Wegman & Zadeck are $O(N)$ for the propagation + NV to compute the birthpoints.
- Kildall ≈ Reif & Lewis
- Wegbreit ≈ Wegman & Zadeck
SSA
Looking Forwards at Wegman & Zadeck

- We had no “vision” of SSA form.
- Wegman & Zadeck is yet another fast technique to perform some transformation that uses a one off data structure.
SSA
Looking Backwards at Wegman & Zadeck

• This is the first SSA optimization algorithm.
• The extra identity assignments change the birthpoints into $\Phi$-functions.
• The algorithm preserves its form while being transformed.
Removal of Invariant Code from Loops

- Ron Cytron
- Andy Lowry
- Kenneth Zadeck

POPL13 - 1986
Removal of Invariant Code from Loops

\[ j = 0 \]

while (...) 

\[ j = j + 1 \]
\[ x = y + 3 \]
\[ z = x + 1 \]

\[ ... = z + j \]

- Both of these statements can be removed from the loop.
- The second can be removed only after the first one is out.
Removal of Invariant Code from Loops

\[
\begin{align*}
    j &= 0 \\
    j &= j \\
\end{align*}
\]

while (...) 
    birthpoint j 
    \[
    \begin{align*}
        j &= j + 1 \\
        x &= y + 3 \\
        z &= x + 1 \\
        \ldots &= z + j \\
        j &= j \\
    \end{align*}
    \]

- Add birthpoints and identity assignments.

The Development of SSA Form
Removal of Invariant Code from Loops

\[ j_1 = 0 \]
\[ j_2 = j_1 \]

while (...) 
  
  birthpoint \( j_2 \)
  
  \[ j_3 = j_2 + 1 \]
  
  \[
  \begin{align*}
  x_1 &= y_1 + 3 \\
  z_1 &= x_1 + 1 \\
  \end{align*}
  \]
  
  \[ ... = z_1 + j_3 \]
  
  \[ j_2 = j_3 \]

• Add birthpoints and identity assignments.
• Rename variables.

The Development of SSA Form
Removal of Invariant Code from Loops

Any insn can be moved outside the loop if:

- the birthpoints of the rhs are outside the loop.
- the statement is not control dependent on a test inside the loop.

\[ j_1 = 0 \]
\[ j_2 = j_1 \]
\[ x_1 = y_1 + 3 \]

while (...) 
  birthpoint \( j_2 \)
  \[ j_3 = j_2 + 1 \]

  \[ z_1 = x_1 + 1 \]
  \[ ... = z_1 + j_3 \]
  \[ j_2 = j_3 \]

The Development of SSA Form
Removal of Invariant Code from Loops

\[ j_1 = 0 \]
\[ j_2 = j_1 \]
\[ x_1 = y_1 + 3 \]
\[ z_1 = x_1 + 1 \]

while (…)
    birthpoint \( j_2 \)
    \[ j_3 = j_2 + 1 \]

\[ \ldots = z_1 + j_3 \]
\[ j_2 = j_3 \]

Any insn can be moved outside the loop if:

- the birthpoints of the rhs are outside the loop.
- the statement is not control dependent on a test inside the loop.
What is in a Name? or The Value of Renaming for Parallelism and Storage Allocation

- Ron Cytron
- Jeanne Ferrante

ICPP87

Proves that the renaming done in the prev paper removes all false dependencies for scalars.
The Origin of Φ-Functions and the Name

• Barry Rosen did not like the identity assignments.
  – He decided to replace them with “phony functions” that were able to see which control flow reached them.
  – A Φ-function was a more publishable name.

• The name Static Single Assignment Form came from the fact that Single Assignment languages were popular then.
Global Value Numbers and Redundant Computations

• Barry Rosen
• Mark Wegman
• Kenneth Zadeck

POPL15 - 1988
Global Value Numbers and Redundant Computations

- Classical value numbering algorithms are restricted to programs with no joins.
- With $\Phi$-functions, it is possible to extend value numbering to acyclic regions.
Global Value Numbers and Redundant Computations

\[ x_3 = \Phi(x_1, x_2) \]

\[ \ldots = x_3 + y_1 \]
Global Value Numbers and Redundant Computations

\[ \ldots = x_1 + y_1 \]
\[ \ldots = x_2 + y_1 \]
\[ x_3 = \Phi(x_1, x_2) \]
\[ \ldots = x_3 + y_1 \]
Detecting Equality of Values in Programs

• Bowen Alpern
• Mark Wegman
• Kenneth Zadeck

POPL15 - 1988
Detecting Equality of Values in Programs

• Convert the program to SSA form.
Detecting Equality of Values in Programs

• Convert the program to SSA form.
• Use Hopcroft's finite state minimization algorithm to partition the program.
  – The dataflow edges are the edges in the graph.
  – Label each $\Phi$-function at join point $n$ to $\Phi_n$.
  – The operators are labels on the nodes. Place all the operations with a given label in the same partition to start.
Detecting Equality of Values in Programs

- Convert the program to SSA form.
- Use Hopcroft's finite state minimization algorithm to partition the program.
  - The dataflow edges are the edges in the graph.
  - Label each $\Phi$-function at join point $n$ to $\Phi_n$.
  - The operators are labels on the nodes. Place all the operations with a given label in the same partition to start.
- After partitioning, any operations in the same partition compute the same value.
Detecting Equality of Values in Programs

• All of us thought this was a very neat trick.
• It is not useful because many people add other tricks to their value numbering.
• We tried for two years to extend this along the lines of those tricks and we failed.
An Efficient Method of Computing Static Single Assignment Form

• Ron Cytron
• Jeanne Ferrante
• Barry Rosen
• Mark Wegman
• Kenneth Zadeck

POPL16 - 1989
An Efficient Method of Computing Static Single Assignment Form

• There should have been two papers in that POPL:
  – An Efficient Method of Computing Static Single Assignment Form by Rosen, Wegman and Zadeck
  – An Efficient Method of Computing the Program Dependence Graph by Cytron and Ferrante.
An Efficient Method of Computing Static Single Assignment Form

• There should have been two papers in that POPL:
  – An Efficient Method of Computing Static Single Assignment Form by Wegman and Zadeck
  – An Efficient Method of Computing the Program Dependence Graph by Cytron and Ferrante.

• We figured out that the algorithms were the same a couple of days before the submission deadline.
  – We barely had time to merge the abstracts.
  – We missed fixing the title.
The Development of SSA Form

An Efficient Method of Computing Static Single Assignment Form

• The algorithm presented here is generally linear.
  – It is a big improvement over Reif & Tarjan which is generally quadratic.

• It has been bettered by:
  – Sreedhar & Gao in POPL22.
The Development of SSA Form

An Efficient Method of Computing Static Single Assignment Form

- The algorithm presented here is generally linear.
  - It is a big improvement over Reif & Tarjan which is generally quadratic.
- It has been bettered by:
  - Sreedhar & Gao in POPL22.
- The journal version has a dead code elimination algorithm.
Analysis of Pointers and Structures

- David Chase
- Mark Wegman
- Kenneth Zadeck

Sigplan 90
Analysis of Pointers and Structures

• One of the first computationally efficient techniques to analyze pointers.

• Makes on minimal use of SSA.
  – Use of the ssa names gives a small amount of flow sensitivity to a problem that otherwise must be solved in a flow insensitive way.
  – This trick is used in other new algorithms.

• Many new and much better techniques have followed.
What Happened Next

• We stopped working on SSA.
  – None of us actually worked on a compiler project.
  – I was at Brown University.
  – We were blocked from transferring SSA to the IBM product compilers.

• People outside of IBM were picking it up.
  – Apollo, DEC, HP, SGI, and SUN were all using it to some extent.
  – We had built a good foundation.
  – It was easy to play the game.
Why Did SSA Win?

• All things being equal, SSA form only accounts for a few percent code quality over the comparable data flow techniques.
  – SSA techniques run much faster.
  – Scanning the program, building the transfer functions, and solving the equations is slow.
  – Incremental data flow never really worked.

• The high gain, parallel extraction techniques need SSA to keep things clean.

• SSA is easier to understand than dataflow.
  – I have no standing to say this.
References

There is a good bibliography online that contains most of the SSA papers:

- [http://www.cs.man.ac.uk/~jsinger/ssa.html](http://www.cs.man.ac.uk/~jsinger/ssa.html)
- It is accessible from the Wikipedia article on SSA.
Postscript

• For the last year I have been working to bring the analysis in the GCC back ends up to date.
  – It is infeasible to use SSA for the back ends.
  – Must maintain compatibility with the existing machine descriptions.
  – The back end is currently state of the art as of about 1986.

• The middle machine independent parts are now all SSA.
Postscript

• For the last year I have been working to bring the analysis in the GCC back ends up to date.
  – It is infeasible to use SSA for the back ends.
  – Must maintain compatibility with the existing machine descriptions.
  – The back end is currently state of the art as of about 1986.

• The middle machine independent parts are now all SSA.

• I still do not speak dataflow equations.