Improvements to the Psi-SSA Representation

François de Ferrière
STMicroelectronics
Grenoble - France

SCOPES ´0 7
The Psi-SSA Representation

- Motivation
- Definition
- Properties
- Benefits
- Construction
- Transformations
- Destruction
- Conclusion
Motivation for Psi-SSA

- Our target processor has partial support for predication
- Internal representation in the compiler backend
  - At target instruction level
  - Based on SSA form for some optimizations
    - If-conversion
    - Value-range analysis
    - Target specific optimizations
- Original Psi-SSA representation was presented in
  “Efficient static single assignment form for predication”
  A.Stouchinin, F. de Ferrière - Micro-34
- Needed some improvements for partial predication
Definition of Psi-SSA

Psi-SSA is a Single Static Assignment representation

Psi-SSA adds support for predicated instructions

\[
\begin{align*}
\text{if (p)} \quad & a_1 = 1 \\
\text{else} \quad & a_2 = -1 \\
a_3 &= \text{Phi}(a_1, a_2) \\
\text{if (q)} \quad & b_1 = 0 \\
b_2 &= \text{Phi}(a_3, b_1) \\
b_3 &= a_3 + b_2
\end{align*}
\]

SSA representation

\[
\begin{align*}
p? & a_1 = 1 \\
!p? & a_2 = -1 \\
a_3 &= \text{Psi}(p?a_1, !p?a_2) \\
q? & b_1 = 0 \\
b_2 &= \text{Psi}(p?a_1, !p?a_2, q?b_1) \\
b_3 &= a_3 + b_2
\end{align*}
\]

Psi-SSA representation
Properties of Psi-SSA

- A Psi operation merges values defined on different predicates.
- Arguments of a Psi operation are non-predicated and predicated definitions.
- The result of a Psi operation is a non-predicated definition.
- The execution of a Psi operation returns the value of the rightmost variable whose predicate is true at runtime.

\[
\begin{align*}
\text{a}_1 &= 1 \\
p? \ a_2 &= -1 \\
\text{a}_3 &= \text{Psi}(1?\text{a}_1, p?\text{a}_2) \\
q? \ b_1 &= 0 \\
\text{b}_2 &= \text{Psi}(1?\text{a}_1, p?\text{a}_2, q?b_1) \\
\text{b}_3 &= \text{a}_3 + \text{b}_2
\end{align*}
\]
Properties of Psi-SSA (cont’d)

- A predicate is associated with each argument
  - Allow for speculation of predicated definitions
  - Add support for partial predication
- Predicate domains need not be disjoint
  - Several predicates may be true at the same time.
- The order of the arguments in a Psi operation is significant

\[
\begin{align*}
a_1 &= 1 \\
p \quad a_2 &= -1 \\
a_3 &= \Psi(1?a_1,p?a_2) \\
q \quad b_1 &= 0 \\
b_2 &= \Psi(1?a_1,p?a_2,q?b_1) \\
b_3 &= a_3 + b_2
\end{align*}
\]
Benefits of Psi-SSA

- Easy to implement on top of an SSA representation
- No penalty if no predicated operation
- More flexibility in optimization ordering for predicated instruction sets
  - SSA algorithms are easy to adapt to the Psi-SSA representation
  - If-Conversion under SSA
- Specific optimizations on predicated code
  - Predicate promotion
Benefits of Psi-SSA (cont’d)

- Standard SSA algorithms can be used on Psi-SSA
  - Predicated instructions are treated as unconditional
  - New rules have to be defined on Psi operations
  - Constant propagation, dead code elimination, global value numbering have been adapted to this representation

- Example: Constant propagation

\[
\begin{align*}
    a_1 &= 1 & \rightarrow 1 \\
    \text{p? } a_2 &= a_1 + 1 & \rightarrow 2 \\
    \text{!p? } a_3 &= 2 & \rightarrow 2 \\
    a_4 &= \Psi(\text{p? } a_2, \text{!p? } a_3) & \rightarrow 2
\end{align*}
\]
Construction of Psi-SSA

During the SSA construction

- Insertion of Psi operation after predicated definitions

\[
\begin{align*}
  a_1 &= 0 \\
p \? a_2 &= 1 \\
a_3 &= \Psi(1?a_1, p?a_2)
\end{align*}
\]

- While in SSA form by an if-conversion algorithm

\[
\begin{align*}
  a_1 &= 0 \\
  \text{if (p)} \\
  a_2 &= 1 \\
a_3 &= \Phi(a_1, a_2)
\end{align*}
\]

\[
\begin{align*}
  a_1 &= 0 \\
p \? a_2 &= 1 \\
a_3 &= \Psi(1?a_1, p?a_2)
\end{align*}
\]
Transformations on Psi-SSA

Three transformations are described in the original article

- **Psi-inlining**
  
  $x = \Psi(p \cdot a, q \cdot b)$
  
  $y = \Psi(p \mid q \cdot x, r \cdot c)$
  
  $\rightarrow y = \Psi(p \cdot a, q \cdot b, r \cdot c)$

- **Psi-reduction**
  
  $x = \Psi(p \cdot a, q \cdot b, p \cdot c)$
  
  $\rightarrow x = \Psi(q \cdot b, p \cdot c)$

- **Psi-projection**
  
  $x = \Psi(p \cdot a, q \cdot b)$
  
  $\rightarrow x_1 = \Psi(p \cdot a)$
  
  $p \cdot z = x$
  
  $\rightarrow p \cdot z = x_1$

One new transformation is introduced in this paper

- **Psi-predicate promotion**
  
  $x = \Psi(p \cdot a, q \cdot b)$
  
  $\rightarrow x = \Psi(1 \cdot a, q \cdot b)$
### Destruction of Psi-SSA

- Variables connected through a Psi operation must be renamed into a single variable, but:
  - Code motion may have changed the order in which predicated definitions occur.
  - Operation speculation may result in a different predicate on a variable’s definition and on its use in a Psi operation.
  - Copy folding may have introduced interferences between variables in Psi operations.

<table>
<thead>
<tr>
<th>( p? )</th>
<th>( a_2 = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 = 0 )</td>
<td></td>
</tr>
<tr>
<td>( a_3 = \text{Psi}(1?a_1,p?a_2) )</td>
<td></td>
</tr>
</tbody>
</table>

| \( a_1 = 0 \) |
|---|---|
| \( a_2 = 1 \) |
| \( a_3 = \text{Psi}(1?a_1,q?a_2) \) |

\[
\begin{align*}
p? & \quad a_2 = 1 \\
& \quad a_1 = 0 \\
& \quad a_3 = \text{Psi}(1?a_1,p?a_2) \\
\text{a}_1 & \quad = 0 \\
\text{a}_2 & \quad = 1 \\
\text{a}_3 & \quad = \text{Psi}(1?a_1,q?a_2) \\
\end{align*}
\]
Destruction of Psi-SSA (cont’d)

Implemented as two steps added before the algorithm

“Translating out of static assignment form”
V. Sreedhar et al. – Static Analysis Symposium, 1999.

A Psi-Normalize step
- Restores the order of predicated definitions
- Uses the same predicate on a variable’s definition and on its use in a Psi operation

A Psi-congruence step
- Insert copies to remove interferences is psi-congruence classes
- Uses a special definition for liveness on normalized Psi operations

\[
\begin{align*}
p? a_1 &= \\
q? a_2 &= \\
a_3 &= \Psi(p?a_1, q?a_2)
\end{align*}
\]
Destruction of Psi-SSA (cont’d)

- Predicated copies are generated to repair non-normalized Psi operations and interferences between Psi arguments.
- Interferences between Psi arguments must also take into account interferences on Phi operations.
- A simple Predicate Query System is used to eliminate false interferences on disjoint predicates.
Conclusion

- Algorithms to build, optimize and deconstruct the Psi-SSA representation are well defined
- The Psi-SSA representation has proven to be a very effective representation to applying transformations on predicated code for our target processors
- Standard SSA algorithms are easy to adapt to Psi-SSA
- The Psi-SSA representation gives more flexibility in the ordering of optimizations in the compiler back-end
Thanks for your attention

francois.de-ferriere@st.com