Data Dependences and Advanced Induction Variables Detection

Sebastian Pop
pop@cri.ensmp.fr

Centre de Recherche en Informatique (CRI)
École des mines de Paris
France
Overview of LNO

Loop Nest Optimizer Branch

- Natural Loops
  - Number of iters
  - Scalar Evolutions
  - Data Dependences

Analyzers

- Invariant Code Motion
- IV Canonicalization
- IV Optimization
- Checks Elimination
- Vectorizer
- Linear Transforms

Optimizers

- Depends on
  - C
  - C++
  - Java
  - F95

Data Dependences and Advanced Induction Variables Detection – p.2
Data Dependence?

DO I=1,8
  DO J=3,8
    A(I, J) = A(I-3, J-2) + 1
  END DO
END DO

At iteration I = 7, J = 4,
A(7,4) = A(7-3,4-2)+1
so,
A(4,2) must be computed before
A(7,4).

This data dependence can be summarized by a mathematical abstraction, like the distance vector:

\[ \text{Dist} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \]
DO I = 0, N
    T[f(I)] = ...
    ... = T[g(I)]
END DO

Are the elements of T accessed several times? i.e. are there some values \( x, y \in [0, N] \) such that:

\[
f(x) = g(y), \quad f(x) = f(y) \quad \text{or} \quad f(x) = g(y)
\]

→ Need a description of the values of \( f \) and \( g \).
DO I = 0, N
    T[a] = ...
    ... = T[b]
END DO

- Variables $a$ and $b$ are **induction variables**: their values may change with successive $I$ values.

- Goal: describe scalar variables in loops
  - give the successive values (when possible),
  - give a range or an envelope of values.
Chains of Recurrences

- Representation of successive values in loops using a form called **chains of recurrences**.
- For instance, the chain of recurrence

\[ \{1, +, 3\} \]

represents the values of \(a\) in the program:

\[
a = 1 \\
DOFOREVER \\
a = a + 3 \\
END DO
\]
Analyzing SSA Programs

\[
a = 1 \\
\text{DO FOREVER} \\
a = a + 3 \\
\text{END DO}
\]

\[
a = 1 \\
\text{loop} \\
b = \phi (a, c) \\
c = b + 3 \\
\text{end loop}
\]

- Use-def links,
- Phi nodes at control flow junctions.
Induction Variable Analysis

- Value of a variable at each iteration of a loop,
- Analyze on demand,
- Store intermediate results,
- Algorithm:
  1. Walk the use-def edges, find a SCC,
  2. Reconstruct the update expression,
  3. Translate to a chain of recurrence,
  4. (optional) Instantiate parameters.
The initial condition is a definition outside the loop.
Example

\[ a = 3 \]
\[ b = 1 \]
\[ \text{loop} \]
\[ c = \text{phi} \ (a, \ f) \]
\[ d = \text{phi} \ (b, \ g) \]
\[ \text{if} \ (d > 123) \ \text{goto end} \]
\[ e = d + 7 \]
\[ f = e + c \]
\[ g = d + 5 \]
\[ \text{endloop} \]
\[ \text{end:} \]

Depth-first walk the use-defs to a loop-phi node:

\[ c \rightarrow f \rightarrow e \rightarrow d \]
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto end
  e = d + 7
  f = e + c
  g = d + 5
endloop
end:

d \neq c, \text{ walk back, search for another loop-phi:}

\[ d \rightarrow e \rightarrow f \]
Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto end
  e = d + 7
  f = e + c
  g = d + 5
endloop
end:
```

Found the starting loop-phi. The SCC is:

\[ c \rightarrow f \rightarrow c \]
Example

```
a = 3
b = 1
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto end
  e = d + 7
  f = e + c
  g = d + 5
endloop
end:
```

Reconstruct the update expression:

\[ c + e \]
Example

\[ a = 3 \]
\[ b = 1 \]

\begin{verbatim}
loop
  c = phi (a, f)
  d = phi (b, g)
  if (d > 123) goto end
  e = d + 7
  f = e + c
  g = d + 5
endloop
end:
\end{verbatim}

\[ c = \phi (a, c + e) \]
\[ c \rightarrow \{a, +, e\} \]
Example

\[a = 3\]
\[b = 1\]

loop
  \[c = \text{phi} (a, f)\]
  \[d = \text{phi} (b, g)\]
  \[\text{if} (d > 123) \text{ goto end}\]
  \[e = d + 7\]
  \[f = e + c\]
  \[g = d + 5\]
endloop
end:

\[c \rightarrow \{a, +, e\} \xrightarrow{\text{Instantiate}} \text{Optional} \cdots\]
Example

\[
\begin{align*}
    a &= 3 \\
    b &= 1 \\
    \text{loop} \\
    \quad c &= \phi(a, f) \\
    \quad d &= \phi(b, g) \\
    \quad \text{if } (d > 123) \text{ goto end} \\
    \quad e &= d + 7 \\
    \quad f &= e + c \\
    \quad g &= d + 5 \\
\text{endloop} \\
\text{end:}
\end{align*}
\]

\[c \rightarrow \{a, +, e\} \xrightarrow{\text{Instantiate}} \{3, +, e\}\]
Example

\[ a = 3 \]
\[ b = 1 \]
\[ \text{loop} \]
\[ c = \phi (a, f) \]
\[ d = \phi (b, g) \]
\[ \text{if} \ (d > 123) \ \text{goto end} \]
\[ e = d + 7 \]
\[ f = e + c \]
\[ g = d + 5 \]
\[ \text{endloop} \]
\[ \text{end:} \]

\[ c \rightarrow \{a, +, e\} \xrightarrow{\text{Instantiate}} \{3, +, e\} \]
\[ e \rightarrow d + 7 \]
Example

\[
\begin{align*}
    a &= 3 \\
    b &= 1 \\
    \text{loop} & \quad \begin{align*}
        c &= \text{phi}(a, f) \\
        d &= \text{phi}(b, g) \\
        \text{if } (d > 123) & \text{ goto end} \\
        e &= d + 7 \\
        f &= e + c \\
        g &= d + 5 \\
    \end{align*} \\
\text{endloop} & \quad \begin{align*}
    c & \rightarrow \{a, +, e\} \xrightarrow{\text{Instantiate}} \{3, +, e\} \\
    e & \rightarrow d + 7 \\
    d & \rightarrow \{1, +, 5\}
\end{align*}
\]
Example

\begin{align*}
a &= 3 \\
b &= 1 \\
loop \\
c &= \text{phi} \ (a, \ f) \\
d &= \text{phi} \ (b, \ g) \\
\text{if} \ (d > 123) \ \text{goto end} \\
e &= d + 7 \\
f &= e + c \\
g &= d + 5 \\
endloop \\
end:
\end{align*}

\[c \rightarrow \{a, +, e\} \xrightarrow{\text{Instantiate}} \{3, +, e\}\]
\[e \rightarrow \{8, +, 5\}\]
\[d \rightarrow \{1, +, 5\}\]
Example

\[ a = 3 \]
\[ b = 1 \]
\[ \text{loop} \]
\[ c = \phi(a, f) \]
\[ d = \phi(b, g) \]
\[ \text{if} \ (d > 123) \text{ goto end} \]
\[ e = d + 7 \]
\[ f = e + c \]
\[ g = d + 5 \]
\[ \text{endloop} \]
\[ \text{end:} \]

\[ c \rightarrow \{a, +, e\} \xrightarrow{\text{Instantiate}} \{3, +, 8, +, 5\} \]
\[ e \rightarrow \{8, +, 5\} \]
\[ d \rightarrow \{1, +, 5\} \]
Summary

From the SSA program:

\[
\text{loop}_1 \\
f = \phi (\text{init}, f + \text{step}) \\
\text{endloop}
\]

Extract the symbolic evolution:

\[
f(x) \rightarrow \{\text{init}, +, \text{step}\}_1(x) \\
x = 0, 1, \ldots, N
\]

Optionally, instantiate the parameters \text{init} and \text{step}. 
Applications

Why another IR?

- information about the evolution of a scalar variable in a loop cannot be represented in SSA
- other analyzers need this information:
  - data dependence testers,
  - number of iterations.
Applications

C  C++  Java  F95

GENERIC

GIMPLE

CFG

SSA

GIMPLE

RTL

Loop Nest Optimizer Branch

Natural Loops

Number of iters

Scalar Evolutions

Data Dependences

Invariant Code Motion

IV Canonicalization

IV Optimization

Checks Elimination

Vectorizer

Linear Transforms

Analyzers

Optimizers

→: Depends on

Data Dependences and Advanced Induction Variables Detection – p.23
Number of iterations

loop
  ...
  if (a > b) goto end
  ...
endloop
end:

1. Find the evolution of $a$ and $b$,
2. Call the niter solver. The result is:
   - an integer constant,
   - a symbolic expression.
Condition Elimination

Algorithm:

1. compute the number of iterations
   - in the loop,
   - in the then clause,
   - in the else clause.

2. when all the iterations fall in one of the branches, eliminate the unused branch.
loop
    ...
    a = ...
    ...
endloop
    a = value after crossing the loop

Algorithm:
1. compute the value of a scalar variable after crossing a loop,
2. assign this value to the variable after the loop,
3. call the constant propagation optimizer.
Conclusion

The LNO branch contains:

- a fast algorithm for analyzing variables in loops,
- the classic Banerjee data dependence testers,
- induction variable optimizations,
- the linear loop transformations,
- the vectorizer.
Merge plan

C C++ Java F95

GENERIC

GIMPLE

CFG

SSA

GIMPLE

RTL

Loop Nest Optimizer Branch

Analyzers

Optimizers

0 Natural Loops

1 Invariant Code Motion

2 Number of iters

3 Data Dependences

4 IV Canonicalization

5 IV Optimization

6 Checks Elimination

7 Vectorizer

8 Linear Transforms

: Depends on

Data Dependences and Advanced Induction Variables Detection – p.28