Preliminary Loop Transformations

Kanika Nema-05305401
Rishiraj Gupta-05305014
Manuj Gupta-05305409

10^{th} November, 2005

Loop transformations

- Normalization (have seen the motivation for this earlier)
- Constant Propogation
- Movement of Loop invariant statements
- Dead Code eliminations

Loop transformations

- Normalization (have seen the motivation for this earlier)
- Constant Propogation
- Movement of Loop invariant statements
- Dead Code eliminations

- Definition-Use graphs - is a graph that contains an edge foreach definition point in the program to every possible use of the variable at run time.)

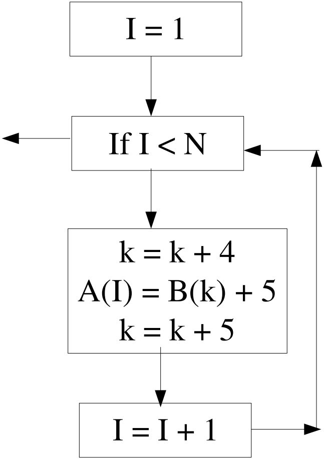
$$
\begin{array}{ll}
S0: & k = 8 \\
 & DO\ I = 1, 10 \\
S1: & k = k + 4 \\
S2: & A[I] = A[k] + 5 \\
100 & ENDO
\end{array}
$$

Construction of Definition Use graph

- uses(b): Set of all variables used within the block, that have no prior definations **within** that block.
- defsout(b): Set of all definitions within the block, that can **reach** other blocks.
- killed(b):Set of all definitions that define variables killed by other definitions **within** block b implies definitions of these variables from other blocks will be killed in this block.

$$reaches(b) = \bigcup(defsout(p) \bigcup(reaches(p) \bigcap \backsim killed(p)))$$

Example



Static Single-Assignment Form

- Motivation

What can be done?

- Each assignement creates a different variable.
- At points where control flow **joins**, insert a special function called ϕ-functions.

Static Single-Assignment Form

- Motivation

What can be done?

- Each assignement creates a different variable.
- At points where control flow **joins**, insert a special function called ϕ-functions.

Where to put ϕ ?

- dominators
- idoms
- dominance frontier $DF(x)$ for a given block x is the set of blocks such that some predecessor of y is dominated in the control flow graph by x, but y is **not**

E.g.

$$
\begin{array}{lll}
100 & DO\ I = 1, 10 & \\
S1: & & A[I] = A[I] + 10 \\
 & \ldots & \\
 & \ldots & \\
 & ENDO &
\end{array}
$$

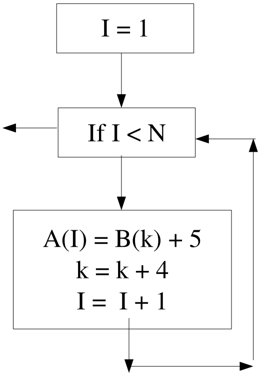
Where to put ϕ ?

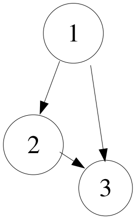
- dominators
- idoms
- dominance frontier $DF(x)$ for a given block x is the set of blocks such that some predecessor of y is dominated in the control flow graph by x, but y is **not**

E.g.

$$
\begin{array}{ll}
100 & DO\ I = 1, 10 \\
S1: & \quad A[I] = A[I] + 10 \\
 & .... \\
 & .... \\
 & ENDO
\end{array}
$$

SSA construction contd..





DF construction

1. Order - postorder traversal of dom tree l
2. while $l \neq \phi$; x first element of l
3. for all y successors of x if iDom(y) $\neq$ x then
   $DF(x) = DF(x) \bigcup y$
4. for all z st if x dom z
   for all y $\in DF(z)$
   if iDom(y) $\neq$ x **then** $DF(x) = DF(x) \bigcup y$

Now, the *phi*-function for definitions in block x must be placed in every block in $DF(x)$

What is Forward expression Substitution?

- Induction Variables
- Loop Invariant

$$
\begin{array}{ll}
 & DO\ I = 1, 10 \\
S1: & \quad K = I + a \\
S2: & \quad A[I] = A[K] + 5 \\
 & \quad .... \\
 & \quad ENDO
\end{array}
$$

$$
\begin{array}{ll}
 & DO\ I = 1, 10 \\
S2: & \quad A[I] = A[I + a] + 5 \\
 & \quad .... \\
 & \quad ENDO
\end{array}
$$

ForwardSub(Candidate stmt *S*, Loop *L*)

ForwardSub carries out substitution of every statement in L whose right-hand side variables include **only** the **loop variable** OR they are **invariant** in L

- Recognizing Auxiliary Induction Variables (In simplest form, these variables can be defined by statment like **K = K** $\pm$ **cexpr**)

$$
\begin{aligned}
&DO I = 1, N \\
&\quad A(I) = B(K) + 1 \\
&\quad K = K + 4 \\
&\quad \ldots\ldots\ldots \\
&\quad D(K) = D(K) + A(I) \\
&ENDO
\end{aligned}
$$

The ϕ-node is required on loop entry to merge the values of the loop induction variable initialized outside the loop with the values incremented inside the loop.

Induction Variable recognition Algorithm

boolean procedure isIV(**args**[])

1. Candidate statement should be part of cycle in SSA graph inside the loop involving only itself and a ϕ-node in loop header.
   If above is not true, isIV returns *False*.
2. If expression is of the form $\mathbf{K} = \mathbf{K} \pm \mathbf{cexpr}$, goto next step;
3. For all edges e coming into cexpr, if source(e) is inside current loop, then the cexpr is loop variant and isIV returns *false*.
   else we store all edges coming from outside of loop, in datastructure cexpr_edges, to be used later in IVSub algorithm.
4. Return *True*

Example of isIV() algorithm.

$$
\begin{array}{l}
DOI + 1, N \\
\quad A(I) = B(K + a) + 1 \\
\quad K = K + 4 \\
\quad \ldots\ldots\ldots \\
\quad D(K) = D(K + a) + A(I) \\
ENDO
\end{array}
$$

Induction Variable Substitution Algorithm.

procedure IVSub(**args**[])

1. **if not** isIV(args) then **return**;
2. Sh denotes the ϕ-node in the loop header for the iV under consideration, S0 is the source of the single edge to Sh from outside loop.
   For **each** edge e out of Sh to a node in same loop (target(e) before S),
   edit target expr and **update SSA edges**.
   For **each** edge e out of S to a node in the same loop (target(e) after S),
   edit target expr and **update SSA edges**.
3. If iV is used outside the loop, then move S outside loop. **Update the SSA** graph edges and **edit** cexpr.

Induction Variable Substitution Driver.

IVDrive performs forward substitution and induction-variable substitution on the loop L, recursively calling itself where necessary.

1. Induction variable should be performed from inner loop to outer loop.
2. Substituting loop-invariant expressionforward may create some new inducions, meaning it should be performed first.

Induction variable substitution when performed on an unnormalized loop, extremely ineffiecient code results.

References I

Chapter 4 of

Ken Kennedy and John R. Allen.
Optimizing compilers for modern architectures: a dependence-based approach.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.