A Solution to a Problem with Morel and Renvoise's "Global Optimization by Suppression of Partial Redundancies"

KARL-HEINZ DRECHSLER and MANFRED P. STADEL
Siemens AG

Morel and Renvoise have previously described a method for global optimization and code motion by suppression of partial redundancies [1]. Morel and Renvoise use data flow analysis to determine expression computations that should be inserted at the end of certain basic blocks and to determine redundant computations that can be eliminated. The execution of these techniques results in the movement of loop invariant expressions out of the loop. In addition to [1] Morel and Renvoise's techniques can also be applied to subexpressions of larger expressions. Then, however, in certain special cases these optimization techniques move expressions to places where some of its subexpressions are neither available nor moved together with the expression. In this paper we present a modification of Morel and Renvoise's algorithm that avoids the above described situations.

Categories and Subject Descriptors: D.3.4 [**Programming Languages**]: Processors—*compilers, optimization*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Code motion, data flow analysis

1. INTRODUCTION

This paper is very closely related to [1]; we use the same notations. However, we try to apply the methods given in [1] not only to entire expressions but also to all of their subexpressions. In Section 2 we present a problem raised by this extended application of [1]. We give an example where the computation of an expression e is inserted at the end of a basic block where some subexpression of e is neither available nor inserted. In Section 3 we propose a modification of Morel and Renvoise's data flow equations and prove that the modified equations insert a computation of an expression e only when all subexpressions of e are either available or also inserted. Finally, by semantically invariant transformation of the program flow graph, we show in Section 4 that the modified optimization technique performs at least as well as the original by Morel and Renvoise.

Authors' address: Siemens AG, ZTI SOF 22, Otto-Hahn-Ring 6, 8000 München 83, Federal Republic of Germany.

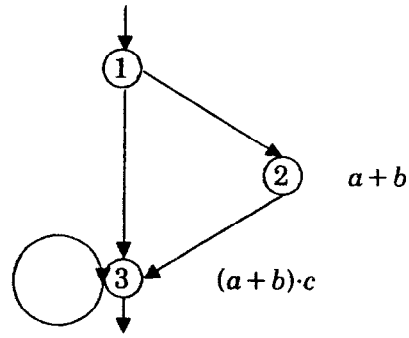

Fig. 1. An example flow graph.

Table I. Data Flow Analysis for the Flow Graph of Figure 1

expression	$a_1 = a + b$			$a_2 = a_1 \cdot c$		
basic block	1	2	3	1	2	3
PPIN	F	F	F	F	T	T
PPOUT	F	F	F	T	T	F
INSERT	F	F	F	T	F	F
DELETE	F	F	F	F	F	T

2. THE PROBLEM

Consider the program graph in Figure 1. Table I shows the largest solutions of the Boolean systems given in paragraph 3.1.1 in [1], where DELETE_i is defined by $\text{DELETE}_i = \text{PPIN}_i \cdot \text{ANTLOC}_i$. Therefore, a new computation of $a_2 = a_1 \cdot c$ should be inserted on exit from block 1, whereas a new computation of a_1 should not be inserted, and a_1 is not available on exit from block 1. The problem is raised by a block (block 2 in our example) that is empty for an expression e but is not empty for one of e's subexpressions.

3. MODIFICATION OF MOREL AND RENVOISE'S EQUATIONS

We propose to reduce the term $\text{CONST}_i(e)$ in [1] to

$$\text{CONST}_i(e) = \text{ANTIN}_i(e) \cdot \text{PAVIN}_i(e)$$

This modification does not affect the truth of Theorems 1 and 2 in [1]. After this modification, none of the computations of the expressions a_1 and a_2 of the example given in Section 2 is moved out of the loop.

At first glance, this seems to diminish the power of the algorithm since $\text{PPIN}_i(e)$ might be TRUE for fewer expressions e than by the original definition of the term $\text{CONST}_i(e)$. In Section 4 we show how this is avoided.

Another solution to our problem would be to use a larger $\text{CONST}_i(e)$ term such that $\text{PPIN}_i(e) = \text{TRUE}$ for more expressions e. In this case, more expressions would be moved to an earlier point in the program graph, and thereby the

number of registers needed for holding expression temporaries may be increased, probably leading to more spill code and hence to a reduction in execution speed.

THEOREM. *With the modified equation* $PPIN_i(e)$ *in* [1], *if the expression* f *is a subexpression of* e *and* f *is available at the point of each computation of* e, *then*

$$\mathrm{INSERT}_j(e) \text{ implies } \mathrm{AVOUT}'_j(f),$$

where AVOUT′ is the value of AVOUT after the insertions.

PROOF. Obviously, if f is a subexpression of e and f is available at the point of each computation of e, the following implications hold:

$$\begin{aligned}
\mathrm{TRANSP}_i(e) &\Rightarrow \mathrm{TRANSP}_i(f)\\
\mathrm{COMP}_i(e) &\Rightarrow \mathrm{AVOUT}_i(f)\\
\mathrm{ANTLOC}_i(e) &\Rightarrow \mathrm{ANTLOC}_i(f) + \mathrm{AVIN}_i(f)\\
\mathrm{AVIN}_i(e) &\Rightarrow \mathrm{AVIN}_i(f)\\
\mathrm{PAVIN}_i(e) &\Rightarrow \mathrm{PAVIN}_i(f)\\
\mathrm{ANTIN}_i(e) &\Rightarrow \mathrm{ANTIN}_i(f) + \mathrm{AVIN}_i(f)
\end{aligned}$$

Therefore, if $\mathrm{PPIN}_i(e) = \mathrm{TRUE}$ and $\mathrm{PPIN}_i(f) = \mathrm{FALSE}$, the only term of equation $\mathrm{PPIN}_i(f)$ in [1] which may be FALSE for f is $\mathrm{ANTIN}_i(f) \cdot \mathrm{ANTLOC}_i(f)$. But in this case we have $\mathrm{AVIN}_i(f) = \mathrm{TRUE}$, and then $\mathrm{AVOUT}_j(f) = \mathrm{TRUE}$ for all $j \in \mathrm{Pred}(i)$. Therefore, $\mathrm{PPIN}_i(f) = \mathrm{FALSE}$ does not propagate through the terms $(\mathrm{PPOUT}_j(f) + \mathrm{AVOUT}_j(f))$ to the successors of i. This proves the implication:

$$\mathrm{PPOUT}_i(e) \Rightarrow \mathrm{PPOUT}_i(f) + \mathrm{AVOUT}_i(f).$$

And with the definition of INSERT and Lemma 1 in [1] we have:

$$\mathrm{INSERT}_i(e) \Rightarrow \mathrm{PPOUT}_i(e) \Rightarrow \mathrm{AVOUT}'_i(f) \qquad \square$$

From the proof we get the following corollary, which is used in Section 4.

COROLLARY. *The theorem remains true if for some of the basic blocks of the flow graph the factor* $PAVIN_i(e)$ *is dropped from the CONST term such that* $CONST_i(e) = ANTIN_i(e)$.

4. COMPENSATION OF THE DEFICIENCY OF THE NEW METHOD

To compensate for the deficiency of the method due to the change in the term $\mathrm{CONST}_i(e)$, we conceptionally insert an empty block ij in each edge from a block i to a block j. Obviously, the extended flow graph is semantically equivalent to the original flow graph.

For the empty block ij we use the original $\mathrm{CONST}_i(e)$ term of [1], whereas for the block i of the original flow graph, we use the modified term $\mathrm{CONST}_i(e) = \mathrm{ANTIN}_i(e) \cdot \mathrm{PAVIN}_i(e)$.

Equations given in paragraphs 2.1.2 and 3.1.1 in [1] for the empty blocks ij reduce (we drop the argument (e) for the sake of clarity) to:

(1) $\text{AVIN}_{ij} = \text{AVOUT}_{ij} = \text{AVOUT}_i$,
(2) $\text{ANTOUT}_{ij} = \text{ANTIN}_{ij} = \text{ANTIN}_j$,
(3) $\text{PAVIN}_{ij} = \text{PAVOUT}_{ij} = \text{PAVOUT}_i$.
(4) $\text{PPOUT}_{ij} = \text{PPIN}_j$,
(5) $\text{PPIN}_{ij} = \text{ANTIN}_j \cdot \text{PPIN}_j \cdot (\text{PPOUT}_i + \text{AVOUT}_i)$

(PPIN_{ij} is calculated as indicated in paragraph 3.1.1 in [1].)

Substituting (5) in the equation for $\text{PPIN}_j(e)$ in [1] leads to

$$\text{PPIN}_j = \text{ANTIN}_j \cdot \text{PAVIN}_j \cdot (\text{ANTLOC}_j + \text{TRANSP}_j \cdot \text{PPOUT}_j) \cdot \prod_{i \in \text{Pred}(j)} (\text{PPIN}_j + \text{AVOUT}_i)$$

and the largest solution for PPIN_j satisfies

(6) $\text{PPIN}_j = \text{ANTIN}_j \cdot \text{PAVIN}_j \cdot (\text{ANTLOC}_j + \text{TRANSP}_j \cdot \text{PPOUT}_j)$

From this equation we get the implication $\text{PPIN}_j \Rightarrow \text{ANTIN}_j$, and therefore we can drop the term ANTIN_j in Eq. (5):

(7) $\text{PPIN}_{ij} = \text{PPIN}_j \cdot (\text{PPOUT}_i + \text{AVOUT}_i)$

Substituting (7) in the equation $\text{PPOUT}_i(e)$ in [1] for a nonexit block i leads to

$$\text{PPOUT}_i = (\text{PPOUT}_i + \text{AVOUT}_i) \cdot \prod_{j \in \text{Succ}(i)} \mathit{PPIN}_j$$

and a largest solution for PPOUT_i satisfies

$$\text{(8)} \qquad \text{PPOUT}_i = \begin{cases} \text{FALSE} & \text{if } i \text{ is an exit block} \\ \prod_{j \in \text{Succ}(i)} \text{PPIN}_j & \text{otherwise} \end{cases}$$

Equations (6) and (8) build a new system of equations not depending on the inserted empty blocks. As a side-effect, we got a simpler system which, in general, can be solved with less iterations than the original one. We search for a largest solution of this new system and then we determine INSERT_{ij} by substituting (4) and (7) in the equation for INSERT_{ij} given in [1]:

(9) $\text{INSERT}_{ij} = \text{PPIN}_j \cdot \overline{\text{PPOUT}_i} \cdot \overline{\text{AVOUT}_i}$

So we really need an empty block in an edge between block i and block j if $\text{INSERT}_{ij} = \text{TRUE}$ for some expressions e; all other empty blocks are only conceptional.

Since $\text{ANTLOC}_i(e) \cdot \text{PAVIN}_i(e) \Rightarrow \text{ANTLOC}_i(e) \cdot \text{PPIN}_i(e) \Rightarrow \text{DELETE}_i(e)$, all partially redundant computations of an expression e are deleted, which is the best we can have. Therefore, the original algorithm of Morel and Renvoise does not delete more computations than our modified version does.

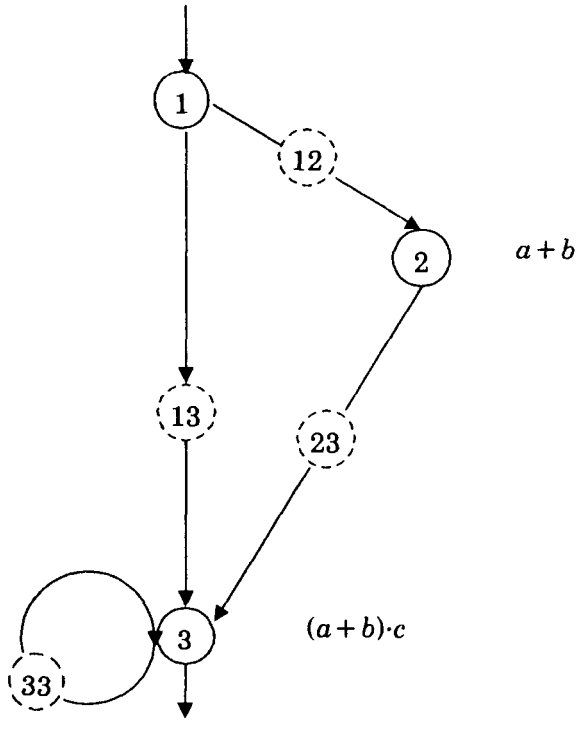

Fig. 2. Empty blocks inserted in flow graph in Figure 1.

Table II. Data Flow Analysis for the Flow Graph of Figure 2

expression	$a_1 = a + b$							$a_2 = a_1 \cdot c$						
basic block	1	2	3	12	13	23	33	1	2	3	12	13	23	33
PPIN	F	F	T	F	F	T	T	F	F	T	F	F	F	T
PPOUT	F	T	F	F	T	T	T	F	T	F	F	T	T	T
INSERT	F	F	F	F	T	F	F	F	T	F	F	T	F	F
DELETE	F	F	T	F	F	F	F	F	F	T	F	F	F	F

In case $\text{INSERT}_{ij} = \text{TRUE}$ for all $i \in \text{Pred}(j)$, insertion should be done at the entry to block j rather than in each empty block ij. From Eq. (9) we derive

$$\text{INSERTIN}_j = \text{PPIN}_j \cdot \prod_{i \in \text{Pred}(j)} \overline{(\text{PPOUT}_i + \text{AVOUT}_i)} \tag{10}$$

Insertion at the entry of block j is done iff $\text{INSERTIN}_j \cdot \overline{ANTLOC_j} = \text{TRUE}$; deletion of the first computation in block j is done iff $\text{DELETE}_j \cdot \overline{\text{INSERTIN}_j} = \text{TRUE}$.

In Figure 2 and Table II we demonstrate our idea on the example given in Figure 1. The new empty block 13 can be used for inserting loop invariant computations of block 3 that would not be inserted in block 1.

Some other examples for situations where the insertions of empty blocks repair the deficiency due to the modification of the term $\text{CONST}_i(e)$ are given in Figures 3 and 4.

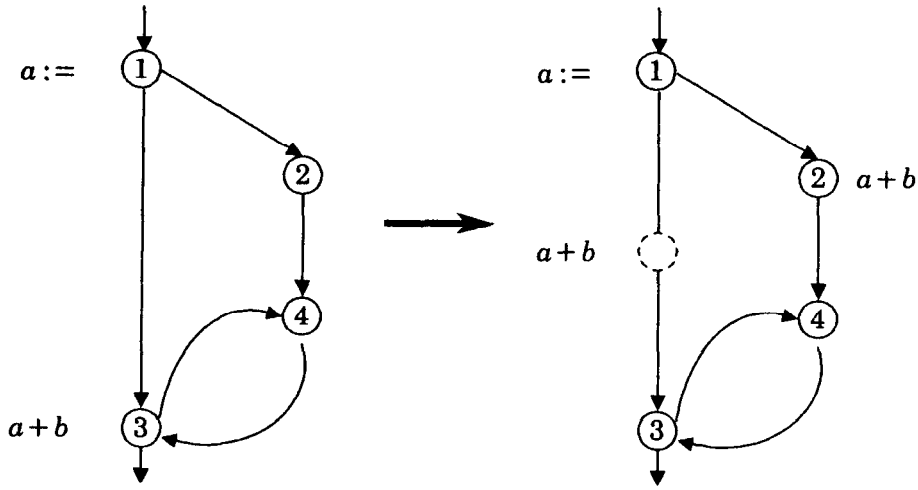

Fig. 3. Moving invariants out of an irreducible loop.

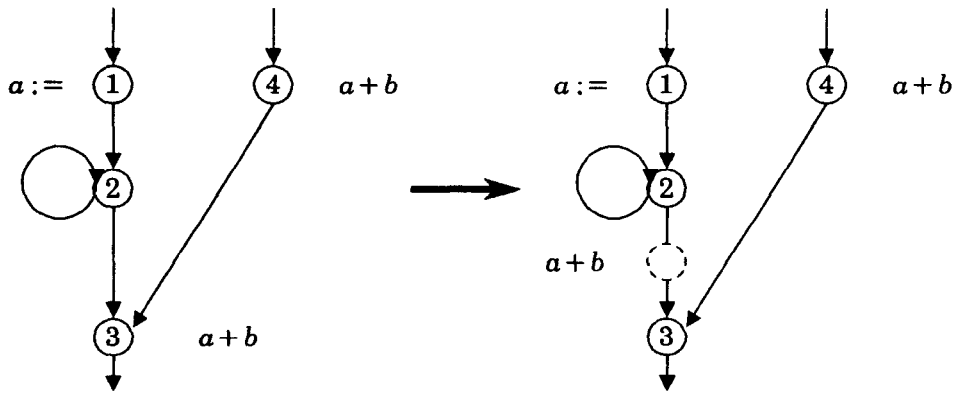

Fig. 4. Avoiding propagation through a loop.

Unfortunately, the method described here still has a deficiency: under certain circumstances additional jump instructions due to empty block insertion may be needed.

ACKNOWLEDGMENTS

We thank the referees of our earlier version of this paper for their useful comments and for their suggestion to search for a repair of the deficiency due to the change in the term $\mathrm{CONST}_i(e)$.

REFERENCE

1. Morel, E., and Renvoise, C. Global optimization by suppression of partial redundancies. *Commun. ACM* 22, 2 (1979), 96–103, 111–126.

Received November 1985; revised November 1986 and May 1988; accepted June 1988