Pros & Cons of Partial Redundancy Elimination Algorithms

RAHIBB[1], Dr. S SARALA[2]
ResearchScholar[1], Assistant Professor & Supervisor[2]
[*]Department of Information Technology
[*]Bharathiar University, Coimbatore, India
rahibb007@gmail.com[1], sarala.bu@gmail.com[2]

Abstract-Partial Redundancy Elimination is an optimization method that eliminates expressions that are redundant on some program execution paths but not necessarily all program paths in a program. In this paper we try to discuss pros and cons of two well known simple algorithms for Partial Redundancy Eliminations. But both the algorithms do not give much importance for eliminating edge splitting, even though the edge splitting is much more expensive than inserting a computation at a node which is already exists in a data flow graph. In this paper we suggest the possibilities for eliminating edge splitting as far as possible to make both the algorithms more compact and attractive.

Keywords: Partial Redundancy Elimination, Data Flow Graph, Availability, Anticipability, Safe Partial Availability, Safe Partial Anticipability, E_path suffix.

1. INTRODUCTION

Partial redundancy elimination (PRE) is a program transformation that removes op ll. erations that are redundant on some execution paths, but not aSuch a transformation requires the partially redundant operation to be hoisted to earlier programpoints where the operation's value was not previously available. A Partially Redundant Expression(PRE) algorithm is a compiler optimization technique for changing partial redundancy of an expression in a DFG into fully redundancy and eliminate the redundancy. A PRE algorithm based on safe insertions is considered to be optimal if no other PRE algorithm which uses safe insertions gives a DFG which contains fewer computations along any path.

Morel And Renvoise (MRA)[1] proposed a bidirectional data flow analysis algorithm to eliminate partial redundancies which does not eliminate all partial redundancies in a program, and it lacks both computational and life time optimality as well. Since MRA fails to split edges, optimization is not possible in many loops. Even though Dhamdhere through Edge Placement Algorithm(EPA) does insertions both in nodes and on edges in DFG [2], he could not completely eliminate redundant code motion. EPA does not provide life time optimality in many cases.

2. PRE ALGORITHM BY VINEETH KUMAR

Vineeth Kumar's algorithm called "a simple, pragmatic, and provably correct algorithm" [3] for PRE is really simple, and computationally and lifetime optimal. The algorithm assumes that all local redundancies are already eliminated by some standard techniques for common subexpression elimination on basic blocks[4]. The algorithm is based on the concepts of availability, anticipability, safe partial availability, and safe partial anticipability.

The Table 1 summarizes the data flow properties and equations of the algorithm. Let e be an expression in a nodei of a data flow graph G. The local data flow property ANTLOCi represents a locally anticipated upwards exposed e in nodei, COMPi represents a locally available downwards exposed e in nodei, and TRANSPi reflects the absence of assignments to the operand(s) of e in nodei. Global properties of availability, anticipability, safe partial availability, safe partial anticipability are used to collect global information. INSERTi and INSERT(i,j), identify e to be inserted in nodei, and on edge(i,j) respectively, and REPLACEi identifies e to be replaced in nodei with a temporary variable, say t.

Table 1 : PRE: a simple, pragmatic, and provably correct algorithm

Local data flow properties	
ANTLOCi	: nodei contains a locally anticipated upwards exposed e.
COMPi	: nodei contains a downwards exposed e.
TRANSPi	: nodei does not contain an assignment to any of the

operands of e.

Global data flow Properties

$AVIN_i/AVOUT_i$	: e is available at the entry/exit of node$_i$.
$ANTIN_i/ANTOUT_i$	: e is anticipated at the entry/exit of node$_i$.
$SAFEIN_i/SAFEOUT_i$	: e is safe at the entry/exit of node$_i$.
$SPAVIN_i/SPAVOUT_i$	: e is safe partially available at the entry/exit of node$_i$.
$SPANTIN_i/SPANTOUT_i$ the	: e is safe partially anticipated at entry/exit of node$_i$.
$REDUND_i$	: e is redundant at the entry of node$_i$.
$SPREDUND_i$	: e is safe partially redundant at the entry of node$_i$.
$ISOLATED_i$	: e is isolated in node$_i$.

Data flow equations

$AVIN_i$ = False if i= start node,
Otherwise $\prod_{\text{[illegible]}(i)} AVOUT_j$

$AVOUT_i = COMP_i + AVIN_i.TRANSP_i$

$ANTIN_i = ANTLOC_i + ANTOUT_i.TRANSP_i$

$ANTOUT_i$ = False if i = exit node,
otherwise $\prod_{\text{[illegible]}(i)} ANTIN_j$

$SAFEIN_i = AVIN_i + ANTIN_i$

$SAFEOUT_i = AVOUT_i + ANTOUT_i$

$SPAVIN_i$ = False if i = start node or $\neg SAFEIN_i$,
Otherwise $\sum_{\text{[illegible]}(i)} SPAVOUT_j$

$SPAVOUT_i$ = False if $\neg SAFEOUT_i$,
Otherwise $COMP_i+SPAVIN_i.TRANSP_i$

$SPANTIN_i$ = False if $\neg SAFEIN_i$,
Otherwise $ANTLOC_i+SPANTOUT_i.TRANSP_i$

$SPANTOUT_i$ = False if i = exit node or $\neg SAFEOUT_i$,
Otherwise $\sum_{\text{[illegible]}(i)} SPANTIN_j$

$SPREDUND_{if} = SPAVIN_i.ANTLOC_i \quad REDUND_{if}$
$= ANTLOC_i.AVIN_i$

$REDUND_{ii} = COMP_i.AV_p$, p is the point just before the last
computation of e in node$_i$

$ISOLATED_{if} = \neg SPAVIN_i.ANTLOC_i \neg(TRANSP_i.SPANTOUT_i)$

$ISOLATED_{ii} = COMP_i.\neg SPANTOUT_i.\neg(TRANSP_i.SPAVIN_i)$

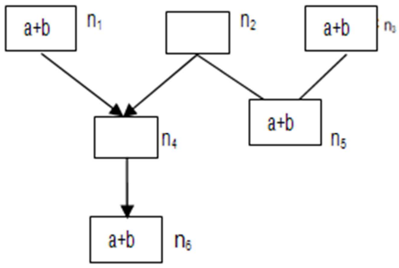
$INSERT_i = COMP_i.SPANTOUT_i \neg(TRANSP_i.SPAVIN_i)$

$INSERT_{(i,j)} = \neg SPAVOUT_i SPAVIN_j SPANTIN_j$

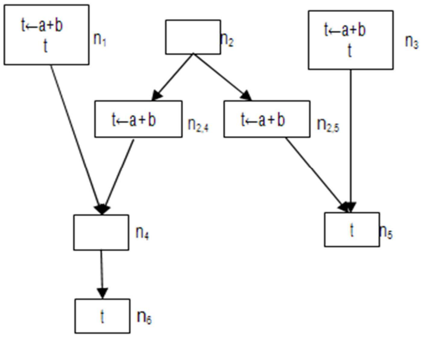
$REPLACE_{if} = ANTLOC_i(SPAVIN_i+TRANSP_i.SPANTOUT_i) \quad REPLACE_{ii}$
$= COMP_i(SPANTOUT_{ii}+TRANSP_i.SPAVIN_i)$

A. An Example

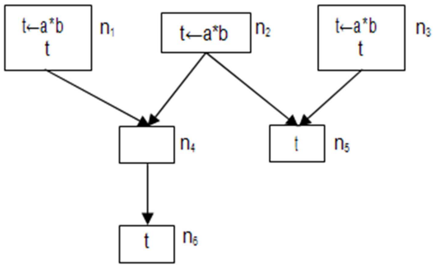
Fig.1(a) shows a DFG with 6 nodes, and Fig.1(b) is DFG after applying the PRE algorithm. Here the algorithm splits the edges (n_2,n_4) and (n_2,n_5) for inserting a node in each edge with the equation: $INSERT(i,j) = \neg SPAVOUT_i.SPAVIN_j.SPANTIN_j$. But edge splitting is much more expensive than inserting a computation in an existing node. But the algorithm inserts a computation for an expression in a node$_i$ only if $COMP_i$ is true: $INSERT_i=COMP_i.SPANTOUT_i(\neg TRANSP_i+\neg SPAVIN_i)$.In this example $COMP_2$= False. But if insertion is done at the node n_2, the edge splitting can be eliminated as shown in Fig.1(c).



(a) before PRE Algorithm



(b) after PRE Algorithm



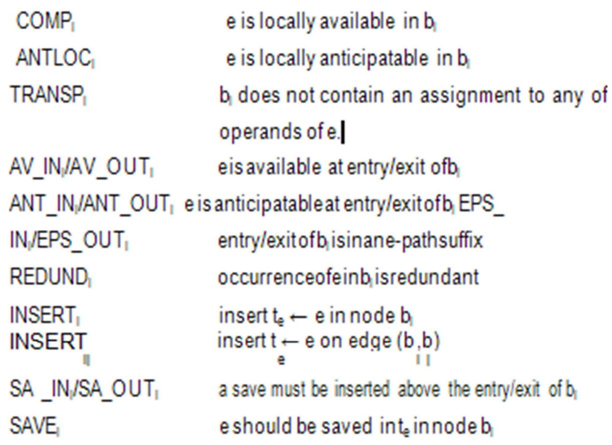
(c)after elimination edge splitting

Fig.1. Partial Redundancy Elimination

3. AN E-PATH_PRE ALGORITHM BY DM DHAMDHERE

DM Dhamdhere presented a PRE algorithm titled “E-Path_PRE – Partial Redundancy Elimination Made Easy” [5]. The algorithm first identifies the insertion points at nodes and on edges and then identify the saves points, and finally the redundant occurrences of an expression for replacement. The Table 2 summarizes the data flow properties and equations of the algorithm. Let e be an expression in a nodei of a data flow graph G. The local data flow property ANTLOCi represents a locally anticipated upwards exposed e in nodei, COMPi represents a locally available downwards exposed e in nodei, and TRANSPi reflects the absence of assignments to the operand(s) of e in nodei. Global properties of availability, anticipability and E-path suffix are used to collect global information. INSERTi and INSERTi,j identify e to be inserted in nodei, and on edge(i,j) respectively, and SAVEi identifies the node bi in which e should be saved.

Table 2 : E-PATH PRE



$COMP_i$	e is locally available in b_i
$ANTLOC_i$	e is locally anticipatable in b_i
$TRANSP_i$	b_i does not contain an assignment to any of operands of e.
AV_IN_i/AV_OUT_i	eisavailable atentry/exit ofb_i
ANT_IN_i/ANT_OUT_i	eisanticipatableatentry/exitofb_i EPS_
IN_i/EPS_OUT_i	entry/exitof$b_i$isinane-pathsuffix
$REDUND_i$	occurrenceofeinb$_i$isredundant
$INSERT_i$	insert $t_e \leftarrow$ e in node b_i
$INSERT_{ij}$	insert $t_e \leftarrow$ e on edge (b_i,b_j)
SA_IN_i/SA_OUT_i	a save must be inserted above the entry/exit of b_i
$SAVE_i$	e should be saved in$t_e$innode b_i

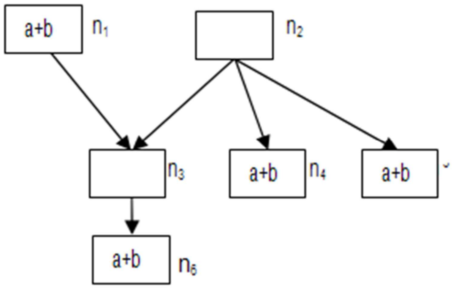
Data flow equations

$AV\_IN_i = \prod_{\square\in\square\square\square\square(\square)} AV\_OUT_p$
$AV\_OUT_i = AV\_IN_i.TRANSP_i +$
$COMP_i$ $ANT\_IN_i =$
$ANT\_OUT_i. TRANSP_i+$
$ANTLOC_i$
$ANT\_OUT_i = \prod_{\square\in\square\square\square\square(\square)} ANT\_IN_s$
$EPS\_ IN_i = \sum_{\square\in\square\square\square\square(\square)}(AV\_OUT_p+$
$EPS\_OUT_p).ANT\_IN_i.\neg AV\_IN_i$ $EPS\_OUT_i =$
$EPS\_ IN_i. \neg ANTLOC_i$
$REDUND_i = (AV\_IN_i + EPS\_ IN_i).ANTLOC_i$
$INSERT_i = \neg AV\_OUT_i. .\neg EPS\_OUT_i.$
$\prod_{\square\in\square\square\square\square(\square)} EPS\_ IN_s$ $INSERT_{ij} =$
$\neg INSERT_i \neg AV\_OUT_i..\neg EPS\_OUT_i.$
EPS_IN_j
$SA\_OUT_i = \sum_{\square\in\square\square\square\square(\square)}(EPS\_ IN_s+ REDUND_s+ SA\_IN_s).$
AV_OUT_i
$SA\_IN_i = SA\_OUT_i. \neg COMP_i$
$SAVE_i = SA\_OUT_i.COMP_i \neg (REDUND_i. TRANSP_i)$

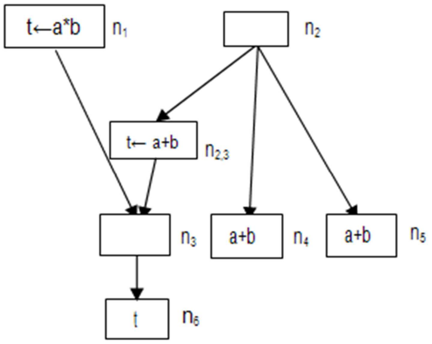
A.An Example

Consider the control flow graph of Fig.2(a) consisting of 6 nodes. Here the E_Path_PRE is n1-n3-n6. So the E_PATH_PRE algorithm saves the expression a+b at n1 in a temporary variable t and replaces it in the node n6 with t. Since $\prod_{\square\in\square\square\square\square(\square\square)}$ EPS_ INs = False for the node n2, insertion of the computation at node n2 is not possible according to the E_Path_PRE algorithm. But insertion on the edge (n2,n3) is possible since INSERT23 is true as shown in the Fig.2(b). Here itself if the insertion is done at the node n2, the edge splitting can be eliminated as shown in Fig.2(c).

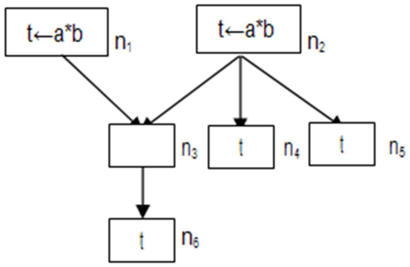
Fig.2(c). According to the Lemma 3: "An expression e in node bj is deleted if and only if e is available at entry to bj in the optimized program". In Fig.2(c) the expression a+b is available at the entry of nodes n4,n5 and n6 , and hence the expression a+b from them are deleted. Insertion at node n2 also replaces isolated expressions from the nodes to some extend without sacrificing the computational and life time optimality of the E_Path_PRE algorithm. The expression a+b at nodes n4 and n5 in Fig.2(a) are the isolated expressions because the E- pth PRE algorithm cannot form E_paths for them. However, they are deleted by inserting the expression at n2 as shown in Fig.2(c), a bonus.



(a) before PRE Algorithm



(b) after PREAlgorithm



(c) after eliminating edge splitting
Fig.2. Partial Redundancy Elimination

4. CONCLUSION

A Simple Algorithm for Partial Redundancy Elimination called “PRE: a simple, pragmatic, and provably correct algorithm”., by Vineeth Kumar, YN Srikant and Priti Shankar, and an E_path_PRE algorithm, by DM Dhamdhere do not take care of eliminating edge splitting much. In this paper we suggested a method for avoiding edge splitting as far as possible to make the PRE algorithm more compact without sacrificing the algorithm’s computational and lifetime optimality with the help of 2 examples.

It is already shown by Vineeth Kumar and DM Dhamdhere that the algorithms given by the papers [1],[4], and [6-10] have one or more of the problems of redundant code motion, unremoved redundancies, or limited applicability due to reducibility restriction of the control flow graph.

REFERENCE

[1] E. Morel and C. Renvoise, “Global optimization by suppression of partial redundancies,” Communications of the ACM, vol. 22, no. 2, pp. 96-103, 1979.
[2] D. M. Dhamdhere. A fast algorithm for code movement optimization. SIGPLAN Notices, 23(10): 172–180, 1988.
[3] V. K. Paleri, Y. N. Srikant, and P. Shankar, “Partial redundancy elimination: a simple, pragmatic, and provably correct algorithm,” Science of Computer Programming, vol. 48, no. 1, pp. 1-20, 2003.
[4] A.V. Aho, R. Sethi, J.D. Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley,
[5] D. M. Dhamdhere. E-path PRE—partial redundancy elimination made easy. ACM SIGPLAN Notices, 37(8): 53–65, 2002.
[6] D. M. Dhamdhere. Practical adaptation of global optimization algorithm by Morel & Renvoise.ACM Transactions on Programming Languages and Systems, 13(2):291–294, 1991.
[7] D. M. Dhamdhere and U. P. Khedker. Complexity of bidirectional data flows. Proceedings of Twentieth annual symposium on POPL, pages 397–408, 1993.
[8] D. M. Dhamdhere and H. Patil. An elimination algorithm for bi-directional data flow analysis using edge placement technique. ACM TOPLAS, 15(2):312–336, 1993.
[9] D. M. Dhamdhere, B. K. Rosen, and F. K. Zadeck. How to analyze large programs efficiently and informatively. Proceedings of ACM SIGPLAN’92 Conference on PLDI, pages 212–223, 1992.
[10] V. M. Dhaneshwar and D. M. Dhamdhere. Strength reduction of large expressions. Journal of Programming Languages, 3:95–120, 1995.