A Refinement on E-Path Partial Redundancy Elimination

Rahibb, S Sarala

Abstract: ***Partial redundancy elimination algorithm is a compiler optimization method that eliminates expressions that are redundant on some programming path but not necessarily all paths in a Data Flow Graph(DFG) of a program. The E-Path Partial Redundancy Elimination) algorithm authored by DM Dhamdhere for Partial Redundancy Elimination(PRE) of common subexpression elimination does not give much importance to the elimination of edge splitting, even though the edge splitting is much more expensive than inserting an expression in an existing node of a DFG of a program. So in this paper we try to refine the E-Path PRE algorithm with a new equation for inserting expressions at nodes avoiding edge splitting as far as possible and hence the E-Path PRE algorithm becomes more compact and beautiful.***

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

I. INTRODUCTION

The redundancy of an expression in a program can exist in the form of common subexpression, a loop-invariant expression, and it can be partial redundancy too, if it is found along some of the paths, but not necessarily along all paths. A Partially Redundant Elimination (PRE) algorithm is an optimization technique for transforming partial redundancy of an expression in a program into fully redundancy and eliminate the redundancy. A PRE algorithm is considered to be optimal if no other PRE algorithm gives a data flow graph which contains fewer computations (less insertions and more deletions) in any path in the data flow graph.

The Morel E and Renvoise C [1] first proposed a bi-directional algorithm for code optimization in compilers in 1979, by suppressing partial redundancies such as moving loop invariant computation out of a loop or deleting redundant computations. The algorithm is referred to as MRA. The MRA algorithm was updated by DM Dhamdhere and SM Joshi [2] by including strength reduction techniques. However, when there is a loop invariant expression that cannot be moved to a node out of the loop, the MRA fails, because it performs insertions strictly in nodes of a data flow graph, and it does not support the edge splitting at all. And the MRA lacks both computational and life time optimality also.

Revised Manuscript Received on 30 May 2019.
* Correspondence Author
Rahibb*, Research Scholar,, Department of Computer Applications, Bharathiar University, Coimbatore, Tamil Nadu, India
Dr. S Sarala, Assistant Professor(SS), Department of Computer Applications, Bharathiar University, Coimbatore, Tamil Nadu, India.

The Edge Placement Algorithm [3] by DM Dhamdhere, called EPA, performs insertions both in nodes and along edges in a DFG. In this algorithm an expression is hoisted as far up as possible to obtain computational optimality, and then it is subjected to sinking to get lifetime optimality without sacrificing computational optimality. However EPA does not provide lifetime optimality in some cases.

The research papers [4] and [5] developed for enhancing PRE algorithm to eliminate partial redundancies of expressions in a computer program.

The problem lies with the paper [4], hoisting-by-sinking, is that the conceptual complexity is so high that it is hard to understand and implement in a compiler. There are research papers [6] , [7] and [10] that do not use hoisting-by-sinking method. However, the paper [6] suffers from the unnecessary edge splitting. But unlike the paper [11] says, the paper [10] does avoid the edge splitting by using a conditional statement in its algorithm.

The PRE algorithm used in the paper [7] is used in the text book [8] and in [9] though, the splitting of the edges before the analysis of the program results in unnecessary edge splitting.

DM Dhamdhere [11] proposed a unidirectional data flow analysis algorithm for partial redundancy elimination which is computationally and lifetime optimal. Though the edge splitting is more expensive than inserting an expression at an existing node [3],[11], the E-Path PRE algorithm does not give much care for eliminating them. In this paper we give an alteration to the insert equation at nodes of the E-Path PRE algorithm to remove the edge splitting of a DFG as much as possible and hence to make the algorithm more beautiful and attractive.

II. E-PATH_PRE ALGORITHM BY DM DHAMDHERE

DM Dhamdhere presented the E-Path algorithm [11]. 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 1 summarizes the data flow properties and equations of the algorithm. Let e be an expression in a $node_i$ of a DFG. The local data flow property $anticip\_loc_i$ represents a locally anticipated upwards exposed e in $node_i$, $compute_i$ represents a locally available downwards exposed e in $node_i$, and $trans_i$ reflects the absence of assignments to the operand(s) of e in $node_i$. Global properties of availability, anticipability and E-path suffix are used to collect global information. $insert_i$ and $insert_{i,j}$ identify e to be inserted in $node_i$, and on edge(i,j) respectively, and $save_i$ identifies the node b_i in which e should be saved.

$compute_i$	:	e is nearby existing in b_i
$anticip\_loc_i$	:	e is nearby anticipatable in b_i
$trans_i$	:	b_i does not include an assignment to any of the operands of e
$avail\_in_i/avail\_out_i$	:	e is existing at entry/exit of b_i
$anticip\_in_i/anticip\_out_i$	:	e is anticipatable at entry/exit of b_i
$e\_ps\_in_i/e\_ps\_out_i$	:	entry/exit of b_i is in an e-path suffix
$redund_i$	:	incidence of e in b_i is redundant
$insert_i$	:	insert $temp_e \leftarrow$ e in node b_i
$insert_{ij}$	:	insert $temp_e \leftarrow$ e on edge (b_i,b_j)
$save\_in_i/save\_out_i$	:	e must be kept above the entry/exit of b_i
$save_i$	:	e should be kept in $temp_e$ in node b_i

Table 1 : E-PATH Partial Redundancy Elimination

Data Flow Properties

Data Flow Equations

$avail\_in_i$	:	$\prod_{p \in pred(i)} av\_out_p$
$avail\_out_i$	:	$avail\_in_i.trans_i + compute_i$
$anticip\_in_i$	:	$anticip\_out_i.\ trans_i + anticip\_loc_i$
$anticip\_out_i$	:	$\prod_{s \in succ(i)} anticip\_in_s$
$e\_ps\_in_i$	:	$\sum_{p \in pred(i)} (avail\_out_p + e\_ps\_out_p) anticip\_in_i \neg avail\_in_i$
$e\_ps\_out_i$	:	$e\_ps\_in_i.\ \neg anticip\_loc_i$
$redund_i$	:	$(avail\_in_i + e\_ps\_in_i).anticip\_loc_i$
$insert_i$	:	$\neg avail\_out_i.\ .\neg e\_ps\_out_i.\ \prod_{s \in succ(i)} e\_ps\_in_s$
$insert_{ij}$	:	$\neg insert_i\ \neg avail\_out_i.\ .\neg e\_ps\_out_i.\ e\_ps\_in_j$
$save\_out_i$	:	$\sum_{s \in succ(i)} (e\_ps\_in_s + redund_s + save\_in_s).\ avail\_out_i$
$save\_in_i$	:	$save\_out_i.\ \neg compute_i$
$save_i$	:	$save\_out_i.compute_i \neg (redund_i.trans_i)$

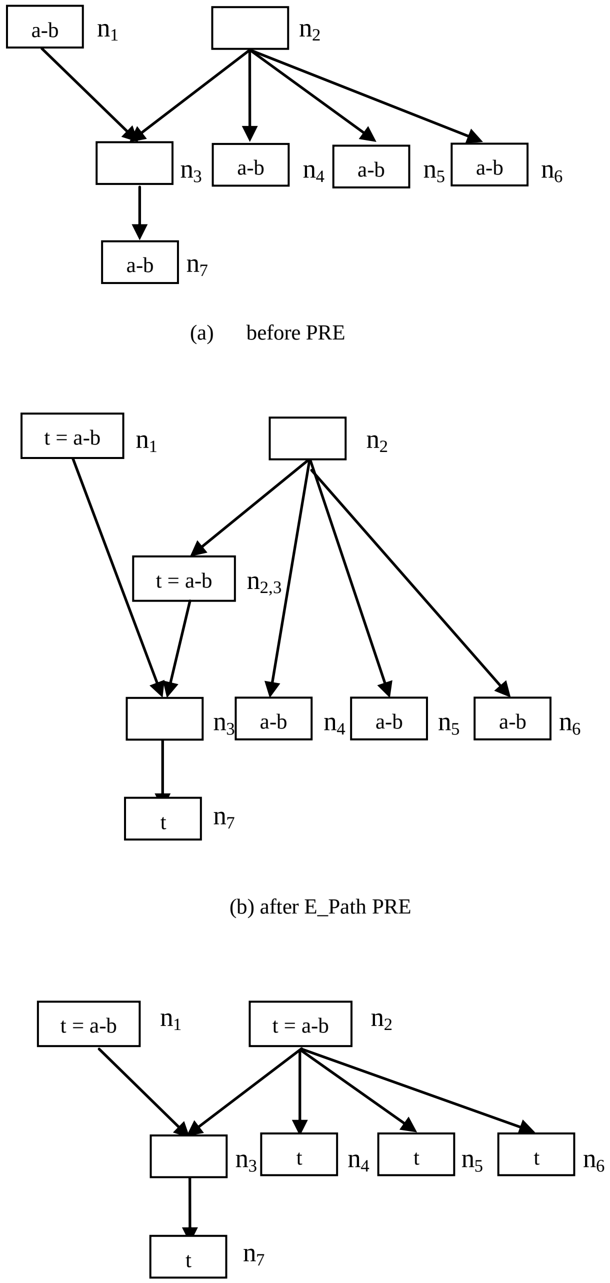
III. A REFINEMENT

Consider the control flow graph of Fig.1(a) consisting of 7 nodes. Here the E_Path_PRE is n_1-n_3-n_7. So the E_PATH_PRE algorithm saves the expression a*b at n_1 in a temporary variable t and replaces it in the node n_7 with t. Since $\prod_{s \in succ(n2)} E\_PS\_IN_s$ = False for the node n_2, insertion of the computation at node n_2 is not possible according to the E-Path PRE algorithm. But insertion on the edge (n_2,n_3) is possible since $INSERT_{23}$ is true as shown in the Fig.1(b). But if we apply the new equation shown in the Table 2, we get the Fig.1(c). According to the lemma III in the E_Path PRE algorithm an expression in a node can be eliminated if and only if that expression is available at beginning of that node in the optimized program. In Fig.1(c) the expression a-b is available at the entry of nodes n_4,n_5, n_6 and n_7, and hence the expression a-b from them are deleted. The application of the new INSERT equation has 2 advantages. One is that it eliminates the edge splitting as much as possible, and the second is it 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 n_4, n_5, and n_6, in Fig.1(a) are the isolated expressions because the E-Path PRE algorithm cannot form E-paths for them. However, they are deleted by the new equation as shown in Fig.1(c).

Table 2 : A Refinement on E_Path PRE Algorithm

$$INSERT_i = \neg AV\_OUT_i . \neg EPS\_OUT_i . \sum_{s \in succ(i)} EPS\_IN_s \prod_{s \in succ(i)} ANT\_IN_s$$

All other equations remain the same as in the PRE algorithm.



(a) before PRE

(b) after E_Path PRE

(c) after applying new equation

Fig.1. Partial Redundancy Elimination

IV. CONCLUSION

The E-Path PRE algorithm, by DM Dhamdhere did not take care of eliminating edge splitting much, even though the edge splitting is much more expensive than inserting a computation in an existing node. In this paper we added much care for avoiding edge splitting as far as possible to make the E-Path PRE algorithm more refined and compact. In this paper we updated the $INSERT_i$ equation to eliminate the edge splitting as mcuh as possible. And the refined algorithm is also computationally and lifetime optimal.

ACKNOWLEDGMENT

We are bound to thank Dr. Vineeth Kumar Paleri, Professor, Department of Computer Science, NIT Calicut, Kerala, India, for his sincere, unconditional and constant guidance for our research work, and we would like to thank the University Grants Commission too, for awarding Teacher Fellowship for completing Ph.D. in Computer Science under the Faculty Development Programme of the UGC during the XII[th] plan period (Letter No. F. No. FIP/12th Plan/KLCA045 TF07, dated: 10-09-2016).

REFERENCES

1. 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. 2. S .M. Joshi, D.M. Dhamdhere : A composite hoisting - strength reduction transformation for global program optimisation - Parts I and II, International Journal of Computer Mathematics,11 (1982),21-41; 111-126 .
3. D. M. Dhamdhere. A fast algorithm for code movement optimization. SIGPLAN Notices, 23(10): 172–180, 1988.
4. V. M. Dhaneshwar and D. M. Dhamdhere. Strength reduction of large expressions. Journal of Programming Languages, 3:95–120, 1995.
5. U. P. Khedker and D. M. Dhamdhere. Bidirectional data flow analysis : Myths and reality. SIGPLAN Notices, 34(6):47–57, 1999.
6. R. Bodik, R. Gupta, and M. L. Soffa. Complete removal of redundant expressions. Proceedings of ACM SIGPLAN '98 Conference on PLDI, pages 1–14, June 1998.
7. J. Knoop, O. Ruthing, and B. Steffen. Optimal code motion: theory and practice. ACM TOPLAS, 30(4):1117–1155, 1994.
8. A.V. Aho, R. Sethi, J.D. Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley,8[th] impression: 2012
9. Sandeep Dasgupta, Tanmay Gangwani, Partial Redundancy Elimination using Lazy Code Motion, Academic Project, pages 1-19, May 11, 2014
10. V. K. Paleri, Y. N. Srikant, and P. Shankar. A simple algorithm for partial redundancy elimination. Sigplan Notices, 33(12):35–43, 1998.
11. D. M. Dhamdhere. E-path PRE—partial redundancy elimination made easy. ACM SIGPLAN Notices, 37(8):53–65, 2002.