A Supplement to "PRE: A Simple, Pragmatic, and Provably Correct Algorithm"

Rahibb and S. Sarala

Abstract A partial redundancy elimination (PRE) is a compiler optimization that eliminates expressions that are redundant on some but not necessarily all paths through a program. A PRE algorithm called "PRE: a simple, pragmatic, and provably correct algorithm," presented by Vineeth Kumar Paleri does not give importance for eliminating edge splitting, even though the edge splitting is more expensive than inserting a computation at an existing node of a data flow graph (DFG). The insert equation of the PRE algorithm does not insert a computation for an expression in an existing node of a DFG if the node does not compute the expression concerned. This leads to unnecessary edge splitting. In this paper, the insert equation of the PRE algorithm is updated to avoid the edge splitting as far as possible, and hence the algorithm becomes more compact and beautiful.

Keywords Data flow graph · Partial redundancy elimination · Availability · Anticipability · Safe partial availability · Safe partial anticipability

1 Introduction

An expression is partially redundant if the value computed by the expression is already available on some but not all paths in a DFG of a program to that expression. A PRE algorithm is a method for transforming partial redundancy of an expression in a DFG into fully redundancy and eliminates the redundancy. A PRE algorithm based on safe insertions is treated to be optimal if no other PRE algorithm that uses safe insertions gives a DFG which contains fewer computations (less insertions and more deletions) in any path.

Rahibb (✉) · S. Sarala
Department of Computer Applications, Bharathiar University, Coimbatore, Tamil Nadu, India
e-mail: rahibb007@gmail.com

S. Sarala
e-mail: sarala.bu@gmail.com

T. Sengodan et al. (eds.), *Advances in Electrical and Computer Technologies*,
Lecture Notes in Electrical Engineering 672,
https://doi.org/10.1007/978-981-15-5558-9_8

Morel and Renvoise (MRA) [1] first proposed a bidirectional data flow analysis algorithm to eliminate partial redundancies. But it does not really eliminate all partial redundancies in a program, and it lacks both computational and lifetime optimality as well. As MRA fails to split edges, optimization is not possible in many loops. Though Dhamdhere [2, 3] through edge placement algorithm (EPA) performs insertions both in nodes and on edges in a DFG, he cannot completely eliminate redundant code motion. EPA does not provide lifetime optimality in many cases too.

It is already shown by Vineeth Kumar that the papers [4–7] have one or more of the problems of redundant code motion, unremoved redundancies, or limited applicability due to reducibility restriction of the flow graph.

The PRE algorithm developed by Vineeth Kumar [8] does not give much importance for eliminating edge splitting(s), and the edge splitting is more expensive than inserting a computation at an existing node of a DFG though. The insert equation of the PRE algorithm for an expression at a node$_i$ in a DFG returns true only if the node$_i$ computes the expression concerned, otherwise it returns false. This may lead to unnecessary edge splitting(s). This paper enriches the insert equation of the PRE algorithm to avoid the edge splitting as much as possible.

2 PRE Algorithm by Vineeth Kumar Paleri

Vineeth Kumar Paleri, YN Srikant, and Priti Shankar proposed a unidirectional data flow analysis algorithm for PRE titled "PRE: a simple, pragmatic, and provably correct algorithm." As the name suggests the algorithm is really simple and computationally and lifetime optimal. The algorithm assumes that all local redundancies are already eliminated by using standard techniques for common subexpression elimination on the basic blocks [9]. The algorithm utilizes the concepts of availability, anticipability, safe partial availability, and safe partial anticipability.

An expression is available at a program point p, if it is computed along all paths from the start node to p without a modification to any of its operands since the last computation, and it is partially available at p if it is computed at least along any path. An expression is anticipated at p if all paths from p have a computation of that expression from the values of the operands of the expression available at p, and it is partially anticipated if it is computed at least along any path from p. A program point is safe for an expression if it is either available or anticipated at that point.

Safe partial availability needs all points on the path along which the computation is partially available to be safe, and safe partial anticipability needs all points on the path along which the computation is partially anticipated to be safe.

The local data flow property ANTLOC$_i$ represents a locally anticipated upward exposed expression e in node$_i$, COMP$_i$ represents a locally available downward exposed e in node$_i$, and TRANSP$_i$ reflects the absence of assignments to the operand(s) of e in node$_i$. The global properties of availability, anticipability, safe partial availability, and safe partial anticipability 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 REPLACE_i identifies e to be replaced in node_i with a temporary variable, say t. The INSERT_i and $\text{INSERT}_{(i,j)}$ equations of the PRE algorithm are as follows:

$$\text{INSERT}_i = \text{COMP}_i \cdot \text{SPANTOUT}_i \neg(\text{TRANSP}_i . \text{SPAVIN}_i). \tag{1}$$

$$\text{INSERT}_{(i,j)} = \neg\text{SPAVOUT}_i \text{SPAVIN}_j \text{SPANTIN}_j \tag{2}$$

2.1 An Example

Figure 1a shows a DFG with 6 nodes, and Fig. 1b is a DFG after applying the PRE algorithm. The algorithm inserts a computation at nodes n_1 and n_5 with the Eq. (1). Here, the algorithm fails to insert a computation at the node n_2 as it does not contain the computation $a * b$. So the algorithm splits the edges (n_2, n_3) and (n_2, n_6) for inserting a node in each edge with the Eq. (2) in order to make the expression fully redundant along all paths to n_4 and n_6.

3 A Supplement to PRE Algorithm

As the PRE algorithm inserts a computation at a node only if the node contains a computation of the expression, it leads to edge splitting as shown in Fig. 1b. Being the edge splitting is very expensive as compared to inserting a computation at a node already existing in a DFG, the INSERT equation of the PRE algorithm is modified by using a program segment code without sacrificing the algorithm's computational and lifetime optimality. The program segment is as follows:

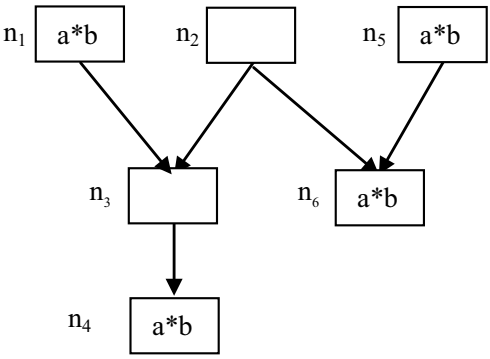
$\text{INSERT}_i = \neg\ (\text{TRANSP}_i . \text{SPAVIN}_i)\ \text{SPANTOUT}_i (\text{COMP}_i + \prod_{s \in succ(i)} \text{ANTIN}_s (|\text{PRED}_s| = 1 + \text{PAVIN}_s) (|\text{PRED}_s| = 1 + \text{PAVIN}_s)$

$\text{INSERT}_{ij} = \neg\ \text{INSERT}_i \neg\text{SPAVOUT}_i . \text{SPAVIN}_j . \text{SPANTIN}_j$

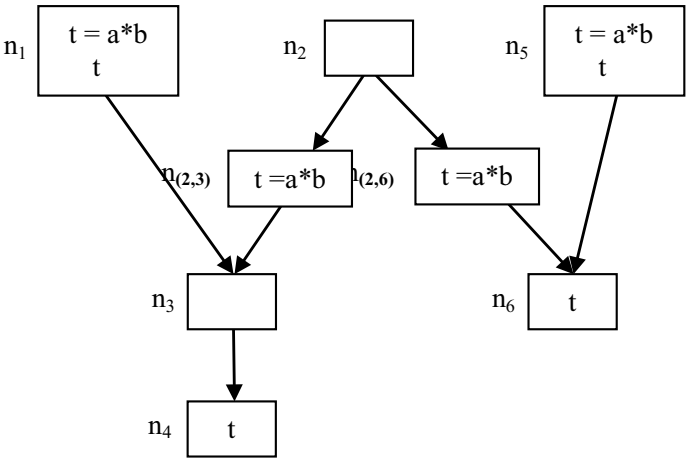
where j is a predecessor of the node_i, and all other equations of the PRE algorithm remain the same.

3.1 An Example

Consider Fig. 1a again. Figure 2 is the DFG after applying the new program code. Note that it has only three insertions and four replacements without any edge splitting.



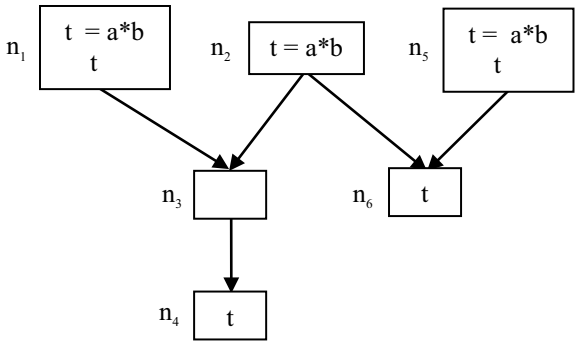
(a) before PRE algorithm



(b) after PRE algorithm

Fig. 1 Partial redundancy elimination using PRE algorithm

Fig. 2 Partial redundancy elimination using new program segment code



4 Conclusion

The goal of this paper stated in the introduction is to eliminate the edge splitting as far as possible. As the insert equation of the PRE algorithm by Vineeth Kumar fails to take care of eliminating the edge splitting much, this paper, by simply updating the INSERT equation of the PRE algorithm, eliminates the edge splitting of the DFG of a program as far as possible, and hence the PRE algorithm is now more clever and attractive without sacrificing the algorithm's computational and lifetime optimality.

Acknowledgements We are bound to thank Vineeth Kumar Paleri 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 Program of the UGC during the XIIth plan period (Letter No. F. No. FIP/12th Plan/KLCA045 TF07, dated: 10-09-2016).

References

1. Morel E, Renvoise C (1979) Global optimization by suppression of partial redundancies. Commun ACM 22(2):96–103
2. Dhamdhere DM (1988) A fast algorithm for code movement optimization. SIGPLAN Notices 23(10):172–180
3. Dhamdhere DM (1991) Practical adaptation of global optimization algorithm by Morel & Renvoise. ACM Trans Program Lang Syst 13(2): 291–294
4. Dhamdhere DM, Rosen DM, Zadeck FK (1992) How to analyze large programs efficiently and informatively. In: Proceedings of ACM SIGPLAN '92 conference on PLDI, pp 212–223
5. Dhamdhere DM, Patil H (1993) An elimination algorithm for bi-directional data flow analysis using edge placement technique. ACM TOPLAS 15(2):312–336
6. Dhamdhere DM, Khedker UP (1993) Complexity of bidirectional data flows. In: Proceedings of twentieth annual symposium on POPL, pp 397–408
7. Dhamdhere DM, Dhaneshwar VM (1995) Strength reduction of large expressions. J Program Lang 3: 95–120
8. Paleri VK, Srikant YN, Shankar P (2003) Partial redundancy elimination: a simple, pragmatic, and provably correct algorithm. Sci Comput Program 48(1):1–20
9. Aho AV, Sethi R, Ullman JD Compilers: principles, techniques, and tools. Addison-Wesley