A Fresh Look at Partial Redundancy Elimination as a Maximum Flow Problem (Abstract)*

Jingling Xue † Jens Knoop ‡

Abstract

This note reports on research which has recently been presented at the *15th International Conference on Compiler Construction* (*CC 2006*) in Vienna, Austria [17]. In this article we have shown that *classic partial redundancy elimination* (*CPRE*), like *speculative partial redundancy elimination* (*SPRE*), is a maximum flow problem, too. Thereby, we have revealed the missing link between CPRE and SPRE, and more importantly, established a common high-level conceptual basis for understanding and reasoning about this important and widely used program optimisation. We demonstrated this by formulating a new and simple unidirectional bit-vector algorithm for CPRE, which is based only on the well-known concepts of availability and anticipatability. Designed to find directly a unique minimum cut for a CFG, which can be proved simply but rigorously, the new algorithm turned out to be simple and intuitive, and its optimality self-evident. We could show that this conceptual simplicity also translates into efficiency. As demonstrated by our experimental results, the new algorithm outperforms its state-of-the-art competitors.

1 Introduction

Partial redundancy elimination (*PRE*) is a compiler optimisation which eliminates computations which are redundant on some but not necessarily all paths in a program. As a result, PRE encompasses both global common subexpression elimination and loop-invariant code motion. Over the years, PRE has also been extended to perform other optimisations at the same time, including strength reduction [4, 6, 8, 10], global value numbering [1] and live-range determination [12]. For these reasons, PRE is regarded as one of the most important optimisations in optimising compilers.

As a code transformation, PRE eliminates a partially redundant computation at a point by inserting its copies on the paths that do not already compute it prior to the point, thereby making the partially redundant computation fully redundant. PRE problems come in two flavours: *classic PRE* and *speculative PRE*. Classic PRE, as described in the seminal work [13], inserts a computation at a point only if the point is *safe* (or *down-safe*) for the computation, i.e., only if the computation is fully anticipatable at the point. If the computation cannot cause an exception and if the execution frequencies of the flow edges in a CFG are available, speculative PRE may find transformations which are beyound the scope of classic PRE and hence missed by it, thereby removing more redundancies in dynamic terms than classic PRE.

In the case of classic PRE, Knoop, Rüthing and Steffen invented an optimal unidirectional bit-vector formulation of the problem [9, 11]. This algorithm, known as *Lazy Code Motion* (*LCM*), was later recasted to operate on static single assignment (SSA) form [7]. Subsequently, a number of alternative formulations have been proposed [3, 4, 5, 14]. While LCM and other earlier algorithms [4, 5] find code insertion points by modelling the optimisation as a code motion transformation, the latter ones [3, 14] avoid this by identifying code insertion points directly. Apparently, a search for a conceptual basis upon which an optimal formulation of classic PRE can be both developed and understood more intuitively has been the driving force behind these research efforts. Up to now, however, this conceptual basis has been elusive. All existing algorithms are developed and reasoned about at the low level of individual program paths.

While classic PRE is profile-independent, speculative PRE is profile-guided [2, 15]. Given a weighted CFG, where the weights of the flow edges represent their execution frequencies, Xue and Cai have shown previously that speculative PRE is a maximum flow problem [16]. Finding an optimal transformation on a CFG amounts to finding a special minimum cut in a flow network derived from the CFG. Furthermore, different optimal transformations on a CFG may result if the weights of the flow edges in the CFG differ.

In [17], we have shown for the first time that classic PRE is also a maximum flow problem. This is the key to the main contribution of this paper: to provide a uniform approach for classic and speculative PRE. The insight behind this finding lies in the following assumption made about classic PRE [9, 11]: all control flow edges are nondeterministic, or equivalently, have nonzero execution frequencies. We could show that finding the optimal transformation for a

*The full version of this work has recently been presented at the *15th International Conference on Compiler Construction* (*CC 2006*), (*Vienna, Austria, March 30-31, 2006*) [17].

†School of Computer Science and Engineering, The University of New South Wales, Sydney, NSW 2052, Australia.

‡Institut für Computersprachen, Technische Universität Wien, Argentinierstr. 8, 1040 Wien, Österreich.

CFG amounts to finding a unique minimum cut in a flow network derived from the CFG. Since all insertions in a CFG must be safe in classic PRE (as mentioned above), this unique minimum cut is invariant of the execution frequencies of the flow edges in the CFG. This establishes the connection and highlights the main difference between classic and speculative PRE. More importantly, our finding provides a common high-level conceptual basis upon which an optimal formulation of PRE can be more systematically and intuitively developed and proved. *Every PRE algorithm, if being optimal, must find the unique minimum cut on a flow network that is derived from a CFG.* As a result, tedious and non-intuitive reasoning that has been practised at the lower level of control flow paths is dispensed with.

Based on this insight, we have developed a new, simple algorithm for classic PRE. Our formulation, applicable to standard basic blocks, consists of solving four unidirectional bit-vector data-flow problems based only on the well-known concepts of availability and anticipatability. Designed to find a unique minimum cut in a flow network derived from a CFG, which is proved simply but rigorously, our data-flow equations reason positively about the global properties computed without using logical negations. Such a formulation is intuitive and its optimality self-evident. This conceptual simplicity also translates into efficiency, as demonstrated by our experimental results. Details on this algorithm and the experimental data can be found in the full version of this paper [17].

References

[1] P. Briggs and K. D. Cooper. Effective partial redundancy elimination. In *Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation* (*PLDI'94*), volume *29*,6 of *ACM SIGPLAN Not.*, pages 159 – 170, 1994.

[2] Q. Cai and J. Xue. Optimal and efficient speculation-based partial redundancy elimination. In *Proc. of the 1st Annual IEEE/ACM Int. Symposium on Code Generation and Optimization* (*CGO 2003*), pages 91 – 102, 2003.

[3] D. M. Dhamdhere. E-path_pre: Partial redundancy elimination made easy. *ACM SIGPLAN Not.*, 37(8):53–65, 2002.

[4] V. M. Dhaneshwar and D. M. Dhamdhere. Strength reduction of large expressions. *Journal of Programming Languages*, 3(2):95 – 120, 1995.

[5] K.-H. Drechsler and M. P. Stadel. A solution to a problem with Morel and Renvoise's "Global optimization by suppression of partial redundancies". *ACM Trans. on Prog. Lang. and Systems*, 10(4):635 – 640, 1988.

[6] M. Hailperin. Cost-optimal code motion. *ACM Trans. on Prog. Lang. and Systems*, 20(6):1297 – 1322, 1998.

[7] R. Kennedy, Sun Chan, Shin-Ming Liu, R. Lo, Peng Tu, and F. Chow. Partial redundancy elimination in SSA form. *ACM Trans. on Prog. Lang. and Systems*, 21(3):627 – 676, 1999.

[8] R. Kennedy, F. Chow, P. Dahl, S.-M. Liu, R. Lo, and M. Streich. Strength reduction via SSAPRE. In *Proc. of the 7th Int.Conf. on Compiler Construction* (*CC'98*), Lecture Notes in Computer Science, vol. 1383, pages 144 – 158, 1998.

[9] J. Knoop, O. Rüthing, and B. Steffen. Lazy code motion. In *Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation* (*PLDI'92*), volume *27*,7 of *ACM SIGPLAN Not.*, pages 224 – 234, 1992.

[10] J. Knoop, O. Rüthing, and B. Steffen. Lazy strength reduction. *Journal of Programming Languages*, 1(1):71 – 91, 1993.

[11] J. Knoop, O. Rüthing, and B. Steffen. Optimal code motion: Theory and practice. *ACM Trans. on Prog. Lang. and Systems*, 16(4):1117–1155, 1994.

[12] R. Lo, F. C. Chow, R. Kennedy, S. M. Liu, and P. Tu. Register promotion by sparse partial redundancy elimination of loads and stores. In *Proc.of the ACM SIGPLAN Conf. on Programming Language Design and Implementation* (*PLDI'98*), volume *33*,5 of *ACM SIGPLAN Not.*, pages 26 – 37, 1998.

[13] E. Morel and C. Renvoise. Global optimization by suppression of partial redundancies. *Communications of the ACM*, 22(2):96 – 103, 1979.

[14] V. K. Paleri, Y. N. Srikant, and P. Shankar. A simple algorithm for partial redundancy elimination. *ACM SIGPLAN Not.*, 33(12):35 – 43, 1998.

[15] B. Scholz, N. R. Horspool, and J. Knoop. Optimizing for space and time usage with speculative partial redundancy elimination. In *Proc. of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems* (*LCTES 2004*), volume *39*,7 of *ACM SIGPLAN Not.*, pages 221 – 230, 2004.

[16] J. Xue and Q. Cai. A life-time optimal algorithm for speculative PRE. ACM Transactions on Architecture and Code Optimization. *To appear.*

[17] J. Xue and J. Knoop. A fresh look at PRE as a maximum flow problem. In *Proc. of the 15th Int. onf. on Compiler Construction* (*CC 2006*), Lecture Notes in Computer Science, vol. 3923, pages 139 – 154, 2006.