Overview

Assignment Motion

Assignment motion is a technique that complements expression motion as presented in the first part of this monograph by incorporating the movement of left-hand side variables of assignments as well. Although such an extension may seem straightforward at first glance, the movement of left-hand sides of assignments has serious consequences that are subject of this part. In contrast to expression motion which has thoroughly been studied in program optimisation [Cho83, Dha83, Dha88, Dha89b, Dha91, DS88, DS93, JD82a, JD82b, Mor84, MR79, MR81, Sor89] the work on assignment motion based algorithms is quite limited. Rosen, Wegman and Zadeck [RWZ88] mention IBM's PL.8-compiler [AH82] for an iterated application of partial redundancy elimination. However, they neither give details on this extensions nor a first-hand reference. The only explicit description of an extension of partial redundancy elimination towards assignments is given by Dhamdhere [Dha91]. However, his proposal does not recognise the full potential that lies in such an extension. This part of the book provides a systematic approach to the phenomena associated with program transformations based upon assignment motion.

Second Order Effects

The most striking phenomenon of assignment motion based transformations are their *second order effects* [RWZ88, DRZ92]: one transformation may provide opportunities for others. As a consequence, assignment motion based transformations usually do not stabilise after their first application. However, the impact of second order effects is twofold: on one side, the optimisation potential increases significantly, as even few first order opportunities may cause numerous second order opportunities. On the other hand, we are faced with the problem that exhausting the optimisation potential completely, requires to iterate the component transformations involved. This gives raises issues on the *confluence* and *complexity* of the process, which will be of major concern in this part.

Assignment Hoisting and Assignment Sinking

For expression motion the direction of code movement is determined by definition, since initialisation sites have to precede their corresponding use sites. Hence expression motion stands as a synonym for expression hoisting. In contrast, assignment motion is not restricted in this way. Both alternatives, the hoisting and sinking of assignments, are reasonable directions of code movement. This symmetry was inspiration to develop an algorithm for *partial dead code elimination* [KRS94b] which complements partial redundancy elimination. While partial redundancy elimination hoists expressions to places where

O. Ruething: Interacting Code Motion Transformations, LNCS 1539, pp. 135-136, 1998.
© Springer-Verlag Berlin Heidelberg 1998

they become redundant with respect to others, partial dead code elimination rests on the idea to sink assignments to places where they become entirely dead. Alternatively, this transformation can even be further strengthened by eliminating *faint assignments* [HDT87, GMW81] rather then dead ones only.

On the other hand, assignment hoisting can be employed equally well in order to eliminate partially redundant assignments. More interesting, however, is the function of assignment hoisting as a catalyst for enhancing the potential of expression motion, which we demonstrated in an algorithm for the *uniform elimination of partially redundant expressions and assignments* [KRS95]. Here this approach will be further improved adapting the techniques for minimising lifetime range of Chapter 4.

A Uniform Framework

Common to assignment motion based program transformations is the fact that they combine a set of admissible assignment motions with a set of corresponding eliminations. Formalising this situation, we present a uniform framework for assignment motion based program transformations that is applicable in all practically relevant situations. In this setting we provide simple criteria that grant confluence and fast convergence of the exhaustive application of elementary transformations.

Structure of the Part

In Chapter 6 we give an overview on the most relevant applications in the field of assignment motion. In particular, we address the main applications of assignment motion: *partial dead (faint) code elimination* and the *uniform elimination of assignments and expressions*. Afterwards, Chapter 7 presents our general framework for assignment motion based transformations. In symmetry to the first part, the final Chapter 8 is again devoted to issues caused by the presence of critical edges.

Conventions

As in the first part of the book we mainly consider flow graphs whose nodes are elementary statements. However, sometimes we switch to the basic block representation. This is for the reason that the basic blocks of a program are structurally invariant during a sequence of transformation steps, whereas the positions of elementary statements may significantly change. Throughout this part we assume a fixed flow graph G which is assumed to be element of $\mathfrak{FG}$ in Chapter 6 and Chapter 7 and to be element of $\mathfrak{FG}_{\mathtt{Crit}}$ in Chapter 8.