Foreword

Is computing an experimental science? For the roots of program optimization the answer to this question raised by Robin Milner ten years ago is clearly yes: it all started with Donald Knuth's extensive empirical study of Fortran programs. This benchmark-driven approach is still popular, but it has in the meantime been complemented by an increasing body of foundational work, based on varying idealizing assumptions ranging from 'space is for free' over 'there are sufficiently many registers' to 'programs consist of 3-address code'. Evaluation of the adequacy of these assumptions lacks the appeal of run-time measurements for benchmarks, which is so simple that one easily forgets about the difficulties in judging how representative the chosen set of benchmarks is. Ultimately, optimizations should pass the (orthogonal) tests of both communities.

This monograph is based on foundational, assumption-based reasoning, but it evolved under the strong pressure of the experimental community, who expressed doubts concerning the practicality of the underlying assumptions. Oliver Rüthing responded by solving a foundational problem that seemed beyond the range of efficient solutions, and proposed a polynomial algorithm general enough to overcome the expressed concerns.

Register Pressure:. A first formally complete solution to the problem of register pressure in code motion – hoisting computations enlarges the corresponding life-time ranges – was proposed for 3-address code. This assumption allowed a separate treatment of single operator expressions in terms of a bitvector analysis.

The algorithm, although it improves on all previous approaches, was criticized for not taking advantage of the flexibility provided by complex expression structures, which essentially boils down to the following trade-off patterns:

- if (two) operand expressions are only used once, within one large expression, one should hoist its evaluation and release the registers holding the operand values;
- if there are multiple uses of the operand expressions, then one should keep the operand values and delay the evaluation of the large expressions.

Based on matching theory, Rüthing proposes an algorithm that optimally resolves this 'trade-off' problem in polynomial time.

Interacting Transformations:. Optimizing transformations may support and/or impede each other, as illustrated by the two trade-off patterns in the previous paragraph: hoisting a large expression is supportive in the first but impeding in the second. In this sense, the corresponding optimal algorithm can be regarded as a complete solution to a quite complex interaction problem. In this spirit, Oliver Rüthing additionally investigates the complexity and the interaction potential of assignment motion algorithms comprising both hoisting and sinking, and establishes a surprisingly low complexity bound for the 'meta-iteration' cycle, resolving all the so-called *second-order effects*.

Finally, the monograph sketches how these two results can be combined in order to achieve independence of the assignment granularity. In particular, the combined algorithm is invariant under assignment decomposition into 3-address code, as required for many other optimization techniques. This is of high practical importance, as this increased stability under structural changes widens the range of application while maintaining the optimizing power. I am optimistic that conceptual results like this, which seriously address the concerns of the experimental community, will help to establish fruitful cross-community links.

Summarizing, this monograph, besides providing a comprehensive account of the practically most accepted program analysis and transformation methods for imperative languages, stepwise develops a scenario that overcomes structural restrictions that had previously been attacked for a long time with little success. In order to do justice to the conceptual complexity behind this breakthrough, Rüthing provides all the required formal proofs. They are not always easy to follow in full detail, but the reader is not forced to the technical level. Rather, details can be consulted on demand, providing students with a deep, yet intuitive and accessible introduction to the central principles of code motion, compiler experts with precise information about the obstacles when moving from the 3-address code to the general situation, and the algorithms' community with a striking application of matching theory.

Bernhard Steffen

Preface

Code motion techniques are integrated in many optimizing production and research *compilers* and are still a major topic of ongoing research in *program optimization*. However, traditional methods are restricted by the narrow viewpoint on their immediate effects. A more aggressive approach calls for an investigation of the *interdependencies* between distinct component transformations.

This monograph shows how interactions can be used successfully in the design of techniques for the movement of expressions and assignments that result in tremendous transformational gains. For *expression motion* we present the first algorithm for computational and lifetime optimal placement of expressions that copes adequately with composite expressions and their subexpressions. This algorithm is further adapted to situations where large expressions are split into sequences of assignments. The core of the algorithm is based upon the computation of maximum matchings in bipartite graphs which are used to model trade-off situations between distinct lifetime ranges.

Program transformations based upon *assignment motion* are characterized by their mutual dependencies. The application of one transformation exposes additional opportunities for others. We present simple criteria that guarantee *confluence* and fast *convergence* of the exhaustive transformational process. These criteria apply to a number of practically relevant techniques, like the elimination of partially dead or faint assignments and the uniform elimination of partially redundant expressions and assignments.

This monograph is a revised version of my doctoral dissertation which was submitted to the Faculty of Engineering of the Christian-Albrechts University at Kiel and accepted in July 1997.

Acknowledgements

First of all, I would like to thank Prof. Dr. Hans Langmaack for giving me the opportunity to work in his group and doing the research that finally found its result in my doctoral thesis, on which this monograph is based. I thank him for sharing his wealth of experience on the substance of computer science.

I am particularly grateful to Bernhard Steffen, who raised my interest in the field of program optimization and abstract interpretation. I certainly benefited most from the excellent cooperation with him and his group, among whom Jens Knoop had a predominant role. Our close cooperation started in Kiel and continued uninterrupted after he joined Bernhard's group in Passau. Jens was always willing to discuss my sometimes fuzzy new ideas and finally took on the proof-reading of earlier and mature versions of the book, which would not be as it is without his support. Finally, I thank Alan Mycroft for acting as the third referee of my thesis and for giving me lots of valuable comments.

In addition, I would like to thank Preston Briggs, Dhananjay Dhamdhere, Vinod Grover, Rajiv Gupta, Barry Rosen, Mary Lou Soffa, and Kenneth Zadeck for several stimulating discussions, mostly at conferences or via email exchange. I owe special thanks to Preston Briggs, as he was the one who called my attention to the problem of lifetime dependencies in code motion.

Last but not least, I want to thank my wife Sabine for steadily encouraging me, and, together with our children Laura, Thore, and Aaron, providing the pleasant and lively atmosphere in which I could relax from the up and downs during writing this book.

Dortmund, September 1998 Oliver Rüthing