Developing a Foundation for Code Optimization

Mary Lou Soffa

Department of Computer Science
University of Pittsburgh,
Pittsburgh, PA
soffa@cs.pitt.edu
www.cs.pitt.edu/~soffa

Abstract. Although optimization technology has been successful over the past 40 years, recent trends are emerging that demand we reconsider the paradigm that we are using for code optimization. In particular, the trends toward dynamic optimization, writing embedded system software in high level languages and the lack of significant performance improvement from current optimization research are forcing us to rethink what we know and do not know about optimization. A number of problems dealing with both semantic and application properties of optimizations have always existed but have been mostly ignored. The challenge in optimization today is to explore properties of optimizations and develop a framework for better understanding and use of optimizations. Fortunately, research is starting to explore properties, including proving the soundness of optimizations, proving the correctness of optimizations, specifications of optimizations, and the predictability of profits of optimizations. Not only must we understand the properties, but we also need to integrate the properties into a framework. Only then can we address the problems of the developing trends.

1 Introduction

The field of optimization has been extremely successful over the past 40+ years. As new languages and new architectures have been introduced, new and effective optimizations have been developed to target and exploit both the software and hardware innovations. Various reports from both research and commercial projects have indicated that the performance of software can be improved by 20% to 40% by applying levels of aggressive optimizations.

Most of the success in the field has come from the development of particular optimizations, such as partial redundancy elimination, speculation, and path sensitive optimization. Although we knew that there were various problems with optimization that were not well understood, they were mostly ignored. Thus instead of trying to understand and solve the problems, we avoided them for the most part because we were getting performance improvements. These problems included knowing when, where and what optimizations to apply for the best improvement. Other problems include showing the soundness of optimizations (an optimization does not change the semantics of a program) and the correctness of the optimizer that implements the optimizations. When optimizations are introduced, seldom is their soundness proved, and likewise optimizers are notorious for being buggy.

2 Technical Challenges

A number of recent events are forcing us to finally take up the challenge of the optimization problems. Because of the continued growth of embedded systems and the competitive market, where time-to-market is critical, there is a movement to write software for embedded system in high level languages. This movement requires an optimizing compiler to generate code that is near the quality of the manually produced code. Today's optimization technology is not able to adequately handle some of the challenges offered by embedded systems. For example, the resource constraints are more severe than in desktop computers and thus optimizations must be able to satisfy the constraints. Furthermore, embedded systems have multiple constraints, including execution time, memory and energy. Most of the prior work in optimization has really focused on a single constraint, namely time. The optimization technology has not been developed to handle the multiple constraints and the trade-offs. Another activity that has brought optimization problems to the forefront is the trend toward dynamic optimization. Dynamic optimization requires that we understand certain properties of optimizations in order in order for them to be effective. Currently, it is unclear when and where to apply optimizations dynamically and how aggressive the optimizations can be and still be profitable after factoring in the cost of applying the optimization. The third challenge is the lack of performance improvement that we are currently experiencing with optimization research. Although new optimizations continue to be developed, the performance improvement is shrinking. The question then is whether the field has reached its limit or is the problems that we have ignored simply limiting our progress. Lastly, the robustness of software has become a major concern, mandating that we ensure that the optimizing compilers are correct and optimizations sound.

To tackle these problems, we need to better understand the properties of optimization. We categorize optimization properties as either (1) semantic or (2) operational. Semantic properties deal with the semantics of the optimization and include

- correctness – the correctness of the implementation of optimizations,
- soundness – the semantics of a program do not change after applying an optimization, and
- specification – being able to specify conditions both the conditions needed and the semantics of applying the optimization.

Operational properties target the application of optimizations and their performance and include

- interaction – the conditions under which optimizations enable and/or disable other optimizations,
- profitability – the profit of applying an optimization at a particular point in the code given the resources of the targeted platform,
- order – the order that optimizations should be applied, based on the interaction property,
- configuration - the best optimization configuration, including tile size of loop tiling and the unroll factor, considering the resources.
- automatic generation – the conditions under which we can specify optimizations and have a tool that automatically implements the optimizations, and
- portability – the conditions to develop plug-in optimization components.

Research on these properties has been limited. However, more recently there has been a flurry of research activity focusing on optimization properties. There are two approaches to exploring the properties. One approach is through formal techniques. These include developing formal specifications of optimizations, analytical models, and proofs through model checking and theorem provers. Another approach is experimental. That is, the properties are explored by actually implementing optimizations and executing the optimized code, using the information gained to determine properties. The experimental approach can be performed prior to or in conjunction with actual program executions.

3 Related Research

Semantic properties have been formally tackled by proving the soundness of the optimizations [1] [2], and the correctness of the optimizers [3]. Formal specifications of optimizations were introduced a number of years ago by Whitfield and Soffa [4] for automatic implementation of optimizations. Recently, specification techniques have been developed to prove the soundness of optimizations [1] [2]. An experimental approach to the correctness of an optimizer involved checking both unoptimized and optimized code [5].

For operational properties, determining the order or the configuration to apply optimizations has recently been experimentally explored by Triantafyllis et al. [6], Cooper et al. [7], and Kisuki et al. [8]. The interaction and ordering property was formally explored by Whitfield and Soffa [9]. The profitability and predictability of optimizations is also being explored [10] [11].

4 Summary

The important challenge of optimization research today is to better understand properties of optimizations and use these properties when decisions about what optimization to apply, when to apply it and under what conditions to apply it. Also, it tion to apply, when to apply it and under what conditions to apply it. Also, it is imperative that we know optimizations are sound and that the implementation of optimizations is robust. We need to develop a foundation that can be used to help us determine the properties both for existing optimizations and for future ones, and to integrate the properties. Such a foundation would enable us to better understand optimizations and help us meet the challenges of emerging technologies.

References

[1] Lacey, E., Jones, N.D., Eric Van Wyk, E., Christian Frederiksen, C., Proving Correctness of Compiler Optimizations by Temporal Logic, Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, (2002) 283-294.

[2] Lerner, S., Millstein, T., and Chambers, C., Automatically Proving the Correctness of Compiler Optimizations, Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation, (2003) 1-19.

[3] Necula, G.C., Translation Validation for an Optimizing Compiler, Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, Vancouver, British Columbia, Canada (2000) 83-94.

[4] Whitfield D., Soffa, M.L., An Approach for Exploring Code Improving Transformations. ACM Transactions on Programming Languages, 19(6) (1997) 1053-1084.

[5] Jaramillo, C., Gupta, R., Soffa, M.L., Comparison Checking: An approach to avoid debugging of optimized code, ACM SIGSOFT Proceedings of Foundation of Software Engineering, (1999) 268-284.

[6] Triantafyllis, S., Vachharajani, M., Vachharajani, N., August, D., Compiler Optimization-space Exploration. 1st International Symposium on Code Generation and Optimization (2003) 204-215.

[7] K. Cooper, K., D. Subramanian, D., Torczon, L., Adaptive Optimizing Compilers for the 21st Century, Proceedings of the 2001 LACSI Symposium, Santa Fe, NM, USA, October (2001).

[8] Kisuki, T., Knijnenburg, P.M.W., O'Boyle, M.F.P., Combined Selection of Tile Size and Unroll Factors Using Iterative Compilation, International Conference on Parallel Architectures and Compilation Techniques (2000) 237-246.

[9] Whitfield, D., Soffa, M.L. An Approach to Ordering Optimizing Transformations, Proceedings ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming, (1990) 137-146.

[10] Zhao, M., Childers, B., Soffa, M.L., Predicting the Impact of Optimizations for Embedded Systems, 2003 ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems, San Diego, CA. (2003) 1-11.

[11] Zhao, M., Childers, B., Soffa, M.L., Profit Driven Optimizations, Technical Report, University of Pittsburgh, Jan. (2004).