Implementation and Comparative Experiments of Translating Algorithms to SSA form based on pointer analysis

Shinsuke Nishimoto (210067)

School of Information Science,
Japan Advanced Institute of Science and Technology

February 13, 2004

Keywords: Static single assignment form (SSA form), Pointer analysis, Compiler framework, XML.

Introduction

Static single assignment form (SSA form) is a program representation. In SSA form, each use of a variable has a single definition point. This property facilitates a possibility and execution efficiency of program optimization.

It is necessary for SSA optimization to translate from an intermediate form in each compiler into SSA form. The algorithm for translating into SSA form proposed by Cytron et al.[1] and that by Sreedhar et al.[2] are well known. In SSA translation, occurrences of alias such as a pointer variable become a subject of discussion, since pointer variables make it more difficult to analyze the data flow. Actually, the conventional algorithms above cannot analyze programs including pointers.

In order to solve this problem, several algorithms that utilize a pointer analysis information to translate into SSA form have already been proposed. Many of these algorithms use flow-sensitive pointer analysis. The algorithm proposed by Cytron et al.[3] is one of this type. In contrast, Hasti et al.[4] suggested the algorithm based on flow-insensitive pointer analysis of late years. Generally, while flow-sensitive analysis provides more precise

information than flow-insensitive one, it is more costly in terms of time and space.

The experiments for comparing these algorithms are seldom carried out on the same compiler framework, though the proposers of the algorithm usually has their own implementations. For example, Nakaya[5] has reported that two SSA translation algorithms[1][2] which have the different order of computational effort are not so much of a difference regarding the performance except for particular cases. Thus, practical performance is sometimes better than theoretical (i.e., worst-case) performance.

Purpose

In this paper, we present (i)our own compiler framework based on XML program representation as testing environment, (ii)implementations of two contrasting algorithms[3][4] for translating into SSA form with pointer analysis information, and (iii)a comparison of the two algorithms.

Conclusions

We have implemented a compiler framework based on XML program representation with a goal of comparing algorithms for translating into SSA form with pointer analysis information. Then, we implemented algorithms[3][4] on our compiler framework, which shows that it has advantages as described below.

- Our compiler framework has sufficient functions to deal with algorithms for SSA translation based on pointer analysis.
- A program representation based on XML reduces the cost of implementations.
- It is possible to compile and execute programs, measure the performance on our compiler framework as a practical matter.

Of course, further comparative experiments for these algorithms are required. Therefore, the evaluation of our compiler framework, that have been carried out in this paper, is regarded as a preliminary experiment

for comparing algorthms. In the future, we need to improve our compiler framework for further detailed experiments.

References

[1] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. An Efficient Method of Computing Static Single Assignment Form. Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 25-25, January 1989.

[2] Vugranam C. Sreedhar and Guang R. Gao. A Linear Time Algorithm for Placing ϕ-Nodes. Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 62-73, January 1995.

[3] Ron Cytron and Reid Gershbein. Efficient accommodation of may-alias information in SSA form. Proceedings of the Conference on Programming Language Design and Implementation, pages 36-45, June 1993.

[4] Rebecca Hasti and Susan Horwitz. Using static single assignment form to improve flow-insensitive pointer analysis. Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation, pages 97-105, 1998.

[5] Toshiharu Nakaya. Implementation and Evaluation of Static Single Assignment Form Converter in Compiler Infrastructure. Tokyo Institute Technology, Information Science, Graduate Thesis, 2001. (In Japanese)