

Static Single Assignment Form in the COINS Compiler Infrastructure – Current Status and Background –

Masataka Sassa[†], Toshiharu Nakaya[†], Masaki Kohama[†],
Takeaki Fukuoka[‡], Masahito Takahashi[‡] and Ikuo Nakata[¶]

[†] Department of Mathematical and Computing Sciences, Tokyo Institute of Technology

{sassa, nakaya8, kohama9}@is.titech.ac.jp

[‡] Kanrikogaku Kenkyusho, Ltd.

[¶] Hosei University

Abstract—Static single assignment (SSA) form is a program representation that is becoming increasingly popular in language processors. In SSA form, each use of a variable has a single definition point. This property facilitates program analysis and optimization in compilers. We give some preliminary results concerning the SSA form in COINS, which is a compiler infrastructure recently developed by Japanese institutions. In this paper we present (i) the current status of optimization using SSA form in COINS infrastructure, (ii) a comparison of two major algorithms for translating from normal intermediate form into SSA form, and (iii) a comparison of two major algorithms for translating back from SSA form into normal intermediate form.

Index Terms—Compiler infrastructure, Optimization, Static single assignment form (SSA form)

1 Organization of this paper

This paper is made of two parts.

The first part is based on the paper “Sassa, M., Nakaya, T., Kohama, M., Fukuoka, T. and Takahashi, M.: Static Single Assignment Form in the COINS Compiler Infrastructure, SSGRR 2003w, No. 54, Jan. 2003, <http://www.ssgrr.it/en/ssgrr2003w/papers/114.pdf>” [5]. We give an outline of the above paper in Section 2, and omit duplicating it. The readers may refer to the reference [5] for details.

The second part gives the background of the topic. The COINS compiler infrastructure is presented in Section 3, and work related to compiler infrastructures is given in Section 4.

2 Static single assignment form in COINS

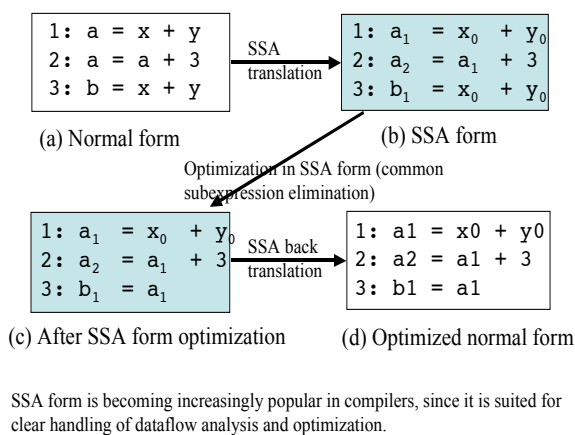
Developing a good compiler is indispensable for producing high-performance software. However, it is not easy to make a compiler that generates efficient object codes. To reduce the efforts needed to develop high quality compilers, two aspects have been studied.

One attempt involves developing *compiler infrastructures*. COINS (a COmpiler INfraStructure) [2] is one such infrastructure being developed in a research project entitled “Research on common infrastructure for parallelizing compilers”. Its development, by Japanese institutions, began in 2000.

Another attempt involves producing a representation designed to be suited for optimizations. *Static single assignment (SSA) form* is a program representation that is becoming increasingly popular in language processors. In SSA form, each use of a variable has a single definition point. This property facilitates program analysis and optimization in the compiler (Fig. 1).

In the paper [5], we give some preliminary results concerning the SSA form in COINS. We present (i) the current status of optimization using SSA form in the COINS infrastructure, (ii) a comparison of two major algorithms for translating from normal (conventional) intermediate form into SSA form, and (iii) a comparison of two major algorithms for translating back from SSA form into normal intermediate form.

A characteristic feature of the SSA module in COINS is that it provides optimization in SSA form and related utility modules as an infrastructure. This makes it easier for the compiler writer to compare and evaluate various optimization methods and to add new optimization methods and translation utilities in SSA form.



SSA form is becoming increasingly popular in compilers, since it is suited for clear handling of dataflow analysis and optimization.

Fig. 1 Optimization in SSA form

3 The COINS Compiler Infrastructure

The following is a brief outline of the COINS compiler infrastructure.

3.1 Aims of the COINS project

This section is cited from the COINS web page [2].

COINS is a project to develop a compiler infrastructure that may be used as a base for constructing various compilers such as research compilers, educational compilers, and production compilers.

COINS has two levels of intermediate representation, HIR: High-level Intermediate Representation, and LIR: Low-level Intermediate Representation. The infrastructure is composed of several components, such as front-ends, back-ends, various kinds of optimizing and parallelizing modules. Optimizations and parallelizations are applied to HIR or LIR. The structure of COINS is given in Fig. 2.

Compiler developers can construct a compiler by selecting appropriate components. They may add their own components or modify some existing components or add new features to their compiler. The main aim of the project is to help compiler developers build good compilers in relatively short term and accelerate the development of compiler technology.

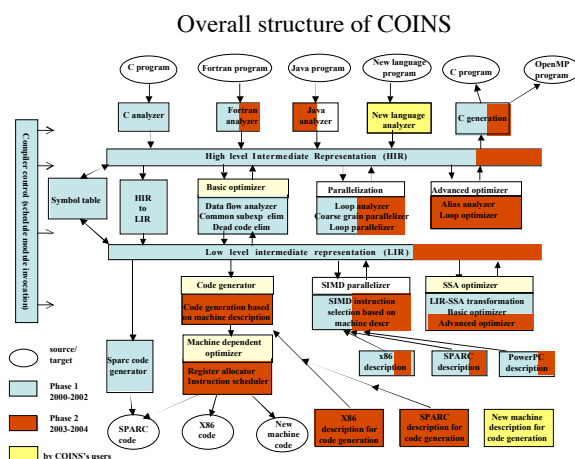


Fig. 2 Overall structure of COINS

3.2 Features of COINS

Characteristic features of COINS are as follows.

- Multiple source languages
- Multiple target architectures
- High-level intermediate representation (HIR) based on abstract syntax tree with attributes
- Low-level intermediate representation (LIR) in register transfer level with formal specification in denotational semantics [1]
- Generation of source program back from both intermediate representations enabling source-to-source translation
- Scalar analysis and optimization (in usual form and in SSA form)
- Basic parallelization (e.g. OpenMP)
- SIMD parallelization
- Code generators generated from machine description
- Written in Java (early error detection), publicly available

4 Related Work

Work related to the treatment of SSA form in compiler infrastructures is given in [5]. Here we focus on compiler infrastructures in general.

There are a couple of compiler infrastructures, for example, SUIF [7], Machine SUIF [4], Zephyr [9], Scale [6], and GCC [3].

4.1 SUIF and Machine SUIF

The SUIF compiler infrastructure [7] seems to be the most popular among them (Fig. 3). SUIF is a free infrastructure designed to support collaborative research in optimizing and parallelizing compilers. SUIF2 is a new version of the SUIF compiler system, the old version being now called SUIF1. SUIF has been widely used as a test bed for compiler research, for example in the Valen-C compiler at Kyushu University.

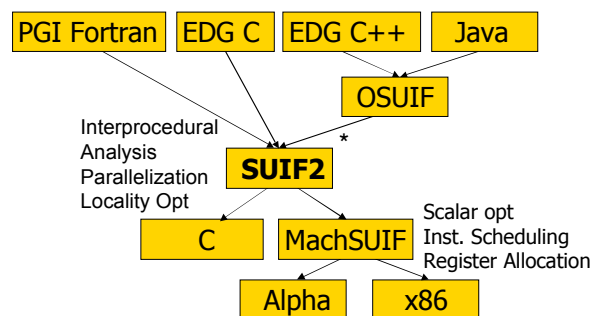
The SUIF compiler system has been

developed by Monica Lam and others at Stanford University. It was co-funded by DARPA and NSF as the National Compiler Infrastructure (NCI) project. Due to the end of NCI, PGI no longer provides support.

The main features of SUIF are the following.

- Extensible IR and utilities (mixture of high-level and low-level constructs, Dismantlers lower the representation)
- Interactive compilation (Suifdriver calls passes through files or memory)
- OSUIF for dealing with object-oriented languages (no threads, exception handling)
- Written in C++ (Smgn generates C++ from Hoof specification)
- High-level analysis infrastructure (graphs, SCCs, iterated dominance frontier)
- Steensgaard's alias analysis
- Scalar optimizations
- Interprocedural array analysis (array dependence & privatization, Presburger arithmetic)
- Interprocedural analysis and parallelization (Affine partitioning for parallelism & locality unifies loop transformation)

The SUIF System



* C++ OSUIF to SUIF is incomplete

Fig. 3 The SUIF System [7]

Machine SUIF [4] is an infrastructure for constructing compiler back ends and can be used with SUIF (Fig. 4). It is developed by Michael Smith and others at Harvard University,

and has been also part of the NCI project. Machine SUIF is used to develop machine-specific optimizations, construct profile-driven optimizations, and evaluate architectural ideas. It includes libraries for control- and data-flow analysis, and passes for register allocation, instruction scheduling, scalar optimization, and code layout.

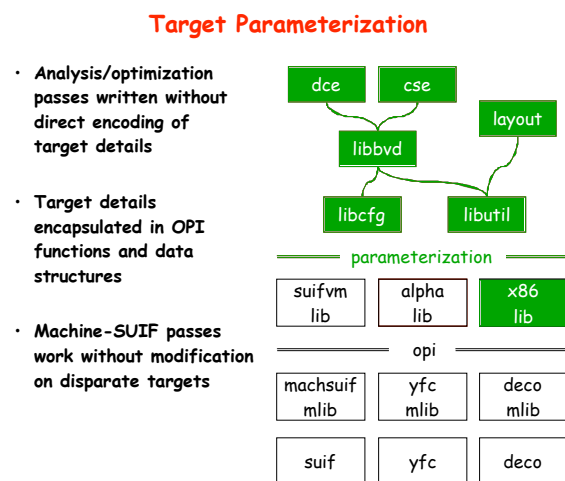


Fig. 4 Machine SUIF [4]

4.2 Other compiler infrastructures

Zephyr [9] is a compiler infrastructure developed at University of Virginia and Princeton University. Zephyr's very portable optimizer (vpo) provides instruction selection, instruction scheduling, and classical global optimization. A translator from SUIF internal representation to that of Zephyr is provided.

The Scale compilation system [6] is a compiler infrastructure developed at University of Massachusetts. Scale currently implements frontends for C, Fortran, and Java, alias analyses, static single assignment form (SSA), a collection of scalar optimizations (PRE, value numbering, copy propagation, dead code elimination, constant propagation), and outputs C. It does not generate machine codes.

4.3 Comparison with COINS

Comparing with other compiler infrastructures, COINS can be characterized by the following features.

- The whole source program is publicly available, integrated and continuously maintained by the development group while in some other infrastructures a part of the source program is proprietary (e.g. PGI Fortran for SUIF was released only in binary form and is no more supported).
- It is written in Java. Java is relatively safer compared with C or C++ that is used to develop most other compiler infrastructures.
- The high-level intermediate representation HIR and the HIR-to-source generator facilitate applications to software engineering field [8].
- The low-level intermediate representation LIR is defined formally in denotational semantics, and the code generation from LIR is based on machine description.
- It includes characteristic optimizations such as SSA form optimization and SIMD parallelization.

5 Concluding remarks

Currently, the development stage of COINS is at the first phase. In the near future, we expect that further algorithms can be developed simply using this compiler infrastructure.

Acknowledgments

This work was supported in part by the Japanese Ministry of Education, Culture, Sports, Science and Technology under Grant "Special Coordination Fund for Promoting Science and Technology" and Grant-in-Aid for Scientific Research (C) (2).

References

- [1] Abe, S., Fujinami, N., Nakata, I. and Hagiya, M.: LIR: the low-level intermediate

- representation of the COINS project (in Japanese), IPSJ SIGPRO, Jun. 2002.
- [2] COINS. <http://www.coins-project.org/>.
- [3] GCC. <http://www.gnu.org/software/gcc/>.
- [4] Machine SUIF. Harvard University.
<http://www.eecs.harvard.edu/machsuiif/>.
- [5] Sassa, M., Nakaya, T., Kohama, M., Fukuoka, T. and Takahashi, M.: Static Single Assignment Form in the COINS Compiler Infrastructure, SSGRR 2003w, No. 54, Jan. 2003.
<http://www.ssgrr.it/en/ssgrr2003w/papers/114.pdf>
- [6] Scale Compiler Group. University of Massachusetts.
<http://www-ali.cs.umass.edu/Scale/>.
- [7] SUIF. Stanford University.
<http://www-suif.stanford.edu/>.
- [8] Toita, K., Yamamoto, S. and Agusa, K.: A common representation for program slicing tool (in Japanese), in IEICE SIGSS, Mar. 2003.
- [9] Zephyr. <http://www.cs.virginia.edu/zephyr/>.