Value Flow Graph Analysis with SATIrE

Gergö Barany

Institute of Computer Languages, Vienna University of Technology
gergo@complang.tuwien.ac.at

Abstract. This work discusses implementation of partial redundancy elimination using the value flow graph, a syntactic program representation modeling semantic equivalences. It allows the combination of simple syntactic partial redundancy elimination with a powerful semantic analysis. This yields an optimization that is computationally optimal and simpler than traditional semantic methods.
A source-to-source optimizer for C++ programs was implemented using the SATIrE program analysis and transformation system. Two tools integrated in SATIrE were used in the implementation: ROSE is a framework for arbitrary analyses and source-to-source transformations of C++ programs, PAG is a tool for generating data flow analyzers from functional specifications.

1 Introduction

Partial redundancy elimination is a common program optimization that attempts to improve execution time by removing superfluous computations from a program. There are two well-known classes of such techniques: syntactic and semantic methods. Syntactic methods eliminate only redundant computations of expressions that are syntactically equal, i. e., common subexpressions in the classical sense [1]. Semantic methods also consider expressions that are equivalent due to the effects of assignment statements in the program; such techniques are typically based on SSA form [2].

While semantic optimization is more powerful, traditional SSA-based algorithms are complicated, heuristic in nature, and unable to perform certain useful optimizations. The value flow graph is a syntactic program representation modeling semantic equivalences; it allows the combination of simple syntactic partial redundancy elimination with the power of a semantic analysis that characterizes all equivalences that arise from assignments between program terms. The result is an optimization that is both simpler and more powerful than traditional semantic methods based on SSA form.

This work introduces the theory behind partial redundancy elimination using the value flow graph. A source-to-source optimizer using the value flow graph for C++ programs was implemented using two tools integrated in the SATIrE program analysis and transformation system: ROSE is a framework for arbitrary analyses and source-to-source transformations of C++ programs, PAG is a tool for generating program analyzers.

2 Value Flow Graph Analysis

This section introduces the value flow graph, a program representation for optimal code motion. It represents semantic equivalences in a syntactic way, which allows the combination of a powerful program analysis with a simple optimizing transformation.

2.1 The Value Flow Graph

The *value flow graph* (VFG) was introduced by Steffen et al. [3, 4] as a way to unify syntactic and semantic approaches to partial redundancy elimination. The graph's nodes are associated with program points and represent equivalence classes of expressions at each point. Equivalences are computed based on assignments in the program; the classes can be represented efficiently by using structured partition DAGs computed using data flow analysis.

A value flow edge connects two nodes if they belong to program points connected by a control flow edge, and if they represent the same value. Figure 1 shows a small example of a value flow graph. Bold nodes and edges are used to represent the program's control flow graph. Each program point is associated with a DAG denoting a partitioning of expressions into equivalence classes. Grey edges represent the value flow between classes of complex expressions.

Since value flow edges follow control flow edges, there is a correspondence between value flow paths and control flow paths. It is this property that allows the use of a simple code motion algorithm on the VFG: The syntactic partial redundancy elimination algorithm of Morel and Renvoise [1] can be lifted from control flow graphs to value flow graphs. However, since VFG paths represent semantic equivalence, the resulting optimization is semantic code motion, and can be made at least as powerful as traditional SSA-based methods [3].

2.2 Inductive Partial Redundancies

The value flow graph allows the optimization of a special type of partial redundancy called *inductive partial redundancy* [5]. This is a redundancy between expressions that are evaluated in different iterations through the body of the same loop.

Figure 2 illustrates an example of elimination of inductive partial redundancies, showing the original and optimized programs side by side. The code models two pointers `r1` and `r2` traversing an array `a`, with `r2` always pointing one word ahead of `r1`. Therefore, the computation of the new value for `r1` inside the loop will always yield the value stored in `r2` at that time, so the addition can be replaced by a simple copy. In the example, the redundant value is retained in a temporary variable introduced by the optimizer. This optimization cannot be realized in SSA-based approaches because SSA form does not offer any way to represent the equivalence `r2` = `r1 + 4`.

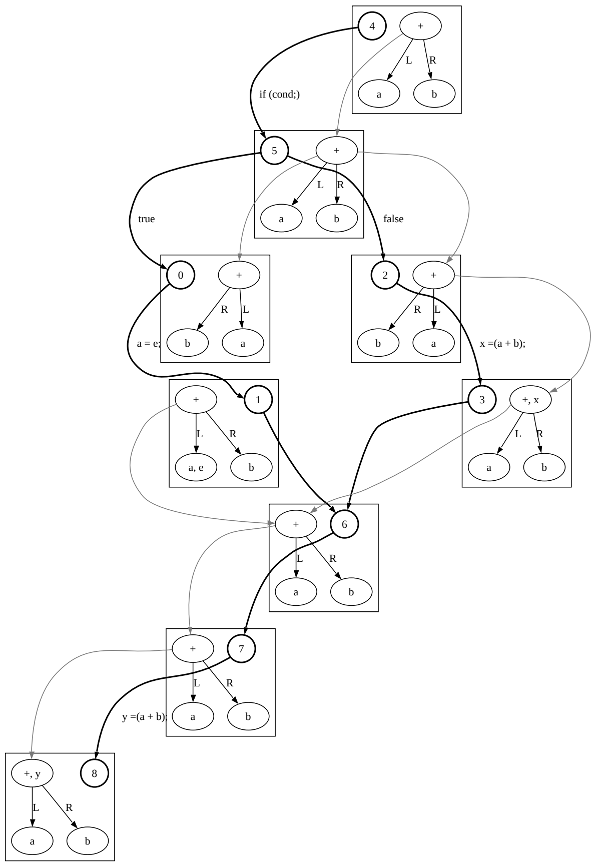

Fig. 1. Example value flow graph.

```
r1 = a;

r2 = a + 4;
do {
    r1 = r1 + 4;

    r2 = r2 + 4;
} while (r2 < end);
```

(a) Original code

```
r1 = a;
vfg_tmp_var_0 = (a + 4);
r2 = vfg_tmp_var_0;
do {
    r1 = vfg_tmp_var_0;
    vfg_tmp_var_0 = (r1 + 4);
    r2 = vfg_tmp_var_0;
} while (r2 < end);
```

(b) Transformed code

Fig. 2. Example of inductive partial redundancy elimination using the automatically derived loop invariant `r2` = `r1 + 4`.

3 Implementation with SATIrE

An optimizer prototype was implemented using the *Static Analysis Tool Integration Engine* (SATIrE) [6]. SATIrE is a framework for integrating various tools for static analysis and program transformation; it allows the exchange and combination of analysis results from several tools and makes it possible to define custom program transformations based on such analyses.

SATIrE is under development at Vienna University of Technology. At the time of writing, it uses three toolchains for program analysis and transformation:

ROSE [7] is a source-to-source analysis and transformation framework for C, C++, and Fortran programs. SATIrE uses the EDG C and C++ frontend used by ROSE, and uses ROSE's abstract syntax tree (AST) for representing programs. ROSE allows arbitrary transformation of such ASTs.

PAG [8] is a tool for generating data flow analyzers. Analyzers are specified in an equational language and translated to C code by PAG. PAG is connected to SATIrE using components that build an interprocedural control flow graph (ICFG) from the ROSE AST, and that map data flow analysis results back to the AST.

Prolog term manipulators use components provided by SATIrE for converting the ROSE AST to Prolog terms and back. Such terms can be analyzed and transformed using Prolog's powerful unification and term manipulation facilities.

In addition, program analysis and transformation tools that process source code can be connected to SATIrE by exchanging annotated or transformed source code. Figure 3 shows the overall architecture of the system. SATIrE is used in teaching program analysis and in research projects for worst-case execution time analysis [9].

The VFG optimizer implemented with SATIrE used the frontend available via ROSE; expression equivalence classes were computed by a data flow analyzer generated by PAG. The computation of VFG edges from these equivalence classes and the computation of optimal computation points were implemented in C++. The optimizing transformation based on this information was implemented using ROSE's AST transformation facilities, and the AST was unparsed to C++ source code by ROSE's backend.

4 Results

The source-to-source optimizer was tested by applying it to various small C++ programs. These programs were limited in size, due mainly to the fact that the version of SATIrE available at the time was in a rather early stage of development and contained some very inefficient parts. Current versions of SATIrE are much more efficient, scaling up to hundreds of thousands of lines of C++ code.

The largest program that was successfully processed was the C version of the Dhrystone benchmark [10], comprising about 720 lines of code. Only one of

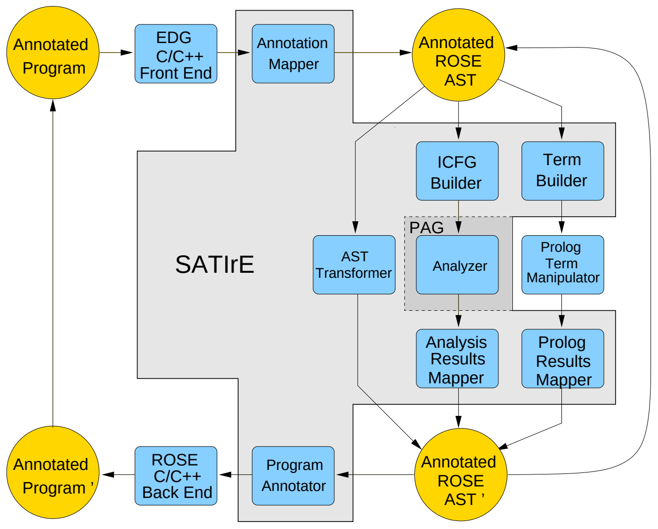

Fig. 3. Architecture of the SATIrE framework.

the routines in the benchmark was significantly altered by the optimizer, but profiling showed that this routine accounted for about 22.5 % of total execution time. Compared to GCC with maximum optimization, the code optimized by SATIrE was about 2 % faster.

This speedup is quite modest. This was to be expected as GCC itself offers powerful partial redundancy elimination. Thus the only speedups we realized were due to some complex array index computations that GCC did not optimize. Partial redundancies that other compilers don't eliminate are rare, particularly at the source code level.

5 Related Work

The optimization discussed in this work is based on the value flow graph, a program representation introduced by Steffen et al. [3]. The approach unifies two different techniques for code optimization by motion of expressions: partial redundancy elimination [1, 11] and global value numbering [2].

A similar optimization was implemented by Bodík and Anik [5]. Their approach is more aggressive in that it also considers expression equivalences due to certain arithmetic identities. To reduce the complexity of the analysis, they only consider certain restricted classes of expressions that arise from simple address computations.

This paper is a short version of the author's master's thesis [12].

6 Conclusions

This paper presented the value flow graph (VFG), a program representation for powerful semantic partial redundancy elimination. The value flow graph connects equivalence classes of expressions at subsequent program points; code motion on the VFG can be implemented using simple data flow analysis. The resulting optimization can be made strictly more powerful than what is possible in SSA form.

A source-to-source optimizer prototype for C++ programs was implemented using the SATIrE framework. The results of the prototype were modest due to limitations of SATIrE at the time of the experiments, and because partially redundant expressions are rare on the source code level.

Acknowledgements

Work on SATIrE was supported by the European Community under the FP6 ARTIST2 Network of Excellence on Embedded Systems Design, contract IST-004527, and under the FP7 project ALL-TIMES, contract IST-215068.

References

1. Morel, E., Renvoise, C.: Global optimization by suppression of partial redundancies. Communications of the ACM **22**(2) (1979) 96–103
2. Rosen, B.K., Wegman, M.N., Zadeck, F.K.: Global value numbers and redundant computations. In: POPL '88. (1988) 12–27
3. Steffen, B., Knoop, J., Rüthing, O.: The value flow graph: A program representation for optimal program transformations. In: ESOP '90. (1990) 389–405
4. Knoop, J., Rüthing, O., Steffen, B.: Code motion and code placement: Just synonyms? Lecture Notes in Computer Science **1381** (1998) 154–169
5. Bodík, R., Anik, S.: Path-sensitive value-flow analysis. In: POPL '98. (1998) 237–251
6. Schordan, M.: Combining tools and languages for static analysis and optimization of high-level abstractions. In: 24. Workshop der GI-Fachgruppe "Programmiersprachen und Rechenkonzepte", Department of Computer Science, Christian-Albrechts-Universitt zu Kiel (2007) 72–81
7. Schordan, M., Quinlan, D.: A source-to-source architecture for user-defined optimizations. In: Proc. Joint Modular Languages Conference. (2003)
8. Martin, F.: PAG – an efficient program analyzer generator. International Journal on Software Tools for Technology Transfer **2**(1) (1998) 46–67
9. Prantl, A.: Source-to-source transformations for WCET analysis: The CoSTA approach. In: 24. Workshop der GI-Fachgruppe "Programmiersprachen und Rechenkonzepte", Department of Computer Science, Christian-Albrechts-Universitt zu Kiel (2007) 51–60
10. Weicker, R.P.: Dhrystone: A synthetic systems programming benchmark. Communications of the ACM **27**(10) (1984) 1013–1030
11. Knoop, J., Rüthing, O., Steffen, B.: Optimal code motion: Theory and practice. ACM TOPLAS **16**(4) (July 1994) 1117–1155
12. Barany, G.: Semantics-based code optimization with SATIrE. Master's thesis, Vienna University of Technology (2008)