An XML based Compiler Framework to Develop Optimizers More Efficiently

Kohji Saiki (110046)

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

February 14, 2003

Keywords: XML, Compiler, Optimization techniques, Framework, DTD design.

1 Background

Modern microprocessors are designed to achieve high performance when code is highly optimized. So far, various optimization techniques have already been studied. Constructing compiler is indispensable to study and develop novel optimization techniques, but it is often very costly and spends much time.

To solve this problem, several compiler frameworks have been proposed and developed. These frameworks are designed to allow optimizer developers to reuse commonly used components in compilers to reduce the cost of developing compilers. Most of them are implemented as object-oriented frameworks.

Existing compiler frameworks are appropriate for reuse of optimizers which are already implemented. Programmers are forced to use programming interfaces provided by compiler frameworks. But, limiting use of interfaces and languages narrows the range of choice of development methods. Therefore, we cannot adopt new development method by which we can expect the improvement of productivity.

2 Purpose

The purpose of this study is to develop a new compiler framework to improve the productivity of optimizer development. We use XML (Extensible Markup Language) to achive our goal. XML is a standard for structured documents. There are many XML related technologies such as DOM (Document Object Model) and XSLT (Extensible Stylesheet Language Transformation).

While the existing compiler frameworks provide object-oriented APIs, our framework is based on XML data schema. Thus, our approach is antithetical to the existing ones. This paper shows the effectiveness of XML and related technologies for improving the productivity of optimizer development.

3 Implementation of XML based compiler framework

The key issue in our XML-based compiler framework is to mark up compiler's intermediate representation (IR for short) using XML. In this approach, we only need to add to the existing compiler component such as GCC a feature to map (i.e., translate) XML and IR. Any programming language can be used to implement optimizers as far as it conforms to the standard of XML data schema.

Optimizers have main three behaviors: (1) to extract dependency relations from IR, (2) to analyze them, and (3) to modify and/or output IR. Our XML based optimizers obtain the dependency relations using XPath, and processe IR represented as XML using DOM.

In this paper, we implement two XML-based compiler frameworks, each of which has distinct strategy. One extends the existing compiler, GCC (GNU Compiler Collection); the other uses a newly developed compiler whose target language is a subset of ANSI C called XC. The former can achieve practical compiler framework, but it needs to analyze GCC's huge source code. In contrast, the latter can design and implement an ideal IR easier, but it is required to implement a new compiler from the scratch.

4 Implementing compiler framework based on the existing compiler

In the strategy using the existing compiler, we designed RTL-XML, which marks up GCC's IR called RTL (Register Transfer Language). By using RTL-XML, we can develop the development tools which are independet from both programming languages and machine architectures. An example of these tools is weaver in aspect-oriented programming.

We experimentally developed a RTL-XML visualizer by the way of trial.

We tried to implement the compiler framework by extending the mechanism of GCC's debug output, but we could not complete it as a fully functional system. This is mainly because there are complex dependencies between RTL and GCC's internal data structures, and we could not know all of them in the early stage of development. From this experience, we found that detailed information about GCC's internals should be well documented, but not.

5 Implementing compiler framework for subset of C

We implemented another compiler framework, whose target language is XC, a subset of ANSI C. This includes designing and implementing XC compiler called `xcc`. Therefore, we could design compiler's internal data structures that are suitable for XML processing. We designed XCC-XML markup language as `xcc`'s IR. Also we designed ODF (Optimizer Description File) which controls XCC-XML optimization pass.

We conducted some experiments on implementing optimizers using XCC-XML and an object-oriented scripting language Python, where 4 optimization techniques were implemented. Each implementation took only about a day.

Our experiments using the subset of C are not good enough to show the effectiveness of XML-based compiler framework, but we believe these preliminary experiments are an

important step to construct practical compiler frameworks.

6 Discussions

In the experiments using XCC-XML, we found that our approach using XML is effective in the following:

- XPath has enough expressive power to retrieve data from IR.
- Once an XCC-XML document is generated and stored, we do not need to generate it again while debugging and testing, which greatly improves the productivity.
- We can conduct several experiments and tests on XCC-XML documents which are loaded into Python interpreter. We do not need to reload them in each case.

The findings presented here are preliminary since our experiments are limited to XCC-XML (subset of C) and 4 optimization techniques. We need more experience to show our approach is really effective.

7 Conclusion

In this paper, we proposed an XML-based compiler framework that aims at improvement of optimizer development. We took two approaches: (1) extending GCC and (2) implementing a new compiler. Although we could not finish the former system, we conducted experiment on the latter implementation.

Although our experiments are preliminary, the results of them indicate XML and related technologies worked well, and our optimizer development method using XML are effective.

We recognized that the latter XCC-XML and its processor should be improved, since the current XCC-XML schema cannot achieve practical optimization such as local register allocation, instruction scheduling, and XCC-XML processor should support ANSI C or GCC extensions not only subset of C.

After achieving the short-term issues, we have a plan to study the following:

- Multi-architecture support.
- Binary and source code compatiblity with existing compilers.
- How IR saved as XML works other than optimization.
- Design and implementation of multilayerd intermediate representation which is according to each kind of optimization.
- Repository for reusing XPath expressions.