Xevolver: an XML-based Programming Framework for Software Evolution

Hiroyuki Takizawa
Graduate School of Information Sciences, Tohoku University
6-6-01 Aramaki-aza-aoba, Aoba, Sendai 980-8579, Japan
tacky@isc.tohoku.ac.jp

Shoichi Hirasawa
Graduate School of Information Sciences, Tohoku University
6-6-01 Aramaki-aza-aoba, Aoba, Sendai 980-8579, Japan
hirasawa@sc.isc.tohoku.ac.jp

Hiroaki Kobayashi
Cyberscience Center, Tohoku University
6-3 Aramaki-aza-aoba, Aoba, Sendai 980-8578, Japan
koba@isc.tohoku.ac.jp

ABSTRACT
In this work, we propose an extensible programming framework, named Xevolver. The framework exposes an abstract syntax tree (AST) in an XML data format to programmers. Hence, the programmers can adopt various XML-related technologies to transform, analyze, and visualize the application code. In this work, we use XSLT to define custom compiler directives for application-specific code transformations. By incrementally inserting the user-defined directives, a real application can be migrated to another system without significantly modifying the original code, because custom code translation rules are written in an external XSLT file. We can change the behaviors of user-defined directives for individual systems by changing XSLT rules in the external file. Accordingly, we can evolutionarily improve the application so as to have a high performance portability without messing up the original code.

1. INTRODUCTION
A High-Performance Computing (HPC) application is often optimized for a particular system, and hence cannot achieve a high performance on another system. On the other hand, HPC system architectures change drastically over time. In general, it is unacceptable to completely rewrite the application for every new system. Therefore, we often need to incrementally improve or evolve existing applications so that they can adapt to new systems, i.e. legacy application migration.

Basically, system-specific optimizations are mandatory to exploit the potential of modern HPC systems. A higher performance can be achieved only by thoroughly specializing the application code for a particular target system. However, system-specific optimizations degrade the performance portability across different system architectures, sizes, and generations. They also make collaboration of computational scientists and expert programmers more difficult. The code modifications for performance optimizations are often non-intuitive, heuristic, and difficult to understand for non-expert programmers. After the code modifications, hence, it could be difficult for a computational scientist to maintain and/or evolve his/her application program any more.

In this work, we propose an extensible programming framework that exposes an abstract syntax tree (AST) to programmers. Although many approaches have been proposed to describe optimizations in various ways such as compiler directives and special script languages[1, 4], non-trivial modifications of the original code that could be system-specific and/or application-specific are still required in practical uses. Therefore, the proposed framework allows programmers to define their own compiler directives in addition to basic code transformations. The code translation rules of user-defined directives are written in an external file, a so-called translation recipe, system-specific optimizations can be isolated from the original application code, and a higher performance can be achieved without messing up the original application code.

2. THE XEVOVER FRAMEWORK
This work proposes an extensible programming framework for software evolution, named Xevolver. In the framework, an application program is first parsed and its AST is then output as XML (eXtensible Markup Language) data, called an XML AST. Accordingly, the Xevolver exposes an AST to a programmer in a human-readable text format.

In this work, we have implemented the Xevolver framework on top of the ROSE compiler infrastructure[3] to achieve interconversion between ROSE Sage III ASTs and XML ASTs. Thus, the Xevolver framework can use quite a huge amount of code transformations provided by the ROSE infrastructure. However, an application code would still need to be modified due to system-specific and/or application-specific reasons. It is impossible for a finite number of rules to cover all the code transformation demands. In the Xevolver framework, therefore, an XML AST is exposed to programmers including computational scientists and expert performance tuners.

Using arbitrary XML transformation tools, expert program-
mers can define code translation rules for special demands of each application. In this work, XSLT is used to describe code translation rules of user-defined compiler directives in a translation recipe, and an application code is just annotated by the directives. As discussed later, simple converters that can be represented with XSLT are yet helpful to avoid applying system-specific optimizations directly to an application code, and thereby to achieve a high performance portability.

3. EVALUATION AND DISCUSSIONS

In this work, we demonstrate that the Xeveloper framework can significantly improve the performance portability of a real application, called Numerical Turbine[2]. Numerical Turbine has originally been developed for a vector supercomputer, the NEC SX-9 system, installed at Cyberscience Center of Tohoku University. In this evaluation, using OpenACC directives, Numerical Turbine is migrated to a CPU-GPU cluster system. As several system-aware optimizations for the SX-9 system have already been applied to the kernels, the Xeveloper framework is used to improve the performance portability across the two different systems without major modification of the application code.

The two systems basically require different loop optimizations. For the SX-9 system, kernel loops are always optimized so that their innermost loops are vectorized to exploit the loop-level parallelism. On the other hand, it is not always the best for GPUs to vectorize the innermost loops, and hence loop interchange is heavily used for the migration. In addition to such a basic loop optimization, application-specific code transformations are also needed for the migration. Numerical Turbine has a lot of similar loop nests, and OpenACC cannot vectorize most of them because the lengths of some loops in the loop nests are decided by the loop indices of their outer loops. To enable OpenACC to vectorize such loops, we often need to reorganize the loop nests, and code modification for the reorganization is likely to be application-specific.

Since most of the kernel loop nests need similar transformations, only a few kinds of simple converters are required to translate the original version to an efficient OpenACC version. In our evaluation, simple converters defined by our XSLT rules can express those application-specific code transformations. Custom compiler directives for the application-specific code transformations are defined in an external translation recipe, and the application code is just annotated by some user-defined directives for the transformations. Since the translation recipes for individual systems would usually be written in separate files by expert performance tuners, their performance optimizations will not mess up the application codes. Therefore, the Xeveloper framework will help an appropriate division of labor between computational scientists and performance tuners.

Figure 1 shows the speedup ratios of the OpenACC version to the original version for major kernel loop nests in the most time-consuming subroutine. These results demonstrate that the code transformations by user-defined directives enable OpenACC to vectorize the loop nests and thus significantly improve the GPU performance. We can evolutionarily improve the GPU performance by gradually replacing system-specific loop optimizations with custom compiler directives.

In Figure 1, the SX-9 performance is degraded by the code transformations. However, the Xeveloper framework enables the SX-9 system as well as the GPU cluster system to achieve a high performance because we can obtain the original code by just disabling the code transformations. Accordingly, by switching the translation recipe, a single application code can be translated to an appropriate version for each system, and achieve a high performance portability.

4. CONCLUSIONS

In this work, we propose an extensible programming framework that can separate system-specific optimizations from the original application code. The proposed framework exposes an XML AST to programmers. Hence, the programmers can adopt various XML-related technologies to transform, analyze, and visualize the application code represented as an XML AST. In this work, we use XSLT to define simple converters as application-specific compiler directives. By incrementally inserting the user-defined directives, a real application can be migrated to another system without significantly modifying the original application code. Accordingly, we can evolutionarily improve the application so as to have a high performance portability without messing up the code.

By introducing an XML AST, we can also use other XML-related technologies. For example, we are now examining XPath to express particular code patterns in an XML AST that potentially cause problems of readability, maintainability, functionality and/or performance. In our future work, we will enhance the Xeveloper framework so that it can work together with more tools.

5. REFERENCES


