International Journal of Research in Engineering, Science and Management
Volume-2, Issue-5, May-2019
www.ijresm.com | ISSN (Online): 2581-5792

COCOVILA for Visual Languages and a Compiler for Processor with Programmable Accelerator

Akuleti Harshini Reddy[1], H. G. Anil Kumar[2], Sini Anna Alex[3]
[1,2]*Student, Dept. of Computer Science and Engineering, Ramaiah Institute of Technology, Bangalore, India*
[3]*Assistant Professor, Dept. of Computer Science and Engg., Ramaiah Institute of Technology, Bangalore, India*

***Abstract*: In this paper, a compiler-compiler for visual languages and a project for compiler that maps program loops onto a processor with programmable accelerator is presented. To translate schemas into textual representation as well as into programs representing the deep meaning of schemas a compiler-compiler framework has been designed for building visual programming environments. Here a large part of the framework is needed for support of interactive usage of a visual language. The processor with programmable architecture could be a system on a chip containing regular computational cores as well as a programmable circuit. The regular ones differ from compiler under study a driver library for data transfer between a CPU and accelerator as well as for the presence of a converter from C to the hardware description language.**

***Keywords*: Compiler, programmable accelerator**

1. Introduction

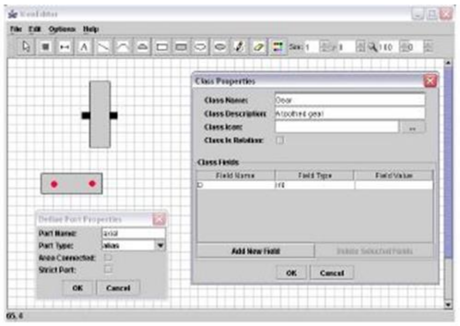
In this paper, the development of a compiler project from C to a processor with programmable accelerator is presented. A processor with programmable accelerator allows adjusting the architecture to the program being optimized. With a reconfigurable accelerator, that particular architecture could be generated which is convenient for the current loop or loop nest. It could be SIMD, MIMD, pipeline, multi-pipeline, or other computational architecture. There are no direct analogs to the suggested project, the methods of programming reconfigurable systems using high-level languages are known from different High-Level Synthesis tools. Vivado HLS (uses source code programs to guide synthesis process), Catapult HLS, Impulse C, C2H (discontinued), C-to-Verilog (discontinued), and many others are the examples.

2. Deep and Shallow linguistics of visual languages

For rapid development of domain-specific visual languages, compilers of programming languages are used as a tool. On the semantic side, we are able to specify precisely shallow semantics which produces a textual representation of schema without any loss of essential information included in a schema. To generate an executable code from a schema, using in essence an extension of attribute grammars to schema languages, we have implemented deep semantics of schema. The framework differs from the earlier ones by its capability to translate a visual sentence in principle into an arbitrary code using semantic programs run as implementations of attribute dependencies. The compiler-compiler COCOVILA supports a language designer for the visual languages, including the specification of graphical objects, syntax and semantics of the language and provides the user with a visual programming environment, which is automatically generated from the visual language definition.

3. Editor Class

Class Editor and Scheme Editor are the two components of COCOVILA. Class Editor is the tool for a visual language developer that support the language designer in defining the visual aspects of class. Schema Editor uses the results of a visual language development which are stored in a package. The user interface is automatically generated from the language definition given in Class Editor.



Class Properties
Class Name: Gear
Class Description: A toothed gear
Class Icon:
Class Is Relation:
Class Fields

Field Name	Field Type	Field Value
D	int	

Define Port Properties
Port Name: axial
Port Type: alias
Area Connected:
Strict Port:

Fig. 1. Class editor

4. Compiler

In our framework the deep semantics has been implemented on Java platform in such a way that a synthesized program becomes a method in a new Java class. Methods of a class representing the object or by equations are given by functional dependencies between attributes of an object. The usage of methods and equations as functional dependencies are specified in a textual specification added to every class that can represent an object in a scheme. Attributes of objects of a scheme are not components of the class of the object are an important point of the implementation.

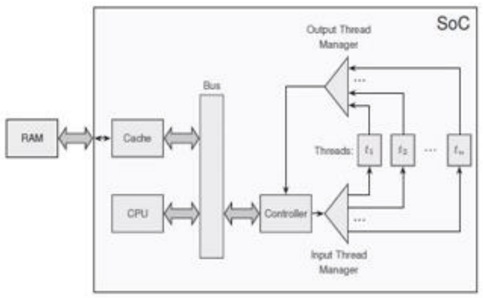


Fig. 2. An outline of a processor with reconfigurable accelerator on one chip

5. Demo Packages

Several packages have been developed in this framework: a package for calculating loads and kinematics in a gearbox, a package for analyzing logical schemes, a package for designing mechanical drives etc. Rapid prototyping of visual languages can be quite effective in the framework. The implementation of the functional aspects of the class diagram editor was under 100 lines of Java code, and specification of the graphical and interactive aspects was done visually easily in Class Editor is an example.

6. Peculiarities of a Compiler for a Processor with Programmable Architecture

The algorithms for automatic mapping of a high-level language onto multi-pipelined architectures is presented. In this paper, automatic generation of multi-pipelined electronic circuits and mapping high-level programs onto them are being discussed. The compiler contains many elements which are not inherent to regular optimizing compilers. On the base of Optimizing Parallelizing System, the compiler is developed. It would allow performing source code optimizations before generating code for the accelerator. It would be difficult to generate VHDL code for a pipeline system from a low-level register-based internal representation, such as LLVM IR of Clang compiler or Gimple and RTL of GCC compiler family.

7. An Algorithm of Mapping Loops onto a Processor

1. An innermost one-dimensional loop is being found.
2. A check is being performed to determine whether all index variables of this loop affinely depend on its counter. If not, the loop will be executed on the CPU and will not be transferred to the accelerator. Exit. Otherwise, proceed next.
3. A dependency graph is being computed.
4. It is being checked which of the classes the loop under consideration.
5. An HDL description of a circuit supporting execution of loops with the given class is being generated.
6. The programmable accelerator is being burned with the compilation result of the HDL description.
7. The optimized loop in the initial program is being replaced with a call to the programmable accelerator.

8. Classification of One-Dimensional and Multi-Dimensional Loops

A special attention is being paid to loops while optimizing a program. A classification of program loops and loop nests, based on information dependencies analysis, is required in order to develop an algorithm of mapping programs onto a processor with reconfigurable accelerator. The classification should determine for which loops a SIMD, MIMD, or pipeline architecture should be generated. Parallelizing loops with bodies containing many statements could be reduced to parallelizing loops with a few statements inside, using “loop splitting” transformation.

A problem of accelerating (parallelizing) loops which do not allow splitting, in particular, with only one assignment statement.

9. Conclusion

As a result of implementing the current project in this paper, the performance of high-level programs which allow mapping onto multi-pipeline architecture should increase, and the time for developing parallel-pipelined programs should decrease.

References

[1] G. Costagliola, A. De Lucia, S. Orefice, G. Tortora. Automatic Generation of Visual Programming Environments. IEEE Computer, 28(3):56-66, 1995.
[2] J-P. Tolvanen, M. Rossi. Metaedit+: Defining and Using domain-Specific modeling languages and Code Generators. In: OOPSLA 2003 demonstration, 2003.
[3] J. de Lara and H. Vangheluwe. Defining visual notations and their manipulation through meta-modelling and graph transformation. Journal of Visual Languages and Computing, 15(3- 4):309-330, 2004.
[4] M. Matskin and E. Tyugu. Strategies of structural synthesis of programs and its extensions. Computing and Informatics, 20:1-25, 2001.
[5] Bambu. URL: http://panda.dei.polimi.it/?page_id=31
[6] Yosi Ben-Asher, Nadav Rotem, and Eddie Shochat. Finding the Best Compromise in Compiling Compound Loops to Verilog. *J. Syst. Archit.*, 56(9):474–486, September 2010.
[7] João M. P. Cardoso and Pedro C. Diniz. Compilation techniques for Reconfigurable Architectures. Springer US, 2009.
[8] Boris Ya. Steinberg, Denis V. Dubrov, Yury Mikhailuts, Alexander S. Roshal, and Roman B. Steinberg. Automatic high-level programs mapping onto programmable architectures. In Victor Malyshkin, editor, *Parallel Computing Technologies*, volume 9251 of *Lecture Notes in Computer Science*, pages 474–485. Springer International Publishing, 2015.
[9] Tools-Impulse accelerated Technologies. http://www.impulseaccelerated.com/tools.
[10] Roman B. Steinberg. Mapping loop nests to multi pipelined architecture. *Programming and Computer Software*, 36(3):177–185, May 2010.