ATLANTIS PRESS
Advances in Computer Science Research, volume 50
4th International Conference on Electrical & Electronics Engineering and Computer Science (ICEEECS 2016)

Field modeling code generation technique for automatic parallelization mutation system

Liyichun, Jiangningkang

1. College Computer Science and Software Engineering of East China Normal University, Shanghai, China 200062

Keywords: domain modeling, code generation, parallel computing, software automation

Abstract. Unlike common imperative program language (such as java, C or ADA), compiler maintains model-based code generator, and generated component has no established method in spite of progress achieved in the field of formal verification. Several test methods are dominant in engineering practice. The paper describes a common testing system with independent tools for using code generator developed based on model. We evaluate validity of our method through testing optimization of TargetLink code generating program. TargetLink code generator is a widely accepted and complex development tool based on model automation development.

Introduction

At present, model-based code generator is not mature as established C or ADA. Technical risk of code generator is high, because they 1) are used by a quite small development team and 2) face a new version caused by an efficient technical improvement in a short period. Therefore, evidence for correctness of a formalizing code generator is impractical in practice. Productivity obtained by using code generator based on modeling tool cannot be fully enhanced. Even though nervous quality and measures has been spent on the model, code generator and code written by hand must be checked with expensive effort.

Model-based code generation

In model-based development, realization of a control algorithm is developed by progressive refinement of model, and a so-called physical model originates from functional requirements specification of software element (See upper left in Figure 1), and description of control function of physical model capturing control algorithm and depending on (continuous) input signal and (internal or external) event. Physical model typically uses floating-point arithmetic (FLP) and functional behavior used for verification of model in regard to requirements stipulated in specification.

In the field of motor machine engineering, embedded system is defined as electronic controller (ECUs). Due to limit of hardware resources, ECU needs small overhead and (advanced) programming language which can effectively use system resources. For economic reasons, microprocessor used in ECU is more suitable for 8, 16 or 32-position fixed-point processor. Therefore, physical model cannot be realized with expert manual refinement; for example, function part is distributed to different tasks and enhanced with necessary implementation details. In addition, FLP algorithm used in physical model is fixed (FXP) algorithm suitable for embedded target processor (see [5] in details). 2. Refined result is realized model. It includes all information required for code generation and establishment of effective C code allowed to be generated by code generator. Depending on development stage and purpose, it is code generated for developing calculator (host machine), and in most cases, it is a standard PC (right in Figure 1); under that circumstance, a traditional compiler/linker combination is used to translate generated code into an executable model. For target hardware—a typical evaluation committee similar to electronic control unit (ECU)—a so-called cross compiler is needed. Here, a connector and a loader are equipped with binary code and installed into embedded devices. Tool chain confirms modeling tool (editor and

simulator), and translation tools from model to code (such as code generator, (cross) compiler, connector, loader), finally, target hardware it self constitutes tool chain of code generator (Figure 1). Model-based code generator is of a major advantage developed based on model. At software implementation stage, the use of code generator leads to a significant improvement in production efficiency. Individual researches show that: development time of code generation [2] software is reduced to 20%. If manual validation process can be lowered at code level, it is reported to save 50%, which is consistent with domestic information provided by other users. In general, compared with traditional manual coding, its production efficiency has been improved by 50%.

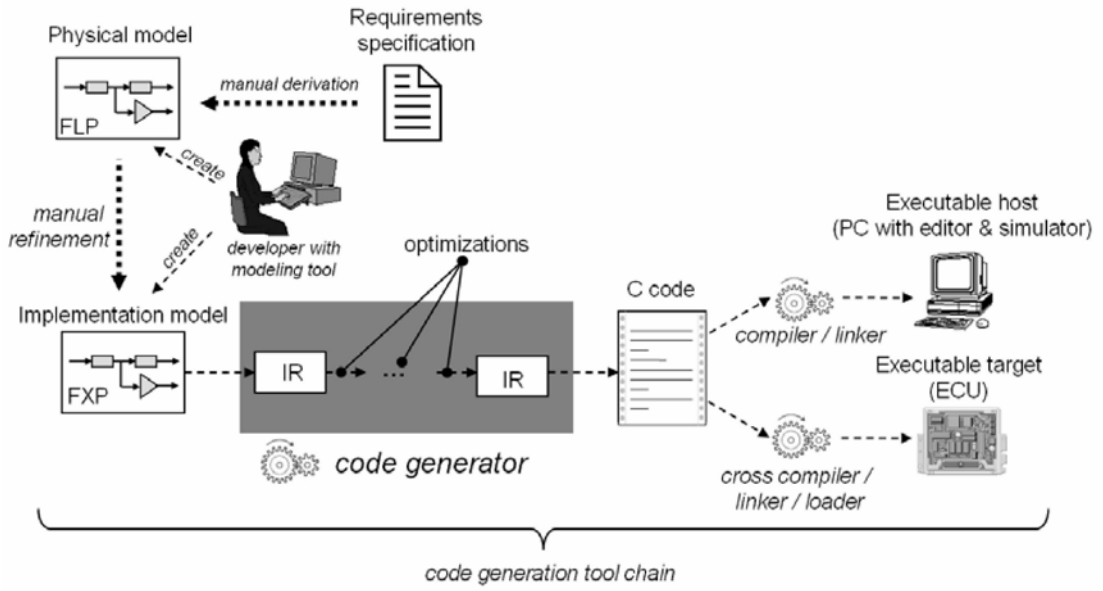

Figure 1 Model-based Code Generation Principle

Optimization of code generator

Code generation of embedded system always has resource limitation.

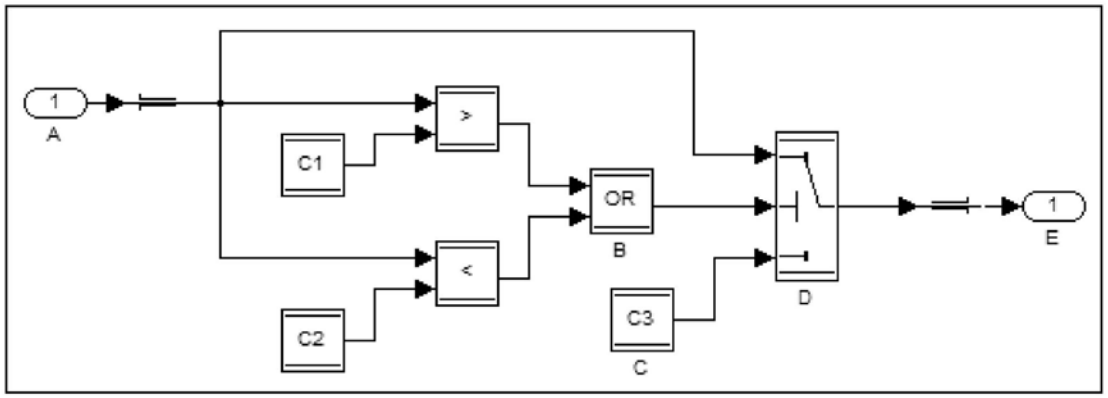

Figure 2 Control Structure of If- Then- Else on Behalf of Optimized Model

Therefore, in terms of generating effective code, optimization technique must be used whenever possible. Such as memory consumption and execution speed. Traditionally, code optimization is to translate source code into executable object code through (cross) compiler. The standard optimization is dead code elimination, constant propagation, loop circuit unwind, which is learned from interpretation of traditional compiler (see Example [8]). On using a code generator, optimization has been executed in highest form, for example, present Form 3 (IR, see middle in Figure 1) in the middle (figure), and use tremendous volume of information presented on model level.

A representative model-based optimization to create effective code model and attempt to combine different pat of model is inter-block optimization. Figure 2 shows a graph model and describes if-then-else control structure, among it, if A is not in separation distance [C2, C1], output E equals to input A. In addition, constant value C3 is propagated to output E. Control logic is

calculated by so-called switch block D, and its operation is as follows: switch propagating or the first input A or the third input E depends on value of control input B (in this case, output or operation). If control input symbol is greater than or equal to internal critical value_t, switch propagates the first input, otherwise, the third input.

When a code generator without optimization translates graph model block-to-block and introduces variable for each output block, a optimized code generator can translate model to a single if-then-else control structure:

```
if ( (A > C1) || (A < C2) )
{ E = A ;
}
else
{ E = C3 ;
  }
```

It should be noticed that all intermediate variables of expression block output have been eliminated and the logic has been inserted into exchange conditions. As control, a block translation will produce additional program code, hence result in a high RAM consumption.

On account of different code generator, code generator setting, or selection of optimization technique, realization of a specific model structure can be diverse. Therefore, different structures between a model and generated code may exist. In code generator optimizing, classification of four major optimizing/translation strategies leads to architectural difference between model and generated code: repeat. Code repeats realization of reusable model structure for each caller. Inline is such a optimization technique. Reuse. It is optimized by taking a set of equivalent but distinguishing model structure and a single reusable element for reducing size of program. A function call with different parameters is a typical example for such optimization technique. Reuse is relative to inline. Simplification. Realization of model structure is to combine or discard for obtaining efficiency (caller example, inter-block optimization) or remove useless part (such as dead path elimination). Enhancement. Additional code is generated with code protection from a potential Division by zero.

Conclusion

Difference between model-based code generator and traditional compiler is as follows: 1) Target language and source language can be executed. Therefore, executable behavior of code generator can be directly compared with simulation behavior of model. 2) Semanteme of model language is often unclear definition. Semanteme can be set depending on layout of information (such as state of position) and internal model (such as block parameter, data type processing). Therefore, semanteme is embedded in interpretation algorithm of simulator [4]. 3) Especially. Simulink defined data driving language builder constitutes a kind of new development tool. Code generator cannot simply execute step-by-step translation from model layered architecture to a abstract target language syntax tree. On the contrary, they must analyze data dependence derived from a appropriate computer sequence, which is essence of code generator.

Reference

[1] Bastoul C. Efficient code generation for automatic parallelization and optimization[C]// International Symposium on Parallel and Distributed Computing, 2003. Proceedings. IEEE, 2003:23-30.

[2] Sugawara Y, Ueno R, Homma N, et al. System for Automatic Generation of Parallel Multipliers over Galois Fields[J]. 2015:54-59.

[3] Mata L D, Pereira F, Ferreira R. Automatic parallelization of canonical loops[J]. Science of Computer Programming, 2013, 78(8):1193-1206.

[4] YUANRUI ZHANG, JUN LIU, EMRE KULTURSAY, et al. AUTOMATIC PARALLEL CODE GENERATION FOR NUFFT DATA TRANSLATION ON MULTICORES[J]. Journal

of Circuits System & Computers, 2012, 21(2):98-104.

[5] Zhao B, Ding R, Han L, et al. Code generation for accurate array redistribution on automatic distributed-memory parallelization[J]. 2014, 2(1).