On the problem of verification of source code transformations: A case study

Qiang Fu[†], Maurice Bruynooghe[†], Gerda Janssens[†], and Francky Catthoor[*]

[†] *Departement Computerwetenschappen, Katholieke Universiteit Leuven, Belgium*

{qiang,maurice,gerda}@cs.kuleuven.ac.be

[*] *IMEC vzw,Kapeldreef 75, B-3001 Hevelee, Belgium*

{catthoor}@imec.be

Abstract

A recent method that uses geometric modelling is able to check the functional equivalence of initial and transformed programs under global source code transformations. However, the method makes strong assumptions about the programming constructs and about the kind of transformations that have been applied. This work investigates to what extent multimedia applications meet these assumptions.

1 Introduction

Nowadays, complex multimedia software is running on embedded systems in various consumer electronic appliances. Due to the limited resources, software solutions have to be power efficient and adapted to specific platform architectures. This requires to apply complex transformations on the orignal source code. They are based on an analysis of maximal memory footprint, communication or access bandwidth bottlenecks, power consumption, etc. The transformations, applied manually or by tools, are global and are performed at the source code level, prior to compilation. One example of this kind of optimization method is Data Transfer and Storage Exploration methodology (DTSE [CWdG+98]).

The complexity of the optimization phase makes the resulting program prone to errors. So it is compulsory to have an independent verification process that checks the correctness of the transformed program with respect to the original one. Although the transformed program is different from the original one, the observable behavior that represents the functionality of the program should be preserved.

To do the verification manually is also prone to error and time consuming because of the complexity of the verification task. An automated method, based on geometric modelling, is described in [SBCJ03]. It is able to verify programs that have undergone loop transformations and common expression propagations. In what follows, we discuss the limitations of this method and analyse to what extent codes developed for some mutimedia applications can be verified.

2 Limitations of the present method

In general, the problem of determining program equivalence is undecidable. In order to obtain a decidable problem, it is required to focus on a decidable subset of programs. The two most important constraints imposed by [SBCJ03] are that the control-flow is static and that expressions in subscripts of array variables and loop bounds are affine.

In addition, it is required that each element of every (array) variable in the program is assigned a value only once, i.e., that the program is in dynamic single-assignment form (DSA form [VJB+03]). The DSA form of a program makes the data flow explicit, hence it simplifies the verification task. Unfortunately, general programs are hardly ever in DSA form. Fortunately, there are methods ([VJB+03]) to convert a program into DSA form.

Furthermore, the current method is an intraprocedural method. It can only verify that the functional behaviour of each procedure/function is preserved. Moreover, functions/procedures should not perform any side effects. Also pointers cannot be handled by the geometric modelling.

These restrictions restrain the class of programs and the class of transformations that can be verified. In order to assess to what extent these limitations compromise the ability to verify designs of real-life multimedia

applications, we have conducted a case study on some representative set. We have analysed which assumptions underlying the geometric modelling are violated by real code as well as the class of transformations that the designers have applied during the development. The results of this study are intended to provide guidance in extending the method so that it better addresses the verification needs of real designers.

3 Case Study

The examples we looked at are kernels of real-life multimedia applications that we obtained from IMEC. They are the following: (i). QSDPCM: Quadtree Structured Difference Pulse Code Modulation is an inter-frame compression technique for video. (ii). MP3: an implementation of the MPEG layer 3 decoder. (iii). MPlayer: a multimedia player that implements DivX decoding technologies. (iv). DAB FFT: an implementation of the FFT algorithm in the Digital Audio Broadcasting application. To improve power efficiency and performance, designers have developed different versions of each example by applying various source code transformations.

These examples are typical multimedia applications; we believe they are representative of the class of programs in this application domain. In the case study, the relevant code characteristics for the verification tool have been extracted and the applied transformations have been analyzed.

Characteristics	**QSDPCM**	**MP3**	**MPlayer**	**DAB FFT**
Multiple assignment	Yes	Yes	Yes	Yes
Non-affine expression for index and loop bounds	no	no	no	no
Pointers	no	Yes	Yes	no
Side effects	no	Yes	no	no
Data-dependent conditionals	Yes	Yes	Yes	Yes
While loops	no	no	no	Yes

Table 1: The program characteristics seen in the examples

Transformations	**QSDPCM**	**MP3**	**MPlayer**	**DAB FFT**
DTSE Preprocessing	Yes	Yes	Yes	Yes
Modification of function Call	no	Yes	Yes	Yes
Dead-code elimination	Yes	Yes	no	no
Data flow transformations	Yes	Yes	Yes	Yes
Loop transformations	Yes	Yes	Yes	Yes
Data Reuse transformations	Yes	no	no	no

Table 2: Transformations in the examples

Code characteristics. Table 1 shows the code characteristics. Only those characteristics that are not supported by the current tool have been extracted. **Multiple assignment code** is very popular in applications since it is the natural way of writing code. A preprocessing step, tranforming code to DSA, should allow the verification tool to cope with it. **Pointers** are also often used in multimedia program to provide more flexibility for programming. The geometric modelling used in the current tool cannot handle them. **Data-dependent conditionals** appear frequently because the applied computations are often dependent on input values. They cannot be handled by the verification tool. Actually, typical for the DTSE design methodology is that they are isolated in separate functions/procedures and are untouched by the subsequent global transformations. The problem with **while loops** is that the loop bounds are unknown.

Transformations. Table 2 shows the kind of transformations that have been applied by the designers. **DTSE preprocessing**, **modification of function calls**, and **dead-code elimination** are transformations that enable further optimization and are currently not the target of verification since the method aims at cleaned-up code (see Figure 1 (a)). The last three transformations aim at code optimization. **Data flow transformations** are needed to optimize algorithms or to enable optimization thereof. Currently, the method to verify global algebraic data flow transformations is in progress (see Shashidhar's article in this proceedings). But how to automatically

verify more complicated data flow transformations is still not clear at this moment. **Loop transformations** and **data reuse transformations** can be verified using the current method.

The observed code characteristics prevent direct application of the current verification method on some multimedia programs. By using an appropriate preprocessing method, the multiple assignment form can be transformed into DSA form ([VJB+03]) and pointers can be removed ([vEG01]). Changing the verification schema from Figure 1 (a) to (b) can largely extend the class of programs that can be verified. However, it is yet unclear what is the impact of such preprocessing on the complexity of the verification task. Another direction for future research is to extend the current method to cope with data-dependent conditionals in the verification and to extend it to deal with more ad hoc data flow transformations as used for example in QSDPCM.

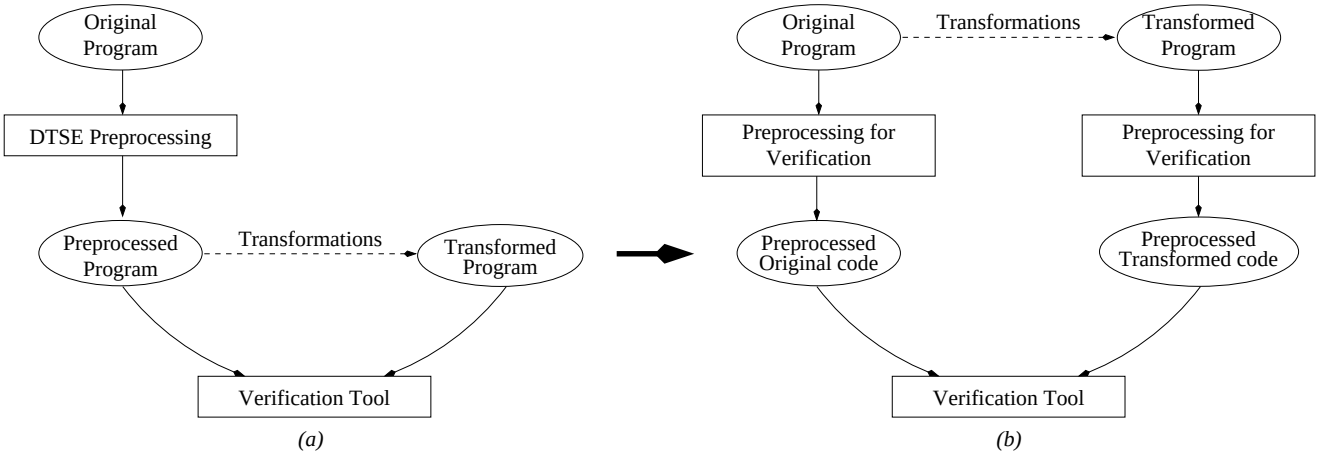

Figure 1: Verification Schema

4 Conclusions

In this abstract, we have analysed to what extent the verification method for global source code transformations of [SBCJ03] addresses the verification needs that arise in the design of energy efficient and high performance embedded mutimedia applications. At this moment, only a certain class of transformations and certain kinds of programs can be verified using this method. By closely looking at the source codes of some representative multimedia applications, we have derived which features are responsible for the distance between what is verifiable and what is to be verified. This work will serve as the guideline of our future research on extending the current verification method.

References

[CWdG+98] Francky Catthoor, Sven Wuytack, Eddy de Greef, Florin Balasa, Lode Nachtergaele, and Arnout Vandecappelle. *Custom Memory Management Methodology Exploration of Memory Organisation for Embedded Multimedia System Design.* Kluwer Academic Publishers, 1998.

[SBCJ03] K. C. Shashidhar, Maurice Bruynooghe, Francky Catthoor, and Gerda Janssens. Automatic functional verification of memory oriented global source code transformations. In *IEEE International High Level Design Validation and Test Workshop*, 2003.

[vEG01] Robert A. van Engelen and Kyle A. Gallivan. An efficient algorithm for pointer-to-array access conversion for compiling and optimizing dsp applications. In *the 2001 International Workshop on Innovative Architectures for Future Generation High-Performance Processors and Systems (IWIA 2001)*, pages 80–89, Maui, Hawaii, January 2001.

[VJB+03] Peter Vanbroekhoven, Gerda Janssens, Maurice Bruynooghe, Henk Corporaal, and Francky Catthoor. A step towards a scalable dynamic single assignment conversion. K.U.Leuven, Department of Computer Science, Report CW 360, April, 2003.