Computer-Aided Recoding for Multi-Core Systems

Rainer Dömer

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697
e-mail : doemer@uci.edu

Abstract - The design of embedded computing systems faces a serious productivity gap due to the increasing complexity of their hardware and software components. One solution to address this problem is the modeling at higher levels of abstraction. However, manually writing proper executable system models is challenging, error-prone, and very time-consuming. We aim to automate critical coding tasks in the creation of system models.
This paper outlines a novel modeling technique called *computer-aided recoding* which automates the process of writing abstract models of embedded systems by use of advanced computer-aided design (CAD) techniques. Using an interactive, designer-controlled approach with automated source code transformations, our omputer-aided recoding technique derives an executable parallel system model directly from available sequential reference code. Specifically, we describe three sets of source code transformations that create structural hierarchy, expose potential parallelism, and create explicit communication and synchronization.
As a result, system modeling is significantly streamlined. Our experimental results demonstrate the shortened design time and higher productivity.

I Introduction

The design of embedded computing systems, such as video-enabled mobile phones and reliable medical devices, which are ubiquitous and pervasive in our daily life, is extremely challenging due to many hard design constraints, including strict timing, multi-core functionality, low power, low price, and short time-to-market. Moreover, we face a growing design productivity gap due to the increasing complexity of the embedded hardware and software components.

A. Motivation

The International Technology Roadmap for Semiconductors (ITRS) lists modeling at higher levels of abstraction as key to overcome the productivity gap [1]. Much like the quality of an architectural blue-print determines the quality of the resulting building, the model of an embedded system is the key to its successful implementation. However, creating and writing proper executable system models is challenging, error-prone, and very time-consuming.

As illustrated in Figure 1, there exists a serious gap of automation in top-down system design flows. Our experiments with a simple MP3 decoder application have shown that more than 90% of the overall design time was spent in creating and editing the abstract system model [2]. Other studies also confirm that the model specification phase is a serious bottleneck in embedded system design.

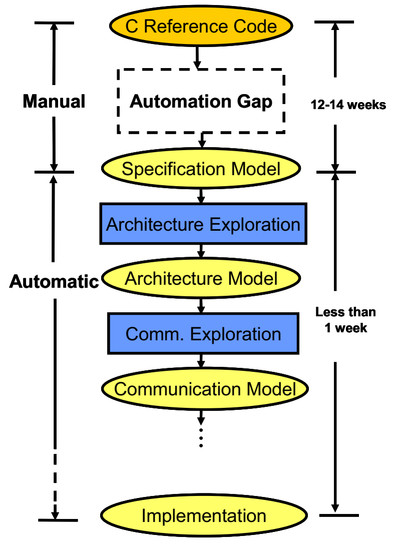

Fig. 1: Automation gap in top-down system design

II. Computer-Aided Recoding

In this work, we bridge the automation gap in specification modeling by *computer-aided recoding,* a model generation and remodeling technique that automates various steps in the process of writing embedded system models. Computer-aided recoding is a designer-controlled approach that relies on automated source code transformations available to the system designer in form of a "smart" integrated development environment [2]. In this approach, the designer makes the decisions, whereas the tool automatically transforms the source code.

Specifically, we present three sets of automated source code transformations, namely to create structural hierarchy, to expose potential parallelism, and to create explicit communication and synchronization.

A. Creating structural hierarchy

In given C reference code, only a flat structure of global variables and global functions is available. However, system-level description languages require the specification of classes (modules/behaviors and channels) which represent the hierarchical block diagram structure of the design model.

In order to create such an explicit structural hierarchy, we have developed several source code transformations that allow the designer to properly organize the initially unstructured “flat” application code [5].

In particular, wie utilize the given function call hierarchy in the given C code and convert it into a structural hierarchy of interconnected blocks. This is essentially a step-wise model transformation.

Our approach is hierarchical encapsulation. We utilize the given function call tree and convert each function step-by-step into a behavioral block. As illustrated in Figure 2, our approach works in a top-down manner.

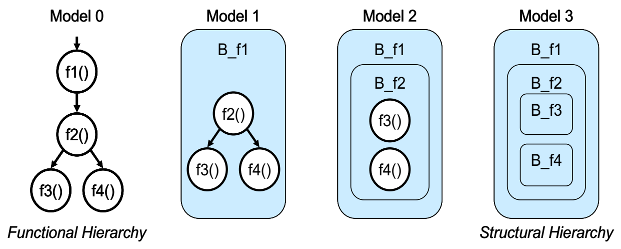

Figure 2: Example for creating structural hierarchy

We start with the root function, i.e. the main function of the C program. We then continue step by step down to the leaf functions, until all functions are converted to behavioral blocks.

B. *Exposing potential parallelism*

After the structural hierarchy has been established in the model, we focus on exposing potential parallelism in the application. The point is, without explicit parallelism in the system model, system-level exploration is very restricted in particular in mapping different block to concurrent executing processing elements.

Without going into too much detail here, there are several recoding tasks necessary which involve (1) the partitioning of code, (2) the partitioning of data elements, and (3) the explicit synchronization of dependent blocks. These steps are illustrated in Figure 3.

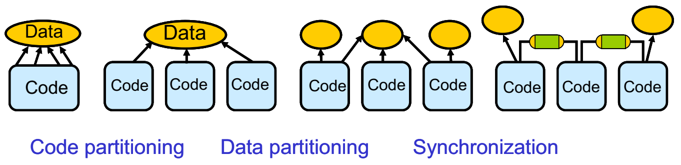

Figure 3: Code and data partitioning, and explicit synchronization

Each of these conceptual transformations consists in our approach of several smaller steps that involve the recoding transformations of (1) loop splitting, (2) cumulative access type analysis, (3) partitioning of vector dependents, and (4) the synchronization of data-dependent variables. Details of these transformations are beyond the scope of this paper but can be found in [6].

As a result of the performed code and data partitioning tasks, a parallel and flexible system model has been created which, however, still requires explicit specification of communication and synchronization channels.

C. *Creating explicit communication and synchronization*

Figure 4 illustrates the need for explicit communication in the system model. With the initial shared memory model in the given C code (i.e. global variables), only a single platform architecture is available for mapping, namely one that has a single shared memory. However, with global variables eliminated and communication properly wrapped into channels, many different platform architectures can be chosen (including the traditional shared-memory architecture).

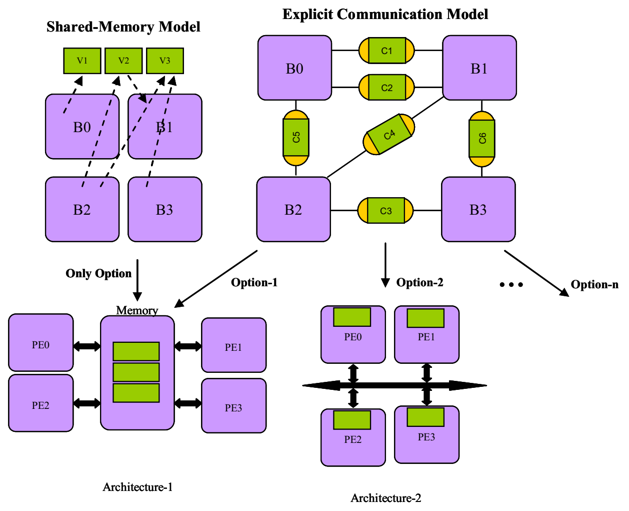

Fig. 4: Explicit communication enables flexible mapping to different architectures

Figure 5 shows our approach to establish the desired explicit communication and synchronization.

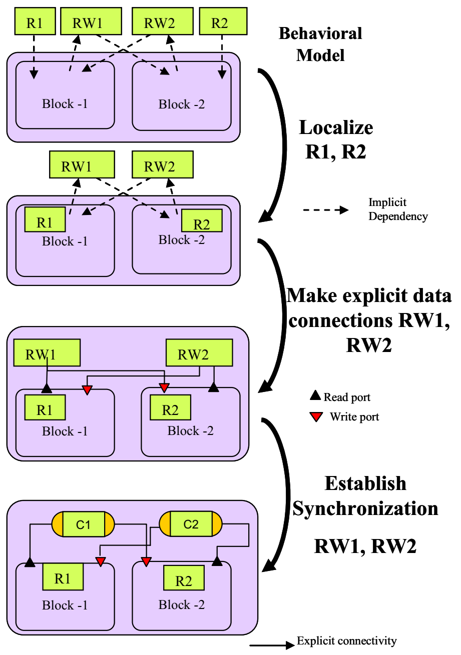

Figure 5: Minimizing global variables and establishing explicit communication and synchronization

Again, the benefit of use explicit communication through message passing channels instead of global variables is that this clearly specifies the needed synchronization scheme and thus guides system exploration tools in their design space exploration.

Our approach, as illustrated in Fig. 5, first localizes variables that are only accessed locally. Variables only accessed by a single block, are localized into that block. Variables accessed by a set of blocks, are localized into the lowest parent behavior containing those blocks.

After variables are moved down to the lowest possible level in the hierarchy, their accesses are made explicit through port connections. This allows the direction (in/out/inout) to be explicitly stated which simplifies dependency analysis.

Finally we create a typed synchronization channel and replace the ports corresponding to the original variable with interfaces of proper type, matching the type of the channel. At the same time, we modify each access to the variable with an explicit method call to the appropriate interface function of the channel. It is this step, where read() / receive() and write() / send() function calls are created in the model.

III. Designer-Controlled Source Recoder

We have implemented our recoding approach in a prototype of an interactive *Source Recoder* [2].

Our recoder is designer-controlled and based on interactive transformations. Unlike traditional automatic compilers, we use interactive source-level transformations which can be chained together by the designer to expose desired properties in the model through code restructuring.

For example, to expose explicit data parallelism in the model, the designer uses her/his application knowledge and invokes re-coding transformations to split loops into code partitions, analyze shared data accesses, split vectors of shared data, localize variable accesses, and finally synchronize accesses to shared data by inserting communication channels. Further, similar code partitioning and data structure re-structuring transformations can be used to expose pipelined parallelism in the model. Additionally, code restructuring to prune the control structure of the code and pointer recoding to replace pointer expressions can be used to enhance the analyzability and synthesizability of the models.

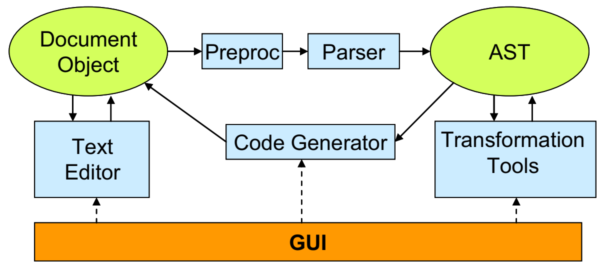

Fig. 6. Designer-controlled Source Recoder [8]

Our Source Recoder is an intelligent union of editor, compiler, and transformation and analysis tools. The conceptual organization of the source recoder is shown in Figure 6. It consists of a text editor maintaining a document object and a set of analysis and transformation tools working on an Abstract Syntax Tree (AST) of the design model. Preprocessor and parser apply changes in the document to the AST, and a code generator synchronizes changes in the AST to the document object. The invoked analysis and transformation tasks are performed and presented to the designer *instantly*. The designer can also make changes to the code by typing and these changes are applied to the AST *on-the-fly*, keeping it updated all the time.

Unlike other program transformation tools, our approach provides complete control to the designer to generate and modify the design model suitable for the design flow. As such, we rely on the designer to concur, augment or overrule the analysis results of the tools, and use the combined intelligence of the recoder and the designer for the modeling task.

IV. Experiments and Results

To measure the benefits of our approach and to obtain realistic productivity gains, we have conducted a real-life design experiment with a class of graduate students. The students were first given instructions to manually implement several types of recoding tasks on a given MP3 decoder model. Next, they were then asked to measure the time needed to perform these code transformations manually. Following that, we introduced the students to our automatic source recoder and asked them to implement the same transformations using the automatic functions available, again while measuring their time. Thus, the same students provided both manual and automatic times needed to implement the transformations.

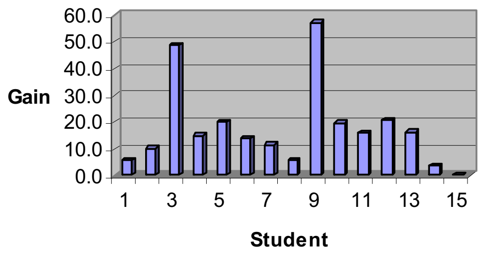

Figure 7: Productivity gains achieved for Function-to-Behavior recoding transformation.

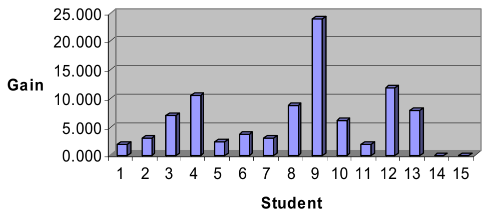

Figure 8: Productivity gains achieved for Statements-to-Behavior recoding transformation.

Based on the reported times by the students, we have analyzed and assessed the productivity factors that were achieved. Figures 7 and 8 plot the productivity factors reported by different students for two different recoding transformations, namely the conversion of a function to a behavior and the conversion of a sequence of code statements into a behavior.

Clearly, the gains achieved vary depending on the designer, the type of transformation, and also due to our still imperfect source recoder implementation. Despite the variability in the gained productivity factors, we can conclude that our automatic recoding transformations indeed result in significant productivity gains and are effective in reducing the overall system design time.

V. Summary and Conclusions

In summary, using automated source code transformations built into a design model aware editor, the designer can quickly recode a system model to create a parallel and flexible specification that can be easily mapped onto a multi-core platform. Our experimental results show a great reduction in modeling time and significant productivity gains up to two orders of magnitude over manual recoding.
Computer-aided recoding can derive an executable parallel system model directly from available sequential reference code. Automatic source code transformations relieve the system designer from complex code analysis and tedious coding tasks, allowing uninterrupted focus on system modeling and design space exploration. As a result, modeling writing is streamlined, enabling a shorter design time and higher productivity, as well as quality improvements in the end design.

Acknowledgments

The author would like to thank Pramod Chandraiah for the implementation of the presented recoding techniques and for his help in conducting the experiments.

This work has been supported in part by funding from the National Science Foundation (NSF) under research grant NSF Award #0747523. The author thanks the NSF for the valuable support. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] International Semiconductor Industry Association: *"International Technology Roadmap for Semiconductors (ITRS)"*. http://www.itrs.net, 2007.

[2] Chandraiah, P., Dömer, R.: *"An Interactive Model Re-Coder for Efficient SoC Specification"*, Proceedings of the International Embedded Systems Symposium, "Embedded System Design: Topics, Techniques and Trends" (ed. A. Rettberg, M. Zanella, R. Dömer, A. Gerstlauer, F. Rammig), Springer, Irvine, California, May 2007.

[3] Chandraiah, P., Dömer, R.: *"Designer-Controlled Generation of Parallel and Flexible Heterogeneous MPSoC Specification"*, Proceedings of the Design Automation Conference 2007, San Diego, California, June 2007.

[4] Chandraiah, P., Dömer, R.: *"Pointer Re-coding for Creating Definitive MPSoC Models"*, Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis, Salzburg, Austria, September 2007.

[5] Chandraiah, P., Dömer, R.: *"Automatic Re-coding of Reference Code into Structured and Analyzable SoC Models"*, Proceedings of the Asia and South Pacific Design Automation Conference 2008, Seoul, Korea, January 2008.

[6] Chandraiah, P., Dömer, R.: *"Code and Data Structure Partitioning for Parallel and Flexible MPSoC Specification Using Designer-Controlled Re-Coding"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems vol. 27, no. 6, pp. 1078-1090, June 2008.