

## Guest Editorial

As the trend toward comprehensive use of electronic information processing continues, more and more *embedded systems* are being designed and used. Examples of such systems include mobile telecommunication devices, information appliances, automotive electronic equipment, and information technology equipment in smart homes. These systems have a number of characteristics in common. Embedded systems typically meet the majority of the following criteria:

- They have to meet hard deadlines for their response time.
- They do not come with a keyboard, a large screen, and a mouse; they hide information processing from the user.
- They have to be cost-, area-, weight- and/or power-efficient.
- They have to be dependable.

To provide the required flexibility, more and more embedded systems are software based. The generation of embedded software requires new software generation techniques that take the special characteristics of embedded systems into account. One of these characteristics is the use of processors optimized for certain application domains or even for certain applications. The main motivation for specialized processors is the need to provide efficient solutions. As an example, processors for digital signal processing (DSP) frequently provide

- specialized multiply/accumulate instructions
- saturating arithmetic
- heterogenous register sets
- specialized addressing modes
- limited parallelism (more recently also very long instruction word (VLIW)-type of parallelism)

If these features are not exploited in compilers, inefficient code is the result and designers have to use assembly languages. To get around this uncomfortable situation, the design of specialized compilation techniques for embedded processors has started. General problems of embedded software also include the need for good specification languages, for fast simulation, for verification, for interprocess communication, and for interfaces to real-time operating systems, to name just a few.

Stimulated by the CHIPS project on compilation for embedded processors, a first European workshop on compilers for embedded processors was held at Schloss Dagstuhl, Germany, in 1994. Due to its success, it was followed by a series of similar workshops. The title of the first three workshops was *Workshop on Code Generation for Embedded Processors*. From the fourth workshop onward, the scope also included software generation for embedded processors in general. Hence, the acronym SCOPEs (software and code generation for

embedded processors) was used from the third workshop onward. The dates and locations of the workshops are as follows:

Workshop	Location	Dates
1	Schloss Dagstuhl, Wadern, Germany	Aug. 31st to Sept. 2nd, 1994
2	Leuven, Belgium	March 18th to March 20th, 1996
3	Witten, Germany	March 4th to March 6th, 1998
4	St. Goar, Germany	Sept. 1st to Sept. 3rd, 1999
5	St. Goar, Germany	March 20th to 22nd, 2001

The fourth workshop was held at a very scenic castle hotel called Schloss Rheinfels at St. Goar, Germany. The view from the conference room onto the river Rhine and its valleys was so nice that it was decided to hold the fifth workshop at the same place.

One of the main goals of the workshop is to stimulate the discussion between participants. Therefore, attendance has so far been restricted to groups of people who have already worked in the area. This goal has been achieved very nicely so far. Another characteristic is that the workshop does not try to compete with well-established publication channels, such as conferences and journals. Rather, the best contributions at the workshop are considered as candidates for a special publication. As a result of the first workshop, the book *Code Generation for Embedded Processors* (edited by Gert Goossens and myself) was published by Kluwer. Unfortunately, time constraints prevented publishing papers from the second workshop. The best papers from the third workshop were published in the April 1999 issue of *Design Automation for Embedded Systems*. The current special issue includes the best papers from the fourth workshop. They have been reviewed by an excellent panel of international reviewers consisting of A. Nicolau (Irvine), S. Malik (Princeton), J. van Meerbergen (Eindhoven), B. Wess (Vienna), and R. Wilhelm (Saarbrücken).

The first paper is entitled “Code Size Minimization and Retargetable Assembly for Custom EPIC and VLIW Instruction Formats.” It was written by **Shail Aditya**, **Scott Mahlke**, and **B. Ramakrishna Rau** of Hewlett-Packard. They describe the PICO system, a system for automatically designing and programming very long instruction word (VLIW) architectures. They focus on techniques for generating compact code for their architectures. Some of these techniques are applied during architecture design; others are applied during program generation. With the current trend toward VLIW architectures, this paper is of potential interest to a major number of designers working on these architectures.

The title of the second paper is “Constraint Analysis for Code Generation: Basic Techniques and Applications in FACTS.” The authors are **Koen van Eijk**, **Bart Mesman**, **Carlos A. Alba Pinto**, **Qin Zhao**, **Marco Bekooij**, **Jef van Meerbergen**, and **Jochen Jess**. The work, a result of cooperation between Philips Research Laboratories and Eindhoven University, Netherlands, is concerned with code generation for embedded processors. Most of these processors are capable of performing a number of operations in parallel. Usually, a separate component of a compiler, called scheduler, is responsible for deciding the order in which operations of a given source program will be executed on the processor.

Due to complexity reasons, this scheduler has only a very limited interaction with other compiler components. As a result, schedulers sometimes take decisions that turn out to be poor as soon as more information (e.g., information about registers) is considered. The key advantage of FACTS is that it considers information about hardware resources (such as registers) already during scheduling. Due to this coupling, better code quality can be achieved. The proposed technique can be a valuable enhancement to many scheduling algorithms.

The third paper, “Graph-Based Code Selection Techniques for Embedded Processors,” is by **Rainer Leupers** and **Steven Bashford**. Both are with the University of Dortmund, Germany. One of the essential functions of any compiler is to select machine instructions for implementing operations specified in the source program. In general, graphs would be required to represent the flow of values between these operations. For common subexpressions in the source program, these graphs describe the flow of the value represented by the subexpression to all operations needing that value. Unfortunately, it has been found that the optimal selection of machine instructions implementing graphs is computationally very expensive (NP-hard). It is therefore usually avoided by splitting graphs into trees and selecting machine instructions for trees. This approach is computationally efficient but leads to suboptimal code. Leupers and Bashford describe techniques for graph-based machine instruction selection that lead to better code than tree-based techniques but are still computationally efficient. These techniques are also capable of exploiting subword parallelism in the form of multimedia instructions.

One very important aspect of embedded software design is that of fast simulation of such software. This is required especially during the prototyping stage. During prototyping, it is very common that large amounts of data have to be simulated. Therefore, *extremely* fast simulation is needed. One approach for obtaining a large simulation speed is described by the authors of the fourth paper, “Retargetable Compiled Simulation of Embedded Processors Using a Machine Description Language.” Its authors are **Stefan Pees**, **Andreas Hoffmann**, and **Heinrich Meyr** of the Technical University (RWTH) at Aachen, Germany. The key toward high simulation speeds is *compiled simulation*. Compiled simulation has already been used for hardware description languages and is now used for simulating digital signal processing (DSP) applications fast, using the retargetable simulator LISA. LISA is between  $37\times$  and  $170\times$  faster than a commercial simulator. The proposed techniques are expected to be very important for simulating embedded software in general.

I hope that this special issue will be one of your key references for work on embedded software generation and simulation. Please enjoy reading the special issue!

*PETER MARWEDEL*  
*Guest Editor*  
*Dortmund, May 2000*