Fast Evaluation Methodology for Automatic Custom Hardware Prototyping

Cecilia González*, Daniel Jiménez-González†, Xavier Martorell*, Carlos Álvarez† and Georgi Gaydadjiev‡

*Barcelona Supercomputing Center
{cecilia.gonzalez,xavier.martorell}@bsc.es

†Universitat Politecnica de Catalunya
{djimenez,calvarez}@ac.upc.edu

‡Technical University of Delft
g.n.gaydadjiev@tudelft.nl

I. INTRODUCTION

Hardware customization for scientific applications has shown a big potential for reducing power consumption and increasing performance. In particular, the automatic generation of ISA extensions for General-Purpose Processors (GPPs) to accelerate domain-specific applications is an active field of research to accelerate [1], [2]. Those domain-specific accelerated processors are mostly evaluated in simulation environments due to technical and programmability issues while using real hardware. There is no automatic mechanism to test those custom units in a real hardware environment. In this paper we present a toolchain that can automatically identify candidate parts of the code suitable for reconfigurable hardware acceleration. We validate our toolchain using ClustalW.

II. AUTOMATIC PROTOTYPING AND EVALUATION

The main objective of our work is to provide an automated toolchain for the generation and evaluation of a domain-specific processor architecture. The diagram of this architecture is shown in Figure 1.
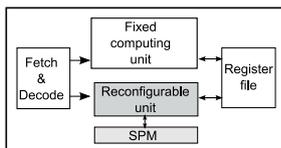

Fig. 1.   The target domain-specific architecture.

The targeted architecture is composed of a fixed ISA computing unit, which is extended with new ISA instructions mapped on the reconfigurable unit that implements and executes those extensions. The extensions are translated into hardware descriptions and mapped in the customizable area of the processor, through reconfiguration technologies. Additionally, there may be a flexible Scratchpad Memory (SPM) connected directly to the main memory of the system through DMA transfers.

The main drawback of testing such architecture is that, for the best of our knowledge, there is not a fast prototyping platform. In this paper, we present our current work on an automatic toolchain to generate specific units for a domain of applications with fast testing.

In Figure 2 we show an outline of the main parts that make up the process of prototyping. In steps 1, 2 and 3, we identify the new ISA extensions for our domain-specific processor. In steps 4 and 5 we generate, respectively, the necessary hardware and binary code to use those new extensions in our applications. The description of the exact steps is as follows:
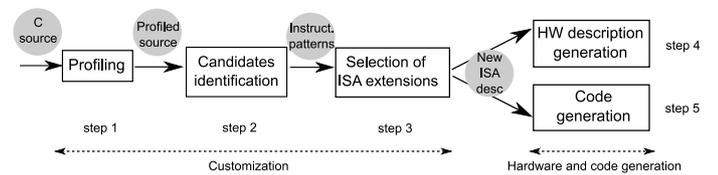

Fig. 2.   Outline of the automated process of prototype generation.

A. Customization: ISA extensions detection

1) **Profiling**. The source code of the application is profiled to obtain the frequency of execution of every code section.
2) **Candidates identification**. The application is represented as the Data Flow Graph (DFG) of the sequence of instructions. The DFG is examined at the basic block level to get subgraphs of basic instructions that meet architectural constraints, e.g. the number of inputs and outputs or the kind of instruction. Each of those identified subgraphs is a candidate as a customized instruction.
3) **Selection of ISA extensions**. The final selection is done using a greedy algorithm. The search is guided by a function that uses the information extracted during the profiling. This function tries to maximize the gain of the new instruction proposed, depending on the metric that has been chosen. The maximum number of new instructions that are selected is limited by the area available in the reconfigurable hardware.

B. Hardware and code generation

4) **Generation of the hardware description**. Once each new instruction is selected, its hardware description is generated. The new unit is placed into the reconfigurable area of the processor, as shown in Figure 1.
5) **Code generation**. The compiler for the target architecture is parametrized to detect the code patterns that match the new instructions. Then, the compiler generates the code using the new ISA extensions.

III. Current Implementation

The prototyping and evaluation platform for our current implementation is MOLEN [3]. MOLEN is a polymorphic processor composed of two main components: a core processor, that performs tasks like a GPP, and a reconfigurable processor with a Custom Computing Unit (CCU), that runs as a coprocessor. In the MOLEN programming paradigm, applications run mainly on the GPP. However, some parts are implemented in the reconfigurable hardware of the CCU to speed-up the overall application.

Profiling and identification of candidates, steps 1 and 2 in Figure 2 respectively, have been implemented within the Trimaran framework [4]. Selection of ISA extensions (step 3), is a standalone program in the toolchain with different guiding functions for the selection of the new instructions.

The hardware description generation (step 4) is done using the DWARV toolset [5], which is a C to VHDL translator specific for MOLEN. It translates the functions preceded by the *to_dfg* pragma directive to VHDL that can be integrated in the MOLEN platform. For instance, in order to customize the MOLEN platform to accelerate the source code of the program in Figure 3.a, we automatically generate the code shown in Figure 3.b. Therefore, for each selected new instruction, we generate a function equivalent with the source code lines that include the new instruction, annotated to be transformed into the hardware description of a CCU.

The code generation (step 5 in the Figure 2) is done using the compiler that targets MOLEN. That compiler identifies the pieces of code that are going to run on the reconfigurable area with the *call_fpga* pragma directive and generates the instructions needed to start the execution of the CCU. We feed the MOLEN compiler with a C code that has been automatically modified with pragmas and calls to the CCUs equivalent to the new instructions (see Figure 3.c).

```
a) original_source_code_line
   /* Including the new instruction */

b) #pragma to_dfg
   int funct_example(int param_example)
   {  original_source_code_line }
   ...
   funct_example(param);

c) #pragma call_fpga CCU_funct_example
   int CCU_funct_example(int param_example)
   {  original_source_code_line }
   ...
   funct_example(param);
```

Fig. 3. (a) Original code. (b) Code for DWARV toolset. (c) Code for MOLEN compiler.

We have tested our toolchain with a bioinformatic application, ClustalW, that is a sequence alignment program. The MOLEN processor has been deployed on a XUP with a FPGA Virtex II Pro. Figure 4 shows the speed-up that every single CCU detected can achieve compared to its software equivalent running on the PowerPC embedded in the FPGA Virtex II Pro.

As it can be seen, a single CCU can achieve up to 8.53x of speed-up. When the best subset of those new instructions is used the overall application achieves a speed-up of 2.15x.
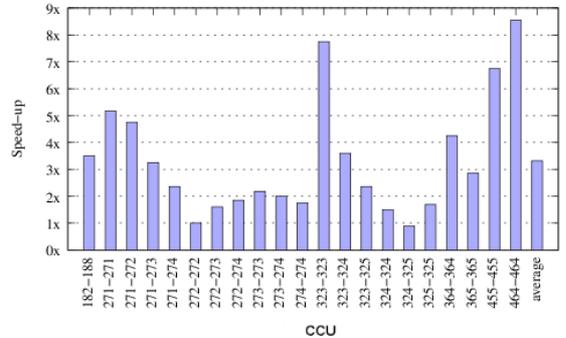

Fig. 4. Speed-up per CCU compared to the software version. The x-axis shows CCU identifiers.

IV. Conclusions and Future Work

This work presents a preliminary toolchain to automatically generate hardware prototypes and test them. Initial results prove that our framework delivers an speed-up over 2x for the tested application on the MOLEN platform. To overcome the limitations of our current implementation, we are changing the prototyping platform to a new one based on OpenSPARC T1 processor. That processor has coprocessor hardware and ISA support. However, Trimaran does not target SPARC. Therefore, we are porting our ISA detection algorithms to the LLVM compiler infrastructure, since it can generate SPARC object files.

References

[1] Diviya Jain, Anshul Kumar, Laura Pozzi, and Paolo Ienne, "Automatically Customising VLIW Architectures with Coarse Grained Application-specific Functional Units," in *Proceedings of the 8th International Workshop on Software and Compilers for Embedded Systems*, 2004.

[2] Nathan T. Clark and Hongtao Zhong, "Automated custom instruction generation for domain-specific processor acceleration," *IEEE Trans. Comput.*, vol. 54, no. 10, pp. 1258–1270, 2005, Member-Scott A. Mahlke.

[3] S. Vassiliadis, S. Wong, G. N. Gaydadjiev, K.L.M. Bertels, G.K. Kuzmanov, and E. Moscu Panainte, "The molen polymorphic processor," *IEEE Transactions on Computers*, pp. 1363– 1375, November 2004.

[4] "Trimaran: An infrastructure for research in backend compilation and architecture exploration," http://www.trimaran.org, August 2008.

[5] Y. D. Yankova, G.K. Kuzmanov, K.L.M. Bertels, G. N. Gaydadjiev, Y. Lu, and S. Vassiliadis, "Dwarv: Delftworkbench automated reconfigurable vhdl generator," in *In Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL07)*, August 2007, pp. 697–701.