

# Preliminary Work on a Mechanism for Testing a Customized Architecture

Cecilia González<sup>\*,1</sup>, Daniel Jiménez-González<sup>†,2</sup>, Xavier Martorell<sup>\*,†,1</sup>, Carlos Álvarez<sup>†,2</sup> and Georgi Gaydadjiev<sup>‡,3</sup>

<sup>\*</sup> *Barcelona Supercomputing Center;*

<sup>†</sup> *Universitat Politècnica de Catalunya;*

<sup>‡</sup> *Technical University of Delft;*

## 1 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. Those domain-specific customized processors are mostly evaluated in simulation environments due to technical and programmability issues while using real hardware. There is no automatic mechanism to test ISA extensions in a real hardware environment. In this paper we present a toolchain that can automatically identify candidate parts of the code suitable for acceleration to test them in a reconfigurable hardware. We validate our toolchain using a bioinformatic application, ClustalW, obtaining an overall speed-up over 2x.

## 2 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.

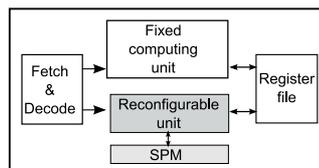


Figure 1: The target domain-specific architecture.

<sup>1</sup>{cecilia.gonzalez,xavier.martorell}@bsc.es

<sup>2</sup>{djimenez,calvarez}@ac.upc.edu

<sup>3</sup>g.n.gaydadjiev@tudelft.nl

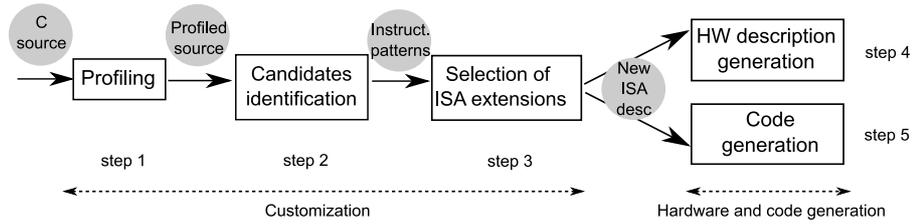


Figure 2: Outline of the automated process of prototype generation.

The target architecture is composed of a fixed ISA with fixed computing units, which is extended with new ISA instructions. Those extensions are translated into hardware descriptions and mapped in the customizable area of the processor, which is the reconfigurable unit in Figure 1. Additionally, there may be a Scratchpad Memory (SPM) connected directly to the main memory of the system through DMA transfers.

The main drawback of testing such architecture is that, to the best of our knowledge, there is not a fast prototyping platform to support that testing. 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 detailed description of the steps is as follows:

## 2.1 Customization: ISA extensions detection

1. **Profiling.** The source code of the application is profiled to obtain the number of executions of every code section.
2. **Candidates identification.** The application is represented as the Data Flow Graph (DFG) of the sequence of instructions in an intermediate representation. 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 customized instruction candidate.
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 performance gain of the applications due to 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. The estimated area of the new instructions is computed in step 4.

## 2.2 Hardware and code generation

4. **Generation of the hardware description.** Once each new instruction is selected, the hardware description of the unit that executes it is generated. The new custom unit is placed into the reconfigurable area of the target architecture, 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.

### 3 Current Implementation

The prototyping and evaluation platform for our current implementation is MOLEN [VWG<sup>+</sup>04]. 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 accelerate the overall application.

Profiling and identification of candidates, steps 1 and 2 in Figure 2 respectively, have been implemented within the Trimaran framework [tri08]. Selection of ISA extensions (step 3), is a standalone program in the toolchain with different guiding functions for the selection of the new ISA extensions candidates.

The hardware description generation (step 4) is done using the DWARV toolset [YKB<sup>+</sup>07], 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. Then the function is annotated to be transformed into the hardware description of a CCU.

The code generation (step 5 in the Figure 2) is done using a compiler that targets MOLEN. That compiler generates the instructions needed to start the execution of the CCU for those parts of code annotated with the *call\_fpga* pragma directive. 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);
```

Figure 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 over the 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 ISA extensions is used, the overall application speed-up is 2.15x.

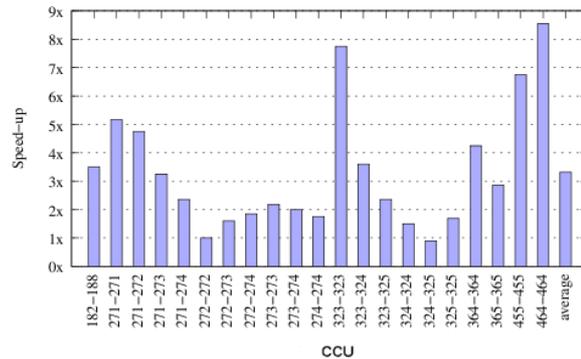


Figure 4: Speed-up per CCU over the software equivalent. The x-axis shows CCU identifiers.

## 4 Conclusions and future work

This work presents a preliminary toolchain to automatically generate hardware prototypes and test them. Initial results prove that our toolchain 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.

## Acknowledgments

We would like to thank Yana Yankova from TU Delft for her support using the DWARV toolset and Pan Yu from the NU of Singapore for his help with the extensions to Trimaran.

The researchers at BSC-UPC were supported by the Spanish Ministry of Science and Innovation (no. TIN2007-60625, CSD2007-00050 and TIN2006-27664-E), the European Commission in the context of the SARC project (no. 27648), the HiPEAC Network of Excellence (no. IST-004408), and the Xilinx University Program.

## References

- [tri08] Trimaran: An infrastructure for research in backend compilation and architecture exploration. <http://www.trimaran.org>, August 2008.
- [VWG<sup>+</sup>04] 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*, pages 1363–1375, November 2004.
- [YKB<sup>+</sup>07] 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)*, pages 697–701, August 2007.