Experiments with the LARA Aspect-Oriented Approach

José G. F. Coutinho
Department of Computing,
Imperial College London,
180 Queen's Gate,
London SW7 2BZ,
United Kingdom
gabriel.figueiredo@imperial.ac.uk

Tiago Carvalho, Sérgio Durand, João M. P. Cardoso
Universidade do Porto,
Faculdade de Engenharia (FEUP),
Dep. de Engenharia Informática
Rua Dr. Roberto Frias, s/n
4200-465 Porto, Portugal
tiago.diogo.carvalho@fe.up.pt,
sergiodurand@gmail.com, jmpc@acm.org

Ricardo Nobre, Pedro C. Diniz
INESC-ID,
Rua Alves Redol 9
1000-029 Lisboa, Portugal
ricardo.nobre@gmail.com,
pedro@esda.inesc-id.pt

Wayne Luk
Department of Computing,
Imperial College London,
180 Queen's Gate,
London SW7 2BZ,
United Kingdom
w.luk@imperial.ac.uk

Abstract

This demonstration presents a novel design-flow and aspect-oriented language called LARA [1], which is currently used to guide the mapping of high-level C application codes to heterogeneous high-performance embedded systems. In particular, LARA is capable of capturing complex strategies and schemes involving: hardware/software partitioning, code specialization, source code transformations and code instrumentation. A key element of LARA, and a distinguishing feature from existing approaches, is its ability to support the specification of non-functional requirements and user knowledge in a non-invasive way in the exploration of suitable implementations. The design-flow incorporates several tools, such as a LARA frontend, a hardware/software partitioning tool, an aspect weaver, cost estimator, and a source-level transformation engine. All these components can be coordinated as part of an elaborate application mapping strategy using LARA.

In this demonstration, we illustrate how non-functional cross-cutting concerns such as runtime monitorization and performance are codified and described in LARA and how the weaving process affects selected applications. Furthermore, we also explain how third-party tools, such as *gprof*, can be incorporated into the design-flow and aspect description, for instance, to affect the hardware/software partitioning process. We demonstrate how LARA can be used to extract run-time information, such as range values of variables, and can control code transformations and compiler optimizations addressing customized implementations of the corresponding computations on FPGAs.

Categories and Subject Descriptors D.3.3 [**Programming Languages**]: Language Constructs and Features – Frameworks. D.3.3 [**Programming Languages**]: Processors – Compilers, Retargetable Compilers, Optimization, Code Generation. C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems, Microprocessor/microcomputer applications. B.7.1 [**Integrated circuits**]: Types and Design Styles – Algorithms implemented in hardware.

General Terms Design, Experimentation, Languages.

Keywords Aspect-Oriented Programming; Compilers; Reconfigurable Computing; FPGAs; Embedded Systems; Domain-Specific Languages

AOSD'12, March 25–30, 2012, Potsdam, Germany.

1. Description

Mapping applications written in high-level languages like C to heterogeneous multi-core embedded platforms is a daunting task. It requires not only sophisticated design-flows that can satisfy both functional and non-functional requirements, such as performance and safety, but also requires considerable expertise in operating and exploiting available tools and APIs (Application Programming Interfaces). Furthermore, the development process must consider a myriad of design choices. For instance, developers must partition the application code into a set of tasks and offload them to the most suited system components (a process commonly known as hardware/software partitioning).

Subsequently, there is a need to deal with multiple compilation tools (sub-chains) that target each specific system component. These problems are exacerbated when dealing with FPGA (Field-Programmable Gate Array) components, a technology that combines the performance of custom hardware with the flexibility of software. As a consequence, users must explore code and mapping transformations specific to each architecture so that the resulting designs meet the overall solution requirements. The development process therefore leads to poorly maintainable code, where the source is transformed beyond recognition as developers typically manually apply an extensive set of architecture-specific transformations and tool-specific directives. As a result, implementing designs for such architectures is slow and error prone, with limited application portability. When the underlying architecture changes developers invariably need to restart the design process.

This demonstration focuses on the REFLECT design-flow [2][3][4] which is steered by LARA specifications [1],

a novel aspect-oriented programming (AOP) language for mapping high-level applications to heterogeneous high-performance embedded systems. The LARA language allows developers to capture non-functional requirements from applications in a structured way, leveraging high-level abstractions such as hardware/software design templates and flexible toolchain interfaces. By decoupling non-functional requirements code from the application code, developers can retain the benefits of preserving the original application source while exploiting the automation benefits of various domain-specific and target component-specific compilation/synthesis tools. In essence, LARA uses AOP mechanisms to offer under the same framework: (a) a vehicle for conveying application-specific requirements that cannot otherwise be specified in the original programming language for design capture, (b) using these requirements to guide the application of transformations and mapping choices, thus facilitating design-space-exploration (DSE), and (c) interfacing in an extensible fashion the compilation/synthesis components included in the toolchain.

Figure 1 illustrates the main idea beyond LARA and the approach demonstrated here. The source code on the left and aspects are input to the toolchain [1] which produces a representation of the computations after weaving.

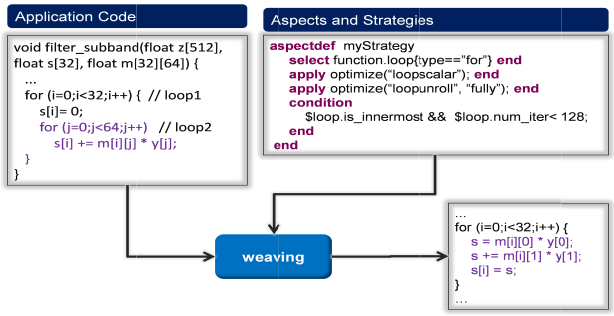

Figure 1. An example of the weaving in the context of LARA. Note that the output of the weaving process is not necessarily high-level code and can be an intermediate representation and/or low-level machine code.

Figure 2 shows the four main topics addressed by LARA and the design-flow [1][3]. The topics include code specialization which deals to the specification of data types and possible source annotations; code mapping which deals with the mapping of code constructs (related to computations and to data structures) to storage elements and to processing elements existent in the target architecture; runtime monitorization and instrumentation of variables, execution paths, function calls, computations, etc.; and code transformations, compiler and synthesis optimizations. This approach has been developed under the REFLECT project (EU FP7) with applications and optimization strategies provided by industry, in particular avionics and audio encoders. The LARA language AOP approach is described in [1].

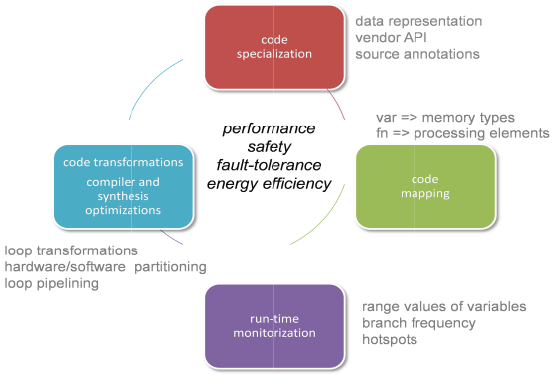

Figure 2. Main topics addressed by the LARA approach.

2. Topics focused by the Demo

The design-flow presented in this demo includes high-levels of flexibility provided by the LARA specification of aspects and the weaving processes in various stages of the flow [1]. This flexibility is illustrated in an abstract way in **Figure 3**. In particular, LARA aspects can codify user knowledge and expertise in order to apply them automatically to other applications (**Figure 3**(a)). In addition, LARA aspects can capture strategies and non-functional requirements that can drive the generation of customized implementations for a single application (**Figure 3**(b)). These strategies can also be parameterized to realize design patterns and templates (**Figure 3**(c)).

With this demonstration we show how to use LARA and the design-flow for:

- injecting code for instrumenting and monitorization of several C code artifacts. Specifically, we demonstrate how to monitor range values of program variables specified by the developer, how to monitor specific function calls and how to count branch-taken paths.
- specifying compiler and synthesis strategies in order to achieve efficient hardware/software FPGA based implementations according to the target architecture.

3. Demonstration Examples

In beginning of the demonstration, the presenter describes a list of concerns (e.g., extracting specific run-time information or performing compiler optimizations). For each concern, a LARA description is presented and explained.

Figure 4 depicts a snapshot of the weaving process performed by the source-to-source weaving stage (an extended version of the Harmonic tool [5] is used). In this example one can see how parts of the LARA aspect (on top of the figure), the original code with the identification of one of the join points (left bottom of the figure), and the actual join point (right bottom of the figure) are related to each other. This information is automatically provided and graphically showed using the Harmonic simulator. The LARA aspect used in this example is responsible to count the number of branch-taken paths.

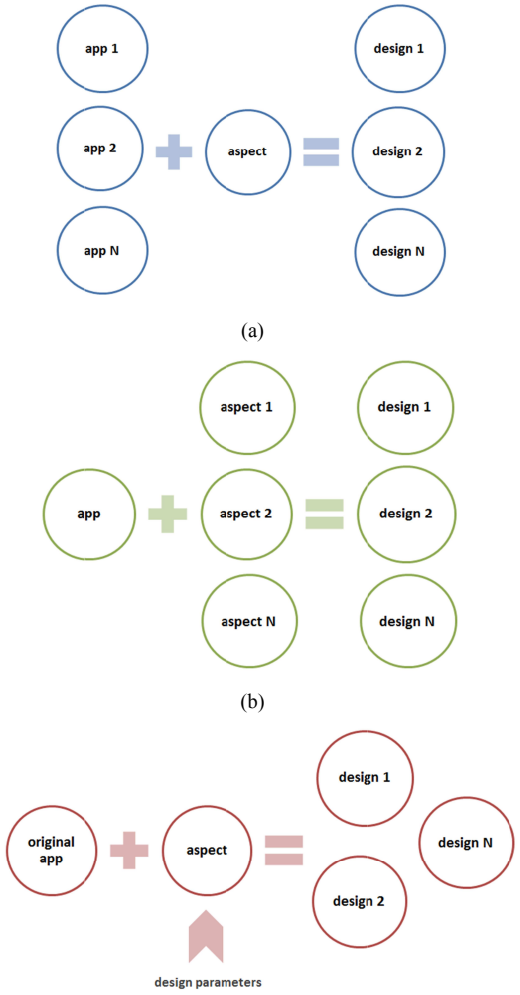

Figure 3. Design-flow flexibility powered by aspects: (a) reusable strategies; (b) retargetability; (c) design space exploration.

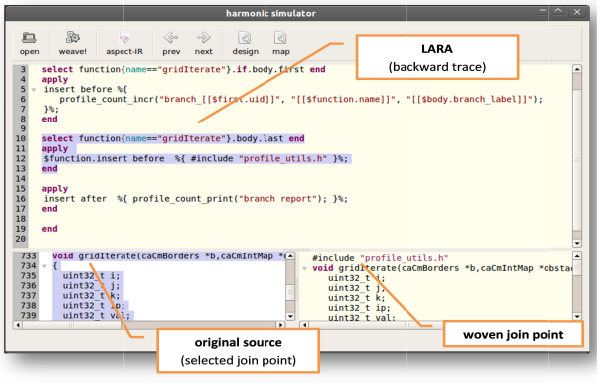

Figure 4. Snapshot showing the weaving process performed by the Harmonic tool.

The following LARA aspect (see **Figure 5**) extracts value ranges (minimum and maximum values) of selected variables, which can help generate resource-efficient FPGA designs using word-length optimization techniques.

Figure 6 shows the comparison between original and woven sources after executing the weaving process using the aspect presented in **Figure 5**. In particular, the tool highlights the differences between both sources.

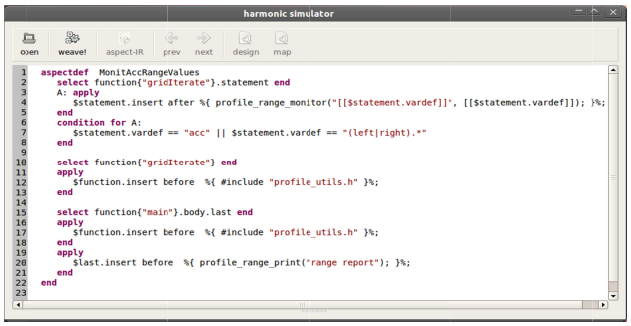

Figure 5. Example of a LARA aspect extracting range values for specific variables.

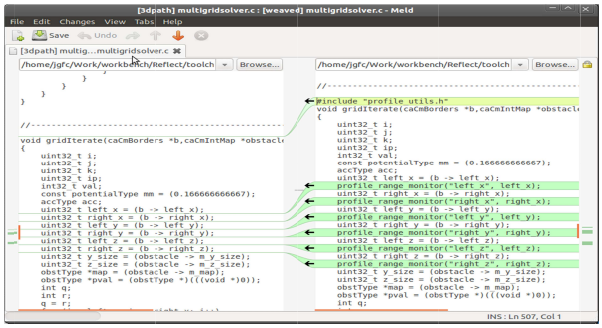

Figure 6. Input C code and the woven output code.

The next example presented in **Figure 7** and **Figure 8** shows the use of LARA to insert code primitives to measure the execution time of a given section of code considering different target architectures. In this figure we consider a host computer (PC) and an embedded system using a Xilinx MicroBlaze processor [6]. This example highlights one of the benefits of LARA: the original code which conveys the functionality of the design can be platform independent, and aspects can be introduced to generate target dependent designs.

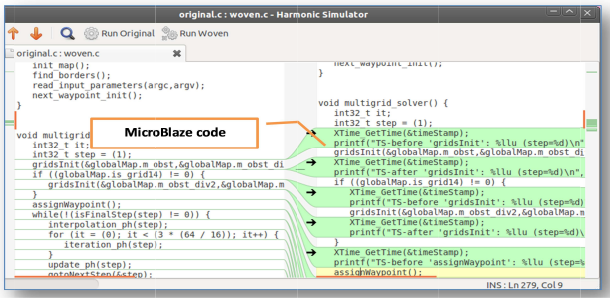

Figure 7. Code injected for measuring execution time considering a system based on a MicroBlaze processor.

We show in the next example how to use LARA to perform hardware/software partitioning and compiler optimizations. In particular, the LARA aspect (see **Figure 9**) instructs the design-flow to map all the application functions to the Virtex-5 (a Xilinx FPGA) [7] as long as its estimated cost is less than PPC (here identifying the IBM PowerPC 440 used in Virtex-5 [8]). In this case, the Virtex-5 partition source file must have its functions inlined, because function calls may not be supported by backend C-to-gates compilers.

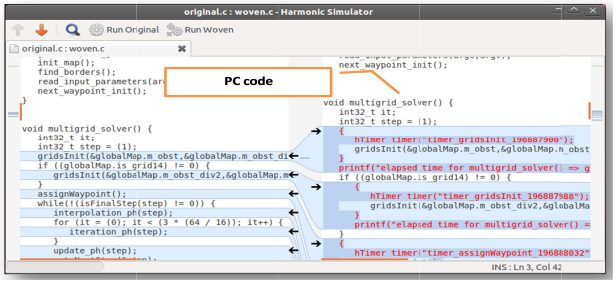

Figure 8. Code injected for measuring execution time considering a host computer (PC).

```
aspectdef GridIterateCoSyOpt2
  A: select function end
  B: apply to A
     $function.optimize("inline");
     $function.map(id:"virtex5");
  end
  condition for B:
    $function.estimated_virtex5 < $function.estimated_ppc
  end
end
```

Figure 9. LARA aspect specifying a hardware/software partitioning strategy.

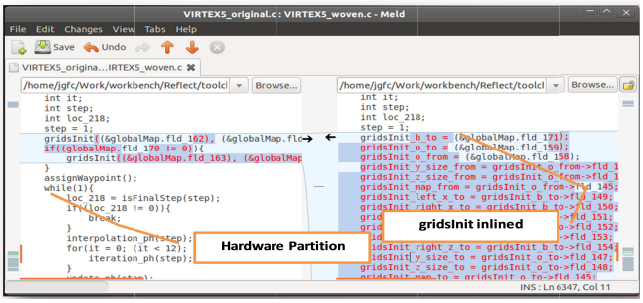

Figure 10. Snapshot showing code after hardware/software partitioning (left) and after function inlining (right).

4. Summary

This paper presented some of the many uses of LARA for injecting code and guiding with strategies, transformations, compiler and synthesis optimizations, and mapping of applications to hardware/software systems. These examples illustrate the current capabilities of the current implementation of the LARA-based toolchain, consisting of a source-to-source transformation tool, a compiler and optimizer, and multiple hardware synthesis tools.

5. Acknowledgments

This work was partially supported by the European Community's Framework Programme 7 (FP7) under contract No. 248976. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the European Community. The authors are grateful to all team members of the REFLECT project for their help and support. Tiago Carvalho and João Cardoso also acknowledge the support of FCT (Portuguese Science Foundation) through the project AMADEUS (POCTI, PTDC/EIA/70271/2006).

6. References

[1] João M. P. Cardoso, Tiago Carvalho, José Gabriel de F. Coutinho, Wayne Luk, Ricardo Nobre, Pedro C. Diniz, Zlatko Petrov, "LARA: An Aspect-Oriented Programming Language for Embedded Systems," in *Proc. of the Intl. Conf. on Aspect-Oriented Software Development (AOSD'12)*, Hasso-Plattner-Institut Potsdam, Germany, March 25-30, 2012.

[2] REFLECT, FP7 EU Project: http://www.reflect-project.eu.

[3] João M. P. Cardoso, Pedro C. Diniz, Zlatko Petrov, Koen Bertels, Michael Hübner, Hans van Someren, Fernando Gonçalves, José Gabriel de F. Coutinho, George Constantinides, Bryan Olivier, Wayne Luk, Juergen Becker, Georgi Kuzmanov, Florian Thoma, Lars Braun, Matthias Kühnle, Razvan Nane, Vlad-Mihai Sima, Kamil Krátký, José Carlos Alves, and João Canas Ferreira, *REFLECT: Rendering FPGAs to Multi-Core Embedded Computing*, book chapter in Reconfigurable Computing: From FPGAs to Hardware/Software Codesign, J. M. P. Cardoso and M. Huebner (eds.), Springer, Aug., 2011, pp. 261-289.

[4] João M. P. Cardoso, Razvan Nane, Pedro C. Diniz, Zlatko Petrov, Kamil Krátký, Koen Bertels, Michael Hübner, Fernando Gonçalves, José Gabriel de F. Coutinho, George Constantinides, Bryan Olivier, Wayne Luk, Juergen Becker, and Georgi Kuzmanov,, "A New Approach to Control and Guide the Mapping of Computations to FPGAs," in *Proc. of the Intl. Conf. Engineering of Reconfigurable Systems and Algorithms (ERSA'11)*, Las Vegas, NV, USA, July 18-21, 2011, CSREA Press. pp. 231-240.

[5] Wayne Luk, José Gabriel de Figueiredo Coutinho, Timothy John Todman, Yuet Ming Lam, William G. Osborne, Kong Woei Susanto, Qiang Liu, and W. S. Wong, "A High-Level Compilation Toolchain for Heterogeneous Systems," in *Proc. IEEE International SoC Conference (SoCC'09)*, Belfast, Northern Ireland, UK, Sept. 2009, pp. 9-18.

[6] Xilinx Inc., *MicroBlaze Processor Reference Guide*, Embedded Development Kit, EDK 13.4, UG081 (v13.4), 2012.

[7] Xilinx Inc., *Virtex-5 Family Overview*, Product Specification, DS100 (v5.0) February 6, 2009. http://www.xilinx.com

[8] Xilinx Inc., *Embedded Processor Block in Virtex-5 FPGAs*, Reference Guide, UG200 (v1.8) February 24, 2010. http://www.xilinx.com