BULETINUL INSTITUTULUI POLITEHNIC DIN IAŞI
Publicat de
Universitatea Tehnică „Gheorghe Asachi” din Iaşi
Volumul 69 (73), Numărul 2, 2023
Secţia
ELECTROTEHNICĂ. ENERGETICĂ. ELECTRONICĂ
DOI:10.2478/bipie-2023-0011

A METHODOLOGY FOR THE SYNTHESIS AND EVALUATION OF HARDWARE ACCELERATORS

BY

CRISTIAN-TIBERIUS AXINTE*

“Gheorghe Asachi” Technical University of Iaşi,
Computer Engineering Department

Received: June 17, 2024
Accepted for publication: July 25, 2024

Abstract. The methodology presented in this paper is based on a system of software automation tools that enable rapid design, implementation and evaluation of hardware accelerators. This system leverages a set of available industry-grade frameworks that can generate hardware implementations of software-described algorithms and computations. The hardware accelerators undergo an evaluation process that highlights the capabilities of the frameworks used to generate them. The methodology and its components could act as a building block for a larger system that performs design space exploration (DSE). The input to the system are mathematical computations described in two different formats: in Operator Language, and, as a Simulink Model, respectively. The system then is capable of generating a set of programs describing the solution implementation in either a lower-level language (e.g. C/C++) or a hardware description language.

Keywords: Design Space Exploration (DSE), High-Level Synthesis (HLS), Domain-Specific Accelerator, Power performance area (PPA) analysis, RISC-V

*Corresponding author; *e-mail*: cristian-tiberius.axinte@academic.tuiasi.ro

1. Introduction

With the slowdown of Moore's Law, researchers are turning their attention towards innovative hardware and software architectures to maintain the momentum in computational acceleration. Enhancing the clock rate of devices is not the sole method to boost computational performance, rather, there are architectures designed to address specific problems. These are increasingly being employed in many academic and industrial contexts. One such context is represented by the European Processor Initiative (EPI), an important endeavor within the European Union aimed at developing intellectual property on computing systems that can deliver high computational power with a minimum energy usage (European Union, 2024). One of the main motivations behind EPI is the reduction of Europe's reliance on non-European processor technologies. It is part of the European Union's broader strategy for digital sovereignty and is supported by significant funding from the EU, as part of the Horizon 2020 program. The technology focus is the adoption and development of computing systems having the central processing unit (CPU) core implementing the RISC-V instruction set architecture (ISA). Its open-source nature allows for greater flexibility and customization, which is crucial for addressing specific European requirements in various application domains like artificial intelligence (AI), automotive systems (especially for autonomous vehicles), edge computing and secure computing, which in turn demand robust, efficient, and scalable processing capabilities (RISC-V Consortium, 2024). Achieving these metrics requires heterogeneous integration of traditional CPU cores with other types of processing units and Single Instruction Multiple Data (SIMD) architectures, like graphics processing units (GPUs), general purpose Vector Processing Units (VPUs) (see for example (Perotti *et al*., 2022; Perotti *et al.*, 2023; Minervini *et al.*, 2023)) as well as application specific hardware, known in the literature as hardware accelerators. An accelerator is a specialized computing unit designed to perform certain types of tasks more efficiently than a traditional, general-purpose CPU. Accelerators are by-design optimized for specific types of operations. To effectively use them, software needs to be written (or adapted) for them. Future computing systems heavily rely on hardware accelerators and specialized computing units for data processing. These accelerators have become the go-to method to enhance performance, yet their parallelism and diversity introduce significant complexity for software engineers (Reuther *et al.*, 2022). See Fig. 1 for the motivation of the present methodology and the connection thereof with hardware accelerators.

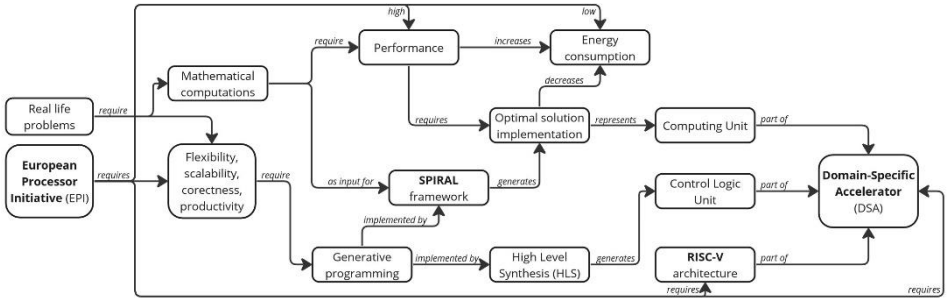

Fig. 1 – The motivation for the proposed methodology and the connection with hardware accelerators.

2. State of the art

The methodology presented in this paper is to be viewed as a building block for a larger system that performs design space exploration (DSE). DSE is a systematic and iterative process used to find, analyze and optimize the possible design choices and parameters of a system or product. The primary goal of DSE is to find the best design solution that meets certain objectives, constraints, and requirements, while considering a wide range of design variables. In the national state of the art, there is a lack of research papers contributing to the architecture and the implementation of a DSE methodology. In the international state of the art, the degree of novelty and relevance of DSE is highlighted in (Mrazek *et al.*, 2019) and (Pimentel, 2017). This subject is particularly relevant in the context of electronic and embedded system design due to the complexity of these systems and the numerous design choices available. Also relevant is in the context of maximizing energy efficiency, while keeping the performance required for achieving the functionality of the embedded system. DSE helps navigate these choices by performing trade-off analysis among different implementation options. The exploration process can be complex and time-consuming, prompting researchers to develop fully- or semi- automated tools to facilitate faster and more efficient decision-making processes. There are various software tools available from different entities, tools that can be used to implement parts of a DSE system. Some of these tools fall under the generative programming paradigm. Generative programming is a software engineering approach focused on creating programs that can produce other programs automatically (Franchetti *et al.*, 2018a).

3. Methodology description

The overarching objective of this methodology is to be integrated as part of a DSE system used for finding the optimum computational algorithm implementation that satisfies the performance and energy efficiency requirements of future near-sensor processor units. These metrics emerge from a power, performance, area (PPA) analysis. The next subsections describe the ideas depicted in Fig. 2 and the possible venues to implement them.

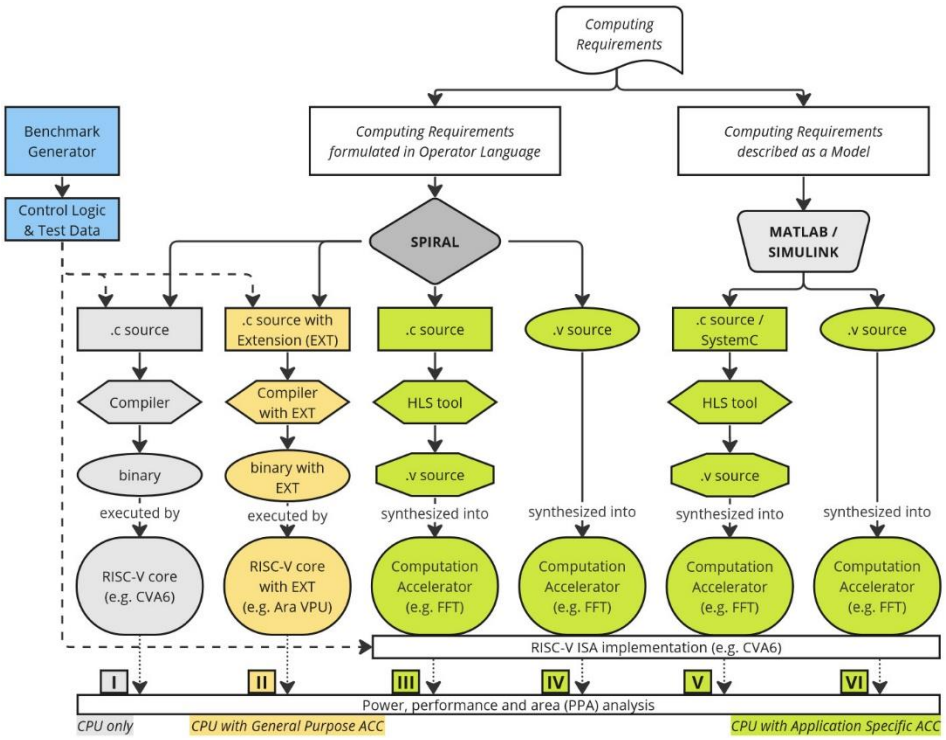

Fig. 2 – The proposed methodology using the SPIRAL framework and the Simulink environment. PPA analysis comparison is made between: CPU only, CPU with general purpose accelerator, and CPU with application specific accelerator respectively.

3.1. The domain-specific language approach

The generative programs are usually written in a high-level language, such as mathematical or signal processing domain-specific languages; one instance thereof can be found in (Franchetti *et al*., 2009), named as the Operator Language (OL), based on the Signal Processing Language (SPL). These domain-specific languages are the basis input language for the generative

framework called SPIRAL. The framework has been developed by the Department of Electrical and Computer Engineering of Carnegie Mellon University, USA. It has been refined over the course of more than 25 years, benefiting from a variety of research grants, proving its maturity and performance. It began as a research project and is extensively presented in the invited research paper (Franchetti *et al*., 2018b). This paper also contains the theoretical developments and numerical simulations using SPIRAL. Recently, it has been used to generate hardware accelerator designs (Xu *et al.*, 2022). The rest of this section presents some of the knowledge and remarks extracted during the review and evaluation of the open-source part of SPIRAL.

The SPIRAL framework represents a collection of functions and parameters that are in part accessible through the command line interface script provided by the authors. The interface script and most of the functions are written in GAP system version 3 (Schönert *et al*., 1997). The framework relies on several GAP packages (*package* is the GAP terminology for *library*). In the early days, SPIRAL was proposed to be part of the GAP system, in the form of a package, but without success in the long run.

An example of how the DFT8 operator is fully expanded into an SPL/OL formula (Franchetti *et al*., 2018b) is shown below:

$$\mathrm{DFT}_8 \rightarrow (\mathrm{F}_2 \otimes \mathrm{I}_4)\mathrm{T}_4^8\big(\mathrm{I}_2 \otimes (\mathrm{F}_2 \otimes \mathrm{I}_2)\mathrm{T}_2^4(\mathrm{I}_2 \otimes \mathrm{F}_2)\mathrm{L}_2^4\big)\mathrm{L}_2^8,$$

where $\otimes$ represents the Kronecker product of matrices A and B, defined as:

$$A \otimes B = [a_{k,\ell}B], \quad \text{for } A = [a_{k,\ell}]$$

Branches I, II and III of Fig. 2 are represented by the software-generation capabilities of the SPIRAL framework. A DFT example can be found on the SPIRAL website, section *Generator*. Additional work is needed to make the generated C code executable by a RISC-V core (branch I of Fig. 2). The same applies for the case of branch II of Fig. 2, where a RISC-V ISA-compliant VPU is integrated along the core, similar to x86's AVX (Advanced Vector Extensions) and Neon – ARM's SIMD architecture extension.

Branch IV of Fig. 2 is represented by the framework's capability to generate customized hardware implementations of several computations. A sequence of GAP code (SPIRAL-software public repository) is presented below, where the function `streamGen` is defined. This function is used to create the optimized hardware intermediary representation (hardware IR) of the computation described by the `SPL` argument, considering the options given in `opts` argument. This makes the SPIRAL framework able to produce custom software and hardware implementations of a given computation, considering also the data type (integer, real or complex) and precision (half, single or double precision floating-point; fixed-point).

```
streamGen := function(SPL, opts)
  local rt, srt, srt2, srt3, srt4;
  rt := RandomRuleTree(SPL, opts);
  srt:= SumsRuleTreeStrategy(rt, SumStreamStrategy, opts);
  srt2:= ApplyStrategy(srt, StreamStrategy, UntilDone,
opts);
  srt3:= Process_fPrecompute(srt2, opts);
  srt4:= srt3.createCode();
  return srt4;
end;
```

An example function call is given below, where the output is the hardware IR of the Discrete Cosinus Transform (DCT).

```
streamDCTUnroll := function(size, width)
  return streamGen(TDCT2(size).withTags([AStream(width)]),
InitStreamUnrollHw());
end;
```

This hardware IR is then processed by the script that produces the Verilog description of the SPL formula. This script is not made publicly available by the authors of SPIRAL. The generated hardware will compute the DCT in an unrolled (i.e. non-iterative) fashion, while streaming data paradigm is employed for the input and output of the hardware block. This kind of computation promotes throughput, while increasing silicon area.

3.2. The model-based approach

MATLAB and Simulink are well known and widely used frameworks in industry and academia. Simulink has code generating capabilities, outputting programs written in C/C++ or C++ with SystemC library (branch V of Fig. 2); or the two mainstream hardware description languages, Verilog and VHDL, using HDL Coder (branch VI of Fig. 2). Simulink implements the model-based design paradigm and is able to generate optimized implementations of various computations, from trigonometric functions to filter banks or FFTs, considering also the data type (integer, real or complex) and precision (half, single or double precision floating-point; fixed-point).

Another option for the model-based approach is Vitis Model Composer, a development environment provided by Xilinx (now AMD). It's aimed at facilitating the design, simulation and implementation of complex signal processing and machine learning algorithms on Xilinx FPGAs and SoCs. It enables automatic generation of RTL code from Simulink models.

Both approaches could also be used to produce the benchmark generator block presented in Fig. 2, in the form of a testbench.

3.3. The High-Level Synthesis approach

High-Level Synthesis (HLS) represents a paradigm shift in the design and implementation of digital hardware. It allows for a functional description of the circuit to be translated into a control-data-flow-graph (CDFG) that is further processed by tools performing various configurable optimizations. HLS tools provide means to boost productivity and to ensure implementation correctness, while considering also the data type (integer, real or complex) and precision (half, single or double precision floating-point; fixed-point). This high-level approach is being used to generate Register Transfer Level (RTL) descriptions for hardware, from C/C++ code; it is also used to accelerate portions of an algorithm described in C/C++ code, by programming the Field-Programmable-Gate-Array (FPGA) hardware to run those portions of the algorithm. However, benefiting from the HLS approach comes at a price represented by the learning curve of its philosophy. The HLS user must accommodate with using a procedural, object-oriented programming language (e.g., C/C++) to describe hardware. C/C++ has been mainly used to write software for the CPU to execute. Employing the paradigm of C/C++ to describe not the computation for the CPU, but the CPU itself can seem counterintuitive. This change in mindset should be accompanied by the HLS design principles and the abstract parallel programming model for HLS. Examples thereof include the three paradigms named producer-consumer, streaming data, and pipelining (Advanced Micro Devices Inc., 2023). Adhering to these principles and rules promotes design correctness, increases performance, while facilitating design scalability and reusability. One venue for leveraging HLS is the Vitis Unified Software platform provided by former Xilinx, now AMD. Other development environments are listed in the survey (Del Sozzo *et al*., 2023). A taxonomy of hardware abstractions together with their main characteristics and timelines, along with the future trends on the subject are also discussed.

Automatic testbench generation is part of the HLS design flow and thus could also be used to generate the control logic and the test vectors needed for functional verification.

3.4. Hardware simulation

Hardware simulation can be performed in several ways, some of which are described next. The traditional way is the event-driven simulation, where the behavior of the digital circuit is monitored, by tracking the changes (or events) in the system as they propagate through the different logic layers. This approach gives a high degree of accuracy, being a suitable method used also in the most complex designs. It is, however, time-consuming, as the duration of a

simulation run for some complex designs could reach days on a personal desktop computer.

Another simulation approach is enabled by **Verilator**, an open-source tool for digital circuits described in the Verilog hardware description language (HDL). It converts Verilog code into C++ or SystemC code. It is then compiled and executed, providing faster simulations than the traditional approach of event-driven simulation, for example, using the Vivado design suite. This makes Verilator a suitable choice when it comes to large design, including CPU cores together with hardware accelerators.

Software-based full-system simulation is used to emulate an entire computing system together with the software written for the emulated ISA. One available tool is QEMU (Quick Emulator), which is free and open source. It has already been employed to virtualize many ISAs on x86 CPUs. It can also emulate peripheral devices such as network interfaces, memory controllers and graphical interfaces. QEMU provides means to save the state of the virtual machine at any point in time. It is also compatible with Kernel-based Virtual Machine (KVM). A DSE environment could greatly benefit from the versatility of QEMU.

Design validation and benchmarking is also possible by leveraging the FPGA technology. Although less transparent than an event-driven simulation, FPGA prototyping measures the real performance of the design and can highlight the performance of the software controlling the packaged hardware IP.

4. Conclusions

The methodology presented in this paper is based on a system of software automation tools that should enable rapid design, validation and evaluation of hardware accelerators. The methodology and its components could act as a building block for a larger system that performs DSE. Future work includes integrating this methodology inside a DSE framework, (Zhang *et al.*, 2018) being a starting point and an example approach for a DSE framework. Another further development is represented by enabling the automatic flow of branches II and III of Fig. 2, by creating custom compiler workflows.

Acknowledgements. The author would like to acknowledge the support received from Andrei STAN and George ULERU, members of the same department.

REFERENCES

Advanced Micro Devices Inc. (2023, December 18). Vitis High-Level Synthesis User Guide (UG1399) v2023.2.

European Union, *European Processor Initiative*, https://www.european-processor-initiative.eu.

Franchetti F., de Mesmay F., McFarlin D., Püschel M., *Operator Language: A Program Generation Framework for Fast Kernels*, W. M. Taha (Ed.), Domain-Specific Languages (pp. 385–409), Springer Berlin Heidelberg, 2009.

Franchetti F., Moura J., Padua D., Dongarra J., *From High-Level Specification to High-Performance Code*, Proceedings of the IEEE, vol. 106, no. 11, pp. 1875-1878, Nov. 2018, https://doi.org/10.1109/JPROC.2018.2875253.

Franchetti F., Low T., Popovici D., Veras R., Spampinato D., Johnson J., Püschel M., Hoe J., Moura J., *SPIRAL: Extreme Performance Portability*, Proceedings of the IEEE, vol. 106, no. 11, pp. 1935-1968, Nov. 2018, https://doi.org/10.1109/JPROC.2018.2873289.

Minervini F., Palomar O., Unsal O., Reggiani E., Quiroga J., Marimon J., Rojas C., Figueras R., Ruiz A., Gonzalez A., Mendoza J., Vargas I., Hernandez C., Cabre J., Khoirunisya L., Bouhali M., Pavon J., Moll F., Olivieri M., Kovac M., Kovac M., Dragic L., Valero M., Cristal A., *Vitruvius+: An Area-Efficient RISC-V Decoupled Vector Coprocessor for High Performance Computing Applications*, ACM Trans. Archit. Code Optim. 20, 2, Article 28 (June 2023), 25 pages, https://doi.org/10.1145/3575861.

Mrazek V., Hanif M., Vasicek Z., Sekanina L., Shafique M., *AutoAx: An Automatic Design Space Exploration and Circuit Building Methodology utilizing Libraries of Approximate Components,* Proceedings of the 56th Annual Design Automation Conference 2019 (DAC '19), Association for Computing Machinery, New York, NY, USA, Article 123, 1–6, 2019, https://doi.org/10.1145/3316781.3317781.

Perotti M., Cavalcante M., Wistoff N., Andri R., Cavigelli L., Benini L., *A "New Ara" for Vector Computing: An Open Source Highly Efficient RISC-V V 1.0 Vector Processor Design*, in 2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP), 2022, https://doi.org/10.48550/arXiv.2210.08882.

Perotti M., Cavalcante M., Andri R., Cavigelli L., Benini L., *Ara2: Exploring Single- and Multi-Core Vector Processing with an Efficient RVV1.0 Compliant Open-Source Processor*, arXiv, Nov. 2023, https://doi.org/10.48550/arXiv.2311.07493.

Pimentel A., *Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration*, in IEEE Design & Test, vol. 34, no. 1, pp. 77-90, Feb. 2017, https://doi.org/10.1109/MDAT.2016.2626445.

Reuther A., Michaleas P., Jones M., Gadepally V., Samsi S., Kepner J., *AI and ML Accelerator Survey and Trends*, 2022 IEEE High Performance Extreme Computing Conference (HPEC), 1–10. Waltham, MA, USA: IEEE, 2022, https://doi.org/10.1109/HPEC55821.2022.9926331.

RISC-V Consortium, *RISC-V Open Standard ISA*, https://riscv.org/.

Schönert M. *et al*., GAP – Groups, Algorithms, and Programming – version 3 release 4 patchlevel 4, (1997). https://www.gap-system.org.

Sozzo E. Del, Conficconi D., Zeni A., Salaris M., Sciuto D., Santambrogio M.D., *Pushing the Level of Abstraction of Digital System Design: A Survey on How to Program FPGAs*, ACM Comput. Surv., 55(5), 2022, https://doi.org/10.1145/3532989.

SPIRAL-software public repository, https://github.com/spiral-software/spiral-software

SPIRAL web generator, https://spiral.net.

Xu G., Hoe J.C., Franchetti F., *Flexible Hardware Accelerator Design Generation with Spiral*, 2022 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2022, pp. 1-7, https://doi.org/10.1109/HPEC55821.2022.9926413.

Zhang J., Tabkhi H., Schirner G., *DS-DSE: Domain-specific design space exploration for streaming applications*, 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018, 165–170, https://doi.org/10.23919/DATE.2018.8341997.

METODOLOGIE PENTRU SINTETIZAREA ȘI EVALUAREA ACCELERATOARELOR HARDWARE

(Rezumat)

Metodologia prezentată în această lucrare se constituie pe un sistem de utilitare software. Acestea facilitează proiectarea, implementarea și evaluarea acceleratoarelor hardware. Acest sistem incorporează framework-uri software dezvoltate de mediul industrial care pot genera implementări hardware ale algoritmilor și calculelor descrise în software. Aceste framework-uri sunt evaluate prin prisma performanțelor implementărilor generate. Această metodologie poate fi privită ca parte componentă a unui sistem mai mare care realizează explorarea spațiului de proiectare. Intrarea sistemului descris în lucrare este un calcul matematic descris în două moduri diferite: în Limbajul Operatorilor, și respectiv, ca un model Simulink. Sistemul generează un set de programe ce descriu implementările calculelor matematice, în limbaje de nivel scăzut de abstractizare, precum C/C++, sau într-un limbaj de descriere hardware.