Multi-objective Optimisation of RISC-V CV32A6 for ML application

Bastien HUBERT, Omar HAMMAMI
Computer Science and Systems Engineering Laboratory (U2IS)
ENSTA Paris
Palaiseau, France
{bastien.hubert, omar.hammami}@ensta-paris.fr

***Abstract*—RISC-V architectures are rapidly gaining in popularity in embedded systems, in which each mW counts. Since AI-related applications such as image recognition or neural networks tend to be highly energy consuming, low-power techniques are required to optimise the autonomy of systems using SoCs to run such applications.**

However, the trade-off between energy consumption, application performance and resource use requires a multi-objective optimisation with a potentially very important number of optimisation parameters to be performed on the SoC. As they rely on heuristics that are likely to only return locally optimal solutions, empirical methods must be excluded.

To address this issue, a technology-agnostic mathematical model is introduced to represent how optimisations are applied to a SoC, and a workflow designed to perform an intelligent exhaustive exploration of the optimisation space has been developed to highlight a subset of optimal processor configurations.

Optimisation of the ARIANE/CV32A6 RISC-V processor, running a CNN propagation on a Xilinx Zynq 7020 FPGA, has shown very encouraging results using low degree configurations, and is likely to perform even better with higher degree configurations.

***Index Terms*—RISC-V, low-power design, embedded systems, multi-objective optimisation, mathematical model, floorplanning, FPGA, simulation**

I. Introduction

The RISC-V architecture is becoming increasingly popular and exploited in many application areas. Predictions about the use of RISC-V architectures in AI indicate that they will be intensively exploited in the near future. The success of the recent RISC-V 2022 Spring Week [1] also reasserted the commitment of the scientific community in its modularity, making it a highly popular choice in embedded system applications.

The goal of this paper is to optimise the energy consumption, application performance and resource use of the ARIANE/CV32A6 RISC-V processor, written in SystemVerilog, on FPGA.

To achieve this, we will propose a mathematical model describing a processor configuration based on the ordered list of optimisation parameters (e.g. whether or not to activate a unit, techniques for reducing power dissipation or tool parameters) used. Combinatorics, set theory and topological considerations will be taken into account to address the high-dimensional problems raised by our model.

A technology independent workflow, able to perform an intelligent exploration of the configuration space from the multi-objective energy/performance/resource trade-off point of view, will be presented and run for a CNN application on CVA6, using Xilinx Vivado as main synthesis tool. Energy simulations will be realised using the Questa Sim simulator, and testbench executions will be run on the Zybo Z7 development board, hosting the ZYNQ 7020 FPGA. A Pareto frontier extraction will eventually be performed to highlight optimal solutions, and statistics regarding the configuration energy consumption, performance and resource use distribution of the configurations will be briefly discussed.

II. Mathematical Foundations

Let $N \in \mathbb{N}$ and $c \in \mathbb{N}^N$. c is called a **configuration** if:

$$\exists\, k \in \mathbb{N} \,|\, \begin{cases} \forall i \in [\![1,\, k]\!], & \exists!\, j \in [\![1,\, N]\!] : \quad c_j = i \\ \forall i > k, & \nexists\, j \in [\![1,\, N]\!] : \quad c_j = i \end{cases} \tag{1}$$

If k exists, it is unique and is called the **degree** of the configuration: $deg(c) := k$.

Any given configuration of degree k is composed of:

- exactly one of each integer between 1 and k. The indices of those integers match with the optimisation parameters used in the configuration, while their values correspond to the order in which they are used.
- $N - k$ zeros (as c cannot have an integer greater than $N - k$). These indices correspond to the optimisation parameters that *aren't* used in the configuration.

For example, if $N = 3$, valid configurations include $(0,\, 0,\, 0)$, $(0,\, 1,\, 0)$, $(2,\, 0,\, 1)$ and $(2,\, 3,\, 1)$. On the other hand, $(1,\, 0,\, 3)$, $(2,\, 2,\, 1)$, and $(0,\, 0,\, -1)$ are not configurations. In particular, $0_{\mathcal{C}} := (0,\, 0,\, ...\, 0) = 0_{\mathbb{N}^N}$ is a configuration, and is called **standard configuration**.

We can thus define $\mathcal{C}$ the set of all configurations, and a family of subsets corresponding to the configurations of a certain degree:

$$\forall k \in \mathbb{N},\, \mathcal{C}_k := \{c \in \mathcal{C} \mid deg(c) = k\} \tag{2}$$

The elements of $\mathcal{C}_k$ are composed of one of each integer between 1 and k, in any order, and $N - k$ zeros. Thus:

$$\#\mathcal{C}_k = k! \times \binom{N}{N-k} = \frac{N!}{(N-k)!} \tag{3}$$

Furthermore, the Taylor-Lagrange formula applied to $f : x \in \mathbb{R} \mapsto e^x \in \mathbb{R}$ and evaluated in $x := 1$ and $a := 0$ states that:

$$e = \sum_{i=0}^{N} \frac{1}{i!} + \int_0^1 \frac{(1-t)^N}{N!} e^t \, \mathrm{d}t \tag{4}$$

Given that $\mathcal{C} = \bigsqcup_{k=0}^{N} \mathcal{C}_k$, the formula can be used with $i = N - k$ to give a simple equivalent of $\#\mathcal{C}$:

$$\#\mathcal{C} = N!\, e - \int_0^1 (1-t)^N e^t \, \mathrm{d}t \underset{N \to \infty}{\sim} N!\, e \tag{5}$$

The fact that the exploration space grows as a factorial (the approximation in Eq. 5 is precise to 10^{-8} as soon as $N \geq 10$) leads to a **combinatorial explosion** even for relatively small values of N. For instance, $\#\mathcal{C} \approx 10^7$ for $N = 10$.

Given that $\mathcal{C} \subset \mathbb{N}^N \subset \mathbb{R}^N$, can define the **canonical distance** d, or genotypical distance, making $(\mathcal{C},\, d)$ a metric space:

$$d : \begin{cases} \mathcal{C} \times \mathcal{C} & \to & \mathbb{R}_+ \\ (c,\, c') & \mapsto & \sqrt{\sum_{k=1}^{N} (c_k - c'_k)^2} \end{cases} \tag{6}$$

However, as N increases, d loses its "discrimination ability", meaning that it is increasingly difficult to distinguish pairs of elements using only d. This means that navigating through $\mathcal{C}$, let alone extracting a Pareto surface to isolate Pareto-equivalent configurations in the energy/performance/resource trade-off, is mathematically difficult.

This phenomenon is referred to as the **curse of dimensionality**, and is a common issue when dealing with high-dimensional metric spaces. One way to illustrate this curse of dimensionality is to consider M outcomes $(v_1,\, v_2,\, ...v_M)$ of a continuous uniform distribution on $[0,\, 1]^N$, and plot the discrimination ratio R as a function of N, as shown by [2] in Fig. 1.

$$R := \frac{max_{(i,\, j) \in [\![1,\, M]\!]^2,\, i \neq j} \{d(v_i,\, v_j)\}}{min_{(i,\, j) \in [\![1,\, M]\!]^2,\, i \neq j} \{d(v_i,\, v_j)\}} \xrightarrow[N \to \infty]{} 1 \tag{7}$$

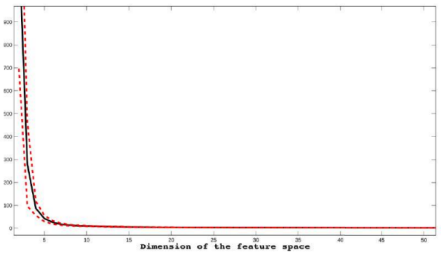

Fig. 1. R as a function of N (M = 500)

To solve this issue, we need to work in a smaller topological space, called **effective space**. This space, noted $\mathcal{EPR} := {\mathbb{R}_+}^3$, corresponds to the 3D plot of the standardised energy consumption, performance and resource use of each configuration. In order to represent a configuration in this space, we need to define a **fitness function** f:

$$f : \begin{cases} \mathcal{C} & \to & \mathcal{EPR} \\ c & \mapsto & (\, E^0(c),\, P^0(c),\, R^0(c)\,) \end{cases} \tag{8}$$

E^0, P^0 and R^0 are the **standardised energy, performance and resource functions** respectively. By noting E, P and R the functions used to evaluate the corresponding metrics, the standardised functions can be defined as such:

$$E^0,\, P^0,\, R^0 : \begin{cases} \mathcal{C} & \to & \mathbb{R}_+ \\ c & \mapsto & \frac{E(0_\mathcal{C})}{E(c)},\, \frac{P(c)}{P(0_\mathcal{C})},\, \frac{R(0_\mathcal{C})}{R(c)} \end{cases} \tag{9}$$

Since we want to maximise performance while minimising energy consumption and resource use, the standardised functions are designed so as to be maximising ratios. This is meant to make plots in the $\mathcal{EPR}$ space and the Pareto surface easier to read while having no impact on the mathematical meaning.

Using f, a new function called **effective distance**, or phenotypical pseudometric, can be defined as follows:

$$d_{eff} : \begin{cases} \mathcal{EPR} \times \mathcal{EPR} & \to & \mathbb{R}_+ \\ (x,\, y) & \mapsto & \sqrt{\sum_{k=1}^{3} (x_k - y_k)^2} \end{cases} \tag{10}$$

Given that f is not injective, these functions only grant pseudometric properties to $(\mathcal{C},\, (f,\, f) \circ d_{eff})$, as two different configurations may have the same standardised energy consumption, performance and resource use. Such points are indistinguishable in the effective space. Because f is used before applying d_{eff}, only three coordinates are used when calculating the distance, effectively eliminating the curse of dimensionality.

However, it is not the only problem raised by the combinatorial explosion of Eq. 5. Since N can be virtually as big as we want, it will be computationally extremely long to explore the entire space. To give an order of magnitude, it takes 3h to evaluate the fitness of a single configuration using the experimental setup of section IV-D, so a complete evaluation of $\mathcal{C}$ roughly takes to 3500 years of computation time for just 10 optimisation parameters. To avoid exhaustive exploration, techniques to reduce the exploration space have been used.

The simplest way to reduce the cardinal of a set without it losing its topological properties is to define a **binary equivalence relation** between the elements of the set, and to consider the space quotiented by the equivalence relation.

In the case of $\mathcal{C}$, a family of binary equivalence relations indexed by $\mathcal{P}([\![1,\, N]\!])$ can be defined as follows:

$$\begin{gathered} \forall L \in \mathcal{P}([\![1,\, N]\!]),\, \forall\, (c,\, c') \in \mathcal{C}^2, \\ c\, \mathcal{R}_L\, c' \overset{def}{\iff} \forall\, i \in [\![1,\, N]\!] \backslash L,\, c_i = c'_i \end{gathered} \tag{11}$$

Let $L \in \mathcal{P}([\![1,\, N]\!])$, the relation defined in Eq. 11 is called ***L*-distinction**, and defines a level of difference between two configurations. Given a configuration c:

- any parameter whose index is *not* in L used in c will also be used by every L-distinct configuration
- any unused parameter whose index is *not* in L will won't be used by any L-distinct configuration

The equivalence class of c is noted $[c]_L$. In particular, $[c]_\emptyset = \{c\}$ and $[c]_{[\![1,\ N]\!]} = \mathcal{C}$.

Unfortunately, the elements of a L-distinct class do not share enough properties to consider only evaluating a representative of each equivalence class instead of the entire configuration space. For instance, L-distinct configurations can have a different degree: $(1,\ 0,\ 0)\ \mathcal{R}_{\{2\}} (1,\ 2,\ 0)$, but $deg((1,\ 0,\ 0)) = 1$ and $deg((1,\ 2,\ 0)) = 2$. A new, degree-invariant, subset of $[c]_L$ can be defined as such:

$$[c]^0_L := \{c' \in [c]_L \mid \forall\, i \in L,\ c'_i \neq 0\} \tag{12}$$

An element of $[c]^0_L$ is called a ***L*-variation** of c. It should be noted that c is not always a L-variation of itself, as it is the case for $(1,\ 0,\ 0)$ and $L = \{2\}$. However, $[c]^0_L$ cannot be an empty set because it is always possible to replace an arbitrary number of zeros in c by $deg(c) + 1$, $deg(c) + 2$ and so on.

Let's consider $c' \in [c]^0_L$. Because the values corresponding to the indices of L cannot be zeros, potential zeros of c' have to be indexed by elements outside of L. Since $c\, \mathcal{R}_L\, c'$, c has exactly the same values for those indices, which means they have the same number of zeros, and thus the same degree.

This also means that, given that the zeros of every L-variation of c are at the same positions, a permutation of $[\![1,\ N]\!]$ can be applied to the indices of every element of $[c]^0_L$ to move the zeros to the last $N - deg(c)$ indices. The tuples extracted by only considering the first $deg(c)$ indices of each L-variations are thus permutations of $[\![1,\ deg(c)]\!]$. Since these configurations are L-distinct of c, another permutation can be applied to the tuples in order to move the values corresponding to the indices of L to the last indices.

Hence, L-variations can only differ from each other by a permutation of the values indexed by L. In particular, L-variations of $0_\mathcal{C}$ correspond to every configurations using exactly the parameters whose indexes are listed in L, and their degree is $\#L$. The two following relations can be deduced from these considerations:

$$\forall k \in [\![1,\ N]\!],\ \mathcal{C}_k = \bigsqcup_{L \in \{l \in \mathcal{P}([\![1,\ N]\!]) \mid \#l = k\}} [0_\mathcal{C}]^0_L \tag{13}$$

$$\mathcal{C} = \bigsqcup_{L \in \mathcal{P}([\![1,\ N]\!])} [0_\mathcal{C}]^0_L \tag{14}$$

It is worth noting that using a representative of $[0_\mathcal{C}]^0_L$ for each $L \in \mathcal{P}([\![1,\ N]\!])$, as suggested by Eq. 14, is equivalent to considering that parameter order has no impact on the fitness of a configuration, effectively reducing the number of configurations to evaluate from $N!\, e$ to 2^N.

Even in the general case where the order in which parameters are implemented has an impact on the fitness of a configuration, Eq. 13 can be used to create **meta parameters**. A meta parameter corresponds to the ordered tuple of used parameters in a representative of $[c]^0_L$ for a given L. Two simple methods can be used to determine which meta parameters must be taken into account when reducing the cardinal of the exploration space: the **ε-clustering** and the **n-clustering**.

These two techniques consist in evaluating, for a given value of $k \in [\![1,\ N]\!]$, the maximum effective distance between two elements of $[0_\mathcal{C}]^0_L$, called **configuration diameter**, for every $L \in \{l \in \mathcal{P}([\![1,\ N]\!]) \mid \#l = k\}$:

$$D(L) := \max_{(c,\ c') \in \left([0_\mathcal{C}]^0_L\right)^2} \{d_{eff}(f(c), f(c'))\} \tag{15}$$

In the case of ε-clustering, if there is L_0 that verifies $D(L_0) < \varepsilon$, the parameters corresponding to L_0 will be removed from the list of all parameters, and replaced by the corresponding meta parameter. For example, if the studied parameters were $(a,\ b,\ c)$ and $L_0 = \{1,\ 3\}$, a meta parameter can be $(c,\ a)$, and the new list of parameters will be $(b,\ (c,\ a))$. This leads to a diminution of N and the total number of configurations: a new configuration space $\widetilde{\mathcal{C}}$ is created, and the ε-clustering can be performed again while it is possible to find a L_0.

The n-clustering iterates on the exploration space exactly as the ε-clustering, but instead of stopping when every cluster has a diameter larger than ε does, it stops after n iterations. For the sake of clarity, $\widetilde{\mathcal{C}}$ will denote the exploration space at the end of the final clustering iteration.

Now that the number of parameters can be reduced at will, exhaustive exploration of $\widetilde{\mathcal{C}}$ can be realised, and evaluated configurations can be represented in the $\mathcal{EPR}$ space. But the question of selecting the best configuration for a given use case remains.

Given that there are no obvious linear relations between energy consumption, performance and resource use, we cannot try to optimise a single function of the form:

$$J : \left\{ \begin{array}{ccc} \mathcal{C} & \rightarrow & \mathbb{R}_+ \\ c & \mapsto & \alpha E^0(c) + \beta P^0(c) + \gamma R^0(c) \end{array} \right. \tag{16}$$

This means that mutlicriteria optimisation tools must be used in order to extract the best configuration(s) from the $\mathcal{EPR}$ space. In this paper, we will use the Pareto frontier extraction method to highlight configurations c such that there is no other configurations that has a better value for *every* metric. Mathematically, a partial order relation, called **domination**, can be defined between a 3D point x and another y as follows:

$$x \succeq y \overset{def}{\iff} \forall k \in [\![1,\ 3]\!],\ x_k \geq y_k \tag{17}$$

With this definition, a **Pareto-efficient** configuration is a configuration whose image by f is not dominated by that of any other configuration:

$$\nexists\, c' \in \mathcal{C} \backslash \{c\} \mid f(c') \succeq f(c) \tag{18}$$

The set of all Pareto-efficient configurations is called the **Pareto frontier** of the space, and is noted $\mathbf{P}(\mathcal{C}) \subset \mathcal{C}$. Fig. 2 shows a 2D example of a Pareto frontier, using Python 3.10.

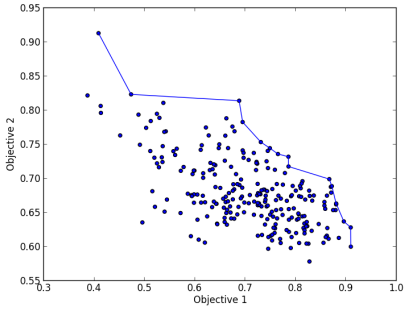

Fig. 2. A 2D Pareto frontier [3]

III. Methodology

The complete workflow used to in this paper to evaluate Pareto-efficient configurations is described in Fig. 3. It consists of three main stages: generating the exploration space $\mathcal{C}$, evaluating the values of $E(c)$, $P(c)$ and $R(c)$ for each configuration during the **fitness evaluation** process, and extracting the Pareto frontier from $\mathcal{EPR}$.

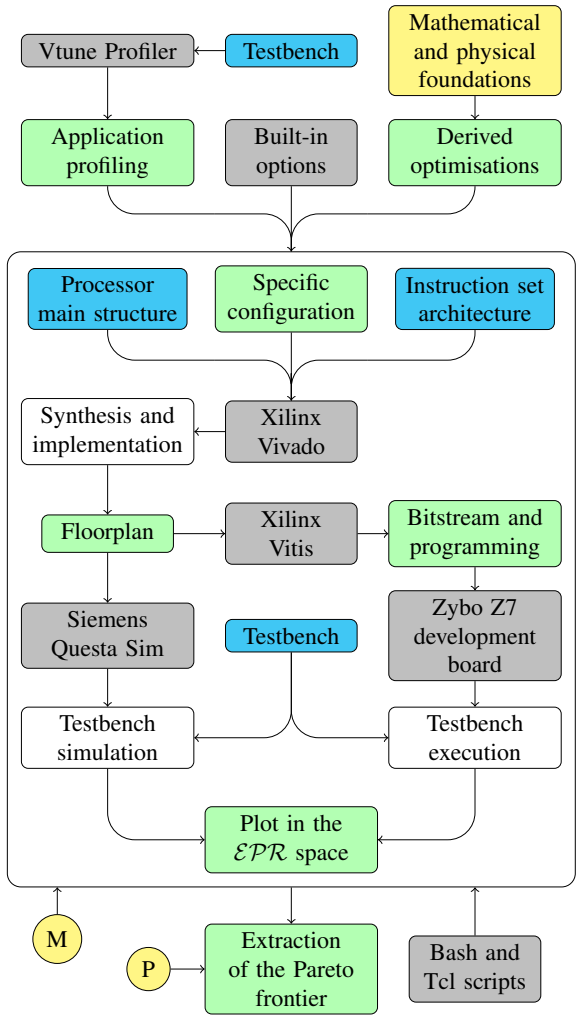

Fig. 3. The complete workflow with its fitness evaluation component ($\mathrm{M} = \#\mathcal{C} \approx N!\, e$, $\mathrm{P} = \#\mathbf{P}(\mathcal{C})$)

Before trying to extract the Pareto frontier to determine which configurations are best suited for different scenarios, we have to define the exploration space $\mathcal{C}$ by choosing which optimisation parameters will taken into account. This can be achieved by three different ways:

- An application profiling of our testbench can be performed using the standard configuration to see which functions or methods were the most time or energy consuming, and determine the impact of different processor components on application performance
- Low power techniques, elaborated based on our mathematical and physical understanding of the couple processor/application, can then be applied to all or some components of the processor
- We can finally choose to include some of the built-in options provided by the tools used to generate a processor configuration

Having defined the list of optimisation parameters, and thus $\mathcal{C}$, we may reduce its size using the ε-clustering or n-clustering methods. We can then begin to run fitness evaluation iterations for every configurations $c \in \widetilde{\mathcal{C}}$.

A fitness evaluation consists in the following steps:

- Alteration of the standard configuration, composed of a main structure and an instruction set architecture, to introduce the variations corresponding to the parameters used in this specific configuration in the correct order
- Register-transfer level (RTL) analysis
- Optional behavioural simulation
- Synthesis
- Placement and routing, which gives the value of $R(c)$
- Testbench simulation, which gives the value of $E(c)$
- Bitstream generation and FPGA programming
- Testbench execution, which gives the value of $P(c)$
- Plot of $f(c)$ in the $\mathcal{EPR}$ space

After having evaluated every configuration in $\widetilde{\mathcal{C}}$, the Pareto frontier can finally be extracted from the $\mathcal{EPR}$ space. Based on the physical constraints of the studied embedded system and taking into account the desired ratio in the energy/performance/resource trade-off, we can eventually choose the Pareto-efficient configuration(s) that best suits the use case scenario.

IV. Experimental Setup

A. The CVA6 Processor

CVA6 [4] [5] is is a 6-stage, single order open source processor implementing the RISC-V instruction set architecture and written in SystemVerilog HDL. It is the industrial evolution of ARIANE [6], a CPU developed by ETH Zürich and the University of Bologna, and is maintained by the OpenHW Group and Thales Group. Two versions of the processor are currently available: CVA6, implementing a 64-bit RISC-V core, and its CV32A6, a variant implementing a 32-bit RISC-V core. Both versions are fully compatible with I, M, A and C extensions. The pipeline stages are listed as follows:

- **PC generation**, generating the next program counter (PC) thanks to its **branch predict unit**.
- **Instruction Fetch** (IF), requesting the instruction from the instruction cache I$
- **Instruction Decoder** (ID), which re-aligns and decodes instructions
- **Issue**, verifying the availability of source operands and whether currently issued instruction will write the same destination register
- **Execute** (EX) stage, where each functional unit performs the operation they were issued
- **Commit** stage, or write-back (WB), retiring instructions and handling exceptions

The CVA6 processor also integrates an AXI (Advanced eXtensible Interface) bus capable of communicating with a DDR3 (Double Data Rate) SDRAM, a SPI (Serial Peripheral Interface) controller to connect to an SD card, an Ethernet controller, a JTAG (Joint Test Action Group) port, an UART (Universal Asynchronous Receiver Transmitter) port and a Bootrom.

Even though the OpenHW Group actively maintains the GitHub repository containing the CVA6 source code up to date, the code itself can be very confusing to read, as little information is provided regarding the way the SystemVerilog modules interact with one another. Furthermore, little to no literature currently exists on CVA6 or how to change its configuration, and only support for the Gensys2 development board [7] is currently provided. In fact, the very recently published [8] and [9] were the only two articles we could find that dived into the CVA6 structure more in depth than presented in the official documentation. A configuration file [10], describing the parameters that can be modified, can however be found amongst other source files. Those parameters are listed below:

- CONFIG_L1I_SIZE
- ICACHE_SET_ASSOC
- ICACHE_INDEX_WIDTH
- ICACHE_TAG_WIDTH
- ICACHE_LINE_WIDTH
- ICACHE_USER_LINE_WIDTH
- CONFIG_L1D_SIZE
- DCACHE_SET_ASSOC
- DCACHE_INDEX_WIDTH
- DCACHE_TAG_WIDTH
- DCACHE_LINE_WIDTH
- DCACHE_USER_LINE_WIDTH
- DCACHE_USER_WIDTH

B. Zybo Z7 and ZYNQ 7020

The FPGA used to program CVA6 is the Digilent Zybo Z7-20 development board [11] featuring a Xilinx ZYNQ 7020 SoC (System on Chip) [12], as shown in Fig. 4. The SoC consists of two separate parts communicating together using AXI ports: the Processing System (PS) and the Programmable Logic (PL).

The PS contains a dual-core ARM Cortex A9, an Advanced Microcontroller Bus Architecture (AMBA) interconnect, a

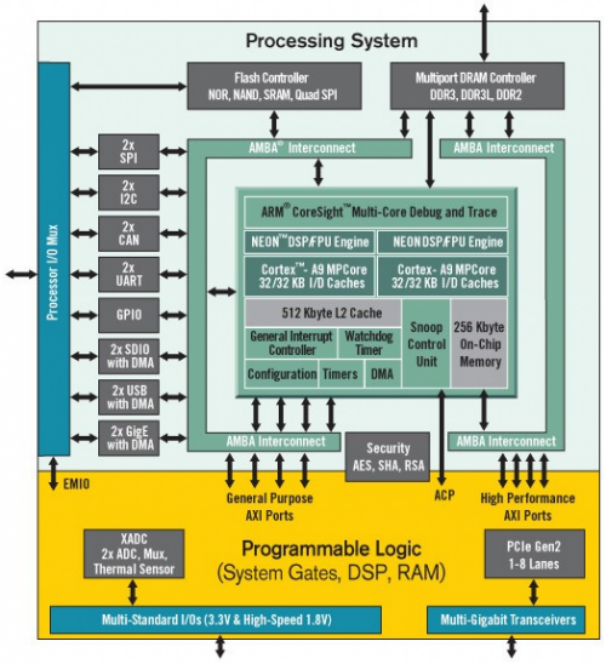

Fig. 4. Functional view of the Xilinx ZYNQ 7020 SoC

DDR3 Memory controller and other peripheral controllers. The Cortex A9 processors act as masters to all the other components of the SoC, including the PL, and can be used to monitor various signals sent by the PL through the AXI ports.

The PL is the configurable part of the FPGA, where CVA6 will be after it has been programmed with the bitstream. It uses a 28nm technology. Table I gives the number of logic cells and flip-flops on the PL, along with the amount of RAM available.

Device Name	Z-7020
Part Number	**XC7Z020**
Xilinx 7 Series Programmable Logic Equivalent	Artix-7 FPGA
Programmable Logic Cells	85K
Look-Up Tables (LUTs)	53,200
Flip-Flops	106,400
Block RAM (# 36 Kb Blocks)	4.9 Mb (140)
DSP Slices (18x25 MACCs)	220
Peak DSP Performance (Symmetric FIR)	276 GMACs
PCI Express (Root Complex or Endpoint)[3]	

Table I. Technical data regarding the PL

It can be programmed using a three-stage boot protocol from three different sources: a micro SD slot, a Quad SPI-flash memory and a JTAG port. Boot stage 0 stage includes

the execution of a program called the BootROM, selecting a boot source, and copying the First Step Boot Loader (FSBL). Stage 1 corresponds to executing the FSBL, setting up PS components and configuring the PL by loading and reading the bitstream. Eventually, stage 2 is the execution of any user application.

It provides a 33.333 MHz clock that allows the processor to operate at 663 MHz and the DDR bus at 533 MHz. An external clock of frequency 125 MHz is directly connected to the PL, allowing it to be completely independent from the PS as well a generating two Mixed-Mode Clock Manager (MMCM) and two Phase-Locked Loop (PLL).

The Zybo Z7 board can be connected using three different power sources: using a USB cable, using a Jack connector, or direct plugging a battery into the circuit. It requires a 5V power supply to be operational.

C. Testbench Application

The studied testbench consists in propagating a 24*24 pixels black and white image through a simple **convolutional neural network** (CNN) written in C. It has a total of four layers, two convolution layers and two fully connected layers. Fig. 5 represents the input, output and hidden neuron distribution between the different layers of the CNN.

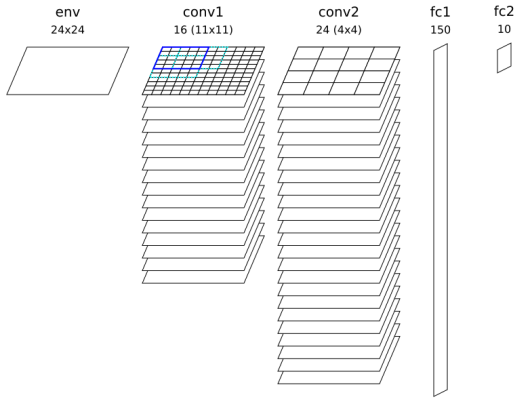

Fig. 5. Neuron distribution in the CNN

The propagated image corresponds the number 4 in shades of grey, and is represented as a buffer of 8-bit unsigned integers. The CNN is already trained (the weights of each link can be found in a header file), so only the loading and the propagation need to be executed. At the end of the propagation, certain values are displayed, among which:

- the expected value (4)
- the predicted value
- the number of instructions executed
- the number of clock cycles

It must be compiled using the RISC-V GNU Compiler Toolchain [13], a C/C++ cross-compiler dedicated to produce a RISC-V compatible **executable and linkable format** (ELF) called elf32-littleriscv. The compiler is very similar in its use to the gcc compiler, can be called using the $riscv32\text{-}unknown\text{-}elf\text{-}gcc$ command. We used the following compilation options:

- O3: the highest standard optimisation possible
- static: the program won't require a dependency on dynamic libraries at runtime
- Wall: enables warning messages
- pedantic: adds warnings to encourage strict adherence to ISO C

The final compiled program is 22931 instructions long, and can be directly loaded into the CVA6 instruction cache during the FPGA programming. Similarly, the weight of the CNN and the buffer corresponding to the image are loaded into the data cache during programming.

D. Software Automation

In September 2021, Thales, along with the GDR SoC2 and the CNFM, launched the 2nd National RISC-V Student Contest [14], a competition designed to motivate the interest of french students in electronics in the RISC-V architecture by analysing and modifying an FPGA emulation of a RISC-V processor. Groups of students were given access to a GitHub repository [15] where they could find various files, including:

- The complete source code of the ARIANE/CV(32)A6 processor
- Source codes of multiple testbench applications written in C to be executed on RISC-V architecture processors
- A link to the RISC-V GNU Compiler Toolchain GitHub repository to compile the testbench on the RISC-V ISA
- Elements of Tcl scripts to automate some aspects of the Vivado workflow (synthesis, simulation, implementation)
- Documents describing the required material, software and how to use the scripts

This contest provided us with a solid base to run and automate the workflow described in section III.

A 32 GB RAM 6 hyper-threaded Intel Core i7 computer using Ubuntu 18.04 was used to compute the entire workflow. To be compliant with the contest, Vivado 2020.1 [16], Vitis 2020.1 [17] and Questa Sim 10.7 [18] were used by the fitness evaluation process. Vtune Profiler 2022.3 [19] was used to perform the application profiling of the testbench, and the $\mathcal{EPR}$ plot and the Pareto frontier extraction were performed under Python 3.10. Communication with the FPGA was established thanks to OpenOCD 0.10 [20] on a minicom terminal. Each fitness evaluation iteration was automated by Tcl scripts while the whole workflow was run by bash scripts.

The clock frequency was set to 45 MHz, as the default value given by the contest organisers.

Due to technical limitations, the execution branch could not return an execution duration, but instead returned the number of instructions and cycles. This is not an issue as execution duration can be obtained by calculating $T = n_{cycles}/f_{CK}$. However, due to branch predict errors, amongst other things, the value of n_{cycles} may be slightly increased because more instructions were executed. To solve this, the value returned by the performance function is the $CPI := n_{cycles}/n_{instr}$ (cycles per instruction) instead of the computation time. We thus have:

$$T = f_{CK} \times n_{instr} \times \underbrace{CPI}_{P(c):=} \tag{19}$$

Since $P^0(c) = P(c)/P(0_{\mathcal{C}})$ and given that clock frequency is the same for every configurations, the only impact of considering CPI instead of T is actually to remove variations in the number of instructions executed, which introduced noise and isn't representative of an optimisation.

Similarly, Questa Sim only gives the total power distribution and not the total energy spent. Because $E = P_{tot} \times T$ and that T is prone to non configuration-related uncertainties, taking power consumption as the value of $E(c)$ also reduces the noise on $E^0(c)$ with changing its meaning.

V. Results and Analysis

16 parameters (clock gating and 15 Vivado implementation parameters) were considered when defining $\mathcal{C}$. $\mathcal{C}_0$ and $\mathcal{C}_1$ were fully explored, and 6 elements of degree 7 to 10 were evaluated. Fig. 6 to 9 show the 23 evaluated configurations plotted in the $\mathcal{EPR}$ space.

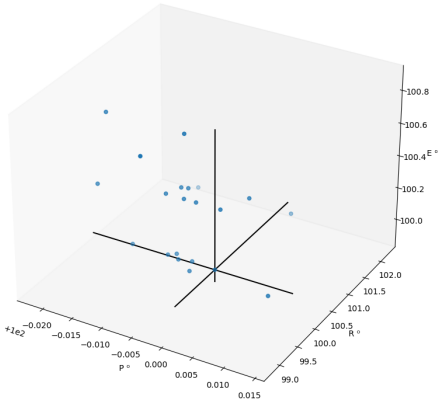

Fig. 6. The $\mathcal{EPR}$ space

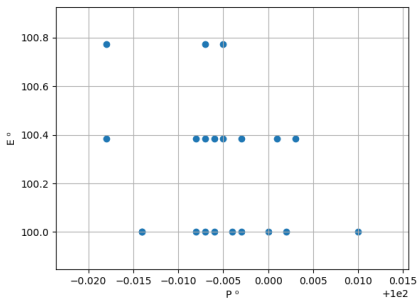

Fig. 7. The $\mathcal{EPR}$ space projected on the $\mathcal{PE}$ plan

In spite of the relatively small number of points, a Pareto Frontier is easily identifiable on the three projections. It must be noted that the power consumption given by Questa were only precise to 1mW, which explains the discretisation that can be observed regarding the distribution of the values of E^0.

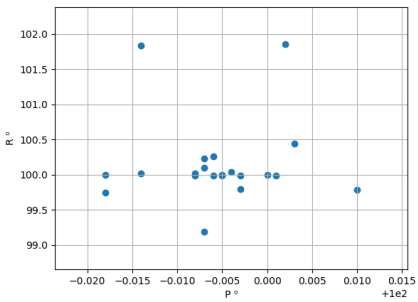

Fig. 8. The $\mathcal{EPR}$ space projected on the $\mathcal{PR}$ plan

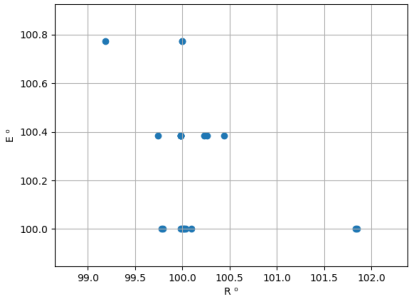

Fig. 9. The $\mathcal{EPR}$ space projected on the $\mathcal{RE}$ plan

The maximum effective distance between two points is 2.77, with an average of $7.14 * 10^{-1}$ and a standard deviation of $6.28 * 10^{-1}$. While those numbers cannot be easily interpreted as such, they show that most evaluated configurations are gathered around $0_{\mathcal{C}}$, with some others being unusually far from them. Analysing the data shows that the first set of points is mostly composed of degree 1 configurations, while outside points usually come from directives. In fact, only considering $\mathcal{C}_1$ drops the maximum distance, average and standard deviation to $7.98 * 10^{-1}$, $3.51 * 10^{-1}$ and $2.06 * 10^{-1}$ respectively. This indicates that the degree of a configuration seems to be in direct correlation with the fitness difference from $0_{\mathcal{C}}$: the more optimisations were used in a configuration, the more its energy consumption, performance and resource use differ from the standard configuration.

If we consider each component of $\mathcal{EPR}$ separately, more information can be gathered. Table II summarises the configuration distribution for E^0, P^0 and R^0 gains respectively.

	E^0	P^0	R^0
maximum	7.72 ‰	$1.00 * 10^{-1}$ ‰	18.5 ‰
minimum	0.00 ‰	$-1.80 * 10^{-1}$ ‰	−8.32 ‰
average	2.56 ‰	$-5.57 * 10^{-2}$ ‰	1.52 ‰
std. dev.	2.75 ‰	$6.67 * 10^{-2}$ ‰	5.96 ‰

Table II. Energy, performance and resource gain statistics, in ‰ of $0_{\mathcal{C}}$

As stated before, most degree 1 configurations bring the values of both the average and the standard deviation for each gain closer to 0. This lack of diversity can hopefully

be balanced by evaluated high degree configurations.

The tendency of the $\mathcal{EPR}$ distribution to be slightly shifted towards negative performance gains, along with performance being the least impacted component of the energy/performance/resource trade-off, is most likely due to the fact that no directly performance-related optimisation were considered, dedicating all the optimisation efforts to energy consumption and resource use.

On the other hand, resource use being by far the most impacted component can be explained by the fact that we almost only considered implementation optimisations, meaning that most of the optimisation efforts were used in trying to optimise placement, i.e. resource use.

VI. Future Work

The work presented in this paper is far from being over. In spite of limited time given to run the fitness evaluation process, the obtained results already are very encouraging, and suggest that evaluations of higher degree configurations may lead to more significant benefits. Parallelising configuration evaluations on multiple computers will allow to generate more data.

Furthermore, a lot more work would be required to have a firm grasp on the software, the RISC-V ISA and CVA6. In particular, the absence of extensive documentation about CVA6 considerably slowed the process of choosing useful parameters from both CVA6 itself and Vivado.

Eventually, since ε-clustering and n-clustering techniques induce approximations by altering the configuration space, we would have liked to also try a different approach. A possible alternative we came up with would be to implement genetic algorithms using Pareto-efficient configurations as base for the selection stage.

VII. Conclusion

In this paper, we have proposed a mathematical model to describe how various energy, performance and resource optimisations can be implemented on a processor to create alternative versions, called configurations. We then developed a general, technology-agnostic workflow taking the list of optimisations as an input, and returning a set of Pareto-efficient configurations for the user to choose from. Eventually, we used the RISC-V ISA CV32A6 processor and a CNN propagation testbench as a proof of concept. Due to schedule limitations, we were not able to run the workflow in its entirety, but we obtained promising results that are highly encouraging for the future.

With the explosion of low-power techniques and the ever growing development of computer-assisted tools for embedded systems, there are still too little technology-independent automated conception designs in modern literature, especially for newer processors. Engineers and researchers are forced to use rule of thumbs and highly heuristic methods, and would greatly benefit from having an automated exhaustive workflow from both a theoretical and practical point of view. We hope that our findings will help making non-heuristic time-efficient methods become more widely used in the fields of system engineering and microelectronics.

Acknowledgements

We thank Goran Frehse, director of the U2IS laboratory, for the hardware and software resources he made available for our research.

We thank Hervé Le Provost and Farhat Thabet for their assistance in setting up and using the software.

We are also grateful for Frederic Dulucq, for showing a different angle to our approach.

References

[1] https://open-src-soc.org/2022-05/posters.html
[2] G. Franchi, "Unsupervised Learning", ENSTA Paris
[3] https://sirinnes.wordpress.com/2013/04/25/pareto-frontier-graphic-via-python/
[4] https://github.com/openhwgroup/cva6
[5] https://docs.openhwgroup.org/projects/cva6-user-manual/
[6] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 11, pp. 2629-2640, Nov. 2019, doi: 10.1109/TVLSI.2019.2926114.
[7] https://digilent.com/reference/programmable-logic/genesys-2/reference-manual?redirect=1
[8] Martinoli, Valentin et al. "CVA6's Data cache: Structure and Behavior." ArXiv abs/2202.03749 (2022)
[9] Open Virtual Plateform, "OVP guide to using processor podels - model specific information for OpenHWGroup CV32A6", 27 July 2022
[10] https://github.com/openhwgroup/cva6/blob/master/core/include/ariane_pkg.sv
[11] https://digilent.com/reference/_media/reference/programmable-logic/zybo-z7/zybo-z7_rm.pdf
[12] https://docs.xilinx.com/v/u/en-US/ds190-Zynq-7000-Overview
[13] https://github.com/riscv-collab/riscv-gnu-toolchain
[14] https://web-pcm.cnfm.fr/wp-content/uploads/2021/10/Annonce-RISC-V-contest-2021-2022-v1.pdf
[15] https://github.com/ThalesGroup/cva6-softcore-contest
[16] https://www.xilinx.com/products/design-tools/vivado.html
[17] https://www.xilinx.com/products/design-tools/vitis.html
[18] https://eda.sw.siemens.com/en-US/ic/questa/simulation/advanced-simulator/
[19] https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html
[20] https://openocd.org/