A High-Level Modeling Framework for Estimating Hardware Metrics of CNN Accelerators

Leonardo Rezende Juracy, Matheus Trevisan Moreira, Alexandre de Morais Amory, Alexandre F. Hampel, and Fernando Gehm Moraes, *Senior Member, IEEE*

***Abstract*— GPUs became the reference platform for both training and inference phases of Convolutional Neural Networks (CNN) due to their tailored architecture to the CNN operators. However, GPUs are power-hungry architectures. A path to enable the deployment of CNNs in energy-constrained devices is adopting hardware accelerators for the inference phase. The design space exploration of CNNs using standard approaches, such as RTL, is limited due to their complexity. Thus, designers need frameworks enabling design space exploration that delivers accurate hardware estimation metrics to deploy CNNs. This work proposes a framework to explore CNNs design space, providing power, performance, and area (PPA) estimations. The heart of the framework is a system simulator. The system simulator front-end is TensorFlow, and the back-end is performance estimations obtained from the physical synthesis of hardware accelerators, not only from components like multipliers and adders. The first set of results evaluate the CNN accuracy using integer quantization, the accelerators PPA after physical synthesis, and the benefits of using a system simulator. These results allow a rich design space exploration, enabling selecting the best set of CNN parameters to meet the design constraints.**

***Index Terms*— CNN, convolution hardware accelerator, system simulator, PPA, design space exploration.**

I. Introduction

MACHINE Learning (ML) is a sub-area of artificial intelligence with algorithms that can solve problems involving knowledge and "learning" characteristics from determined patterns. This "learning" feature allows decision capability to solve problems of classification and pattern recognition [1]. For these reasons, industrial applications adopt ML on their products [2], [3].

Manuscript received April 21, 2021; revised June 29, 2021; accepted August 10, 2021. This work was supported in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under Grant 309605/2020-2 and in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Finance Code 001. This article was recommended by Associate Editor F. Rivet. *(Corresponding author: Fernando Gehm Moraes.)*

Leonardo Rezende Juracy, Alexandre F. Hampel, and Fernando Gehm Moraes are with the School of Technology, Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre 90619-900, Brazil (e-mail: leonardo.juracy@acad.pucrs.br; alexandre.hampel@acad.pucrs.br; fernando.moraes@pucrs.br).

Matheus Trevisan Moreira is with Chronos Tech, San Diego, CA 92122 USA (e-mail: matheus@chronostech.com).

Alexandre de Morais Amory is with TeCIP Institute, Scuola Superiore Sant'Anna, 56124 Pisa, Italy (e-mail: alexandre.amory@santannapisa.it).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2021.3104644.

Digital Object Identifier 10.1109/TCSI.2021.3104644

One of the most common ways to deliver ML is by using Artificial Neural Networks (ANN). ANNs contain thousands of interconnected neurons. Synapses have data input samples, plus a weight that works similarly to a filter and activation function, which creates an output used in synapses of subsequent neurons [1], [4].

Convolutional Neural Networks (CNNs) are a type of ANNs. CNNs have the advantage of having sparse connections, which reduces the number of interconnected neurons [1]. A CNN contains four main layers: (*i*) convolutional layer, which is the CNN core, and performs the synapses by multiplying and accumulating weights and input feature maps; (*ii*) activation function, nonlinear transformation sent to the next layer of neurons; (*iii*) pooling layer, reduces the amount of data processed by the CNN; (*iv*) fully connected layer, used in the classification result.

The deployment of CNNs comprises two phases: training and inference [4]. The training phase defines the synapses weight values. The inference phase uses the weights previously computed to classify or predict output values using unknown inputs. The success of CNNs led to the development of frameworks that help developers to build their models by offering mechanisms required for training and inference. Examples of frameworks include Caffe [5], Pytorch [6] and TensorFlow [7].

Classically, CPUs have been a common approach to execute CNNs. Even with optimized instruction set architectures, CPUs are inefficient in terms of performance and energy (e.g., AlexNet from 2012 [8] requires billions of operations to process a single input). Thus, GPUs became the reference platform for both training and inference phases due to their tailored architecture to the CNN operators. The main GPU drawback is its considerable energy consumption. Considering energy-constrained applications, such as IoT, autonomous driving, wearable devices, the adoption of hardware accelerators became a trend for the inference phase [9]–[12].

Accelerators are hard to implement and verify using classic design flows. The design of these blocks using register transfer level (RTL) abstraction limits the design space exploration (DSE). Despite the efforts to increase the abstraction level for accelerators using high-level synthesis (HLS) [13], [14], this approach also has challenges related to performance and power estimation, once the goal of HLS is to generate an RTL description as output.

System simulators [15], [16] are important tools for accelerators DSE. These simulators are typically described in

high-level abstraction languages, as Python and C++, reducing the design time and providing power, performance, and area (PPA) evaluation. The main drawback of system simulators is the PPA accuracy, typically estimated from the number of executed operations, as multiplier–accumulator (MAC) [15], [17].

The *goal* of the present work is to propose a framework to estimate hardware metrics, using the advantages of the TensorFlow regarding CNN modeling, and the benefits of system simulators regarding design time and PPA evaluation.

This work brings three main *contributions*:

1) integration of TensorFlow into a system simulator to evaluate accelerators in the inference phase;
2) proposal of two convolution accelerators at the RTL level, detailing their memory accesses and internal architectures. These accelerators are synthesized to provide PPA metrics;
3) integration of the two above contributions into a system simulator resulting in a framework enabling design space exploration from PPA estimations taken from the physical synthesis of accelerators, leading to accurate hardware estimations.

These contributions lead to the main original contribution: the method to estimate the PPA trade-offs based on the entire convolution and not on its basic components, such as multipliers and adders. This approach allows generating accurate results because it considers the convolutional accelerator with its arithmetic modules, buffers, and wire delays. The related work section unveils this gap, fulfilled by this work.

This article extends the authors' previous work [18], which shares the proposal of a framework for PPA evaluation of CNNs. This work brings updated related work, refinement of the framework description, use of two hardware accelerators types (Section IV), and a set of new results, including the synthesis of accelerators synthesis (Sections V-B and V-C), and design space exploration (Section V-D), summarized in Figure 8.

The remainder of this paper is organized as follows. Section II presents related works related to simulators and frameworks for CNNs. Section III details the proposed framework. Section IV presents the descriptions of the proposed accelerators, comprising a systolic 2D and a 1D array. Section V presents 4 sets of results: (i) evaluation of the CNN accuracy using integer quantization; (ii) the accelerators PPA after physical synthesis; (iii) the benefits of using a system simulator; (iv) design space exploration, enabling selecting the best set of the CNN parameters to meet the design constraints. Finally, Section VI concludes this paper, pointing out the direction for future works.

II. Related Work

This section describes works that generate PPA analyses focused on simulators of CNNs and frameworks related to our proposal. Estimation frameworks can use a simulator to estimate PPA based on the hardware behavior or use analytical methods to evaluate PPA quickly. The simulators are commonly implemented using high-level program languages, such as Python and C++, and simulate the CNN accelerator faster than RTL approaches.

A. Frameworks and Simulators

MAESTRO [19] is a framework to describe and analyze Neural Network hardware, which allows obtaining the hardware cost to implement a target architecture. It has a domain-specific language (DSL) to describe the dataflow that specifies the number of PEs, the memory size, and NoC bandwidth parameters. The results generated by the framework are focused on performance analyses. In recent work [25], MAESTRO was used to estimates cost-benefit tradeoffs between execution time and energy efficiency for CNN models, such as VGG and AlexNet, and hardware features, like buffer size. MAESTRO does not allow the accelerator simulation, which limits the performance evaluation.

SCALE-Sim (Systolic CNN Accelerator Simulator) [20] is a cycle-accurate systolic array simulator. This simulator allows configuring micro-architectural features such as array size, array aspect ratio, scratchpad memory size, and dataflow mapping strategy. SCALE-SIM simulates convolutions and matrix multiplications, and it models the compute unit as a systolic array. Also, it allows simulation in a system context with CPU and DMA components. The authors show detailed experiments to understand the design space and tradeoff in designing a systolic array-based CNN accelerator. Also, SCALE-Sim provides an analytic model to find the best accelerator configuration based on parameters like DRAM bandwidth. However, SCALE-Sim does not give power or energy results.

Timeloop [15] is a DSE framework for CNNs which emulates accelerators such as the NVDLA [26]. It focuses on the convolution layer analyses. Timeloop uses as input a workload description, such as input dimension and weight values, a hardware architecture description, such as arithmetic modules, and the hardware architecture constraints, such as the computation partition. Instead of using a cycle-accurate simulator, Timeloop uses data transfers deterministic behavior to perform analytic analyses. As energy models, Timeloop has memory, arithmetic units, and wire/network models based in TSMC 16nm FinFET. Timeloop provides PPA based on basic operations, such as adders and multipliers.

Accelergy [17] allows estimating the energy of accelerators without a complete hardware description and fine-grained analyses using a library of basic components. Accelergy uses a high-level architectural description to capture the circuit behavior characteristics, such as memory reads. The obtained results are compared to post-layout results, showing an error of 5% in the energy estimation for the Eyeriss accelerator. Even considering the number of memory reads, Accelergy does not take into account some important features of an accelerator. Accelergy considers whether the memory access pattern is random or whether it reads the same address repetitively, but it does not take into account dataflow types and data movement through the array. Besides, Accelergy does not provide a simulation environment.

TABLE I
STATE-OF-THE-ART SUMMARY (PPA: POWER, PERFORMANCE, AND AREA)

Work	Integration with CNN frameworks	Simulation	Evaluation metrics based on basic components	PPA analyses based on entire convolution
MAESTRO – 2019 [19]	–	–	Performance	–
SCALE-Sim – 2020 [20]	–	✓	Performance, Area	–
Timeloop – 2019 [15]	–	–	PPA	–
Accelergy – 2019 [17]	–	–	Power	–
STONNE – 2020 [16]	✓	✓	Performance	–
SimuNN – 2020 [21]	✓	✓	PPA	–
AccTLMSim – 2020 [22]	–	✓	Performance	–
Heidorn et al. – 2020 [23]	–	–	PPA	–
Zhao et al. – 2020 [24]	–	–	PPA	–
Our proposal	✓	✓	**–**	**PPA**

STONNE [16] is a cycle-accurate architecture simulator for CNNs which allows end-to-end evaluation. It is connected with Caffe framework [5] to generate the CNNs, and it models the MAERI accelerator [27]. The results are focused on performance and hardware utilization regarding the percentage of multipliers.

SimuNN [21] is a neural network simulator that allows pre-RTL verification and fast prototyping. It is compatible with TensorFlow, allowing using software application values to evaluate the hardware accelerator. The results generated by SimuNN are based on a fixed accelerator proposed by the authors. The accelerator comprises a micro-controller, an instruction RAM, a DDR controller, a weight buffer, a feature map buffer, a feeder controller, a collector unit, and 14 three-stage pipelines PEs with nine multipliers each. The Authors show latency and energy results based on Altera FPGAs and ASICs, although the ASIC technology node is not mentioned.

STONNE and SimuNN are similar frameworks when compared to our proposal. Both integrate a flow that starts with frameworks to model CNNs, and both provide the accelerator simulation. However, SimuNN uses a fixed 2D array style, not comparing it with other styles like 1D (discussed in Section IV). SimuNN has an energy estimation based on basic elements, not considering data movement through the accelerator. STONNE does not address power estimation, but the authors discuss that it is possible to integrate STONNE with Accelergy.

AccTLMSim [22] is a pre-RTL cycle-accurate CNN accelerator simulator based on SystemC transaction-level modeling (TLM). The simulator allows maximizing the throughput performance for a given on-chip SRAM size. An accelerator is proposed to validate the simulator, composed of a MAC array of 12 units, a double buffer scheme to enable memory read and MAC executions in parallel, and a DRAM controller. Each of the hardware blocks is implemented as a SystemC module using sockets, and the accelerator was also prototyped in a Xilinx Zynq FPGA using High-Level Synthesis (HLS). AccTLMSim is focused only on performance, not power or area.

Heidorn *et al.* [23] propose an analytical model that estimates throughput and energy to a given hardware constraint. A DSE is proposed to determine the accelerator architecture limits in terms of throughput, number of parallel operations, and memory. The Authors propose an accelerator to evaluate the model with a tile-local memory, a bus, and a coarse-grained reconfigurable array (CGRA). Each CGRA presents a two-dimensional array of PEs, and the accelerator can have more than one CGRA to parallelize the processing. Compared to implementations that execute a CNN layer-by-layer sequentially, results show that layer-parallel processing can reduce energy consumption by 3.6 times, hardware cost by 1.2 times, and increase throughput by 6.2 times for a MobileNet.

Zhao *et al.* [24] propose an analytical performance predictor to estimate energy, throughput, and latency for ASIC and FPGA. The predictor uses DNN models, hardware architecture, dataflows types, and hardware cost regarding a technology node. The results are generated with AlexNet and SkyNet DNN models, with Eyeriss, an FPGA implementation from [28], and synthesized results of a proposed accelerator. They show that the error achieves a minimum of 0.25% and a maximum of 17.66% for different DNN models, hardware architectures, and dataflow types.

Works [23] and [24] show analytical results for power, performance, and area. Also, [24] consider features like the dataflow type, which can contribute to the power consumption. However, both [23] and [24] do not support simulation neither integration with CNN frameworks.

Specific-domain frameworks targeting commercial platforms like Vitis AI from Xilinx [29] and TensorRT from NVIDIA [30] aid in model hardware for CNNs. These frameworks model the CNN using frameworks such as TensorFlow, and using high-level synthesis, convert the model to the hardware using custom IPs (Xilinx) or specific platforms (e.g., Jetson). However, these frameworks use proprietary IPs, limiting the design space exploration.

B. State-of-the-Art Summary

Table I summarizes the reviewed works. The 1th column represents if the work has integration with high-level modeling CNN frameworks, such as TensorFlow and Caffe. The 2th column shows if the work provides a simulation environment. The 3th and 4th columns are related to the evaluated

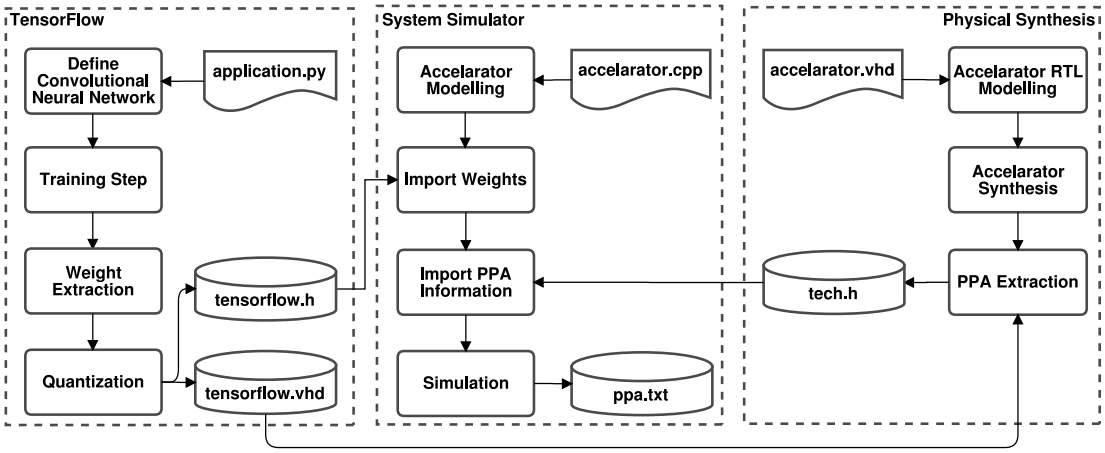


Fig. 1. Convolution accelerator hardware metric extraction framework. Adapted from [18].

metrics. The 3th column represents evaluated metrics based on basic components, such as MACs and register files. The 4th column shows the evaluated metrics regarding the entire convolution.

Four of nine works present a PPA analysis. MAESTRO focuses only on performance, while Acelergy on energy. These works must be integrated with simulators to provide a PPA analysis. Few works, as STONNE and SimuNN, integrate a high-level framework to build a CNN, with a simulator environment capable of validating the accelerator behavior and a PPA analyses method. Methods relying on counting operations, like SCALE-Sim and Accelergy, do not consider how these operators are interconnected (e.g., 1D or 2D systolic arrays or adder trees), resulting in imprecise hardware metrics, such as power.

As shown in the last line of Table I, the proposed framework has integration with TensorFlow and a system simulator. Our proposal fulfills an important gap identified in the literature, estimating the hardware metrics related to the CNN convolution operators considering the arithmetic modules, buffers, and wire delays, and not only MAC counting.

III. Proposed Framework

Figure 1 presents the proposed framework. TensorFlow models the CNN, being responsible for training and inference phases, resulting in capturing the weight and input feature maps values.

This work adopts an integer quantization to avoid floating-point operations in the accelerator by applying power of two multiplications. The last action executed using TensorFlow is exporting a header file containing the weight and feature values to be used by the system simulator.

The physical synthesis corresponds to the synthesis of the CNN accelerators. For each one of the accelerators described in Section IV, this step generates the layout of the CNN operators, and a netlist with extracted parasitic capacitances. The simulation of this netlist enables extracting the switching activity to characterize the accelerator dynamic power. This simulation uses weight and input feature map values generated from TensorFlow, showed in Figure 1 as **tensorflow.vhd**. Thus,

```
# Clearup everything before running
keras.backend.clear_session()

# Create model
model = keras.models.Sequential()

# Add layers
model.add(keras.layers.Conv2D(16, (3,3), strides=(2, 2), activation='relu',
↪  input_shape=(28, 28, 1)))
model.add(keras.layers.Conv2D(8,  (3,3), strides=(1, 1), activation='relu'))
model.add(keras.layers.Conv2D(3,  (3,3), strides=(2, 2), activation='relu'))
model.add(keras.layers.Conv2D(1,  (3,3), strides=(1, 1), activation='relu'))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation='softmax'))

# Build model and print summary
model.build(input_shape=featureShape)
model.summary()
```

Fig. 2. TensorFlow code example [18].

it is possible to measure the power of real CNN architectures using actual inputs. The result of the physical synthesis is the PPA report.

The URSA cycle-accurate system simulator [31] models the hardware accelerator, integrated with the CNN model generated by TensorFlow and the PPA reports generated by the physical synthesis. The simulator captures information related to the CNN execution, presenting a summary with accelerator performance, area, and energy results. The next sections detail the framework.

A. TensorFlow

TensorFlow [7] is a Google framework providing libraries to implement ML applications. TensorFlow allows implementing CNNs, including the training and inference phases. It is possible to use CNN functions such as 2D convolution, max pooling, and ReLU. This work uses the TensorFlow for:

1) Modeling the CNN and exploring its architecture;
2) Extracting the weight values of the selected network;
3) Extracting network output values to validate post-layout simulation;
4) Evaluating the weights quantization from 32-bit floating-point to 8-bits integer.

Figure 2 shows an example of a TensorFlow code, which corresponds to the **application.py** in Figure 1. The environment allows exploring CNN architectures and their accuracy regarding the network depth, stride dimension, activation functions, and the number of filters. Thus, it is possible to

tune the CNN architecture based on an target accuracy. The example in Figure 2 shows a CNN with four convolution layers with 16, 8, 3, and 1 filters, a fully connected layer, and strides with dimensions 2×2 and 1×1.

TensorFlow also allows extracting the value of weights after reaching the target accuracy – training phase. A post-extraction quantization happens after the training phase by converting the floating-point weights to 8-bit integers by multiplying the weight values by a power of two. Adopting integer values avoids floating-point arithmetic in the accelerator, reducing its area and power consumption.

Two files are exported at the end of this step: (*i*) a header with the weight and feature map values to be imported by the system simulator (**tensorflow.h** in Figure 1); (*ii*) a VHDL package that contains the weight, feature map, and expected output values to simulate and validate the hardware implementation (**tensorflow.vhd** in Figure 1). These files are created directly by TensorFlow, given that this framework is extensible using the Python language. These values come from real datasets, such as MNIST and CIFAR10.

B. PPA Extraction

This step requires the CNN accelerator RTL description. This work provides two distinct descriptions, described in Section IV, to validate the proposed framework.

Cadence Genus and Innovus tools are used to execute synthesis and place and route (P&R). The accelerator area includes gates and wires, and not only cell counting. The simulation of the post-P&R netlist provides the accelerator performance (operating frequency) and the switching activity (value change dump (VCD) file). The VCD file provides the inputs for dynamic power estimation. This post-layout simulation uses the **tensorflow.vhd** file exported from TensorFlow to simulate and validate the hardware implementation using real application values. The PPA metrics are exported to the system simulator in a header format (**tech.h** in Figure 1).

C. URSA System Simulator

URSA [31] is a C++ API for system-level modeling and simulation. It provides a set of language-related assets that can be used to create system-level, cycle-accurate hardware simulators, like SystemC. The URSA hardware models are modeled as a set of finite state machines (FSM), and its underlying simulation is based on discrete-event simulation. A clock cycle in URSA corresponds to the activation of the transition function of the FSMs of the simulated system. This high-level model brings the following advantages: (*i*) possibility to describe hardware modules in an abstract way; (*ii*) generate gold models for circuit verification; (*iii*) the object-oriented approach allows reusing the hardware description, making it easier to build new hardware models.

This work uses the URSA to:

1) Model and simulate the CNN;
2) Model and simulate the accelerator;
3) Validate the CNN accuracy;
4) Generate PPA evaluation of the CNN.

```
void Testbench::TBInit(){
        if (_array->GetEOP() == 0 && _wait_eop == 0 && _end_of_layer1 == 0) {
                Conv2d(Layer1_Weights,Input,16,3,3,2,2,28,28,1);
        }
}

void Testbench::TBStore(){
        if(_array->GetEOP() == 1) {
                StoreOfmap(Layer2_Input,Layer1_Bias,&_end_of_layer1,1,16);
        }
}
```

Fig. 3. URSA simulator code example.

Figure 3 shows how a CNN is modeled in URSA. One layer of a CNN is simulated in this example, composed of 16 filters with dimension 3×3, strides with dimension 2×2, and the input feature map (IFMAP) with dimension $28 \times 28 \times 1$, same parameters of the first convolution layer showed in Figure 2. The **TBInit()** function is responsible for performing the memory read, feeding the accelerator, and executing it. When the accelerator is done (signalized by **_array->GetEOP() == 1**), the output value is stored in the **TBStore()** function, which is also responsible for controlling the end of the simulation.

The CNN application is simulated in URSA using the header files generated in Section III-A (**tensorflow.h**), the technology reports generated in Section III-B (**tech.h**), and the accelerator array (**accelerator.cpp** in Figure 1). Thus, it is possible to simulate a complete CNN faster than RTL simulations (Section V-C). Also, the simulator reports the CNN energy and performance estimation when the simulation finishes, according to the number of executed convolutions. Thus, the simulator performs analyses regarding the PPA values extracted from the physical synthesis, and the captured application information at the simulation, resulting in a fast estimation.

IV. Accelerators Description

This section introduces concepts related to CNN accelerators and proposes two accelerators to validate the proposed framework. According to [32], accelerators may be classified according to the following criteria:

- Array style: 1D systolic [33], 2D systolic [34], 1D array [35], and 2D matrix [36];
- Dataflow types: Weight Stationary (WS) [33], Output Stationary (OS) [36], Input Stationary (IS) [10], No Local Reuse (NLR) [35], and Row Stationary (RS) [37];
- External Memory.

The basic array component is the Processing Element (PE), which contains the Arithmetic-logic Unit (ALU), the Control Unit (CU), and the Register file (RF). The ALU includes elements such as adders and multipliers. CU refers to the logic to control the array. Finally, the RF contains the registers and buffers where the computation is locally stored, including IFMAP values, weight values, and partial outputs.

In the 1D systolic architecture, PEs are connected sequentially, where each PE has a maximum of two neighbors, with data transferred in a pipeline fashion. A systolic 2D is similar but can have more than two neighbors, arranged as a matrix. The 1D array is similar to the systolic 1D, but data is transferred in broadcast to the PEs. Similarly, the 2D matrix also transfers data in broadcast, but in a matrix arrangement.

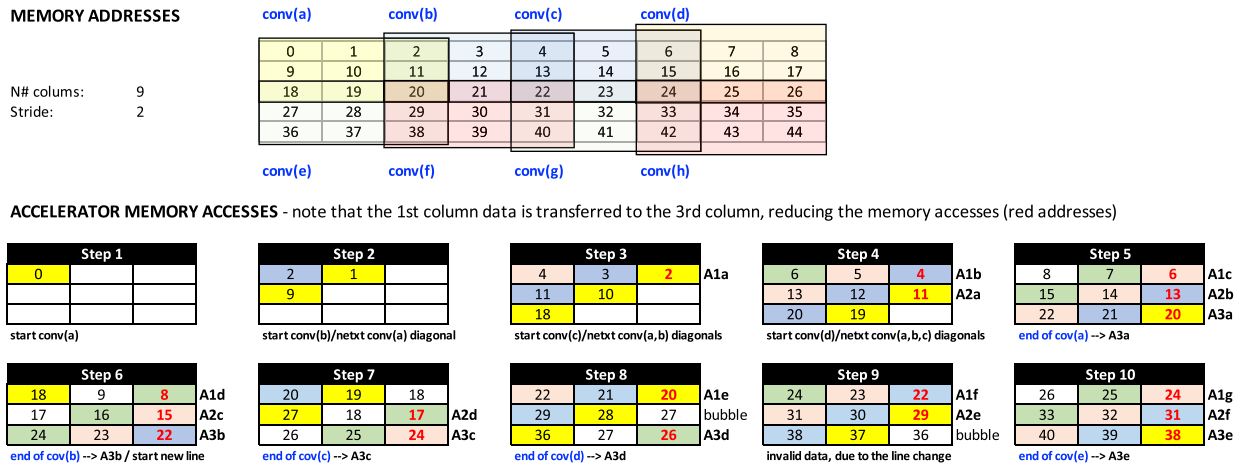


Fig. 4. Convolution 2D - memory accesses and processing flow.

The dataflow type refers to how the data to be processed is mapped in a given accelerator array. The mapping determines how to load and generate the data into the array. The dataflow is characterized by parameters such as latency, throughput, and data reuse.

Weight Stationary is a dataflow where the weight values are stored in internal buffers, and the IFMAP values change through the array. The weights values change when the computation of the output finishes. As an advantage, the partition can be done according to the filters, where each PE is a pair of input feature value and weight value. As a disadvantage, it is necessary to store the matrix filters, which can be expensive regarding RFs [32].

Output Stationary is a dataflow where the partial outputs are stored in internal buffers, and the weight values change through the array. The partial values are transferred to the neighbor PE to be reused. This approach improves the parallelism if the array size is big enough to support more than one iteration at the same time.

Input Stationary has the same approach that Output Stationary. However, instead of partial outputs, the IFMAP values are stored in internal buffers. The weight values still change through the array.

Row Stationary uses all types of reuses to process the output feature map. It stores the weight values, IFMAPs and partial outputs in internal buffers. Each PE processes a row of the input feature map and the filter. Filter rows are horizontally stored in the PE through the columns. The rows of the input feature map are stored in the diagonal. The partial outputs are computed vertically, making each column generate an output.

No Local Reuse has no storage in internal buffers for weights, values, or partial output values. All the data can be transferred through the array in a pipeline or broadcast approach.

Finally, External Memory regards the off-chip memory, which are memories that use DDR connectors, such as DDR3 and DDR5, or High Bandwidth Memory (HBM).

We propose two accelerator architectures to validate this work. The first one is a systolic 2D accelerator, with two relevant features: memory accesses reduction with high sustained throughput. The second one is a 1D array accelerator, with the goal to reduce area and power consumption at the cost of reduced performance. Both accelerators adopt weight stationary dataflow, described in VHDL RTL, validated using the CIFAR10 dataset.

A. Systolic 2D Accelerator

Systolic accelerators presented in the literature [34], [38] have the data load through the matrix left and upper borders, with the transfer and execution inside the matrix in a pipeline fashion. This data load feature increases the data generation latency and reduces throughput.

The proposed accelerator differs from these architectures making a parallel data loading and reusing data already read according to the stride size. These features lead to the following advantages: (*i*) reduces memory accesses; (*ii*) sustained high throughput.

With 3×3 weight filters and stride equal to 2, there is a 33% memory reading reduction (6 readings instead 9) for each convolution. According to [37], memory access can spend 100 times more energy compared to the accelerator array in the convolution stage. The throughput is guaranteed by parallelizing memory reading with computation. The proposal accelerator requires 7 clock cycles for data reading and buffering. Thus, there is valid output data at every 7 clock cycles, with a 7 clock cycles bubble at the end of each line.

Before introducing the accelerator architecture, we present the data flow for executing the convolution. Figure 4 shows the memory addresses related to the IFMAP data at the top, considering an IFMAP with 9 columns and stride equal to 2. Thus, there are 4 convolutions per line, marked as *conv(a)* to *conv(d)*. The next convolutions correspond to *conv(e)* to *conv(h)*. The weight values reading occurs before the convolution starting, characterizing this approach as a weight stationary.

The bottom part of the figure has 10 *steps*. Each step corresponds to the memory reading, and in parallel, the arithmetic operations execution. We use steps 1 to 5 to illustrate a single convolution. At the end of the 3th step, the computation of the first line (addresses 0/1/2) is stored in a register (*A1a*). At the end of the 4th step, the result of the second line (addresses

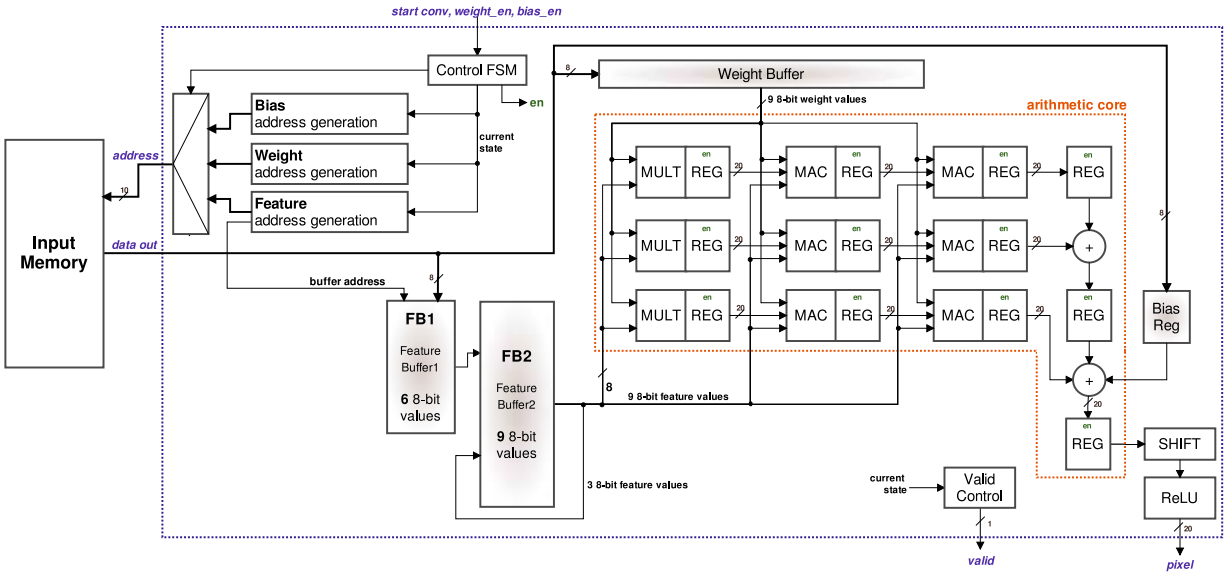


Fig. 5. Systolic 2D array accelerator architecture.

9/10/11) is added to *A1a*, stored in a register (*A2a*). At the end of the 5th step, the result of the third line (addresses 18/19/20) is added to *A2a*, stored in a register (*A3a*). The value of this register corresponds to the output of the first convolution.

All steps make 6 memory accesses. The first four steps read values that are not used, corresponding to the filling of the matrix. Take, for example, step 5. In this step, 6 values are read from memory, corresponding to the addresses of the first 2 columns (8/7/15/14/22/21). The third column is filled with data from the first column (once stride is equal to 2, allowing to reuse these values), thus reducing memory access (red numbers). Once the matrix is filled, 5 convolutions are processed simultaneously, one on each matrix diagonal.

Note that there is a bubble step between lines (at step 9). This is due to the load of the value in the last row column. The reading process ends in the last column that generates a valid convolution. This management is necessary because the IFMAPs may have an extra column due to the absence of padding.

Equation 1 computes the number of clock cycles required to compute the convolution for a given IFMAP.

$$\begin{aligned} \#convX &= \left| \frac{N - W_x}{S_x} + 1 \right| \\ \#convY &= \left| \frac{M - W_y}{S_y} + 1 \right| \\ conv\_cycles &= (F + \#convX * \#convY + \#convX) \times C \end{aligned} \quad (1)$$

where: $N \times M$ – IFMAP size; $W_x \times W_y$ – weight filter size; $S_x \times S_y$ – stride size; F – *steps* to fill the accelerator (5 in this accelerator); C number of clock cycles to execute one *step* (7 in this accelerator).

Thus, a 128 × 128 IFMAP, 3 × 3 weight matrix, and stride equal to 2 require 28,259 clock cycles to compute a convolution, which corresponds to 28.2us@1GHz, including IFMAP reading and arithmetic processing.

Systolic 2D Architecture

Figure 5 illustrates the accelerator architecture, with its external interfaces and the input memory connection, which stores the bias value, weights, and IFMAP. This memory is assumed pre-loaded before the convolution process, delivering 1 byte per clock cycle (8 Gbps@1GHz). The arithmetic core contains a 3×3 matrix with 3 multipliers, 6 MACs, 3 adders, and 12 registers. The accelerator presents a double buffer approach for the feature reading (FB1/FB2), making it possible to read the memory values and execute the arithmetic process in parallel.

The initialization process occurs by loading the weight values (`weight_en`) and the bias value (`bias_en`) in the `weight_buffer` and `bias_reg` buffers. Next, the activation of the `start_conv` signal starts the convolution process, according to Figure 4.

The convolution execution follows a loop controlled by the "control FSM", until completing the IFMAP reading:

- cycle 0: transfer of the 6 values read from memory from FB1 to FB2, reuse 3 values from FB2, update FB1 addresses. Given the combinational implementations of the arithmetic blocks (multipliers and adders) in the "arithmetic core", these start computing new values at the end of this cycle.
- cycles 1 to 6: read the IFMAP values from the input memory to FB1, according to the sequence shown in Figure 4.
- cycle 5: at the end of the fifth cycle, the "control FSM" activates signal `en` for all arithmetic core registers, generating a new output value. This value goes through two combinational blocks, SHIFT and ReLU.
- cycle 6: the "valid control" block activates the `valid` signal according to the convolution being executed. This block controls the bubbles at the end of the lines, as described in Figure 4.

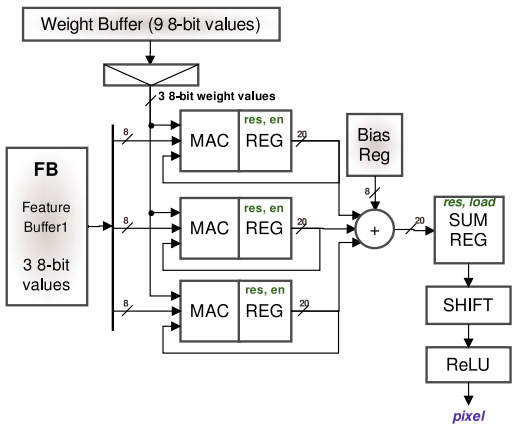


Fig. 6. 1D array accelerator architecture (buffers and arithmetic core).

After convolution, the accelerator executes the activation function. For the two accelerators, we adopted the ReLU, but other nonlinear functions can be supported, like LeakyReLU and PReLU [39]. The memory of the next convolution layer receives the `pixel` output. We recommend an approach similar to FB1/FB2, with two "input memories", to parallelize convolutions. The required throughput of the input memory is achieved with static memories or DDR5 memories. Thus, it is possible to use several memory and accelerator sets in parallel to maximize the CNN performance.

B. 1D Array Accelerator

The second implementation, 1D array, has a straightforward architecture to reduce area and consumption. Figure 6 illustrates the buffers and the arithmetic core of the 1D array. The initialization process is the same as for the 2D architecture, with the weights and bias load. The process of generating a valid output comprises a loop repeated three times, requiring six clock cycles at each interaction. The reading of three IFMAP values occurs in the first three clock cycles. In the subsequent three clock cycles, MACs compute new values. At the end of the sixth cycle, registers store values generated by each MAC (signal `en`). At the end of 3-column processing, the MAC registers are reset (signal `res`), and the resulting addition is stored in SUM REG by activating the `load` signal. Note that the throughput is constant, without the generation of bubbles at the end of each line.

Equation 2 computes the number of clock cycles required to compute the convolution for a given IFMAP using the 1D array accelerator.

$$conv\_cycles = (\#convX * \#convY) \times C \times 3 \qquad (2)$$

where: $\#convX$ and $\#convY$ defined in Equation 1; C number of clock cycles to execute one integration (6 in this accelerator).

Thus, a 128 × 128 IFMAP, 3 × 3 weight matrix, and stride equal to 2 require 71,442 clock cycles to compute a convolution, which corresponds to 71.441us@1GHz, including IFMAP reading and arithmetic processing. The systolic 2D, for the same IFMAP size, computes the convolution 2.3 times faster than the 1D array.

TABLE II
COMPARISON BETWEEN TENSORFLOW ACCURACY AND HARDWARE ACCURACY (URSA) FOR THE MNIST DATASET

TensorFlow		URSA	
# Conv. Layers	Accuracy (%)	# Captured Conv. Operations	Accuracy With Quantization (%)
2	95	62,800	90
3	95	174,000	92
4	96	375,600	93

V. RESULTS

This section contains 4 Subsections. Section V-A evaluates the error introduced by the 8-bit integer quantization. Section V-B details the PPA results for both accelerators obtained from industrial EDA tools. Section V-C evaluates the advantage of using URSA, i.e., simulation time, and its accuracy to estimate power and energy. Finally, Section V-D uses previous results to execute a design space exploration varying the accelerator architecture and parallelism degree using the PPA results.

A. Accuracy Results

The first set of results presented in this section corresponds to the evaluation of the impact in the CNN accuracy of the results due to the 8-bit integer quantization, described in Section III-A. These results are related to the CNN architecture and not to the hardware accelerator.

Three CNNs were generated by TensorFlow using convolution operations, varying the network depth from 2 to 4 layers, with 4, 12, and 38 filters, respectively. All three CNNs were trained using the MNIST dataset, with 3×3 filters with strides between 1×1 and 2×2, ReLU as activation function, and a fully-connected layer with a softmax activation function. TensorFlow executed the training step for five epochs. URSA executes the fully-connected layer, as it is not accelerated in hardware. The reason to adopt the MNIST dataset is the possibility of implementing CNNs that achieve a high accuracy only using convolution layers.

Table II presents the accuracy results. The 1th and 2th columns are TensorFlow parameters: number of convolution layers and CNN accuracy. The 3th and 4th columns are captured from URSA, corresponding to the number of executed convolutions and the accuracy using quantization.

Results show that the 8-bit integer quantization introduces a decrease in the CNN accuracy compared to float-point values. However, this decrease is 5% or less and is expected when applied quantization techniques [40].

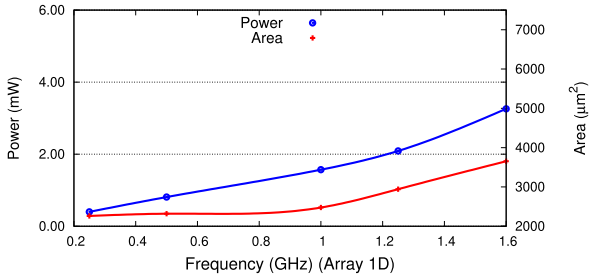
B. Synthesis Results

This section presents the PPA results of the accelerators after physical synthesis. As depicted in Figure 1, the system simulator uses these results to evaluate different CNN architectures. Table III shows the results varying the accelerator architecture (2D/1D) and the technology node (65nm/28nm). For both technologies, the frequency is obtained with a slack time equal to or near zero.

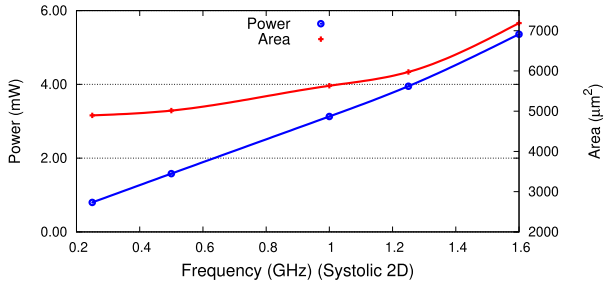
TABLE III

PPA RESULTS FOR ACCELERATORS AFTER PHYSICAL SYNTHESIS (65nm@1GHz, 28nm@1.6GHz)

Accelerator	Technology	Area – μm^2	Cell Count	Power – mW
Systolic 2D	65nm	17,478.24	3,183	7.27
	28nm	7,190.91	5,922	5.36
Array 1D	65nm	8,288.28	1,562	3.69
	28nm	3,654.70	2,964	3.26



(a) 1D array area-power data



(b) systolic 2D area-power data

Fig. 7. Area-power results for 28nm as function of the frequency.

Cadence Genus and Innovus tools were used for logic and physical synthesis. The power dissipation is obtained with a value change dump (VCD) file generated with a post-P&R netlist simulation and Cadence Voltus tool. The netlist simulation input is a $32 \times 32 \times 3$ feature map, 16 3×3 filters, stride 2, generating a $15 \times 15 \times 16$ output. This simulation setup corresponds to the first convolutional layer of the CNN showed in Figure 2, using the CIFAR10 dataset. CIFAR10 dataset was chosen to evaluate power because it presents fewer zero values than the MNIST dataset, increasing the switching activity of the accelerators. That MNIST is a black and white dataset while CIFAR10 is RGB. Thus, power values come from a real dataset and not synthetic values.

Figure 7 presents the physical synthesis results for the 28nm technological node for both accelerators as a function of the frequency (0.25–1.6 GHz). As expected, the power increases with the frequency. Note that the area rises for frequencies higher than 1GHz, due to the synthesis tool effort to meet the target frequency, mainly for the 1D array architecture. URSA uses these points for design space exploration (DSE).

Results presented in Table III and Figure 7 are consistent with the accelerator architectures since the 2D architecture has nine MACs (in fact 6 MACs, 3 adders, 3 multipliers), and the 1D has three MACs in the arithmetic core.

TABLE IV

PPA NORMALIZED RESULTS PER MAC FROM THE LITERATURE FOR CNN ACCELERATORS

Style	Accelerator	Norm. Area (μm^2)	Norm. Power (mW)
1D Array	Hsiao et al. [12] 28 nm	15,548	0.94
	Our Proposal 65 nm	2,763	1.23
	Our Proposal 28 nm	1,218	1.08
2D Systolic	Swallow [38] 65 nm	18,047	3.33
	Our Proposal 65 nm	1,942	0.81
	Our Proposal 28 nm	799	0.60

TABLE V

COMPARISON OF NETLIST VERSUS SYSTEM SIMULATOR (65nm@1GHz, 28nm@1.6GHz)

Accelerator	Technology	Simulation Time – sec		Energy – uJ	
		Netlist	URSA	Netlist	URSA
Systolic 2D	65nm	119.23	1.63	603.49	603.73
	28nm	128.19	1.67	264.45	264.74
Array 1D	65nm	118.87	0.31	720.52	727.75
	28nm	143.83	0.33	373.58	380.24

Normalizing values in Table III by the number of MACs, enables a comparison with related works. Table IV presents such normalization for our proposal and related works. The 1D array values are higher than the 2D array due to the register banks and control FSMs, common to both accelerators.

Table VI presents area and power values reported in the literature. Swallow *et al.* [38] present a 2D accelerator with 256 16-bit MACs, resulting in 18,047 μm^2 and 3,33 mW per MAC. Even considering that the arithmetic blocks (named NPEs in [38]) uses 16-bit MACs and additional modules as DMA, the reported area (9.3×) and power (4.1×) are still significantly higher than the proposed 2D accelerator. Hsiao and Chang [12] present an 1D accelerator for a 28 nm node, resulting in 15,548 μm^2 (12.7× higher) and 0.94 mW (0.83× smaller) per MAC.

Such results corroborate the hypothesis of smaller silicon area and similar power footprint of the proposed accelerators than related works.

C. Energy and Simulation Time Results

The two first results evaluated the accuracy using integer quantization (Section V-A) and the PPA data (Section V-B). This third set of results justifies the system simulator adoption and evaluates the accuracy relative to the energy estimation.

Table V presents the simulation time and the consumed energy, using the experimental setup of the previous Section ($32 \times 32 \times 3$ feature map, 16 3×3 filters). The

TABLE VI
PPA RESULTS FROM THE LITERATURE FOR CNN ACCELERATORS

Style	Accelerator	# MAC / Mult.	Precision (fixed-point)	Tech.	Freq. (GHz)	Area (μm^2)	Cell Count	Power (mW)
1D Array	Hsiao et al. [12]	128	16-bit	28nm	0.2	1,990,110	NA	121
	DianNao [35]	256	16-bit	65nm	0.98	3,023,077	NA	485
2D Systolic	Eyeriss V2 [41]	192	8-bit	65nm	0.2	NA	2,695,000	NA
	Swallow [38]	256	16-bit	65nm	0.8	4,620,240	NA	852

simulation time using URSA is 77 (2D) and 370 (1D) times faster than the netlist simulation (average values). Such results justify adopting the system simulator to execute a fast design space exploration using physical synthesis data. It is worth mentioning that this experiment simulated a small input feature map with a small number of filters. With the increase of the input feature map size and the number of filters, the speed up using URSA compared to the netlist simulation is expected to increase.

The energy values related to the netlist simulation were obtained from the power estimated by industry-standard EDA (Voltus) multiplied by the simulation time, considering the simulation with 16 filters. The energy values related to the URSA simulation consider the pre-characterized power obtained from the previous physical synthesis and the number of clock cycles to execute a convolution (Equations 1 and 2). The energy estimation using URSA introduced an average error compared to the netlist simulation equal to 0.68%, being the worst-case 1.8% (1D, 28nm). This energy estimation error is smaller than, e.g., results presented by Accelergy work [17], which is 5%.

Therefore, the adoption of the URSA system simulator enables faster simulations with an accurate energy estimation, showing that accurate analyses need to regard the entire convolution accelerator, and not only fundamental components, as adders and multipliers.

D. URSA Design Space Exploration Results

This section presents the CNN design space exploration (DSE), using the URSA system simulator, with data obtained from the physical synthesis of accelerators.

URSA explores the design space using *five* parameters:

- Accelerators Architecture: 1D array and Systolic 2D.
- Parallelism: as presented in Table VI, accelerators available in the literature present from 128 to 256 MACs. Our accelerators have 9/3 MACs (2D/1D), making it possible to parallelize these accelerators to process several channels simultaneously. The DSE explores from 1 to 16 accelerators in parallel, ranging from 9/3 to 144/48 MACs (2D/1D).
- Power, area: design parameters obtained from the physical synthesis for different frequencies.
- Performance: execution time to execute one $32 \times 32 \times 3$ convolution, with 3×3 filters, stride 2×2, and 16 channels, according to Equations 1 and 2.

Graphs presented in Figure 8 summarize the obtained results for 40 evaluated scenarios (two accelerators architectures, four parallelism configurations, and five operating frequencies). The graphs present the PPA for each scenario.

From the graphs, it is possible to observe, for example:

- 1D array is, as expected, indicated for smaller area and power when compared to 2D systolic at the same frequency and number of filters, as shown in the scenario highlighted in red in both charts from Figure 8 (16 parallel accelerators@1.6GHz).
- systolic 2D is, as expected, indicated for higher performance when compared to 1D array (also shown in the scenario highlighted in red). Observe that the adoption of 16 accelerators for 2D systolic is only justified at frequencies higher than 1 GHz. For smaller frequencies, eight accelerators deliver similar performance, with smaller area and power.
- consider the 2D architecture, Figure 8(b), for a 6.4 mW power budget (green rectangles). The candidate configurations are 1 acc@1.6GHz, 4 acc@500MHz, and 8 acc@250MHz (*acc* stands for accelerator). The power and performance data are similar for these scenarios, but the area is much smaller using 1 accelerator. This graph allows the user to select the optimum accelerator configuration according to its constraints.
- others points can be observed through these charts. For example, still considering 2D architecture (Figure 8(b)). It is possible to note that it is preferable to use 1 acc@1GHz than 4 acc@250MHz, once it presents similar power and performance, but 4 times smaller area. Similar behavior occurs with 4acc@1GHz compared to 8 acc@500MHz.
- comparing 1D with 2D architectures for a 3.2mW power budget:
  - 1D, 4 acc@500MHz: 9,276 μm^2, and 97.2 ms;
  - 2D, 4 acc@250MHz: 19,58 μm^2, and 83.32 ms.

  In this case, the 1D array is the choice since, despite 15% lower performance (97.2 versus 82.32 ms), it presents 50% smaller area (9,276 versus 19,58 μm^2).

The average energy consumption for the 1D array is 313 μJ up to 1.25GHz, increasing to 380 μJ@1.6GHz. On the other hand, the systolic 2D presents an average energy consumption equal to 261 μJ, regardless the frequency. Thus, the systolic 2D presents a better energy efficiency than the 1D array due to its performance. Such result reveals that one cannot consider only the number of arithmetic cores for decision making since a set of blocks are common to both architectures, as the register files.

These examples show that the framework reached its goal, making DSE using multiple design parameters. Note that the

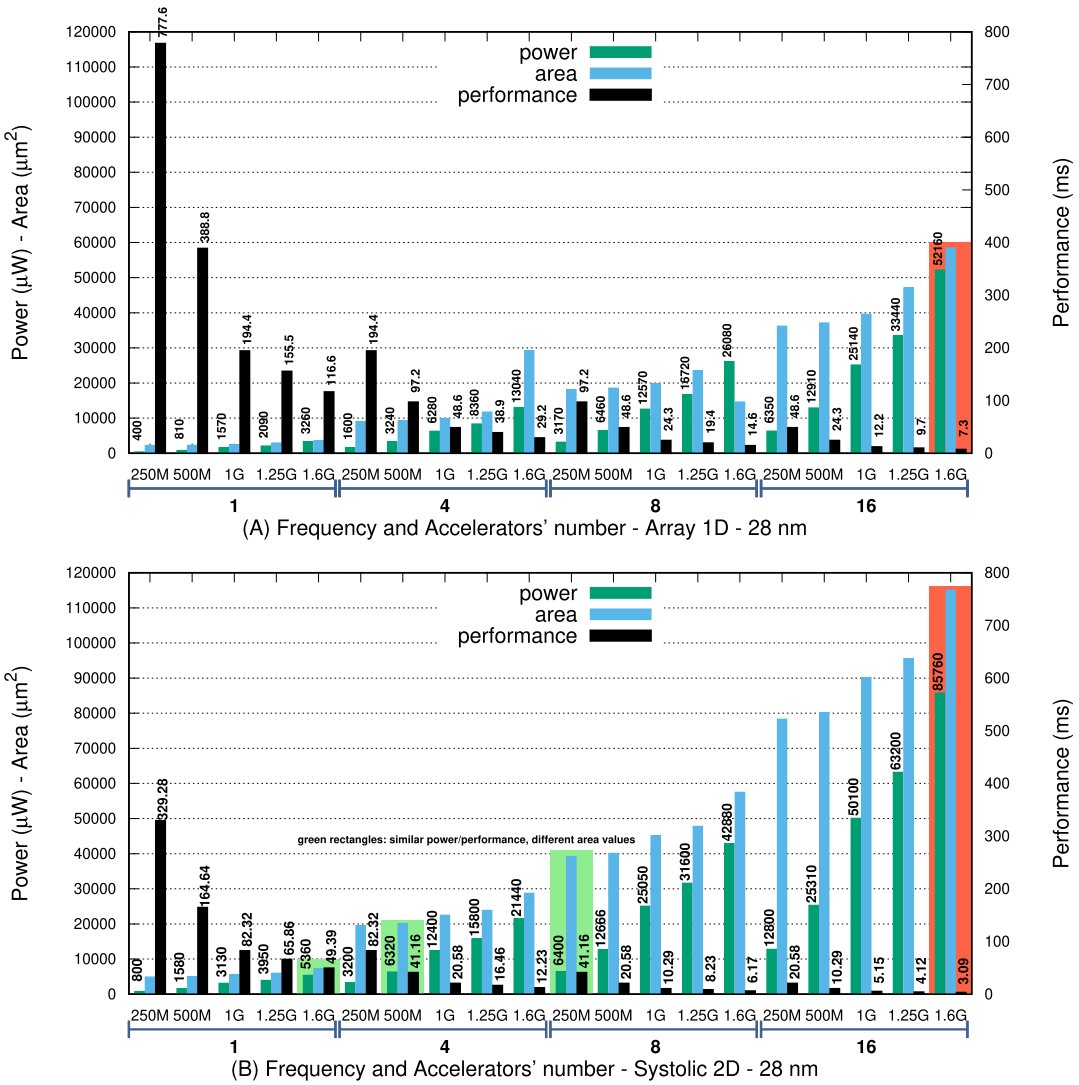


Fig. 8. DSE results obtained with URSA for 28nm for 1D array and systolic 2D (note that power is presented in μW).

values shown in the graphs could be obtained analytically. However, the URSA simulator allows to obtain the presented data and simulate the CNN architecture with a reduced execution time (Table V).

To summarize, the framework enables the evaluation of the CNN accelerator starting from TensorFlow. The framework ensures accurate PPA results regarding a real CNN with real dataset values, such as MNIST and CIFAR10, instead of estimating a constant switching activity for the entire CNN. Thus, the framework allows an evaluation, starting with a high-level CNN modeling, ending with a fast and accurate DSE.

VI. Conclusion

This work proposed a framework to analyze hardware metrics regarding convolution accelerators. The framework allows to: (i) build CNNs with TensorFlow; (ii) extract their weights; (iii) execute the network using a high-level accelerator model in a system simulator; (iv) estimate PPA results and to perform design space exploration. Thus, it is possible to validate hardware models using actual datasets.

This work proposed systolic 2D and 1D array accelerators, detailing their hardware architecture, describing their memory accesses, buffer utilization, and array style. The PPA data considered results obtained after physical synthesis, and switching activity using actual datasets. Using physical synthesis data differentiates our work from others that make estimations based on the CNN architecture. While we consider data from the accelerator that performs the convolution after physical synthesis, many works perform high-level estimation based on the number of MACs.

The framework allows analyzing trade-offs, such as performance-area or area-power consumption. Besides, it is possible to estimate hardware trends, varying the accelerator architecture, the parallelism degree, the technological node, and the operating frequency. It is worth mentioning that the energy estimation error using the system simulator is smaller than 2% compared to the netlist simulation.

At the hardware level, future works include:

- increase the parametrization of the hardware model in terms of array size and word size;
- extend the framework to support other CNN operations, like max-pooling and a fully-connected layer, to explore different CNN configurations;

- model and evaluate other array and dataflow types (such as input stationary and output stationary);
- extend the PPA evaluation to explore synthesis results regarding different transistors types (like regular and low voltage threshold – RVT and LVT), and voltage levels;
- compare the proposed energy analysis method with methods that used basic components, a common technique used in the literature.

At the front-end level (currently TensorFlow), future works include:

- migrate functions currently assigned to the system simulator to frameworks as TensorFlow, including the PPA analysis in these frameworks;
- explore the accelerators parameterization to make a multi-dimensional design space exploration, using as main performance goals power and energy;
- add in the front-end level the cost related to the external memory, considering the required bandwidth to sustain accelerators working at maximum speed, without bubbles.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[2] Google. (2020). *Google Assistant, Your Own Personal Google*. [Online]. Available: https://assistant.google.com

[3] Tesla. (2020). *Autopilot*. [Online]. Available: https://www.tesla.com

[4] S. S. Haykin, *Neural Networks and Learning Machines*. London, U.K.: Pearson, 2009.

[5] (2020). *Caffe*. [Online]. Available: https://caffe.berkeleyvision.org/

[6] (2020). *PyTorch*. [Online]. Available: https://pytorch.org/

[7] (2020). *TensorFlow*. [Online]. Available: https://www.tensorflow.org/

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.

[9] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[10] S.-F. Hsiao, K.-C. Chen, C.-C. Lin, H.-J. Chang, and B.-C. Tsai, "Design of a sparsity-aware reconfigurable deep learning accelerator supporting various types of operations," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 3, pp. 376–387, Sep. 2020.

[11] F. Spagnolo, S. Perri, F. Frustaci, and P. Corsonello, "Reconfigurable convolution architecture for heterogeneous systems-on-chip," in *Proc. 9th Medit. Conf. Embedded Comput. (MECO)*, Jun. 2020, pp. 1–5.

[12] S.-F. Hsiao and H.-J. Chang, "Sparsity-aware deep learning accelerator design supporting CNN and LSTM operations," in *Proc. Int. Symp. Circuits Syst. (ISCAS)*, 2020, pp. 1–4.

[13] D. Giri, K.-L. Chiu, G. Di Guglielmo, P. Mantovani, and L. P. Carloni, "ESP4ML: Platform-based design of systems-on-chip for embedded machine learning," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 1049–1054.

[14] R. Venkatesan *et al.*, "MAGNet: A modular accelerator generator for neural networks," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.

[15] A. Parashar *et al.*, "Timeloop: A systematic approach to DNN accelerator evaluation," in *Proc. Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, 2019, pp. 304–315.

[16] F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio, and T. Krishna, "STONNE: A detailed architectural simulator for flexible neural network accelerators," 2020, *arXiv:2006.07137*. [Online]. Available: http://arxiv.org/abs/2006.07137

[17] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *Proc. Int. Conf. Comput.-Aided Design, (ICCAD)*, 2019, pp. 1–8.

[18] L. R. Juracy, M. T. Moreira, A. M. Amory, and F. G. Moraes, "A TensorFlow and system simulator integration approach to estimate hardware metrics of convolution accelerators," in *Proc. IEEE 12th Latin Amer. Symp. Circuits Syst. (LASCAS)*, Feb. 2021, pp. 217–230.

[19] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 754–768.

[20] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim," in *Proc. Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, 2020, pp. 58–68.

[21] S. Cao, W. Deng, Z. Bao, C. Xue, S. Xu, and S. Zhang, "SimuNN: A pre-RTL inference, simulation and evaluation framework for neural networks," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 10, no. 2, pp. 217–230, Jun. 2020.

[22] S. Kim *et al.*, "Transaction-level model simulator for communication-limited accelerators," 2020, *arXiv:2007.14897*. [Online]. Available: http://arxiv.org/abs/2007.14897

[23] C. Heidorn, F. Hannig, and J. Teich, "Design space exploration for layer-parallel execution of convolutional neural networks on CGRAs," in *Proc. Softw. Compil. Embedded Syst. (SCOPES)*, 2020, pp. 26–31.

[24] Y. Zhao, C. Li, Y. Wang, P. Xu, Y. Zhang, and Y. Lin, "DNN-chip predictor: An analytical performance predictor for DNN accelerators with various dataflows and hardware architectures," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 1593–1597.

[25] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings," *IEEE Micro*, vol. 40, no. 3, pp. 20–29, May 2020.

[26] (2019). *NVDLA*. [Online]. Available: http://nvdla.org/

[27] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," *ACM Archit. Support Program. Lang. Oper. Syst.*, vol. 53, pp. 461–475, Mar. 2018.

[28] C. Hao *et al.*, "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. Design Automat. Conf. (DAC)*, 2019, pp. 1–6.

[29] XILINX. (2021). *Vitis AI*. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html

[30] NVIDIA. (2021). *TensorRT*. [Online]. Available: https://developer.nvidia.com/tensorrt

[31] A. R. P. Domingues, "ORCA: A self-adaptive, multiprocessor system-on-chip platform," M.S. thesis, Dept. Comput. Sci., PUCRS, Porto Alegre, Brazil, 2020. [Online]. Available: http://tede2.pucrs.br/tede2/handle/tede/9344

[32] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Accelerating CNN inference on ASICs: A survey," *J. Syst. Archit.*, vol. 113, Feb. 2021, Art. no. 101887.

[33] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 gops/s mobile coprocessor for deep neural networks," in *Proc. Comput. Vis. Pattern Recognit. (CVPR)*, 2014, pp. 682–687.

[34] S. Das, A. Roy, K. K. Chandrasekharan, A. Deshwal, and S. Lee, "A systolic dataflow based accelerator for CNNs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Oct. 2020, pp. 1–5.

[35] T. Chen *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 269–284, Feb. 2014.

[36] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015, pp. 92–104.

[37] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 367–379, 2016.

[38] B. Liu, X. Chen, Y. Han, and H. Xu, "Swallow: A versatile accelerator for sparse neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4881–4893, Dec. 2020.

[39] Keras. (2021). *PReLU Layer*. [Online]. Available: https://keras.io/api/layers/activations/

[40] J. Cheng, J. Wu, C. Leng, Y. Wang, and Q. Hu, "Quantized CNN: A unified approach to accelerate and compress convolutional networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4730–4743, Oct. 2018.

[41] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss V2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

Leonardo Rezende Juracy received the bachelor's degree in computer engineering and the M.Sc. degree in computer science from the Pontifical Catholic University of Rio Grande do Sul (PUCRS) in 2015 and 2018, respectively, where he is currently pursuing the Ph.D. degree. His research interests include design for testability, fault-tolerant designs, asynchronous designs, resilient designs, and machine learning hardware accelerators.

Matheus Trevisan Moreira received the bachelor's degree in computer engineering and the Ph.D. degree in computer science from the Pontifical Catholic University of Rio Grande do Sul (PUCRS) in 2011 and 2015, respectively. He is currently with Chronos Tech, San Diego, USA. He has experience in different fields of microelectronics with emphasis on non-synchronous circuits design. He has over 80 published articles. His thesis received an award from the Brazilian Society of Microelectronics and CEITEC S.A., as the Best Ph.D. Thesis in Design, EDA, and Test of Integrated Circuits in 2016.

Alexandre de Morais Amory received the Ph.D. degree in computer science from the Federal University of Rio Grande do Sul (UFRGS), Brazil, in 2007. His professional experience include an internship at Philips Research Labs, The Netherlands, in 2005, a Lead Verification Engineer at CEITEC (design house) from 2007 to 2009, and a Post-Doctoral Fellow at the Pontifical Catholic University of Rio Grande do Sul (PUCRS) from 2009 to 2012, where he was also a Professor from 2012 to 2020. He is currently a Research Fellow at Scuola Superiore Sant'anna, Italy. His research interests include design, test, fault-tolerance, and safety-critical systems.

Alexandre F. Hampel is currently pursuing the degree in computer engineering with the Pontifical Catholic University of Rio Grande do Sul (PUCRS). He worked as a Software Developer at Dell from 2018 to 2019. His research interests include NoCs, MPSoCs, and machine learning hardware accelerators.

Fernando Gehm Moraes (Senior Member, IEEE) received the degree in electrical engineering, the M.Sc. degrees from the Federal University of Rio Grande do Sul (UFRGS), Brazil, in 1987 and 1990, respectively, and the Ph.D. degree from the Laboratoire d'Informatique, de Robotique et Microélectronique de Montpellier, France, in 1994. He has been a Full Professor with the Pontifical Catholic University of Rio Grande do Sul (PUCRS) since 2002. He has authored or coauthored 44 peer-refereed journal articles in the field of VLSI design. His primary research interests include microelectronics, security, MPSoCs, NoCs, and hardware accelerators.