Journal of Signal Processing Systems (2021) 93:1365–1385
https://doi.org/10.1007/s11265-021-01687-7

A Reconfigurable Posit Tensor Unit with Variable-Precision Arithmetic and Automatic Data Streaming

Nuno Neves[1] · **Pedro Tomás**[2] · **Nuno Roma**[2]

Received: 25 October 2020 / Revised: 11 April 2021 / Accepted: 26 July 2021 / Published online: 28 November 2021

Abstract
The increased adoption of DNN applications drove the emergence of dedicated tensor computing units to accelerate multi-dimensional matrix multiplication operations. Although they deploy highly efficient computing architectures, they often lack support for more general-purpose application domains. Such a limitation occurs both due to their consolidated computation scheme (restricted to matrix multiplication) and due to their frequent adoption of low-precision/custom floating-point formats (unsuited for general application domains). In contrast, this paper proposes a new Reconfigurable Tensor Unit (RTU) which deploys an array of variable-precision Vector Multiply-Accumulate (VMA) units. Furthermore, each VMA unit leverages the new Posit floating-point format and supports the full range of standardized posit precisions in a single SIMD unit, with variable vector-element width. Moreover, the proposed RTU explores the Posit format features for fused operations, together with spatial and time-multiplexing reconfiguration mechanisms to fuse and combine multiple VMAs to map high-level and complex operations. The proposed RTU is also supported by an automatic data streaming infrastructure and a pipelined data transfer scheme, allowing it to accelerate the computation of most data-parallel patterns commonly present in vectorizable applications. The proposed RTU showed to outperform state-of-the-art tensor and SIMD units present in off-the-shelf platforms, and with dedicated FPGA-based accelerators, in turn resulting in significant energy-efficiency improvements.

Keywords Tensor Computation · Posit Number System · Variable-Precision SIMD · Spatial · Temporal Reconfiguration · Data Stream Computing

1 Introduction

The current computing demands for processing throughput and energy efficiency have been pushing the industrial and academic research focus to domain-specific and reconfigurable architectures [1, 2]. Furthermore, new algorithmic advances allied with the ever-increasing amount of data availability, pushed the computational capacity of off-the-shelf processing platforms to their limit and led to classical design paradigms to be revisited, to cope with the current computing and energy efficiency demands in several application domains. In particular, the growing adoption of Deep Neural Networks (DNNs) drove the research on dedicated hardware to boost the performance of tensor (n-dimensional matrices) multiplication [3–8]. Such calculations are essential to both the training and the inference phases of neural network applications. Accordingly, tensor computing units are usually designed as arrays of Fused Multiply-Accumulate (FMA) elements, supported by dedicated data communication schemes (e.g., data streaming) to maximize throughput.

From the number representation perspective, tensor units are also often based on custom floating-point formats with reduced precision, as an alternative to the IEEE-754 standard. This may not only provide straightforward computing accelerations [7, 9, 10], but also significant reductions in

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under projects UIDB/50021/2020 and PTDC/EEI-HAC/30485/2017.

✉ Nuno Neves
nuno.neves@inesc-id.pt

Pedro Tomás
pedro.tomas@inesc-id.pt

Nuno Roma
nuno.roma@inesc-id.pt

[1] INESC-ID, Instituto de Telecomunicações, Lisbon, Portugal

[2] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal

chip area, allowing to release silicon area for additional computing and storage resources. As a result, less memory storage is required per operand and higher computing bandwidths can be achieved, while reaching lower power and energy consumptions. Hence, major computing market players, such as Intel [10], Google [9], NVIDIA [7], Xilinx [4], and Microsoft [3], have already proposed or adopted such alternative formats in their off-the-shelf platforms and accelerators.

Despite their success in DNN applications, these tensor units present quite consolidated architectures that often constrain their usage on operations other than tensor multiplication. For example, the tensor cores equiping recent NVIDIA Graphics Processing Units (GPUs) [7] are restricted to multiply-accumulate operations, with strict rules on the shape of input tensors [11]. Moreover, tensor units often adopt very-low precision floating-point formats [9], imposing accuracy losses in higher-precision applications, or are limited to the IEEE-754 format [7], hence not supporting lower precision arithmetic.

To that end, the Posit number system [12] has been gaining a growing attention as a possible alternative (or complement) to the IEEE-754 standard, by consistently attaining similar accuracies to IEEE-754, with significant fewer bits [13, 14]. Posits offer an intrinsic trade-off between a wider dynamic range and an increased decimal precision, effectively allowing a higher decimal accuracy, while lowering the operand precision. Additionally, the posit format is particularly suited for fused operations (such as multiply-accumulate), since it avoids overflow and accuracy losses by *i)* adopting an exact accumulator structure (named *quire*) and *ii)* not requiring re-normalization of intermediate results [15].

From the hardware perspective, while some existing tensor-based accelerators present reconfiguration capabilities [16], they are mostly used to adapt the accelerator to the shape of the network. However, there is also an opportunity to further exploit the resources of a tensor unit, in order to deploy higher-level and more complex operations. This can be done by introducing spatial and time-multiplexing reconfiguration mechanisms at the level of the tensor unit, as it is typically deployed in Coarse-Grain Reconfigurable Architectures (CGRAs) and Field-Programmable Gate Arrays (FPGAs) at an accelerator level [17–19]. These mechanisms would allow the reconfiguration of the tensor unit to combine multiple FMA blocks and map operations with diverse complexity, by switching between several configurations to accommodate the deployment of multiple execution phases in a single hardware structure.

Although such a solution broadens the range of supported application domains, the increase in operation complexity does not change the data-parallel and control-free nature of the supported computations (mostly still matrix-based). As such, data streaming [16, 18–20] remains the most suited approach to support the data communication infrastructure of a reconfigurable tensor unit. Besides their natural support for spatial computing schemes [18], stream-based execution models allow a complete detachment between data indexation and computation, allowing independent data acquisition. This removes memory address calculation from the critical path, in turn accelerating execution.

Accordingly, this paper proposes a new Reconfigurable Tensor Unit (RTU) architecture that deploys a data-stream computing model in a 2D array of Posit-based Processing Elements (PEs). The proposed RTU introduces the following contributions and features:

- A new Vector Multiply-Accumulate (VMA) unit (included in each PE) that deploys a variable-precision Single-Instruction Multiple-Data (SIMD) computing scheme with a fully vectorized datapath. It also exploits the Posit format to increase data and computing throughput, by lowering the width of the vector elements whenever possible.
- A combined spatial and time-multiplexing reconfiguration mechanism, allowing each PE to instantly reconfigure to switch between different vector precisions and interconnection schemes with neighbour PEs. Additionally, a novel PE fusing technique is proposed, leveraging Posit fused operations to combine multiple VMAs and deploy more complex operations.
- An efficient data streaming infrastructure, capable of autonomously generating the most common data patterns susceptible to streaming. It is also combined with a banked memory organization, maximizing the exploitation of data-locality and data reutilization.

The proposed RTU was fully implemented in RTL and synthesized with a 45*nm* technology. The obtained results show that the combination of the RTU data streaming and the reconfigurable execution models, paired with its Posit-enabled variable-precision SIMD capabilities, allow to efficiently execute a broader range of applications than those supported by standard tensor units. In particular, it was capable of attaining SIMD speedups up to 346x, 14x, and 31x when compared with SIMD/tensor units equiping an ARM Cortex-A9, an Intel i7-8700K and an NVIDIA GV100, respectively. The RTU is also capable of outperforming specialized FPGA-based accelerators, deployed on a Xilinx ZC702 FPGA device, by as much as 372x. The obtained performance gains also resulted in significant energy efficiency improvements over all platforms.

The remainder of the manuscript is organized as follows. Section 2 provides some background information regarding the Posit format and discusses the most relevant works in the areas of reconfigurable accelerators and data streaming.

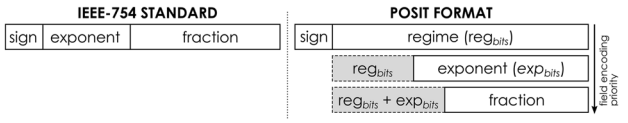

Fig. 1 IEEE-754 standard (left) and Posit (right) formats.

In Sect. 3, it is presented an overview of the proposed RTU architecture and execution model. Sections 4 and 5 describe the architecture of each PE and the data streaming mechanism, respectively. Section 6 presents experimental results and performance comparisons. The manuscript is concluded in Sect. 7, by addressing the main contributions and achievements.

2 Background and Related Work

2.1 Posit Number System and Implementations

The posit number system was proposed in 2017 [12], being the third iteration of *unum*, a numbering format defined to represent real values in computer arithmetic as an alternative to the IEEE-754 standard. The posit format was introduced by relaxing some mathematical properties from the previous iterations and by making its utilization more hardware-friendly.

A posit number [12] is defined as *posit*<*n,es*>, where n is the total number of bits (precision) and es is the maximum exponent size, and it is represented as:

$$\underbrace{\overbrace{s}^{sign}\ \overbrace{r_0\ r_1 \cdots \overline{r_{m+1}}}^{regime}\ \overbrace{e_0\ e_1 \cdots e_{es-1}}^{exponent}\ \overbrace{f_0 f_1 f_2 \cdots}^{fraction}}_{posit(n\ bits)} \tag{1}$$

Similarly to the IEEE-754, the structure of the posit format (also depicted in Fig. 1) includes a *sign* bit field, an *exponent* field, and a *fraction* (or mantissa) field. The posit also comprises a variable-sized *regime* field (with the bit format $rrr...\bar{r}$) that encodes a signed value k. Together with the exponent field, the regime (k) represents the working range of the represented value (or scale factor). The numerical value of k is determined by the run length (m) of 1s or 0s in the regime bits, such that:

$$k = \begin{cases} m-1 & , \ if\ r_0 = 1 \\ -m & , \ otherwise \end{cases} \tag{2}$$

As a result, the exponent and fraction contents are unknown until the regime is decoded (see Fig. 1). In fact, depending on the run length, they can be partly (or fully) left out of the binary encoding. Accordingly, the posit value is given by:

$$(-1)^s \times 2^{exp+k2^{es}} \times 1.fraction \tag{3}$$

The posit format also provides two binary representations that are reserved to encode the zero value (000...0) and Not-a-Real (100...0). The latter comprises all mathematical exceptions. In the posit format, there are no subnormal numbers [12]. Despite the arbitrary format of the precision and exponent parameters of a posit, there are 4 standardized configurations ($n = 64/32/16/8$, $es = 3/2/1/0$) that correspond to the most commonly adopted precisions and dynamic ranges used in IEEE-754 floating-point arithmetic [15].

Finally, for fused operations (such as fused multiply-accumulate), the posit format makes use of a quire [15]. This quire is a fixed-point 2's complement value, of length $n^2/2$, with enough precision to avoid overflow and cancellations. Accordingly, for the standard 64/32/16/8-bit posit precisions, the quire maintains a length of 2048/512/128/32 bits.

Some hardware implementations have already been proposed to support the adoption of the posit format. Jaiswal et al. [21] proposed one of the first parameterized algorithmic computational flows for posit addition/subtraction arithmetic and modeled its architecture implementation. Forget et al. [22] introduced a template library to implement operators for custom size posits and their associated quire. Following these initial approaches, Chaurasiya et al. [14] proposed a parameterized pre-synthesis posit unit generator for adders and multipliers of any bit-width. They observed that the area and energy consumption of the operators are comparable to their IEEE-754 compliant counterparts, and that they can provide comparable accuracies to the IEEE-754 standard for FIR filter implementations. More recently, Charmichael et al. [13] applied the posit format to DNNs. They proposed the Deep Positron with ≤8-bit posit precisions for the inference phase. By implementing a precision-adaptable FPGA soft-core for the exact multiply-and-accumulate (MAC) operation, they demonstrated that the 8-bit posit precision achieves an accuracy comparable to those obtained with a 32-bit IEEE-754 floating-point implementation. Zhang et al. [23] proposed the first Application-Specific Integrated Circuit (ASIC) implementation of a posit-based accelerator, by introducing a posit MAC unit generator for deep learning applications. They presented a 5-stage pipeline design capable of meeting the speed requirements of modern processors.

2.2 Data Streaming Schemes

Data streaming follows the general principle that regular applications are characterized by complex memory access

patterns that can be represented by an n-dimensional affine function [24]. The memory address (y) is calculated by considering an initial *offset*, and pairs (one per dimension) of increment variables x_k and $stride_k$ multiplication factors:

$$y(X) = offset + \sum_{k=1}^{n} x_k \times stride_k \ , \ x_k \in \{0, \cdots, dim_k\} \quad (4)$$

Such a representation allows indexing a significant amount of regular access patterns. Nonetheless, several approaches have been proposed that rely on dedicated ISAs [18, 25] and descriptor-based mechanisms [20, 26] to represent patterns with higher complexity (by combining multiple functions). As an example, Hussain et al. [26] proposed a 3D regular data-fetching mechanism with support for scatter-gather and tiled accesses. To ease the description of regular data-patterns, the Hotstream framework [25] adopts an assembly-like programmable approach. Similarly, Nowatzki et al. [18] proposed a stream-dataflow ISA capable of generating streams with 2D affine patterns. By taking a step further from the description of low-dimentionality patterns, Neves et al. [20] proposed a dynamic descriptor specification to encode arbitrarily complex (regular) data-patterns.

2.3 Domain-Specific Accelerators

An outstanding emergence of Domain-Specific Architectures (DSAs) as been observed in recent years. In particular, the high computational requirements of DNN applications drove the development of new and sophisticated tensor-based architectures. Being Google's Tensor Processing Unit [9] one of the current flagships in this class of processors, it is solely focused on accelerating DNNs. While still focused on tensor multiplication, NVIDIA's tensor cores [7] provide an slightly increased level of usability through their integration in general-purpose GPUs. Other accelerators have also been proposed to tackle this application domain. As an example, Chen et al. [16] proposed an accelerator capable of reconfiguring itself to support different DNN filter shapes.

To provide a more general-purpose support, new reconfigurable accelerators have also been proposed in recent years [18, 19]. In particular, CGRAs have been deployed by combining spatial and temporal computation schemes to achieve energy-efficient acceleration. They are designed to take advantage of highly parallel computing resources and data transfer channels (spatial computation), combined with time-multiplexing resources to perform multiple operations with different levels of complexity (temporal computation). As an example, Prabhakar et al. [19] proposed the Plasticine, designed to efficiently execute parallel patterns, through a 2D array of reconfigurable units. Nowatzki et al. [18] proposed a stream-dataflow CGRA capable of reconfiguring its datapath and memory streams.

To ease the deployment of such accelerators, there has also been a growing interest in Domain-Specific Languages (DSLs) that allow the mapping of high-level programming structures to reconfigurable hardware accelerators. One example is Spatial [17], which allows the description of hardware in Scala, by using highly optimized and parameterized templates and provides a set of low-level abstractions for control and memory access. This framework uses a design space exploration algorithm to explore large design spaces, including coarse-grain pipelining and parallelization factors, to select optimal design implementations for FPGAs and CGRAs.

3 Proposed Reconfigurable Tensor Unit

The proposed Reconfigurable Tensor Unit (RTU) architecture (depicted in Fig. 2) is composed of a dense processing structure, comprising a 2D array of reconfigurable PEs (described in Sect. 4), each implementing a 64-bit posit Vector Multiply-Accumulate (VMA) unit (see Fig. 3). Its execution model is based on a data streaming operation, supported by autonomous stream generators connected to a banked scratchpad/buffering memory structure (see Sect. 5). The proposed unit is programmed *i)* by providing the sequence of configurations for each individual PE (locally managed by dedicated low-footprint controllers); and *ii)* by defining a memory access pattern descriptor for each data stream generator. Although the definition of pattern descriptions is out of the scope of this paper, this information can be easily obtained by modern compilation tools [27] or derived from existing DSLs, making the proposed RTU suitable for deployment both in CPUs (as a functional unit) or in dedicated accelerators.

3.1 RTU Reconfiguration and Execution Models

The proposed RTU takes a step further from existing tensor units by adopting a combination of data-streaming with spatial and temporal computation mechanisms, deployed by a high-throughput reconfigurable processing architecture.

3.1.1 Stream-Based Computation

The RTU execution model was devised by observing that the most common data patterns and computation schemes present in matrix-based applications are susceptible to data streaming. This is mainly because those applications typically present data-parallel and control-free computing characteristics, allied with compile-time deterministic memory accesses. While the first allow a straightforward exploitation of spatial computing schemes (e.g., vectorization), the latter effectively allows an explicit detachment of memory

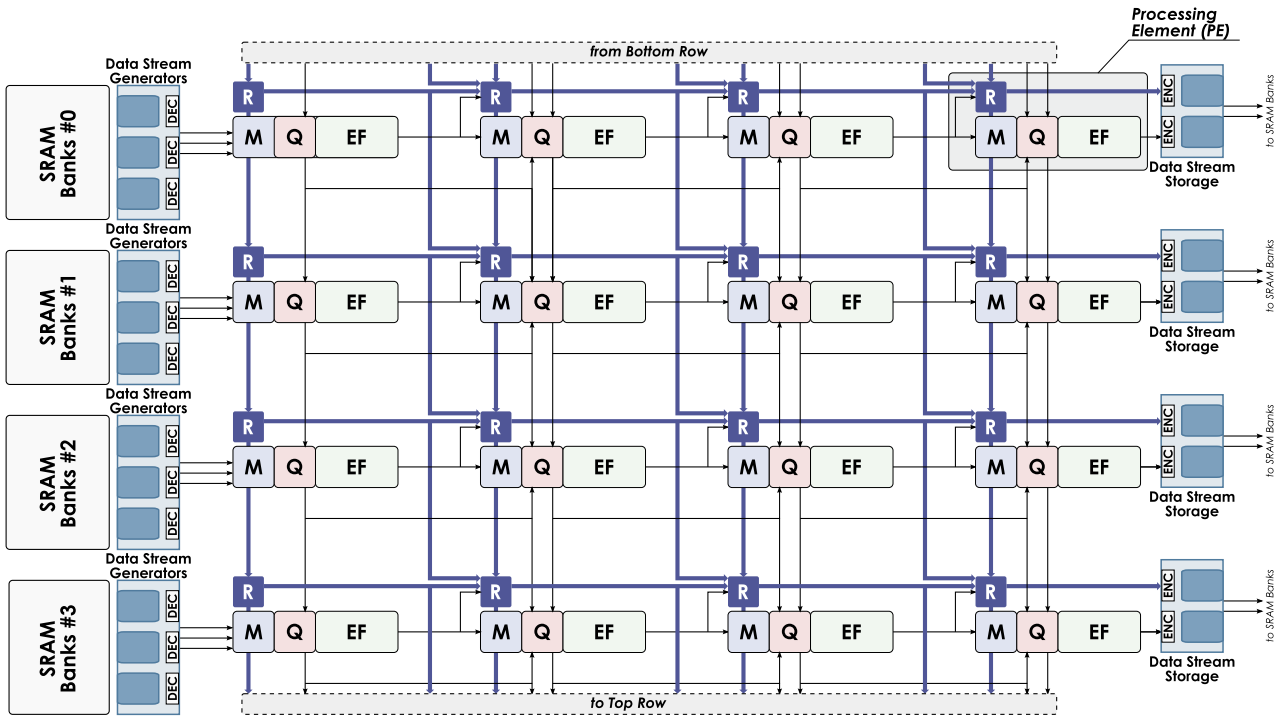

Fig. 2 Proposed RTU block diagram, depicting the PE array and data streaming structures.

address calculation and computation. Such characteristics provide the opportunity to explore a two-fold acceleration, by deploying a stream-based execution model.

3.1.2 Time-Multiplexing Reconfiguration

When mapping computing kernels other than tensor multiplication, it is possible that part of the RTU resources become underutilized and susceptible to be turned off. This can occur when the computing scheme is too computationally intensive and does not present enough data parallelism, or when the required data throughput saturates the available bandwidth. To maximize the resource utilization, the proposed RTU adopts a time-multiplexing reconfiguration scheme. This allows each individual PE to modify its own configuration (at runtime), enabling a simultaneous mapping and switching of multiple operations with different levels of complexity, in distinct areas of the PE array. As an example, a reconfigurable execution scenario is shown in Fig. 4 to obtain the average of a matrix columns.

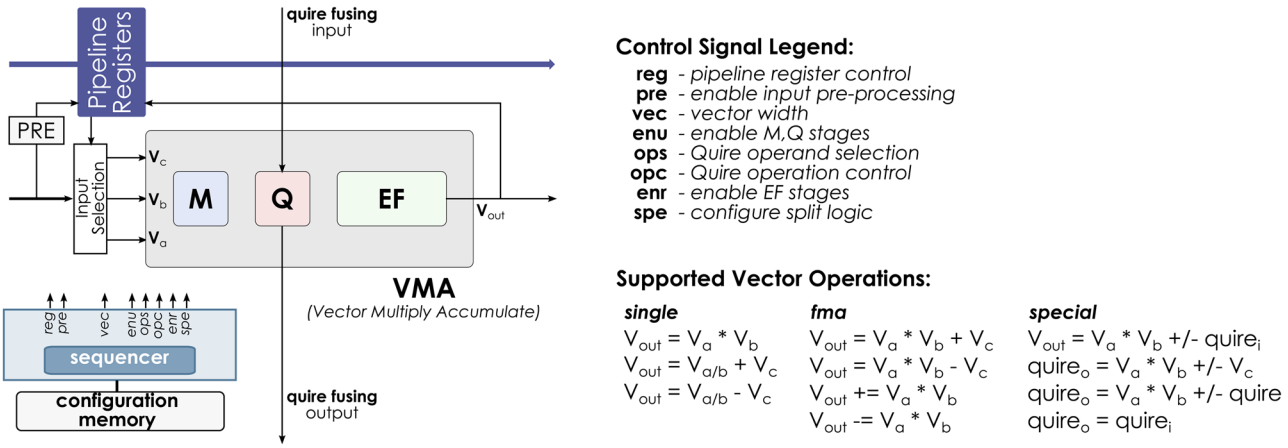

Fig. 3 Overview of the PE module, its components and functionality.

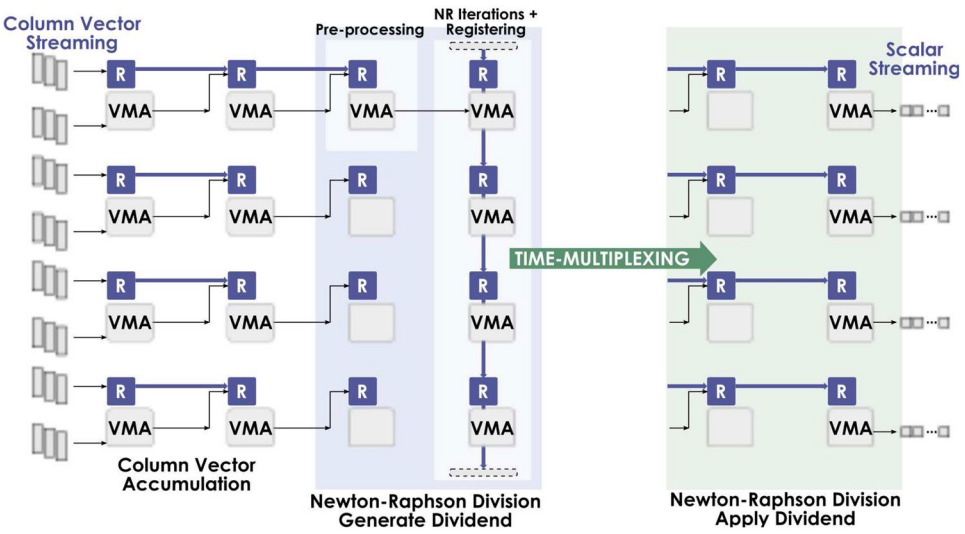

Fig. 4 Example configuration of the RTU illustrating the computation of the column average of a matrix. In the example, the left half of the RTU is accumulating the column vectors, while the right half PE array is initially configured to perform a division pre-computation and it is later reconfigured to calculate the final average result for each column.

3.1.3 Posit-based Fused Arithmetic and SIMD

The proposed RTU adopts the Posit floating-point format in each PE's VMA. This design choice allows the deployment of a floating-point format that supports standardized precisions ranging from 8 to 64 bits, contrasting with the IEEE-754 (which does not support precisions lower than 16 bits) and with custom very-low precisions (that are unsuited for general-purpose computation). Furthermore, since the Posit format does not require intermediate normalization and rounding in fused operations [15], it is possible to fuse multiple VMAs - at the quire level (see Fig. 3) - to map more complex operations (such as reduction trees - illustrated in Fig. 5). However, the main benefit of the Posit format is its capability of attaining similar accuracy with half the precision (or even lower) of the IEEE-754 standard [13, 14, 28]. Although such a scenario is usually dependent on the dataset's dynamic range, it still provides an opportunity to increase the SIMD vectorization, allowing a decrease in the effective memory bandwidth requirement per data element and, in turn, increasing the unit's throughput.

3.2 Data Communication Schemes

To support the proposed reconfiguration and execution models, the RTU's PE array implements a number of data-parallel communication mechanisms, including *i)* data streaming; *ii)* 2D pipelined execution; and *iii)* VMA fusing.

- **Data Streaming:** Data stream acquisition and storage is assured by an autonomous data streaming infrastructure (see Sect. 5), by deploying a dedicated pattern generator for each input operand of the PEs located in the first column of the array, and for each PE output in the last column (see Fig. 2). Each pattern generator leverages a descriptor-based approach to generate the most common data patterns present in regular applications.
- **2D Pipelined Execution:** Intercommunication between adjacent PEs is supported by a 2D pipelined register transfer grid (implemented by the `R` modules in Fig. 2). This is done by placing a pipeline register bank attached to each PE (see Fig. 3 and Sect. 4), allowing data forwarding to three adjacent PEs (right, bottom, and bottom-right), as it is depicted in Fig. 4.
- **VMA Fusing:** By leveraging the Posit fused-operations, each VMA is capable of forwarding its quire to one of the three adjacent PEs. This allows the configuration of more complex fused operations than the $y = a \times b + c$ format, as well as high-level constructs such as parallel reductions (through the mapping of a reduction tree within the PE array - see Fig. 5).

The combination of the pipelined execution and VMA fusing also allows the mapping of several FMA-based arithmetic operations typically deployed in Digital Signal Processors (DSPs), such as non-restoring division and square-root (see Sect. 4). This is done by including a small pre-processing module (`PRE` - see Fig. 3) at the input of each VMA (see Sect. 4).

Finally, all internal communication schemes are defined through the individual configuration of each PE. In particular, data is flown across the array by adopting a configuration in each data-recipient PE that selects the correct data input.

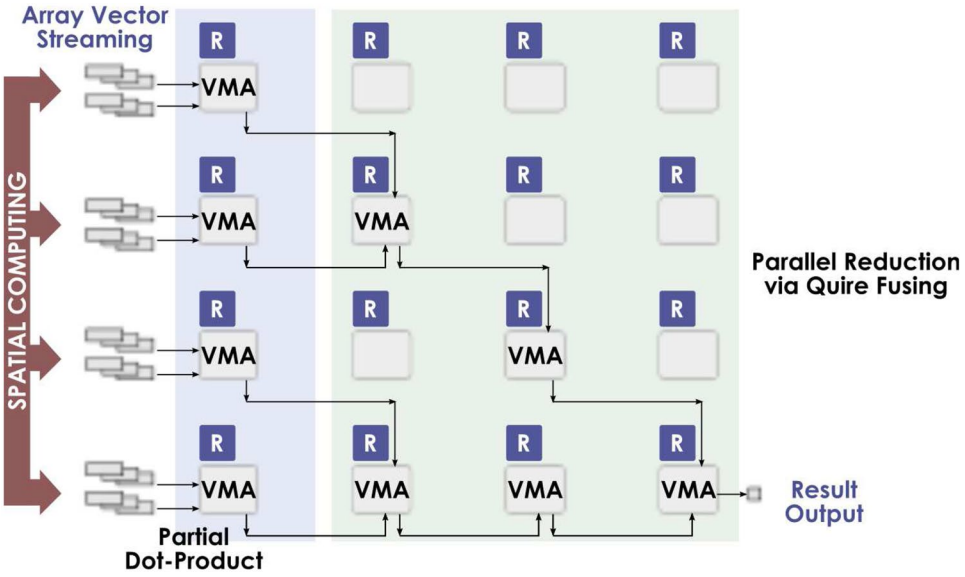

Fig. 5 Example configuration of the RTU illustrating a dot-product operation with a parallel reduction. The example shows the reduction tree implemented via VMA fusing.

4 Variable-Precision Processing Element

As it was referred before, each PE of the RTU is composed of a variable-precision Vector Multiply-Accumulate (VMA) unit (see Figs. 2 and 6). Its architecture comprises a pipelined 64-bit posit SIMD datapath, supporting vector arithmetic for 1×64, 2×32, 4×16, and 8×8-bit posit vectors. This is achieved by reconfiguring the datapath, allowing it to support different vector configurations using the same hardware resources as it would need for a 64-bit scalar operation. Each PE is also paired with *i)* a set of pipeline registers, to support the RTU's pipelined execution scheme; and *ii)* a dedicated controller module that manages the configuration of the PE.

To reduce the hardware footprint and latency of each PE, it was decided to move the required posit decoding and encoding logic to the data streaming infrastructure of the RTU (see Sect. 5), due to their high hardware complexity. As such, each streamed posit operand is decoded before entering the PE array and the output results are encoded only after leaving the array. Accordingly, each PE accepts and outputs data already decoded in sign, exponent and fraction vectors. As a side benefit, it allows operating over other floating-point storage formats, simply by changing the decoding and encoding logic on the streaming infrastructure. The following sections detail each PE component.

4.1 Variable-Precision VMA unit

The VMA architecture (depicted in Fig. 6) implements a 4-stage pipeline FMA compute unit with three input vector operands ($\mathbf{V}_a$,$\mathbf{V}_b$ and $\mathbf{V}_c$). It is composed of the following modules: *i)* 1-stage *floating-point multiply* (**M**); *ii)* 1-stage *quire arithmetic unit* (**Q**); and *iii)* 2-stage *fraction and exponent extraction* (**EF**). Each unit accepts 3 input decoded posit vectors and outputs 1 result vector. It supports: *i)* common vector addition, subtraction, and multiplication operations; *ii)* fused multiply-add and multiply-accumulate operations; and *iii)* a vector-to-scalar reduction operation. To implement the VMA fusing within the RTU, each unit also accepts and forwards the quire values from/to other VMAs in adjacent PEs.

4.1.1 Vector Data Formats

To support the variable-precision hardware that implements the VMA, 64-bit posit vectors (see Fig. 7A) are decoded (during streaming) into three unified vector formats that gather the posit *sign*, *exponent* and *fraction* components, for each supported vector element precision (see Fig. 7B). Hence, each operand of the VMA corresponds to a 104-bit vector format, comprising an 8-bit sign vector, a 32-bit exponent vector, and a 64-bit fraction vector. The same scheme is used for the quire vector, by adopting a 2048-bit vector format that gathers the quire for each vector precision (see Fig. 7C). In the adopted formats, bits that are unnecessary to represent vector element values are set to '0'.

4.1.2 Floating-Point Multiplier

The first VMA stage (see Fig. 6) performs the multiplication of the $\mathbf{V}_a$ and $\mathbf{V}_b$ vectors (and propagates $\mathbf{V}_c$ to the next stage). To provide the aimed variable-precision functionality, exponent vectors are added with a specialized carry-lookahead adder (described in Fig. 8). This module is capable of breaking its carry-chain (through single-bit multiplexers) to perform the addition of either 1×32-, 2×16-, 4×8-, or 8×4-bit vectors. Similarly, the fraction components are

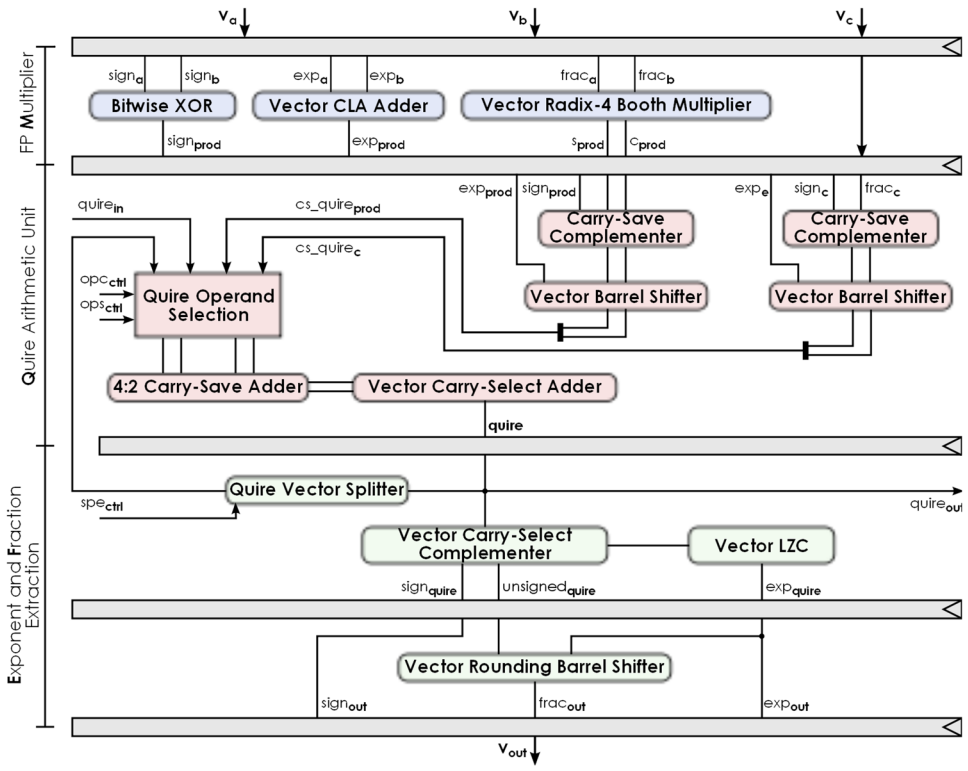

Fig. 6 Vector Multiply-Accumulate (VMA) unit architecture.

multiplied with the aid of a dedicated module implementing a vectorized radix-4 Booth multiplier (see Fig. 9), generating 64 partial products in carry-save format, generated by 64 radix-4 booth encoders. These partial results are gathered and added through a carry-save accumulator tree, resulting in a 128-bit carry-save value. The multiplier is configured to the supported vector configurations by activating and deactivating specific encoders, as depicted in Figs. 9B1-B4. Finally, the resulting sign vector is calculated by performing a bitwise XOR to the input sign vectors. Due to the adopted fraction vector format, all configurations can be multiplied the same way since every element has a padding of at least 2 bits, protecting each element from overflowing to its left neighbour.

4.1.3 Quire Arithmetic Unit

The second stage of the VMA unit (see Fig. 6) implements an arithmetic unit for the quire vector. In the first step, it obtains the two's complement of the fraction vectors computed by the **M** stage and from the $\mathbf{V}_c$ operand. This is done by complementing each vector element and incrementing the value depending on the corresponding sign bit with a carry-save adder. Next, both fraction vectors are converted to the quire fixed-point format, by sign-extending the fraction vector elements and shifting them according to the corresponding exponent value. This is done with a specialized left barrel shifter that performs partial shifts within a 2048-bit word and unifies them by OR'ing the results between

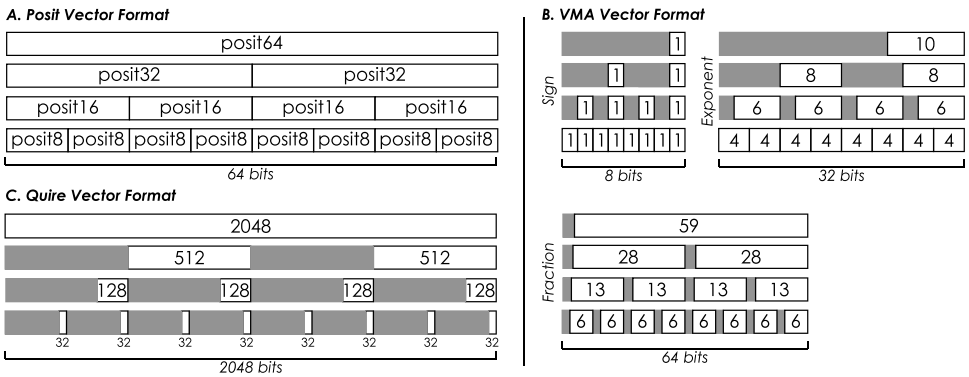

Fig. 7 Vector data formats for (**A**) posit vectors, (**B**) VMA input/output vectors, and (**C**) quire vectors. Grey areas represent unused bits (set to '0').

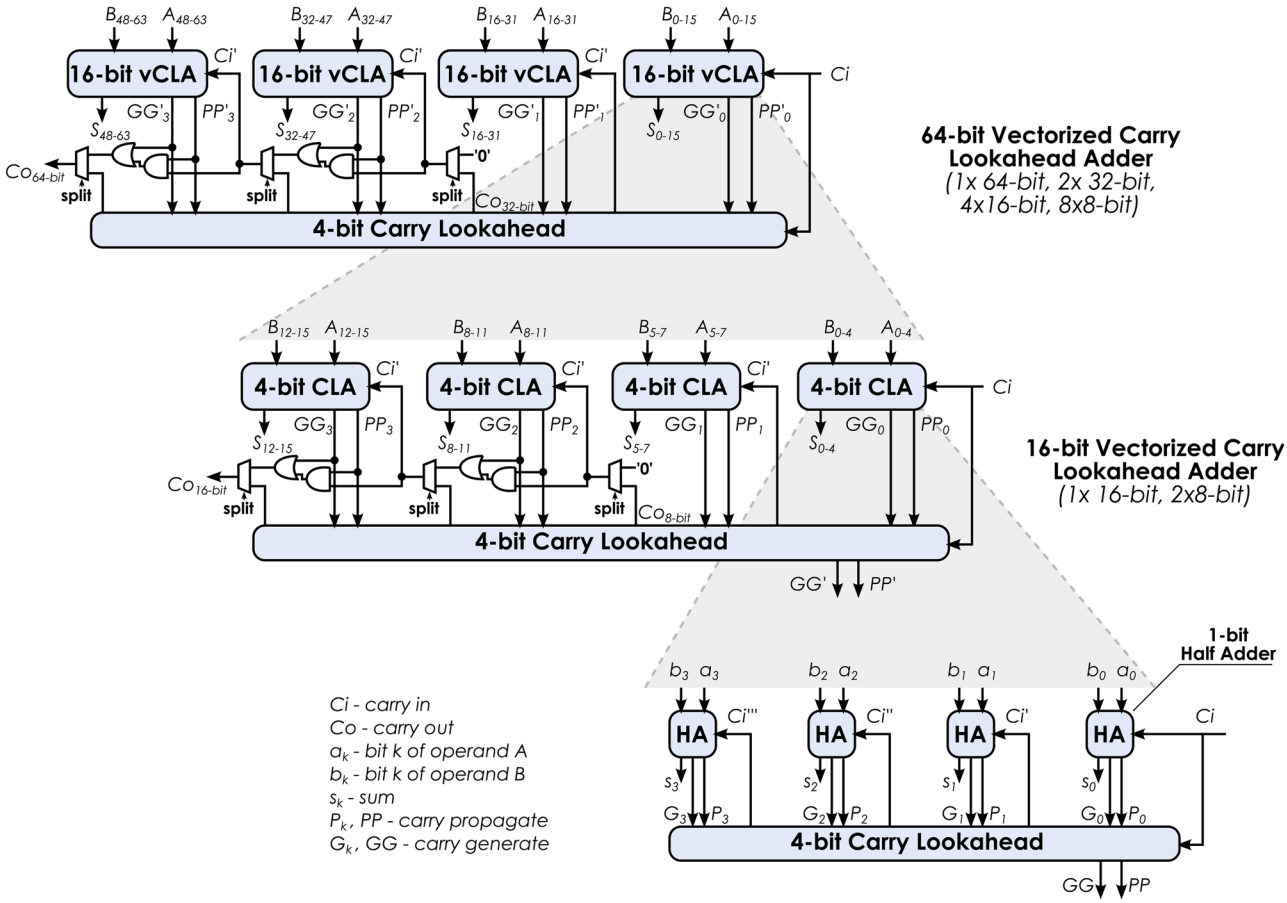

Fig. 8 Vectorized carry-lookahead adder architecture for 64-bit vectors. It is built with 3 levels of 4-bit carry-lookahead structures with a specialized modification allowing each of them to be split in half, by breaking the carry chain (with a multiplixer) and redirecting the carry propagation and generation logic to an alternative 2-bit carry-lookahead structure. The introduced modification allows the 64-bit adder to be instantly reconfigured for each supported vector configuration (i.e., 1x64-bit, 2x32-bit, 4x16-bit, and 8x8-bit).

shifting levels, depending on the considered precision (see Fig. 10A). At this point, two operands for the quire arithmetic unit are selected from *i)* the product quire; *ii)* the $\mathbf{V}_c$ quire; *iii)* a forwarded quire value (from an adjacent PE); or *iv)* a registered quire value (for accumulation). Upon selecting the two operands, they are sent to a 4:2 carry-save adder/subtractor module and the output is accumulated with a chain of 64 32-bit carry-select adders.

4.1.4 Fraction and Exponent Extraction

The final two stages of the VMA (see Fig. 6) are responsible for re-normalizing the quire and extracting the sign, exponent, and fraction vectors. Accordingly, the quire vector is first converted to unsigned (via two's complement with another carry-select module) and the sign vector is obtained. Next, the unsigned quire is sent to a vectorized leading-zero counter, which obtains partial counts for each vector element and generates a final zero-count vector (depicted in Fig. 11). The final stage of the VMA takes the unsigned quire vector and the computed zero-count vector (which corresponds to the exponent vector) and generates a normalized fraction vector with the aid of a vectorized right barrel shifter, that also performs rounding by OR'ing shifted-out bits (see Fig. 10B).

4.1.5 Quire Forwarding and Vector-to-Scalar Reduction

The quire vector values registered in the **Q** stage are also forwarded to adjacent PEs, in order to support the RTU's VMA fusing scheme. Moreover, to support vector-to-scalar reduction operations, the VMA offers an optional module that is capable of splitting a quire vector in half and generating two quire vector values to be fed back to the **Q** stage (see Fig. 6). By successfully performing this operation, it is possible to reduce a vector to a single scalar value (of the same precision).

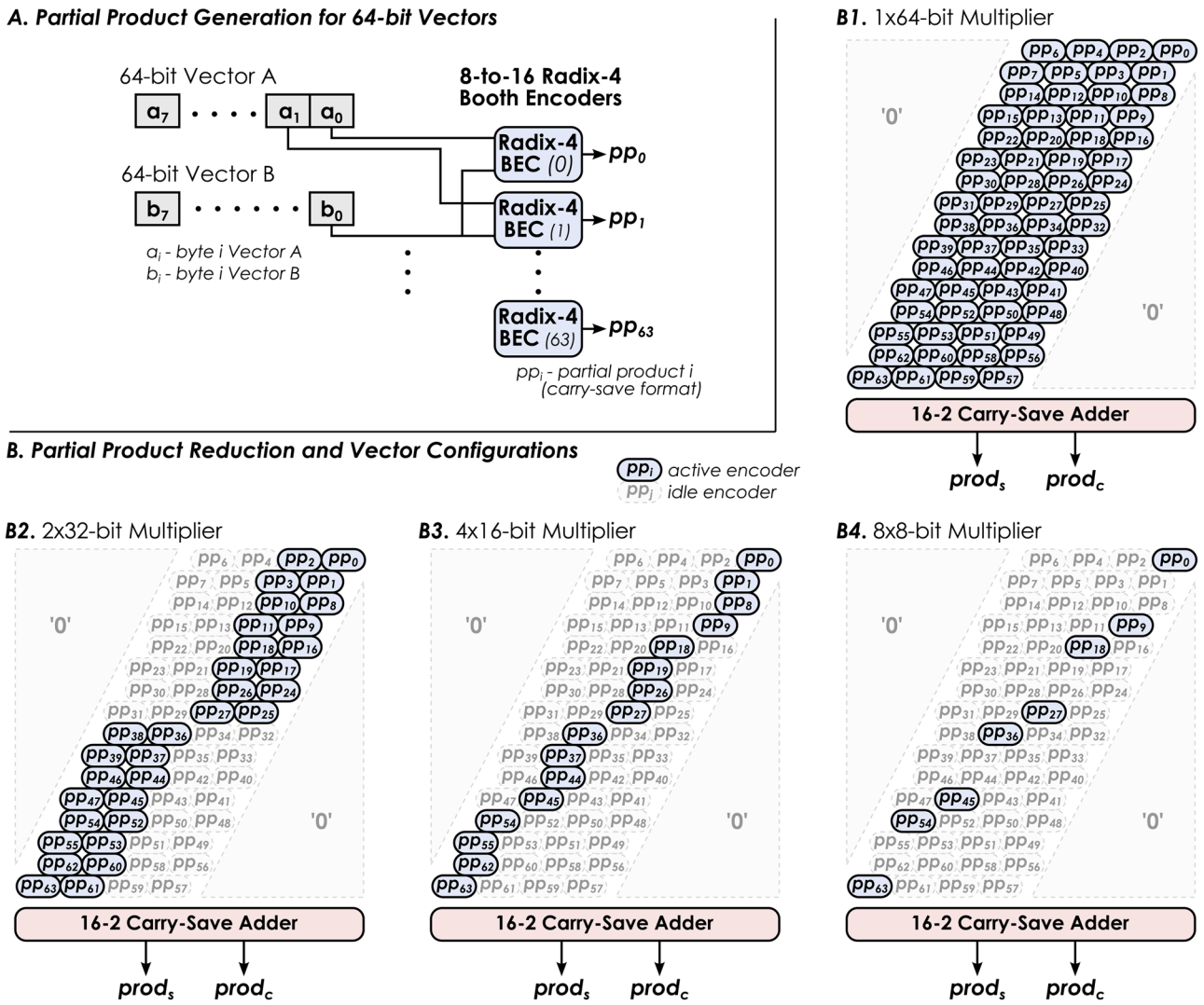

Fig. 9 Vectorized radix-4 Booth multiplier architecture for 64-bit vectors. The block diagram shows (**A**) the partial product generation scheme with radix-4 booth encoders and (**B**) the encoder activations for each supported vector configuration, i.e., 1x64-bit (B1), 2x32-bit (B2), 4x16-bit (B3), and 8x8-bit (B4).

4.1.6 Input Pre-Processing for Non-Restoring Arithmetic

The execution model of the proposed RTU allows the mapping of several FMA-based algorithms typically deployed in DSPs. Some examples of these algorithms comprise the Newton–Raphson and/or Goldschmidt algorithms [29] for non-restoring division (and square-root). These algorithms perform a predefined number of FMA iterations to find the reciprocal of the divisor, and then multiplying it by the dividend [29]. To do so, it is first necessary to scale the divisor to the [0.5; 1] numerical interval and apply the same scaling factor to the dividend. This is done by a dedicated pre-processing module (`PRE` - see Fig. 2B) placed at the input of the PE, to scale the input value and generate the corresponding scaling factor.

4.1.7 Pipeline Registers

Each VMA unit is paired with a local 8×106-bit register file with a dual functionality. These 106-bit registers can be used both for local vector storage (e.g., for intermediate results or constant storage) or as pipeline registers (for data forwarding between adjacent PEs). Dedicated input and output masks are used to select which registers are used to accept input data and which are forwarded to adjacent PEs.

4.2 Configuration Controller

Each PE is managed by a dedicated configuration controller (see Fig. 2B). It deploys a low-profile sequencer module, composed of a counter and a local configuration memory. To

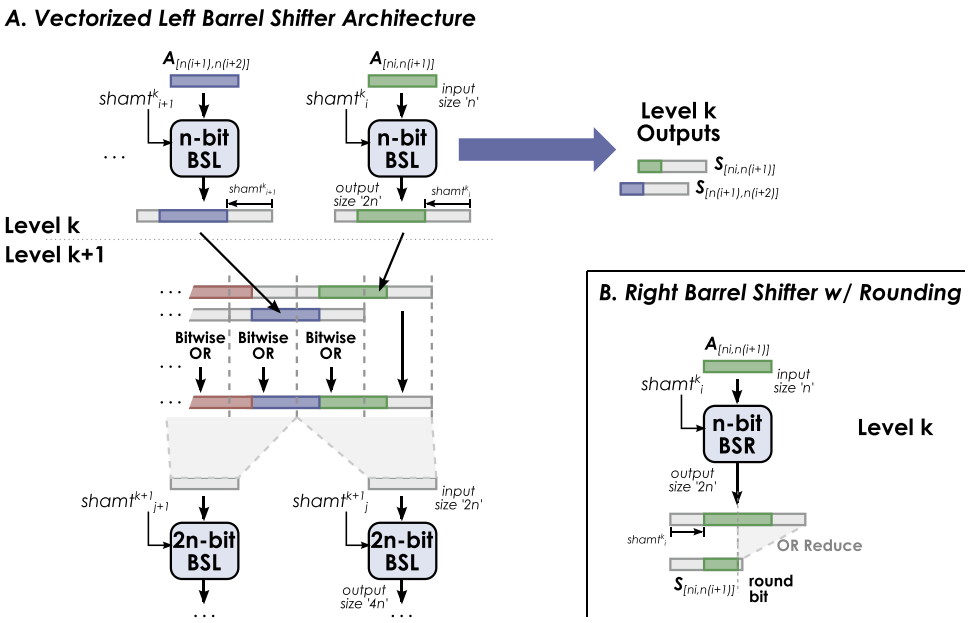

Fig. 10 Overview of the vectorized barrel shifter architecture. The block diagram illustrates the (**A**) multi-level architecture of the left barrel shifter, showing how partial results can be extracted between each level to obtain shifted value for each supported vector configuration. It also demonstrates the unification of partial results via a bitwise OR operation and their propagation to a subsequent level. Finally, subfigure (**B**) shows the required modifications to introduce carry-out rounding on reversed right barrel shifter architecture.

configure the PE, the controller makes use of a 64-bit control word that generates all the necessary control signals (depicted in Fig. 2B) for: *i)* register control, pipeline masks, operand storage and input selection; *ii)* pre-processing module activation; *iii)* vector configuration; *iv)* VMA stage activation; *v)* quire operation and operand selection; and *vi)* quire splitting logic activation. Accordingly, the sequencer operates by first reading a configuration word from the local configuration memory, which comprises a tuple formed by the control word and a count value. Then, it uses the control word to assign the control signals and configure the PE. Depending on the executed operation, after a number of clock cycles (defined by the tuple count value), the controller obtains a new configuration word and re-configures the PE accordingly. Finally, each controller also keeps an interface to the load sequences of configurations to the local memory[1].

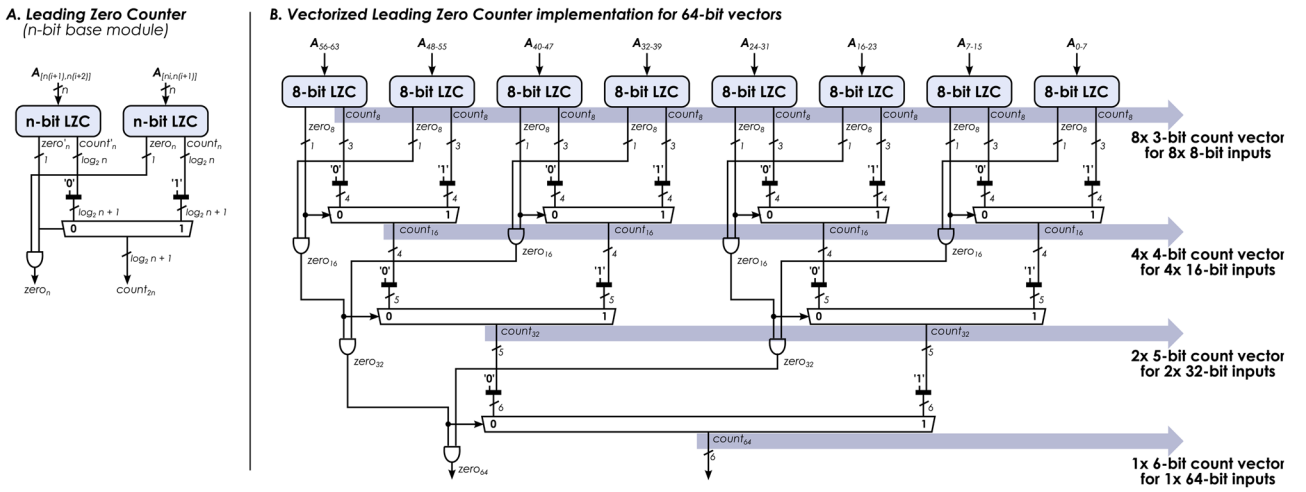

Fig. 11 Vectorized leading zero counter architecture, illustrating (**A**) a basic n-bit module and (**B**) an implementation of a 64-bit vectorized leading zero counter.

[1] Although it is out of the scope of this work, to deploy a VMA (or the RTU) either as a CPU functional unit or as a dedicated accelerator, it is only required to connect each controller to a centralized mechanism to facilitate its programming.

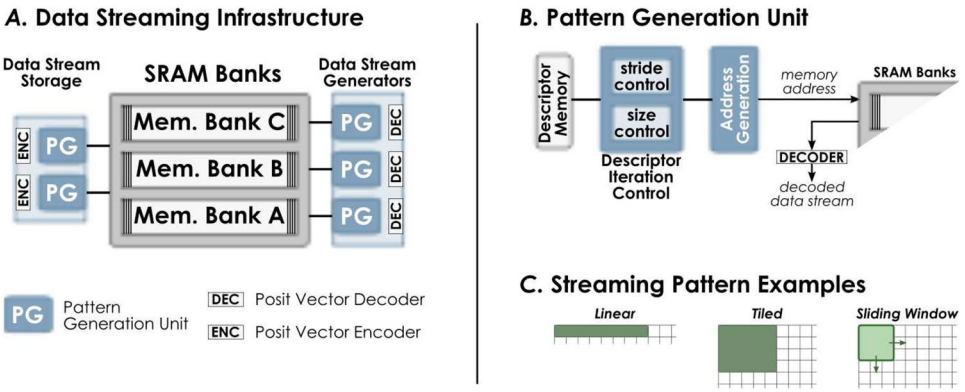

Fig. 12 Overview of the proposed RTU's data streaming infrastructure.

5 Data Streaming Mechanism

The proposed RTU deploys an autonomous data streaming infrastructure, composed of: *i)* a set of stream generators and storage controllers; and *ii)* a set of banked SRAM modules. Each stream generator/storage controller is composed of a set of descriptor-based pattern generation units, paired with Posit vector decoding/encoding logic (see Fig. 12).

Accordingly, the RTU's stream-based computation models are supported by a dedicated pattern generation unit per input (output) attached to each PE located in the left (right) column of the array. Moreover, a 3-bank SRAM memory module is deployed per row of the PE array, ensuring maximum data locality exploitation and parallelism (see Figs. 2 and 12A). These serve both as scratchpad memories (local to the RTU) and stream buffers, allowing streams to flow in and out of the PE array and promoting data reutilization.

5.1 Pattern Generation Units

Each data streaming pattern generation unit (see Fig. 12B) adopts a descriptor format based on the affine function defined in Eq. 4. The descriptor format is capable of describing both linear and tiled accesses, by generating incremental *stride* factors (`stride` and `count control` modules - see Fig. 12B), and adding them to a base address *offset* (`address generation` module - see Fig. 12B and C). Combinations of multiple descriptors allow the generation of patterns with higher levels of complexity (such as sliding window or banded patterns). To do so, the set of descriptors that are used to generate a given pattern are stored in a local descriptor memory and iterated over in the aimed sequence.

5.2 Posit Decoding and Encoding

Each pattern generation unit is also paired with a Posit *decode* or *encode* module to perform the translation from in-memory Posit vectors to the input vector unified format, and vice-versa. Each module is fully vectorized and performs the translation according to the schemes described in [12–14, 21].

The *decode* module (see Fig. 13A) translates an input posit vector ($\mathbf{V}_{posit}$) to the corresponding vectors for the sign ($\mathbf{sign}_{posit}$), exponent ($\mathbf{exp}_{posit}$), and fraction ($\mathbf{frac}_{posit}$) values.

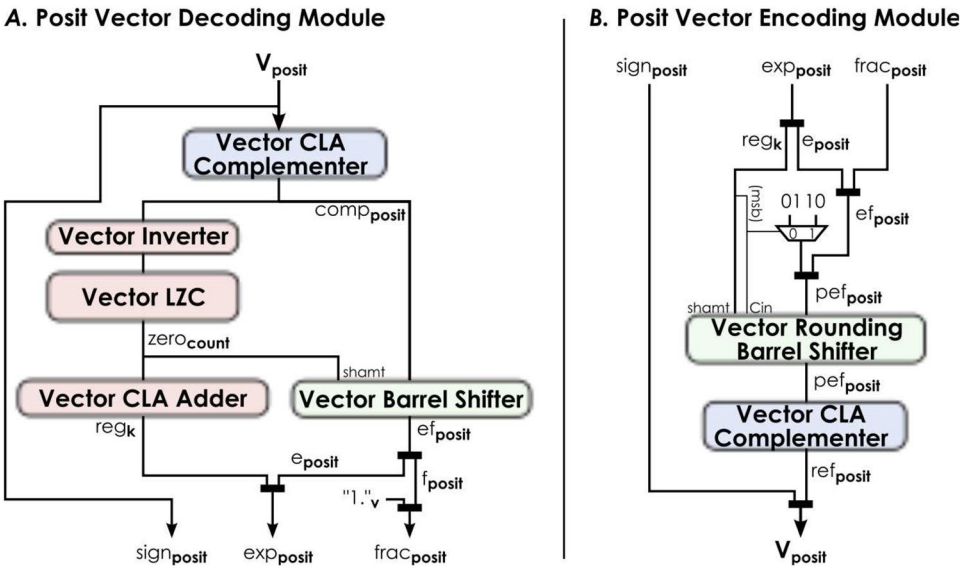

Fig. 13 Overview of the posit vector (**A**) decoding and (**B**) encoding modules.

For each posit vector, it starts by taking the 2's complement according to the sign bit of the posit, with the help of a vectorized carry-lookahead adder (as described in Sect. 4.1.2 and depicted in Fig. 8). Then, the regimes are decoded by first inverting each vector element according to the first bit of the regime. This step allows the run-length of the regime to be obtained without the need for a leading one counter. Instead, it is done solely with the aid of a vectorized leading zero counter (depicted in Fig. 11). The zero count is then simultaneously used *i)* to calculate the regime value, according to Eq. 2; and *ii)* to shift out the regime bits from the each vector element from $\mathbf{V}_{posit}$ with a vectorized left barrel shifter (as described in Sect. 4.1.3 and depicted in Fig. 10A). The resulting shifted vector contains the exponent and fraction fields for each element, which are then split according to the posit precision. Finally, the regime is concatenated to the exponent value an a '1' bit is concatenated with the fraction to obtain the fraction vector.

Conversely, the *encode* module (see Fig. 13B) translates back the sign, exponent and fraction vectors to a posit vector format. It starts by concatenating the exponent ($\mathbf{exp}_{posit}$) and fraction ($\mathbf{frac}_{posit}$) fields for each vector element, and by extracting out the regime values from the exponent value. The regime values are then used to concatenate a '01' or '10' prefix to each vector element, according to their most-significant bit (sign). Then, the regime are used to right-shift the vector elements, with the aid of a vectorized rounding barrel shifter (as described in Sect. 4.1.4 and depicted in Fig. 10B). Finally, the resulting vector elements are complemented according to the sign bits ($\mathbf{sign}_{posit}$) with a vectorized carry-lookahead adder, which are then concatenated to the corresponding element, resulting in the final posit vector ($\mathbf{V}_{posit}$).

6 Experimental Results

This section presents an evaluation of the proposed RTU in what concerns its performance and energy efficiency, by considering an ASIC implementation. The RTU is initially compared with alternative SIMD units deployed in off-the-shelf platforms, with a set of representative benchmarks particularly suited for vectorization. The evaluation is concluded with a case study comparing the RTU with dedicated architecture implementations deployed in an FPGA device.

6.1 Hardware Implementation

The proposed RTU architecture was fully synthesized for an ASIC implementation, by considering the Nangate 45*nm* PDK. Although other configurations could be considered, the RTU was implemented by assuming a 4×4 PE array to facilitate the comparison with alternative computing topologies, such as the NVIDIA tensor cores [7]. The supporting data-streaming infrastructure comprises 4 banked scratchpad memories (one per row of the array), each composed of three 8kB SRAM memories. Hardware resources and power estimation results were obtained with Cadence Genus 19.11 and the SRAM banks were generated with the OpenRAM [30] memory compiler.

The considered RTU configuration was successfully synthesized with an operating frequency of 800 MHz. An area breakdown of each RTU component is presented in Table 1, amounting to a total area of 14.204 mm^2 and an estimated peak power dissipation of about 11.7 W. As it could be expected, most of the area footprint is occupied by the PEs (782 μm^2), with the array occupying 91% of the RTU's area. This is mainly due to the VMA's 2048-bit quire arithmetic logic required for the 64-bit precision. Nonetheless, this area was kept to a minimum by sharing all the resources required to implement all the supported vector precisions, by relying on the adopted data unified formats. On the other hand, it can be ascertained that the area overhead of the whole streaming infrastructure only amounts to a total of 9% of the RTU's area, as a result of the low-profile architecture of the pattern generator units. Such a low footprint leaves room for the deployment of more complex and robust data communication schemes in future implementations.

6.2 Reference Setups and Workloads

To evaluate the proposed RTU performance, it was compared with several off-the-shelf platforms featuring advanced SIMD units (see Table 2), including: *i)* an Intel i7-8700K out-of-order processor (with the AVX2 vector extension); *ii)* an ARM Cortex-A9 embedded processor (with the Neon vector extension); *iii)* a NVIDIA GV100 GPU (equipped with tensor cores[2] and native SIMD operation in each simultaneous multiprocessor (SM)). Several setups were devised for each platform, by considering floating-point double, single, and half (only in the GPU) precisions, resulting in 7 different setups: AVX-DP, AVX-SP, NEON-DP, NEON-SP, SM-DP, SM-SP, and SM-HP. For the proposed RTU, different setups with 64-, 32-, 16-, and 8-bit Posit precisions were considered, corresponding to RTU-P64, RTU-P32, RTU-P32, RTU-P8, respectively.

A set of benchmarks (characterized in Table 3) was selected based on real-word applications, with the goal of evaluating different properties of the proposed RTU. They are divided into three categories: `vdot` and `outer`

[2] The adopted NVIDIA tensor core was used as a representative platform in the domain of tensor accelerators not only due to its accessibility, but also because it consists on a fair and valid comparison basis since its topology is close to that of the RTU base architecture.

implement highly-parallel algebra operations; `gemm` and `conv2d` represent matrix-multiplication kernels usually targeted to tensor cores; `covar` and `sgd` represent multi-phase applications composed of multiple kernels and with complex arithmetic.

The presented evaluation aims at solely comparing the proposed RTU architecture with the considered off-the-shelf SIMD units. To achieve a fair comparison, all applications were parameterized to target a single core of each platform (see Table 2), with data sets that fit in the first level of the cache hierarchy (to ensure that all these units operate with minimum latency and memory access delays). In the particular case of the GPU implementations, all benchmarks were dimensioned to target either a ***single SM block*** or its tensor cores (through the cuBLAS and cuDNN libraries) whenever possible (in the SM-HP setup). The execution times and clock cycles measurements for the RTU were obtained through cycle-accurate simulations in Cadence Incisive 19.03. For the Intel i7 and ARM processors, the applications were timed and analyzed both in terms of clock cycle count and power by accessing their internal performance counters with the PAPI library. For the GPU, the corresponding measurements were obtained with the NVIDIA profiling tools.

6.3 Architectural Evaluation

Table 4 presents the measured clock cycle count for each benchmark and setup, normalized to the NEON-DP setup (as it is the least performant in all benchmarks). The obtained clock cycle measurements clearly demonstrate the architectural efficiency of the proposed RTU, showing average clock cycle gains of 44x/12x/9x (in 64-bit precision), 64x/19x/15x (in 32-bit precision), and 20x (in 16-bit precision) when compared to the NEON-DP/AVX-DP/SM-DP, NEON-SP/AVX-SP/SM-SP, and SM-HP setups, respectively. Such gains are a result of the three-fold combination of: *i)* the parallel nature of the RTU PE array, allowing a two-level parallelization both across the PEs and within the SIMD architecture of the VMAs; *ii)* the versatility introduced by the reconfiguration mechanisms, allowing an efficient resource utilization and code-free mapping of complex operations, which requires the utilization of different compute units in the reference setups; and *iii)* the supporting data streaming infrastructure, by detaching memory accesses from computation, reducing the execution critical path, and by autonomously and efficiently generating streams in parallel with computation.

6.4 Performance Evaluation

The architectural benefits of the proposed RTU are further highlighted when comparing its execution time to the other setups, as it can be observed in Fig. 14. When considering the `vdot` benchmark, for example, it is possible to ascertain the benefit of the VMA fusing characteristics to deploy a parallel reduction tree. In fact, while all reference setups perform this operation with successive shuffling instructions, the RTU is capable of reconfiguring unused PEs to perform the reduction in parallel with the dot-product partial accumulations (see Fig. 5), in turn achieving 16x/3x/4x speedups over NEON-DP/AVX-DP/SM-DP. On the other hand, the spatial computation characteristics of the RTU become evident when considering the `outer` benchmark, where the combination of the PE array topology and the vectorization of the VMAs allows exploiting massively parallel computation. This results in a performance speedup as high as 346x, when comparing the most extreme RTU-P8 and NEON-DP setups.

Table 1 Area breakdown for the RTU and its components.

	Component	Area (mm^2)	Power (W)
PE		0.782	0.684
Stream	Pattern Generator	0.019	0.024
	Posit Decode	0.008	0.008
	Posit Encode	0.009	0.010
	SRAM Bank (8kB)	0.094	0.007
Streaming	12 PGs + Decode	0.324	0.397
Infrastructure	8 PGs + Encode	0.224	0.279
	12 SRAM Banks	1.128	0.095
RTU	4x4 PE Array	12.528	10.936
	Streaming Infr.	1.676	0.772
Total	RTU	14.204	11.708

Furthermore, thanks to its base tensor-like computing architecture, the RTU was also compared with the tensor cores present in the NVIDIA GPU (see green bars in Fig. 14 - *SM-HP). However, the strict set of restrictions imposed by NVIDIA tensor cores for the type and shape of matrix multiplications [11] only made it possible to map the `gemm` and `conv2d` benchmarks, denoting the lack of flexibility presented by these types of units. Additionally, although the `conv2d` benchmark is also based on tensor multiplication, it adopts the most common 3x3 filter shape, which is not natively supported by the NVIDIA tensor cores. To allow its mapping, the NVIDIA tools need to add padding elements to the filter kernel (increasing the memory footprint) and to transform the operation to a common matrix multiplication (similar to `gemm`). Nevertheless, when comparing the execution of `gemm` and `conv2d`, the proposed RTU using a 16-bit posit precision format is capable of matching and outperforming the NVIDIA tensor cores by 1.8x and 5.3x, respectively.

The proposed RTU still introduces an increased level of processing efficiency over the other setups by applying

Table 2 Reference SIMD-enabled platforms.

	Intel i7-8700K	ARM Cortex-A9	Nvidia GV100
Technology	14 *nm*	28 *nm*	12 *nm*
Freq. (MHz)	3700	667	1200
TDP (W)	95	1.9	250
Est. Power/Core	15.8	0.8	3.125
SIMD Tech.	AVX2	Neon	GPU SM
DP Vector-width	8	2	8 / SM Block
SP Vector-width	16	4	16 / SM Block
HP Vector-width	-	-	32 / SM Block
Tensor Cores	-	-	2 / SM Block
L1 Data Cache	32kB	32kB	128kB

time-multiplexing reconfiguration to maximize its resource utilization. This is especially emphasized when mapping full kernels with multiple phases and/or complex operations, as it is shown by the gains obtained in the `covar` and `sgd` benchmarks (see Fig. 14). In particular, `covar` takes advantage of the RTU support to map high-latency arithmetic functions (in this case, a division operation). This is done by reconfiguring unused resources in the PE array, allowing the operation to be performed in parallel with other computations. When such runtime array-wide reconfiguration between kernel phases is combined with the data reutilization offered by data streaming, the RTU achieves average speedups of 2.4x/7.8x and 1.9x/8.1x for `covar` and `sgd`, when compared to AVX/SM (with equivalent precisions). The streaming of patterns with high complexity (such as sliding windows) is also evidenced by the 2.5x/3.1x speedups obtained for `conv2d`, when comparing the same setups.

By acknowledging that the Posit format allows reducing the precision at the minimal expense of the output accuracy (depending on the dataset) [12–14], it is possible to identify the maximum performance gains attainable by the RTU. Accordingly, by halving the vector precision, it is possible to attain average speedups of 89x/5x/13x, when compared to the NEON/AVX/SM setups. On the other hand, by fully reducing the precision to 8-bit Posit vectors, the RTU is capable of attaining gains as high as 372x/14x/40x, when compared to the NEON-DP/AVX-DP/SM-DP setups.

Finally, it is also important to note that the observed gains were obtained by considering a CMOS process technology (45*nm*) to implement the RTU that is much greater than the state-of-the-art process technologies used in the other devices (28*nm*, 14*nm*, and 12*nm* - see Table 2). Accordingly, it is safe to assume that the operating frequency of the RTU would scale to the range observed in the reference setups if implemented in similar technologies. Naturally, such a performance increase would allow the RTU to attain further levels of acceleration when compared to the reference setups.

Table 3 Considered evaluation benchmarks.

Benchmark	Description	Characteristics
VDOT	Vector Dot-Product	FMA, Parallel Reduction, Linear Streaming
OUTER	Matrix Outer Product	Massively-Parallel, Bandwidth Saturation, Linear Streaming
GEMM	General Matrix-Mult. (C=αAB + βC)	Tensor-optimized FMA, Tiled Streaming
CONV2D	2D Convolution 3x3 Filter	Resource Underuse, Reduction Sliding Window Streaming
COVAR	Covariance Kernel	Multi-phase, Division Linear+Tiled Streaming
SGD	Mini-Batch Stochastic Gradient Descent	Multi-phase, Reduction Data Reuse, Linear Streaming

6.5 Energy Efficiency

The observed performance gains have a significant impact in the total energy consumption of the proposed RTU, as shown in Fig. 15A. In this graph, it can be observed that the RTU consumes a much lower amount of energy when compared to the reference platforms. As an example, the RTU-P64 consumes 2.5x less energy (on average) than the NEON-DP setup. This is a direct result of the applied data streaming mechanics, together with the spatial and temporal execution models of the RTU. When operating the RTU with an 8-bit Posit precision, it is capable of attaining further reductions, as high as 7.46x.

To gather all the observed results in a single metric, an additional energy efficiency study was performed. In this case, it was used an energy-delay product (EDP) metric (see Fig. 15B), calculated by multiplying the total energy consumption by the average execution time, in all benchmarks. By keeping in mind that lower values represent a higher efficiency, the measured results not only reflect the lower energy consumption of the RTU but also highlight the efficiency of its combined execution model. Accordingly, it is possible to observe an overall performance-energy efficiency improvement of 87x (on average), when comparing the proposed RTU with all the considered reference setups.

6.6 Comparison with Dedicated FPGA-Based Accelerators

To consolidate the presented discussion about the advantages of the proposed RTU, it was also compared with dedicated FPGA implementations for each of the considered benchmarked applications. The evaluation presented below

Table 4 Clock cycle count (normalized to NEON-DP).

Bench.	NEON-DP	NEON-SP	AVX-DP	AVX-SP	SM-DP	SM-SP	SM-HP	SM-HP(*)	RTU-P64	RTU-P32	RTU-P16	**RTU-P8**
VDOT	1.00	1.25	0.96	1.29	1.87	2.11	2.33	-	16.87	32.75	61.01	106.05
OUTER	1.00	1.31	4.86	6.05	6.51	7.66	8.48	-	57.01	111.07	211.18	384.46
GEMM	1.00	1.07	5.09	5.34	5.07	5.52	10.09	56.66	53.02	105.69	209.80	413.02
CONV2D	1.00	1.09	4.50	5.08	11.23	12.13	16.53	15.72	63.76	84.95	168.91	164.11
COVAR	1.00	1.27	2.82	3.14	3.37	3.80	5.62	-	30.99	61.62	121.84	238.06
SGD	1.00	1.15	4.09	4.26	2.92	3.50	4.36	-	41.44	60.15	74.59	81.07

(*)SM-HP benchmarks mapped to Tensor Cores when supported

provides an additional level of validation of the RTU, by directly comparing it to a representative set of dedicated architectures, placing it among the domain of reconfigurable and embedded accelerators. To do so, each benchmark from Table 3 was individually designed and implemented with the Spatial DSL framework [17]. The adoption of this DSL allows extracting the inherent characteristics of each benchmark (see Table 3), automatically mapping them to high-level constructs, and subsequently compiling the design to an FPGA target. This results in a set of dedicated accelerators optimized for the platform.

Accordingly, all the adopted benchmarks were successfully implemented by targeting a Xilinx ZC702 FPGA, by considering an operating frequency of 200 MHz (refer to Table 5 for implementation details regarding resource utilization and power consumption), and compared with the proposed RTU ASIC implementation (running at 800 MHz). Similarly to the initial study, the dataset of each benchmark was carefully dimensioned to keep off-chip DRAM accesses to the minimum and by aiming to extract the maximum possible amount of parallelism in the computation kernels, without violating the resource availability constraints imposed by the FPGA device (see Table 5). For this study, the FPGA implementations only adopt a 32-bit floating-point format. This is because the adopted compilation tool instantiates data access ports with the same width as the bit precision of the data operands. As such, the number of required data accesses per application is always the same, independently of the data precision. Nonetheless, as it was done in the previous study, the four RTU setups (RTU-P64, RTU-P32, RTU-P32, RTU-P8) were still considered to demonstrate the data vectorization capabilities that can be exploited in the RTU, by reducing data precision (once again assuming that it is allowed by the characteristics of the dataset).

6.6.1 Evaluation of Measured Performance

Fig. 16 presents the measured performance speedup when comparing the RTU setups to the FPGA implementations. The obtained results further validate the benefits introduced by the proposed RTU, even when compared to dedicated architectures. In particular, when considering the most parallel `vdot`, `outer`, and `gemm` benchmarks, the obtained results further highlight the spatial computation capabilities of the RTU's PE array. In fact, even without considering VMA vectorization, the RTU is capable of achieving a performance speedup of 9x/11x/19x for the `vdot`/`outer`/`gemm` benchmarks, when comparing the RTU-P64 setup with the corresponding FPGA implementations. Furthermore, when exploiting the characteristics of the Posit format to fully reduce precision and maximize data vectorization, it is possible to obtain performance speedups

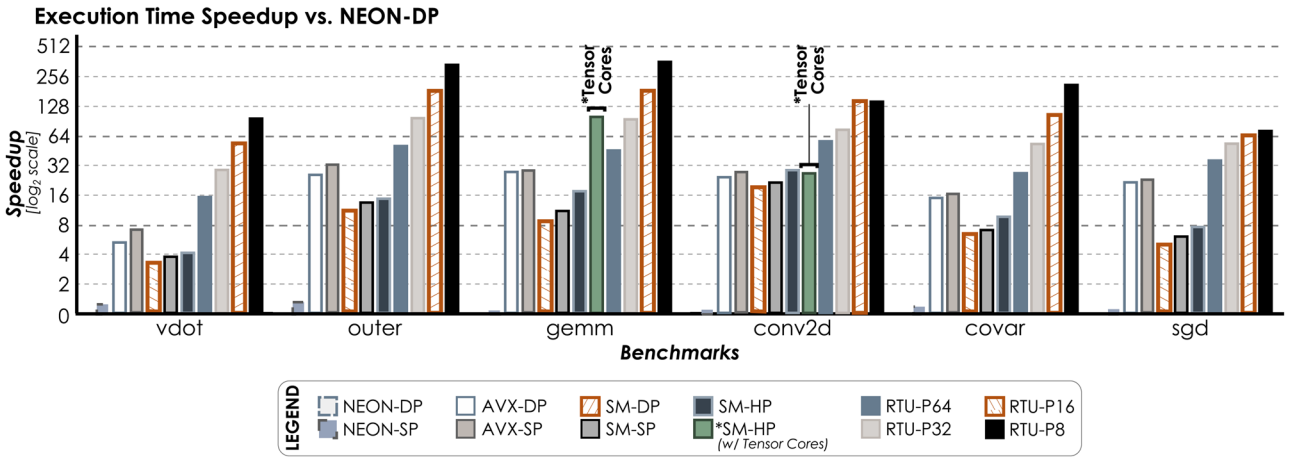

Fig. 14 Performance comparison results, including execution time speedup, normalized to NEON-DP.

as high as 59x/73x/151x for the `vdot`/`outer`/`gemm` benchmarks, when considering the RTU-P8 setup.

Furthermore, the reconfiguration mechanisms introduced by the RTU provide a level of efficiency over the obtained FPGA implementations. In particular, the VMA fusing capability to deploy in-situ parallel reduction trees is clearly visible when considering the `conv` benchmark. While the RTU is capable of reconfiguring itself to operate over data moving through the PE array, the parallel reduction trees required to deploy the convolution operation in the FPGA are highly limited by the available data bandwidth, in turn, decreasing the performance of the design. This results in an observable average speedup of 144x in the RTU for the `conv` benchmark.

On the other hand, the time-multiplexing benefits to deal with multi-phase applications and the adaptability of the RTU to deploy complex operations are once again highlighted by the `covar` and `sgd` benchmarks. Even though the FPGA designs are deployed with specialized architectures, the efficiency of the implementation is still limited by the amount, placement, and specialization of the resources available in the device. As already mentioned, these RTU capabilities allow instant specialization to deploy high-latency arithmetic, which benefits applications such as the `covar` benchmark that requires floating-point division operations. In fact, while the FPGA design has to map these operations to DSPs and redirect data to particular regions of the device, the RTU is capable of performing these operations within the PE array by combining multiple units.

Moreover, the RTU is also particularly suited to deal with multi-phase applications, such as `sgd` and `covar`. In combination with the data reutilization capabilities of the underlying data streaming infrastructure, this allows the RTU to instantly reconfigure itself to deploy different computing kernels while keeping intermediate data close to the computing resources. Such capabilities, when combined with the spatial computation offered by the RTU, result in average speedups of 177x and 81x for `covar` and `sgd`, when compared to the corresponding FPGA implementations.

Notwithstanding, it is important to note that the FPGA fabric is also reconfigurable and the same time-multiplexing functionalities could also be deployed by exploiting its partial reconfiguration capabilities. However, the Spatial DSL framework [17] does not support this functionality. Nonetheless, while the RTU is capable of reconfiguring itself in a single clock cycle, the partial reconfiguration procedure of an FPGA fabric takes several milliseconds and requires a non-negligible amount of additional control structures and hardware resources.

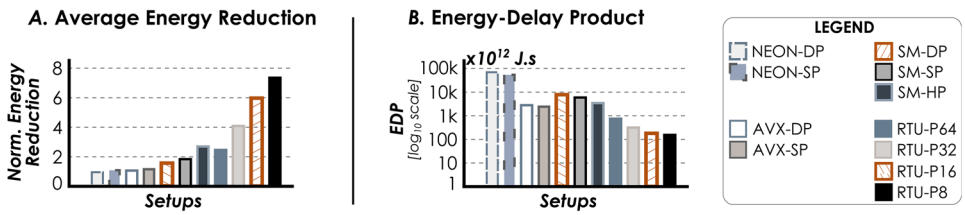

Fig. 15 Energy consumption results, including (**A**) average energy savings normalized to NEON-DP, and (**B**) overall energy efficiency (in the form of an EDP metric).

Table 5 Reference implementation details for the Xilinx ZC702 FPGA (@ 200 MHz).

Bench.	LUT	Registers	BRAM	DSP	Total Power
VDOT	25753 (48%)	36761 (35%)	4 (3%)	192 (87%)	2.32 W
OUTER	23100 (43%)	26770 (25%)	35.5 (25%)	19 (9%)	2.17 W
GEMM	27017 (51%)	34150 (32%)	8 (6%)	68 (31%)	2.14 W
CONV2D	34242 (64%)	27790 (26%)	2.5 (2%)	62 (28%)	1.95 W
COVAR	39042 (73%)	48511 (46%)	16 (11%)	45 (20%)	2.27 W
SGD	17568 (33%)	20447 (19%)	32 (23%)	52 (24%)	2.06 W

6.6.2 FPGA Performance Limitations and Speedup Normalization

To better contextualize the performance results of both the proposed RTU and the FPGA-based accelerators that were considered in the presented comparison, it is important to remark some of the underlying limitations of the FPGA platform. In particular, it would be expected that a dedicated accelerator computing architecture would be able to deliver a higher performance than a more general-purpose architecture, such as the proposed RTU. However, even though the accelerators generated by the Spatial DSL framework [17] are highly optimized, their performance is heavily limited by their maximum operating frequency and the data bandwidth that is made available to the FPGA fabric. In particular, the generated architectures can only run with a maximum frequency of up to 200 MHz in the adopted Xilinx ZC702 FPGA, whereas the RTU runs at 4x higher frequency (800 MHz). Additionally, the adopted Xilinx ZC702 FPGA has a maximum available bandwidth for data loading of 3.2 GB/s [31]. Conversely, the RTU is capable of streaming data to the PE array at a maximum rate of 72 GB/s (4×3 64-bit read channels @ 800 MHz). This results in a 22.5x data transfer bandwidth increase over the FPGA designs.

Accordingly, to take this factor into account, the previously presented performance speedup results were normalized both to the operating frequency and to the available data bandwidth for each platform, as it is shown in the graphs presented in Figs. 17 and 18, respectively.

As it would be expected, the obtained results show a smaller performance gap between the RTU and the FPGA implementations. In particular, when considering the speedup normalized to the operating frequency, it is possible to observe that the performance gap between the RTU and the FPGA implementations is reduced by a factor of 4x, while maintaining a proportional relation (see Fig. 17) to the performance values discussed in Sect. 6.6.1. This results in an average performance gain of 9.1x/14.7x/26.6x/39.8x for the RTU-P64/RTU-P32/RTU-P32/RTU-P8 setup, when compared to the FPGA implementations.

Conversely, when considering the speedup normalized to the available data bandwidth (see Fig. 18), the performance gap becomes ever closer, allowing to further highlight the benefits of the proposed RTU. In particular, when considering the `vdot`,`outer`, and `gemm` benchmarks, it can be observed that the RTU-P64 setup would be slightly outperformed by the corresponding FPGA implementations. However, it should be noted that the RTU is still capable of fully exploiting the VMA vectorization, attaining a performance speedup as high as 2.6x/3.26x/6.73x when comparing the RTU-P8 setup with the corresponding FPGA implementations for `vdot`/`outer`/`gemm`.

When considering the `conv2d`, `covar`, and `sgd` benchmarks, the benefits that are introduced by the combination of the RTU's reconfiguration mechanisms and its supporting data streaming infrastructure become evident once again. In particular, the RTU shows a performance speedup of 3.39x/2.16x/2.33x for `conv2d`/`covar`/`sgd`, when comparing the RTU-P64 setup with the FPGA implementations.

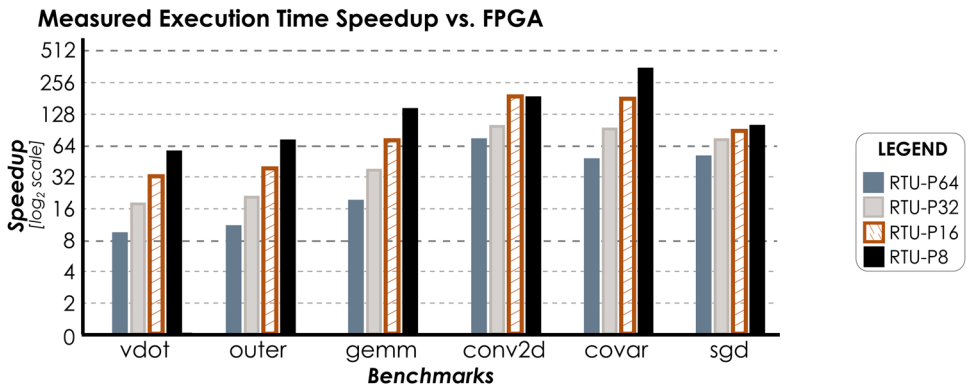

Fig. 16 Performance comparison results against the FPGA setups running at 200 MHz.

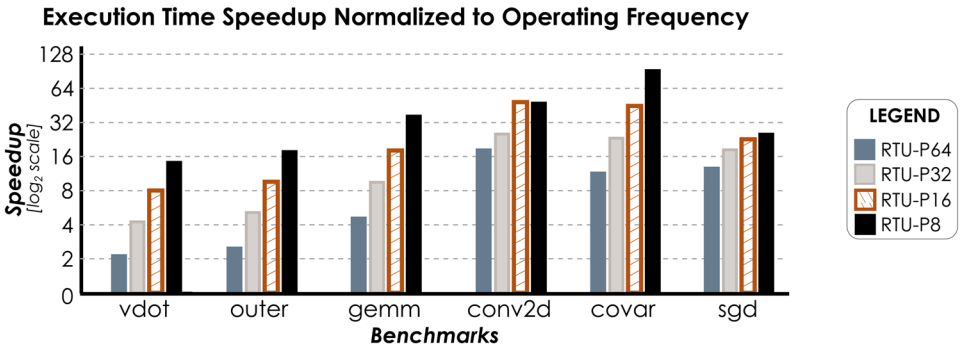

Fig. 17 Performance comparison results against the FPGA setups, normalized by the operating frequency.

Besides the architectural versatility and adaptability, this is a direct result of the RTU's data streaming infrastructure that not only allows optimizing data movement and reutilization within the design, but it also transparently handles data movement behind computation. Naturally, when considering the maximum possible vectorization, RTU-P8 setup is capable of achieving performance speedups of 8.73x/16.56x/4.56x for `conv2d`/`covar`/`sgd`, when compared with the corresponding FPGA implementations.

6.6.3 Energy Efficiency Evaluation

Finally, the impact of the RTU's advantages is also evident by in terms of total energy consumption observed in both setups, as shown in Fig. 19A. In this graph, it can be observed that despite the lower average power consumption of 2.15 W (see Table 5) imposed by the FPGA implementations (vs. 12 W in the RTU), the RTU consumes 160x less energy (on average), as a direct result of improved performance. This is further highlighted by the calculated EDP metric (see Fig. 19B), showing an overall performance-energy efficiency improvement ranging from 3 to 5 orders of magnitude higher, when comparing the proposed RTU with the FPGA implementations.

6.7 Discussion on Alternative PE Array Topologies

Although the evaluated RTU prototype was implemented by assuming a 4×4 PE array, other different array topologies could equally be considered.

In particular, the communication interfaces of the PE architecture were especially designed to ease the deployment of different array topologies. In fact, the considered PEs were made as modular as possible, making it only necessary to connect additional rows and/or columns of PEs to increase the size of the array. This can be achieved without requiring any modifications to the PE architecture.

Naturally, when such different topologies are exploited, different outcomes are expected to arise from increasing the width and depth of the array. One the one hand, by widening the array, it is naturally possible to exploit higher levels of spatial parallelism. On the other hand, by deepening the array, it is possible to deploy larger and more compute-intensive kernels, by providing a much larger pool of computing resources. As such, the amount of exploited parallelism and computing complexity can be adjusted by defining the width and depth of the array. Such compromise is only limited by the balance between the available data bandwidth and the available chip area for the RTU.

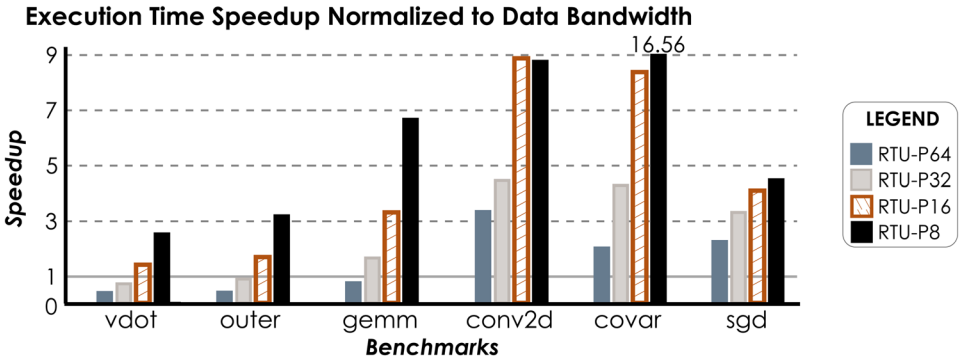

Fig. 18 Performance comparison results against the FPGA setups, normalized by the available memory bandwidth.

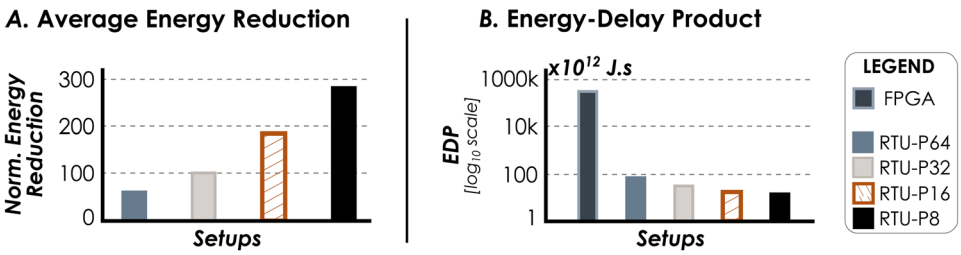

Fig. 19 Energy consumption evaluation, including (**A**) average energy savings normalized to the FPGA setups, and (**B**) overall energy efficiency (in the form of an EDP metric).

7 Conclusions

This paper proposed a new RTU architecture that leverages the new Posit floating-point format to deploy a 2D computing array of variable-precision SIMD units. The proposed unit was designed by recognizing the opportunity to explore the resources of existing tensor units for more general-purpose computing contexts. To do so, the proposed RTU deploys a combined data streaming, spatial and temporal execution model, to deploy a reconfigurable compute unit that is capable of fusing multiple PEs to map high-level operations, by exploiting time-multiplexing reconfiguration mechanisms. The obtained results for a 45*nm* ASIC implementation show that the proposed RTU provides an increased performance not only over existing state-of-the-art tensor and SIMD units present in off-the-shelf platforms, but also over dedicated FPGA-based accelerators, resulting in significant energy efficiency gains.

References

1. Dean, J., Patterson, D., & Young, C. (2018). A new golden age in computer architecture: Empowering the machine-learning revolution. *IEEE Micro, 38*(2), 21–29.
2. Hennessy, J. L., & Patterson, D. A. (2019). A new golden age for computer architecture. *Communications of the ACM, 62*(2), 48–60.
3. Chung, E., Fowers, J., Ovtcharov, K., Papamichael, M., Caulfield, A., Massengill, T., Liu, M., Lo, D., Alkalay, S., Haselman, M., et al. (2018). Serving dnns in real time at datacenter scale with project brainwave. *IEEE Micro, 38*(2), 8–20.
4. Delaye, E., Sirasao, A., Dudha, C., & Das, S. (2017). Deep learning challenges and solutions with xilinx fpgas. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, pp. 908–913.
5. Fowers, J., Ovtcharov, K., Papamichael, M., Massengill, T., Liu, M., Lo, D., Alkalay, S., Haselman, M., Adams, L., Ghandi, M., et al. (2018). A configurable cloud-scale dnn processor for real-time ai. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, pp. 1–14.
6. Jouppi, N. P., Young, C., Patil, N., & Patterson, D. (2018). A domain-specific architecture for deep neural networks. *Communications of the ACM, 61*(9), 50–59.
7. NVIDIA. (2017). Nvidia tesla v100 GPU architecture. *White paper*.
8. Reagen, B., Whatmough, P., Adolf, R., Rama, S., Lee, H., Lee, S. K., Hernández-Lobato, J. M., Wei, G.-Y., & Brooks, D. (2016). Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, IEEE, pp. 267–278.
9. Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 1–12.
10. Köster, U., Webb, T., Wang, X., Nassar, M., Bansal, A. K., Constable, W., Elibol, O., Gray, S., Hall, S., Hornof, L., et al. (2017). Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In *Advances in neural information processing systems*, pp. 1742–1752.
11. Markidis, S., Der Chien, S. W., Laure, E., Peng, I. B., & Vetter, J. S. (2018). Nvidia tensor core programmability, performance & precision. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE, pp. 522–531.
12. Gustafson, J. L., & Yonemoto, I. T. (2017). Beating floating point at its own game: Posit arithmetic. *Supercomputing Frontiers and Innovations, 4*(2), 71–86.
13. Carmichael, Z., Langroudi, H. F., Khazanov, C., Lillie, J., Gustafson, J. L., & Kudithipudi, D. (2019). Deep positron: A deep neural network using the posit number system. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, pp. 1421–1426.
14. Chaurasiya, R., Gustafson, J., Shrestha, R., Neudorfer, J., Nambiar, S., Niyogi, K., Merchant, F., & Leupers, R. (2018). Parameterized posit arithmetic hardware generator. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, IEEE, pp. 334–341.
15. P. W, & Group. (2018). *Posit standard documentation. Release, 3,* 2.
16. Chen, Y.-H., Krishna, T., Emer, J. S., & Sze, V. (2016). Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits, 52*(1), 127–138.
17. Koeplinger, D., Feldman, M., Prabhakar, R., Zhang, Y., Hadjis, S., Fiszel, R., Zhao, T., Nardi, L., Pedram, A., Kozyrakis, C., et al. (2018). Spatial: A language and compiler for application accelerators. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 296–311.
18. Nowatzki, T., Gangadhar, V., Ardalani, N., & Sankaralingam, K. (2017). Stream-dataflow acceleration. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, pp. 416–429.
19. Prabhakar, R., Zhang, Y., Koeplinger, D., Feldman, M., Zhao, T., Hadjis, S., Pedram, A., Kozyrakis, C., & Olukotun, K. (2017). Plasticine: A reconfigurable architecture for parallel patterns. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, pp. 389–402.

20. Neves, N., Tomás, P., & Roma, N. (2017). Adaptive in-cache streaming for efficient data management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 25*, 7, 2130–2143.
21. Jaiswal, M. K., and So, H. K. (2018). Architecture generator for type-3 unum posit adder/subtractor. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, pp. 1–5.
22. Forget, L., Uguen, Y., & De Dinechin, F. (2019). Hardware cost evaluation of the posit number system. In *Compas'2019 - Conférence d'informatique en Parallélisme, Architecture et Système*, pp. 1–7.
23. Zhang, H., et al. (2019) Efficient posit multiply-accumulate unit generator for deep learning applications. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, pp. 1–5.
24. Ghosh, S., Martonosi, M., & Malik, S. (1997). Cache miss equations: An analytical representation of cache misses. In *Proceedings of the 11th international conference on Supercomputing*, pp. 317–324.
25. Paiágua, S., Pratas, F., Tomás, P., Roma, N., & Chaves, R. (2013). Hotstream: Efficient data streaming of complex patterns to multiple accelerating kernels. In *2013 25th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, IEEE, pp. 17–24.
26. Hussain, T., Palomar, O., Unsal, O., Cristal, A., Ayguadé, E., & Valero, M. (2014). Advanced Pattern based Memory Controller for FPGA based HPC applications. In *2014 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, pp. 287–294.
27. Grosser, T., Zheng, H., Aloor, R., Simbürger, A., Größlinger, A., & Pouchet, L.-N. (2011). Polly-polyhedral optimization in llvm. *In Proceedings of the First International Workshop on Polyhedral Compilation Techniques (IMPACT), 2011*, 1.
28. De Dinechin, F., Forget, L., Muller, J.-M., & Uguen, Y. (2019). Posits: the good, the bad and the ugly. *In Proceedings of the Conference for Next Generation Arithmetic, 2019*, 1–10.
29. Viitanen, T., Jääskeläinen, P., Esko, O., & Takala, J. (2013). Simplified floating-point division and square root. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, IEEE, pp. 2707–2711.
30. Guthaus, M. R., Stine, J. E., Ataei, S., Chen, B., Wu, B., & Sarwar, M. (2016). Openram: An open-source memory compiler. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, pp. 1–6.
31. Svensson, B. J. (2016). Exploring opencl memory throughput on the zynq. *Technical Report*.

Nuno Neves received the Ph.D. degree (2019) in Electrical and Computer Engineering from Instituto Superior Técnico (IST), Universidade de Lisboa (UL), Portugal. He is currently a Junior Researcher at Instituto de Telecomunicações and at the High Performance Computing Architectures and Systems (HPCAS) of Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID). His main research interests include reconfigurable architectures, stream-based computing, domain-specific accelerators and languages, and compilers. He is a member of the IEEE Circuits and Systems Society.

Pedro Tomás received the Ph.D. in Electrical and Computer Engineering (ECE) from Instituto Superior Técnico(IST), Technical University of Lisbon, Portugal, in 2009. He is an assistant professor in the Department of ECE, IST, and a senior researcher at Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID). His research activities include computer microarchitectures, specialized computational structures, and high-performance computing. He is a member of the IEEE Computer Society and has contributed to more than 70 papers to international peer-reviewed journals and conferences.

Nuno Roma received the Ph.D. degree in electrical and computer engineering from Instituto Superior Técnico (IST), Universidade Técnica de Lisboa, Portugal, in 2008. Currently, he is an Associate Professor with the Department of Electrical and Computer Engineering of IST and he is a Senior Researcher and Coordinator of the High Performance Computing Architectures and Systems (HPCAS) research area of Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID) - a not for profit R&D institute affiliated with IST. His research interests include computer architectures, specialized and dedicated structures for digital signal processing, energy-aware computing, parallel processing and high-performance computing systems. He contributed to more than 120 manuscripts to journals and international conferences and served as a Guest Editor of Springer Journal of Real-Time Image Processing (JRTIP) and of EURASIP Journal on Embedded Systems (JES). He has also acted as the organizing chair of several workshops and special sessions. He has a consolidated experience on funded research projects leadership and he is member of several research Networks of Excellence (NoE), including HiPEAC (European Network of Excellence on High Performance and Embedded Architecture and Compilation). Dr. Roma is a Senior Member of both IEEE and ACM.