Reconfigurable Stream-based Tensor Unit with Variable-Precision Posit Arithmetic

Nuno Neves
INESC-ID
Lisbon, Portugal
nuno.neves@inesc-id.pt

Pedro Tomás
INESC-ID, Instituto Superior Técnico
Universidade de Lisboa
Lisbon, Portugal
pedro.tomas@inesc-id.pt

Nuno Roma
INESC-ID, Instituto Superior Técnico
Universidade de Lisboa
Lisbon, Portugal
nuno.roma@inesc-id.pt

***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 RTU is also supported by an automatic data streaming infrastructure and a pipelined data movement 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, in turn resulting in significant energy-efficiency improvements.**

***Index Terms*—Tensor Computation, Posit Number System, Variable-Precision SIMD, Spatial and Temporal Reconfiguration, Data Stream Computing**

I. Introduction

New algorithmic advances, allied with the ever-increasing availability of data, led the industrial and academic research to shift to domain-specific and reconfigurable architectures [1], [2]. 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]. 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.

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 [8]–[10], but also significant reductions in chip area. 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 [9], Google [10], NVIDIA [8], Xilinx [6], and Microsoft [5], have already proposed or adopted such alternative formats in their off-the-shelf platforms and accelerators.

Despite their success, tensor units are often overspecialized, constraining their usage on operations other than tensor multiplication. For example, the tensor cores equiping recent NVIDIA Graphics Processing Units (GPUs) [8] 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 [10], imposing accuracy losses in higher-precision applications, or are limited to the IEEE-754 format [8], 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 renormalization of intermediate results [15].

From the dataflow perspective, 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 [16], [17]. These mechanisms would allow 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 the range of supported application domains broadens with such a solution, 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]–[19] remains the most suited approach to support a reconfigurable tensor unit. Besides their natural support for spatial computing schemes [16], stream-

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, and by funds from the European Union Horizon 2020 research and innovation programme under grant agreement No. 826647.

2160-052X/20/$31.00 ©2020 IEEE
DOI 10.1109/ASAP49362.2020.00033

Authorized licensed use limited to: b-on: Universidade de Lisboa Reitoria. Downloaded on September 06,2020 at 06:27:23 UTC from IEEE Xplore. Restrictions apply.

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 vector element width 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 that leverages 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 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. Such gains also resulted in significant energy efficiency improvements.

II. Background and Related Work

A. Posit Number System and Implementations

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

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

Similarly to the IEEE-754, the structure of the posit format (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 $rrr\ldots\bar{r}$) that encodes a signed value k. Together with the exponent field, the regime k represents the working range of the represented real value (or scale factor). The numerical value of k is determined by the run length of 1s or 0s in the regime bits. As a result, the exponent and fraction contents are unknown before decoding the regime (see Fig. 1). Depending on the run length, they can be partly (or fully) left out of the binary encoding. Accordingly, the posit value is given by:

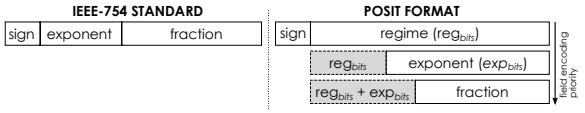

Fig. 1. IEEE-754 standard (left) and Posit (right) formats.

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

Despite the precision and exponent parameters of a posit format being arbitrary, 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 that seek the adoption of the Posit format. Jaiswal et al. [20] and Chaurasiya et al. [14] proposed the first parameterized posit arithmetic architecture implementations. They observed the area and energy of each operator is similar to their IEEE-754 counterparts. Recently, Charmichael et al. [13] applied the posit format to DNNs, showing that the 8-bit Posit precision achieves an inference accuracy that is comparable to that obtained with a 32-bit IEEE-754 implementation.

B. 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 [21]. 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:

$$y(X) = offset + \sum_{k=1}^{n} x_k \times stride_k\ ,\ x_k \in \{0, \cdots, dim_k\} \quad (3)$$

Such a representation allows indexing a significant amount of regular access patterns. Nonetheless, several approaches have been proposed that rely on dedicated ISAs [16] and descriptor-based mechanisms [19] to represent patterns with higher complexity (by combining of multiple functions). For instance, Neves et al. [19] proposed a dynamic descriptor specification to encode arbitrarily complex (regular) data-patterns. Also, Nowatzki et al. [16] proposed a stream-dataflow ISA capable of generating streams with 2D affine patterns.

C. Domain-Specific Accelerators

An outstanding emergence of Domain-Specific Architectures (DSAs) as been observed in recent years. The particular requirements of DNN applications drove the development of new tensor-based architectures. Being Google's Tensor Processing Unit [10] 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 [8] provide an slightly increased level of usability

Authorized licensed use limited to: b-on: Universidade de Lisboa Reitoria. Downloaded on September 06,2020 at 06:27:23 UTC from IEEE Xplore. Restrictions apply.

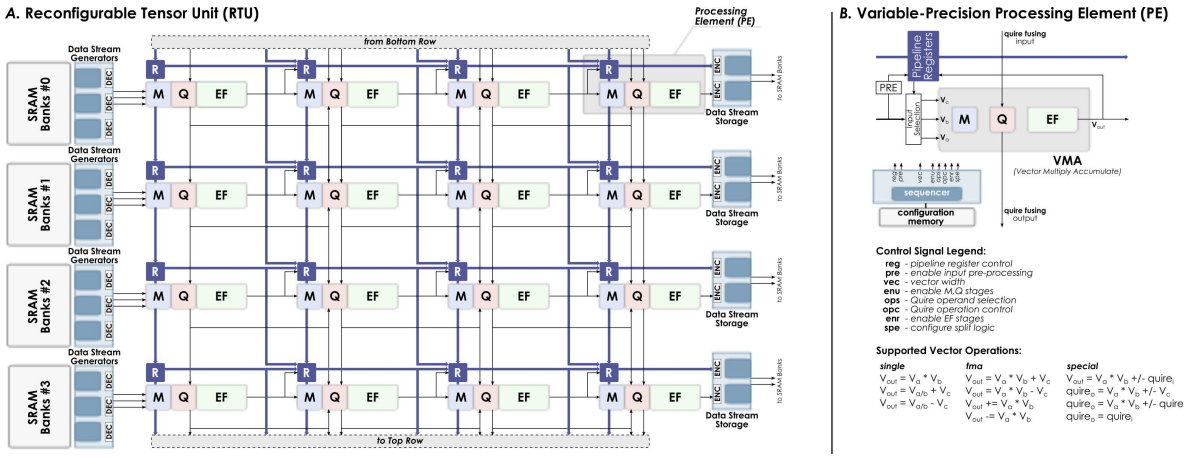

Fig. 2. Proposed RTU block diagram, depicting (A) the PE array and data streaming structures, and (B) the PE module, its components and functionality.

through their integration in general-purpose GPUs. Other accelerators have also been proposed to tackle this application domain. As an example, Chen et al. [18] 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 [16], [17]. In particular, CGRAs have been deployed by combining spatial and temporal computation schemes to achieve energy-efficient acceleration. As an example, Prabhakar et al. [17] proposed the Plasticine, designed to efficiently execute parallel patterns, through a 2D array of reconfigurable units. Nowatzki et al. [16] proposed a stream-dataflow CGRA capable of reconfiguring its datapath and memory streams.

III. 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 Section IV), each implementing a 64-bit posit Vector Multiply-Accumulate (VMA) unit (see Fig. 2.B). Its execution model is based on a data streaming operation, supported by autonomous stream generators connected to a banked scratchpad/buffering memory structure (see Section V). 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 [22] or derived from existing Domain-Specific Languages (DSLs), making the proposed RTU suitable for deployment both in CPUs (as a functional unit) or dedicated accelerators.

A. RTU Reconfiguration and Execution Models

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

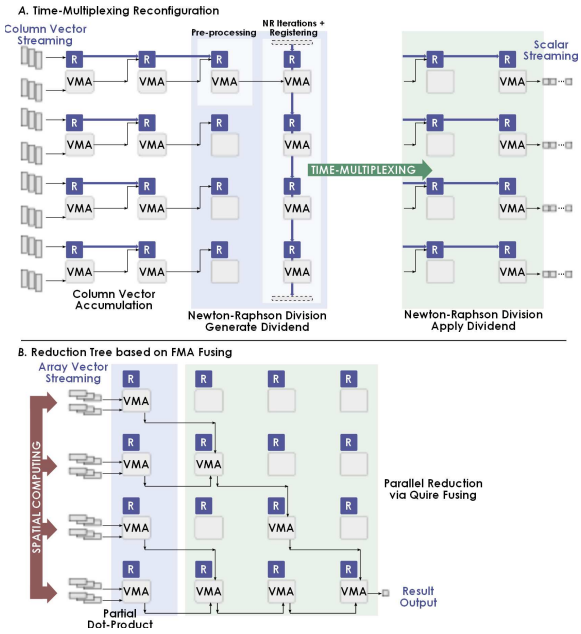

Fig. 3. Examples illustrating (A) the computation of the column average of a matrix and (B) a dot-product operation with a reduction tree. In example (A), the left half of the RTU is accumulating the column vectors, while the right half PE array is initially configured to perform a division pre-computation and it is later reconfigured to calculate the final average result for each column. Example (B) shows a reduction tree implemented via VMA fusing.

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

Authorized licensed use limited to: b-on: Universidade de Lisboa Reitoria. Downloaded on September 06,2020 at 06:27:23 UTC from IEEE Xplore. Restrictions apply.

Time-Multiplexing Reconfiguration: When mapping computing kernels other than tensor multiplication, it is possible that part of the RTU's resources become underutilized and they can 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. A reconfigurable execution scenario is shown in Fig. 3.A to obtain the average of a matrix columns.

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. 2.B) - to map more complex operations (such as reduction trees - illustrated in Fig. 3.B). 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], [23]. Although such a scenario is 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.

B. 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 Section V), 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.A). Each pattern generator leverages a descriptor-based approach to generate the most commonly adopted regular data patterns.

2D Pipelined Execution: Intercommunication between PEs is supported by a 2D pipelined register transfer grid (implemented by the `R` modules in Fig. 2.A). This is done by placing a pipeline register bank attached to each PE (see Fig. 2.B and Section IV), allowing data forwarding to three adjacent PEs (right, bottom, and bottom-right), as it is depicted in Fig. 3.A.

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. 3.B).

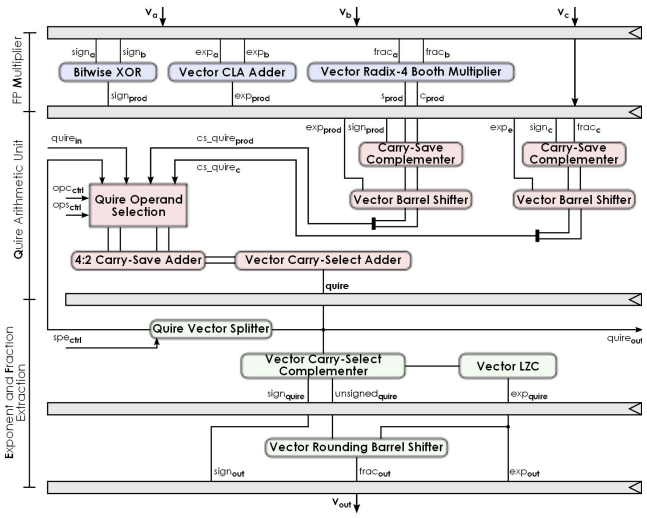

Fig. 4. Vector Multiply-Accumulate (VMA) unit architecture.

IV. 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 4). 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 vectors 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.

As a design decision to reduce the hardware footprint and latency of each PE, the required posit decoding and encoding logic was moved to the data streaming infrastructure of the RTU (see Section V), 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. The following sections detail each PE component.

A. Variable-Precision VMA unit

The VMA architecture (depicted in Fig. 4) implements a 4-stage pipeline FMA compute unit with three input 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.

Vector Data Formats: To support the variable-precision hardware that implements the VMA, 64-bit posit vectors (see Fig. 5.A) are decoded (during streaming) into three unified vector formats that gather the posit sign, exponent and fraction

Authorized licensed use limited to: b-on: Universidade de Lisboa Reitoria. Downloaded on September 06,2020 at 06:27:23 UTC from IEEE Xplore. Restrictions apply.

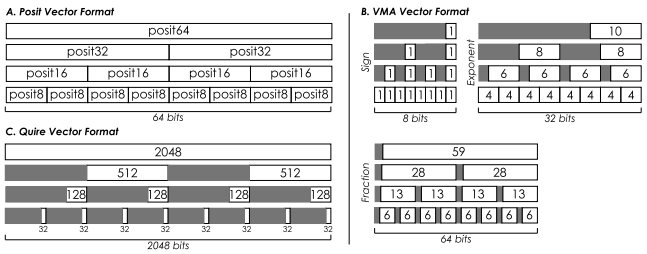

Fig. 5. Vector data formats for (A) posit vectors, (B) VMA input/output vectors, and (C) quire vectors. Grey areas represent unused bits (set to '0').

components, for each supported vector element precision (see Fig. 5.B). 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. 5.C). In the adopted formats, bits that are unnecessary to represent vector element values are set to '0'.

Floating-Point Multiplier: The first VMA stage (see Fig. 4) performs the multiplication of the $\mathbf{V}_a$ and $\mathbf{V}_b$ vectors (and propagates $\mathbf{V}_c$ to the next stage). To provide variable-precision functionality, exponent vectors are added with a specialized carry-lookahead adder. 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 multiplied with the aid of a dedicated module implementing a 8×8 structure of 8-bit radix-4 Booth multipliers, generating 64 partial products in carry-save format. These partial results are gathered and added through a carry-save accumulator tree, resulting in a 128-bit carry-save value. Finally, the resulting sign vector is calculated by performing a bitwise XOR to the input sign vectors.

Quire Arithmetic Unit: The second stage of the VMA (see Fig. 4) 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 form 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 shifting levels, depending on the considered precision. 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.

Fraction and Exponent Extraction: The final two stages of the VMA (see Fig. 4) 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. 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.

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. 4). By successfully performing this operation, it is possible to reduce a vector to a single scalar value (of the same precision).

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 Digital Signal Processors (DSPs). Examples comprise the Newton–Raphson and/or Goldschmidt algorithms [24] 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 [24]. 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. 2.B) placed at the input of the PE, to scales an input value and generates the corresponding scaling factor.

B. Pipeline Registers

Each VMA unit is paired with a local 8x106-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.

C. Configuration Controller

Each PE is managed by a dedicated configuration controller (see Fig. 2.B). It deploys a low-profile sequencer module composed of a 32-bit counter and a local configuration memory. To configure the PE, the controller makes use of a 64-bit control word that generates all the necessary control signals (depicted in Fig. 2.B) for: *i)* register control, pipeline masks, operand storage and input selection; *ii)* pre-processing module activation; *iii)* vector configuration; *iv)* VMA stage activations; *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. 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].

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

Authorized licensed use limited to: b-on: Universidade de Lisboa Reitoria. Downloaded on September 06,2020 at 06:27:23 UTC from IEEE Xplore. Restrictions apply.

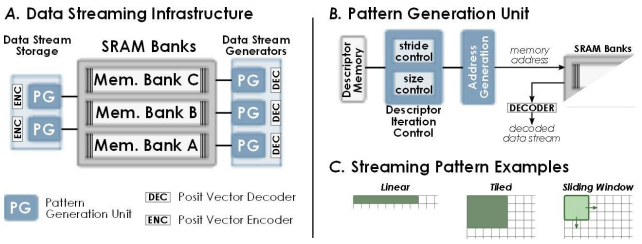

Fig. 6. Overview of the proposed RTU's data streaming infrastructure.

V. 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. 6).

Accordingly, the RTU's stream-based computation models are supported by a dedicated pattern generation unit per input (output) of each PE 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 6.A). 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.

Each data streaming pattern generation unit (see Fig. 6.B) adopts a descriptor format based on the affine function from Eq. 3. The descriptor format is capable of generating linear and tiled accesses by generating incremental *stride* factors (`stride` and `count control` modules - see Fig. 6.B), and adding them to a base address *offset* (`address generation` module - see Fig. 6.B 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. Finally, each unit is 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], [20].

VI. Experimental Results

This section evaluates the performance and energy efficiency of the proposed RTU when compared with alternative SIMD units deployed in off-the-shelf platforms.

A. Hardware Implementation

The proposed RTU architecture was fully designed for an Application-Specific Integrated Circuit (ASIC) implementation by considering the Nangate 45*nm* PDK. Although other configurations can 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 [8]. The supporting data-streaming infrastructure comprises 4 banked scratchpad memories (one per row of the array) each composed of three 8kB SRAM memories.

TABLE I
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 Infrastructure	12 PGs + Decode	0.324	0.397
	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

TABLE II
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.	AVX-512	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

TABLE III
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, FMA 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

Hardware resources and power estimation results were obtained with Cadence Genus 19.11 and the SRAM banks were generated with the OpenRAM [25] memory compiler.

The RTU was successfully synthesized with an operating frequency of 800 MHz. An area breakdown of each RTU component is presented in Table I, 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 resources 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.

B. Reference Setups and Workloads

To evaluate the RTU performance, it was compared with several off-the-shelf platforms featuring advanced SIMD units

Authorized licensed use limited to: b-on: Universidade de Lisboa Reitoria. Downloaded on September 06,2020 at 06:27:23 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
CLOCK CYCLE IMPROVEMENT (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	-	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.

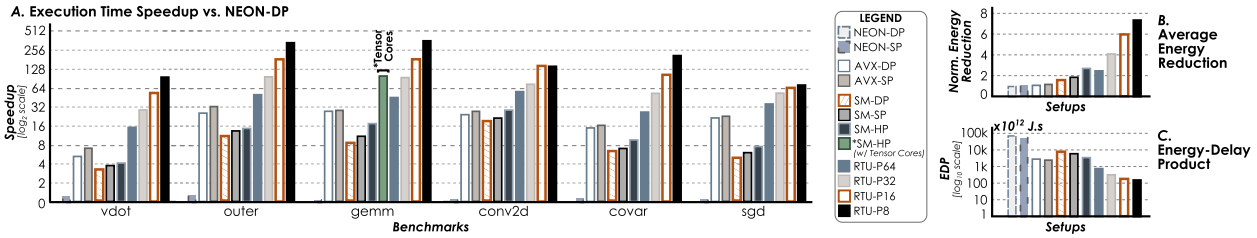

Fig. 7. Performance comparison results, including (A) execution time speedup and (B) average energy savings, both normalized to NEON-DP, and (C) overall energy efficiency (in the form of an EDP metric).

(see Table II), including: *i)* an Intel i7-8700K out-of-order processor (with the AVX-512 vector extension); *ii)* an ARM Cortex-A9 embedded processor (with the Neon vector extension); *iii)* a NVIDIA GV100 GPU (with tensor cores and native SIMD 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, SM-HP. For the proposed RTU, setups with 64-, 32-, 16-, and 8-bit Posit precisions were considered: RTU-P64, RTU-P32, RTU-P32, RTU-P8.

A set of benchmarks (characterized in Table III) 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` implement highly-parallel algebra operations; `gemm` and `conv2d` represent matrix-multiplication kernels that usually target 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 off-the-shelf SIMD units. As such, all applications were parameterized to target a single core of each platform (see Table II), with data sets that fit in the first level of the cache hierarchy (minimizing memory access delays). In the particular case of the GPU implementations, all benchmarks were dimensioned to target a ***single SM block*** and by allowing it to use its tensor cores (through the cuBLAS library) 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 by accessing internal performance counters with the PAPI library. For the GPU, measurements were obtained with the NVIDIA profiling tools.

C. Performance evaluation

Table IV and Fig. 7.A present the measured clock cycles and performance speedup for each benchmark and setup. 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.

Such architectural benefits are further highlighted when comparing the RTU execution time to the other setups, as it can be observed in Fig. 7.A. 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. 3.B), 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 the NEON-DP setups.

Given its base tensor-like computing architecture, the RTU was also compared with the tensor cores present in the NVIDIA GPU. By taking into account the restrictions imposed by NVIDIA tensor cores for the type and shape of matrix multiplications [11], it was only possible to map the `gemm` benchmark, denoting the lack of flexibility presented by these types of units. In fact, although the `conv2d` benchmark is also

Authorized licensed use limited to: b-on: Universidade de Lisboa Reitoria. Downloaded on September 06,2020 at 06:27:23 UTC from IEEE Xplore. Restrictions apply.

based on tensor multiplication, it adopts the most common 3x3 filter shape, which is not supported by the NVIDIA cores, in turn not allowing its mapping. Nevertheless, when comparing the execution of `gemm`, the proposed RTU using a 16-bit posit precision format is capable of matching and outperforming the NVIDIA tensor cores by 1.8x.

The RTU introduces an increased level of processing efficiency over the other setups by applying time-multiplexing reconfiguration to maximize its resource utilization. This is 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. 7.A). In particular, `covar` takes advantage of the RTU support to map high-latency arithmetic functions (in this case, a division). This is done by reconfiguring unused resources in the PE array, allowing the operation to be performed in parallel with other computations. When combined with a runtime array-wide reconfiguration between kernel phases and 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 data accuracy (depending on the dataset) [12]–[14], it is possible to observe 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 comparing a 45*nm* process (RTU) with much smaller technologies (28*nm*, 14*nm*, and 12*nm* - see Table II). 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 an increase would allow the RTU to attain further levels of acceleration when compared to the reference setups.

D. Energy Efficiency

The observed performance gains have a significant impact in the total energy consumption of the proposed RTU, as shown in Fig. 7.B. 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 than the NEON-DP setup. This is a direct result of the data streaming, 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, by up to 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, 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).

VII. 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, capable of fusing multiple PEs to map high-level operations, through time-multiplexing reconfiguration mechanisms. The obtained results for a 45*nm* ASIC implementation show that the proposed RTU provides an increased performance over existing state-of-the-art tensor and SIMD units present in off-the-shelf platforms, resulting in significant energy efficiency gains.

References

[1] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture." *Comms. of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[2] J. Dean *et al.*, "A new golden age in computer architecture: Empowering the machine-learning revolution," *IEEE Micro*, vol. 38, no. 2, pp. 21–29, 2018.

[3] N. P. Jouppi *et al.*, "A domain-specific architecture for deep neural networks," *Communications of the ACM*, vol. 61, no. 9, pp. 50–59, 2018.

[4] J. Fowers *et al.*, "A configurable cloud-scale DNN processor for real-time AI," in *ISCA*, 2018, pp. 1–14.

[5] E. Chung *et al.*, "Serving dnns in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[6] E. Delaye *et al.*, "Deep learning challenges and solutions with xilinx fpgas," in *ICCAD*, 2017, pp. 908–913.

[7] B. Reagen *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *ISCA*, 2016, pp. 267–278.

[8] NVIDIA, "Nvidia tesla v100 gpu architecture." *White paper.*, 2017.

[9] U. Köster *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *NIPS*, 2017, pp. 1742–1752.

[10] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017, pp. 1–12.

[11] S. Markidis *et al.*, "Nvidia tensor core programmability, performance & precision," in *IPDPSW*, 2018, pp. 522–531.

[12] J. L. Gustafson *et al.*, "Beating floating point at its own game: Posit arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, pp. 71–86, 2017.

[13] Z. Carmichael *et al.*, "Deep positron: A deep neural network using the posit number system," in *DATE*, 2019, pp. 1421–1426.

[14] R. Chaurasiya *et al.*, "Parameterized posit arithmetic hardware generator," in *ICCD*, 2018, pp. 334–341.

[15] P. W. Group, "Posit standard documentation," *Release 3.2*, Jun. 2018.

[16] T. Nowatzki *et al.*, "Stream-dataflow acceleration," in *ISCA*, 2017, pp. 416–429.

[17] R. Prabhakar *et al.*, "Plasticine: A reconfigurable architecture for parallel patterns," in *ISCA*, 2017, pp. 389–402.

[18] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE JSSC*, vol. 52, no. 1, pp. 127–138, 2016.

[19] N. Neves *et al.*, "Adaptive in-cache streaming for efficient data management," *IEEE TVLSI*, vol. PP, no. 99, pp. 1–14, 2017.

[20] M. K. Jaiswal *et al.*, "Architecture generator for type-3 unum posit adder/subtractor," in *ISCAS*, 2018, pp. 1–5.

[21] S. Ghosh *et al.*, "Cache miss equations: An analytical representation of cache misses," in *ICS*, 1997, pp. 317–324.

[22] T. Grosser *et al.*, "Polly-polyhedral optimization in llvm," in *IMPACT*, vol. 2011, 2011.

[23] F. De Dinechin *et al.*, "Posits: the good, the bad and the ugly," in *CoNGA*, 2019, p. 6.

[24] T. Viitanen *et al.*, "Simplified floating-point division and square root," in *ICASSP*, 2013, pp. 2707–2711.

[25] M. R. Guthaus *et al.*, "Openram: An open-source memory compiler," in *ICCAD*, 2016, pp. 1–6.

Authorized licensed use limited to: b-on: Universidade de Lisboa Reitoria. Downloaded on September 06,2020 at 06:27:23 UTC from IEEE Xplore. Restrictions apply.