A Generator of Numerically-Tailored and High-Throughput Accelerators for Batched GEMMs

Louis Ledoux
Barcelona Supercomputing Center (BSC)
Universitat Politecnica de Catalunya (UPC)
Barcelona, Spain
louis.ledoux@bsc.es

Marc Casas
Barcelona Supercomputing Center (BSC)
Universitat Politecnica de Catalunya (UPC)
Barcelona, Spain
marc.casas@bsc.es

***Abstract*—We propose a hardware generator of GEMM accelerators. Our generator produces vendor-agnostic HDL describing highly customizable systolic arrays guided by accuracy and energy efficiency goals. The generated arrays have three main novel aspects. First, the accelerators handle a large variety of computer number formats using intermediate representations based on our *Sign Scale Significand (S3)* format. Second, the processing elements perform all intermediate dot-product arithmetic operations required by the GEMM kernel without any intermediate rounding, which makes it possible to deliver better energy efficiency than state-of-the-art approaches while offering more accuracy and reproducible results. Third, our accelerators feature the Half-Speed Sink Down (HSSD) mechanism, which maximizes the overlap of host-accelerator data transfers with GEMM computations.**

We evaluate our automatically generated designs in a cutting-edge setup composed of a POWER9 host, CAPI (Coherent Accelerator Processor Interface) link, and a Virtex Ultrascale Plus FPGA. Arrays can operate at the speed of the link and saturate it to reach a 13GB/s throughput. Our fine-grain customization approach allows to cover a wide range of accuracy versus efficiency scenarios and can reach 0.65GOps/s/W while producing 1024 accurate bits or 148.7GOps/s/W with 6 accurate bits. Our configurations achieve up to 1 613GOps/s system performance and power efficiencies of up to 2 40GOps/s/W for the FPGA. This automatic generator is the first being able to produce such a variety of designs. We improve the single-precision energy efficiency of state-of-the-art FPGA GEMM accelerators by 1.86×.

I. Introduction

A large variety of scientific computing workloads rely on the GEneral Matrix-Matrix (GEMM) multiplication kernel. The requirements of these workloads in terms of numerical accuracy or performance are very heterogeneous. Indeed, emerging workloads such as Deep Neural Networks are naturally resilient to arithmetic tinkering [34], even though they embed millions of parameters [57] [25]. Conversely, ill-conditioned linear systems require much higher precision arithmetic [6] [68] [16] [26] [42] [10] [28]. Despite this large variety of accuracy requirements, commodity hardware just supports integers or 32- and 64-bits IEEE-754 arithmetic [1], which often results in a waste of computing resources.

To address these issues, computer number formats alternative to the IEEE-754 standard, like the *16-bit Brainfloat (Bfloat16)* [2] [35], the *Tapered Floating-Point (TFP)* [51] or the *Posit* [24], are attracting the attention of computer architects. There is a variety of previous works implementing arithmetic-logic units (ALUs) [33] [4] [59] and bare arithmetic data paths [8] [67] [60] for alternative formats, and comparing their area requirements to IEEE-754 units. In addition, some studies have analyzed the numerical stability of the Posit and the Bfloat16 arithmetics with respect to IEEE-754 [12] [7]. Despite this extensive amount of work, there are no approaches considering the area, performance, power, and accuracy trade-offs of these alternative computer number formats in the context of a real system implementing them. Also, there is no unified proposal leveraging the large flexibility of alternative number formats with respect to the bitwidth or the accumulator size, which covers an ample scope in terms of performance, accuracy, area, or power.

To handle these issues, this paper makes the following contributions beyond the state-of-the-art:

- The automated generation of arithmetic datapaths composed of *Sign Scale Significand (S3)*, a hardware-friendly floating-point format, in conjunction with extended or reduced internal precision accumulators. We propose a systolic array generator that exploits these automatically generated arithmetic datapaths to support a large variety of computer formats with customizable accumulators while producing reproducible results.
- The elimination of intermediate rounding, which provides large benefits in terms of energy efficiency, reproducibility, and accuracy.
- The Half-Speed Sink Down (HSSD) mechanism, a stalling-free output scheme that allows generated designs to maximize the use of processing elements and bandwidth.
- It provides a large evaluation campaign of our generated accelerators considering 64 different designs combining four different bitwidths, three accumulator sizes, and four computer number formats. We deploy the generated hardware on a high-end FPGA (Virtex Ultrascale Plus) attached to a POWER9 system via IBM CAPI link. This campaign exhibits the wide variety of scenarios covered by the generated designs. When combining small bitwidths with exact accumulators, our designs deliver $240GOps/s/W$ and display relative numerical errors below 2^{-4}. To the contrary, high-precision designs can deliver 1024 bits of accuracy and $0.65GOps/s/W$. Section V shows that our 32-bit designs deliver $19.5GOps/s/W$, which outperforms the FPGA state-of-the-art.

 DOI 10.1109/FCCM53951.2022.9786164

II. Related Work

A. Matrix Multiplication Systolic Arrays

Early systolic arrays originated during the 1980s [39] and were mostly used for convolutions [41], linear systems [38] and matrix multiplications [55]. Because it is not trivial to map a time-sequential algorithm into an ad-hoc space-time hardwired algorithm, previous works propose to formalize this process [40] [5] [53]. Systolic arrays' prominence and popularity rekindle due to the massive demands of computations brought by recent AI algorithms. In fact, commercialized ASICs for systolic arrays comprise cloud- and edge-deployed tapeouts with different array sizes. Notable examples are the Google's TPUs [22] for the cloud and Tesla FSD [43] for the edge.

Gemmini [20] is a systolic array generator targeting ASIC with a RISC-V host for edge devices. Gemmini generator has been utilized for the tapeouts of two System-On-Chips (SoC) in TSMC 16nm and Intel22FFL process technologies. The fact that this generator is written in Chisel, makes it data-type generic at the cost of suboptimal datapaths.

B. Arithmetic-level Optimizations

The IEEE-754 [1] 1985 standard describes various floating-point formats with different bitwidths: half-, single-, and double- precision numbers, which occupy 16, 32, 64 bits of storage, respectively. This small set of general-purpose datatypes can not optimally handle the plethora of specific numerical requirements proposed by modern workloads. This is the reason why scientists propose emerging arithmetics, ALUs, and even mathematical frameworks to describe real numbers [45].

Bfloat16 [61] is 16-bit wide and built from an IEEE-754 32-bit truncated from its LSBs and therefore shares identical dynamic range as its 32-bit counterpart. It is becoming a popular format with software [62] and hardware (FPGA [9] [19] [54], TPU [63], ARMv8 ISA [56]) support.

The Tapered Floating-Point (TFP) [51] format uses variable-sized entries to represent the mantissa and scale instead of the fixed-length entries of IEEE-754. We define TFPs$\langle \omega S, \omega F \rangle$ as numbers represented in scientific notations with a scale on ωS bits, a fraction on ωF bit, and a sign bit. We distinguish TFP from the IEEE-754 floating-point by its absence of denormal numbers.

The Posit number system [24] is a particular case of tapered formats that improves entropy per bit by trading exponent for mantissa storage bits via a Golomb-Rice [21] encoding. One posit configuration is usually refered as "$posit\ \langle N, es \rangle$", where N is the bitwidth, and es the variable length exponent size.

III. Systolic Array Generator

Our hardware generator of GEMM accelerators relies on systolic arrays. The GEMM accelerators can target any FPGA or ASIC because we describe RTL logic neither correlated with timings nor hardware target. In fact, we leverage *flopoco* [13] [32] for the back-end translation, which internally builds up timing graphs and inserts necessary and sufficient clocked elements.

A. Architecture of generated Accelerators

We define the generator parameters in a similar way as BLAS level-3 routines. Parameters n, m, and p correspond to the dimensions of the matrices involved in the GEMM operation, which we denote as A, B, and C.

$$n, m, p \in \mathbb{N}^+, \alpha, \beta \in \mathbb{R} \colon C_{n,m} \leftarrow \alpha A_{n,p} B_{p,m} + \beta C_{n,m}$$

Fig. 1 illustrates the overall architecture of a $3 \cdot 3$ array. Input data and control come from the bus on the left. Our architectures align data with clocked registers and translate them using *A2S3* units before data enter the actual array. Section III-C describes in detail the *A2S3* units. Once in the array, every translated numerical value is expressed in the *S3* hardware format, which is described in Section III-B2, and travels through the corresponding Processing Elements (PEs). Section III-C describes the *PEs* design. Start/End Of Block (*SOB* and *EOB*) are 2 bits generated by the main memory/host that control the array. *SOB* and *EOB* indicate when to flush to zero accumulated values and when accumulated values are valid to be carried out, respectively. This local and distributed control of the array allows efficient scaling. The last step is to round and normalize the output values via the normalization modules *L2A* to the output format. Section III-C describes *L2A* modules. Output and input formats can be different.

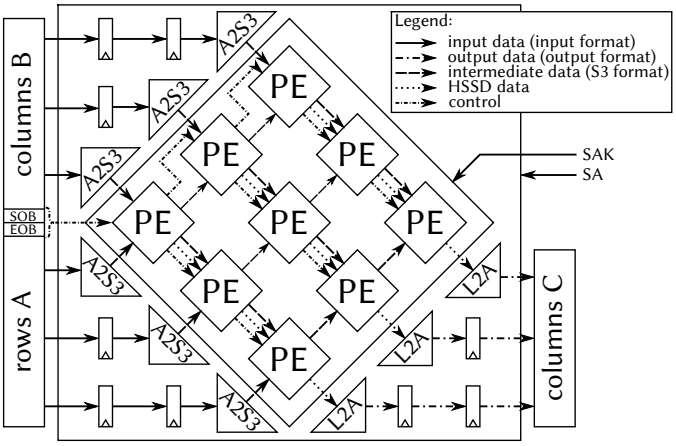

Fig. 1: Generated array with $3 \cdot 3$ Processing Elements.

To extract the in-place accumulated value from the output-stationary arrays, several options are possible with their respective trade-offs and requirements. Previous work [36] exposes two of them: having global lines (buses) from *PEs* to output buses to form semi-broadcast arrays; or reusing *PEs* internal registers while stalling the incoming data. None of the induced constraints are suitable for our needs since stalling the incoming pipeline would seriously harm computation and communication overlap and deliver poor performance. Furthermore, given that we aim to produce generic-sized and fast systolic arrays, we can not accept the unpredictable delays caused by high fan-outs yield by global lines, and therefore, we do not consider them.

To remedy this problem, our generator leverages a stalling-free output scheme for the output matrix C that allows extracting final and intermediate results through a propagating

TABLE I: Examples of common computer number formats with their corresponding *S3* translation.

Computer arithmetic	Value	$S3\ \omega S$	bias	$S3\ \omega F$	S3 quintuple
IEEE-754 16	0	5	$2^{\omega S-1}-1=15$	10	$\langle 0, x, xxxxx, 0, 0000000000\rangle$
posit $\langle 8,0\rangle$	1	4	$(N-2)\cdot 2^{es}=6$	5	$\langle 0, 0, 0110, 1, 00000\rangle$
posit $\langle 16,2\rangle$	NaN	7	$(N-2)\cdot 2^{es}=56$	11	$\langle 1, x, xxxxxxx, x, xxxxxxxxxx\rangle$
bfloat16	3.5	8	$2^{\omega S-1}-1=127$	7	$\langle 0, 0, 10000000, 1, 1100000\rangle$

chain called the Half-Speed Sink Down (HSSD). *HSSD* comes with a resource cost of $\mathcal{O}(n \cdot m \cdot \omega LA)$ registered elements, where ωLA is the bitwidth of the fixed-point accumulator. Since our architecture is output-stationary, it does not need to preload any input matrix. Therefore, combined with the *HSSD* chain that allows the output matrix to exit simultaneously, *PEs* are constantly computing and are fully used after the initial traversal of the array. The *HSSD* chain allows all links between *PEs* to be local, thus inducing no global routes except for the clock and the reset, which totally matches FPGAs' building blocks. We describe our *HSSD* design in Section III-C2 at the *PE* scale level to clearly show how output data is transferred out of the systolic array. Our *HSSD* design makes it possible for the systolic arrays to perpetually pump data from the input borders while simultaneously feeding the output border where the normalizing and rounding units reside, as we show in terms of dotted lines in Fig. 1. Moreover, due to the non-stalling pipeline introduced by *HSSD*, all transfers can be pipelined and overlapped to hide the latency of the systolic array, which directly translates into increased throughput. As a consequence of the *HSSD* extraction capability, matrix blocks can be sent as batches without interruptions to perform many independent GEMM operations.

The generated designs fuse dot-product arithmetic operations so that no intermediate rounding occurs. The reason to postpone the rounding step and shift it to the bottom of the array is twofold: On the one hand, it allows removing the expensive rounding logic from the $n \cdot m$ *PEs* since it is just required by the m *L2A* modules (see Section III-C4). On the other hand, deferred rounding retains the full accuracy of accumulated values by avoiding cancellations when scales are drastically different. Section III-C3 describes in detail this deferred rounding mechanism.

B. Generator Configurability

The generator automatically produces GEMM accelerators featuring a wide range of options in terms of accuracy, area, performance, and energy consumption. These are the main reconfiguration parameters:

1) Array Dimensions: The height (h) and the width (w) of generated arrays define the number of *PEs* ($number_{PEs} = h \cdot w$). The more the $h \cdot w$ product increases, the more operations per second (Ops/s) the array can deliver since every *PE* performs a fused-dot-product at every clock edge.

2) Arithmetic: There are three aspects of our generator that define the arithmetic of the accelerators:

Sign Scale Significant (S3) format: The *S3* format is a hardware-friendly floating-point format that maps well to FPGA internal basic elements. *S3* is composed of the following fixed-size fields that code the fixed and variable size of input numbers:

- Sign on 1 bit: active high when the number is negative.
- Scale on ωS bits: unsigned scale with a bias. The bias is calculated differently depending on the arithmetic.
- Significand on $1 + \omega F$ bits: an implicit bit prepended to the unsigned significand of the number. The 0 value is coded with implicit and significand set to zeros.
- NaN on 1 bit: Active high when the number is not a real number or when a problem occurs, such as overflows.

The fields form the $\langle NaN, sign, scale, implicit, fraction\rangle$ quintuple whose elements are concatenated in one bus of size $\omega S + \omega F + 3$ bits. *S3* can handle any computer number by just adjusting the $\langle NaN, sign, scale, implicit, fraction\rangle$ values. Table I provides examples illustrating how computer number formats can be expressed in terms of the *S3* encoding.

Accumulator size: *PEs* can be coupled with variable-size accumulators, from exact to light configurations. The following parameters, which are inspired by previous work [14], define the accuracy and the size of the numerical fixed-point accumulators that our designs contain:

- LSB: Is the weight of the Least Significant Bit of the accumulator. Adding bits to the right of the accumulator adds precision and controls the final accuracy. However, such bits are expensive because they augment the size of internal arithmetic modules (especially the barrel shifter, see Fig. 3).
- MSB: Is the expected weight of the summand. Similarly, lowering MSB can save hardware resources by decreasing the barrel shifter size.
- OVF: Is the number of carry bits (see Fig. 3 OVF) prepended to the left as MSBs in order to prevent overflows due to accumulation. The accumulator bitwidth is $\omega LA = OVF + MSB - LSB + 1$.

Normalizing units: Located at the bottom of the arrays, they translate the internal accumulations expressed in fixed-point format to the output computer number format. We propose three approaches to generate the rounding units:

- *Same*: A standard approach is to perform rounding and normalizing of the output values into the format they entered the systolic array.
- *Specific Arithmetic*: By taking advantage of the hardware reconfigurability, our generator can normalize the two's complement accumulations into a different arithmetic format than the input one. This has many applications, such as promoting to a bigger data type to avoid the final roundoff error, sending to another hardware that supports a specific type only, or even demoting to a lower data type to save bandwidth.
- *Exact*: This option does not perform the final normalization step and, does not require the *L2A* modules. The accelerators implementing this option send the final data as fixed-point.

C. Modules designs

This section describes the *Systolic Array* (SA) and *Systolic Array Kernel* (SAK) components (see Fig. 1).

1) A2S3: Denormalizing units are called *A2S3* and are the modules responsible for the conversion from an incoming arithmetic word to a *S3* quintuple. They take an N-bit bus and output a $\omega S + \omega F + 3$ bits bus. Any computer format

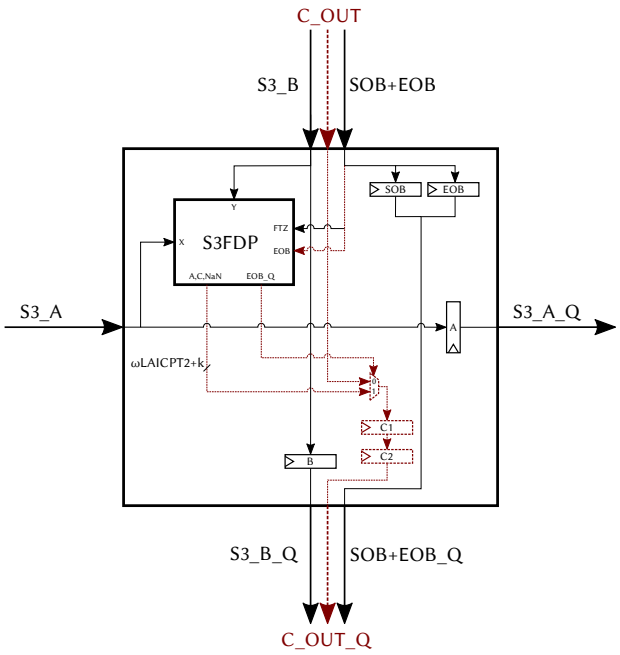

Fig. 2: Schematic view of a Processing Element (PE) with the vertical carry chain (HSSD) highlighted in red-dashed lines

can be translated to *S3* and work with the array as long as the corresponding *A2S3* unit exists.

2) Processing Element (PE): Our generator hardwired algorithm consists in four operations that *PEs* execute: First, register the incoming *S3s* values (S3_A and S3_B in Fig. 2) arriving respectively from the left and the right, to present them to their corresponding opposite border (right for S3_A_Q and bottom for S3_B_Q). Second, register the control signals and expose them to the opposite border (*SOB* and *EOB* from top to bottom). This allows a distributed and local control of the array and avoids global lines, which are often a problem. Third, compute in augmented precision without rounding the dot-product of incoming *S3s* values in an output-stationary fashion. This is done by a dedicated module discussed in Section III-C3. Fourth, extract via the *HSSD* mechanism the output-stationary values of C vertically with two registers *C1* and *C2* and a multiplexer, as Fig. 2 shows.

3) S3 Fused Dot Product (S3FDP): S3FDP is the kernel of our generator as it is the module that performs the fused-dot-product between values encoded with the *S3* format. Fig. 3 depicts the three main stages of the *S3* fused-dot-product. Firstly (Fig. 3 ①), the exact product is performed. Secondly (Fig. 3 ②), the mantissa of the product is aligned with regard to the accumulator weights depending on the scale of the product. Finally (Fig. 3 ③), the fixed-point addition is performed. The last operation requires peculiar timing-analysis. In fact, as fixed-point accumulators require as many bits as the dynamic range of the product contains binades, the carry-rippling can quickly become the critical path. If such, our generator automatically generates Carry-Save-Adders (CSAs) with *flopoco* timing analysis to suggest the chunk-size.

4) Normalizing modules (L2A): Their role is to translate the fixed-point values arriving from the *HSSD* chains to the specified output format. As mentioned in Section III-B2, the output format can vary and even be the fixed-point output coming from the S3FDP itself. The logic to normalize and round comprises the following main steps: First, to count Leading Zero/One (sign) to compute the scale of the accumulator. Second, to shift the accumulator by its scale to get the significand and a pre-sticky bit. Third, to build exponent fields based on the computed scale and the accumulator sign. The second and third steps are merged thanks to the *LZOCStickyShifter* operator from *flopoco*. And, fourth, to build the final word taking into account possible exceptions and applying a rounding scheme.

IV. Design Space Exploration

We carry out a design space exploration of the designs on a Virtex Ultrascale Plus (VUP) speedgrade-2 FPGA without loss of generality. Although the specific results of this design exploration apply to the VUP FPGA, the whole methodology can be applied to a generic FPGA. In fact, modern FPGAs share common properties and the utilization ratio between different building blocks is analogous.

A. Evaluated Designs

We consider four computer number formats: IEEE-754, Bfloat16, TFP, Posit. All of them are described in Section II. We consider four different bitwidths (8, 16, 32, and 64) for the IEEE-754 and the TFP formats, that is, 4 different variations for each format. For the case of Bfloat16 we consider the 16-bit width, since it is the only one it admits by definition. For the case of Posit, we consider the following 13 different formats: the $posit\,\langle 4,0\rangle$ 4-bit design, the $posit\,\langle 8,0\rangle$, $posit\,\langle 8,1\rangle$, $posit\,\langle 8,2\rangle$ 8-bit designs; the

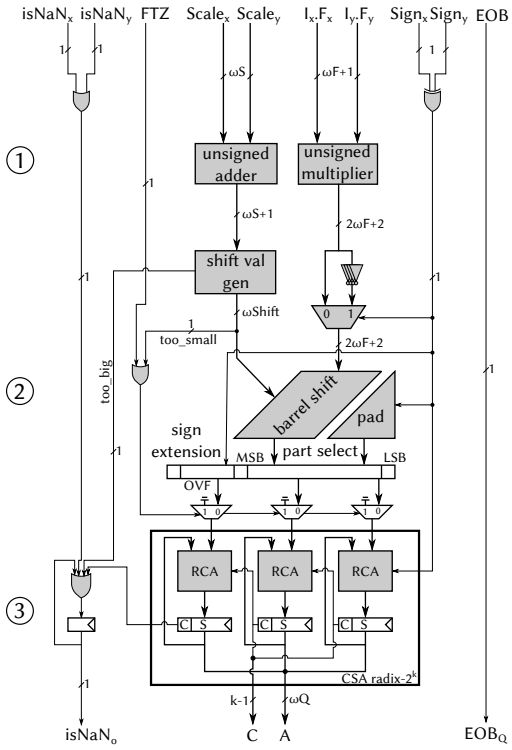

Fig. 3: Circuit design of a generic S3FDP

TABLE II: Area and power consumption of IEEE-754-$\alpha/\beta/\gamma$ S3FDPs and Xilinx FMA IP for VUP-2 FPGA @250MHz

Bitwidth	8-bit				16-bit				32-bit				64-bit			
Config.	α	β	γ	FMA	α	β	γ	FMA	α	β	γ	FMA	α	β	γ	FMA
LUTs	85	109	166	233	134	187	200	425	345	1039	416	769	1806	11414	1668	1785
FFs	18	42	102	342	34	124	102	680	225	865	317	1278	1383	8996	1089	2884
DSPs	0	0	0	1	1	1	1	1	2	2	2	2	6	6	6	10
CARRY8s	2	6	13	17	4	16	13	22	12	73	19	37	38	272	34	70
Power(PE)(W)	0.003	0.004	0.006	0.016	0.008	0.018	0.014	0.032	0.019	0.058	0.023	0.066	0.085	0.460	0.081	0.191

$posit\,\langle 16,0\rangle$, $posit\,\langle 16,1\rangle$, and $posit\,\langle 16,2\rangle$ 16-bit designs; the $posit\,\langle 32,0\rangle$, $posit\,\langle 32,1\rangle$, and $posit\,\langle 32,2\rangle$ 32-bit designs; and the $posit\,\langle 64,1\rangle$, $posit\,\langle 64,2\rangle$, and $posit\,\langle 64,3\rangle$ 64-bit designs. Our evaluation includes a total of 22 computers number formats.

Additionally, we propose three different accumulator configurations. We define them in terms of the parameters we describe in Section III-B2. We propose the following accumulator designs:

AI(α)*:* The α accumulator targets AI workloads. Since previous work [58] shows that neural networks weights values distribution are relatively narrow (e.g. within the $[-15, 21]$ interval for *Resnet50* and transformer models) and resilient to arithmetic tinkering [34], we set the msb parameter to be 5. We set the lsb parameter to build a small accumulator of factor-2 linear size with regard to the arithmetic bitwidth. We have the size $\omega LA = 2 \cdot N$, and $msb = 5, ovf = 2, lsb = 2 \cdot N - 1 - msb - ovf$

EXACT(β)*:* The β accumulator size is dictated by the size of the *S3* fields to ensure that any possible result of the exact multiplication can be represented in the accumulator without accuracy loss. The *Kulisch* [37] and *Quire* [46] accumulators land in this category.

CONSTANT_100(γ)*:* The γ accumulator configuration correlates neither with the arithmetic internal fields nor with any workload numerical specificity. The size $\omega LA = 1{+}9(ovf){+}40(msb){+}50(-lsb)$ allows all arithmetic to have the same internal accumulator, which enables comparisons to be made on other aspects besides the accumulator size.

We consider the three accumulators for each one of the 22 computer number formats we evaluate, except for the case of $posit\,\langle 4,0\rangle$ where we just consider the β accumulator since α and γ do not produce meaningful accuracy for this format. We have thus a total of 64 different combinations.

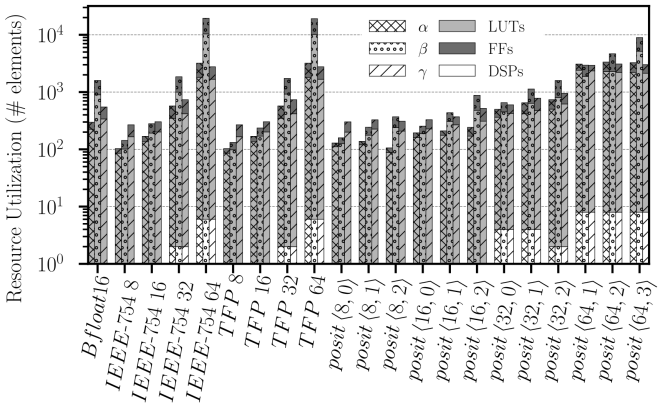

Fig. 4: Resource utilization of routed S3FDPs at 250MHz for VU3P-2 FPGA in logarithmic scale.

B. Hardware cost

1) Cost of S3FDP: We report in Fig. 4 how the resource numbers grow vis-à-vis the arithmetic and the accumulator configuration. These numbers are obtained after *place and route* with "Explore" directives through the generation of all design checkpoints. We can see how the bitwidth of the computer number format largely impacts the resource requirements of *S3FDP* units. Table II reports these numbers for the IEEE-754 format and bitwidths 8, 16, 32, and 64. For instance, the smallest IEEE-754 β does not use DSPs as all arithmetic datapaths are emulated with LUTs. To the contrary, the 53 bits multiplication and the 4228 bits addition of IEEE-754 64β require 6 DSPS, 272 CARRY8s with more than 10k LUTs. We also compare to α-, γ-*S3FDPs* and accumulators built on conventional FMA blocks infered thanks to Xilinx IPs. Our configurations always save resources against traditional FMA hardware, and thus, save power consumption. The only exception is the IEEE-754 64β which requires to maintain more than 4000 binades in fixed-point format.

2) Cost of PE, SAK and SA: The costs of *PE*, *SAK*, and *SA* arise from the cost of *S3FDP* components. The extra hardware added to an *S3FDP* to form a *PE* does not depend on the frequency and thus can be calculated without the use of tools. This extra cost comes from the delaying of input coefficients of A and B matrices (2 S3-sized registers), control delaying (2 Flip-Flops for *SOB* and *EOB*), and *HSSD* (accumulator-sized mux 2:1 and two accumulator-sized registers). Similarly, with the *PE* cost we can compute the cost of the *SAK* that contains $n*m$ *PEs* as follows: $cost_{SAK}(n, m, arith, ovf, msb, lsb) = n \cdot m \cdot cost_{PE}(arith, ovf, msb, lsb)$.

V. Evaluation

A. Experimental Setup

All experiments run on an AC922 POWER9 system with attached FPGAs as accelerators. The POWER9 system has 2 sockets (chips), each with 20 cores and 4 threads/core for a total of 160 threads running at $2.3GHz$. The FPGA is a xcvu3p-

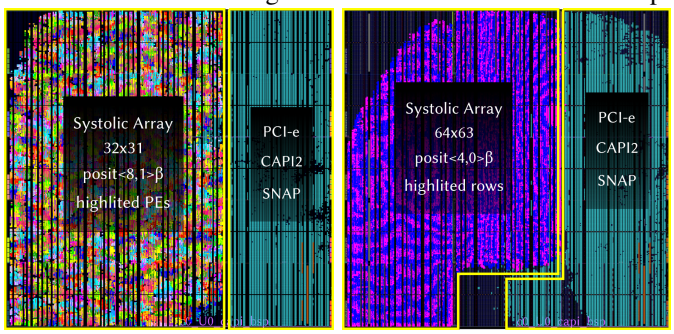

Fig. 5: Floorplaning of 32·31 and 64·63 arrays with additional PCIe/CAPI/SNAP logic (blue).

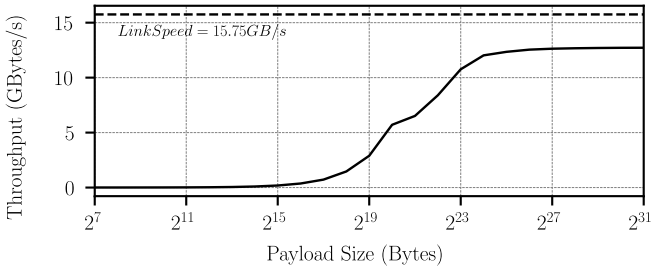

Fig. 6: Measured and averaged vs theoretical Throughput (GBytes/s) for different Payload Sizes (Bytes).

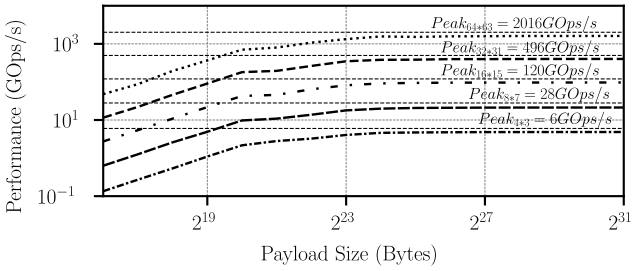

Fig. 7: Measured and averaged vs theoretical Performance (GOps/s) for different array sizes and Payload Sizes (Bytes).

ffvc1517-2 [3] [66] which embeds 384k LUTs, 788k FFs, 2280 DSPs, 25.3Mb of BRAM, and 90Mb of UltraRAM (URAM). The physical link between the host and the FPGA is PCIe gen4-8lanes, whose 16GT/s provide up to 15.754 GB/s [65]. All communications are done on top of the IBM Coherent Accelerator Processor Interface 2(CAPI2 [48] [47] [64]) protocol with the IBM SNAP framework [29]. To avoid Clock Domain Crossing (CDC), we generate all the arrays for an operating frequency of $250MHz$, which is the SNAP buses clock.

B. Performance, Energy Efficiency, Accuracy, and Throughput

We evaluate the throughput, performance, energy efficiency, and accuracy of generated designs by considering the 64 configurations we describe in Section IV-A running at $250MHz$. The number of PEs varies depending on the bitwidths: $64 \cdot 63$ for 4-bits, $32 \cdot 31$ for 8-bits, $16 \cdot 15$ for 16-bits, $8 \cdot 7$ for 32-bits, and $4 \cdot 3$ for 64-bit widths. For instance, Fig. 5 shows placed and routed arrays of 32·31 and 64·63 PEs with the additional communication logic, which represents 30% of the total floorplaning.

All metrics we consider depend on the matrix dimensions (see Eq. III-A) and the number of matrix batches (b). We propose a *Payload Size* metric, which combines both matrix dimensions and the number of matrix batches. This metric is defined in terms of Bytes as follows: $Payload\ size = N \cdot n \cdot m \cdot p \cdot b \cdot 2/8$. The Payload Size is equivalent to the *DMA* size. For throughput, performance, and energy efficiency, the number of accumulations per batch does not impact the results. However, accuracy is strongly dependant on the number of numerical accumulations per batch, as we discuss in Section V-B4. We display results setting $b = 1$, except if we precise the contrary. All displayed values result from an average of 10 runs. We generate the coefficients of our matrices by using a uniformly distributed random variable within the interval $[-1, 1]$.

1) Physical Link Throughput: Fig. 6 shows the average throughput achieved by the 64 arrays and the peak bandwidth offered by the 8-lanes PCIe gen4 link. The x-axis shows the payload size of the GEMM problem in terms of Bytes, and the y-axis shows the measured throughput in GB/s. All the measured throughput are averaged and plotted as one curve as the standard deviation is never more than 5% of the expected saturation throughput. The maximum measured throughput reaches $\sim 83\%$ of the peak link bandwidth, which corresponds to payload sizes of 2^{24}Bytes or larger. As previous work describes [44], the hardware PCIe protocol, which contains several layers and encoding schemes, makes it impossible to achieve the peak link bandwidth. In fact, our designs saturate data link bandwidth at their disposal. Fig. 8 compares throughputs at application level for different array sizes with and without our *HSSD* extraction chain. The experiment uses a constant payload size of 2^{30} Bytes with a decreasing number of batches b, and therefore, an increasing size of the common dimension p (shown by x-axis of Fig. 8). Without *HSSD*, and thus, without overlaping incoming and outgoing DMAs, an array stalling of n clock edges happens every p clock edges to be able to read out the data. This experiment exhibits that stalling the pipeline for small matrices hinders performances up to two order of magnitude.

2) Performance: Fig. 7 shows the performance achieved by the 64 designs in terms of GOps/s. The x-axis shows data payload in Bytes, and the y-axis displays performance in terms of GOps/s. Our experiments clearly show how the designs experiment 5 possible behaviors in terms of performance depending on the bitwidth. The reason why bitwidths define the performance behavior of the designs is the fact that the number of PEs in the array is defined by the data format bitwidth, as we explain in Section V-B. The designs achieve a performance peak of 4.8, 22.5, 96.7, 399.7, and 1613 GOps/s for 64-, 32-, 16-, 8- and 4-bit numerical formats, respectively. We observe these peaks for payload sizes equal to or larger than 2^{24}Bytes. They are very close the maximum theoretical values, which are 6, 28, 120, 496, and 2016 Gops/s for 64-, 32-, 16-, 8-, and 4-bit numerical formats, respectively, which correspond to arrays with $4 \cdot 3$, $8 \cdot 7$, $16 \cdot 15$, $32 \cdot 31$, and $64 \cdot 63$ PEs. The theoretical peaks are not achieved due to the physical link bandwidth saturation that we observe for payload sizes equal or larger than 2^{24}Bytes in Fig. 6.

3) Energy Efficiency: We report the rate of operations delivered by the system per watt of power consumed in Fig. 9. We also compare to energy efficiencies delivered by equivalent arrays but with intermediate roundings, thus, mimicking the behavior of traditional FMA hardware. The Figure is split into five categories corresponding to the five arithmetic bitwidths and array sizes that we evaluate. The most efficient design is the $64 \cdot 63$ array of $posit\ \langle 4, 0\rangle\ \beta$ PEs, which delivers $240GOps/s/W$. The 4 most performant α-designs are $posit\ \langle 8, 0\rangle\ \alpha$, IEEE-754 8α, TFP8α, and $posit\ \langle 8, 1\rangle\ \alpha$ that deliver energy efficiencies of 148.7, 146.6, 146.7, and 136.6 GOps/s/W, respectively, for payload sizes equal or larger

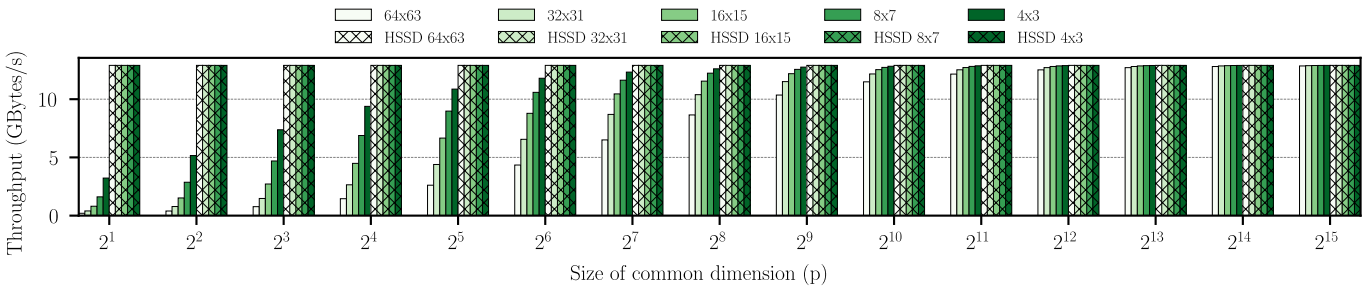

Fig. 8: Throughput comparison with and without HSSD for a workload of 2^{30} Bytes for different common dimension sizes (p).

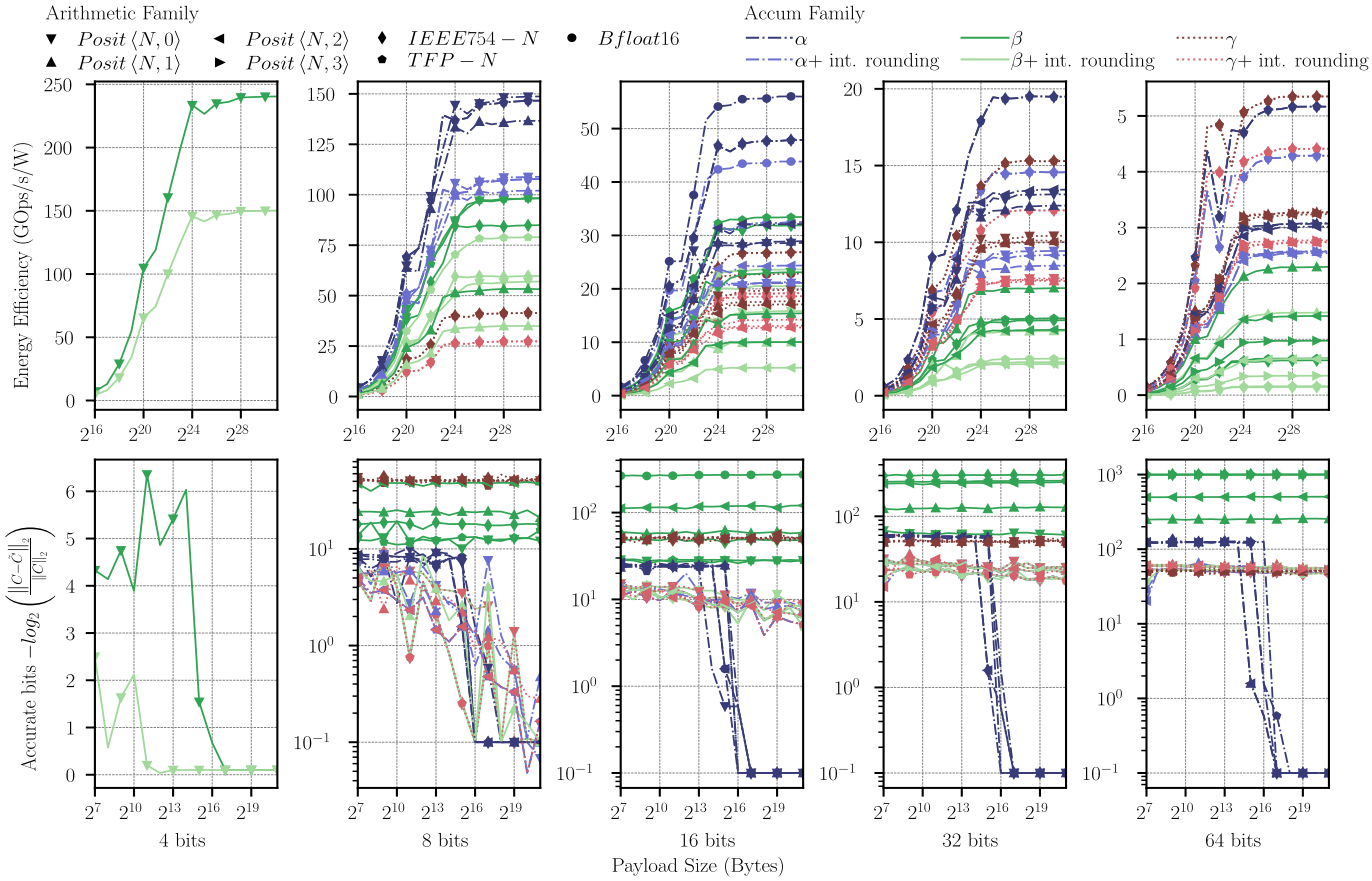

Fig. 9: Energy efficiency (top) against Accuracy (bottom) for different arithmetic bitwidths for different Payload Sizes (Bytes).

TABLE III: Comparison with previous related works. Box with"-" means that the information is not available.

work	year	device (process node)	freq (MHz)	% util.	HOST	link	best perf. (GOps/s) 64/32	energy eff. (GOps/s/W) 64/32	accurate bits (significant)	# data types	reproducibility (of computations)	generator	vendor agnostic
Guan [23]	2017	Stratix V(28nm) [11]	150	95	Xeon E5-2650v2 [31]	PCI-e Gen2 x8 [65]	-/~81	-/3.24	max. 23	2	no	**yes**	-
Moss [52]	2018	HARPv2(14/20nm) [17]	312.5	99	Xeon E5-26xxv2 [31]	QPI [30]	-/~99	-/~2.8	max. 23	8	no	no	no
de Fine Licht [15]	2020	VU9P(16nm) [66]	[146-190]	[69-90]	-	-	**122/409**	3.13/10.9	max. 52	6	no	no	no
this work	2021	VU3P(16nm)	250	[68-92]	POWER9 [50]	CAPI2 [49]	4.8/22.5	**5.3/19.5**	**1024**	**64**	**yes**	**yes**	**yes**

than 2^{28}Bytes. The corresponding β-designs, $posit\,\langle 8,0\rangle\,\beta$, IEEE-754 8β, and TFP8β, deliver maximum energy efficiencies of 98.3, 84.9, 98.2 GOps/s/W, respectively. The difference between the power efficiencies delivered by these two groups arises from the size of the accumulator, which significantly impacts the power cost of arithmetic operations. This trend is confirmed for other arithmetic bitwidths. For instance, all β accumulators of 32- and 64- bits arithmetics are less efficient than α and γ accumulators due to the quadratic scaling of internal operators with regard to the arithmetic bitwidth. The most efficient 16-, 32-, and 64-bit designs are TFP16α, TFP32α, and TFP64γ, which deliver maximum efficiencies of 47.9, 19.5, and 5.3 GOps/s/W, respectively. In terms of comparing the energy efficiency of the four formats we consider, we can see how TFP and IEEE-754 formats deliver very similar efficiency when using the same bitwidth, while posit can be either better or worse than TFP and IEEE-754 for the same bitwidth depending on the es parameter, which is defined in Section II. The energy efficiency of designs with *PEs* including rounding units is much worse than the ones without intermediate rounding. For example, IEEE-32β with rounding units reaches energy efficiencies up to $14.5GOps/s/W$, while its non intermediate rounding counterpart reaches $19.5GOps/s/W$. In fact, considering all the de-

signs, we observe an average loss of 70% of energy efficiency when the rounding unit is present in the accumulation step.

4) Accuracy: We report in Fig. 9 the final accuracy for different payload sizes. Workloads are composed of a single batch, so the number of numerical accumulations per PE is the same as the payload size. We compute the $log2$ of the relative error, which indicates the number of exact significant bits with respect to a high-precision oracle obtained with the mpfr library [18]. The oracle consists in the same computation made in software with a fixed-point of 1024 significant bits as fractional part, and 1000 bits to code the integer part.

We observe that the smallest accumulators (all α and $posit\,\langle 4,0\rangle\,\beta$) fail to maintain their initial accuracies while the number of accumulations increases. They are more statistically sensitive to overflows with respect to their larger counterparts, which can result in a NaN propagation (lines falling down in Fig. 9). All γ accumulators yield 50 accurate bits as it is equal to $-lsb$. These experiments show that the final rounding significantly impacts the accuracy as the internal precision can be lost to fit in a mantissa of general purpose size. We also observe how large accumulators are able to achieve good accuracy levels as accumulation counts increase. Some interesting 8 bits combinations are TFP8β and IEEE-754 8β, which sustain 10 and 15 bits of accuracy for all the considered accumulation counts and, according to Fig. 9, they deliver 98.2 and 84.9 GOps/s/Watt, respectively. More drastic combinations are $posit\,\langle 4,0\rangle\,\beta$ and TFP 64β. The TFP 64β always maintains the correct 1024 bits after millions of accumulated values while delivering $0.65GOps/s/W$. Conversely, $posit\,\langle 4,0\rangle\,\beta$ delivers $240GOps/s/W$ but tends to overflow after 2^{14} fused accumulations, which makes it lose its initial 4 accurate bits. The accurcy of designs with *PEs* including rounding units is much worse than the ones without intermediate rounding units. The most extreme cases are IEEE 64β, and any 64-bit-γ, where versions with intermediate rounding lose 971 and 0 accurate bits, respectively, with respect to their counterparts without intermediate rounding.

C. Comparison with the State-of-the-art

Table III displays the most relevant characteristics of previous approaches implementing hardware accelerators for GEMM. The numbers are extracted from the respective manuscripts. Table columns show the year of publication, FPGA device, frequency, FPGA resource utilization, host system, physical link, performance/energy-efficiency they achieve for 64- and 32-bits numerical formats, the maximum number of accurate bits, the number of computer arithmetics they can use, and other aspects like numerical reproducibility, automatic generation, or vendor agnosticism. The column "reproducibility" specifies whether accumulation order hinders vector-dot-product results.

Moss et al. [52] exploit the flexibility offered by the HPC hybrid *Xeon/Arria 10 HARPv2* to offer a customizable Matrix Multiplication framework that supports 8 distinct data types. They leverage only data types that map directly into the DSPs of the *Arria 10* target, and therefore support single-precision IEEE-754 and other fixed-point formats. The design is heavily specialized for Deep Neural Networks (DNNs) and delivers a performance of ~$99GOps/s$ and $2.8GOps/s/W$ energy efficiency for the IEEE-754 32-bit format.

Similarly, Guan et al. [23] specialize their matrix-multiplication for DNNs by proposing *FP-DNN*. It is the first work to ease the development of DNNs for FPGAs. Indeed, it generates hybrid RTL-HLS and host source code from the outputs of a trained *Tensorflow* [2] model. When using the IEEE-754 32-bit format, this design delivers ~$81GOps/s$ and an energy efficiency of ~$3.24GOps/s/W$.

De Fine Licht et al. [15] propose a high level synthesis (HLS) open-source approach for minimizing bandwidth requirements by augmenting the operational intensity. They report performances of $122GOps/s$ for the floating-point 64-bit, $409GOps/s$ for the 32-bit version, with energy efficiencies of $3.31GOps/s/W$, $5.3GOps/s/W$, respectively. The proposed designs use DDR4 modules to offload the matrices with DMAs between the HOST and the logic. This creates copies of the data in both directions (user space, kernel space, FPGA DDR4) and partially explains their lower energy efficiencies compared to ours. The authors only leverage the arithmetic data types proposed by Vivado HLS which comprise: *FP64, FP32, FP16, uint32, uint16 and uint8*. Additionally, since the authors use Vivado HLS, the floating-point types are not IEEE-754 compliant as they do not handle the subnormnals [27].

The fourth row of Table III corresponds to the generated designs running on the FPGA platform described in Section V-A. Sections V-B2 and V-B3 describe the performance and energy efficiency metrics present in this row. Our arrays deliver 19.5GOps/s/W when operating with 32-bit floating-point data types, which outperforms the single-precision energy efficiency of the best state-of-the-art FPGA-based GEMM accelerators by $1.86\times$. Thus, our generator is more energy-efficient than state-of-the-art approaches while offering more arithmetic reconfigurability in terms of number of number formats, data bitwidths, and accumulator configurations.

VI. Conclusions

Our array generator exploits hardware reconfigurability to produce GEMM kernels tailored to and dictated by specific numerical requirements. We conduct an extensive design space exploration and a system-wise evaluation, which show that our generator improves the state-of-the-art in terms of energy efficiency. The generated designs can retain accuracy with large problem sizes (millions of accumulations). Ranging from $240GOps/s/W$ with 4 accurate significant bits to $0.65GOps/s/W$ with 1024 accurate bits, we adequately generate drastically different configurations targeting many application domains.

This paper illustrates how the increasing size of reconfigurable hardware, along with emerging communication links, brings many opportunities beyond the IEEE-754 standard, which may produce a paradigm shift in terms of computer arithmetic and hardware heterogeneity.

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 955606
Marc Casas is supported by Grant RYC-2017-23269 funded by MCIN/AEI/ 10.13039/501100011033 and by "ESF Investing in your future"

REFERENCES

[1] "Ieee standard for binary floating-point arithmetic," *ANSI/IEEE Std 754-1985*, pp. 1–20, 1985.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *arXiv:1603.04467 [cs]*, Mar. 2016, arXiv: 1603.04467. [Online]. Available: http://arxiv.org/abs/1603.04467

[3] AlphaData, "ADM-PCIE-9V3 Support & Development Kit Release: 1.1.0 V1.1," p. 10, 2018.

[4] V. ArunkumarM, S. G. Bhairathi, and H. Hayatnagarkar, "PERC: Posit Enhanced Rocket Chip," 2020.

[5] H. Aso and Y. Inagaki, "Formal description of systolic algorithms and an analysis of the information flow," *Systems and Computers in Japan*, vol. 19, no. 6, pp. 14–24, 1988, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/scj.4690190602. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/scj.4690190602

[6] G. Beliakov and Y. Matiyasevich, "A Parallel Algorithm for Calculation of Large Determinants with High Accuracy for GPUs and MPI clusters," *arXiv:1308.1536 [cs, math]*, Aug. 2013, arXiv: 1308.1536. [Online]. Available: http://arxiv.org/abs/1308.1536

[7] N. Buoncristiani, S. Shah, D. Donofrio, and J. Shalf, "Evaluating the Numerical Stability of Posit Arithmetic," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2020, pp. 612–621, iSSN: 1530-2075.

[8] R. Chaurasiya, J. Gustafson, R. Shrestha, J. Neudorfer, S. Nambiar, K. Niyogi, F. Merchant, and R. Leupers, "Parameterized Posit Arithmetic Hardware Generator," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*. Orlando, FL, USA: IEEE, Oct. 2018, pp. 334–341. [Online]. Available: https://ieeexplore.ieee.org/document/8615707/

[9] J. Chromczak, M. Wheeler, C. Chiasson, D. How, M. Langhammer, T. Vanderhoek, G. Zgheib, and I. Ganusov, "Architectural Enhancements in Intel Agilex FPGAs," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 140–149. [Online]. Available: https://doi.org/10.1145/3373087.3375308

[10] S. Collange, D. Defour, S. Graillat, and R. Iakymchuk, "Full-Speed Deterministic Bit-Accurate Parallel Floating-Point Summation on Multi- and Many-Core Architectures," Feb. 2014.

[11] A. Corporation, "Stratix V Device Overview," p. 24, 2020.

[12] F. de Dinechin, L. Forget, J.-M. Muller, and Y. Uguen, "Posits: the good, the bad and the ugly," in *CoNGA 2019 - Conference on Next-Generation Arithmetic*. Singapore, Singapore: ACM Press, Mar. 2019, pp. 1–10. [Online]. Available: https://hal.inria.fr/hal-01959581

[13] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, Jul. 2011.

[14] F. de Dinechin, B. Pasca, O. Cret, and R. Tudoran, "An FPGA-specific approach to floating-point accumulation and sum-of-products," in *2008 International Conference on Field-Programmable Technology*. Taipei, Taiwan: IEEE, Dec. 2008, pp. 33–40. [Online]. Available: http://ieeexplore.ieee.org/document/4762363/

[15] J. de Fine Licht, G. Kwasniewski, and T. Hoefler, "Flexible Communication Avoiding Matrix Multiplication on FPGA with High-Level Synthesis," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Seaside CA USA: ACM, Feb. 2020, pp. 244–254. [Online]. Available: https://dl.acm.org/doi/10.1145/3373087.3375296

[16] R. K. Ellis, W. T. Giele, Z. Kunszt, K. Melnikov, and G. Zanderighi, "One-loop amplitudes for W+3 jet production in hadron collisions," *Journal of High Energy Physics*, vol. 2009, no. 01, pp. 012–012, Jan. 2009, arXiv: 0810.2762. [Online]. Available: http://arxiv.org/abs/0810.2762

[17] T. Faict, E. D'Hollander, and B. Goossens, "Mapping a Guided Image Filter on the HARP Reconfigurable Architecture Using OpenCL," *Algorithms*, vol. 12, p. 149, Jul. 2019.

[18] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, "MPFR: A multiple-precision binary floating-point library with correct rounding," *ACM Transactions on Mathematical Software*, vol. 33, no. 2, pp. 13–es, Jun. 2007. [Online]. Available: https://doi.org/10.1145/1236463.1236468

[19] I. K. Ganusov, M. A. Iyer, N. Cheng, and A. Meisler, "Agilex™ Generation of Intel® FPGAs," in *2020 IEEE Hot Chips 32 Symposium (HCS)*. Palo Alto, CA, USA: IEEE, Aug. 2020, pp. 1–26. [Online]. Available: https://ieeexplore.ieee.org/document/9220557/

[20] H. Genc, A. Haj-Ali, V. Iyer, A. Amid, H. Mao, J. Wright, C. Schmidt, J. Zhao, A. Ou, M. Banister, Y. S. Shao, B. Nikolic, I. Stoica, and K. Asanovic, "Gemmini: An Agile Systolic Array Generator Enabling Systematic Evaluations of Deep-Learning Architectures," *arXiv:1911.09925 [cs]*, Dec. 2019, arXiv: 1911.09925. [Online]. Available: http://arxiv.org/abs/1911.09925

[21] S. Golomb, "Run-length encodings (Corresp.)," *IEEE Transactions on Information Theory*, vol. 12, no. 3, pp. 399–401, Sep. 2006. [Online]. Available: https://doi.org/10.1109/TIT.1966.1053907

[22] Google, "System Architecture | Cloud TPU." [Online]. Available: https://cloud.google.com/tpu/docs/system-architecture

[23] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Apr. 2017, pp. 152–159.

[24] J. L. Gustafson and I. Yonemoto, "Beating Floating Point at its Own Game: Posit Arithmetic," p. 16.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015, arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385

[26] Y. He and C. DING, "Using Accurate Arithmetics to Improve Numerical Reproducibility and Stability in Parallel Applications," *Journal of Supercomputing*, vol. 18, Sep. 2000.

[27] J. Hrica, "Floating-Point Design with Vivado HLS," p. 13, 2012.

[28] R. Iakymchuk, D. Defour, S. Collange, and S. Graillat, "Reproducible and Accurate Matrix Multiplication for GPU Accelerators," Jan. 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01102877

[29] IBM, "CAPI SNAP Framework Hardware and Software. Contribute to open-power/snap development by creating an account on GitHub," apr 2019, original-date: 2016-09-20T20:10:34Z. [Online]. Available: https://github.com/open-power/snap

[30] Intel®, "An Introduction to the Intel® QuickPath Interconnect," 2019. [Online]. Available: https://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html

[31] Intel®, "Intel® Xeon® Processor E5 Family Product Specifications," 2021. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/series/59138/intel-xeon-processor-e5-family.html

[32] M. Istoan and F. de Dinechin, "Automating the pipeline of arithmetic datapaths," in *Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*, Lausanne, Switzerland, Mar. 2017. [Online]. Available: https://hal.inria.fr/hal-01373937

[33] R. Jain, N. Sharma, F. Merchant, S. Patkar, and R. Leupers, "CLARINET: A RISC-V Based Framework for Posit Arithmetic Empiricism," *arXiv:2006.00364 [cs]*, Jun. 2020, arXiv: 2006.00364. [Online]. Available: http://arxiv.org/abs/2006.00364

[34] J. Johnson, "Rethinking floating point for deep learning," *arXiv:1811.01721 [cs]*, Nov. 2018, arXiv: 1811.01721. [Online]. Available: http://arxiv.org/abs/1811.01721

[35] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey, "A Study of BFLOAT16 for Deep Learning Training," *arXiv:1905.12322 [cs, stat]*, Jun. 2019, arXiv: 1905.12322. [Online]. Available: http://arxiv.org/abs/1905.12322

[36] M. Khan, M.-C. Anisiu, L. Domoszali, A. Iványi, Z. Kasa, S. Pirzada, L. Szécsi, F. Szidarovszky, L. Szirmay-Kalos, and B. Vizvári, *Algorithms of Informatics, Volume III*, Sep. 2013.

[37] U. Kulisch and W. Miranker, "The arithmetic of the digital computer: A new approach," *SIAM Review*, vol. 28, no. 1, pp. 1–40, 1986. [Online]. Available: https://doi.org/10.1137/1028001

[38] Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 1, pp. 37–46, Jan. 1982, conference Name: Computer.

[39] H. Kung, C. Leiserson, C.-M. U. P. P. D. o. C. SCIENCE, and C.-M. U. C. S. Department, *Systolic Arrays for (VLSI)*, ser. CMU-CS. Carnegie-Mellon University, Department of Computer Science, 1978. [Online]. Available: https://books.google.fr/books?id=pAKfHAAACAAJ

[40] S.-Y. Kung, S.-C. Lo, and Lewis, "Optimal Systolic Design for the Transitive Closure and the Shortest Path Problems," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 603–614, May 1987, conference Name: IEEE Transactions on Computers.

[41] H. T. Kung'and and W. Song, "A Systolic 2-1) Convolution Chip," p. 15.

[42] G. Lake, T. Quinn, and D. Richardson, "From Sir Isaac to the Sloan Survey Calculating the Structure and Chaos Owing to Gravity in the Universe," Nov. 1996.

[43] F. Lamert, "Accelerated Mathematical Engine Tesla." [Online]. Available: https://www.scribd.com/document/398220774/Accelerated-Mathematical-Engine-Tesla

[44] J. Lawley, "Understanding Performance of PCI Express Systems," p. 16, 2014.

[45] P. Lindstrom, S. Lloyd, and J. Hittinger, "Universal coding of the reals: Alternatives to ieee floating point," in *Proceedings of the Conference for Next Generation Arithmetic*, ser. CoNGA '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3190339.3190344

[46] K. Mercado, "mightymercado/PySigmoid," jun 2020, original-date: 2017-11-03T14:30:27Z. [Online]. Available: https://github.com/mightymercado/PySigmoid

[47] T. P. Morgan, "Big Blue Aims For The Sky With Power9," Aug. 2016. [Online]. Available: https://www.nextplatform.com/2016/08/24/big-blue-aims-sky-power9/

[48] T. P. Morgan, "Opening Up The Server Bus For Coherent Acceleration," Oct. 2016. [Online]. Available: https://www.nextplatform.com/2016/10/17/opening-server-bus-coherent-acceleration/

[49] T. P. Morgan, "Opening Up The Server Bus For Coherent Acceleration," Oct. 2016. [Online]. Available: https://www.nextplatform.com/2016/10/17/opening-server-bus-coherent-acceleration/

[50] T. P. Morgan, "Power9 To The People," dec 2017. [Online]. Available: https://www.nextplatform.com/2017/12/05/power9-to-the-people/

[51] R. Morris, "Tapered floating point: A new floating-point representation," *IEEE Transactions on Computers*, vol. C-20, no. 12, pp. 1578–1579, 1971.

[52] D. J. Moss, S. Krishnan, E. Nurvitadhi, P. Ratuszniak, C. Johnson, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and P. H. Leong, "A Customizable Matrix Multiplication Framework for the Intel HARPv2 Xeon+FPGA Platform: A Deep Learning Case Study," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: Association for Computing Machinery, Feb. 2018, pp. 107–116. [Online]. Available: https://doi.org/10.1145/3174243.3174258

[53] J. J. Navarro, J. M. Llabería, and M. Valero, "Computing size-independent matrix problems on systolic array processors," in *Proceedings of the 13th Annual Symposium on Computer Architecture, Tokyo, Japan, June 1986*, H. Aiso, Ed. IEEE Computer Society, 1986, pp. 271–278. [Online]. Available: https://dl.acm.org/citation.cfm?id=17388

[54] B. B. Petrov, "Using of Bfloat16 Format in Deep Learning Embedded Accelerators based on FPGA with Limited Quantity of Dedicated Multipliers," in *2020 28th National Conference with International Participation*, oct 2020, pp. 82–85.

[55] P. Quinton, B. Joinnault, and P. Gachet, "A new matrix multiplication systolic array," p. 15, 1986.

[56] N. Stephens, "BFloat16 extensions for Armv8-A." [Online]. Available: https://community.arm.com/developer/ip-products/processors/b/ml-ip-blog/posts/bfloat16-processing-for-neural-networks-on-armv8_2d00_a

[57] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv:1409.4842 [cs]*, Sep. 2014, arXiv: 1409.4842. [Online]. Available: http://arxiv.org/abs/1409.4842

[58] T. Tambe, E.-Y. Yang, Z. Wan, Y. Deng, V. J. Reddi, A. Rush, D. Brooks, and G.-Y. Wei, "AdaptivFloat: A Floating-point based Data Type for Resilient Deep Learning Inference," *arXiv:1909.13271 [cs, stat]*, Feb. 2020, arXiv: 1909.13271. [Online]. Available: http://arxiv.org/abs/1909.13271

[59] S. Tiwari, N. Gala, C. Rebeiro, and V. Kamakoti, "PERI: A Configurable Posit Enabled RISC-V Core," *ACM Transactions on Architecture and Code Optimization*, vol. 18, no. 3, pp. 25:1–25:26, Apr. 2021. [Online]. Available: https://doi.org/10.1145/3446210

[60] Y. Uguen, L. Forget, and F. de Dinechin, "Evaluating the Hardware Cost of the Posit Number System," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. Barcelona, Spain: IEEE, Sep. 2019, pp. 106–113. [Online]. Available: https://ieeexplore.ieee.org/document/8892116/

[61] S. Wang, "BFloat16: The secret to high performance on Cloud TPUs." [Online]. Available: https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus/

[62] W. Wang and N. Hasabnis, "Distributed MLPerf ResNet50 Training on Intel Xeon Architectures with TensorFlow," in *The International Conference on High Performance Computing in Asia-Pacific Region Companion*, ser. HPC Asia 2021. New York, NY, USA: Association for Computing Machinery, Jan. 2021, pp. 29–35. [Online]. Available: https://doi.org/10.1145/3440722.3440880

[63] Y. E. Wang, G.-Y. Wei, and D. Brooks, "Benchmarking TPU, GPU, and CPU Platforms for Deep Learning," *arXiv:1907.10701 [cs, stat]*, Oct. 2019, arXiv: 1907.10701. [Online]. Available: http://arxiv.org/abs/1907.10701

[64] Wikipedia, "Coherent Accelerator Processor Interface," aug 2019, page Version ID: 912091796. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Coherent_Accelerator_Processor_Interface\&oldid=912091796

[65] Wikipedia, "PCI Express," Oct. 2020, page Version ID: 982964877. [Online]. Available: https://en.wikipedia.org/w/index.php?title=PCI_Express\&oldid=982964877

[66] Xilinx, "Virtex UltraScale+," 2016. [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html

[67] H. Zhang and S.-B. Ko, "Design of Power Efficient Posit Multiplier," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 861–865, May 2020, conference Name: IEEE Transactions on Circuits and Systems II: Express Briefs.

[68] T. Zhang, Z.-C. Yan, and G. W. F. Drake, "Qed corrections of $o(mc^2\alpha^7 \ln\alpha)$ to the fine structure splittings of helium and he-like ions," *Physical Review Letters*, vol. 77, no. 9, pp. 1715–1718, Aug. 1996, publisher: American Physical Society. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.77.1715