Design-Space Exploration of Mixed-precision DNN Accelerators based on Sum-Together Multipliers

Design-Space Exploration of Mixed-precision DNN Accelerators based on Sum-Together Multipliers

Luca Urbinati and Mario R. Casu
Dept. Electronics and Telecommunications, Politecnico di Torino, Turin, Italy, {luca.urbinati,mario.casu}@polito.it

***Abstract*—Mixed-precision quantization (MPQ) is gaining momentum in academia and industry as a way to improve the trade-off between accuracy and latency of Deep Neural Networks (DNNs) in edge applications. MPQ requires dedicated hardware to support different bit-widths. One approach uses Precision-Scalable MAC units (PSMACs) based on multipliers operating in Sum-Together (ST) mode. These can be configured to compute N = 1, 2, 4 multiplications/dot-products in parallel with operands at 16/N bits. We contribute to the State of the Art (SoA) in three directions: we compare for the first time the SoA ST multipliers architectures in performance, power and area; compared to previous work, we contribute to the portfolio of ST-based accelerators proposing three designs for the most common DNN algorithms: 2D-Convolution, Depth-wise Convolution and Fully-Connected; we show how these accelerators can be obtained with a High-Level Synthesis (HLS) flow. In particular, we perform a design-space exploration (DSE) in area, latency, power, varying many knobs, including PSMAC units parallelism, clock frequency and ST multipliers type. From the DSE on a 28-nm technology we observe that both at multiplier level and at accelerator level there is no one-fits-all solution for each possible scenario. Our findings allow accelerators' designers to choose, out of a rich variety, the best combination of ST multiplier and HLS knobs depending on the target, either high performance, low area, or low power.**

***Index Terms*—Variable-Precision Multiplier, Mixed-Precision DNN Accelerators, High-Level Synthesis.**

I. Introduction

Mixed-Precision Quantization (MPQ) quantizes DNN layers individually to find the best trade-off between accuracy and latency [1]. This method requires hardware with variable bit-width precision for weights and activations. Since the basic Deep Learning (DL) operations are matrix multiplications and convolutions, which in turn consist of scalar multiplications and dot products, various Precision-Scalable Multiply-and-Accumulate units (PSMACs) and DNN accelerators have recently emerged [2]. One approach to PSMACs is based on the so called Sum-Together (ST) multiplier [3], whose inputs usually pack $N = 1, 2, 4$ operands depending on the configuration, with precision inversely proportional to N (e.g., 16/N bits, Fig. 1(a)-(b)). They compute in one shot N multiplications in parallel and their low-precision products are *summed together* directly within the multiplier itself without requiring an external addition. In other words, they perform either a multiplication at full precision or a dot-product at lower precision. When used inside the PSMAC units of DNN accelerators, they can speed up the overall layer computation by a factor up to N [4].

In this work we contribute to the State of the Art (SoA) in three directions:

1) For the first time we compare the SoA ST multipliers in performance, power and area (PPA);
2) Compared to previous work, we enrich the portfolio of accelerators based on ST multipliers and propose three implementations for the most common layers: 2D-Convolution (2D-Conv), Depth-wise Convolution (DW-Conv) and Fully-Connected (FC).
3) We show how to obtain these accelerators with a High-Level Synthesis (HLS) approach and for each of them we identify the best hardware configuration knobs for a given PPA target. To this end, we report the results of a rich design-space exploration (DSE) varying many knobs, including parallelism, clock frequency and ST multiplier type.

II. Related Work

The most complete work on PSMACs is [2]. It extensively analyzes all types of SoA PSMAC architectures, including those based on ST multipliers, and classify them in subword-parallel, divide-and-conquer and bit-serial. Our work considers not only the ST multipliers selected in [2], i.e. the 16-bit Baugh-Wooley ST multiplier of [3] and the Fusion Unit of [5], but also the subword-parallel ST multipliers of [6] and [7], and the divide-and-conquer one of [8]. In particular, we first compare these ST multipliers among each other as independent blocks rather than as PSMAC unit subcomponents. Then, we show how to exploit the precision reconfigurability of these multipliers in the PSMAC units of three layer-specific DNN accelerators for inference, namely 2D-Conv, DW-Conv and Fully-Connected, while [3] did it only for a precision-configurable FC kernel and [5] for a general-purpose systolic array. Finally, we are not aware of any other works related to HLS techniques used to develop precision-scalable DNN accelerators based on ST multipliers, except our initial research [4].

III. Our DNN Accelerators with ST Multipliers

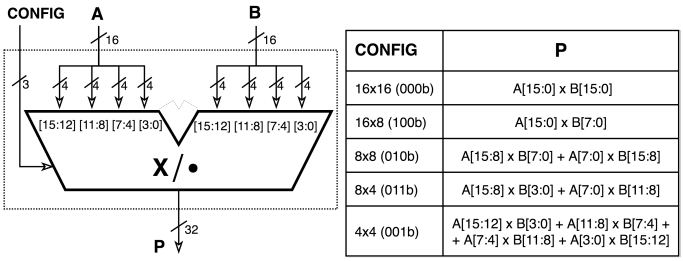
A. Working Principle

Although we refer to an ST multiplier with precision configurations as in Figs. 1(a)-(b), the same concepts can be easily extended to ST multipliers with other configurations.

The working principle of the three ST-based accelerators is explained in Fig. 1(c). The first column shows the number of activation/weight pairs (N) that the ST multiplier can accommodate in the 16-bit input operands, depending on the selected configuration. In the remaining three columns we sketch how the input (blue) and weight (orange) tensors are processed in 2D-Conv, DW-Conv and FC, respectively.

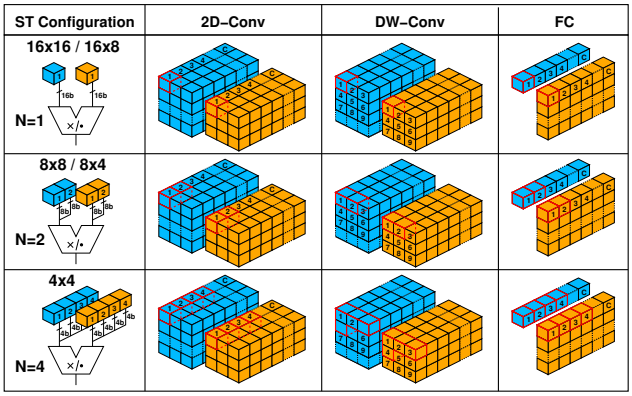
In 2D-Conv, for every orange filter with C kernels the C input tensor channels are multiplied with the corresponding

979-8-3503-0320-9/23/$31.00



CONFIG	P
16x16 (000b)	A[15:0] x B[15:0]
16x8 (100b)	A[15:0] x B[7:0]
8x8 (010b)	A[15:8] x B[7:0] + A[7:0] x B[15:8]
8x4 (011b)	A[15:8] x B[3:0] + A[7:0] x B[11:8]
4x4 (001b)	A[15:12] x B[3:0] + A[11:8] x B[7:4] + + A[7:4] x B[11:8] + A[3:0] x B[15:12]

(a) (b)



(c)

Fig. 1: Generic ST multiplier (a), its five configurations (b), and working principles of our ST-based DNN accelerators (c).

weight kernels and these partial results are added channel-wise. While at full precision ($N=1$) an ST multiplier processes one weight and one "pixel" of the input channel at a time, at reduced precision we can feed the ST multiplier with two ($N=2$) or four ($N=4$) pixel/weight pairs coming from the channels dimension, as reported in red in the second and third row of Fig.1(c). The number of MAC cycles scales as C/N and the corresponding latency as $1/N$.

A different approach is required for DW-Conv. In fact, every output channel of a depth-wise convolution is obtained by simply convolving every input channel with the corresponding weight kernel. Since there is no accumulation over the channel dimension, we can not use the ST multiplier as in 2D-Conv. As shown in the third column, depending on the configuration, we feed the ST multiplier with 1, 2, or 4 pairs coming from the receptive field of the input tensor and the corresponding weight kernel. As a result, the number of MAC cycles to compute an output "pixel" is $\lceil K^2/N \rceil$, where K is the kernel size ($K=3$ in Fig. 1(c)) [4]. Since K^2 is typically an odd number, while N is always even in low-precision configurations, the latency reduction is less than in 2D-Conv because the last iteration of the convolution between inputs and weights in a given channel will always lead to one or three spare elements (one for $N=2$ and three for $N=4$). This overhead decreases when the kernel size increases.

The working principle of the ST-based FC accelerator is shown in the last column of Fig. 1(c). It performs the matrix-vector product between the weight matrix and the linear array of input activations. Inspired by [3], the ST multiplier works

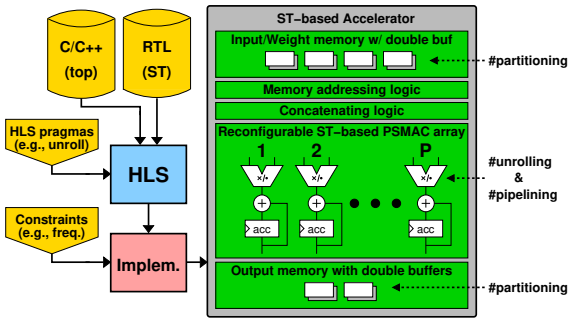


Fig. 2: HLS flow and architecture of the ST-based accelerators.

on the activations array and a weights row in order to produce an output activation of summed together low-precision products. The number of MAC cycles scales as C/N and the corresponding latency as $1/N$, as in the 2D-Conv case.

B. Architecture and HLS Flow

An overview of the general architecture of our ST-based DNN accelerators is in the grey and green block in Fig. 2. Different accelerators can have different size/shape of memories, behavior of addressing and concatenating logic, and PSMAC array parallelism (P). The entire set of variables that define the design space and their values is shown in Tab. I.

TABLE I: Design space variables and possible values.

Hw. Config. Knobs	**2D-Conv**	**DW-Conv**	**FC**
ST MUL type	[3], [6], [7], [8] ([5])		
IN_{CH/ACT}	4, 8, 16, 32	= OUT_CH	256, 512, 1024
OUT_{CH/ACT}	4, 8, 16, 32	1, 2, 4, 8, 16	8, 16, 32, 64
MAC array parallelism (P)	OUT_CH	OUT_CH	OUT_ACT
IN/OUT memory (shape)	18×18× ×{IN/OUT}_CH	22×22× ×OUT_CH	{IN/OUT}_ACT
WEIGHT memory (shape)	7×7×IN_CH× ×OUT_CH	5×5× ×OUT_CH	IN_ACT× ×OUT_ACT
Clock freq.	[100÷1000] MHz, 10 steps		

The PSMAC array operates on tensor tiles of dimensions defined by width, height, input and output channels (for DW- and 2D-Conv) or by input and output activations (for FC). The iteration over different tiles is handled by the host that invokes the accelerators. The array consists of P units, each composed by one (for DW- and 2D-Conv) or two (for FC [3]) ST multipliers, one adder, and one accumulation register. The P parallelism is one of the DSE variables and is connected to the number of output channels/activations, as clear from Tab. I. Thus, each PSMAC unit works on a different filter (2D-Conv), weight kernel (DW-Conv), or row of the weights matrix (FC) and iterates sequentially over input channels/activations, width and height of the tensor tile.

The Memory architecture consists of input, weight and output memories with double buffers. Input and weight memories are organized in 4-bit banks, to enable the data access patterns of the accelerators at low-precision configurations shown in Fig. 1(c), while the output memory is organized in 32-bit banks.

The size/shape of these memories is affected by the dimensions of the tensor tiles, which have been selected by analyzing the statistics of the layer shapes of the most common DNNs for edge devices, such as the various families of Mobilenet, EfficienNet, and ResNet, as shown in the table (e.g., width/height of 18 and 22, power-of-two ranges for the channels). The values that we selected for 2D-Conv and DW-Conv are a reasonable trade-off between their memory area and the number of iterations of the accelerators over multiple tensor tiles. For FC, we took the minimum values of IN_ACT/OUT_ACT from [3], while we chose the range with the same criteria used for the other accelerators.

The Memory Addressing and the Concatenating Logics implement the working principles of Sec. III-A according to the type of accelerator and selected configuration. The first one prepares the addresses to access input and weight memories and reads the proper four 4-bit data from each memory. The second one packs these data in the 16-bit operands of the ST multipliers and sign-extends the lower precision operands in the asymmetric configurations, i.e. 16×8 and 8×4.

The HLS flow shown in the left part of Fig. 2 generates the accelerator architecture, whose high-level C/C++ description is one the inputs of the flow (*C/C++ (top)* block). The other input (*RTL (ST)*) is one of the SoA ST multipliers described at Register-Transfer Level (RTL) [3], [6]–[8]. In fact, which ST multiplier to use in the MAC array is another knob of our DSE. These inputs are passed to Mentor Catapult for the high-level synthesis (*HLS* block); the resulting RTL is then passed to Synopsys Design Compiler for logic synthesis (*Implem.* block). The remaining yellow blocks are directives and constraints for the *HLS* and *Implem.* blocks. We pass *HLS pragmas* to the HLS tool to perform several optimizations. We set to 1.0 the *Initiation Interval* of the innermost loop that runs over the elements of the weight kernels, in order to pipeline the loop execution and increase the throughput of 2D-Conv and DW-Conv. We also set the *unrolling factor* of the output loops of the accelerators to P (OUT_CH or OUT_ACT, depending on the accelerator type), hence creating the P parallel PSMAC units of Fig. 2. We partition all the memories with the HLS directive *interleave* to allow parallel memory accesses to take full advantage of the parallelism created by the unrolling directive. The *Constraints* for the logic synthesis tool include the clock frequency, which we let vary in our DSE in Sec. IV.

C. SoA ST Multipliers Implementation

The SoA ST multipliers mentioned in Sec. II ([3], [6]–[8]) support a wide variety of heterogeneous precisions for input and weights. For a fair comparison we re-implement their core ideas in VHDL, slightly adapting their designs to meet our selected ST multiplier's configurations (Figs. 1(a)-(b)). All the selected ST multipliers are implemented with a structural RTL description, except [7] which is described behaviorally because the authors did same, too. Notice that [8] proposed seventeen years in advance a divide-and-conquer architecture very similar to the more recent [5]; therefore, we decide to give more credit to the former by implementing that version.

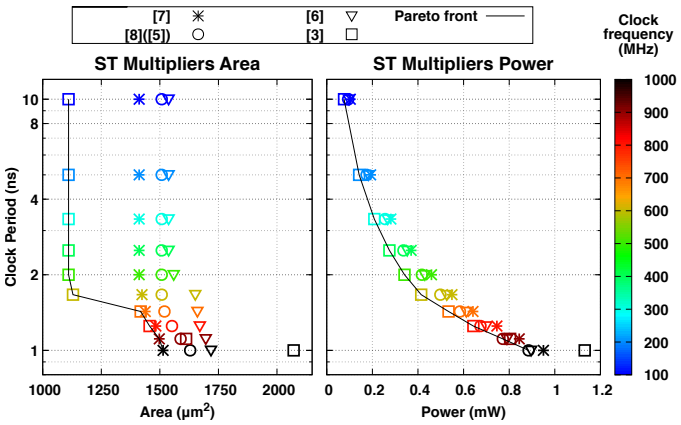


Fig. 3: DSE of the SoA ST multipliers.

IV. Results

A. PPA Comparison of ST Multipliers

We synthesized on a 28-nm technology at 0.9 V all the ST multipliers considered in Sec. III-C with input and output registers, and varied the target clock frequency from 100 to 1000 MHz in 10 steps. The PPA results of this DSE are reported in Fig. 3.

In the graph of area vs clock period, [3] performs well for frequencies lower than 600 MHz, with area less than 1250 μm^2, while [7] starts to be competitive between 700 MHz and 800 MHz and finally dominates at high frequencies. The reasons why [3] behaves poorly at high frequency, is the long diagonal critical path of the Baugh-Wooley full-adder array. As a result, the logic synthesis tool infers larger logic gates to meet tight timing constraints; on the other hand, the tool is not able to optimize the area of [7] for frequencies less than 700 MHz because its heuristics finds a solution that satisfies the desired clock period without particular effort. However, since the behavioral description of [7] allows the tool to freely choose the best implementation for the internal inferred multipliers and adders, the solutions at higher clock frequency use fast logic circuits that are more area efficient than [3].

In the graph of power vs clock period, [3] is the best up to 800 MHz. From that point onward, [6] and [8] triumph.

To conclude this first assessment, we observe that determining the best ST implementation in the PPA space is not straightforward because it depends on the design constraints.

B. Results of DSE of DNN Accelerators

We perform a DSE in area, latency and power for the three ST-based DNN accelerators, using the HLS flow described in Sec. III-B. For each type of accelerator we vary the hardware configuration knobs of Tab. I to synthesize a rich set of unique design points characterized by area, power, clock frequency, and ST multiplier type. To evaluate their latency when executing the most frequent layers from the selected DNNs of Sec.III-B, we first divide in tiles the input and output tensors of each selected layer such that the size of the tiles matches the accelerator's memory sizes. Then, for each implementation of an accelerator type, we evaluate the overall latency to

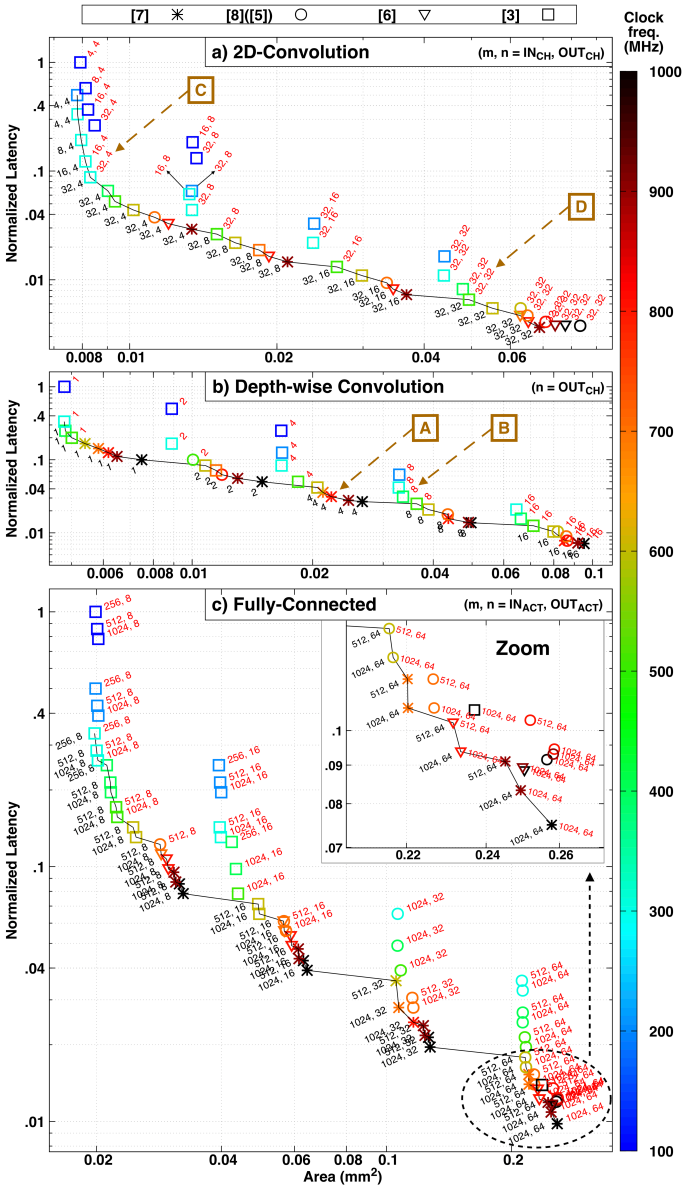


Fig. 4: Latency vs Area: results of DSE for (a) 2D-Convolution, (b) Depth-wise convolution, and (c) Fully-Connected accelerators. Points with black and red labels are Pareto points in Latency vs Area and Latency vs Power, respectively.

iterate over all these tiles, taking into account the double buffer mechanism that hides the tile transfer time. Finally, we normalize the results to make the results of the DSE layer-independent. We confirm our approach by doing the same test with other DNN layers, obtaining the same DSE trends for the three accelerators. Therefore the results that we report can be considered valid for any layer.

For each accelerator type, we project the Pareto design points from the tri-dimensional PPA space to two bi-dimensional spaces: Latency vs Area (LA) and Latency vs Power (LP). Fig. 4 reports these two projections on the same Latency vs Area plot due to space limitation. (A similar graph could be done for the Latency vs Power projection as well.)

The solutions connected by the black solid line and with black labels are LA-optimal (Pareto-optimal in the LA space), while those with red labels are LP-optimal, i.e. would sit on the Pareto front in the Latency vs Power graph (in the LP space). The labels indicate input/output channels for 2D-Conv, output channels for DW-Conv, or input/output activation pairs for FC.

Notice how the majority of the points that are optimal in LA are suboptimal in LP and vice versa. For example, an SoC designer planning to reserve an area of $0.03\,\mathrm{mm}^2$ for a DW-Conv accelerator could choose solution (A) with 4 channels optimized at 800 MHz, with normalized latency 0.03 and ST multiplier from [7]. However, for the same latency, the power-optimal point is solution (B) with 8 channels, optimized at 400 MHz, with ST multiplier from [3].

Indeed, there are few points that are both LA- and LP-optimal and sit on the Pareto front of both projections. For example, the low-area and low-power 2D-Conv accelerator named (C) with (32, 4) input/output channels pair at 300 MHz with IP [3], or—at the other end of the spectrum—the design (D) with very low latency at 500 MHz with ST multiplier from [3].

To conclude this second experiment, we confirm that the results at accelerator level are consistent with those of Sec. IV-A: there is no best solution for each possible scenario, but a rich variety that can satisfy the designers' constraints, from low area, to low power, or to high performance.

V. Conclusion

In this paper we have presented our contribution in the area of precision-scalable ST multipliers and mixed-precision DNN accelerators. We compared the ST multipliers of the literature and then we derived ST-based DNN accelerators with a high-level design flow, which allowed us to explore a large design space resulting from many solutions and architecture-level variables. Our results allow designers to select the best type of ST multiplier for the PSMAC unit of their accelerators in conjunction with the best configuration of hardware parameters for a given target in the PPA space.

References

[1] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient convnets through approximate computing," in *Proc. IEEE WACV*, 2016, pp. 1–8.

[2] V. Camus, L. Mei, C. Enz, and M. Verhelst, "Review and benchmarking of precision-scalable multiply-accumulate unit architectures for embedded neural-network processing," *IEEE JETCAS*, vol. 9, no. 4, pp. 697–711, 2019.

[3] L. Mei *et al.*, "Sub-word parallel precision-scalable MAC engines for efficient embedded dnn inference," in *Proc. AICAS*, 2019, pp. 6–10.

[4] L. Urbinati and M. R. Casu, "A reconfigurable depth-wise convolution module for heterogeneously quantized DNNs," in *Proc. IEEE ISCAS*, 2022, pp. 128–132.

[5] H. Sharma *et al.*, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *Proc. ACM/IEEE ISCA*, 2018, pp. 764–775.

[6] X. Zhang, Z. Li, and Q. Zheng, "Design of a configurable fixed-point multiplier for digital signal processor," in *Proc. IEEE PRIMEASIA*, 2009, pp. 217–220.

[7] M. Gautschi *et al.*, "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Trans. VLSI Syst.*, vol. 25, no. 10, pp. 2700–2713, Oct 2017.

[8] R. Lin, "Reconfigurable parallel inner product processor architectures," *IEEE Trans. VLSI Syst.*, vol. 9, no. 2, pp. 261–272, April 2001.