Preprint of paper from ISVLSI 2023, Brazil, June 2023.

tubGEMM: Energy-Efficient and Sparsity-Effective Temporal-Unary-Binary Based Matrix Multiply Unit

Prabhu Vellaisamy*†, Harideep Nair*†, Joseph Finn*, Manav Trivedi*, Albert Chen*, Anna Li*
Tsung-Han Lin†, Perry Wang†, Shawn Blanton*, and John Paul Shen*
*ECE, Carnegie Mellon University, {pvellais, hpnair, jafinn, mtrivedi, albertc1, ayli1, rblanton, jpshen}@andrew.cmu.edu
†MediaTek USA Inc., {prabhu.vellaisamy, harideep.nair, tsung-han.lin, perry.wang}@mediatek.com

***Abstract*—General Matrix Multiplication (GEMM) is a ubiquitous compute kernel in deep learning (DL). To support energy-efficient edge-native processing, new GEMM hardware units have been proposed that operate on unary encoded bitstreams using much simpler hardware. Most unary approaches thus far focus on rate-based unary encoding of values and perform stochastic approximate computation. This work presents *tubGEMM*, a novel matrix-multiply unit design that employs hybrid temporal-unary and binary *(tub)* encoding and performs exact (not approximate) GEMM. It intrinsically exploits dynamic value sparsity to improve energy efficiency. Compared to the current best unary design uGEMM, tubGEMM significantly reduces area, power, and energy by 89%, 87%, and 50% respectively. A tubGEMM design performing 128x128 matrix multiply on 8-bit integers, in commercial TSMC N5 (5nm) process node, consumes just 0.22 mm^2 die area, 417.72 mW power, and 8.86 μJ energy, assuming no sparsity. Typical sparsity in DL workloads (MobileNetv2, ResNet-50) reduces energy by more than 3x, and lowering precision to 4 and 2 bits further reduces it by 24x and 104x respectively.**

***Index Terms*—GEMM, temporal unary compute, sparsity**

I. Introduction & Background

General matrix multiplication (GEMM) is the primary compute kernel within the fully-connected and convolution layers of deep neural networks (DNNs). Though traditionally implemented as software libraries [6], dedicated GEMM hardware units have been introduced more recently, specifically in deep learning accelerators (DLAs) where the GEMM compute kernel is directly implemented as a matrix multiply unit [5]. This work focuses on a novel GEMM microarchitecture and direct CMOS implementation of low precision (2-8 bits) matrix multiply units targeting edge-native DLAs.

Lowering precision is fast becoming the de facto approach to increasing performance and decreasing energy consumption for DLAs. 16-bit formats for training are now widely used in industry. 16-bit accelerators are shown to provide 4x to 8x performance relative to 32-bit designs [11]. Further, authors in [11] report successful 4-bit based training on a collection of Deep Learning (DL) workloads with minimal accuracy loss. IBM researchers achieved 8-bit precision for training and 4-bit precision for inference across many DL datasets [12]. Ongoing research demonstrates the efficacy of pursuing low-precision compute to boost the hardware performance of current DLAs, with the potential of moving down to 4-bit and 2-bit compute while maintaining close to ideal inference accuracy.

Unary computing has been touted as a promising solution to building area and energy-optimized GEMM units [4]. This

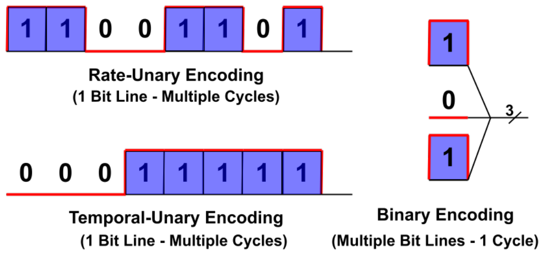


Fig. 1: Rate-Unary Encoding vs. Temporal-Unary Encoding vs. 3-Bit Binary Encoding for the value '5'

paradigm substitutes the need for multiple parallel bits in binary computing with serial unary bitstreams, resulting in significantly simpler and more energy-efficient hardware [1], [7], [9], [13]. Unary computing can manifest in two forms of encoding: *rate-based unary* and *temporal-based unary*; and two formats: *unipolar* (unsigned) or *bipolar* (signed) [13]. Data representation in rate-based unary coding is based on the frequency of 1s and 0s within a bitstream, with the number of 1s in the bitstream proportional to the data value. For temporal-based unary encoding, data is represented in the form of a sequence of 1s followed by a sequence of 0s, with the number of consecutive 1s representing the value. In contrast to temporal coding, rate coding results in approximate computation and suffers from the correlation problem [2]. Fig. 1 illustrates the three data encoding schemes of: 1) rate-based unary, 2) temporal-based unary, and 3) conventional binary.

Recent work on *unified*-unary GEMM design, uGEMM [13], depicts the state-of-the-art in all key hardware metrics. Although it supports temporal encoding, the underlying hardware is still rate-based targeting stochastic computing. While uGEMM focuses on *unified* encoding-agnostic hardware, our goal is to extract maximum area-power-energy efficiency by specializing the hardware for temporal encoding. Further, we believe, GEMM hardware targeting current DLAs should ideally perform *exact* compute with no accuracy loss.

In this work, we present *tubGEMM*, a highly efficient GEMM unit operating on temporal-unary and binary (tub) encoded values and using temporal processing hardware design, targeting deployment in low-precision edge-native DLAs. tubGEMM performs *exact* compute and does not incur accuracy loss like current unary stochastic GEMM hardware. It also utilizes a modified temporal encoding, termed *twos-unary*, to optimize compute latency with minimal hardware overhead.

The major contributions of this paper are as follows:

- We present a novel GEMM design, *tubGEMM*, employing hybrid *temporal-unary and binary* "tub" approach with a modified *twos-unary* temporal encoding.
- Unlike previous unary GEMM approaches that operate on *rate-unary* encoded bitstreams for approximate computing, tubGEMM performs exact computing.
- tubGEMM is 9x, 8x and 2x more efficient in area, power, and energy (i.e., achieves 89%, 87%, and 50% reduction) respectively than state-of-the-art unary design *uGEMM*, while incurring no inference accuracy loss.
- We perform rigorous industry-standard evaluation of tubGEMM's hardware complexity in collaboration with MediaTek USA Inc., using MediaTek's commercial design tools and the current TSMC N5 (5nm) process node. We scale the input matrix dimensions from 16x16 to 128x128 and precision from 2 bits to 8 bits, and report post-synthesis power-performance-area (PPA) results.
- We demonstrate that tubGEMM can naturally exploit data value sparsity on actual workloads (MobileNetv2 and ResNet-50) to dynamically improve its effective compute latency, power consumption and energy consumption.

II. Value Encoding and Processing

Traditional multiply-accumulate (MAC) units operate on binary-encoded inputs (Fig. 1) using bit-parallel multipliers and adders that deliver low latency at the cost of high area and power. Unary encoding with bit-serial processing is a viable alternative that tries to trade off latency for improved area and power. As shown in Fig. 1, it encodes values either stochastically in the frequency of randomly distributed ones within a bitstream (rate-unary encoding) or deterministically in the number of consecutive ones in a bitstream (temporal-unary encoding). Our work focuses on temporal-unary encoding in contrast to rate-unary encoding for three reasons: 1) it avoids costly random number generators, 2) it is capable of deterministic exact computation, and 3) it incurs lower dynamic power due to fewer edge transitions (only two transitions).

Analogous to Smith's temporal algebra [10], temporal-unary encoding here uses an n-cycle wide logic pulse to represent a value n, and can thereby consume up to a maximum of $(2^b - 1)$ cycles to represent a b-bit value and $(2^b - 1)^2$ cycles to represent multiplication result of two such b-bit values. To reduce this latency while still reaping the benefits of temporal-unary encoding, we use a hybrid *temporal-unary and binary* (tub) approach, similar to [14]. With only one of the inputs temporal-unary encoded and the other input kept in binary, the maximum multiplication latency is reduced from $(2^b - 1)^2$ cycles to $(2^b - 1)$ cycles. We further optimize this latency via a modified encoding as described in the next section.

III. tubGEMM Microarchitecture

In this section, we present our proposed *tubGEMM* microarchitecture (Fig. 2) for matrix-multiply unit design, which consists of an MxP array of processing elements (PEs) designed to perform: $\mathbf{Y} = \mathbf{A}$ x $\mathbf{B} + \mathbf{C}$, where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ are generic MxN, NxP and MxP input matrices, and $\mathbf{Y}$ is the MxP output matrix. Alongside the tubGEMM microarchitecture, we propose a modified temporal-unary encoding scheme, *Twos-Unary (2-Unary)*, that significantly reduces the latency with minimal hardware overhead. This novel encoding optimization is described in the following subsection.

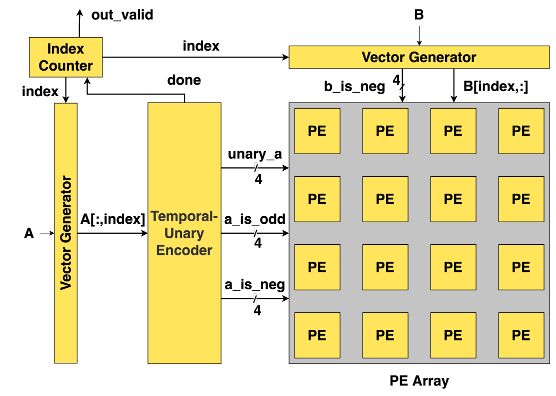


Fig. 2: 4x4 tubGEMM Architectural Block Diagram

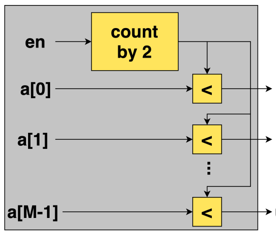
A. Twos-Unary Temporal-Unary Encoding

As discussed earlier, the hybrid *tub* approach incurs upto $(2^b - 1)$ cycles for a single multiplication, assuming each unary bit represents magnitude of 1. The encoding latency can be halved if each unary bit represents a magnitude of 2 instead. The only overhead required is a correction mechanism for odd values. This is done with no extra hardware as the existing adder can be reused to add one additional value in the last cycle (described further in the following subsections). With the *2-Unary* temporal encoding, the maximum latency is only (2^{b-1}) cycles, half of the original $(2^b - 1)$ cycle latency.

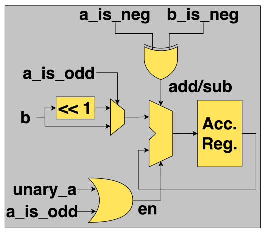
This encoding can be generalized to arbitrary n (n-*Unary*), where n is a power of 2. However, with higher powers of 2 (e.g., 4, 8), the correction mechanism requires an increasing number of shifters. Our exploration found that n=2 provides a good balance between additional complexity and latency reduction. Next, we describe how the input matrices are streamed into tubGEMM, utilizing the *2-Unary tub* approach.

B. Input Matrices Dataflow

Fig. 2 shows the block diagram for the proposed tubGEMM with MxP nodes in the PE array. It takes in temporal-unary $\mathbf{A}$ input from the left (after the binary to the temporal-unary encoder) and binary $\mathbf{B}$ input from the top. Input $\mathbf{A}$ arrives one column (M elements) at a time, and $\mathbf{B}$ arrives one row (P elements) at a time, in lockstep fashion. Both row and column are indexed identically within their corresponding matrices, i.e., k^{th} column of $\mathbf{A}$ arrives with k^{th} row of $\mathbf{B}$, and their outer product is computed. Thus, the computation occurs in N *steps* where each *step* performs a single *column-row* outer product ($\mathbf{A}$ has N columns, and $\mathbf{B}$ has N rows). The MxP PEs keep accumulating the MxP output values from the outer



(a) Temporal-Unary Encoder



(b) Processing Element

Fig. 3: tubGEMM components

product after each *step*, taking as many cycles as the magnitude of the maximum temporal-unary input value in every *step*. If we initialize the MxP PEs with **C** matrix values, it computes **Y** at the end of N *steps*. Other than the PE array (described in Section III-C), the proposed design has three components:

1) Index Counter: Each column of **A** is indexed simultaneously with the corresponding row of **B**. This is coordinated with the help of an *index counter* that counts from 0 to N, incrementing each time the *done* signal is asserted. Once the count reaches N, it asserts an *output_valid* signal, indicating the end of the matrix multiply computation.

2) Vector Generator: This component receives the index from the *index counter* and outputs a corresponding M-dimensional column of **A** and P-dimensional row of **B**. Two *vector generators* exist in the design - one each for **A** and **B**.

3) Temporal-Unary Encoder: As shown in Fig. 3a, it consists of a single *twos*-counter that counts up with increments of 2, and a set of M comparators that convert the M binary values within one column of **A** into M unary signals. Each comparator asserts a high signal at the output for as long as the binary value is greater than the counter value ($\lfloor \frac{n}{2} \rfloor$ cycles for binary value n). Since each unary cycle counts up by two, odd values need to have a correction factor, which is facilitated by asserting *a_is_odd* signal in the last cycle, sent to the PEs.

C. Multiply-Accumulate Processing Element

As shown in Fig. 3b, each node in the PE array is a MAC unit that multiplies a unary signal with a binary signal and adds the resulting binary output to the previously stored binary value. We refer to each element within **A** and **B** matrices as **a** and **b**, respectively. Akin to T-MAC [8], a sequential multiplier is used to accumulate the binary input for as many cycles as the unary input is asserted. We employ bipolar (signed) processing where both inputs **a** and **b** are assumed to be in twos-complement format (b-bits wide), with the maximum magnitude being 2^{b-1}. With the twos-unary encoding, a single multiplication operation can take up to 2^{b-2} cycles.

Additional mechanisms are employed to handle negative and odd inputs. *a_is_odd* signal controls a multiplexer, which outputs **b** when it is enabled; 2***b** otherwise. XOR of the most significant bits from **a** and **b** determines whether to add or subtract the output of the multiplexer to the accumulating register. The sequential multiplication is enabled through the OR of the unary input and *a_is_odd* signal.

TABLE I: 45nm Bipolar 8-bit Results for 16x16 matrices

GEMM Hardware	Area (mm^2)	Power (W)	Latency (us)	Energy (uJ)	Accuracy (%)
uGEMM	0.77	0.20	0.64	0.13	61.37
tubGEMM	**0.086**	**0.025**	**2.65 (WC)**	**0.066 (WC)**	**100**

TABLE II: 45nm Unipolar 8-bit Results for 16x16 matrices

GEMM Hardware	Area (mm^2)	Power (W)	Latency (us)	Energy (uJ)	Accuracy (%)
Gaines	1.55	0.49	0.64	0.31	70.45
Sim	0.50	0.15	0.64	0.09	70.93
uGEMM	0.44	0.12	0.64	0.07	100
tubGEMM	**0.057**	**0.012**	**5.29 (WC)**	**0.063 (WC)**	**100**

IV. Experimental Results

To perform fair comparisons with uGEMM [13], tubGEMM is first synthesized using NanGate45 Open Cell Library using Synopsys Design Compiler. tubGEMM is configured to the same dimensions in [13] with 16x16 input matrices, 8-bit precision, and the clock speed set to 400 MHz. Further, we extend the evaluation to get accurate 5nm PPA and energy results using commercial design tools and technology library used at MediaTek. The process design kit (PDK) and technology library used is the industry-standard TSMC N5 (5nm) process node, with Synopsys design tools employed for simulation, synthesis, and power calculations. The tubGEMM RTL design is first created in SystemVerilog, with functional verification performed using Synopsys VCS. Consequently, lint check is performed on the SystemVerilog source file using Synopsys SpyGlass and then synthesis is performed to convert the RTL-level design into a gate-level netlist using Synopsys Design Compiler, sourcing TSMC N5 library files. Gate-level simulation is then performed for netlist verification and switching activity collection in the form of a SAIF dump. The SAIF dump is then sourced along with the netlist to perform accurate power calculations using Synopsys PrimeTime PX.

The experimental results presented in the following four subsections include: 1) Comparison of tubGEMM vs. other unary GEMM designs including uGEMM, in 45nm CMOS; 2) Design space evaluation of tubGEMM in TSMC N5 by scaling the matrix dimensions from 16x16 to 128x128, and bit precision from 2 bits to 8 bits; 3) Assessment of the inherent potential of tubGEMM to exploit dynamic data sparsity; and 4) Evaluation of tubGEMM for two real DNN workloads.

Sections IV-A and IV-B derive latency and energy results based on the maximum possible cycle count for tubGEMM (2^{b-1} for unipolar and 2^{b-2} for bipolar, with b-bit inputs) and is denoted as worst-case (WC). The (significantly better) average-case latency and energy values for real DNN workloads are presented in Sections IV-C and IV-D.

A. tubGEMM vs. Other Unary GEMM Designs

We compare against Gaines [4], Sim [9] and uGEMM results for bipolar and unipolar non-scaled implementations as provided in uGEMM [13]. We illustrate tubGEMM's results for both bipolar (signed) and unipolar (unsigned) GEMM

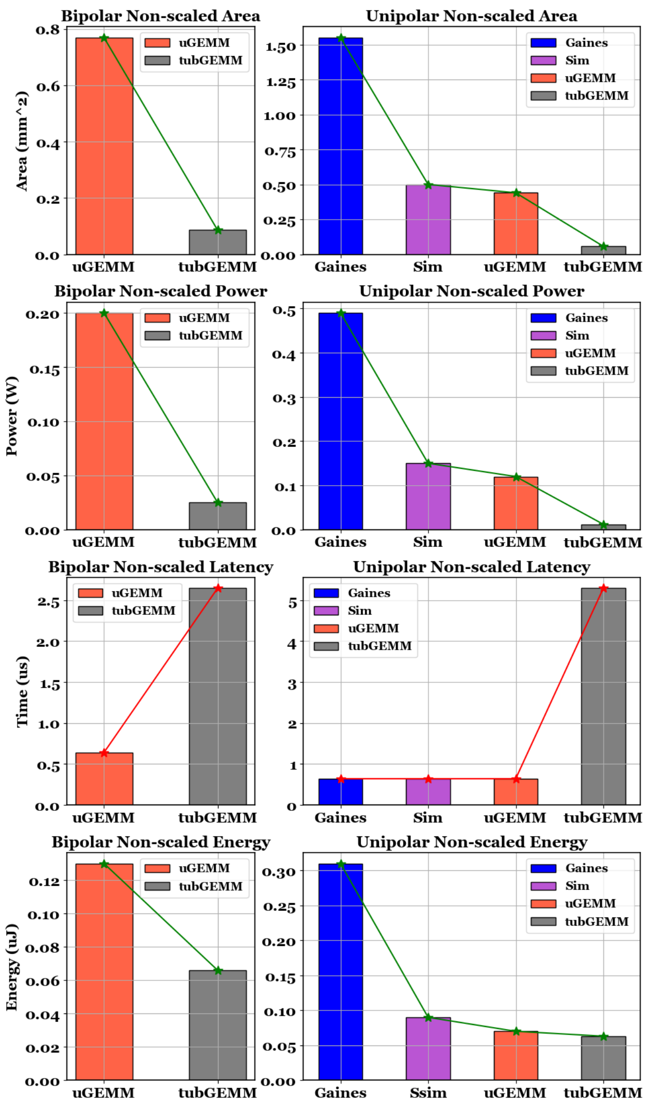


Fig. 4: uGEMM vs. tubGEMM 45nm post-synthesis WC PPA and energy values for bipolar and unipolar non-scaled GEMM

operations in Fig. 4, and the numbers are provided in Tables I and II, respectively, along with the computation accuracy values. As reported in Table I, bipolar tubGEMM outperforms bipolar uGEMM in all hardware metrics by 9x on area (89% less), 8x on power (87% less), and 2x on energy (50% less), while increasing latency by 4.1x. A key highlight here is that on top of being much more hardware efficient, tubGEMM achieves ideal accuracy as compared to uGEMM's reported accuracy of 61.37% for temporal coding.

Among the unipolar non-scaled designs (Table II), tubGEMM again provides better metrics except latency, which is double the bipolar non-scaled latency number. Sim [9], Gaines [4], and uGEMM [13] require 8.3x less latency compared to tubGEMM. However, the high latency is offset by the other metrics. To detail further, compared to uGEMM which is the best performing out of the existing unary approaches, tubGEMM is 7.7x, 10x, and 1.1x more optimized for area,

TABLE III: TSMC N5 post-synthesis tubGEMM PPA and energy values for various bit-widths and input matrix sizes

PE Array	Bit -width	Area (um^2)	Power (mW)	WC-Latency (us)	WC-Energy (nJ)
16 x 16	8-bit	2777.71	3.75	2.65	9.93
	4-bit	1275.12	1.66	0.25	0.42
	2-bit	692.50	0.91	0.13	0.12
32 x 32	8-bit	12560.52	21.80	5.30	115.55
	4-bit	5270.51	8.41	0.50	4.21
	2-bit	2804.12	3.19	0.26	0.83
64 x 64	8-bit	50384.67	85.13	10.60	902.33
	4-bit	22507.20	39.07	1.00	39.07
	2-bit	12108.43	16.10	0.52	8.37
128 x 128	8-bit	221689.65	417.72	21.20	8855.72
	4-bit	99030.84	209.91	2.00	419.82
	2-bit	53765.53	96.45	1.04	100.31

power, and energy consumption, respectively. Additionally, it maintains the ideal accuracy results demonstrated by uGEMM. These results show that tubGEMM can be a promising candidate for edge-native DLAs because of its extremely hardware-efficient implementation and low energy consumption. We address its one shortcoming of longer latency in Section IV-C.

Key Takeaway: Compared to current state-of-the-art uGEMM, tubGEMM can achieve significant improvements in area, power, and energy, even with its worst-case latency.

B. 5nm CMOS Scaling of tubGEMM

We depict tubGEMM's hardware complexity in 5nm CMOS as we scale input matrix (or MAC array) dimensions: 16x16, 32x32, 64x64, and 128x128, and bit precision: 8-bit, 4-bit, and 2-bit in Fig. 5 (y-axis is in log scale), with its corresponding post-synthesis numbers in Table III. As seen in the latency subplot in Fig. 5, latency scales linearly with matrix size and has a noticeable speedup upon transitioning from 8-bit to 4-bit precision due to an exponential decrease in cycle count.

1) Hardware Complexity Scaling with Increasing Input-Matrix Dimensions: The area, power, and energy consumption values scale nearly quadratically with an increase in the input dimension. This is equivalent to a linear increase with the number of PEs. With each doubling of one dimension of the input matrices, or MAC arrays, the tubGEMM design scales approximately 4.3x, 4.9x, and 9.8x for the area, power, and energy consumption on average, respectively.

2) Hardware Complexity Scaling with Decreasing Element Value Bit-width: Here, the input dimension size is set constant, and the scaling trend for decreasing bit-width precision is analyzed. On average, lowering from 8-bit to 4-bit compute decreases area, power, and energy by 2.3x, 2.4x, and 23.9x, respectively. The considerable reduction in energy from 8-bit to 4-bit is due to an 11x decrease in latency. Further, transitioning from 4-bit to 2-bit compute results in 1.8x, 2.2x, and 4.4x reduction in area, power, and energy, respectively.

Key Takeaway: The tubGEMM design for 8-bit 128x128 matrices (same as Google TPU v3) consumes only 0.22 mm^2 and 418 mW. Lowering the precision reduces the energy from 8,856 nJ (8-bit) to 420 nJ (4-bit) and 100 nJ (2-bit); this bodes well with the projected trend toward 4-bit and 2-bit DLAs [12].

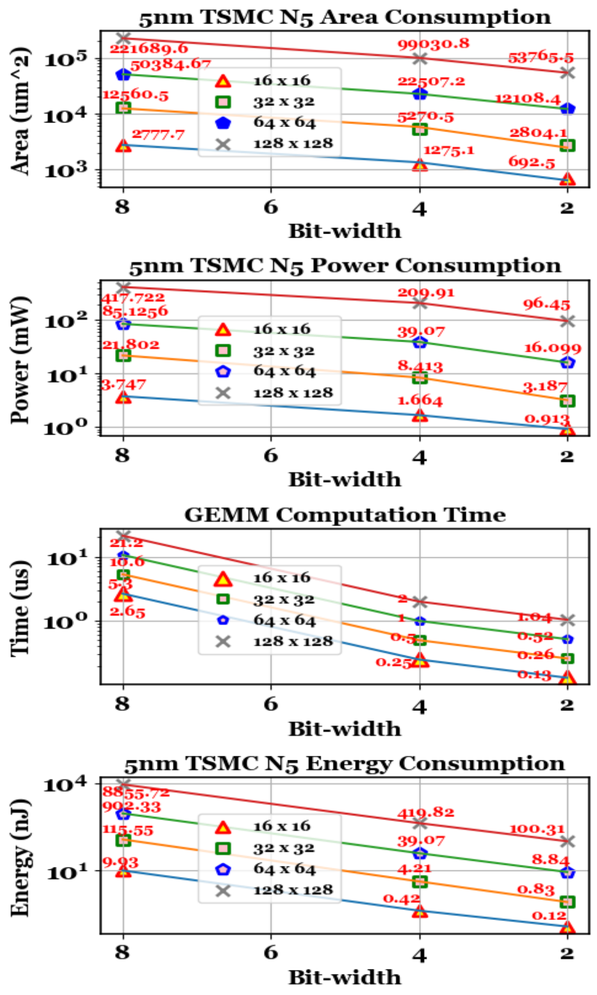


Fig. 5: tubGEMM TSMC N5 PPA and energy scaling, for input matrix dimensions of 16x16, 32x32, 64x64, and 128x128, and matrix element value bit-widths of 8 bits, 4 bits, and 2 bits

C. Dynamic Sparsity Evaluation

The tubGEMM latency values illustrated previously reflect the worst-case scenario, assuming each multiply involves the largest possible absolute value (127 for bipolar and 255 for unipolar with 8-bit values), thereby taking a proportional number of cycles to finish one compute. However, as a result of less frequently occurring large values and frequently occurring zeros in typical DL compute, tubGEMM can naturally leverage such bit-level and word-level value sparsities to significantly reduce the effective compute latency. Here, word-level sparsity refers to zero values in typical weights and activations, which has been widely explored in literature. On the other hand, bit-level sparsity here refers to a small number of ones followed by many zeros in the unary bitstream. As the value of a temporal-unary encoded signal is equal to the number of ones in the bitstream, bit-level sparsity directly results from small magnitude values. Bit-level sparsity subsumes word-level sparsity in the extreme case when all bits in the unary bitstream are zeros. Hence, we focus on bit sparsity here.

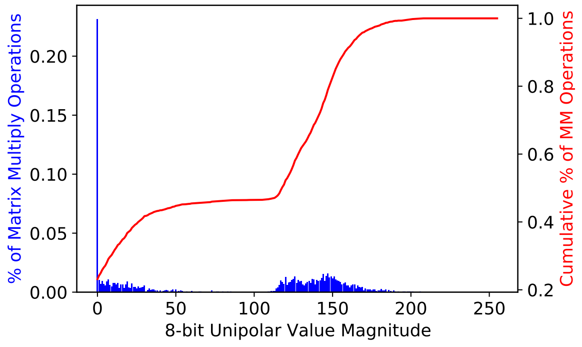


Fig. 6: Bit Sparsity: Percentage of matrix multiply operations that involve the corresponding X-axis values as the maximum magnitude during inference of INT8 unipolar Quantized MobileNetv2. Right Y-axis plots the cumulative percentage.

TABLE IV: 45nm Average-Case Latency, Energy and Energy-Delay Product for Unipolar Non-Scaled 16x16 tubGEMM based on Bit-Sparsity in INT8 Quantized MobileNetv2

16x16 GEMM Hardware	Latency (us)	Energy (uJ)	Energy-Delay Product (uJ-us)
uGEMM	0.64	0.070	0.045
tubGEMM (worst-case)	5.29	0.063	0.333
tubGEMM (average-case)	**1.72**	**0.021**	**0.036**

To profile bit sparsity, we perform inference on a representative DNN workload, namely, pre-trained unipolar INT8 quantized MobileNetv2, and keep track of the maximum values in every feature map within each convolutional layer. The underlying assumption is that each feature map computation can be packaged as a matrix multiply operation, whose latency is bounded by the maximum magnitude value in that feature map. Inference is performed using PyTorch, and the pre-trained model is taken from Torchvision library. During inference, we calculate the number of times every value between 0 and 255 (maximum magnitude for 8-bit unipolar compute) manifests as the maximum value within a feature map. This occurrence of maximum values derived as a percentage of the total number of matrix multiply operations per inference is averaged across 1000 randomly selected input images from ImageNet and plotted in blue in Fig. 6 along with the cumulative percentage in red. Fig. 6 clearly illustrates that close to 25% of the operations involve only zeros and thereby take zero cycles; any other value occurs less than 2% of the times as the maximum value (255 rarely occurs as the maximum value). Cumulatively, 90% of tubGEMM operations in MobileNetv2 involve maximum values smaller than 150.

From Fig. 6, the expected maximum feature map value can be calculated using the area under the blue curve, which gives 82. Based on this data, the average-case tubGEMM latency for MobileNetv2 comes out to be 1.72 us (3x reduction from 5.29 us), leading to a 3x reduction in energy as can be seen from Table IV. It can also be seen that the Energy-Delay Product (EDP) of tubGEMM reduces even below that of uGEMM

TABLE V: TSMC N5 Post-synthesis Power, Latency, Energy and Energy-Delay Product for representative INT8 DNN workloads, MobileNetv2 and ResNet-50, on 64x64 tubGEMM

Workload Type	Power (mW)	Latency (us)	Energy (uJ)	Energy-Delay Product (uJ-us)
Random (Table III)	85.13	10.60	0.90	9.54
MobileNetv2	49.43	5.54	0.27	1.52
ResNet-50	56.35	4.66	0.26	1.22

(1.25x improvement over uGEMM and 9.25x improvement over worst-case tubGEMM). This demonstrates the efficacy of tubGEMM in typical edge DL scenarios where much of the exponential overhead in latency is hidden due to the sparse nature of input data values (both weights and activations).

Key Takeaway: tubGEMM can implicitly exploit sparsity in data (both bit sparsity and word sparsity) to further improve all three metrics: power, latency, and energy. This dynamic form of data-dependent run-time optimization is built into the tubGEMM design. As a result, dynamic power is reduced, and the effective latency is much smaller than the worst-case latency, leading to a significant reduction in energy and EDP.

D. DNN Workload Analysis

In this section, we build on the bit sparsity analysis from Section IV-C and perform RTL hardware benchmarking with two real DL workloads, specifically, MobileNetv2 and ResNet-50, which are both INT8 quantized. Representative test vectors are generated from weights and activations in both these workloads for tubGEMM with 64x64 PE array and 8-bit precision, and are used to produce switching activity files for logic synthesis. The average compute latency per GEMM operation is derived through RTL functional simulation and then combined with post-synthesis power consumption to calculate workload-specific energy consumption and energy-delay product (EDP), which are presented in Table V. Note that 64x64 tubGEMM PE array size is used for evaluation (Table V) since it is a realistic configuration used in Edge Tensor Processing Units (TPUs) [3], as opposed to 16x16 array size used in Table IV to match with the baseline uGEMM.

The first entry in Table V corresponds to the 64x64 8-bit values from Table III, which are generated using uniform random test vectors. As can be seen from Table V, by leveraging dynamic activation and weight sparsity present within real MobileNetv2 and ResNet-50 workloads, tubGEMM is able to reduce power, latency, energy and EDP by 1.7x, 1.9x, 3.2x and 6.3x respectively for MobileNetv2, and 1.5x, 2.3x, 3.4x and 7.7x respectively for ResNet-50. In Sec IV-A, we illustrated tubGEMM's superiority over existing unary designs, including uGEMM, while still considering worst-case latency. This clearly demonstrates that tubGEMM, in spite of its long worst-case latency, can run real DNN workloads with much lower latency and very high energy efficiency.

Key Takeaway: Compared to worst-case latency, dynamic power and energy consumption incurred by tubGEMM are significantly lower for typical real DNN workloads due to high word-level and bit-level sparsity in weights and activations.

V. Concluding Remarks

We present a novel GEMM design, tubGEMM, that operates on temporal-unary encoded data, performs exact computation, and significantly reduces area, power, and energy (by 89%, 87%, and 50% respectively) relative to state-of-the-art unary GEMM design, uGEMM. We implement tubGEMM in 45nm as well as 5nm CMOS and show very promising PPA and energy results for scaling from 16x16 to 128x128 matrix multiply and for precision ranging from 8 bits down to 2 bits. tubGEMM can efficiently and naturally exploit data sparsity in DL computations at both word and bit levels, significantly reducing latency as well as dynamic power consumption. Typical sparsity in DNNs can potentially achieve 3x and 8x reduction in energy and EDP, respectively. This can facilitate energy-efficient online continuous learning for edge devices. We believe, tubGEMM, which can seamlessly ingest binary data (and convert to *tub* internally), can be widely adopted as a matrix multiply unit within low-precision edge-native DLAs. Our future work will extend the hardware design of the matrix multiply unit, focus of this paper, to system-level DLA design incorporating memory subsystem in TSMC 5nm process node.

References

[1] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transactions on Embedded computing systems (TECS)*, vol. 12, no. 2s, pp. 1–19, 2013.

[2] T. Baker and J. Hayes, "Impact of autocorrelation on stochastic circuit accuracy," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 271–277.

[3] S. Cass, "Taking ai to the edge: Google's tpu now comes in a maker-friendly package," *IEEE Spectrum*, vol. 56, no. 5, pp. 16–17, 2019.

[4] B. R. Gaines, "Stochastic computing," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 149–156.

[5] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017.

[6] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for fortran usage," *ACM Transactions on Mathematical Software (TOMS)*, vol. 5, no. 3, pp. 308–323, 1979.

[7] V. T. Lee, A. Alaghi, R. Pamula, V. S. Sathe, L. Ceze, and M. Oskin, "Architecture considerations for stochastic computing accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2277–2289, 2018.

[8] Z. Pan, D. Wu, and J. S. Miguel, "T-MAC: Temporal Multiplication with Accumulation," in *Young Architect Workshop (YArch)*, 2022.

[9] H. Sim and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[10] J. Smith, "Space-time algebra: A model for neocortical computation," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 289–300.

[11] X. Sun, N. Wang, C.-Y. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. V. Srinivasan, and K. Gopalakrishnan, "Ultra-low precision 4-bit training of deep neural networks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1796–1807.

[12] N. Wang, J. Choi, and K. Gopalakrishnan, "8-bit precision for training deep learning systems," 2018.

[13] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. S. Miguel, "Ugemm: Unary computing architecture for gemm applications," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 377–390.

[14] D. Wu and J. S. Miguel, "uSystolic: Byte-Crawling Unary Systolic Array," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2022.