Tempus Core: Area-Power Efficient Temporal-Unary Convolution Core for Low-Precision Edge DLAs

Prabhu Vellaisamy, Harideep Nair, Thomas Kang, Yichen Ni, Haoyang Fan, Bin Qi, Jeff Chen, Shawn Blanton, and John Paul Shen
ECE Department, Carnegie Mellon University
{*pvellais, hpnair, thomaska, yichenn, hfan2, binq, jhchen2, rblanton, jpshen*}*@andrew.cmu.edu*

***Abstract*—The increasing complexity of deep neural networks (DNNs) poses significant challenges for edge inference deployment due to resource and power constraints of edge devices. Recent works on unary-based matrix multiplication hardware aim to leverage data sparsity and low-precision values to enhance hardware efficiency. However, the adoption and integration of such unary hardware into commercial deep learning accelerators (DLA) remain limited due to processing element (PE) array dataflow differences. This work presents *Tempus Core*, a convolution core with highly scalable unary-based PE array comprising of *tub* (temporal-unary-binary) multipliers that seamlessly integrates with the NVDLA (NVIDIA's open-source DLA for accelerating CNNs) while maintaining dataflow compliance and boosting hardware efficiency. Analysis across various datapath granularities shows that for INT8 precision in 45nm CMOS, Tempus Core's PE cell unit (PCU) yields 59.3% and 15.3% reductions in area and power consumption, respectively, over NVDLA's CMAC unit. Considering a 16x16 PE array in Tempus Core, area and power improves by 75% and 62%, respectively, while delivering 5x and 4x iso-area throughput improvements for INT8 and INT4 precisions. Post-place and route analysis of Tempus Core's PCU shows that the 16x4 PE array for INT4 precision in 45nm CMOS requires only 0.017mm^2 die area and consumes only 6.2mW of total power. We demonstrate that area-power efficient unary-based hardware can be seamlessly integrated into conventional DLAs, paving the path for efficient unary hardware for edge AI inference.**

***Index Terms*—NVDLA, GEMM, Temporal-Unary, Convolution MAC core, Low Precision, CNNs**

I. Introduction

Since the introduction of AlexNet [1] in 2012, AI computational demands have grown rapidly [2], driving the adoption of hardware accelerators like GPUs and TPUs for deep learning (DL) training and inference. However, DL's computational needs have quickly outpaced the growth of available compute power, leading to what is now termed the "AI compute gap". This gap has renewed interest in innovative computational techniques and architectures aimed at closing this compute performance and energy efficiency gap.

In recent years, quantization techniques have become an effective strategy for deploying AI models with reduced model complexity. While 32-bit floating point (FP32) was the standard for DL training, FP16 is now widely adopted, offering 4x-8x performance improvements [3]. Building on this trend, FP8 precision for both training and inference has been proposed [4] and implemented by Nvidia in their H100 GPUs, which support FP8 models through the Transformer

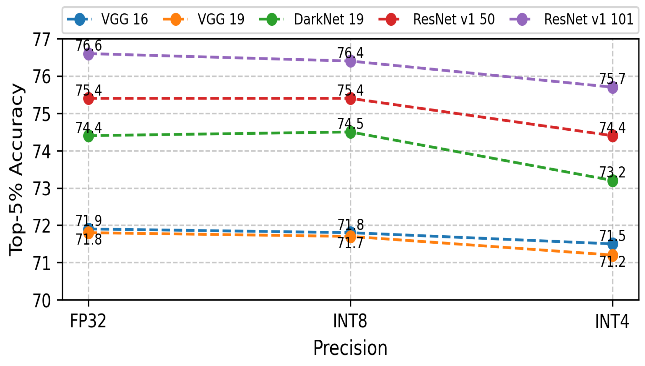

Fig. 1. Quantization training accuracies achieved on different ImageNet CNNs for different integer-based precisions when compared to baseline FP32 precision [8]. Results show minimal accuracy decrease with lower precisions.

Engine library [5]. In particular, FP8 training on various convolutional neural network (CNN) models has shown minimal accuracy degradation while increasing throughput by 2x-4x [3]. In addition to floating point formats, 8-bit integer (INT8) training has been demonstrated to reduce the training time of CNNs on Pascal GPUs by 22% [6]. Although INT8 has become the standard for DL inference, a shift toward INT4 precision has led to a 77% performance improvement over INT8 [7]. The results of ImageNet trainings for quantized CNN models and their corresponding Top-5% accuracy are presented in Fig. 1, showing minimal accuracy degradation for INT4 quantized models compared to their baseline FP32 implementations. These quantization results offer tremendous promise particularly for edge inference deployment, where computational resources are limited.

Unary computing has recently emerged as a promising paradigm for low-precision AI, enabling highly area- and power-efficient hardware by trading off latency for efficiency. In particular, temporal-unary techniques have been shown to significantly reduce computational overhead in matrix multiplication units, as demonstrated by recent works [9] [10] [11], which exploit the inherent sparsity of DL models. Further, it performs deterministic compute, realizing output without accuracy degradation. However, existing implementations focus primarily on GEMM-level designs and are not scaled to inherently support convolution dataflows. This work addresses that gap by extending unary computing to convolution dataflows,

TABLE I
WORD SPARSITY INDICATING THE PERCENTAGE OF ZERO WEIGHTS IN EIGHT QUANTIZED INT8 CNNS [11].

CNN	**Word (%) 8 bits**
MobileNetV2	2.25
MobileNetV3	9.52
GoogleNet	1.91
InceptionV3	1.99
ShuffleNetV3	1.43
ResNet18	2.043
ResNet50	2.45
ResNeXt101	2.64

while ensuring compatibility with widely-used deep learning accelerators (DLAs) like NVDLA [12]. Table I illustrates the inherent weight sparsity for INT8 quantized CNNs that can be exploited by unary computing.

Despite recent progress, there remains a substantial gap in research. Only a few custom unary-based accelerators have been proposed, such as in [13] and [14]. These accelerators rely on stochastic computation, which trades off computational accuracy for hardware efficiency. However, as lower precisions become the standard, deterministic computation becomes essential to minimize accuracy degradation. Furthermore, these stochastic designs are not, nor easily made, compatible with existing off-the-shelf DLAs, which are widely used in today's installed hardware base.

This work presents *Tempus Core*, a highly hardware-efficient temporal-unary-binary convolution engine designed to integrate seamlessly into existing DLAs without sacrificing compute accuracy. *Tempus Core* employs *tub* multipliers [11] within its processing element (PE) array to execute convolution operations. Seamless integration into current DLAs is demonstrated through its incorporation into the NVDLA architecture [12]. While maintaining the computational accuracy of binary-based arithmetic designs, *Tempus Core* delivers improved hardware efficiency compared to the NVDLA convolution core and remains fully compatible with NVDLA's dataflow.

The main contributions of this work are as follows.

1) *Novel High Iso-Area Throughput Convolution Engine*: Tempus Core employs *tub* multipliers for unary-based convolution, designed as a drop-in replacement for modern DLA convolution datapaths. Unlike previous temporal GEMM designs [9][10] that follow outer-product GEMM dataflow, Tempus Core serves as a convolution engine supporting inner-product convolution dataflow, improving efficiency without sacrificing compatibility. For Tempus Core comprising of 16×16 PE array, 5x and 4x increase in iso-area throughput is realized for INT8 and INT4 precisions, compared to standard binary-based NVDLA.
2) *Integration with Industry-Grade DLAs*: Unlike previous unary DLA datapaths, Tempus Core is designed for seamless integration into existing binary-based DLAs, enhancing key hardware performance and efficiency metrics while maintaining functional integrity. This integration enables programmers to retain the full functionality of the DLA's existing software stack.
3) *Comprehensive Comparative Evaluation*: We perform a detailed comparison between the convolution datapath of NVDLA's convolution core (CC) and Tempus Core across multiple levels of hierarchy, ranging from a single "PE Cell" to PE array (multiple cells), and up to entire PCU (PE Cell Unit). Rigorous evaluation is performed across low integer precisions and array sizes using 45 nm CMOS technology to report post-synthesis metrics.
4) *Place-and-Route Analysis*: Unlike previous studies, we conduct a post place-and-route analysis to obtain more precise measurements of area and power between a Tempus Core's PE cell unit (PCU) and NVDLA's CMAC unit for 16×4 array. Results show improvements of 53% in area efficiency and 44% in power efficiency, respectively, in the case of Tempus Core's PCU.
5) *Fine-Grained CNN Profiling*: Additionally, weight distribution analysis for PE array tile size of 16×16 is conducted across convolution layers for INT8-quantized MobileNetV2 and ResNeXt101 models. This analysis provides insights into application-specific, sparsity-driven latency and thereby energy for Tempus Core.

This paper is structured as follows. Section II provides the background and overview of relevant topics. Section III describes the Tempus Core design and its integration into the NVDLA design. Section IV discusses the evaluation methodology used in this study, with Section V reporting the results of the experiments along with their analysis. Finally, Section VI summarizes conclusions and future work.

II. BACKGROUND AND RELATED WORKS

A. General Matrix Multiplication

General matrix multiplication (GEMM) is a key DL operation involving activation and weight matrix multiplication, defined in the BLAS library [15]. The compute-intensive nature of fully connected and convolutional layers in CNNs allows them to be efficiently mapped to matrix multiply units. A study indicates that up to 89% of CPU runtime and 95% of GPU runtime are spent processing fully connected and convolutional layers in AlexNet [16]. Similarly, matrix multiplications constitute at least 75% of the processing time across various language model sizes and input token lengths [17], underscoring the critical role of GEMM in DL computations.

B. tubGEMM

*tub*GEMM [10][11] is a novel temporal-binary hybrid INT-based GEMM architecture that inherently leverages the sparsity found in DL workloads, resulting in significant improvements in latency and energy efficiency. This architecture processes activation data as binary inputs while encoding weight bits into a single temporal bitstream. The processing element (PE) of *tub*GEMM consists of multiplexers, shifters, and registers, contributing to a streamlined microarchitectural design. It employs a unique $2s$-unary encoding scheme, where each unary bit or cycle is interpreted as a data value of 2, effectively halving the latency. This advancement significantly improves upon the performance of *tu*GEMM [9], which has a worst-case latency of $N * (2^{w-1})^2$ cycles. In contrast, *tub*GEMM reduces the worst-case latency to $N * (2^{w-2})$ cycles, where N represents the common dimension of input matrices, and w denotes the bitwidth. Fig. 2 illustrates the dataflow of an INT4 *tub* multiplier. A comparative design analysis of existing unary-based GEMM architectures [11] concludes *tub*GEMM to be the optimal unary design.

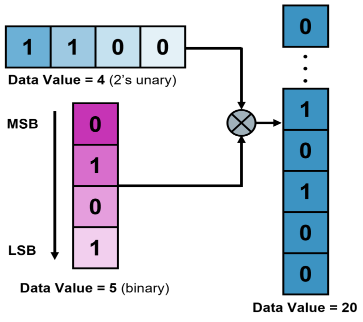

Fig. 2. An example dataflow of an INT4 *tub* multiplier. The INT4 *tub* multiplier take a 4-bit binary-encoded value and a single temporal-coded bitstream as inputs. For each "1" bit in the bit-serial temporal-coded input, the binary value is accumulated, producing the desired output result.

C. NVDLA Accelerator

NVDLA [12] is an open-source, industry-grade inference engine developed by NVIDIA, integrated into Xavier systems on chip (SoCs) for self-driving applications. It offers an end-to-end software-hardware stack that provides a DL framework along with a runtime environment and spans all the way to RTL implementation. This study employs the *nv_small* configuration, which is tailored for embedded and cost-sensitive applications [12]. Unlike the *nv_large* variant, the *nv_small* design does not include a dedicated control processor; instead, it relies on the host processor for task scheduling, memory management, and NVDLA control. Although other open-source DLAs, such as Gemmini [18], exist, they are primarily academic designs focused on GEMM acceleration. Moreover, NVDLA was reported to be 3.8x faster than Gemmini when comparing equivalently sized configurations running ResNet-50 [19]. Given its performance and proven industry reliability, we select NVDLA as our experimental platform.

Fig. 3 provides a high-level overview of Tempus Core integration into NVDLA's convolution pipeline. While based on NVDLA's structure, the figure specifically highlights the modifications introduced. The original NVDLA architecture comprises (i) the convolution buffer (CB), which stores input activations and filter weights, (ii) the convolution core (CC), which serves as the convolution datapath, and (iii) the post-processing unit, which includes the activation engine, the pooling engine, and other components essential for accelerating CNNs. The primary datapath performing MAC operations to

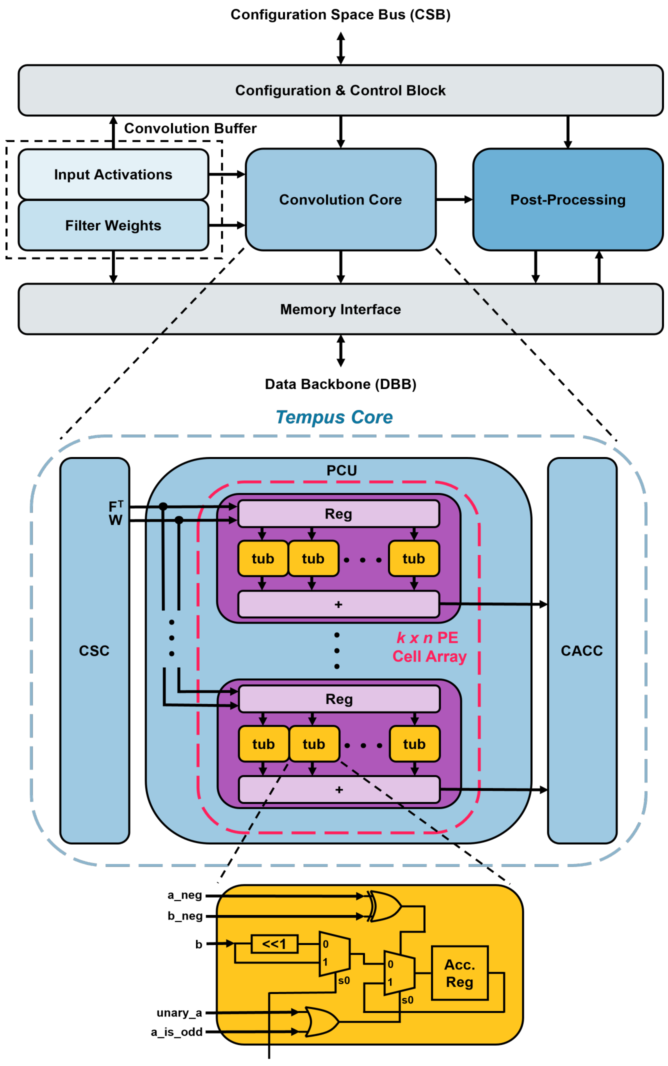

Fig. 3. Overview of Tempus Core integration into NVDLA. In NVDLA, the convolution buffer (CB) stores both activation and weight values, which are fed into the Convolution Core (CC) consisting of the convolution sequence controller (CSC), the Convolution MAC (CMAC) unit (containing the $k \times n$ MAC array, output registers, and handshaking logic) and the convolution accumulator (CACC). In this work, CC is replaced by Tempus Core, which contains modified CSC and a PE cell unit (PCU) containing k *tub*-based PE cells replacing CMAC. Each PE cell (purple box) consists of n *tub*-based multipliers. Additional handshaking logic to facilitate multi-cycle convolution operation is present, along with output registers to maintain functionality.

calculate partial sums in CC is the convolution MAC (CMAC) unit. The CMAC unit consists of k processing element (PE) cells (termed MAC Cells in [12]), each with n multipliers, producing a partial result for each kernel. Data is broadcast from the CB via a convolution scheduler (CSC) to the k PE cells, with partial results accumulated in the adder trees of the convolution accumulation (CACC) unit. Each cell supports clock gating to reduce dynamic power consumption during idle or underutilized conditions when the kernel count is insufficient for full utilization. In addition, CMAC features intermediate registers that facilitate retiming and pipelining.

To summarize the NVDLA microarchitecture hierarchy:

1) The convolution engine in NVDLA is called convolution core (CC), the focus of this work.
2) The CC comprises CSC, CMAC, and CACC. The CMAC unit contains $k \times n$ PE array, denoting k number of PE (MAC) cells, each consisting of n multipliers and registers for caching partial sums.
3) In addition to the n multipliers, each PE cell contains local registers and an adder tree to accumulate the intermediate results, producing one partial sum per PE cell.

III. Tempus Core Microarchitecture

The *Tempus Core* microarchitecture (Fig. 3) is implemented as a temporal-unary-binary (*tub*) hybrid convolution engine, designed as a drop-in replacement for the convolution core (CC) in NVDLA. Tempus Core adheres to the original dataflow in NVDLA and can directly replace its convolution core. Tempus Core consists of a modified CSC, PE cell unit (PCU), and the CACC. The PCU is analogous to the CMAC unit in NVDLA, containing $k \times n$ *tub*-based PE array for efficient MAC operation. In this setup, direct convolution involves sharing the feature data array across k PE cells, with each cell caching a different weight data array. Each PE cell contributes a partial sum, producing k partial sums per feature and weight data array. These partial sums are subsequently accumulated in the CACC unit. While NVDLA's CMAC unit generates k partial sums per cycle, the Tempus Core's PCU generates k partial sums over multiple cycles, with the cycle count equal to the largest weight magnitude in the $k \times n$ array. The modified CSC in Tempus Core feeds transposed feature data since $W \times F^T = \text{accum}(W \odot F)$. Note each PE cell also incorporates n *2s*-unary blocks into the temporal encoder, allowing optimal temporal-to-binary data conversion. The registers corresponding to the PE cells cache the partial sums, which are only forwarded to the CACC once all partial sums have been generated across the cells.

Having detailed the key components of Tempus Core, we now outline its dataflow structure and its seamless integration into existing DLAs.

- The PCU comprises a $k \times n$ PE array, with dedicated registers for caching outputs and handshaking logic to coordinate multi-cycle operations.
- The input data and initial weight data cubes are divided into $1{\times}1{\times}n$ element cubes. Each PE cell caches one weight cube from the weight kernel, and the single input data cube is shared between the k PE cells.
- Each *tub* multiplier performs a multiplication between a single weight and input data element over multiple cycles, and the adder tree in the cell accumulates the results into a partial sum to be fed next to the CACC unit.

Further microarchitectural enhancements in *Tempus Core* include the integration of additional handshaking protocols with buffer blocks to accommodate multiple *tub* cycles per partial sum computation within each PE cell. This modification ensures efficient synchronization and data integrity. Utilizing *tub* multipliers further ensures deterministic compute, with output accuracy equivalent to standard binary-based convolution approaches. The number of compute cycles for $k \times n$ array in *Tempus Core* is determined by the largest weight magnitude present in the array. We profile quantized CNNs to report application-specific latency per array in Section V.

TABLE II
Post-synthesis cell area (top) and total power (bottom) results in 45nm CMOS: for a Single PE Cell ($k = 1$) with n PEs.

Configuration		Binary PE Cell	*tub* PE Cell	Improvement
Precision	Number of PEs (n)	Area (μm^2)	Area (μm^2)	(%)
INT4	16	0.0022	0.0006	71.89
	256	0.0371	0.0046	87.53
	1024	0.1462	0.0171	88.30
INT8	16	0.0056	0.0011	80.15
	256	0.1063	0.0093	91.24
	1024	0.4334	0.0355	91.81

Configuration		Binary PE Cell	*tub* PE Cell	Improvement
Precision	Number of PEs (n)	Power (mW)	Power (mW)	(%)
INT4	16	0.09	0.06	25.9
	256	1.03	0.19	81.7
	1024	3.98	0.51	87.3
INT8	16	0.20	0.088	54.7
	256	3.00	0.32	89.4
	1024	12.20	1.06	91.3

IV. Evaluation Methodology

This section details the evaluation setup, profiling methodology, and key EDA tools used to measure latency and energy values across designs. Comparative analysis between NVDLA's CC and Tempus Core is performed at three levels of granularity: (i) single PE cell, (ii) $k \times n$ PE array, and (iii) entire CMAC unit versus PCU in Tempus Core. The analysis spans both INT8 and INT4 precisions, with varying $k \times n$ configurations. The designs are synthesized using NanGate45 [20] open-source cell library in Synopsys Design Compiler, operating at a fixed 250 MHz clock frequency to maintain consistent timing across evaluations. The binary PE cells are elaborated with DesignWare-optimized multipliers and Wallace adder tree during synthesis. Furthermore, place-and-route results are obtained using Cadence Innovus with the NanGate45 library to further extend the evaluation.

Weight-value profiling of INT8-quantized MobileNetV2 and ResNeXt101 models from the Torch Vision library is performed to determine application-specific latency and energy consumption. Using a 16×16 (k=16, n=16) max pool across weights present in the model's convolution layers, the largest weight value within a tile is determined, and its frequency of occurrence as the largest value across is derived. This directly correlates to the compute cycles since the largest value across an array of 16 PE cells with 16 multipliers bottlenecks the *tub* model compute. Note that the PCU takes a few extra cycles to cache the values in and out, and hence, the analysis pertains to the array. Additionally, sparsity is analyzed similarly to estimate the average number of "silent" PEs per array, where *tub* multipliers remain inactive for zero-valued weights.

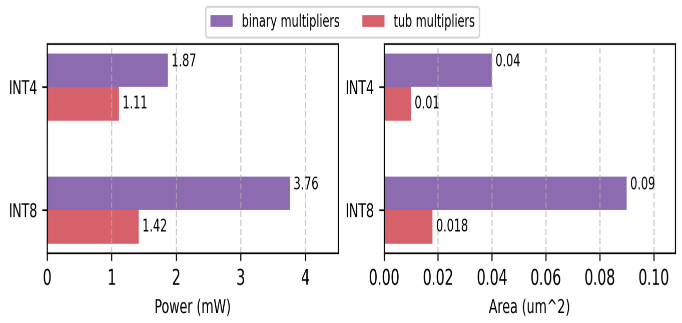

Fig. 4. Post-synthesis total power consumption (left) and cell area utilization (right) in 45nm CMOS for the two different 16 × 16 array designs, for both INT4 and INT8 precisions.

V. Results

A. Area-Power Efficiency

Post-synthesis results for a single PE cell, containing registers, multipliers, and an adder tree, are summarized in Table II. Single *tub*-based PE cells significantly outperform their binary counterparts in both area and power efficiency. For INT8, tub multipliers reduce area by 91.8% and power by 91.3% for n=1024 PEs. INT4 implementations achieve 88.3% area and 87.2% power reductions. PE cell with *tub* multipliers demonstrate superior scalability across n: area scales by 7.7x for INT4 and 8.5x for INT8, compared to 16.9x and 19x for binary PE cell. Power consumption scales by 2.9x for INT4 and 3.5x for INT8, versus 11.4x and 15x for binary PE cell.

Tempus Core's scalability is maintained as we scale from a single PE cell to a 16 × 16 array. At INT8 precision, the binary-based implementation requires 0.09 μm^2 of area and 3.8 mW of power, respectively. In comparison, Tempus Core's *tub*-based PE cells consume only 0.02 μm^2 and 1.42 mW, achieving a 75% area reduction and 62% power savings. Similarly, for INT4, the reductions are 80% in area and 41% in power. These results are illustrated in Fig. 4.

Finally, moving further up in the hierarchy, post-synthesis results of the entire CMAC unit and the PCU of Tempus Core are compared and illustrated in Fig. 5, for various widths (n) in INT8, INT4, and INT2 precisions. The PCU improves area and power consumption by 59.3% and 15.3%, respectively.

Key Takeaway: PCU (and in turn, Tempus Core) demonstrates significantly superior area-power efficiency as compared to the baseline binary-based CMAC unit across diverse integer-based precisions and granularities of compute datapath, paving the path for more silicon-optimized and scalable DLAs.

B. Place-and-Route Results

Place-and-route analysis for INT4-based CMAC and PCU units with 16 × 4 array is conducted using 45nm CMOS, maintaining 70% floorplan utilization for a fair comparison. As shown in Table III, PCU reduces total area by 53% and power by 44% compared to binary multipliers. This confirms the potential of temporal-unary designs for low-precision edge-based DLAs. The area utilization for both designs is illustrated in Fig. 6, which depicts the decrease in area utilization of the PCU design compared to the CMAC unit.

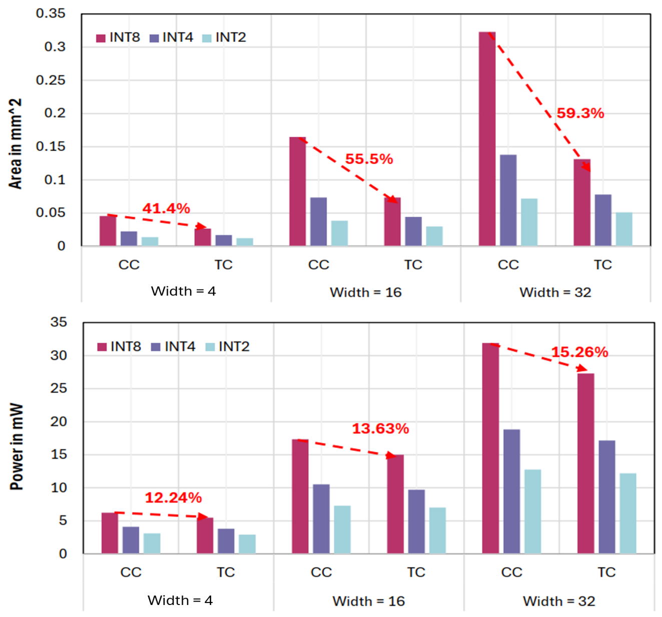

Fig. 5. Post-synthesis area utilization and total power consumption in 45nm CMOS across entire CMAC and PCU units for different array widths ($16 \times n$) with n = 4, 16, and 32. CC refers to the CMAC unit in CC, and TC denotes the PCU inside Tempus Core. For a constant core configuration of 16 PE Cells (array height), number of multipliers are varied across INT8, INT4, and INT2 precisions. Percentage decrease in area and power consumption for INT8 are denoted by the red dotted arrows.

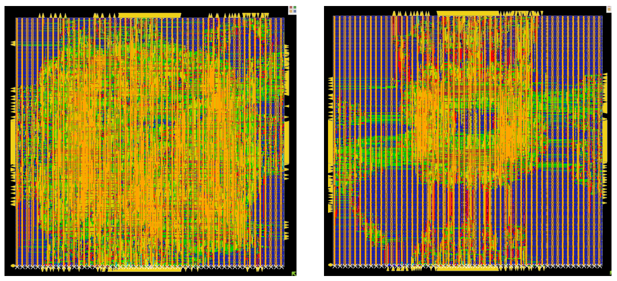

Fig. 6. Layout plots for CMAC (left) and PCU (right) for an INT4-based 16 × 4 array in 45nm CMOS. Note the significant reduction in logic complexity for the latter for the same floorplan size, indicating less area utilization.

TABLE III
Post-place-and-route results for CMAC and PCU units with a 16x4 array for 45nm CMOS.

Design	NanGate45 Total Area (mm^2)	NanGate45 Total Power (mW)
CMAC Core	0.04	10.7
Tempus Core	0.02	6.12

C. Energy Evaluation with Workload-Dependent Latency

Tempus Core's $2s$-unary encoding [10] allows it to exploit dynamic value sparsity, reducing the compute latency. We profiled two DNN workloads, MobileNetV2 and ResNeXt101, using max-pooling over 16 × 16 tiles of convolution layer

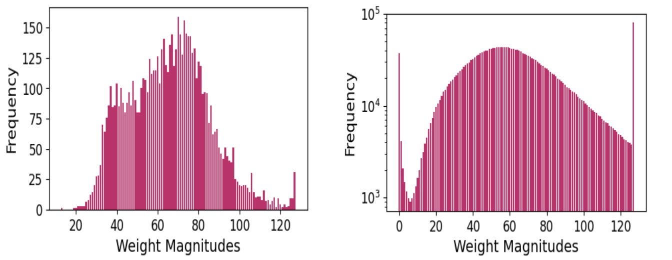

Fig. 7. Weight magnitude profiling across the convolution layer weights of MobileNetV2 (left) and ResNeXt101 (right) with max pool of 16 × 16.

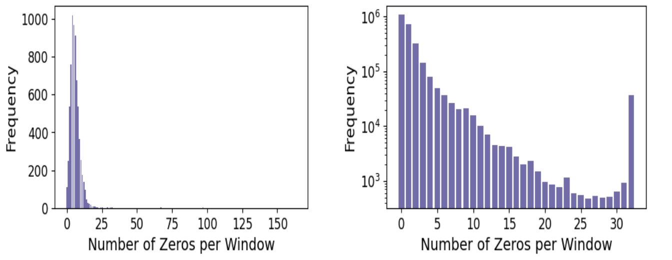

Fig. 8. Sparsity profiling across the convolution layer weights of MobileNetV2 (left) and ResNeXt101 (right) with tile size of 16 × 16.

weights. Fig. 7 shows the frequency of weight values (0-128) which correlate with compute latency in a 16 × 16 array. The area under the curve normalized by the total sum of frequencies provides the average workload-dependent latency. Using this methodology, MobileNetV2 incurs 33 cycles, and ResNeXt101 incurs 31 cycles, on average, using $2s$-unary encoding. Using these workload-dependent cycle counts and the 4ns clock period (250 MHz) along with INT8 power consumption values for 16 × 16 array (Fig. 4), application-specific energy is derived. For the binary-based array, the energy consumption is 15 pJ. For *tub*-based array, it amounts to 187 pJ and 176 pJ for MobilNetV2 and ResNeXt101, respectively, due to the higher cycle counts. Although the higher cycle count results in lower energy efficiency for Tempus Core for INT8 precision, there is potential to reduce this gap by leveraging zero-value weights to disable the corresponding PE compute. From zero-value weight profiling per 16×16 tile size in Fig. 8, the average number of silent PEs for MobileNetV2 is 6 (250 active PEs) and 2 for ResNeXt101. Hence, the above energy calculation is an overestimate, as it assumes that all 256 PEs in the tile are active for 31 and 33 cycles.

The energy overhead is reduced for lower precision. With INT4, the worst-case latency is 4 cycles for tub multipliers with twos-unary encoding. This results in 7.48 pJ for the binary PE array, and 17.76 pJ for *tub*-based PE array, indicating decrease in energy gap from 11.7x (INT8) to 2.3x (INT4).

D. Iso-Area Throughput Improvements

While the Tempus Core design incurs more cycles for convolution due to the incorporation of the *tub* PEs, throughput improvements can transcend the latency increase. In NVDLA's binary PE array, k partial sums are produced per cycle, whereas the *tub*-based array in Tempus Core requires $m_{k \times n}$ cycles, where $m_{k \times n}$ represents half of the largest weight magnitude in the array ($2s$-unary encoding). INT8 analysis shows 16 binary PE cells consume 0.09 µm² (from Fig. 4), whereas 80 tub PE cells fit in the same area (0.018 µm² for 16 PE cells), yielding a 5x throughput improvement. Similarly, INT4 shows 4x iso-area throughput boost for the same array size. For a single PE cell, scaling across different n multipliers (reported in Table II), the iso-area throughput improvements are illustrated in Fig. 9 for both INT8 and INT4 precisions (assuming the same m cycles to generate one partial product). From the area scaling estimates, we can further project iso-area throughput improvements for a quadratic increase from $n = 256$ to $n = 65536$ multipliers. The throughput increases by as much as 26x and 18x for INT8 and INT4 precisions, respectively. Based on these observations, iso-area throughput gains can scale even further with increasing array sizes, compensating for the increased compute cycles, improving the practical feasibility of Tempus Core.

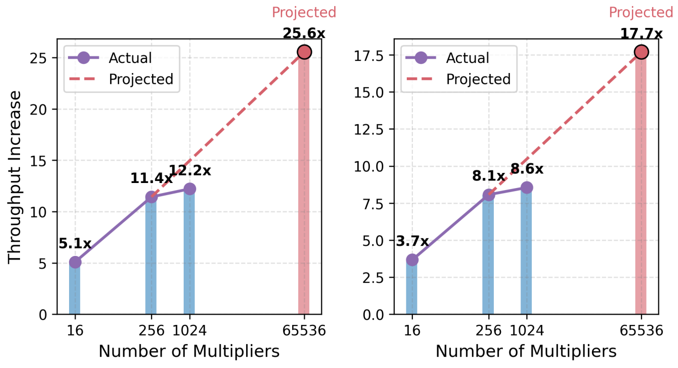

Fig. 9. Iso-area throughput improvements for a single PE cell ($k = 1$) across varying number of multipliers for INT8 (left) and INT4 (right) precisions, assuming same m cycles of latency incurred. Red dotted trend lines project iso-area throughput improvements based on area scaling from Table II.

VI. Conclusions and Future Work

In this paper, we introduced Tempus Core, a temporal-unary-binary hybrid convolution engine optimized for seamless integration into modern edge DLAs. By extending unary computing beyond GEMM-level optimizations to support full convolution operations, Tempus Core demonstrates substantial improvement in area and power efficiency. By demonstrating 5x and 4x iso-area throughput improvements for INT8 and INT4 precisions, respectively, for a 16 × 16 PE array, we have established a pathway to significantly enhance the performance-per-unit cost ratio of edge AI accelerators. In the future, we plan to extend the work towards unary-based compute architectures targeting ultra-low precision quantized large language models (LLMs). Additionally, we aim to explore custom dataflows and compiler optimizations that further reduce latency and energy consumption, enhancing the practicality of unary computing in edge-AI hardware.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[2] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. Aga, J. Huang, C. Bai *et al.*, "Sustainable ai: Environmental implications, challenges and opportunities," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 795–813, 2022.

[3] 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," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1796–1807, 2020.

[4] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu *et al.*, "Fp8 formats for deep learning," *arXiv preprint arXiv:2209.05433*, 2022.

[5] J. Choquette, "Nvidia hopper gpu: Scaling performance," in *2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society, 2022, pp. 1–46.

[6] F. Zhu, R. Gong, F. Yu, X. Liu, Y. Wang, Z. Li, X. Yang, and J. Yan, "Towards unified int8 training for convolutional neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1969–1979.

[7] T. Han, T. Zhang, D. Li, G. Liu, L. Tian, D. Xie, and Y. S. Shan, "Convolutional neural network with int4 optimization on xilinx devices," *Xilinx White Paper, WP521*, 2020.

[8] S. Jain, A. Gural, M. Wu, and C. Dick, "Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 112–128, 2020.

[9] H. Nair, P. Vellaisamy, A. Chen, J. Finn, A. Li, M. Trivedi, and J. P. Shen, "tugemm: Area-power-efficient temporal unary gemm architecture for low-precision edge ai," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–5.

[10] P. Vellaisamy, H. Nair, J. Finn, M. Trivedi, A. Chen, A. Li, T.-H. Lin, P. Wang, S. Blanton, and J. P. Shen, "tubgemm: Energy-efficient and sparsity-effective temporal-unary-binary based matrix multiply unit," in *2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2023, pp. 1–6.

[11] P. Vellaisamy, H. Nair, D. Wu, S. Blanton, and J. P. Shen, "Exploration of unary arithmetic-based matrix multiply units for low precision dl accelerators," in *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2024.

[12] "Nvdla primer," http://nvdla.org/primer.html.

[13] S. Li, A. O. Glova, X. Hu, P. Gu, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Scope: A stochastic computing engine for dram-based in-situ accelerator," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 696–709.

[14] I. G. Thakkar, S. M. Shivanandamurthy, and S. A. Salehi, "Low-latency, energy-efficient in-dram cnn acceleration with bit-parallel unary computing," in *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing: Hardware Architectures*. Springer, 2023, pp. 393–409.

[15] L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry *et al.*, "An updated set of basic linear algebra subprograms (blas)," *ACM Transactions on Mathematical Software*, vol. 28, no. 2, pp. 135–151, 2002.

[16] Y. Jia, *Learning semantic image representations at a large scale*. University of California, Berkeley, 2014.

[17] G. Park, B. Park, M. Kim, S. Lee, J. Kim, B. Kwon, S. J. Kwon, B. Kim, Y. Lee, and D. Lee, "Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models," *arXiv preprint arXiv:2206.09557*, 2022.

[18] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 769–774.

[19] A. Gonzalez and C. Hong, "A chipyard comparison of nvdla and gemmini," *Berkeley, CA, USA, Tech. Rep. EE*, pp. 290–2, 2020.

[20] "Nangate freepdk45 open cell library," 2008.