Exploration of Unary Arithmetic-Based Matrix Multiply Units for Low Precision DL Accelerators

Prabhu Vellaisamy*, Harideep Nair*, Di Wu†, Shawn Blanton*, John Paul Shen*
*Electrical and Computer Engineering Department, Carnegie Mellon University
{pvellais, hpnair, rblanton, jpshen}@andrew.cmu.edu
†Department of Electrical and Computer Engineering, University of Central Florida
di.wu@ucf.edu

***Abstract*—General matrix multiplication (GEMM) is a fundamental operation in deep learning (DL) compute. Deep learning accelerators (DLAs) typically comprise arrays of multiply-accumulate (MAC) units to facilitate efficient matrix multiplication. With the current trend of DL moving increasingly towards low precision, recent research touts unary computing as a promising alternative to the highly parallelized binary-based computing paradigm, with novel unary-based GEMM designs being proposed that trade-off latency for low hardware complexity. This paper performs a detailed exploration of three state-of-the-art unary GEMM designs for integer (INT)-based DL inference, namely, uGEMM, *tu*GEMM, and *tub*GEMM, comparing them against a conventional binary-based matrix multiplication unit. Rigorous post-synthesis evaluations in 45 nm CMOS are performed across varying bitwidths (2 to 8 bits) to assess the designs' tradeoffs. Further, the paper performs weight sparsity analysis across a spectrum of pre-trained convolutional neural networks (CNNs) and the LlamaV2 large language model (LLM), paving the way for recommendations on how unary computing can effectively mitigate its latency and energy efficiency gap relative to conventional binary matrix multiplication units.**

***Index Terms*—unary-computing, matrix multiplication, deep learning accelerators**

I. Introduction

The remarkable advancements in artificial intelligence (AI), primarily driven by deep learning (DL), have made it possible to outperform humans in various complex tasks [1]. However, computation cost for running DL models is ever-increasing at an exponential rate [2]. The need for efficient hardware substrates has spurred interest in dedicated deep learning accelerators (DLAs) that implement optimized general matrix multiplication (GEMM), the fundamental computing operation in DL. Some recent examples are tensor cores in GPUs and matrix multiplication units (MXU) in Google's TPUs, edge DLAs like Edge TPU, and NVIDIA's Jetson Xavier NX. These hardware units employ conventional binary arithmetic and achieve hardware efficiency through low-precision computing and optimized dataflow.

Unary computing, touted as a promising alternative to relatively complex binary arithmetic, manifests in two forms: (i) *rate-coding*, and (ii) *temporal-coding*. It particularly excels in low-precision hardware arithmetic. Three recently proposed state-of-the-art (SOTA) unary-based GEMM designs are: (i) uGEMM, a unified rate-and-temporal encoded GEMM architecture executing stochastic GEMM operations [3], (ii) *tu*GEMM, the first fully-temporal GEMM design ensuring deterministic compute with full accuracy [4], and (iii) *tub*GEMM, a temporal-binary hybrid successor to *tu*GEMM that significantly reduces latency with relatively modest hardware overhead, improving overall energy efficiency [5]. Despite comparisons against previous unary GEMM approaches, a comprehensive evaluation against conventional binary GEMM designs used in today's DLAs is yet unexplored.

Unary computing gains prominence in AI inference as the industry gravitates toward lower precision computing, propelled by advancements in quantization techniques. Although precision scaling poses challenges during training, strides have been made in retaining accuracy from FP32 (32-bit floating point) to as low as FP8 (8-bit floating point) for training and as low as INT4 and INT2 (4-bit and 2-bit integer formats, respectively) for inference [6, 7]. In the space of large language models (LLMs), NVIDIA's Grace Hopper, featuring FP8 precision in its Transformer Engine, accelerates transformer models [8]. Latest works like BitNet b1.58 [9], utilizing ternary weights, pave the way for 1-bit LLMs.

This study aims to assess unary-based matrix multiplication cores for low-precision edge DLAs, focusing on three SOTA unary-based GEMM units. Comparison with a conventional binary-based GEMM unit is undertaken to delineate advantages/disadvantages and offer recommendations for bridging the computational efficiency gap between the two computing domains in AI through sparsity analysis of DL models.

Key contributions of this work are:

- SOTA unary-based GEMM architectures are analyzed and juxtaposed against binary-based GEMM architecture for integer (INT) DL inference within DLAs.
- Post-synthesis 45nm PPA results are obtained across varying bit-width (2-, 4-, and 8-bits) and Mobile SoC DLA array configurations (16x16, 32x32) for fair comparisons across all architectures.
- Weight sparsity analysis of pre-trained SOTA convolutional neural networks (CNN) and LLaMA2 LLM is performed to assess how unary-based designs can inherently exploit sparsity to mitigate the latency overhead and decrease energy consumption.
- Design tradeoffs are assessed amongst the unary and binary designs, and potential future directions for unary GEMM designs targeting edge DLAs are discussed.

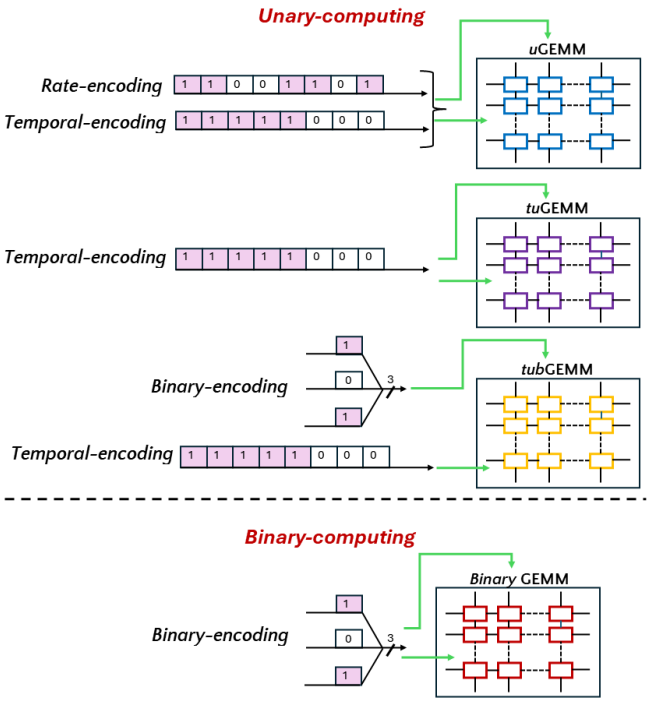

Fig. 1. Overview of the different GEMM designs and their respective input encoding formats: uGEMM encodes both inputs as unary (rate/temporal), *tu*GEMM temporal-unary, and binary GEMM conventional binary whereas *tub*GEMM encodes one input as temporal-unary and the other as binary. Dataflow to processing element (PE) array is indicated by the green arrows.

II. Background and Motivation

A. General Matrix Multiplication (GEMM)

GEMM serves as the core compute kernel in deep learning that executes matrix multiplication of model weights and activation values. It is an integral part of Basic Linear Algebra Subprograms (BLAS), represented as $Y = \alpha AB + \beta C$, where A, B, and C are dense matrices and α and β are scalar coefficients. This study specifically focuses on the non-scaled (without α and β) implementation of GEMM designs.

In DL, the fully-connected and convolution layers rely heavily on GEMM operations, with up to 89% of CPU runtime and 95% of GPU runtime [10] spent on an AlexNet forward pass, underscoring the importance of GEMM optimization for computational efficiency.

B. Unary Computing

Unary computing uses serial bitstreams for value representation, differing from the multiple parallel bits in binary computing. It takes two forms - *Rate*-unary and *Temporal*-unary, presenting values in *unipolar* (unsigned) or *bipolar* (signed) formats. In rate-based unary computing, data is encoded based on the frequency of 1s, directly proportional to the represented value. Temporal-unary encoding represents data as a consecutive sequence of 1s followed by 0s, with the number of 1s indicating the data value.

Traditionally, rate-unary encoding allows for low-complexity arithmetic hardware suitable for stochastic computing. It significantly simplifies circuits [11], providing a reasonable approximation of the ideal result with an inherent accuracy compromise. In contrast, temporal-unary encoding enables exact deterministic compute. Temporal arithmetic units have largely been unexplored [3].

C. State-of-the-art unary GEMM Architectures

This study categorizes three SOTA unary-based General Matrix Multiplication (GEMM) architectures for AI inference, as illustrated in Figure 1. The paper's primary focus is to evaluate the tradeoffs among these designs and compare them with a conventional binary-encoded matrix multiplication unit, aiming to provide insights into their strengths, weaknesses, and performance characteristics. This nuanced analysis contributes to informed decision-making in the evolving landscape of unary and binary computing paradigms for GEMM operations. Key architecture differences are summarized in Table I.

TABLE I
CLASSIFICATION OF GEMM ARCHITECTURES FOR EVALUATION, WITH THEIR RESPECTIVE ENCODING SUPPORT, TYPE OF COMPUTE, AND ARITHMETIC UNITS FOR MATRIX MULTIPLICATION

GEMM	Encoding	Compute	Arithmetic Units
Binary GEMM	Binary-coding	Deterministic	Multipliers, Adders and Accumulators
uGEMM	Rate-coding & Temporal-coding	Stochastic	uMUL, uSADD, uNSADD
*tu*GEMM	Temporal-coding	Deterministic	Counters
*tub*GEMM	Temporal-coding & Binary-coding	Deterministic	Sequential Multipliers and Accumulators

1) uGEMM: uGEMM [3] adopts a *unified*-unary GEMM architecture, supporting both rate-coding and temporal-coding. It introduces *unified*-unary computing units and incorporates *early termination*, reporting a 98% increase in energy efficiency and a minimal loss of less than 0.5% from the ideal output. A non-streaming variant is evaluated for a fair comparison with other designs, using binary inputs converted to unary-coded inputs. uGEMM requires 2^w cycles for a GEMM computation, with w as the bitwidth. Parallel adder trees are used to accumulate all partial sum bitstreams.

*2) tuGEMM: tu*GEMM [4] is the pioneering GEMM architecture supporting deterministic computing for temporal-encoding. Two design variants, *serial* and *parallel*, offer different area-latency tradeoffs. Comparison with uGEMM for 45nm CMOS shows lower area-and-power consumption than uGEMM [4]. The worst-case latency for *tu*GEMM scales exponentially with bitwidth due to operation on temporal-coded bitstreams, leading to a worst-case latency of $N * (2^{w-1})^2$ cycles for a GEMM computation, with N as the common dimension of the input matrices.

*3) tubGEMM: tub*GEMM utilizes hybrid temporal-unary and binary (tub) encoding, performing exact (deterministic) computation. In comparison to the leading unary design uGEMM, *tub*GEMM achieves substantial reductions in area

(89%), power (87%), and energy (50%). The authors conduct PPA evaluations on the TSMC5 (N5) process node. Compared to *tu*GEMM, latency is reduced exponentially to $N * (2^{w-2})$ cycles for a GEMM computation, optimizing the encoding with a 2-*unary* scheme [5].

III. Evaluation Methodology

This section details the evaluation methodology for post-synthesis Power-Performance-Area (PPA) metrics, ensuring a fair comparison among all GEMM designs. Additionally, it presents weight-sparsity analysis for 8 SOTA CNN models and LlaMA2-70B, where word-sparsity signifies zero values (percentage of weights that are zero), and bit-sparsity denotes small magnitude values (percentage of '0' bits in the unary bitstream). Note that higher word sparsity and higher bit sparsity (i.e., lower percentage of '1' bits in the unary bitstream which is equivalent to small magnitude value) are desirable.

A. Sparsity Profiling of DL models

Pretrained weights of 8 CNN models and a quantized INT32 LLaMA2-70B LLM are profiled to perform sparsity analysis. Due to constraints on accessing low-precision quantized LLaMA2 model weights, a pragmatic approach is adopted by considering the eight most significant bits (MSBs) for sparsity profiling, used in prior works [7, 12] without impacting the distribution and sparsity significantly [13]. The weights are categorized into two groups: two fully connected (FC) layers in the attention layer and a 2-layer feed-forward network (FFN) after the attention layer. As the attention layer involves self-attention, we also profile the query (Q) and key (K) tokens directly multiplied at the input. Self-attention Q and K profiling is input-dependent, with a small token sample used as representative input. The average maximum value per 32x32 block is considered for bit sparsity, as the largest value serves as the compute bottleneck for GEMM compute.

We utilize eight ImageNet pre-trained and quantized DNN models with INT8 precision from Torchvision. These models, part of PyTorch's Torchvision library, are widely used in leading DL literature and MLPerf benchmarking. Selected architectures are: 1) *MobileNetV2*, 2) *MobileNetV3*, 3) *InceptionV3*, 4) *ShuffleNetV2*, 5) *GoogleNet*, 6) *ResNet18*, 7) *ResNet50*, and 8) *ResNeXt101*. We analyze bit sparsity by extracting the average number of one bits (equivalent to value magnitude in temporal-unary encoding) in every 8-bit weight value across all fully-connected and convolution layers.

B. Setup for PPA Analysis

The designs are synthesized using the Nangate45 library with Synopsys Design Compiler, with an operating clock frequency of 400 MHz. The synthesis process is conducted for 2-, 4-, and 8-bits across 16x16 and 32x32 array sizes (Mobile SoC DLA array sizes). The GEMM designs are non-scaled, bipolar variants [3]. We refer to the conventional binary-based GEMM unit as 'bGEMM'. Additionally, we consider the *serial* variant of the *tu*GEMM design from [4]. The synthesized designs contain only the arithmetic units, without any buffers or control logic to facilitate tiling.

TABLE II
45nm post-synthesis area (in μm^2) for the four GEMM designs with varying bit-widths and matrix dimensions

Configuration		uGEMM	*tu*GEMM	*tub*GEMM	bGEMM
2-bit	16x16	99,445.7	13,436.4	19,112.6	16,739.1
	32x32	791,794.4	52,272.4	76,375.5	67,201.7
4-bit	16x16	203,920.7	29,061.0	38,912.6	44,925.8
	32x32	1,799,961.0	117,261.3	151,933.6	180,458.6
8-bit	16x16	445,396.2	61,064.0	99,916.8	132,786.9
	32x32	3,689,829.0	235,470.9	338,692.7	560,778.5

TABLE III
45nm post-synthesis power (in mW) for the four GEMM designs with varying bit-widths and matrix dimensions

Configuration		uGEMM	*tu*GEMM	*tub*GEMM	bGEMM
2-bit	16x16	42.2	4.9	5.0	7.7
	32x32	323.8	18.3	19.8	30.9
4-bit	16x16	64.1	9.2	9.9	22.4
	32x32	513.6	37.2	39.1	88.3
8-bit	16x16	100.8	19.7	26.1	72.8
	32x32	784.4	74.7	90.9	321.3

IV. Experimental Results

A. Area-Power Evaluation

Tables II and III show 45nm post-synthesis area and power results. *tu*GEMM leads in area-power efficiency for all configurations, followed by *tub*GEMM at 4 and 8 bits. At 2 bits, bGEMM has better area efficiency than *tub*GEMM but worsens with higher bitwidths. uGEMM is the least efficient. More specifically, uGEMM, *tub*GEMM, and bGEMM are approximately 7x, 1.5x, and 2x worse than *tu*GEMM on average for 16x16 GEMM area. uGEMM scales poorly with matrix size, resulting in an increased gap of 15x for 32x32 GEMM, while *tu*GEMM, *tub*GEMM and bGEMM scale similarly with matrix sizes. In power, *tu*GEMM and *tub*GEMM are close and consistently outperform uGEMM and bGEMM, with uGEMM consuming the most power (about 10x for 8-bit 32x32 compared to *tu*GEMM). Note that temporal encoding provides the advantage of only two signal transitions due to consecutive ones followed by zeros, in contrast to rate-unary and binary encoding with multiple signal transitions. The optimal compromise for unary designs appears to be at 4-bits, considering the exponential latency increase with bit-width and area-power efficiency, with *tu*GEMM and *tub*GEMM emerging as the most efficient designs.

B. Sparsity-Latency Evaluation

The weight sparsity profiling results are summarized in Table IV. Across most CNN models, there is a consistent word sparsity of approximately 2%, with MobileNetV3 being the outlier with 9.5% of its weights as zeros. For the LLaMA2-70B LLM model, the 8-bit self-attention tokens exhibit comparable weight sparsity, averaging around 2.8% (equivalent to approximately 2B zero weights out of the total 70B weights). 4-bit (36%) and 2-bit (84%) word sparsities are significantly higher. In contrast, the attention FC and FFN layers involve much lower word sparsities (negligible for 8-bits).

TABLE IV
PROFILED WEIGHT SPARSITIES OF CNNS AND LLAMA2-70B. WORD SPARSITY INDICATES THE PERCENTAGE OF ZERO WEIGHTS WHEREAS BIT SPARSITY DENOTES THE PERCENTAGE OF ZERO BITS WITHIN TEMPORAL-UNARY-ENCODED WEIGHTS. HIGHER BIT SPARSITY LEADS TO LOWER LATENCY AND ENERGY.

CNN	**Word (%) 8 bits**	**Bit (%) 8 bits**
MobileNetV2	2.25	44.66
MobileNetV3	9.52	38.59
GoogleNet	1.91	45.91
InceptionV3	1.99	45.61
ShuffleNetV3	1.43	47.18
ResNet18	2.04	45.3
ResNet50	2.45	46.24
ResNeXt101	2.64	44.23
LLaMA2-70B Layer (LLM)	**Word (%) 2/4/8 bits**	**Bit (%) 2/4/8 bits**
Attention FC layer weight	20.7 / 2.85 / 0.0613	50.00 / 12.50 / 0.82
FFN layer weight	20.8 / 3.02 / 0.0524	50.00 / 12.5 / 0.80
Self attention Q	82.8 / 35.0 / 2.71	0.56 / 8.89 / 28.84
Self attention K	85.1 / 37.4 / 2.94	8.19 / 8.58 / 32.52

Bit sparsity subsumes word sparsity and translates to latency and energy improvements for *tu*GEMM and *tub*GEMM. CNNs display significantly better bit sparsity (~43%) compared to LLM (negligible for 8-bit FC/FFN layers and ~30% for tokens). 4- and 2-bit values show varying sparsities for LLM. Note that only the two temporal-unary designs can leverage bit sparsity. Upon leveraging sparsity, notable improvements for *tub*GEMM and *tu*GEMM will include reduced latency and improved energy efficiency as they inherently exploit sparsity.

V. POTENTIAL FOR UNARY-BASED AI COMPUTE

Based on the empirical findings presented above, the following potential areas are recommended to be explored for the development of future unary-compute designs for AI inference aimed at optimizing efficiency and mitigating the computational efficiency gap between unary and binary paradigms:

1) Temporal-Unary Compute: While stochastic computing using rate-unary methods has undergone extensive exploration, temporal-unary computing remains relatively unexplored. Notably, *tu*GEMM and *tub*GEMM exemplify the potential of employing conventional digital circuits to implement temporal-unary hardware for AI applications. Stochastic computing exacerbates accuracy loss, with reported degradation from a 96.08% baseline INT8 quantized MLP model using binary processing elements to 94.7% after adopting uGEMM [3]. In contrast, temporal-unary designs facilitate deterministic computation, eliminating such accuracy loss. Leveraging the *n*-unary encoding scheme from [5] holds promise for additional latency and energy efficiency optimization, with potential correction schemes to mitigate associated penalties.

2) Leveraging Sparsity: The sparsity profiling results presented in Table IV suggest significant potential for optimizing latency through the exploitation of bit sparsity. With pruning techniques and SparseML libraries becoming increasingly indispensable for edge AI compute, the potential for leveraging sparsity (as done in *tu*GEMM and *tub*GEMM) holds immense promise for latency and, thereby, energy improvement.

3) Low-Precision Compute: Given the ongoing advancements in quantization techniques, AI inference is progressively embracing ultra-low bit-precisions. This trend affords unary-based designs the opportunity for exponential reductions in compute latency, rendering them a viable alternative to binary-based processing elements for 8 bits and below.

REFERENCES

[1] O. Kaynak, "The golden age of artificial intelligence: Inaugural editorial," pp. 1–7, 2021.

[2] OpenAI. (2018) Ai and compute. [Online]. Available: https://openai.com/blog/ai-and-compute/

[3] D. Wu, J. Li, R. Yin, H. Hsiao, Y. Kim, and J. San Miguel, "uGEMM: Unary Computing Architecture for GEMM Applications," in *ISCA*, 2020.

[4] 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 *ISCAS*. IEEE, 2023.

[5] 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 *ISVLSI*. IEEE, 2023, pp. 1–6.

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

[7] J. Choi, S. Venkataramani, V. V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, "Accurate and efficient 2-bit quantized neural networks," *MLSys*, vol. 1, pp. 348–359, 2019.

[8] A. C. Elster and T. A. Haugdahl, "Nvidia hopper gpu and grace cpu highlights," *Computing in Science & Engineering*, vol. 24, no. 2, pp. 95–100, 2022.

[9] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, and F. Wei, "The era of 1-bit llms: All large language models are in 1.58 bits," *arXiv preprint arXiv:2402.17764*, 2024.

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

[11] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *TCAD*, vol. 37, no. 8, pp. 1515–1531, 2017.

[12] C. Lee, J. Jin, T. Kim, H. Kim, and E. Park, "OWQ: Outlier-Aware Weight Quantization for Efficient Fine-Tuning and Inference of LLMs," *arXiv*, 2024.

[13] Kung, H. T. and McDanel, Bradley and Zhang, Sai Qian, "Term quantization: Furthering quantization at run time," in *SC*, 2020.